DATA+AI SUMMIT 2022

Learn To Efficiently Test ETL Pipelines

A story told with code

Jacqueline Bilston
Software Developer, Yelp

ORGANIZED BY databricks

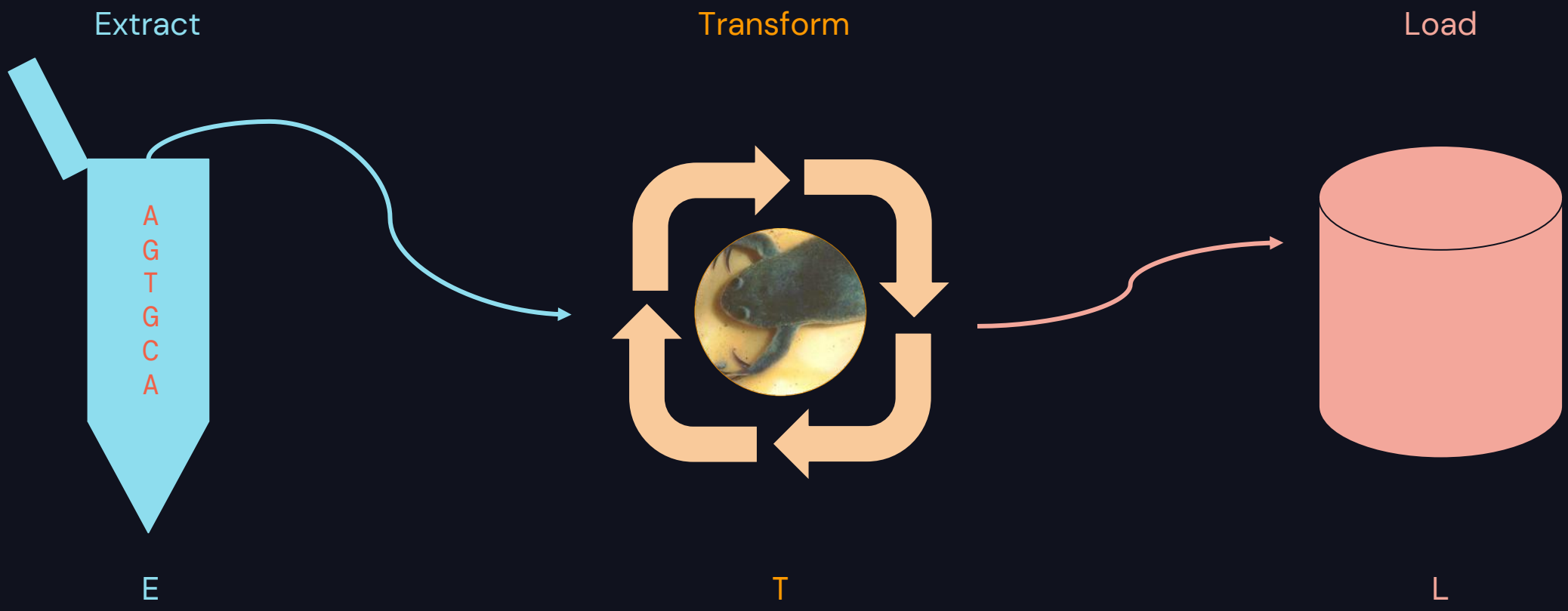
Resources

github.com/jmasonlee/Talks/blob/master/LearnToEfficientlyTestETLPipelines.md


Xenopus

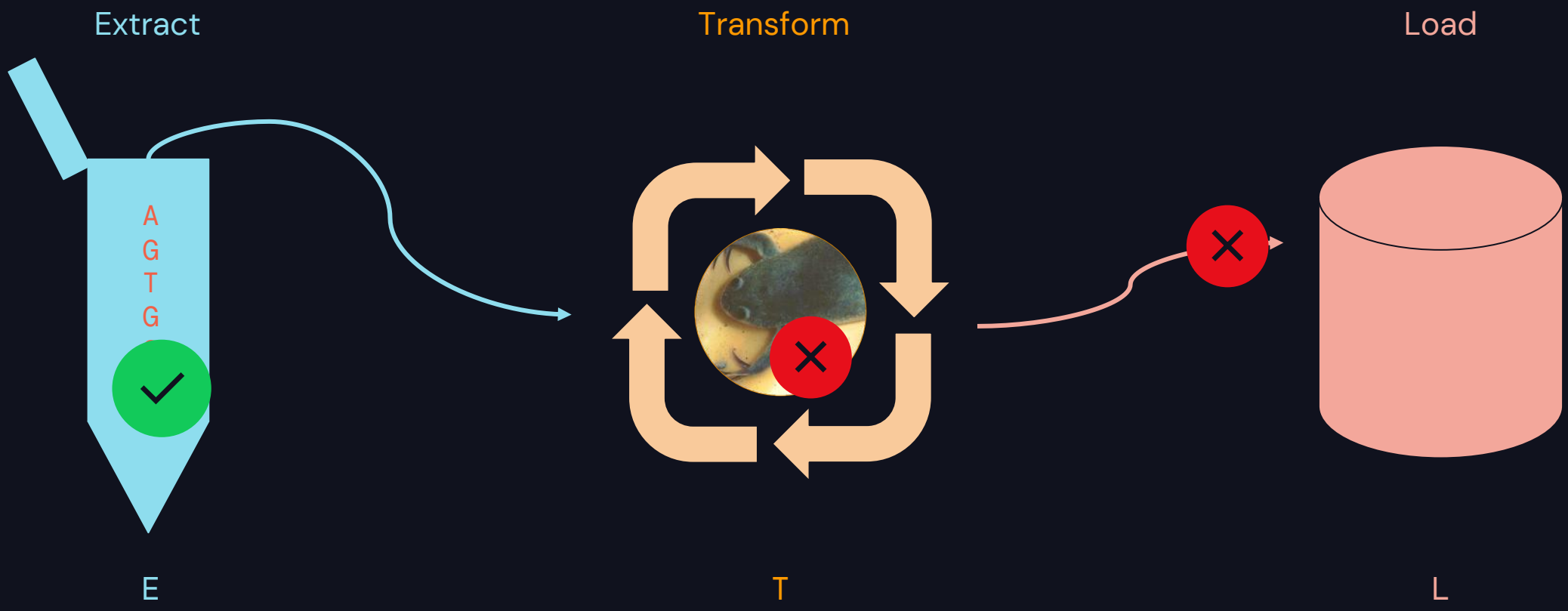
Frog Genomics

They're Hoppin'



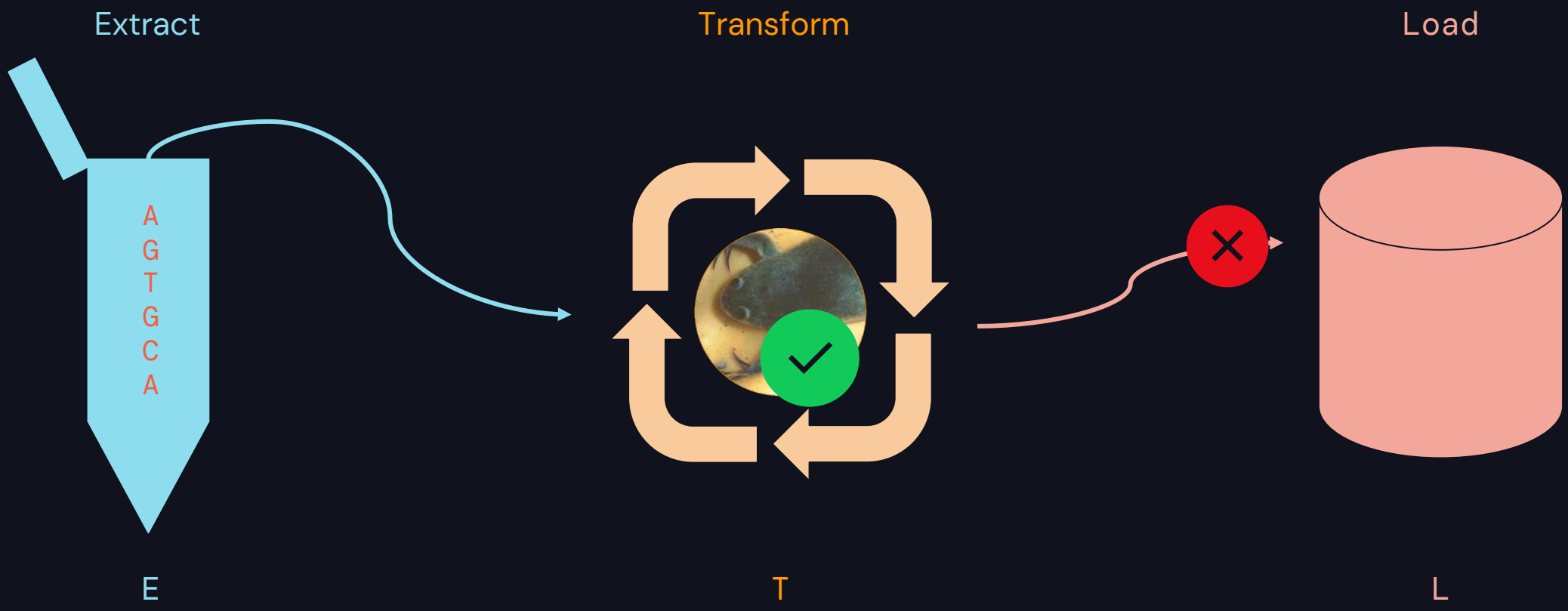
Frog Genomics

They're Hoppin'



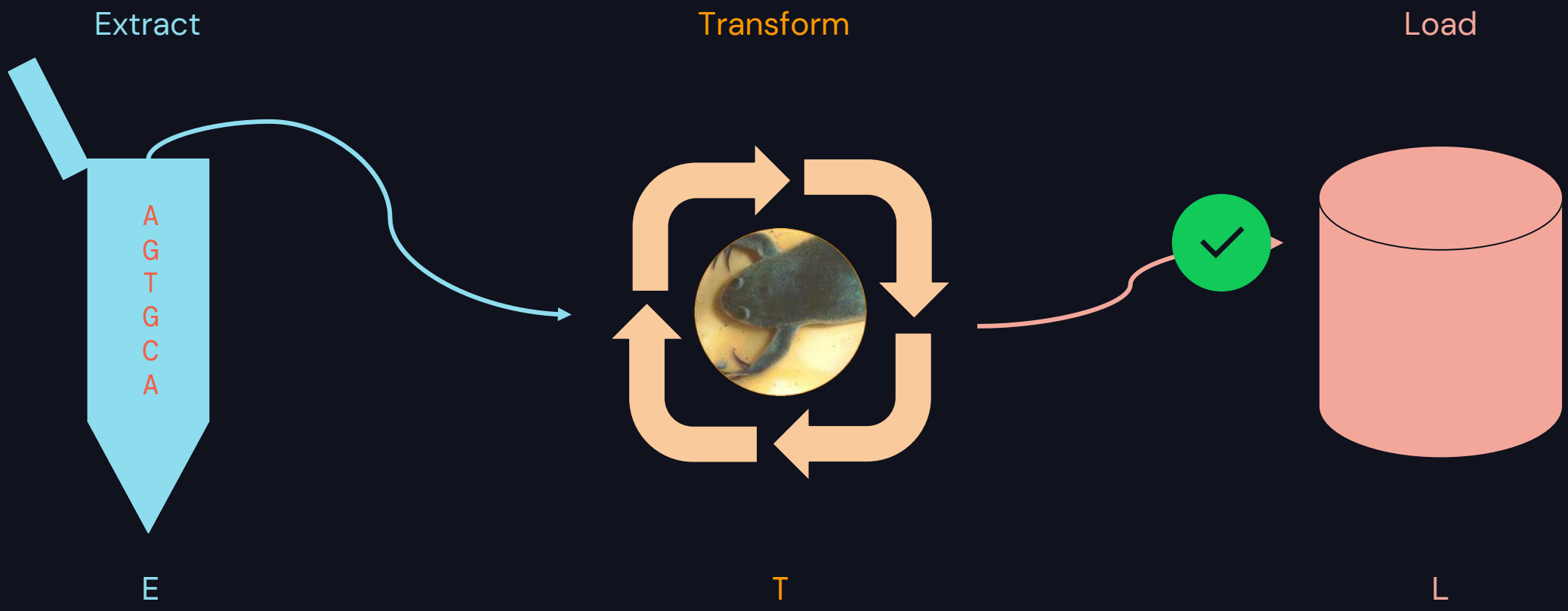
Frog Genomics

They're Hoppin'



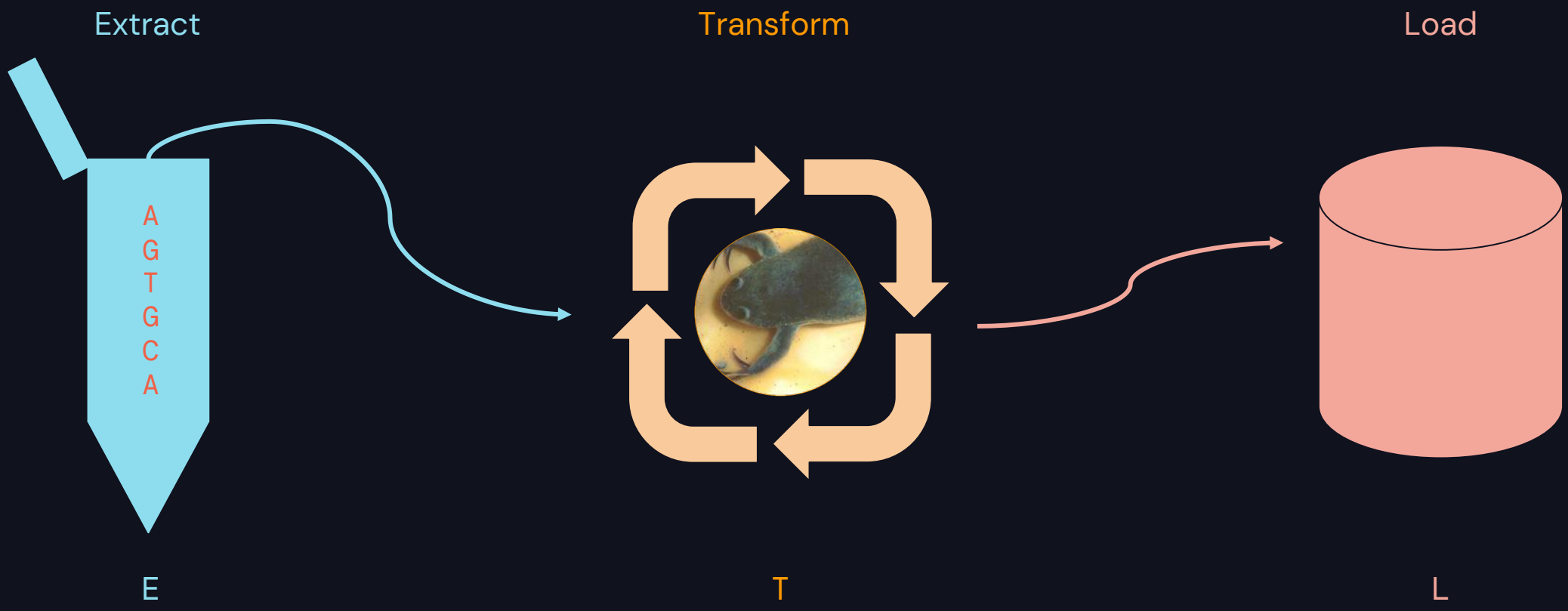
Frog Genomics

They're Hoppin'



Frog Genomics

They're Hoppin'



The Bug

“The number of interactions in the pandemic_recovery table doesn’t look right”

The Problem Table

The PANDEMIC_RECOVERY table

business_id	business_name	review_count	checkin_count	tip_count	interaction_count	date
bBDDEgkFA1Otx9Lfe7BZ	Sonic Drive-In	0	1	3	4	2022-06-28
kOhlBqXX-BtOvf1op7Jr1w	Tsevi's Pub And Grill	2	0	1	3	2022-06-28
n_0UpQx1hsNbnPUSlodU	Famous Footwear	0	0	0	0	2022-06-28
mWMc6_wTdE0EUBKIGX	Perkiomen Valley Brewery	1	3	1	5	2022-06-28
MTSW4McQd7CbVtyjqoe	St Honore Pastries	0	0	0	0	2022-06-28

The Problem

The PANDEMIC_RECOVERY table

business_id	business_name	review_count	checkin_count	tip_count	interaction_count	date
bBDDEgkFA1Otx9Lfe7BZ	Sonic Drive-In	0	1	3	4	2022-06-28
kOhlBqXX-BtOvf1op7Jr1w	Tsevi's Pub And Grill	2	0	1	3	2022-06-28
n_0UpQx1hsNbnPUSlodU	Famous Footwear	0	0	0	0	2022-06-28
mWMc6_wTdE0EUBKIGX	Perkiomen Valley Brewery	1	3	1	5	2022-06-28
MTSW4McQd7CbVtyjqoe	St Honore Pastries	0	0	0	0	2022-06-28

The Problem

The PANDEMIC_RECOVERY table

business_id	business_name	review_count	checkin_count	tip_count	interaction_count	date
bBDDEgkFA1Otx9Lfe7BZ	Sonic Drive-In	0	1	3	4	2022-06-28
kOhlBqXX-BtOvf1op7Jr1w	Tsevi's Pub And Grill	2	0	1	3	2022-06-28
n_0UpQx1hsNbnPUSlodU	Famous Footwear	0	0	0	0	2022-06-28
mWMc6_wTdE0EUBKIGX	Perkiomen Valley Brewery	1	3	1	5	2022-06-28
MTSW4McQd7CbVtyjqoe	St Honore Pastries	0	0	0	0	2022-06-28

The Problem

The PANDEMIC_RECOVERY table

business_id	business_name	review_count	checkin_count	tip_count	interaction_count	date
bBDDEgkFA1Otx9Lfe7BZ	Sonic Drive-In	0	1	3	4	2022-06-28
kOhlBqXX-BtOvf1op7Jr1w	Tsevi's Pub And Grill	2	0	1	3	2022-06-28
n_0UpQx1hsNbnPUSlodU	Famous Footwear	0	0	0	0	2022-06-28
mWMc6_wTdE0EUBKIGX	Perkiomen Valley Brewery	1	3	1	5	2022-06-28
MTSW4McQd7CbVtyjqoe	St Honore Pastries	0	0	0	0	2022-06-28

The Problem

The PANDEMIC_RECOVERY table

business_id	business_name	review_count	checkin_count	tip_count	interaction_count	date
bBDDEgkFA1Otx9Lfe7BZ	Sonic Drive-In	0	1	3	4	2022-06-28
kOhlBqXX-BtOvf1op7Jr1w	Tsevi's Pub And Grill	2	0	1	3	2022-06-28
n_0UpQx1hsNbnPUSlodU	Famous Footwear	0	0	0	0	2022-06-28
mWMc6_wTdE0EUBKIGX	Perkiomen Valley Brewery	1	3	1	5	2022-06-28
MTSW4McQd7CbVtyjqoe	St Honore Pastries	0	0	0	0	2022-06-28

The Problem

The PANDEMIC_RECOVERY table

business_id	business_name	review_count	checkin_count	tip_count	interaction_count	date
bBDDEgkFA1Otx9Lfe7BZ	Sonic Drive-In	0	1	3	4	2022-06-28
kOhlBqXX-BtOvf1op7Jr1w	Tsevi's Pub And Grill	2	0	1	3	2022-06-28
n_0UpQx1hsNbnPUSlodU	Famous Footwear	0	0	0	0	2022-06-28
mWMc6_wTdE0EUBKIGX	Perkiomen Valley Brewery	1	3	1	5	2022-06-28
MTSW4McQd7CbVtyjqoe	St Honore Pastries	0	0	0	0	2022-06-28

The Problem

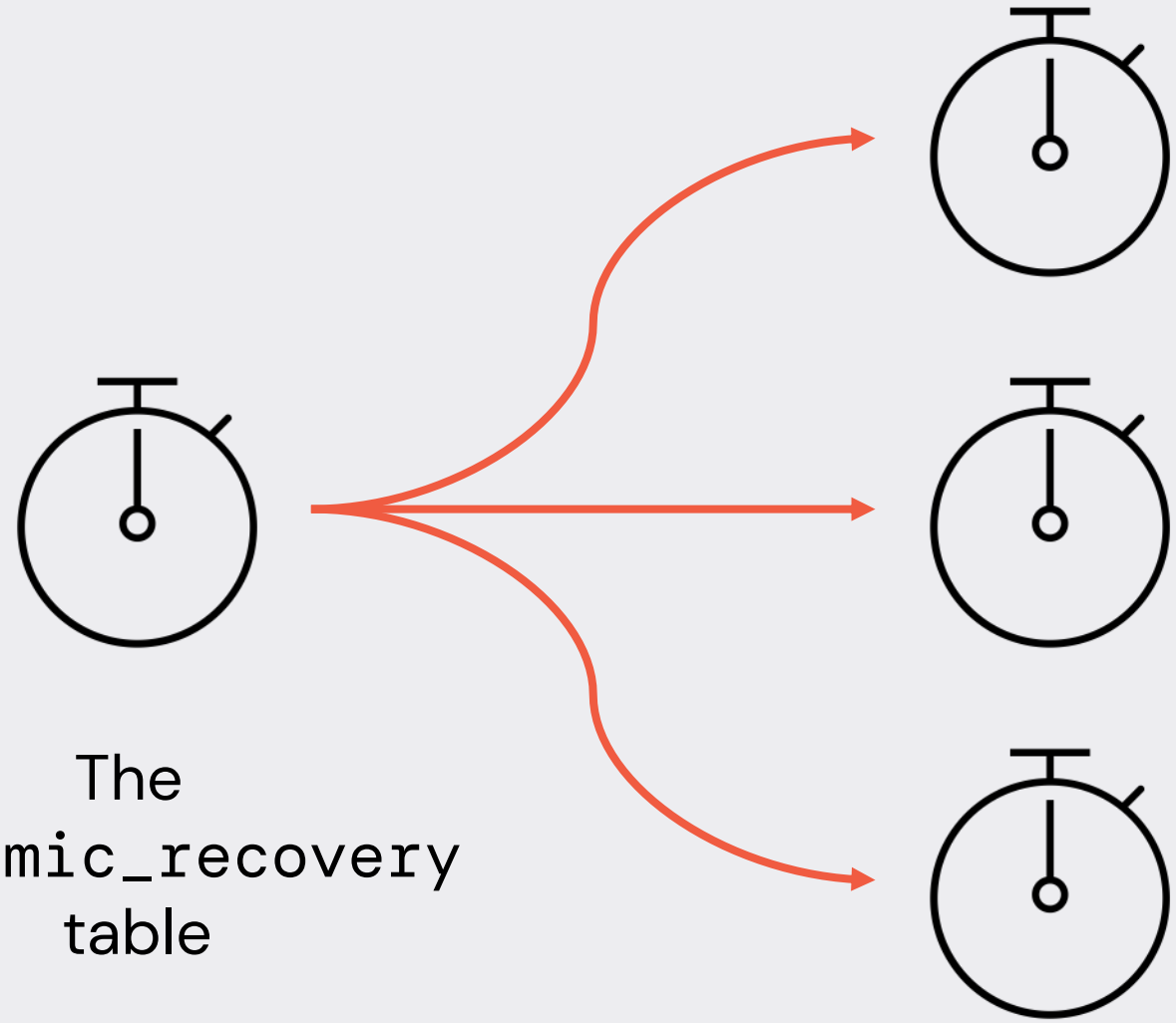
The PANDEMIC_RECOVERY table

business_id	business_name	review_count	checkin_count	tip_count	interaction_count	date
bBDDEgkFA1Otx9Lfe7BZ	Sonic Drive-In	0	1	3	4	2022-06-28
kOhlBqXX-BtOvf1op7Jr1w	Tsevi's Pub And Grill	2	0	1	3	2022-06-28
n_0UpQx1hsNbnPUSlodU	Famous Footwear	0	0	0	0	2022-06-28
mWMc6_wTdE0EUBKIGX	Perkiomen Valley Brewery	1	3	1	5	2022-06-28
MTSW4McQd7CbVtyjqoe	St Honore Pastries	0	0	0	0	2022-06-28

The Problem

The PANDEMIC_RECOVERY table

business_name	review_count	tip_count	checkin_count	interaction_count	date
Perkiomen Valley Brewery	~~2~~ 4	1	3	~~6~~ 8	2022-06-28

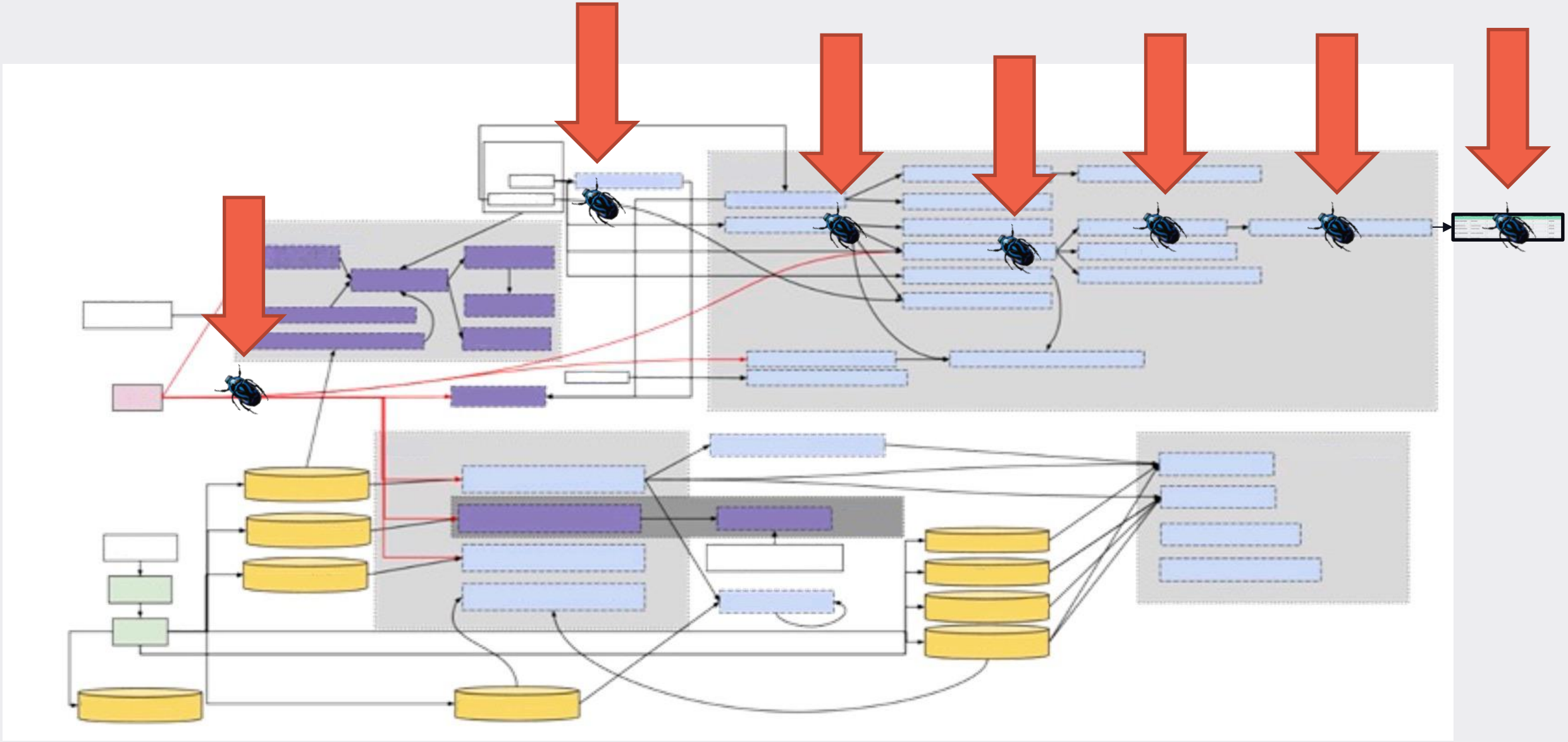


The `pandemic_recovery` table

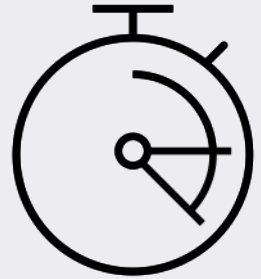
Machine Learning Models

Reports

Other Pipelines

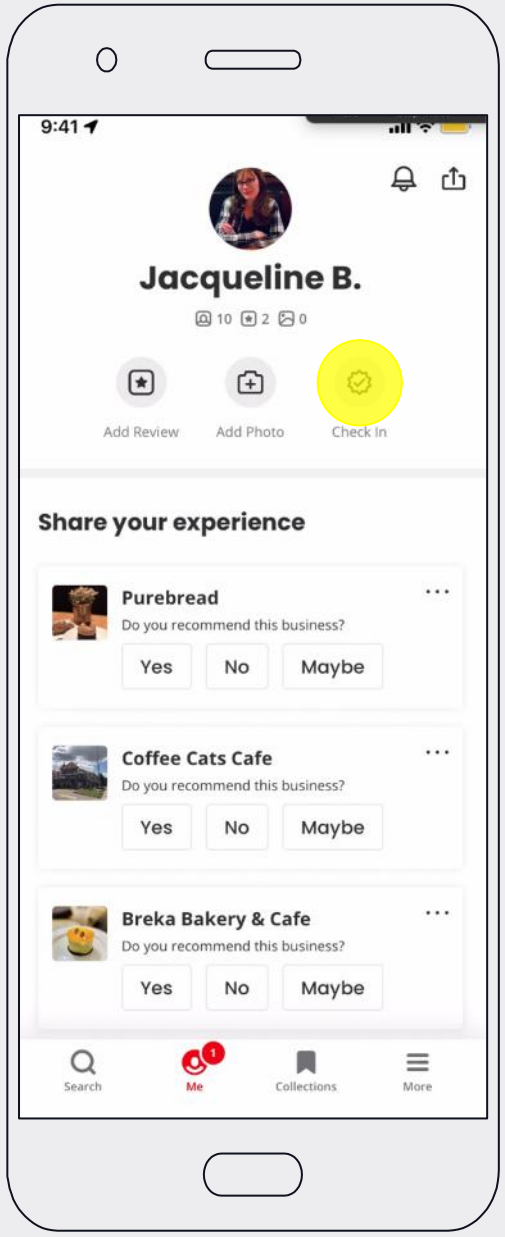


5 days to find the bug,

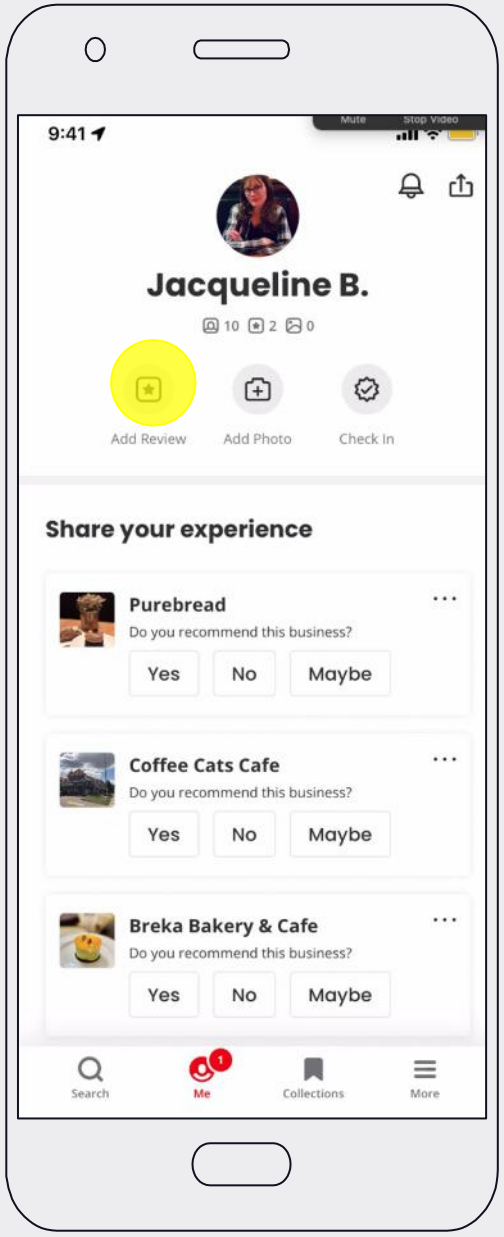


5 days to find the bug,

3 days to fix it,

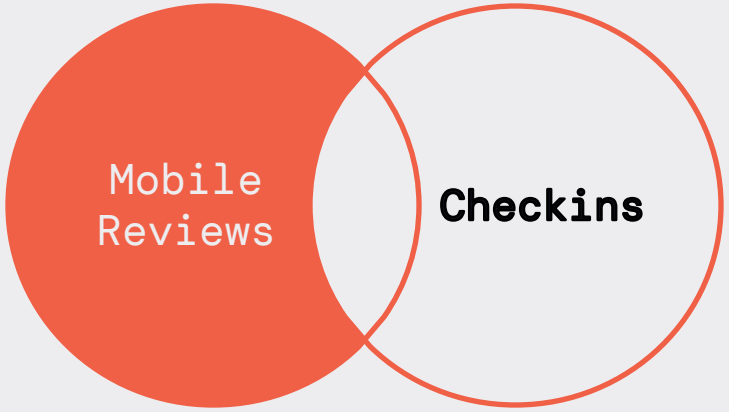


checkin_count
3



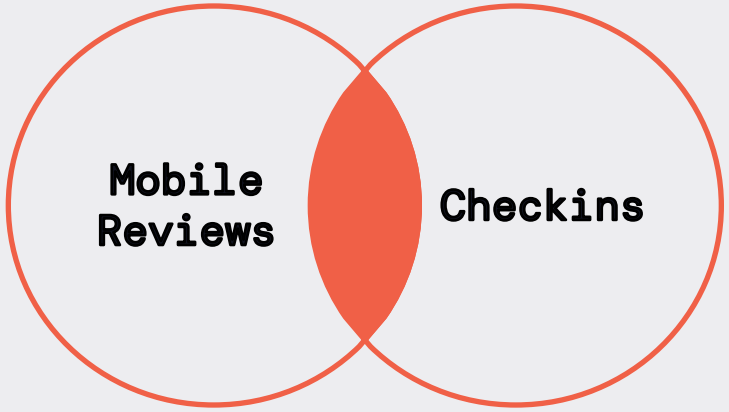
review_count
1

Left-Anti Join

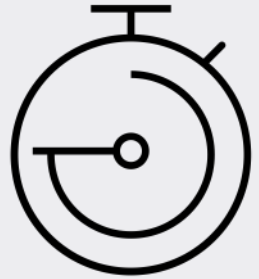



```
mobile_reviews.join(
    checkins,
    on=[”user_id”, “business_id”, “date”]
    how= “left_anti”
)
```

Inner Join



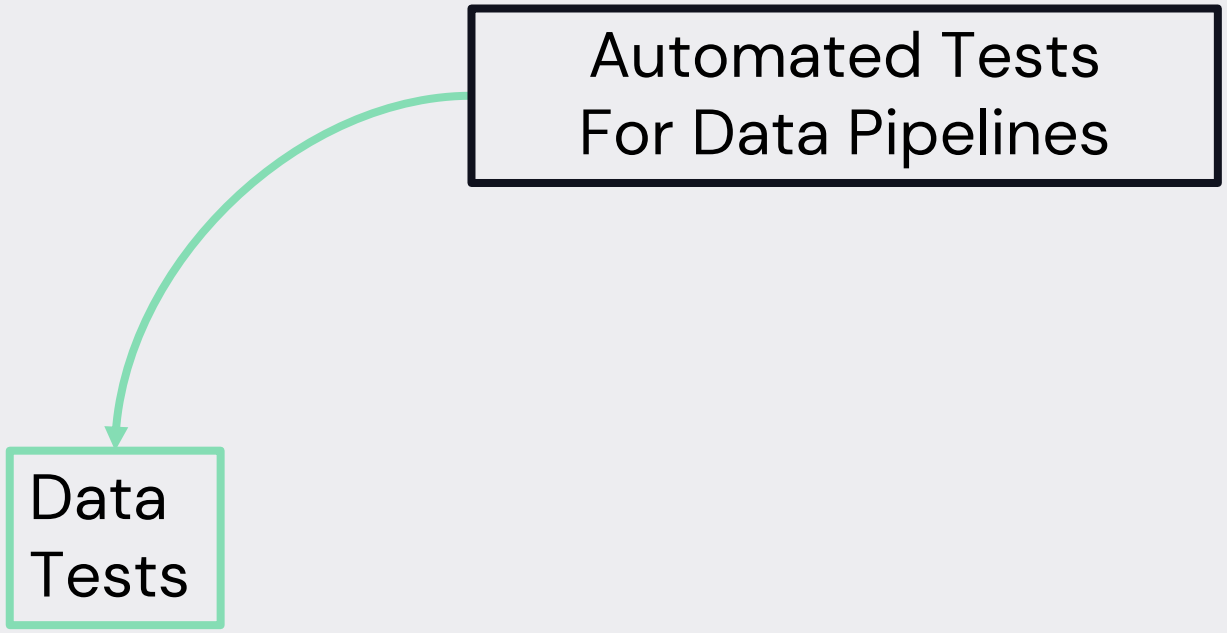

```
mobile_reviews.join(
    checkins,
    on=[”user_id”, “business_id”, “date”]
)
```



5 days to find the bug,

3 days to fix it,

followed by 7 days of running backfills for 2 months of data



ID	Name	Age	Eye color
01452	Alan	452	blue
64521	NULL	29	green
49031	Margaret	45	brown
49031	Margaret	45	brown

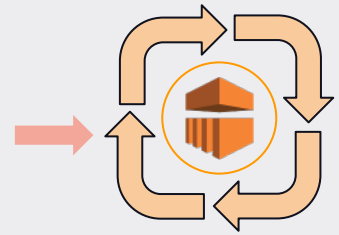
Automated Tests For Data Pipelines

Data Tests

Logic Tests

ID	Name	Age	Eye color
01452	Alan	452	blue
64521	NULL	29	green
49031	Margaret	45	brown
49031	Margaret	45	brown

City
Trono
Calgry
Vancover



City
Toronto
Calgary
Vancouver


—Jay Bazuzi

"What bug are you trying to catch?"

—Jay Bazuzi

The Logic Test

```
def test_will_do_the_right_thing(spark: SparkSession) -> None:
  browser_reviews_df = create_df_from_json("fixtures/browser_reviews.json", spark)
  checkin_df   = create_df_from_json("fixtures/checkin.json", spark)
  tips_df      = create_df_from_json("fixtures/tips.json", spark)
  business_df  = create_df_from_json("fixtures/business.json", spark)
  mobile_reviews_df = create_df_from_json("fixtures/mobile_reviews.json", spark)

  actual_df = transform(
    business_df,
    checkin_df,
    browser_reviews_df,
    tips_df,
    mobile_reviews_df,
    datetime(2022, 4, 14)
  )


  assert data_frame_to_json(actual_df) == expected_json()
```

The Logic Test

```
def test_will_do_the_right_thing(spark: SparkSession) -> None:
  browser_reviews_df = create_df_from_json("fixtures/browser_reviews.json", spark)
  checkin_df   = create_df_from_json("fixtures/checkin.json", spark)
  tips_df      = create_df_from_json("fixtures/tips.json", spark)
  business_df  = create_df_from_json("fixtures/business.json", spark)
  mobile_reviews_df = create_df_from_json("fixtures/mobile_reviews.json", spark)

  actual_df = transform(
    business_df,
    checkin_df,
    browser_reviews_df,
    tips_df,
    mobile_reviews_df,
    datetime(2022, 4, 14)
  )


  assert data_frame_to_json(actual_df) == expected_json()
```

The Logic Test

```
def test_will_do_the_right_thing(spark: SparkSession) -> None:
  browser_reviews_df = create_df_from_json("fixtures/browser_reviews.json", spark)
  checkin_df    = create_df_from_json("fixtures/checkin.json", spark)
  tips_df       = create_df_from_json("fixtures/tips.json", spark)
  business_df   = create_df_from_json("fixtures/business.json", spark)
  mobile_reviews_df = create_df_from_json("fixtures/mobile_reviews.json", spark)

  actual_df = transform(
    business_df,
    checkin_df,
    browser_reviews_df,
    tips_df,
    mobile_reviews_df,
    datetime(2022, 4, 14)
  )

  assert data_frame_to_json(actual_df) == expected_json()
```

The Logic Test

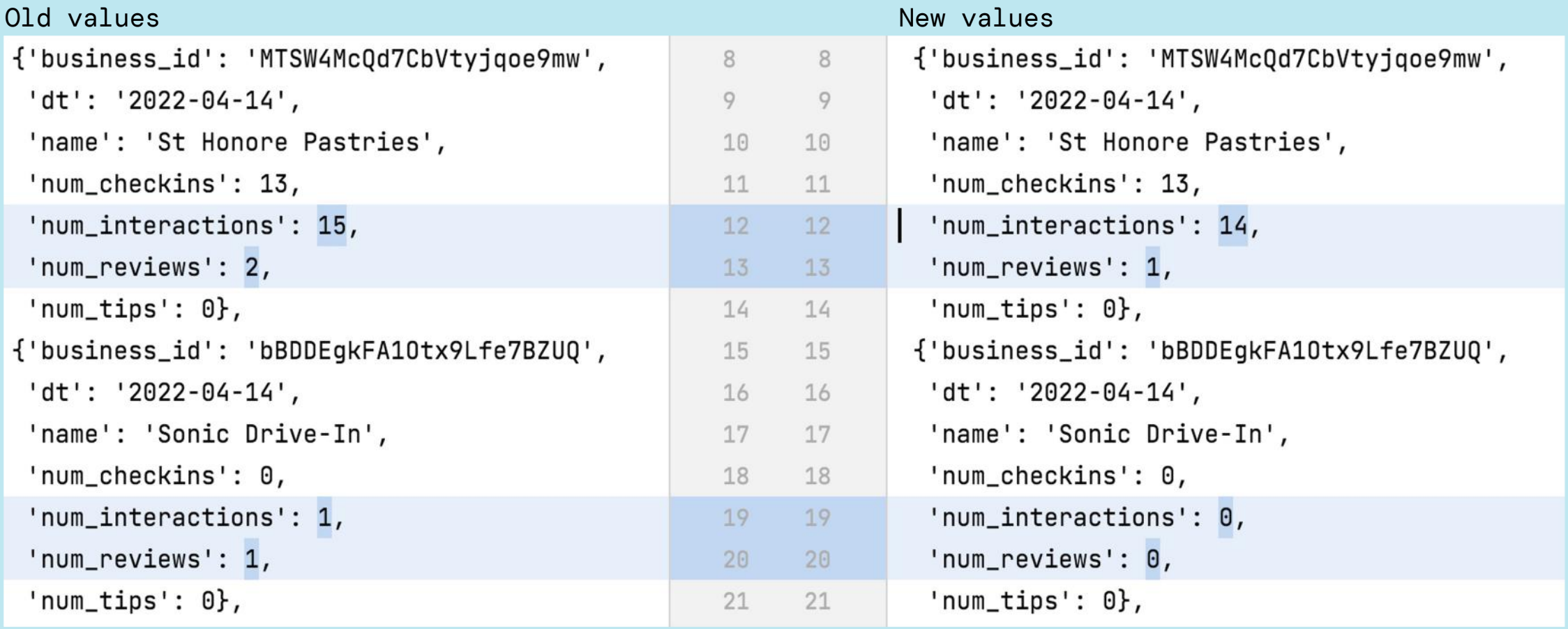
```
def test_will_do_the_right_thing(spark: SparkSession) -> None:
  browser_reviews_df = create_df_from_json("fixtures/browser_reviews.json", spark)
  checkin_df   = create_df_from_json("fixtures/checkin.json", spark)
  tips_df      = create_df_from_json("fixtures/tips.json", spark)
  business_df  = create_df_from_json("fixtures/business.json", spark)
  mobile_reviews_df = create_df_from_json("fixtures/mobile_reviews.json", spark)

  actual_df = transform(
    business_df,
    checkin_df,
    browser_reviews_df,
    tips_df,
    mobile_reviews_df,
    datetime(2022, 4, 14)
  )


  assert data_frame_to_json(actual_df) == expected_json()
```

expected.json


```
  {
    "business_id": "CF33F8-E6oudUQ46HnavjQ",
    "name": "Sonic Drive-In",
    "num_tips": 1,
    "num_checkins": 0,
    "num_reviews": 1,
    "num_interactions": 2,
    "dt": "2022-04-14"
  },
  {
    "business_id": "MTSW4McQd7CbVtyjqoe9mw",
    "name": "St Honore Pastries",
    "num_tips": 0,
    "num_checkins": 13,
    "num_reviews": 1,
    "num_interactions": 14,
    "dt": "2022-04-14"
  },
  {
    "business_id": "Pns2l4eNsfO8kk83dixA6A",
    "name": "Abby Rappoport, LAC, CMQ",
    "num_tips": 1,
    "num_checkins": 6,
    "num_reviews": 1,
    "num_interactions": 8,
    "dt": "2022-04-14"
  },
```

```
},
{
  "business_id": "n_0UpQx1hsNbnPUSlodU8w",
  "name": "Famous Footwear",
  "num_tips": 0,
  "num_checkins": 8,
  "num_reviews": 0,
  "num_interactions": 8,
  "dt": "2022-04-14"
},
{
  "business_id": "qkRM_2X51Yqxk3btlwAQIg",
  "name": "Temple Beth-El",
  "num_tips": 0,
  "num_checkins": 1,
  "num_reviews": 1,
  "num_interactions": 2,
  "dt": "2022-04-14"
},
{
  "business_id": "tUFrWirKiKi_TAnsVWINQQ",
  "name": "Target",
  "num_tips": 0,
  "num_checkins": 1,
  "num_reviews": 0,
  "num_interactions": 1,
  "dt": "2022-04-14"
}]
```

Large Outputs



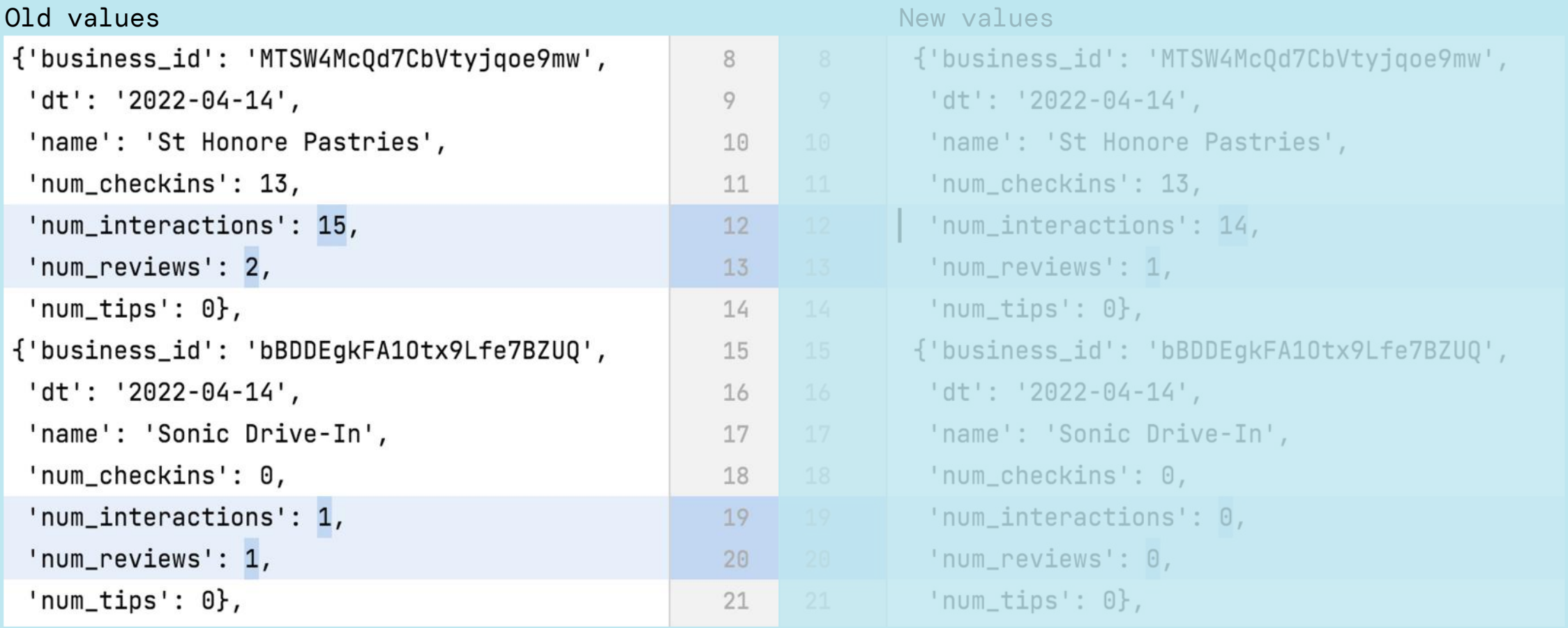
Old values

```
{'business_id': 'MTSW4McQd7CbVtyjqoe9mw',
 'dt': '2022-04-14',
 'name': 'St Honore Pastries',
 'num_checkins': 13,
 'num_interactions': 15,
 'num_reviews': 2,
 'num_tips': 0},
{'business_id': 'bBDDEgkFA1Otx9Lfe7BZUQ',
 'dt': '2022-04-14',
 'name': 'Sonic Drive-In',
 'num_checkins': 0,
 'num_interactions': 1,
 'num_reviews': 1,
 'num_tips': 0},
```

New values

```
{'business_id': 'MTSW4McQd7CbVtyjqoe9mw',
 'dt': '2022-04-14',
 'name': 'St Honore Pastries',
 'num_checkins': 13,
 'num_interactions': 14,
 'num_reviews': 1,
 'num_tips': 0},
{'business_id': 'bBDDEgkFA1Otx9Lfe7BZUQ',
 'dt': '2022-04-14',
 'name': 'Sonic Drive-In',
 'num_checkins': 0,
 'num_interactions': 0,
 'num_reviews': 0,
 'num_tips': 0},
```

Large Outputs



Large Outputs

Old values

```
{'business_id': 'MTSW4McQd7CbVtyjqoe9mw',
 'dt': '2022-04-14',
 'name': 'St Honore Pastries',
 'num_checkins': 13,
 'num_interactions': 15,
 'num_reviews': 2,
 'num_tips': 0},
{'business_id': 'bBDDEgkFA1Otx9Lfe7BZUQ',
 'dt': '2022-04-14',
 'name': 'Sonic Drive-In',
 'num_checkins': 0,
 'num_interactions': 1,
 'num_reviews': 1,
 'num_tips': 0},
```

New values

```
 {'business_id': 'MTSW4McQd7CbVtyjqoe9mw',
  'dt': '2022-04-14',
  'name': 'St Honore Pastries',
  'num_checkins': 13,
  'num_interactions': 14,
  'num_reviews': 1,
  'num_tips': 0},
 {'business_id': 'bBDDEgkFA1Otx9Lfe7BZUQ',
  'dt': '2022-04-14',
  'name': 'Sonic Drive-In',
  'num_checkins': 0,
  'num_interactions': 0,
  'num_reviews': 0,
  'num_tips': 0},
```

Large Outputs

Old values

```
{'business_id': 'MTSW4McQd7CbVtyjqoe9mw',
 'dt': '2022-04-14',
 'name': 'St Honore Pastries',
 'num_checkins': 13,
 'num_interactions': 15,
 'num_reviews': 2,
 'num_tips': 0},
{'business_id': 'bBDDEgkFA1Otx9Lfe7BZUQ',
 'dt': '2022-04-14',
 'name': 'Sonic Drive-In',
 'num_checkins': 0,
 'num_interactions': 1,
 'num_reviews': 1,
 'num_tips': 0},
```

New values

```
 {'business_id': 'MTSW4McQd7CbVtyjqoe9mw',
  'dt': '2022-04-14',
  'name': 'St Honore Pastries',
  'num_checkins': 13,
  'num_interactions': 14,
  'num_reviews': 1,
  'num_tips': 0},
 {'business_id': 'bBDDEgkFA1Otx9Lfe7BZUQ',
  'dt': '2022-04-14',
  'name': 'Sonic Drive-In',
  'num_checkins': 0,
  'num_interactions': 0,
  'num_reviews': 0,
  'num_tips': 0},
```

Large Outputs

```
actual_df = transform(
    business_df,
    checkin_df,
    reviews_df,
    tips_df,
    mobile_reviews_df,
    datetime(2022, 4, 14).strftime('%Y-%m-%d')
)


assert data_frame_to_json(actual_df) == expected_json()
#overwrite_expected(actual_df)
```

—Jay Bazuzi

"Have you ever heard of a Saff squeeze?"

—Jay Bazuzi

The Saff Squeeze

Create a simpler test based off of a big one

```python
def transform(business_df: DataFrame,
              checkin_df: DataFrame,
              browser_reviews_df: DataFrame,
              tips_df: DataFrame,
              mobile_reviews_df: DataFrame,
              run_date: datetime = datetime.today()):
    checkin_df = checkin_df.withColumn("checkins_list", F.split(checkin_df.date, ","))
    checkin_df = checkin_df.select(F.col("business_id"), F.explode(F.col("checkins_list")).alias("date"))

    reviews_df = count_reviews(checkin_df, mobile_reviews_df, browser_reviews_df, run_date)

    checkin_df = count_checkins(checkin_df, run_date)

    tips_df = count_tips(tips_df, run_date)

    return construct_post_pandemic_recovery_df(
        business_df, checkin_df, reviews_df, run_date, tips_df
    )


def construct_post_pandemic_recovery_df(business_df, checkin_df, reviews_df, run_date, tips_df):
    entity_with_activity_df = business_df.join(checkin_df, on="business_id", how='left').fillna(0)
    entity_with_activity_df = entity_with_activity_df.join(reviews_df, on="business_id", how='left').fillna(0)
    entity_with_activity_df = entity_with_activity_df.join(tips_df, on="business_id", how='left').fillna(0)
    entity_with_activity_df = entity_with_activity_df.withColumn("num_interactions",
                                                                 entity_with_activity_df.num_reviews +
                                                                 entity_with_activity_df.num_tips +
                                                                 entity_with_activity_df.num_checkins)
    entity_with_activity_df = entity_with_activity_df.withColumn("dt", F.lit(run_date))
    entity_with_activity_df = entity_with_activity_df.select(
        "business_id", "name", "num_tips", "num_checkins", "num_reviews", "num_interactions", "dt"
    )
    return entity_with_activity_df


def count_tips(tips_df, run_date):
    tips_df = tips_df.filter(tips_df.date == run_date.date())
    tips_df = tips_df.groupby("business_id").count()
    tips_df = tips_df.withColumnRenamed("count", "num_tips")
    return tips_df


def count_checkins(checkin_df, run_date):
    checkin_df = checkin_df.filter(checkin_df.date == run_date.date())
    checkin_df = checkin_df.groupby("business_id").count()
    checkin_df = checkin_df.withColumnRenamed("count", "num_checkins")
    return checkin_df


def count_reviews(checkin_df, mobile_reviews_df, browser_reviews_df, run_date):
    mobile_reviews_df = mobile_reviews_df.join(checkin_df, on=['business_id'], how="left_anti").select(mobile_reviews_df.columns)
    reviews_df = mobile_reviews_df.union(browser_reviews_df)
    reviews_df = reviews_df.filter(reviews_df.date == run_date.date())
    reviews_df = reviews_df.groupby("business_id").count()
    reviews_df = reviews_df.withColumnRenamed("count", "num_reviews")
    return reviews_df
```

A large integration test covers all of this code


```python
def transform(business_df: DataFrame,
              checkin_df: DataFrame,
              browser_reviews_df: DataFrame,
              tips_df: DataFrame,
              mobile_reviews_df: DataFrame,
              run_date: datetime = datetime.today()):
    checkin_df = checkin_df.withColumn("checkins_list", F.split(checkin_df.date, ","))
    checkin_df = checkin_df.select(F.col("business_id"), F.explode(F.col("checkins_list")).alias("date"))

    reviews_df = count_reviews(checkin_df, mobile_reviews_df, browser_reviews_df, run_date)

    checkin_df = count_checkins(checkin_df, run_date)

    tips_df = count_tips(tips_df, run_date)

    return construct_post_pandemic_recovery_df(
        business_df, checkin_df, reviews_df, run_date, tips_df
    )


def construct_post_pandemic_recovery_df(business_df, checkin_df, reviews_df, run_date, tips_df):
    entity_with_activity_df = business_df.join(checkin_df, on="business_id", how='left').fillna(0)
    entity_with_activity_df = entity_with_activity_df.join(reviews_df, on="business_id", how='left').fillna(0)
    entity_with_activity_df = entity_with_activity_df.join(tips_df, on="business_id", how='left').fillna(0)
    entity_with_activity_df = entity_with_activity_df.withColumn("num_interactions",
                                                                 entity_with_activity_df.num_reviews +
                                                                 entity_with_activity_df.num_tips +
                                                                 entity_with_activity_df.num_checkins)
    entity_with_activity_df = entity_with_activity_df.withColumn("dt", F.lit(run_date))
    entity_with_activity_df = entity_with_activity_df.select(
        "business_id", "name", "num_tips", "num_checkins", "num_reviews", "num_interactions", "dt"
    )
    return entity_with_activity_df


def count_tips(tips_df, run_date):
    tips_df = tips_df.filter(tips_df.date == run_date.date())
    tips_df = tips_df.groupby("business_id").count()
    tips_df = tips_df.withColumnRenamed("count", "num_tips")
    return tips_df


def count_checkins(checkin_df, run_date):
    checkin_df = checkin_df.filter(checkin_df.date == run_date.date())
    checkin_df = checkin_df.groupby("business_id").count()
    checkin_df = checkin_df.withColumnRenamed("count", "num_checkins")
    return checkin_df
```


```python
def count_reviews(checkin_df, mobile_reviews_df, browser_reviews_df, run_date):
    mobile_reviews_df = mobile_reviews_df.join(checkin_df, on=['business_id'], how="left_anti").select(mobile_reviews_df.columns)
    reviews_df = mobile_reviews_df.union(browser_reviews_df)
    reviews_df = reviews_df.filter(reviews_df.date == run_date.date())
    reviews_df = reviews_df.groupby("business_id").count()
    reviews_df = reviews_df.withColumnRenamed("count", "num_reviews")
    return reviews_df
```

A Saff Squeeze will let us write a test for only the logic we care about

Set Things Up

Duplicate the large test

The code that we are testing

```python
from pandemic_recovery_batch import transform


def test_will_do_the_right_thing(spark: SparkSession) -> None:
    b_reviews_df = create_df_from_json("fixtures/browser_reviews.json", spark)
    checkin_df = create_df_from_json("fixtures/checkin.json", spark)
    tips_df = create_df_from_json("fixtures/tips.json", spark)
    business_df = create_df_from_json("fixtures/business.json", spark)
    m_reviews_df = create_df_from_json("fixtures/mobile_reviews.json", spark)

    actual_df = transform(
        business_df,
        checkin_df,
        b_reviews_df,
        tips_df,
        m_reviews_df,
        datetime(2022, 4, 14)
    )

    assert data_frame_to_json(actual_df) == expected_json()


def create_df_from_json(json_file, spark):
    return spark.read.option("multiline", "true").json(json_file)


def data_frame_to_json(df: DataFrame) -> List:
    output = [json.loads(item) for item in df.toJSON().collect()]
    output.sort(key=lambda item: item["business_id"])
    return output

def expected_json():
    with open("fixtures/expected.json") as f:
        return json.loads(f.read())
```

The large integration test

Set Things Up

Duplicate the large test

The code that we are testing

```python
from pandemic_recovery_batch import transform


def test_will_do_the_right_thing(spark: SparkSession) -> None:
    b_reviews_df = create_df_from_json("fixtures/browser_reviews.json", spark)
    checkin_df = create_df_from_json("fixtures/checkin.json", spark)
    tips_df = create_df_from_json("fixtures/tips.json", spark)
    business_df = create_df_from_json("fixtures/business.json", spark)
    m_reviews_df = create_df_from_json("fixtures/mobile_reviews.json", spark)

    actual_df = transform(
        business_df,
        checkin_df,
        b_reviews_df,
        tips_df,
        m_reviews_df,
        datetime(2022, 4, 14)
    )

    assert data_frame_to_json(actual_df) == expected_json()


def test_will_not_count_mobile_reviews_at_the_same_time_as_checkins(spark: SparkSession) -> None:
    b_reviews_df = create_df_from_json("fixtures/browser_reviews.json", spark)
    checkin_df = create_df_from_json("fixtures/checkin.json", spark)
    tips_df = create_df_from_json("fixtures/tips.json", spark)
    business_df = create_df_from_json("fixtures/business.json", spark)
    m_reviews_df = create_df_from_json("fixtures/mobile_reviews.json", spark)

    actual_df = transform(
        business_df,
        checkin_df,
        b_reviews_df,
        tips_df,
        m_reviews_df,
        datetime(2022, 4, 14)
    )

    assert data_frame_to_json(actual_df) == expected_json()


def create_df_from_json(json_file, spark):
    return spark.read.option("multiline", "true").json(json_file)


def data_frame_to_json(df: DataFrame) -> List:
    output = [json.loads(item) for item in df.toJSON().collect()]
    output.sort(key=lambda item: item["business_id"])
    return output

def expected_json():
    with open("fixtures/expected.json") as f:
        return json.loads(f.read())
```

The large integration test

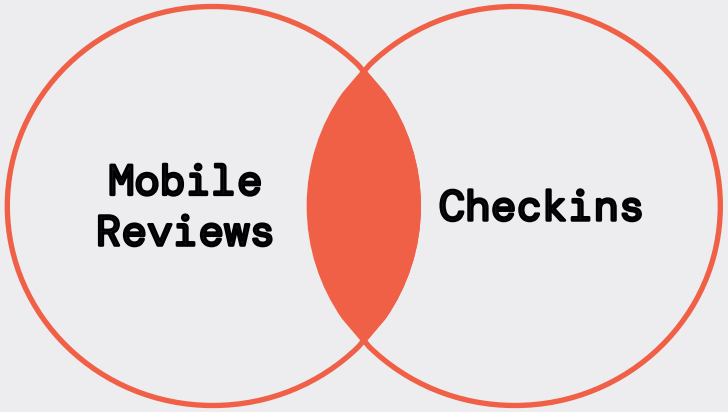
Starting with the Duplicated Test…

```
def test_will_do_the_right_thing(spark: SparkSession) -> None:

  browser_reviews_df = create_df_from_json("fixtures/browser_reviews.json", spark)
  checkin_df    = create_df_from_json("fixtures/checkin.json", spark)
  tips_df       = create_df_from_json("fixtures/tips.json", spark)
  business_df   = create_df_from_json("fixtures/business.json", spark)
  mobile_reviews_df = create_df_from_json("fixtures/mobile_reviews.json", spark)

  actual_df = transform(
    business_df,
    checkin_df,
    browser_reviews_df,
    tips_df,
    mobile_reviews_df,
    datetime(2022, 4, 14)
  )

  expected_json = read_json()
  assert data_frame_to_json(actual_df) == expected_json
  #overwrite_expected(actual_df)
```

Give it a better name

```
def test_will_do_the_right_thing(spark: SparkSession) -> None:

  browser_reviews_df = create_df_from_json("fixtures/browser_reviews.json", spark)
  checkin_df    = create_df_from_json("fixtures/checkin.json", spark)
  tips_df       = create_df_from_json("fixtures/tips.json", spark)
  business_df   = create_df_from_json("fixtures/business.json", spark)
  mobile_reviews_df = create_df_from_json("fixtures/mobile_reviews.json", spark)

  actual_df = transform(
    business_df,
    checkin_df,
    browser_reviews_df,
    tips_df,
    mobile_reviews_df,
    datetime(2022, 4, 14)
  )

  expected_json = read_json()
  assert data_frame_to_json(actual_df) == expected_json
  #overwrite_expected(actual_df)
```

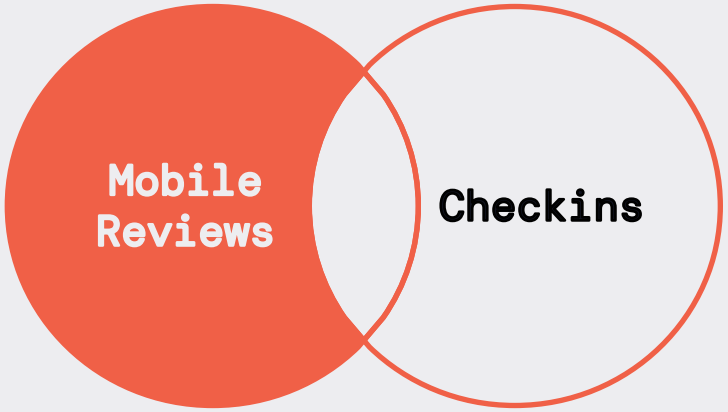
This is the bug



1 Mobile Review + 0 Checkins = 0 Reviews

1 Mobile Review + 1 Checkin = 1 Review

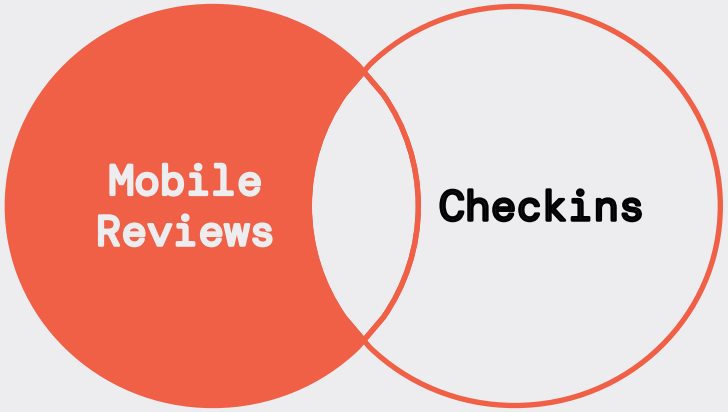
This is what we want

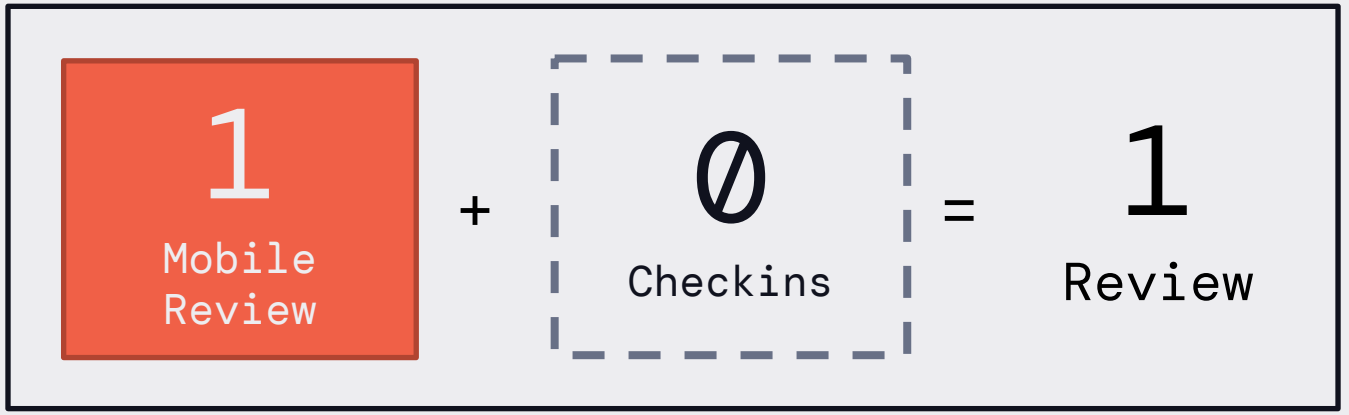


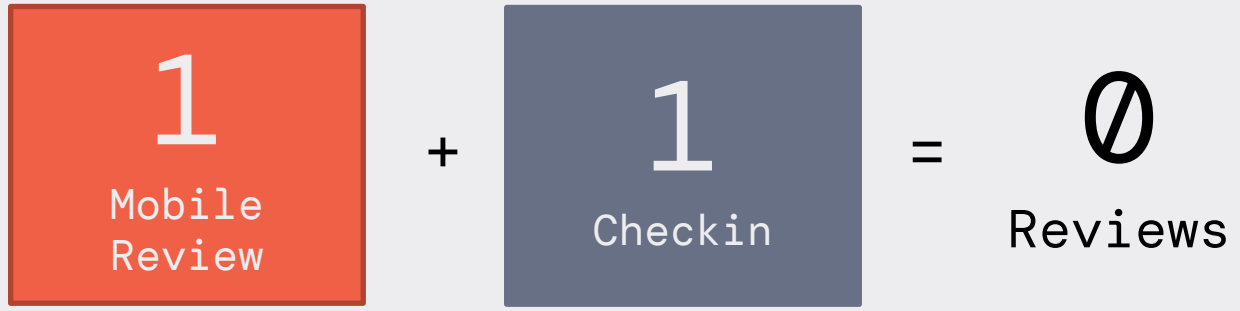
1 Mobile Review + 0 Checkins = 1 Review

1 Mobile Review + 1 Checkin = 0 Reviews

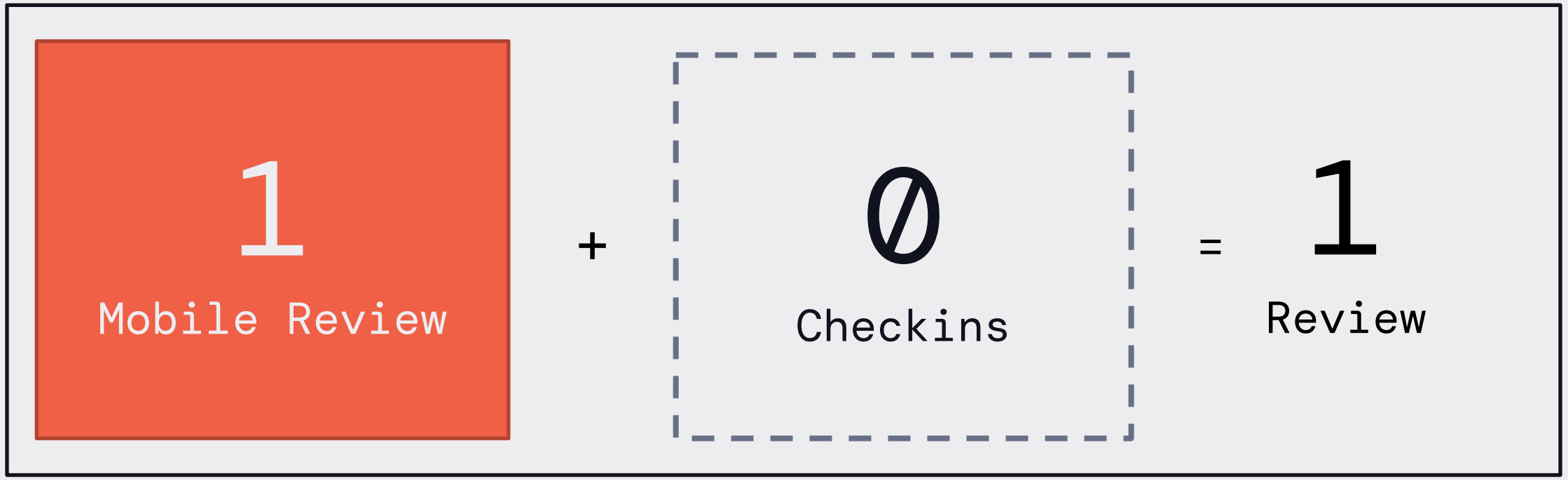
This is what we want







This is what we want



Give it a better name

```
def test_will_count_mobile_reviews_without_matching_checkins(spark: SparkSession) -> None:

  browser_reviews_df = create_df_from_json("fixtures/browser_reviews.json", spark)
  checkin_df    = create_df_from_json("fixtures/checkin.json", spark)
  tips_df       = create_df_from_json("fixtures/tips.json", spark)
  business_df   = create_df_from_json("fixtures/business.json", spark)
  mobile_reviews_df = create_df_from_json("fixtures/mobile_reviews.json", spark)

  actual_df = transform(
    business_df,
    checkin_df,
    browser_reviews_df,
    tips_df,
    mobile_reviews_df,
    datetime(2022, 4, 14)
  )

  expected_json = read_json()
  assert data_frame_to_json(actual_df) == expected_json
  #overwrite_expected(actual_df)
```

Check out the test input

```
def test_will_count_mobile_reviews_without_matching_checkins(spark: SparkSession) -> None:

  browser_reviews_df = create_df_from_json("fixtures/browser_reviews.json", spark)
  checkin_df    = create_df_from_json("fixtures/checkin.json", spark)
  tips_df       = create_df_from_json("fixtures/tips.json", spark)
  business_df   = create_df_from_json("fixtures/business.json", spark)
  mobile_reviews_df = create_df_from_json("fixtures/mobile_reviews.json", spark)

  actual_df = transform(
    business_df,
    checkin_df,
    browser_reviews_df,
    tips_df,
    mobile_reviews_df,
    datetime(2022, 4, 14)
  )

  expected_json = read_json()
  assert data_frame_to_json(actual_df) == expected_json
  #overwrite_expected(actual_df)
```

mobile_reviews.json


```
[
  {
    "review_id": "XQfwVwDr-v0ZS3_CbbE5Xw",
    "user_id": "mh_-eMZ6K5RLWhZyISBhwA",
    "business_id": "Pns2l4eNsfO8kk83dixA6A",
    "stars": 3.0,
    "useful": 0,
    "funny": 0,
    "cool": 0,
    "text": "If you decide to eat here, just be aware it is going to take about 2 hours from beginning to end. We have tried it multiple times, because I want to like it! I have been to its other locations in NJ and never had a bad experience. \n\nThe food is good, but it takes a very long time to come out. The waitstaff is very young, but usually pleasant. We have just had too many experiences where we spent way too long waiting. We usually opt for another diner or restaurant on the weekends, in order to be done quicker.",
    "date": "2018-07-07 22:09:11"
  },
  {
    "review_id": "7ATYjTIgM3jUlt4UM3IypQ",
    "user_id": "OyoGAe7OKpv6SyGZT5g77Q",
    "business_id": "qkRM_2X51Yqxk3btlwAQIg",
    "stars": 5.0,
    "useful": 1,
    "funny": 0,
    "cool": 1,
    "text": "Ive taken a lot of spin classes over the years, and nothing compares to the classes
```

```
    "user_id": "59MxRhNVhU9MYndMkz0wtw",
    "business_id": "pUycOfUwM8vqX7KjRRhUEA",
    "stars": 3.0,
    "useful": 0,
    "funny": 0,
    "cool": 0,
    "text": "Had a party of 6 here for hibachi. Our waitress brought our separate sushi orders on
one plate so we couldnt really tell whos was whos and forgot several items on an order. I
understand making mistakes but the restaraunt was really quiet so we were kind of surprised.
Usually hibachi is a fun lively experience and our  cook  said maybe three words, but he cooked
very well his name was Francisco. Service was fishy, food was pretty good, and im hoping it was
just an off night here. But for the money I wouldnt go back.",
    "date": "2016-07-25 07:31:06"
  },
 {
    "review_id": "be02921c1ccd4c5dabdc63bd29f3bd35",
    "user_id": "0dea283273fa4f40ac12ea210994d5f8",
    "business_id": "41189bbb4fb446a586b0d69aba01d39b",
    "stars": 5.0,
    "useful": 1,
    "funny": 0,
    "cool": 0,
    "text": "They have the absolute best pizza on this entire planet. Gooey and greasy, without the
soggy crust, they are allergy friendly and know what is in all their ingredients and their dough!",
    "date": "2022-04-14 07:31:06"
  }
]
```

```
    "user_id": "59MxRhNVhU9MYndMkz0wtw",
    "business_id": "pUycOfUwM8vqX7KjRRhUEA",
    "stars": 3.0,
    "useful": 0,
    "funny": 0,
    "cool": 0,
    "text": "Had a party of 6 here for hibachi. Our waitress brought our separate sushi orders on
one plate so we couldnt really tell whos was whos and forgot several items on an order. I
understand making mistakes but the restaraunt was really quiet so we were kind of surprised.
Usually hibachi is a fun lively experience and our  cook  said maybe three words, but he cooked
very well his name was Francisco. Service was fishy, food was pretty good, and im hoping it was
just an off night here. But for the money I wouldnt go back.",
    "date": "2016-07-25 07:31:06"
  },
 {
    "review_id": "be02921c1ccd4c5dabdc63bd29f3bd35",
    "user_id": "0dea283273fa4f40ac12ea210994d5f8",
    "business_id": "41189bbb4fb446a586b0d69aba01d39b",
    "stars": 5.0,
    "useful": 1,
    "funny": 0,
    "cool": 0,
    "text": "They have the absolute best pizza on this entire planet. Gooey and greasy, without the
soggy crust, they are allergy friendly and know what is in all their ingredients and their dough!",
    "date": "2022-04-14 07:31:06"
  }
]
```

```
    "user_id": "59MxRhNVhU9MYndMkz0wtw",
    "business_id": "pUycOfUwM8vqX7KjRRhUEA",
    "stars": 3.0,
    "useful": 0,
    "funny": 0,
    "cool": 0,
    "text": "Had a party of 6 here for hibachi. Our waitress brought our separate sushi orders on
one plate so we couldnt really tell whos was whos and forgot several items on an order. I
understand making mistakes but the restaraunt was really quiet so we were kind of surprised.
Usually hibachi is a fun lively experience and our  cook  said maybe three words, but he cooked
very well his name was Francisco. Service was fishy, food was pretty good, and im hoping it was
just an off night here. But for the money I wouldnt go back.",
    "date": "2016-07-25 07:31:06"
  },
 {
    "review_id": "be02921c1ccd4c5dabdc63bd29f3bd35",
    "user_id": "0dea283273fa4f40ac12ea210994d5f8",
    "business_id": "mobile_review_no_checkins",
    "stars": 5.0,
    "useful": 1,
    "funny": 0,
    "cool": 0,
    "text": "They have the absolute best pizza on this entire planet. Gooey and greasy, without the
soggy crust, they are allergy friendly and know what is in all their ingredients and their dough!",
    "date": "2022-04-14 07:31:06"
  }
]
```

Make Sure the Test Passes

```
def test_will_count_mobile_reviews_without_matching_checkins(spark: SparkSession) -> None:

  browser_reviews_df = create_df_from_json("fixtures/browser_reviews.json", spark)
  checkin_df    = create_df_from_json("fixtures/checkin.json", spark)
  tips_df       = create_df_from_json("fixtures/tips.json", spark)
  business_df   = create_df_from_json("fixtures/business.json", spark)
  mobile_reviews_df = create_df_from_json("fixtures/mobile_reviews.json", spark)

  actual_df = transform(
    business_df,
    checkin_df,
    browser_reviews_df,
    tips_df,
    mobile_reviews_df,
    datetime(2022, 4, 14)
  )

  expected_json = read_json()
  assert data_frame_to_json(actual_df) == expected_json
  #overwrite_expected(actual_df)
```

Arrange, Act, Assert

```
def test_will_count_mobile_reviews_without_matching_checkins(spark: SparkSession) -> None:

    browser_reviews_df = create_df_from_json("fixtures/browser_reviews.json", spark)
    checkin_df    = create_df_from_json("fixtures/checkin.json", spark)
    tips_df       = create_df_from_json("fixtures/tips.json", spark)
    business_df   = create_df_from_json("fixtures/business.json", spark)
    mobile_reviews_df = create_df_from_json("fixtures/mobile_reviews.json", spark)

    actual_df = transform(
      business_df,
      checkin_df,
      browser_reviews_df,
      tips_df,
      mobile_reviews_df,
      datetime(2022, 4, 14)
    )

    expected_json = read_json()
    assert data_frame_to_json(actual_df) == expected_json
    #overwrite_expected(actual_df)
```

Make the assertion specific

```
def test_will_count_mobile_reviews_without_matching_checkins(spark: SparkSession) -> None:

  browser_reviews_df = create_df_from_json("fixtures/browser_reviews.json", spark)
  checkin_df    = create_df_from_json("fixtures/checkin.json", spark)
  tips_df       = create_df_from_json("fixtures/tips.json", spark)
  business_df   = create_df_from_json("fixtures/business.json", spark)
  mobile_reviews_df = create_df_from_json("fixtures/mobile_reviews.json", spark)

  actual_df = transform(
    business_df,
    checkin_df,
    browser_reviews_df,
    tips_df,
    mobile_reviews_df,
    datetime(2022, 4, 14)
  )

  expected_json = read_json()
  assert data_frame_to_json(actual_df) == expected_json
  #overwrite_expected(actual_df)
```

Make the assertion specific

```
expected_json = read_json()
assert data_frame_to_json(actual_df) == expected_json
#overwrite_expected(actual_df)
```

Make the assertion specific

```
expected_json = read_json()
assert data_frame_to_json(actual_df) == expected_json
```

Make the assertion specific

```
expected_json = read_json()
assert data_frame_to_json(actual_df) == expected_json
```

actual_df


```
[
  {
    "business_id": "mobile_review_no_checkins",
    "name": ”Inglewood Pizza",
    "num_tips": 0,
    "num_checkins": 0,
    "num_reviews": 1,
    "num_interactions": 1,
    "dt": "2022-04-14"
  },
 {
    "business_id": "MTSW4McQd7CbVtyjqoe9mw",
    "name": "St Honore Pastries",
    "num_tips": 0,
    "num_checkins": 13,
    "num_reviews": 1,
    "num_interactions": 14,
    "dt": "2022-04-14"
  },
  {
    "business_id": "Pns2l4eNsfO8kk83dixA6A",
    "name": "Abby Rappoport, LAC, CMQ",
    "num_tips": 1,
```

actual_df


```
[
  {
    "business_id": "mobile_review_no_checkins",
    "name": ”Inglewood Pizza",
    "num_tips": 0,
    "num_checkins": 0,
    "num_reviews": 1,
    "num_interactions": 1,
    "dt": "2022-04-14"
  },
  {
    "business_id": "MTSW4McQd7CbVtyjqoe9mw",
    "name": "St Honore Pastries",
    "num_tips": 0,
    "num_checkins": 13,
    "num_reviews": 1,
    "num_interactions": 14,
    "dt": "2022-04-14"
  },
  {
    "business_id": "Pns2l4eNsfO8kk83dixA6A",
    "name": "Abby Rappoport, LAC, CMQ",
    "num_tips": 1,
```

Make the assertion specific

```
expected_json = read_json()
assert data_frame_to_json(actual_df) == expected_json
```

```
{
  "business_id": "mobile_review_no_checkins",
  "name": ”Inglewood Pizza",
  "num_tips": 0,
  "num_checkins": 0,
  "num_reviews": 1,
  "num_interactions": 1,
  "dt": "2022-04-14"
},
```

Make the assertion specific

```
expected_json = read_json()
assert data_frame_to_json(actual_df)[0] == expected_json[0]
```

```
{
  "business_id": "mobile_review_no_checkins",
  "name": ”Inglewood Pizza",
  "num_tips": 0,
  "num_checkins": 0,
  "num_reviews": 1,
  "num_interactions": 1,
  "dt": "2022-04-14"
},
```

Make the assertion specific

```
expected_json = read_json()
assert data_frame_to_json(actual_df)[0] == expected_json[0]
```

```
{
  "business_id": "mobile_review_no_checkins",
  "name": ”Inglewood Pizza",
  "num_tips": 0,
  "num_checkins": 0,
  "num_reviews": 1,
  "num_interactions": 1,
  "dt": "2022-04-14"
},
```

Make the assertion specific

```
assert data_frame_to_json(actual_df)[0] == expected_json[0]
```

```
{
  "business_id": "mobile_review_no_checkins",
  "name": ”Inglewood Pizza",
  "num_tips": 0,
  "num_checkins": 0,
  "num_reviews": 1,
  "num_interactions": 1,
  "dt": "2022-04-14"
},
```

Make the assertion specific

```
assert data_frame_to_json(actual_df)[0]["business_id"] == "mobile_review_no_checkins"
assert data_frame_to_json(actual_df)[0]["num_reviews"] == 1
```


```
{
  "business_id": "mobile_review_no_checkins",
  "name": ”Inglewood Pizza",
  "num_tips": 0,
  "num_checkins": 0,
  "num_reviews": 1,
  "num_interactions": 1,
  "dt": "2022-04-14"
},
```

Make the assertion specific

```
assert data_frame_to_json(actual_df)[0]["business_id"] == "mobile_review_no_checkins"
assert data_frame_to_json(actual_df)[0]["num_reviews"] == 1
```

Make the assertion specific

```
assert_inglewood_pizza_has_correct_review_count(actual_df, 1)
```

```
def assert_inglewood_pizza_has_correct_review_count(actual_df, review_count):
  assert data_frame_to_json(actual_df)[0]["business_id"] == "mobile_review_no_checkins"
  assert data_frame_to_json(actual_df)[0]["num_reviews"] == review_count
```

Make the assertion specific

```
assert_inglewood_pizza_has_correct_review_count(actual_df, 1)
```

```
def assert_inglewood_pizza_has_correct_review_count(actual_df, review_count):
  assert data_frame_to_json(actual_df)[0]["business_id"] == "mobile_review_no_checkins"
  assert data_frame_to_json(actual_df)[0]["num_reviews"] == review_count
```

Make the assertion specific

```
def test_mobile_reviews_count_without_checkins(spark: SparkSession) -> None:
    browser_reviews_df  = create_df_from_json("fixtures/browser_reviews.json", spark)
    checkin_df      = create_df_from_json("fixtures/checkin.json", spark)
    tips_df         = create_df_from_json("fixtures/tips.json", spark)
    business_df     = create_df_from_json("fixtures/business.json", spark)
    mobile_reviews_df = create_df_from_json("fixtures/mobile_reviews.json", spark)


    actual_df = transform(
        business_df,
        checkin_df,
        browser_reviews_df,
        tips_df,
        mobile_reviews_df,
        datetime(2022, 4, 14)
    )
```

```
assert inglewood_pizza_has_correct_review_count(actual_df, 1)
```

Squeeze the Bottom

```
def test_mobile_reviews_count_without_checkins(spark: SparkSession) -> None:
  browser_reviews_df   = create_df_from_json("fixtures/browser_reviews.json", spark)
  checkin_df   = create_df_from_json("fixtures/checkin.json", spark)
  tips_df      = create_df_from_json("fixtures/tips.json", spark)
  business_df  = create_df_from_json("fixtures/business.json", spark)
  mobile_reviews_df = create_df_from_json("fixtures/mobile_reviews.json", spark)



  actual_df = transform(
       business_df,
       checkin_df,
       browser_reviews_df,
       tips_df,
       mobile_reviews_df,
       datetime(2022, 4, 14)
   )


  assert_inglewood_pizza_has_correct_review_count(actual_df, 1)
```

Squeeze the Bottom

```
def test_mobile_reviews_count_without_checkins(spark: SparkSession) -> None:
  browser_reviews_df   = create_df_from_json("fixtures/browser_reviews.json", spark)
  checkin_df    = create_df_from_json("fixtures/checkin.json", spark)
  tips_df       = create_df_from_json("fixtures/tips.json", spark)
  business_df   = create_df_from_json("fixtures/business.json", spark)
  mobile_reviews_df = create_df_from_json("fixtures/mobile_reviews.json", spark)
```

```
actual_df = transform(
      business_df,
      checkin_df,
      browser_reviews_df,
      tips_df,
      mobile_reviews_df,
      datetime(2022, 4, 14)
 )
```

```
  assert_inglewood_pizza_has_correct_review_count(actual_df, 1)
```

Squeeze the Bottom

```
def transform(
                business_df: DataFrame,
                checkin_df: DataFrame,
                browser_reviews_df: DataFrame,
                tips_df: DataFrame,
                mobile_reviews_df: DataFrame,
                run_date: datetime = datetime.today()
):

 checkin_df = create_checkin_df_with_one_date_per_row(checkin_df)

 reviews_df = count_reviews(checkin_df, mobile_reviews_df, browser_reviews_df, run_date)
 checkin_df = count_checkins(checkin_df, run_date)
 tips_df = count_tips(tips_df, run_date)

 return construct_post_pandemic_recovery_df(
     business_df, checkin_df, reviews_df, run_date, tips_df
 )
```

Squeeze the Bottom

```
def test_mobile_reviews_count_without_checkins(spark: SparkSession) -> None:
  browser_reviews_df   = create_df_from_json("fixtures/browser_reviews.json", spark)
  checkin_df    = create_df_from_json("fixtures/checkin.json", spark)
  tips_df       = create_df_from_json("fixtures/tips.json", spark)
  business_df   = create_df_from_json("fixtures/business.json", spark)
  mobile_reviews_df = create_df_from_json("fixtures/mobile_reviews.json", spark)



  actual_df = transform(
       business_df,
       checkin_df,
       browser_reviews_df,
       tips_df,
       mobile_reviews_df,
       datetime(2022, 4, 14)
   )


  assert_inglewood_pizza_has_correct_review_count(actual_df, 1)
```

Squeeze the Bottom

```
def test_mobile_reviews_count_without_checkins(spark: SparkSession) -> None:
  browser_reviews_df   = create_df_from_json("fixtures/browser_reviews.json", spark)
  checkin_df   = create_df_from_json("fixtures/checkin.json", spark)
  tips_df      = create_df_from_json("fixtures/tips.json", spark)
  business_df  = create_df_from_json("fixtures/business.json", spark)
  mobile_reviews_df = create_df_from_json("fixtures/mobile_reviews.json", spark)
```

```
checkin_df = create_checkin_df_with_one_date_per_row(checkin_df)

reviews_df = count_reviews(checkin_df, mobile_reviews_df, browser_reviews_df, run_date)
checkin_df = count_checkins(checkin_df, run_date)
tips_df = count_tips(tips_df, run_date)

actual_df = construct_post_pandemic_recovery_df(
    business_df, checkin_df, reviews_df, run_date, tips_df
)
```

```
assert_inglewood_pizza_has_correct_review_count(actual_df, 1)
```

Squeeze the Bottom

```
def test_mobile_reviews_count_without_checkins(spark: SparkSession) -> None:
  browser_reviews_df   = create_df_from_json("fixtures/browser_reviews.json", spark)
  checkin_df   = create_df_from_json("fixtures/checkin.json", spark)
  tips_df      = create_df_from_json("fixtures/tips.json", spark)
  business_df  = create_df_from_json("fixtures/business.json", spark)
  mobile_reviews_df = create_df_from_json("fixtures/mobile_reviews.json", spark)
```

```
checkin_df = create_checkin_df_with_one_date_per_row(checkin_df)

reviews_df = count_reviews(checkin_df, mobile_reviews_df, browser_reviews_df, run_date)
checkin_df = count_checkins(checkin_df, run_date)
tips_df = count_tips(tips_df, run_date)

actual_df = construct_post_pandemic_recovery_df(
    business_df, checkin_df, reviews_df, run_date, tips_df
)
```

```
assert_inglewood_pizza_has_correct_review_count(actual_df, 1)
```

Squeeze the Bottom

```
def test_mobile_reviews_count_without_checkins(spark: SparkSession) -> None:
  browser_reviews_df   = create_df_from_json("fixtures/browser_reviews.json", spark)
  checkin_df   = create_df_from_json("fixtures/checkin.json", spark)
  tips_df      = create_df_from_json("fixtures/tips.json", spark)
  business_df  = create_df_from_json("fixtures/business.json", spark)
  mobile_reviews_df = create_df_from_json("fixtures/mobile_reviews.json", spark)


  checkin_df = create_checkin_df_with_one_date_per_row(checkin_df)

  reviews_df = count_reviews(checkin_df, mobile_reviews_df, browser_reviews_df, run_date)
  checkin_df = count_checkins(checkin_df, run_date)
  tips_df = count_tips(tips_df, run_date)

  assert_inglewood_pizza_has_correct_review_count(actual_df, 1)

  actual_df = construct_post_pandemic_recovery_df(
      business_df, checkin_df, reviews_df, run_date, tips_df
  )
```

Squeeze the Bottom

```
def test_mobile_reviews_count_without_checkins(spark: SparkSession) -> None:
  browser_reviews_df   = create_df_from_json("fixtures/browser_reviews.json", spark)
  checkin_df    = create_df_from_json("fixtures/checkin.json", spark)
  tips_df       = create_df_from_json("fixtures/tips.json", spark)
  business_df   = create_df_from_json("fixtures/business.json", spark)
  mobile_reviews_df = create_df_from_json("fixtures/mobile_reviews.json", spark)


  checkin_df = create_checkin_df_with_one_date_per_row(checkin_df)

  reviews_df = count_reviews(checkin_df, mobile_reviews_df, browser_reviews_df, run_date)
  checkin_df = count_checkins(checkin_df, run_date)
  tips_df = count_tips(tips_df, run_date)


 actual_df = construct_post_pandemic_recovery_df(
     business_df, checkin_df, reviews_df, run_date, tips_df
 )


  assert_inglewood_pizza_has_correct_review_count(actual_df, 1)
```

Squeeze the Bottom

```
actual_df = construct_post_pandemic_recovery_df(
    business_df, checkin_df, reviews_df, run_date, tips_df
)
```

```
assert_inglewood_pizza_has_correct_review_count(actual_df, 1)
```

Squeeze the Bottom

```
pandemic_recovery_df = business_df.join(
                            checkin_df, on="business_id", how='left').fillna(0)
pandemic_recovery_df = pandemic_recovery_df.join(
                            reviews_df, on="business_id",how='left').fillna(0)
pandemic_recovery_df = pandemic_recovery_df.join(
                            tips_df, on="business_id",how='left').fillna(0)

pandemic_recovery_df = pandemic_recovery_df.withColumn(
                            "num_interactions",
                            pandemic_recovery_df.num_reviews +
                            pandemic_recovery_df.num_tips +
                            pandemic_recovery_df.num_checkins)
pandemic_recovery_df = pandemic_recovery_df.withColumn(
                            "dt", F.lit(run_date.strftime("%Y-%m-%d")))
pandemic_recovery_df = select_important_columns(pandemic_recovery_df)

actual_df = pandemic_recovery_df
```

```
assert_inglewood_pizza_has_correct_review_count(actual_df, 1)
```

Squeeze the Bottom

```
pandemic_recovery_df = business_df.join(
                            checkin_df, on="business_id", how='left').fillna(0)
pandemic_recovery_df = pandemic_recovery_df.join(
                            reviews_df, on="business_id",how='left').fillna(0)
pandemic_recovery_df = pandemic_recovery_df.join(
                            tips_df, on="business_id",how='left').fillna(0)

pandemic_recovery_df = pandemic_recovery_df.withColumn(
                            "num_interactions",
                            pandemic_recovery_df.num_reviews +
                            pandemic_recovery_df.num_tips +
                            pandemic_recovery_df.num_checkins)
pandemic_recovery_df = pandemic_recovery_df.withColumn(
                            "dt", F.lit(run_date.strftime("%Y-%m-%d")))
pandemic_recovery_df = select_important_columns(pandemic_recovery_df)

actual_df = pandemic_recovery_df
```

```
assert_inglewood_pizza_has_correct_review_count(actual_df, 1)
```

Squeeze the Bottom

```
pandemic_recovery_df = business_df.join(
                            checkin_df, on="business_id", how='left').fillna(0)
pandemic_recovery_df = pandemic_recovery_df.join(
                            reviews_df, on="business_id",how='left').fillna(0)
pandemic_recovery_df = pandemic_recovery_df.join(
                            tips_df, on="business_id",how='left').fillna(0)

pandemic_recovery_df = pandemic_recovery_df.withColumn(
                            "num_interactions",
                            pandemic_recovery_df.num_reviews +
                            pandemic_recovery_df.num_tips +
                            pandemic_recovery_df.num_checkins)
pandemic_recovery_df = pandemic_recovery_df.withColumn(
                            "dt", F.lit(run_date.strftime("%Y-%m-%d")))
pandemic_recovery_df = select_important_columns(pandemic_recovery_df)

actual_df = pandemic_recovery_df
```

```
assert_inglewood_pizza_has_correct_review_count(pandemic_recovery_df, 1)
```

Squeeze the Bottom

```
pandemic_recovery_df = business_df.join(
                        checkin_df, on="business_id", how='left').fillna(0)
pandemic_recovery_df = pandemic_recovery_df.join(
                        reviews_df, on="business_id",how='left').fillna(0)
pandemic_recovery_df = pandemic_recovery_df.join(
                        tips_df, on="business_id",how='left').fillna(0)

pandemic_recovery_df = pandemic_recovery_df.withColumn(
                        "num_interactions",
                        pandemic_recovery_df.num_reviews +
                        pandemic_recovery_df.num_tips +
                        pandemic_recovery_df.num_checkins)
pandemic_recovery_df = pandemic_recovery_df.withColumn(
                        "dt", F.lit(run_date.strftime("%Y-%m-%d")))
pandemic_recovery_df = select_important_columns(pandemic_recovery_df)
```

```
assert_inglewood_pizza_has_correct_review_count(pandemic_recovery_df, 1)
```

Squeeze the Bottom

```
pandemic_recovery_df = business_df.join(
                        checkin_df, on="business_id", how='left').fillna(0)
pandemic_recovery_df = pandemic_recovery_df.join(
                        reviews_df, on="business_id",how='left').fillna(0)
pandemic_recovery_df = pandemic_recovery_df.join(
                        tips_df, on="business_id",how='left').fillna(0)

pandemic_recovery_df = pandemic_recovery_df.withColumn(
                        "num_interactions",
                        pandemic_recovery_df.num_reviews +
                        pandemic_recovery_df.num_tips +
                        pandemic_recovery_df.num_checkins)
pandemic_recovery_df = pandemic_recovery_df.withColumn(
                        "dt", F.lit(run_date.strftime("%Y-%m-%d")))
```

```
assert_inglewood_pizza_has_correct_review_count(pandemic_recovery_df, 1)
```

```
pandemic_recovery_df = select_important_columns(pandemic_recovery_df)
```

Squeeze the Bottom

```
pandemic_recovery_df = business_df.join(
                            checkin_df, on="business_id", how='left').fillna(0)
pandemic_recovery_df = pandemic_recovery_df.join(
                            reviews_df, on="business_id",how='left').fillna(0)
pandemic_recovery_df = pandemic_recovery_df.join(
                            tips_df, on="business_id",how='left').fillna(0)

pandemic_recovery_df = pandemic_recovery_df.withColumn(
                            "num_interactions",
                            pandemic_recovery_df.num_reviews +
                            pandemic_recovery_df.num_tips +
                            pandemic_recovery_df.num_checkins)
pandemic_recovery_df = pandemic_recovery_df.withColumn(
                            "dt", F.lit(run_date.strftime("%Y-%m-%d")))
```

```
assert_inglewood_pizza_has_correct_review_count(pandemic_recovery_df, 1)
```

Squeeze the Bottom

```
pandemic_recovery_df = business_df.join(
                            checkin_df, on="business_id", how='left').fillna(0)
pandemic_recovery_df = pandemic_recovery_df.join(
                            reviews_df, on="business_id",how='left').fillna(0)
pandemic_recovery_df = pandemic_recovery_df.join(
                            tips_df, on="business_id",how='left').fillna(0)

pandemic_recovery_df = pandemic_recovery_df.withColumn(
                            "num_interactions",
                            pandemic_recovery_df.num_reviews +
                            pandemic_recovery_df.num_tips +
                            pandemic_recovery_df.num_checkins)
```

```
assert_inglewood_pizza_has_correct_review_count(pandemic_recovery_df, 1)
```

```
pandemic_recovery_df = pandemic_recovery_df.withColumn(
                            "dt", F.lit(run_date.strftime("%Y-%m-%d")))
```

Squeeze the Bottom

```
pandemic_recovery_df = business_df.join(
                            checkin_df, on="business_id", how='left').fillna(0)
pandemic_recovery_df = pandemic_recovery_df.join(
                            reviews_df, on="business_id",how='left').fillna(0)
pandemic_recovery_df = pandemic_recovery_df.join(
                            tips_df, on="business_id",how='left').fillna(0)

pandemic_recovery_df = pandemic_recovery_df.withColumn(
                            "num_interactions",
                            pandemic_recovery_df.num_reviews +
                            pandemic_recovery_df.num_tips +
                            pandemic_recovery_df.num_checkins)
```

```
assert_inglewood_pizza_has_correct_review_count(pandemic_recovery_df, 1)
```

Squeeze the Bottom

```
pandemic_recovery_df = business_df.join(
                            checkin_df, on="business_id", how='left').fillna(0)
pandemic_recovery_df = pandemic_recovery_df.join(
                            reviews_df, on="business_id",how='left').fillna(0)
pandemic_recovery_df = pandemic_recovery_df.join(
                            tips_df, on="business_id",how='left').fillna(0)
```

```
assert_inglewood_pizza_has_correct_review_count(pandemic_recovery_df, 1)
```

```
pandemic_recovery_df = pandemic_recovery_df.withColumn(
                            "num_interactions",
                            pandemic_recovery_df.num_reviews +
                            pandemic_recovery_df.num_tips +
                            pandemic_recovery_df.num_checkins)
```

Squeeze the Bottom

```
pandemic_recovery_df = business_df.join(
                            checkin_df, on="business_id", how='left').fillna(0)
pandemic_recovery_df = pandemic_recovery_df.join(
                            reviews_df, on="business_id",how='left').fillna(0)
pandemic_recovery_df = pandemic_recovery_df.join(
                            tips_df, on="business_id",how='left').fillna(0)
```

```
assert_inglewood_pizza_has_correct_review_count(pandemic_recovery_df, 1)
```

Squeeze the Bottom

```
tips_df = count_tips(tips_df, run_date)

pandemic_recovery_df = business_df.join(
                            checkin_df, on="business_id", how='left').fillna(0)
pandemic_recovery_df = pandemic_recovery_df.join(
                            reviews_df, on="business_id",how='left').fillna(0)
```

```
assert_inglewood_pizza_has_correct_review_count(pandemic_recovery_df, 1)
```

```
pandemic_recovery_df = pandemic_recovery_df.join(
                            tips_df, on="business_id",how='left').fillna(0)
```

Squeeze the Bottom

```
tips_df = count_tips(tips_df, run_date)

pandemic_recovery_df = business_df.join(
                            checkin_df, on="business_id", how='left').fillna(0)
pandemic_recovery_df = pandemic_recovery_df.join(
                            reviews_df, on="business_id",how='left').fillna(0)
```

```
assert_inglewood_pizza_has_correct_review_count(pandemic_recovery_df, 1)
```

Squeeze the Bottom

```
tips_df = count_tips(tips_df, run_date)

pandemic_recovery_df = business_df.join(
                            checkin_df, on="business_id", how='left').fillna(0)

assert_inglewood_pizza_has_correct_review_count(pandemic_recovery_df, 1)

pandemic_recovery_df = pandemic_recovery_df.join(
                            reviews_df, on="business_id",how='left').fillna(0)
```

Squeeze the Bottom

```
tips_df = count_tips(tips_df, run_date)

pandemic_recovery_df = business_df.join(
                            checkin_df, on="business_id", how='left').fillna(0)
pandemic_recovery_df = pandemic_recovery_df.join(
                            reviews_df, on="business_id",how='left').fillna(0)
```

```
assert_inglewood_pizza_has_correct_review_count(pandemic_recovery_df, 1)
```

Look at the error

```
pandemic_recovery_df = business_df.join(checkin_df, on="business_id", how='left').fillna(0)
>       assert_inglewood_pizza_has_correct_review_count(pandemic_recovery_df, 1)
E       KeyError: 'num_reviews'
```

Look at the error

```
pandemic_recovery_df = business_df.join(checkin_df, on="business_id", how='left').fillna(0)
>       assert_inglewood_pizza_has_correct_review_count(pandemic_recovery_df, 1)
E       KeyError: 'num_reviews'
```

Squeeze the Bottom

```
tips_df = count_tips(tips_df, run_date)

pandemic_recovery_df = business_df.join(
                            checkin_df, on="business_id", how='left').fillna(0)
pandemic_recovery_df = pandemic_recovery_df.join(
                            reviews_df, on="business_id",how='left').fillna(0)
```

```
assert_inglewood_pizza_has_correct_review_count(pandemic_recovery_df, 1)
```

Squeeze the Bottom

```
tips_df = count_tips(tips_df, run_date)

pandemic_recovery_df = business_df.join(
                            checkin_df, on="business_id", how='left').fillna(0)
```

```
pandemic_recovery_df.printSchema()
reviews_df.printSchema()
```

```
pandemic_recovery_df = pandemic_recovery_df.join(
                            reviews_df, on="business_id",how='left').fillna(0)
```

```
assert_inglewood_pizza_has_correct_review_count(pandemic_recovery_df, 1)
```

pandemic_recovery_df.printSchema()

```
root
 |-- business_id: string (nullable = true)
 |-- address: string (nullable = true)
 |-- attributes: struct (nullable = true)
 |    |-- Alcohol: string (nullable = true)
 |    |-- Ambience: string (nullable = true)
 |    |-- BikeParking: string (nullable = true)
 |    |-- BusinessAcceptsCreditCards: string (nullable = true)
 |    |-- BusinessParking: string (nullable = true)
 |    |-- ByAppointmentOnly: string (nullable = true)
 |    |-- Caters: string (nullable = true)
 |    |-- CoatCheck: string (nullable = true)
 |    |-- DogsAllowed: string (nullable = true)
 |    |-- DriveThru: string (nullable = true)
 |    |-- GoodForKids: string (nullable = true)
 |    |-- HappyHour: string (nullable = true)
 |    |-- HasTV: string (nullable = true)
 |    |-- NoiseLevel: string (nullable = true)
 |    |-- OutdoorSeating: string (nullable = true)
 |    |-- RestaurantsAttire: string (nullable = true)
 |    |-- RestaurantsDelivery: string (nullable = true)
 |    |-- RestaurantsGoodForGroups: string (nullable = true)
 |    |-- RestaurantsPriceRange2: string (nullable = true)
```

pandemic_recovery_df.printSchema()

```
|    |-- RestaurantsTakeOut: string (nullable = true)
|    |-- WheelchairAccessible: string (nullable = true)
|    |-- WiFi: string (nullable = true)
|-- categories: string (nullable = true)
|-- city: string (nullable = true)
|-- hours: struct (nullable = true)
|    |-- Friday: string (nullable = true)
|    |-- Monday: string (nullable = true)
|    |-- Saturday: string (nullable = true)
|    |-- Sunday: string (nullable = true)
|    |-- Thursday: string (nullable = true)
|    |-- Tuesday: string (nullable = true)
|    |-- Wednesday: string (nullable = true)
|-- is_open: long (nullable = true)
|-- latitude: double (nullable = false)
|-- longitude: double (nullable = false)
|-- name: string (nullable = true)
|-- postal_code: string (nullable = true)
|-- review_count: long (nullable = true)
|-- stars: double (nullable = false)
|-- state: string (nullable = true)
|-- num_checkins: long (nullable = true)
```

reviews_df

```
root
 |-- business_id: string (nullable = true)
 |-- num_reviews: long (nullable = false)
```

Squeeze the Bottom

```
tips_df = count_tips(tips_df, run_date)

pandemic_recovery_df = business_df.join(
                            checkin_df, on="business_id", how='left').fillna(0)
pandemic_recovery_df = pandemic_recovery_df.join(
                            reviews_df, on="business_id",how='left').fillna(0)
```

```
assert_inglewood_pizza_has_correct_review_count(pandemic_recovery_df, 1)
```

Squeeze the Bottom

```
tips_df = count_tips(tips_df, run_date)

pandemic_recovery_df = business_df.join(
                            checkin_df, on="business_id", how='left').fillna(0)
pandemic_recovery_df = pandemic_recovery_df.join(
                            reviews_df, on="business_id",how='left').fillna(0)
```

```
assert_inglewood_pizza_has_correct_review_count(pandemic_recovery_df, 1)
```

Squeeze the Bottom

```
tips_df = count_tips(tips_df, run_date)

pandemic_recovery_df = business_df.join(
                            checkin_df, on="business_id", how='left').fillna(0)
pandemic_recovery_df = pandemic_recovery_df.join(
                            reviews_df, on="business_id",how='left').fillna(0)
```

```
assert_inglewood_pizza_has_correct_review_count(reviews_df, 1)
```

Squeeze the Bottom

```
reviews_df = count_reviews(checkin_df, mobile_reviews_df, browser_reviews_df, run_date)
checkin_df = count_checkins(checkin_df, run_date)
tips_df = count_tips(tips_df, run_date)

pandemic_recovery_df = business_df.join(
                            checkin_df, on="business_id", how='left').fillna(0)
```

```
assert_inglewood_pizza_has_correct_review_count(reviews_df, 1)
```

```
pandemic_recovery_df = pandemic_recovery_df.join(
                            reviews_df, on="business_id",how='left').fillna(0)
```

Squeeze the Bottom

```
reviews_df = count_reviews(checkin_df, mobile_reviews_df, browser_reviews_df, run_date)
checkin_df = count_checkins(checkin_df, run_date)
tips_df = count_tips(tips_df, run_date)

pandemic_recovery_df = business_df.join(
                            checkin_df, on="business_id", how='left').fillna(0)
```

```
assert_inglewood_pizza_has_correct_review_count(reviews_df, 1)
```

Squeeze the Bottom

```
checkin_df = create_checkin_df_with_one_date_per_row(checkin_df)

reviews_df = count_reviews(checkin_df, mobile_reviews_df, browser_reviews_df, run_date)
checkin_df = count_checkins(checkin_df, run_date)
tips_df = count_tips(tips_df, run_date)
```

```
assert_inglewood_pizza_has_correct_review_count(reviews_df, 1)
```

```
pandemic_recovery_df = business_df.join(
                            checkin_df, on="business_id", how='left').fillna(0)
```

Squeeze the Bottom

```
checkin_df = create_checkin_df_with_one_date_per_row(checkin_df)

reviews_df = count_reviews(checkin_df, mobile_reviews_df, browser_reviews_df, run_date)
checkin_df = count_checkins(checkin_df, run_date)
tips_df = count_tips(tips_df, run_date)
```

```
assert_inglewood_pizza_has_correct_review_count(reviews_df, 1)
```

Squeeze the Bottom

```
checkin_df = create_checkin_df_with_one_date_per_row(checkin_df)

reviews_df = count_reviews(checkin_df, mobile_reviews_df, browser_reviews_df, run_date)
checkin_df = count_checkins(checkin_df, run_date)
```

```
assert_inglewood_pizza_has_correct_review_count(reviews_df, 1)
```

```
tips_df = count_tips(tips_df, run_date)
```

Squeeze the Bottom

```
checkin_df = create_checkin_df_with_one_date_per_row(checkin_df)

reviews_df = count_reviews(checkin_df, mobile_reviews_df, browser_reviews_df, run_date)
checkin_df = count_checkins(checkin_df, run_date)
```

```
assert_inglewood_pizza_has_correct_review_count(reviews_df, 1)
```

Squeeze the Bottom

```
checkin_df = create_checkin_df_with_one_date_per_row(checkin_df)

reviews_df = count_reviews(checkin_df, mobile_reviews_df, browser_reviews_df, run_date)

assert_inglewood_pizza_has_correct_review_count(reviews_df, 1)

checkin_df = count_checkins(checkin_df, run_date)
```

Squeeze the Bottom

```
checkin_df = create_checkin_df_with_one_date_per_row(checkin_df)

reviews_df = count_reviews(checkin_df, mobile_reviews_df, browser_reviews_df, run_date)
```

```
assert_inglewood_pizza_has_correct_review_count(reviews_df, 1)
```

Squeeze the Bottom

```
checkin_df = create_checkin_df_with_one_date_per_row(checkin_df)

assert_inglewood_pizza_has_correct_review_count(reviews_df, 1)

reviews_df = count_reviews(checkin_df, mobile_reviews_df, browser_reviews_df, run_date)
```

Squeeze the Bottom

```
checkin_df = create_checkin_df_with_one_date_per_row(checkin_df)

reviews_df = count_reviews(checkin_df, mobile_reviews_df, browser_reviews_df, run_date)
```

```
assert_inglewood_pizza_has_correct_review_count(reviews_df, 1)
```

Squeeze the Bottom

```
checkin_df = create_checkin_df_with_one_date_per_row(checkin_df)

reviews_df = count_reviews(checkin_df, mobile_reviews_df, browser_reviews_df, run_date)

assert_inglewood_pizza_has_correct_review_count(reviews_df, 1)

def count_reviews(checkin_df, mobile_reviews_df, browser_reviews_df, run_date):
  mobile_reviews_df = (
    mobile_reviews_df
    .join(
      checkin_df,
      on=['business_id','date', 'user_id'],
      how="left_anti")
    .select(
       *mobile_reviews_df.columns)
  )
  reviews_df = mobile_reviews_df.union(browser_reviews_df)
  reviews_df = reviews_df.filter(reviews_df.date == run_date.date())
  reviews_df = reviews_df.groupby("business_id").count()
  reviews_df = reviews_df.withColumnRenamed("count", "num_reviews")
  return reviews_df
```

Squeeze the Bottom

```
def count_reviews(checkin_df, mobile_reviews_df, browser_reviews_df, run_date):
  mobile_reviews_df = (
    mobile_reviews_df
    .join(
      checkin_df,
      on=['business_id','date', 'user_id’],
      how="left_anti”)
    .select(
       *mobile_reviews_df.columns)
  )
  reviews_df = mobile_reviews_df.union(browser_reviews_df)
  reviews_df = reviews_df.filter(reviews_df.date == run_date.date())
  reviews_df = reviews_df.groupby("business_id").count()
  reviews_df = reviews_df.withColumnRenamed("count", "num_reviews")
  return reviews_df
```

Squeeze the Bottom

```
def count_reviews(checkin_df, mobile_reviews_df, browser_reviews_df, run_date):
  mobile_reviews_df = (
    mobile_reviews_df
    .join(
      checkin_df,
      on=['business_id','date', 'user_id’],
      how="left_anti”)
    .select(
       *mobile_reviews_df.columns)
  )
  reviews_df = mobile_reviews_df.union(browser_reviews_df)
  reviews_df = reviews_df.filter(reviews_df.date == run_date.date())
  reviews_df = reviews_df.groupby("business_id").count()
  reviews_df = reviews_df.withColumnRenamed("count", "num_reviews")
  return reviews_df
```

Squeeze the Bottom

```
def test_mobile_reviews_count_without_checkins(spark: SparkSession) -> None:
  browser_reviews_df   = create_df_from_json("fixtures/browser_reviews.json", spark)
  checkin_df     = create_df_from_json("fixtures/checkin.json", spark)
  tips_df        = create_df_from_json("fixtures/tips.json", spark)
  business_df    = create_df_from_json("fixtures/business.json", spark)
  mobile_reviews_df   = create_df_from_json("fixtures/mobile_reviews.json", spark)


  checkin_df = create_checkin_df_with_one_date_per_row(checkin_df)

  reviews_df = count_reviews(checkin_df, mobile_reviews_df, browser_reviews_df, run_date)

  assert_inglewood_pizza_has_correct_review_count(reviews_df, 1)
```

Squeeze the Top

```
def test_mobile_reviews_count_without_checkins(spark: SparkSession) -> None:
  browser_reviews_df   = create_df_from_json("fixtures/browser_reviews.json", spark)
  checkin_df     = create_df_from_json("fixtures/checkin.json", spark)
  tips_df        = create_df_from_json("fixtures/tips.json", spark)
  business_df    = create_df_from_json("fixtures/business.json", spark)
  mobile_reviews_df   = create_df_from_json("fixtures/mobile_reviews.json", spark)


  checkin_df = create_checkin_df_with_one_date_per_row(checkin_df)
```

```
  reviews_df = count_reviews(checkin_df, mobile_reviews_df, browser_reviews_df, run_date)

  assert_inglewood_pizza_has_correct_review_count(reviews_df, 1)
```

~~Squeeze~~ Simplify the Top

```
def test_mobile_reviews_count_without_checkins(spark: SparkSession) -> None:
  browser_reviews_df   = create_df_from_json("fixtures/browser_reviews.json", spark)
  checkin_df     = create_df_from_json("fixtures/checkin.json", spark)
  tips_df        = create_df_from_json("fixtures/tips.json", spark)
  business_df    = create_df_from_json("fixtures/business.json", spark)
  mobile_reviews_df   = create_df_from_json("fixtures/mobile_reviews.json", spark)

  checkin_df = create_checkin_df_with_one_date_per_row(checkin_df)

  reviews_df = count_reviews(checkin_df, mobile_reviews_df, browser_reviews_df, run_date)

  assert_inglewood_pizza_has_correct_review_count(reviews_df, 1)
```

Simplify the Top

```
def test_mobile_reviews_count_without_checkins(spark: SparkSession) -> None:
  browser_reviews_df   = create_df_from_json("fixtures/browser_reviews.json", spark)
  checkin_df     = create_df_from_json("fixtures/checkin.json", spark)
  tips_df        = create_df_from_json("fixtures/tips.json", spark)
  business_df    = create_df_from_json("fixtures/business.json", spark)
  mobile_reviews_df   = create_df_from_json("fixtures/mobile_reviews.json", spark)

  checkin_df = create_checkin_df_with_one_date_per_row(checkin_df)

  reviews_df = count_reviews(checkin_df, mobile_reviews_df, browser_reviews_df, run_date)

  assert_inglewood_pizza_has_correct_review_count(reviews_df, 1)
```

Simplify the Top

```
def test_mobile_reviews_count_without_checkins(spark: SparkSession) -> None:
    browser_reviews_df   = create_df_from_json("fixtures/browser_reviews.json", spark)
    checkin_df           = create_df_from_json("fixtures/checkin.json", spark)
    tips_df              = create_df_from_json("fixtures/tips.json", spark)
    business_df          = create_df_from_json("fixtures/business.json", spark)
    mobile_reviews_df    = create_df_from_json("fixtures/mobile_reviews.json", spark)

    checkin_df = create_checkin_df_with_one_date_per_row(checkin_df)

    reviews_df = count_reviews(checkin_df, mobile_reviews_df, browser_reviews_df, run_date)

    assert_inglewood_pizza_has_correct_review_count(reviews_df, 1)
```

Simplify the Top

```
def test_mobile_reviews_count_without_checkins(spark: SparkSession) -> None:
  browser_reviews_df   = create_df_from_json("fixtures/browser_reviews.json", spark)
  checkin_df     = create_df_from_json("fixtures/checkin.json", spark)


  mobile_reviews_df   = create_df_from_json("fixtures/mobile_reviews.json", spark)


  checkin_df = create_checkin_df_with_one_date_per_row(checkin_df)

  reviews_df = count_reviews(checkin_df, mobile_reviews_df, browser_reviews_df, run_date)

  assert_inglewood_pizza_has_correct_review_count(reviews_df, 1)
```

Simplify the Top

```
def test_mobile_reviews_count_without_checkins(spark: SparkSession) -> None:
  browser_reviews_df   = create_df_from_json("fixtures/browser_reviews.json", spark)
  checkin_df     = create_df_from_json("fixtures/checkin.json", spark)
  mobile_reviews_df   = create_df_from_json("fixtures/mobile_reviews.json", spark)

  checkin_df = create_checkin_df_with_one_date_per_row(checkin_df)

  reviews_df = count_reviews(checkin_df, mobile_reviews_df, browser_reviews_df, run_date)

  assert_inglewood_pizza_has_correct_review_count(reviews_df, 1)
```

This code is part of our system under test

```
def test_mobile_reviews_count_without_checkins(spark: SparkSession) -> None:
  browser_reviews_df   = create_df_from_json("fixtures/browser_reviews.json", spark)
  checkin_df     = create_df_from_json("fixtures/checkin.json", spark)
  mobile_reviews_df   = create_df_from_json("fixtures/mobile_reviews.json", spark)

  checkin_df = create_checkin_df_with_one_date_per_row(checkin_df)

  reviews_df = count_reviews(checkin_df, mobile_reviews_df, browser_reviews_df, run_date)

  assert_inglewood_pizza_has_correct_review_count(reviews_df, 1)
```

This is what we are testing

Simplify the Top

```
def test_mobile_reviews_count_without_checkins(spark: SparkSession) -> None:
  browser_reviews_df   = create_df_from_json("fixtures/browser_reviews.json", spark)
  checkin_df     = create_df_from_json("fixtures/checkin.json", spark)
  mobile_reviews_df   = create_df_from_json("fixtures/mobile_reviews.json", spark)

  checkin_df = create_checkin_df_with_one_date_per_row(checkin_df)

  reviews_df = count_reviews(checkin_df, mobile_reviews_df, browser_reviews_df, run_date)

  assert_inglewood_pizza_has_correct_review_count(reviews_df, 1)
```

Simplify the Top

```
def test_mobile_reviews_count_without_checkins(spark: SparkSession) -> None:
  browser_reviews_df   = create_df_from_json("fixtures/browser_reviews.json", spark)
  checkin_df     = create_df_from_json("fixtures/checkin.json", spark)
  mobile_reviews_df   = create_df_from_json("fixtures/mobile_reviews.json", spark)

  empty = [{
    "user_id": "",
    "business_id": "",
    "date": ""
  }]

  checkin_df = create_checkin_df_with_one_date_per_row(checkin_df)

  reviews_df = count_reviews(checkin_df, mobile_reviews_df, browser_reviews_df, run_date)

  assert_inglewood_pizza_has_correct_review_count(reviews_df, 1)
```

Simplify the Top

```
def test_mobile_reviews_count_without_checkins(spark: SparkSession) -> None:
  browser_reviews_df   = create_df_from_json("fixtures/browser_reviews.json", spark)
  checkin_df     = create_df_from_json("fixtures/checkin.json", spark)
  mobile_reviews_df   = create_df_from_json("fixtures/mobile_reviews.json", spark)

  empty = [{
    "user_id": "",
    "business_id": "",
    "date": ""
  }]

  checkin_df = spark.createDataFrame(pd.DataFrame(empty))

  reviews_df = count_reviews(checkin_df, mobile_reviews_df, browser_reviews_df, run_date)

  assert_inglewood_pizza_has_correct_review_count(reviews_df, 1)
```

Simplify the Top

```
def test_mobile_reviews_count_without_checkins(spark: SparkSession) -> None:
  browser_reviews_df   = create_df_from_json("fixtures/browser_reviews.json", spark)
  checkin_df     = create_df_from_json("fixtures/checkin.json", spark)
  mobile_reviews_df   = create_df_from_json("fixtures/mobile_reviews.json", spark)

  empty = [{
    "user_id": "",
    "business_id": "",
    "date": ""
  }]

  checkin_df = spark.createDataFrame(pd.DataFrame(empty))

  reviews_df = count_reviews(checkin_df, mobile_reviews_df, browser_reviews_df, run_date)

  assert_inglewood_pizza_has_correct_review_count(reviews_df, 1)
```

Simplify the Top

```
def test_mobile_reviews_count_without_checkins(spark: SparkSession) -> None:
  browser_reviews_df   = create_df_from_json("fixtures/browser_reviews.json", spark)
  checkin_df           = create_df_from_json("fixtures/checkin.json", spark)
  mobile_reviews_df    = create_df_from_json("fixtures/mobile_reviews.json", spark)

  empty = [{
    "user_id": "",
    "business_id": "",
    "date": ""
  }]

  checkin_df = spark.createDataFrame(pd.DataFrame(empty))

  reviews_df = count_reviews(checkin_df, mobile_reviews_df, browser_reviews_df, run_date)

  assert_inglewood_pizza_has_correct_review_count(reviews_df, 1)
```

Simplify the Top

```
def test_mobile_reviews_count_without_checkins(spark: SparkSession) -> None:
  browser_reviews_df   = create_df_from_json("fixtures/browser_reviews.json", spark)

  mobile_reviews_df   = create_df_from_json("fixtures/mobile_reviews.json", spark)

  empty = [{
    "user_id": "",
    "business_id": "",
    "date": ""
  }]

  checkin_df = spark.createDataFrame(pd.DataFrame(empty))

  reviews_df = count_reviews(checkin_df, mobile_reviews_df, browser_reviews_df, run_date)


  assert_inglewood_pizza_has_correct_review_count(reviews_df, 1)
```

Simplify the Top

```
def test_mobile_reviews_count_without_checkins(spark: SparkSession) -> None:
  browser_reviews_df   = create_df_from_json("fixtures/browser_reviews.json", spark)
  mobile_reviews_df    = create_df_from_json("fixtures/mobile_reviews.json", spark)


  empty = [{
    "user_id": "",
    "business_id": "",
    "date": ""
  }]

  checkin_df = spark.createDataFrame(pd.DataFrame(empty))

  reviews_df = count_reviews(checkin_df, mobile_reviews_df, browser_reviews_df, run_date)

  assert_inglewood_pizza_has_correct_review_count(reviews_df, 1)
```

Simplify the Top

```
def test_mobile_reviews_count_without_checkins(spark: SparkSession) -> None:
  mobile_reviews_df   = create_df_from_json("fixtures/mobile_reviews.json", spark)
  empty = [{
    "user_id": "",
    "business_id": "",
    "date": ""
  }]


  browser_reviews_df   = create_df_from_json("fixtures/browser_reviews.json", spark)
  checkin_df = spark.createDataFrame(pd.DataFrame(empty))

  reviews_df = count_reviews(checkin_df, mobile_reviews_df, browser_reviews_df, run_date)

  assert_inglewood_pizza_has_correct_review_count(reviews_df, 1)
```

Simplify the Top

```
def test_mobile_reviews_count_without_checkins(spark: SparkSession) -> None:
  mobile_reviews_df   = create_df_from_json("fixtures/mobile_reviews.json", spark)
  empty = [{
    "user_id": "",
    "business_id": "",
    "date": ""
  }]


  browser_reviews_df   = spark.createDataFrame(pd.DataFrame(empty))
  checkin_df = spark.createDataFrame(pd.DataFrame(empty))

  reviews_df = count_reviews(checkin_df, mobile_reviews_df, browser_reviews_df, run_date)

  assert_inglewood_pizza_has_correct_review_count(reviews_df, 1)
```

Simplify the Top

```
def test_mobile_reviews_count_without_checkins(spark: SparkSession) -> None:
  mobile_reviews_df   = create_df_from_json("fixtures/mobile_reviews.json", spark)
  empty = [{
    "user_id": "",
    "business_id": "",
    "date": ""
  }]


  browser_reviews_df   = spark.createDataFrame(pd.DataFrame(empty))
  checkin_df = spark.createDataFrame(pd.DataFrame(empty))

  reviews_df = count_reviews(checkin_df, mobile_reviews_df, browser_reviews_df, run_date)

  assert_inglewood_pizza_has_correct_review_count(reviews_df, 1)
```

Simplify the Top

```
def test_mobile_reviews_count_without_checkins(spark: SparkSession) -> None:
  review = [{
     "user_id": "0dea283273fa4f40ac12ea210994d5f8",
     "business_id": "mobile_review_no_checkins",
     "date": "2022-04-14 07:31:06"
  }]
  mobile_reviews_df    = create_df_from_json("fixtures/mobile_reviews.json", spark)


  empty = [{
    "user_id": "",
    "business_id": "",
    "date": ""
  }]
  browser_reviews_df    = spark.createDataFrame(pd.DataFrame(empty))
  checkin_df = spark.createDataFrame(pd.DataFrame(empty))

  reviews_df = count_reviews(checkin_df, mobile_reviews_df, browser_reviews_df, run_date)

  assert_inglewood_pizza_has_correct_review_count(reviews_df, 1)
```

Simplify the Top

```
def test_mobile_reviews_count_without_checkins(spark: SparkSession) -> None:
  review = [{
     "user_id": "0dea283273fa4f40ac12ea210994d5f8",
     "business_id": "mobile_review_no_checkins",
     "date": "2022-04-14 07:31:06"
  }]
  mobile_reviews_df    = spark.createDataFrame(pd.DataFrame(review))


  empty = [{
    "user_id": "",
    "business_id": "",
    "date": ""
  }]
  browser_reviews_df    = spark.createDataFrame(pd.DataFrame(empty))
  checkin_df = spark.createDataFrame(pd.DataFrame(empty))

  reviews_df = count_reviews(checkin_df, mobile_reviews_df, browser_reviews_df, run_date)

  assert_inglewood_pizza_has_correct_review_count(reviews_df, 1)
```

Simplify the Top

```
def test_mobile_reviews_count_without_checkins(spark: SparkSession) -> None:
  review = [{
     "user_id": "0dea283273fa4f40ac12ea210994d5f8",
     "business_id": "mobile_review_no_checkins",
     "date": "2022-04-14 07:31:06"
  }]
  mobile_reviews_df    = spark.createDataFrame(pd.DataFrame(review))


  empty = [{
    "user_id": "",
    "business_id": "",
    "date": ""
  }]
  browser_reviews_df    = spark.createDataFrame(pd.DataFrame(empty))
  checkin_df = spark.createDataFrame(pd.DataFrame(empty))

  reviews_df = count_reviews(checkin_df, mobile_reviews_df, browser_reviews_df, run_date)

  assert_inglewood_pizza_has_correct_review_count(reviews_df, 1)
```

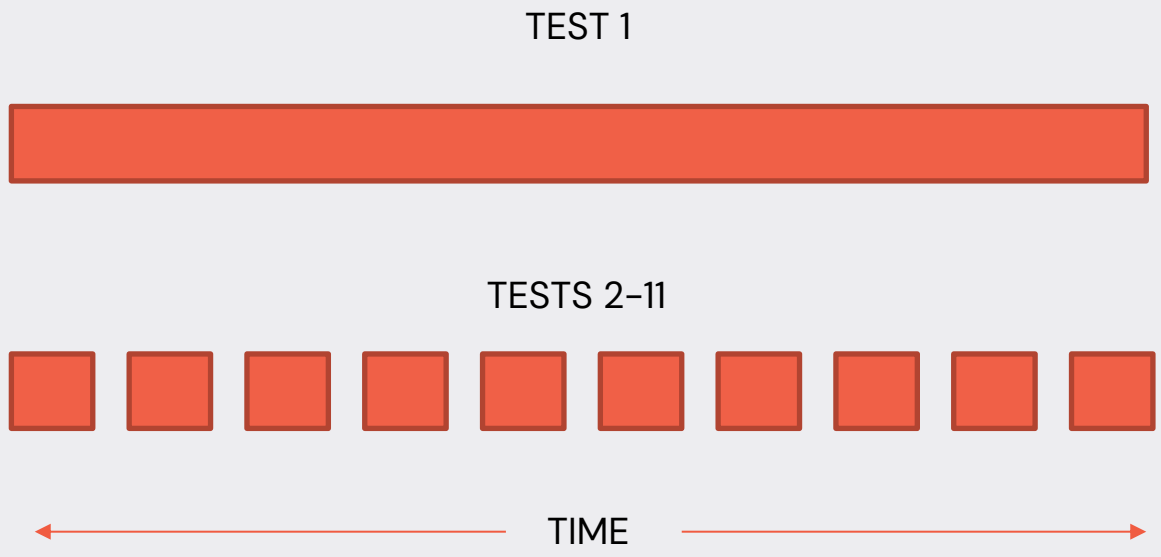
Is It Worth It?

Is It Worth It?

The

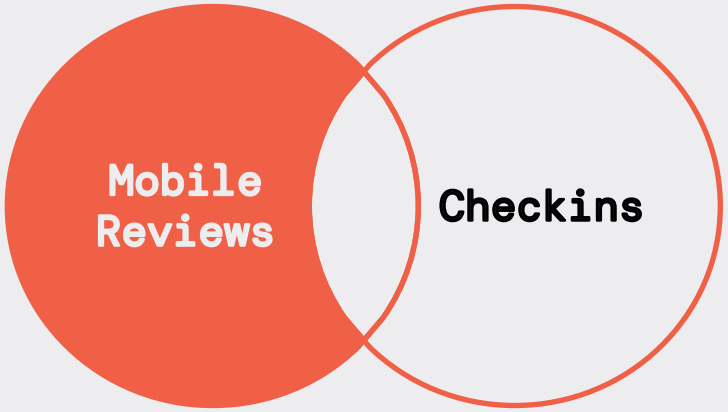
O to 1

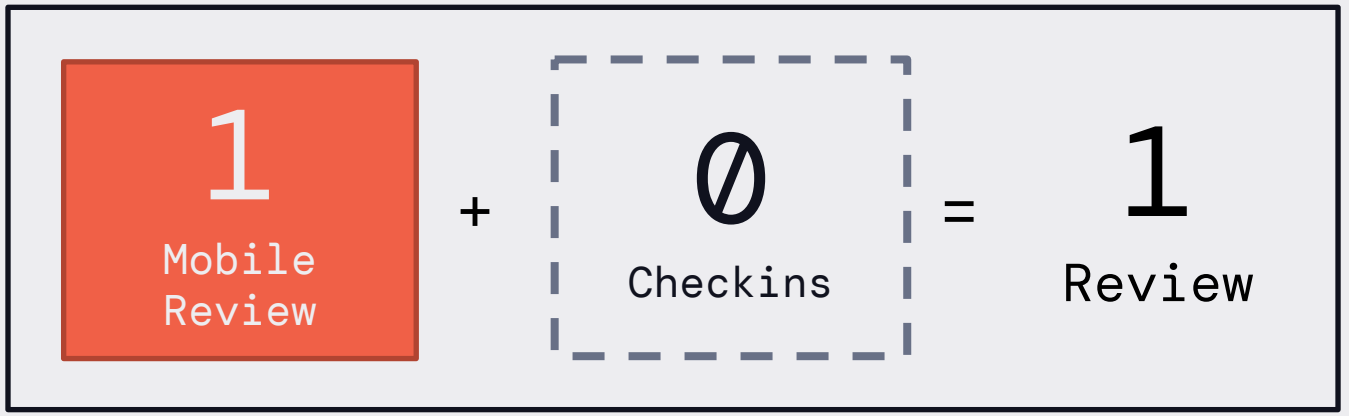
Phenomenon

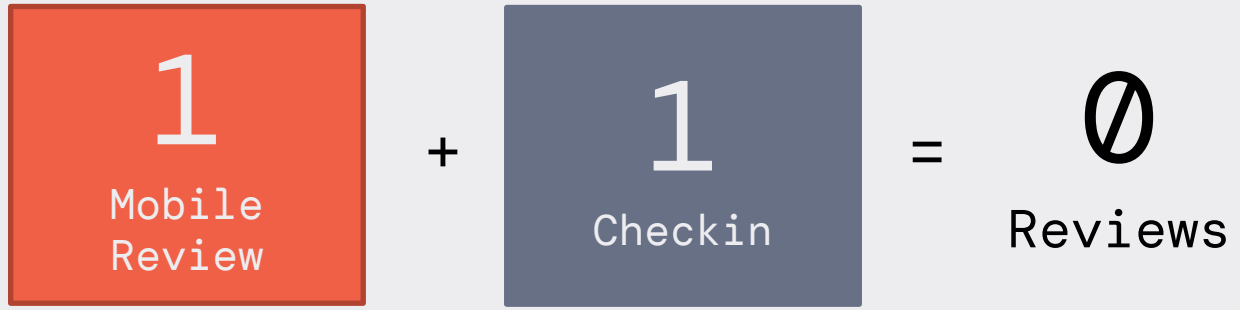


This is what we are testing

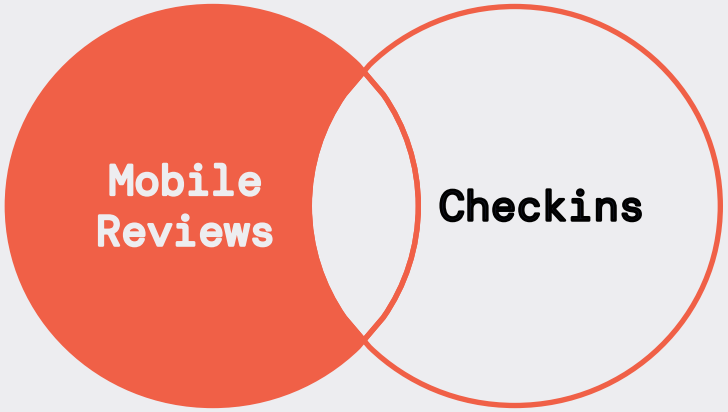
This is what we want

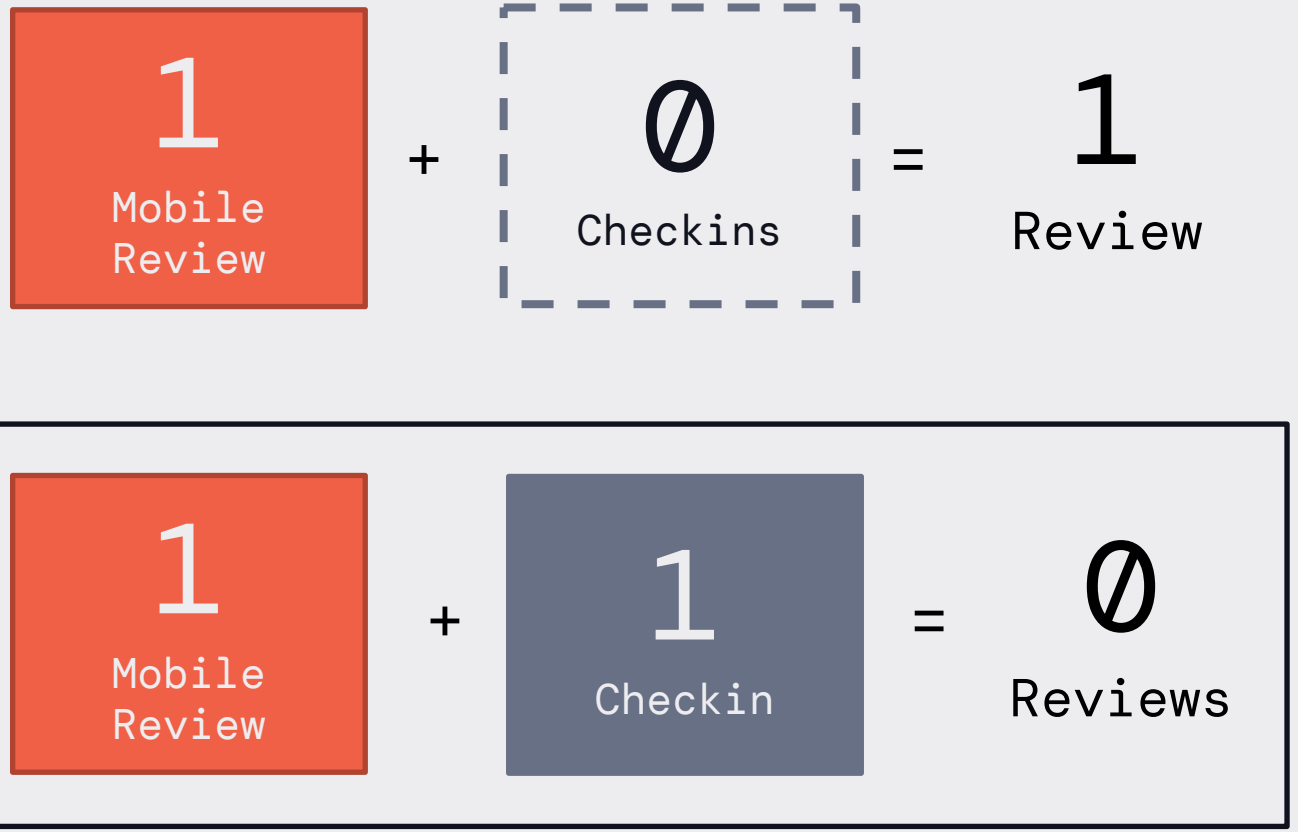






This is what we want





This is what we are testing

Duplicate the Test

```
def test_mobile_reviews_count_without_checkins(spark: SparkSession) -> None:
  review = [{
     "user_id": "0dea283273fa4f40ac12ea210994d5f8",
     "business_id": "mobile_review_no_checkins",
     "date": "2022-04-14 07:31:06"
  }]
  mobile_reviews_df    = spark.createDataFrame(pd.DataFrame(review))


  empty = [{
    "user_id": "",
    "business_id": "",
    "date": ""
  }]
  browser_reviews_df    = spark.createDataFrame(pd.DataFrame(empty))
  checkin_df = spark.createDataFrame(pd.DataFrame(empty))

  reviews_df = count_reviews(checkin_df, mobile_reviews_df, browser_reviews_df, run_date)

  assert_inglewood_pizza_has_correct_review_count(reviews_df, 1)
```

Make the Name Specific

```
def test_mobile_reviews_count_with_checkins(spark: SparkSession) -> None:
  review = [{
     "user_id": "0dea283273fa4f40ac12ea210994d5f8",
     "business_id": "mobile_review_no_checkins",
     "date": "2022-04-14 07:31:06"
  }]
  mobile_reviews_df    = spark.createDataFrame(pd.DataFrame(review))


  empty = [{
    "user_id": "",
    "business_id": "",
    "date": ""
  }]
  browser_reviews_df    = spark.createDataFrame(pd.DataFrame(empty))
  checkin_df = spark.createDataFrame(pd.DataFrame(empty))

  reviews_df = count_reviews(checkin_df, mobile_reviews_df, browser_reviews_df, run_date)

  assert_inglewood_pizza_has_correct_review_count(reviews_df, 1)
```

Make the assertion specific

```
def test_mobile_reviews_count_with_checkins(spark: SparkSession) -> None:
  review = [{
     "user_id": "0dea283273fa4f40ac12ea210994d5f8",
     "business_id": "mobile_review_no_checkins",
     "date": "2022-04-14 07:31:06"
  }]
  mobile_reviews_df    = spark.createDataFrame(pd.DataFrame(review))


  empty = [{
    "user_id": "",
    "business_id": "",
    "date": ""
  }]
  browser_reviews_df    = spark.createDataFrame(pd.DataFrame(empty))
  checkin_df = spark.createDataFrame(pd.DataFrame(empty))

  reviews_df = count_reviews(checkin_df, mobile_reviews_df, browser_reviews_df, run_date)

  assert_inglewood_pizza_has_correct_review_count(reviews_df, 1)
```

Make the assertion specific

```
def test_mobile_reviews_count_with_checkins(spark: SparkSession) -> None:
  review = [{
     "user_id": "0dea283273fa4f40ac12ea210994d5f8",
     "business_id": "mobile_review_no_checkins",
     "date": "2022-04-14 07:31:06"
  }]
  mobile_reviews_df    = spark.createDataFrame(pd.DataFrame(review))


  empty = [{
    "user_id": "",
    "business_id": "",
    "date": ""
  }]
  browser_reviews_df    = spark.createDataFrame(pd.DataFrame(empty))
  checkin_df = spark.createDataFrame(pd.DataFrame(empty))

  reviews_df = count_reviews(checkin_df, mobile_reviews_df, browser_reviews_df, run_date)

  assert_inglewood_pizza_has_correct_review_count(reviews_df, 1)
```

Make the assertion specific

```
def test_mobile_reviews_count_with_checkins(spark: SparkSession) -> None:
  review = [{
     "user_id": "0dea283273fa4f40ac12ea210994d5f8",
     "business_id": "mobile_review_no_checkins",
     "date": "2022-04-14 07:31:06"
  }]
  mobile_reviews_df    = spark.createDataFrame(pd.DataFrame(review))


  empty = [{
    "user_id": "",
    "business_id": "",
    "date": ""
  }]
  browser_reviews_df    = spark.createDataFrame(pd.DataFrame(empty))
  checkin_df = spark.createDataFrame(pd.DataFrame(empty))

  reviews_df = count_reviews(checkin_df, mobile_reviews_df, browser_reviews_df, run_date)

  assert_inglewood_pizza_has_correct_review_count(reviews_df, 0)
```

Rename the Matching Properties

```
def test_mobile_reviews_count_with_checkins(spark: SparkSession) -> None:
  review= [{
     "user_id": "0dea283273fa4f40ac12ea210994d5f8",
     "business_id": "mobile_review_no_checkins",
     "date": "2022-04-14 07:31:06"
  }]
  mobile_reviews_df    = spark.createDataFrame(pd.DataFrame(review))


  empty = [{
    "user_id": "",
    "business_id": "",
    "date": ""
  }]
  browser_reviews_df    = spark.createDataFrame(pd.DataFrame(empty))
  checkin_df = spark.createDataFrame(pd.DataFrame(empty))

  reviews_df = count_reviews(checkin_df, mobile_reviews_df, browser_reviews_df, run_date)

  assert_inglewood_pizza_has_correct_review_count(reviews_df, 0)
```

Rename the Matching Properties

```
def test_mobile_reviews_count_with_checkins(spark: SparkSession) -> None:
  matching_on = [{
     "user_id": "0dea283273fa4f40ac12ea210994d5f8",
     "business_id": "mobile_review_no_checkins",
     "date": "2022-04-14 07:31:06"
  }]
  mobile_reviews_df    = spark.createDataFrame(pd.DataFrame(matching_on))


  empty = [{
    "user_id": "",
    "business_id": "",
    "date": ""
  }]
  browser_reviews_df    = spark.createDataFrame(pd.DataFrame(empty))
  checkin_df = spark.createDataFrame(pd.DataFrame(empty))

  reviews_df = count_reviews(checkin_df, mobile_reviews_df, browser_reviews_df, run_date)

  assert_inglewood_pizza_has_correct_review_count(reviews_df, 0)
```

Rename the Matching Properties

```
def test_mobile_reviews_count_with_checkins(spark: SparkSession) -> None:
  matching_on = [{
     "user_id": "0dea283273fa4f40ac12ea210994d5f8",
     "business_id": "mobile_review_no_checkins",
     "date": "2022-04-14 07:31:06"
  }]
  mobile_reviews_df    = spark.createDataFrame(pd.DataFrame(matching_on))


  empty = [{
    "user_id": "",
    "business_id": "",
    "date": ""
  }]
  browser_reviews_df    = spark.createDataFrame(pd.DataFrame(empty))
  checkin_df = spark.createDataFrame(pd.DataFrame(empty))

  reviews_df = count_reviews(checkin_df, mobile_reviews_df, browser_reviews_df, run_date)

  assert_inglewood_pizza_has_correct_review_count(reviews_df, 0)
```

Rename the Matching Properties

```
def test_mobile_reviews_count_with_checkins(spark: SparkSession) -> None:
  matching_on = [{
     "user_id": "0dea283273fa4f40ac12ea210994d5f8",
     "business_id": "mobile_review_no_checkins",
     "date": "2022-04-14 07:31:06"
  }]
  mobile_reviews_df   = spark.createDataFrame(pd.DataFrame(matching_on))
  checkin_df = spark.createDataFrame(pd.DataFrame(empty))


  empty = [{
    "user_id": "",
    "business_id": "",
    "date": ""
  }]
  browser_reviews_df   = spark.createDataFrame(pd.DataFrame(empty))
  reviews_df = count_reviews(checkin_df, mobile_reviews_df, browser_reviews_df, run_date)

  assert_inglewood_pizza_has_correct_review_count(reviews_df, 0)
```

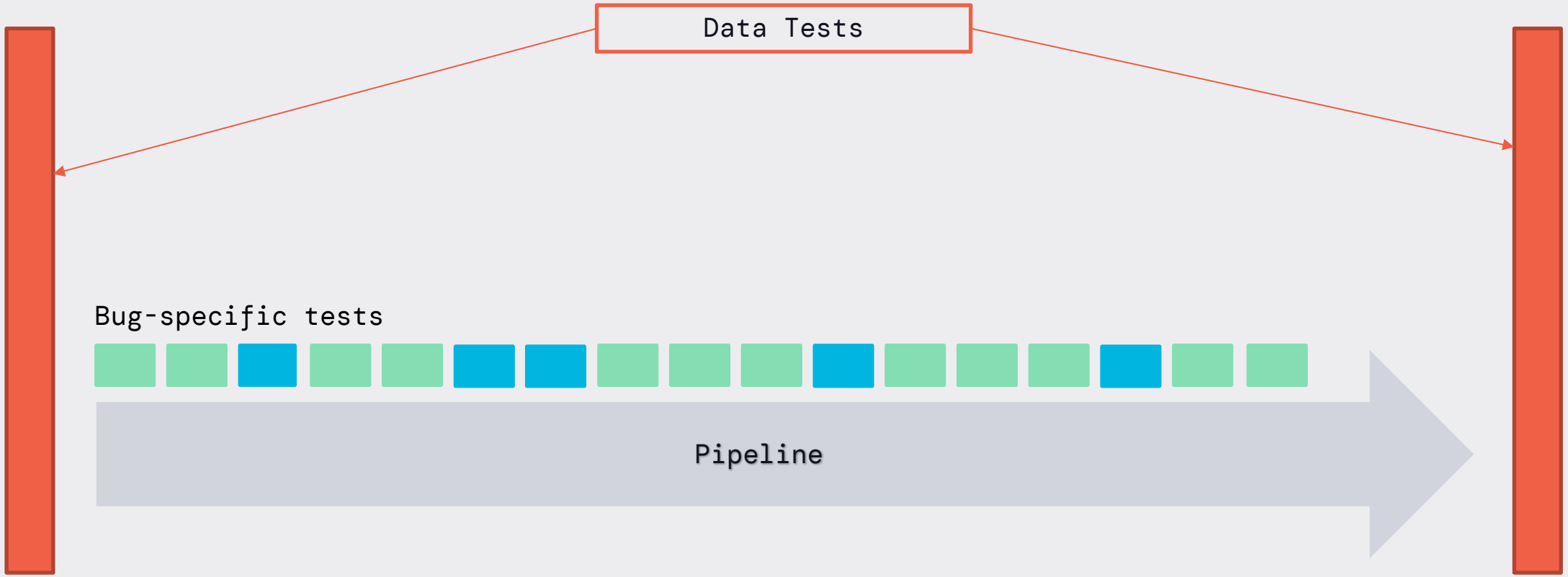
Rename the Matching Properties

```
def test_mobile_reviews_count_with_checkins(spark: SparkSession) -> None:
  matching_on = [{
     "user_id": "0dea283273fa4f40ac12ea210994d5f8",
     "business_id": "mobile_review_no_checkins",
     "date": "2022-04-14 07:31:06"
  }]
  mobile_reviews_df   = spark.createDataFrame(pd.DataFrame(matching_on))
  checkin_df = spark.createDataFrame(pd.DataFrame(matching_on))


  empty = [{
    "user_id": "",
    "business_id": "",
    "date": ""
  }]
  browser_reviews_df   = spark.createDataFrame(pd.DataFrame(empty))
  reviews_df = count_reviews(checkin_df, mobile_reviews_df, browser_reviews_df, run_date)

  assert_inglewood_pizza_has_correct_review_count(reviews_df, 0)
```

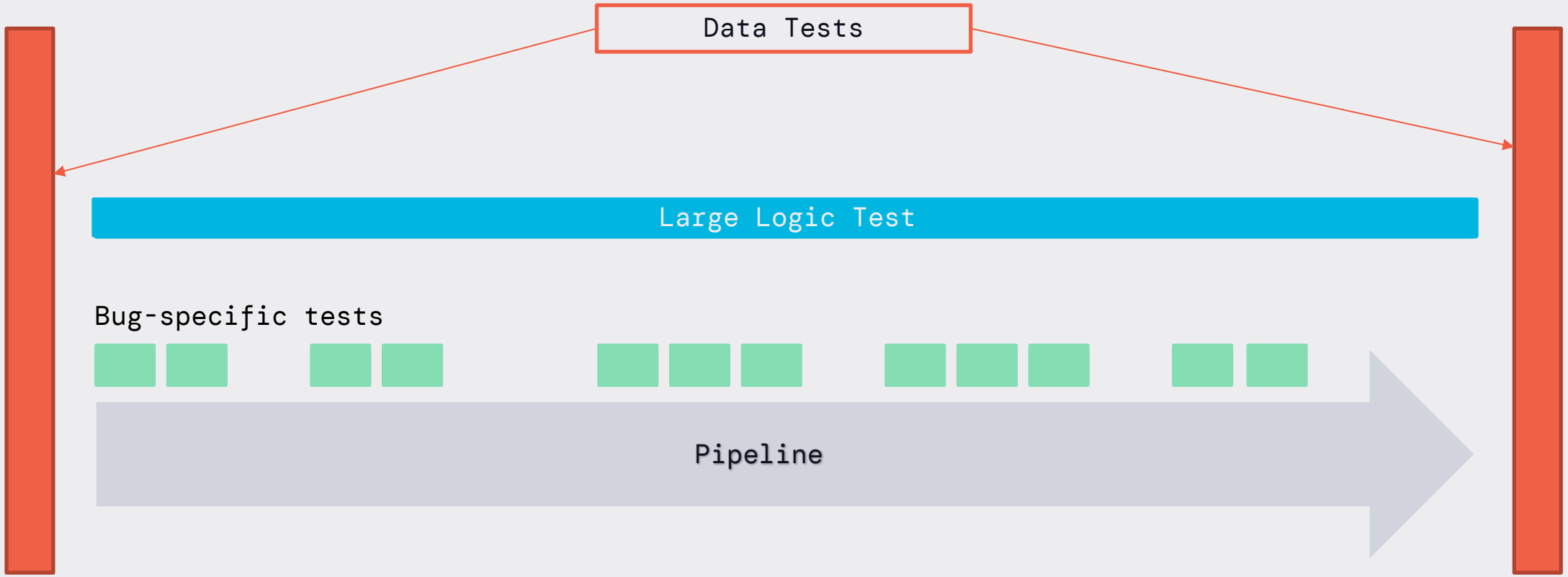
Rename the Matching Properties

```
def test_mobile_reviews_count_with_checkins(spark: SparkSession) -> None:
  matching_on = [{
     "user_id": "0dea283273fa4f40ac12ea210994d5f8",
     "business_id": "mobile_review_no_checkins",
     "date": "2022-04-14 07:31:06"
  }]
  mobile_reviews_df   = spark.createDataFrame(pd.DataFrame(matching_on))
  checkin_df = spark.createDataFrame(pd.DataFrame(matching_on))


  empty = [{
    "user_id": "",
    "business_id": "",
    "date": ""
  }]
  browser_reviews_df   = spark.createDataFrame(pd.DataFrame(empty))
  reviews_df = count_reviews(checkin_df, mobile_reviews_df, browser_reviews_df, run_date)

  assert_inglewood_pizza_has_correct_review_count(reviews_df, 0)
```

Multi-Layered Tests



Multi-Layered Tests



Resources

github.com/jmasonlee/Talks/blob/master/LearnToEfficientlyTestETLPipelines.md

DATA+AI SUMMIT 2022

Talk Resources:

Thank you

Jacqueline Bilston

Software Developer, Yelp

Llewellyn Falco
Rocky Rhodes
Tony Vo
Jonathan Rioux
Sam Livingston-Grey

Jay Bazuzi
Stephen Bilston
Hansel Qiu
Jaimie Russ
Lynn Langit