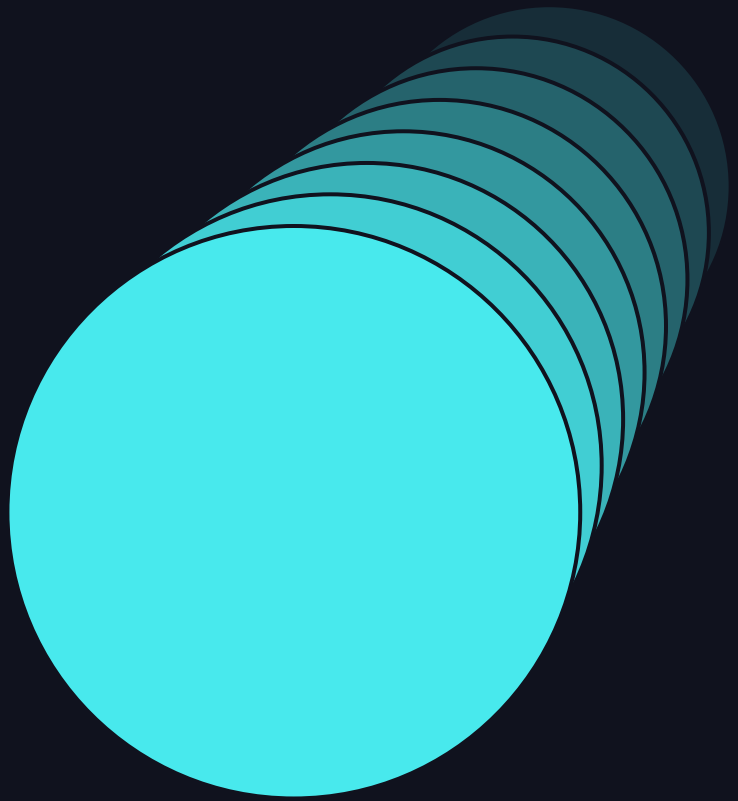
DATA+AI SUMMIT
BY databricks

Predictive ML: How Unilever is improving its forecasting



Bill Tsiligkiridis, Data Science Lead, Unilever Europe
Ozge Koroglu, Data Scientist
Gaius Noordhoek Hegt, Data Scientist
Patrick Van Dalen, ML Engineer

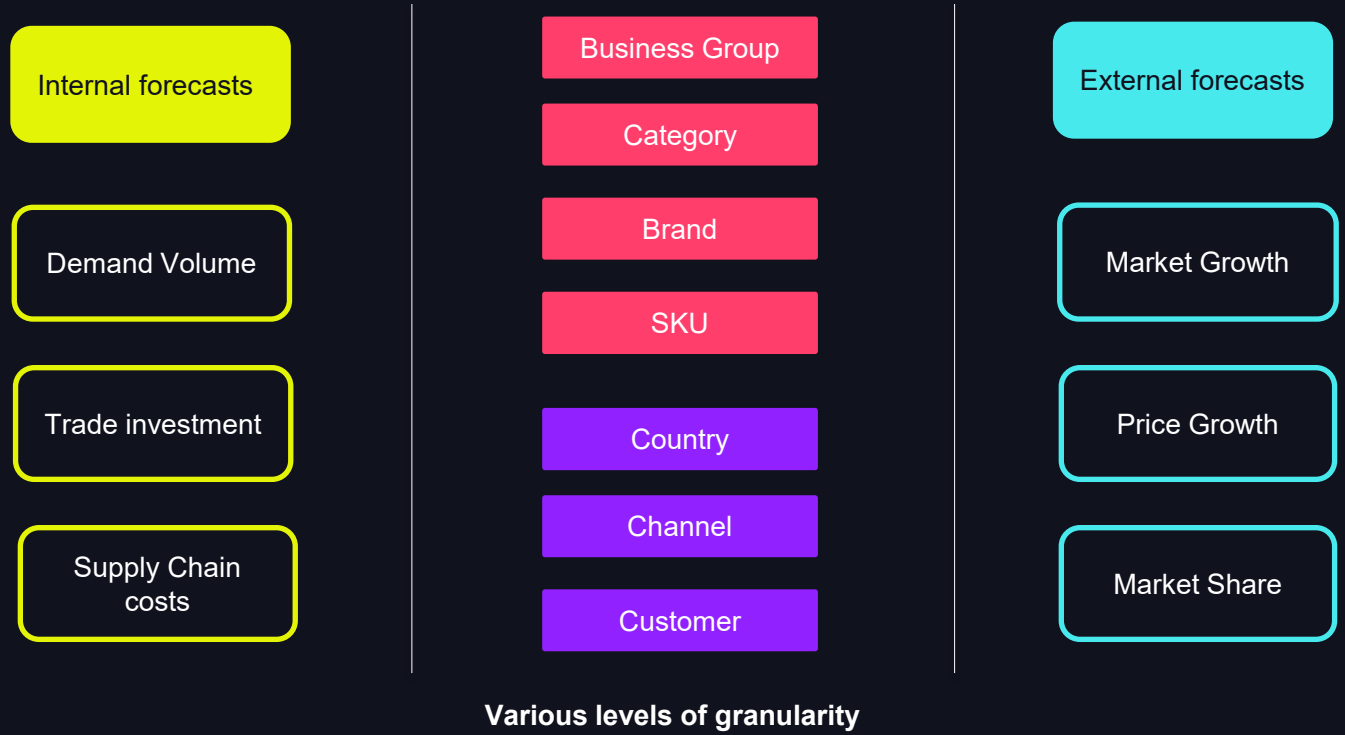
Agenda

- What we are trying to solve for
- Overview of solution
- Working towards sustainable maintainance
- Key Learnings
- Q&A

Forecasting

(It’s difficult business)

Examples of typical forecasts in CPG



The challenges of forecasting

Either approach will have its own set of challenges

Manual / Expert forecasts

- Very expensive to generate at scale
- Also expensive to update frequently
- Inconsistent set of assumptions
- Human Bias

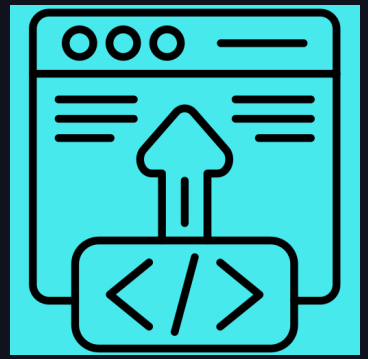
Automated / Machine Learning forecasts

- Historical data availability
- Forecast dependencies & compounding errors
- Solution maintainance
- Explainability

Key Technical Requirements

For a Data Science team developing ML forecasting solutions

Easy experimentation



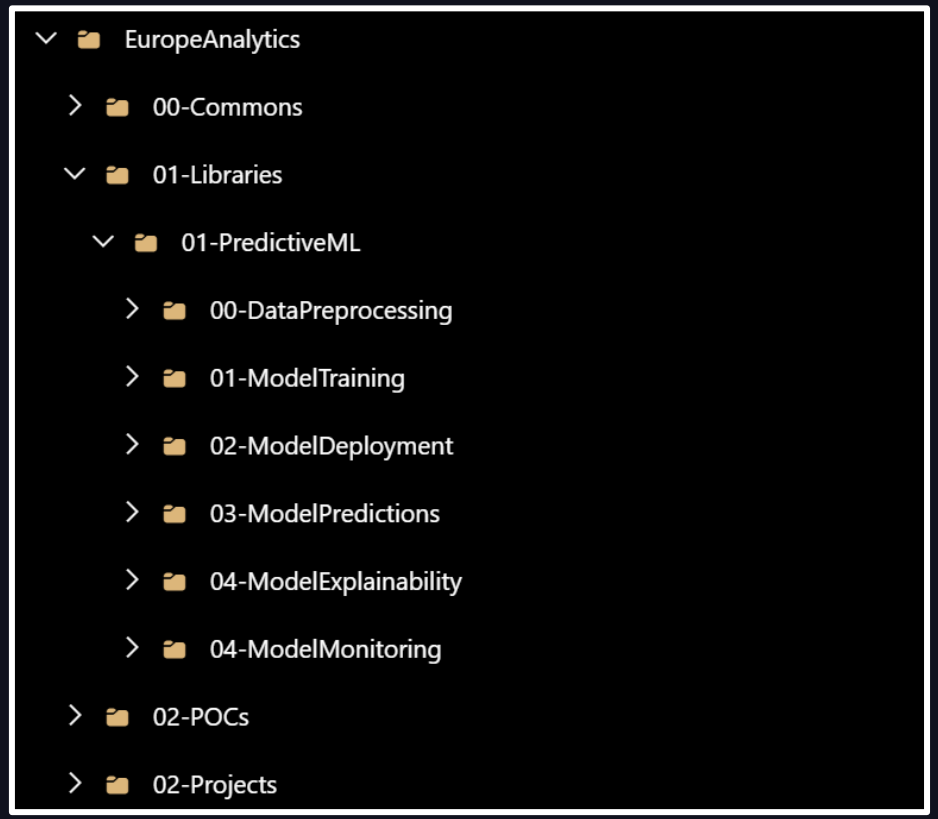
Deployment process

Sustainable maintenance

images: Flaticon.com

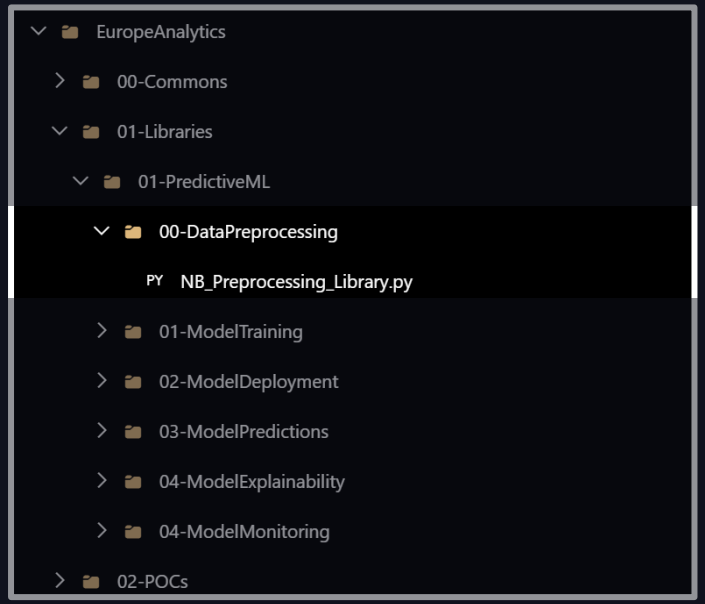
Developing a framework for predictive ML

Overview of framework



Data preprocessing

For Feature Engineering

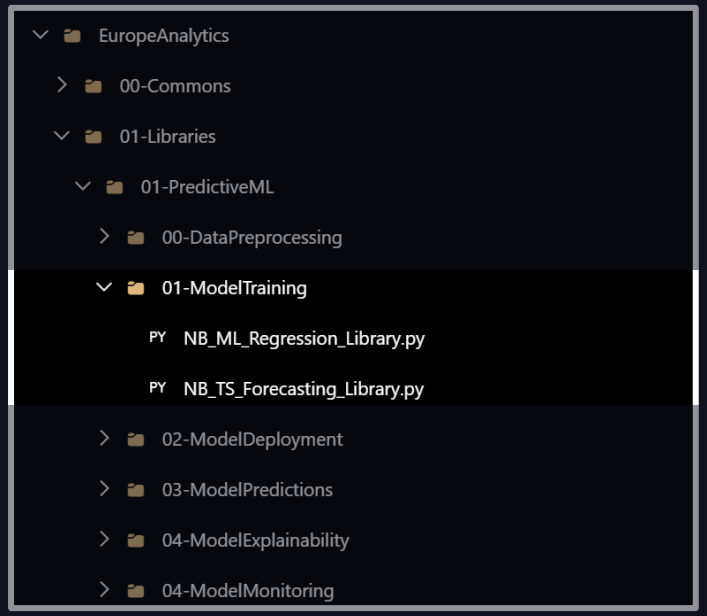



```
prep_obj = PreprocessingExecutor("value", h_period)

prep_obj.transform_imputer()
prep_obj.transform_OHE()
prep_obj.transform_standardScaler()
prep_obj.transform_powerTransform()
prep_obj.transform_SHAP(
    mapping_thresh=mapping_thresh,
    key=feature_mapping_category,
    path=feature_mapping_path,
)

df_xa = prep_obj.prep_x_ahead(df, col_list)
df_pass = prep_obj.execute_preprocess_pipeline(df_xa, "Retailer_Country_step")
pip, steps = prep_obj.getPiplineAndSteps()
```

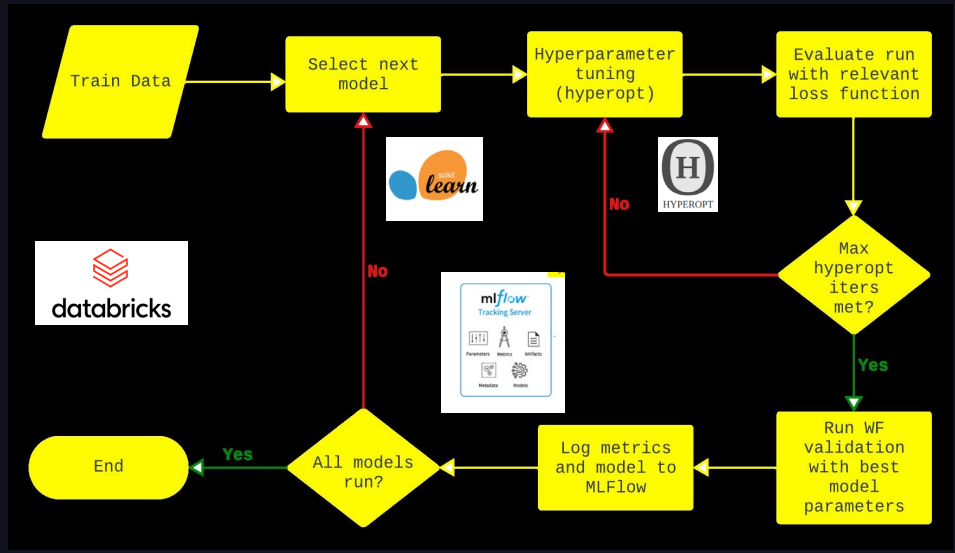
Transformation	Function Name	Parameters	Type
One Hot Encode	transform_OHE	-	Categorical
Simple Imputer	transform_imputer	-	Numeric
Standard Scaler	transform_standardScaler	-	Numeric
Power Transform	transform_powerTransform	-	Numeric
Select Columns	transform_selectCols	Cols ->List	Feature Select
Select K Best	transform_selectKBest	k->int	Feature Select
Recursive Feature Elimination	transform_RFE	n_feat->int	Feature Select
RFE Cross Validation	transform_RFECV	-	Feature Select
Correlation Filter	transform_correlationFilter	Thresh->float	Feature Select
Elastic Net Filter	transform_ENFilter	Thresh->float	Feature Select
PCA	transform_PCA	-	Dimension Reduction

Model Training

Time Series ML Regression Library

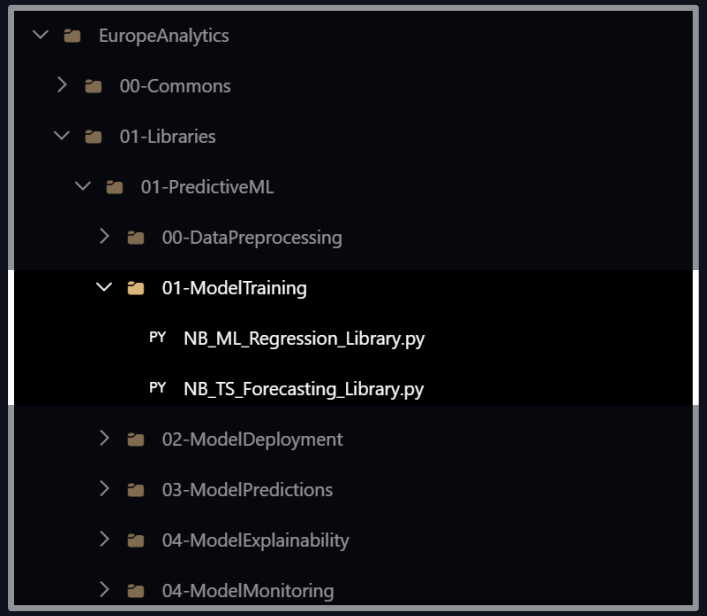



```
reg_obj = RegForecast("value", model_list, h_period, mlflow_exp, None, None, None)
reg_obj.setIDCol("Retailer_Country_step")
reg_obj.setModelCategory(category_cl)
reg_obj.setLossFunc(loss_func)
reg_obj.setNJobs(0)
reg_obj.setBootstrap(False)
reg_obj.setTransTarget(False)

df_out, pip_t = reg_obj.train_models(
    df_pass, pip, steps, project_name, artifact_path
)
```



Model Training

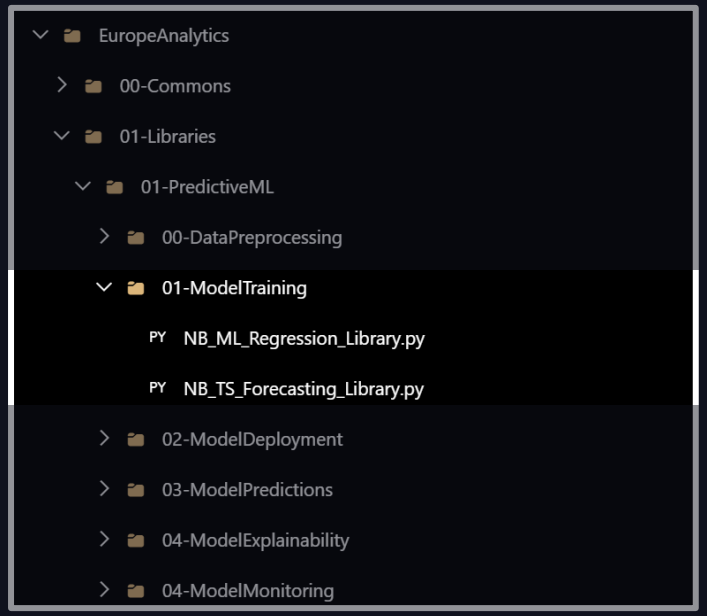
Time Series Forecasting Library



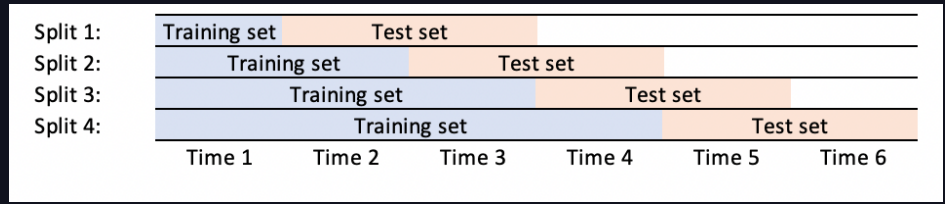

```
fc_util = ULForecastUtil(
    p_period,
    h_period,
    seasonal_period,
    correct_outliers,
    batch_size,
    hyperopt_search_list,
    project_name,
    exp_name,
    mlflow_exp,
    exp_training_file_path,
    model_list,
    region,
    run_type,
    model_type,
    box_cox,
)

print("Training data loaded")
if df_train.count() != 0:

    create_mlflow_path = mlflow_path(ctx, mlflow_exp)

    result = (
        df_train.select("category", "time", "value")
        .groupBy("category")
        .applyInPandas(execute_tune_train, schema=result_schema)
    ).cache()

    result, result_agg = fc_util.logResults(df_train, result)
    print("Logged results")
```

Model Training

Walk forward validation



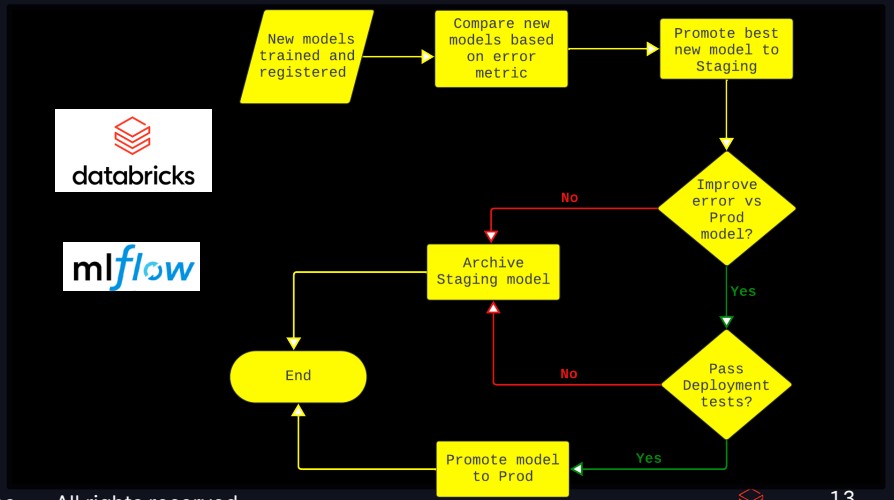

```python
def objective_wf(self, params):
    """
    Parameters
        ----------
        params : dict

    Returns
        ----------
        dict

    This is the main hyperopt objective function aiming at minimising the model metric
    """
    # start_time = time.perf_counter()
    model = params["model"]
    df = params["df"]
    del params["model"]
    del params["df"]
    print(params)
    pip = Pipeline(steps=[("regressor", model(**params))])
    if self.trans_target:
        pip = TransformedTargetRegressor(regressor=pip, transformer=self.trans_target)
    total_pred_df = self.walk_fw_valid(pip, df)
    total_pred_df, metric = self.calc_metric_wf(total_pred_df)
    if loss_func is not None:
        metric = loss_func(total_pred_df)

    return {"loss": metric, "status": STATUS_OK}
```



Model Deployment

Deploy through mlflow in Databricks

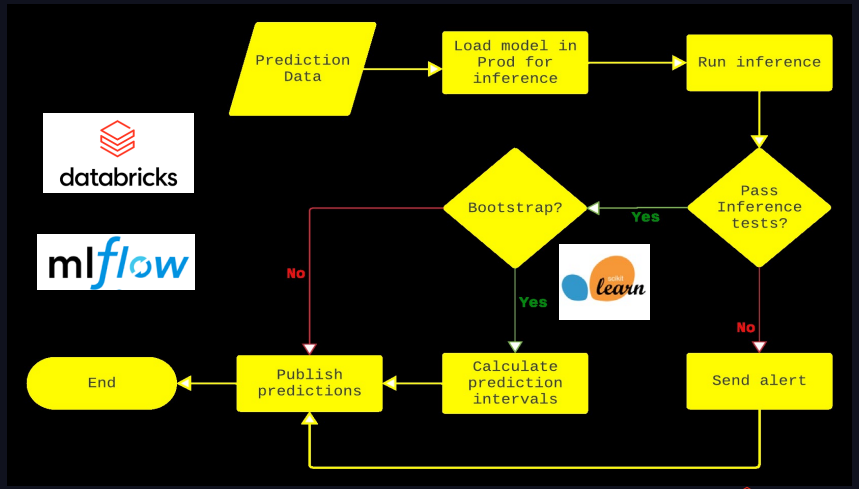

```
client = MlflowClient()

project_name_full = forecast_model.generateModelName()
df_results = load_models(project_name_full)
if df_results.filter("current_stage = 'None'").count() != 0:
    df_results_upd = drop_models(df_results, df_train)
    if df_results_upd.filter("current_stage = 'None'").count() !=
    0:
        df_ts_result = select_models(df_results_upd)
        df_all_models, df_models_promote, df_archive =
        deploy_models(df_ts_result, df_results, df_results_upd)
    else:
        archive_models(df_results.filter("current_stage = 'None'").
        toPandas())
```



Model Inference

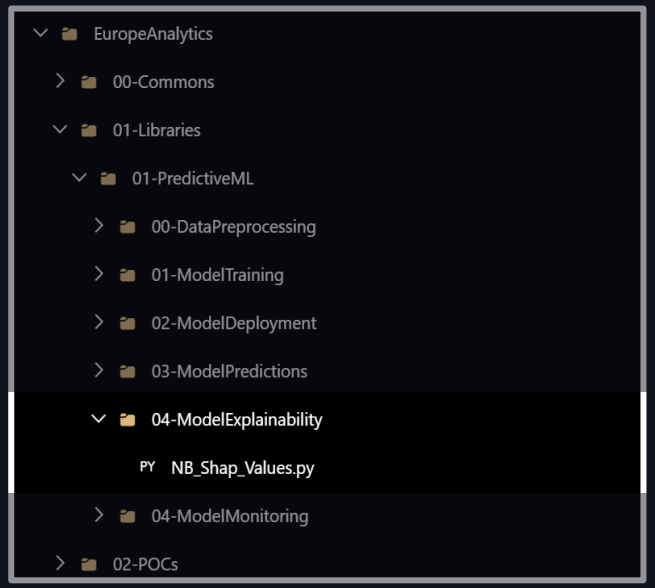
Load prod models and predict


```python
# Get predictions
def execute_fit_forecast(df):
    print("Starting execution.....")

    project_name_full = forecast_model.generateModelName()
    print(project_name_full)

    return forecast_model.fit_forecast(df, project_name_full)


result_schema = StructType(
    [
        StructField("category", StringType(), True),
        StructField("time", DateType(), True),
        StructField("value", DoubleType(), True),
        StructField("predicted", IntegerType(), True),
        StructField("lowervalue", DoubleType(), True),
        StructField("uppervalue", DoubleType(), True),
        StructField("processed_at", StringType(), True),
    ]
)

result = (
    df_train.select("category", "version", "Algorithm", "time", "value")
    .groupBy("category")
    .applyInPandas(execute_fit_forecast, schema=result_schema)
)

# Clean data
result = fc_util.clean_columns(result)
```



Model Explainability

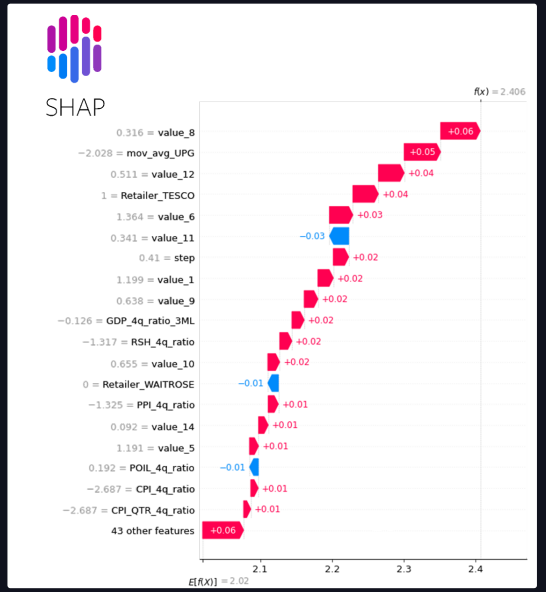
SHAP library



SHAPley additive explanations

- Model agnostic
- Local & Global interpretability

```
def model_transform_shap(X, cell):

    X = X.drop(columns=["Cell", "time"])

    # Load model - Structure FTM_'run_type'_'p_period'm Model_'Cell'
    cell_cl = "".join(e for e in cell if e.isalnum())
    model_name = project_name + "_{}".format(cell_cl)
    model_version_uri = "models:/{model_name}/{stage}".format(
        model_name=model_name, stage="Production"
    )
    loaded_model = mlflow_sklearn.load_model(model_version_uri)

    # Apply Pipeline transformations to X df
    unified_pipeline = loaded_model.named_steps["pre_pip"]
    X = unified_pipeline.transform(X)
    feature_names = X.columns.tolist()

    # SHAP
    explainer = shap.TreeExplainer(loaded_model.named_steps["model"])
    shap_values = explainer.shap_values(X)  # X_train or one instance from X_test

    return X, explainer, shap_values, feature_names
```

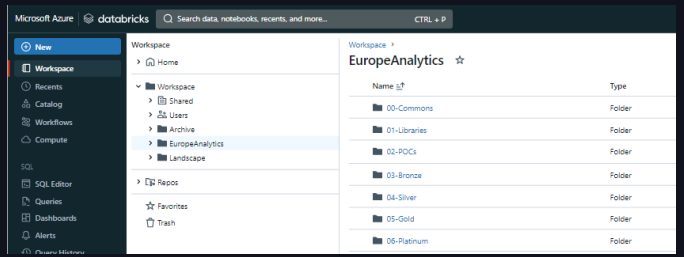


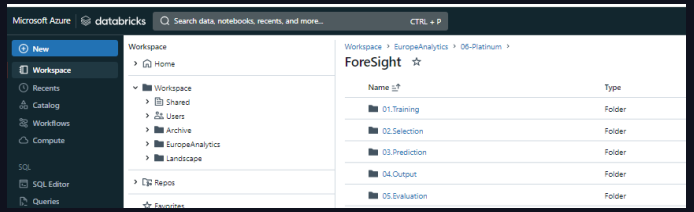
Working towards sustainable maintainance

Modularisation

Modularising code to enable easier testing, refactoring and boost reusability

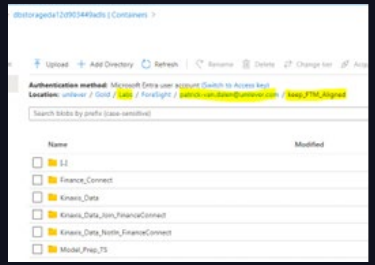
Organise in library functions & classes

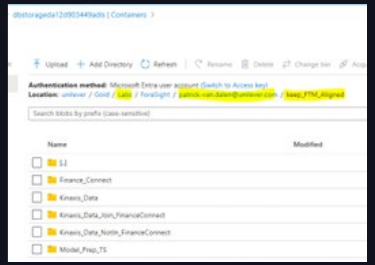




Labs for development

```
1  base_path = base_conf_path(ctx)

You're working in Branch-mode: /mnt/adls/Platinum/Labs/ForeSight/patrick-van.dalen@unilever.com/keep_FTM_Aligned
```





Parallelisation

Parallelising using Spark and applyInPandas

```
def execute_tune_train(df):
    print("Starting execution.....")
    forecast_model = ULForecast(
        p_period,
        h_period,
        seasonal_period,
        correct_outliers,
        batch_size,
        hyperopt_search_list,
        project_name,
        exp_name,
        mlflow_exp,
        exp_training_file_path,
        model_list,
        region,
        run_type,
        model_type,
        pickle_files,
        box_cox
    )
    forecast_model.setHolidayParams(holiday_year, holiday_country)
    print("Category .." + str(df.category.unique()))
    return forecast_model.train_tune_models(df)

print("Training data loaded")
if df_train.count() != 0:

    create_mlflow_path = mlflow_path(ctx, mlflow_exp)

    result = (
        df_train.select("category", "time", "value")
        .groupBy("category")
        .applyInPandas(execute_tune_train, schema=result_schema)
    ).cache()

    result, result_agg = fc_util.logResults(df_train, result)
    print("Logged results")
```

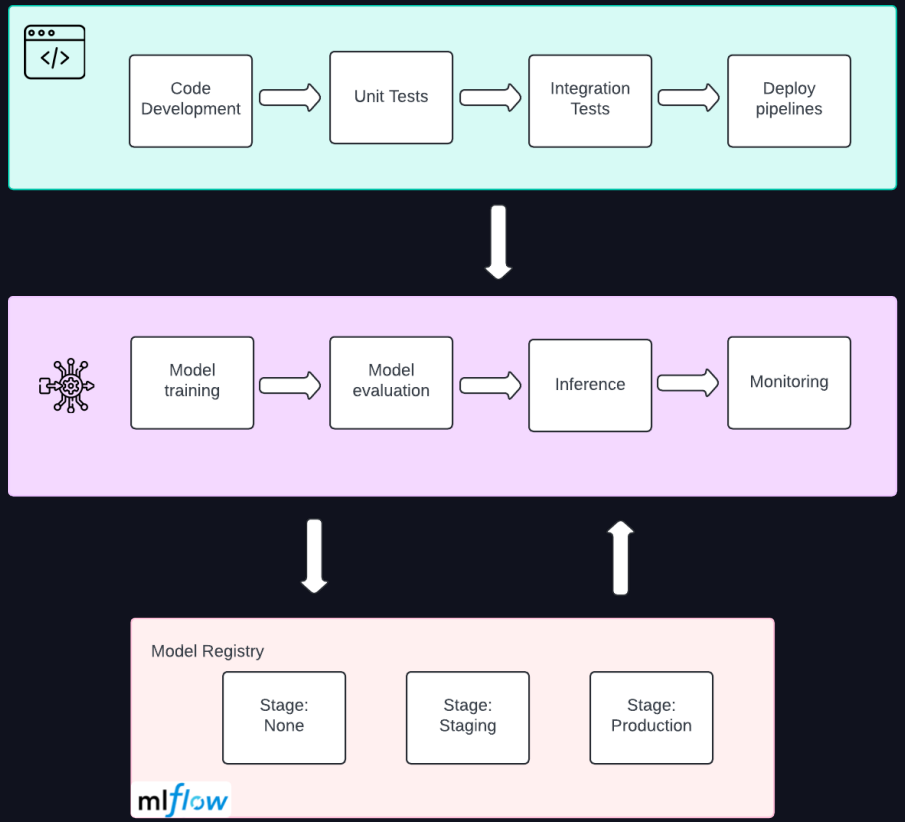
Parameterisation

Use of config files, widgets, metadata files

```
# Config paths
ctx = json.loads(dbutils.notebook.entry_point.getDbutils().notebook().getContext().toJson())
base_conf_path = base_conf_path_check(ctx)
mlflow_conf_path = mlflow_conf_path_check(ctx)

# Config paths
input_path = '/dbfs' + base_conf_path + "/input.ini"
output_path = '/dbfs' + base_conf_path + "/output.ini"
param_path = '/dbfs' + base_conf_path + "/param.ini"

total_region = 'europe'

root_path_gold = base_conf_path.replace("/platinum/", "/gold/") + "/" + total_region
root_path_platinum_training = base_conf_path + "/training" + "/" + total_region
root_path_platinum_selection = base_conf_path + "/selection" + "/" + total_region
root_path_platinum_predictions = base_conf_path + "/predictions" + "/" + total_region
root_path_platinum_output = base_conf_path + "/output" + "/" + total_region
root_path_platinum_evaluation = base_conf_path + "/evaluation" + "/" + total_region
root_path_mlflow = mlflow_conf_path
```

```
def clear_conf(confPath):
    path = confPath
    dbutils.fs.rm(path, False)

def clear_all_conf():
    path = base_conf_path
    dbutils.fs.rm(path + "/input.ini", False)
    dbutils.fs.rm(path + "/output.ini", False)
    dbutils.fs.rm(path + "/param.ini", False)

def load_conf(confPath):
    conf = configparser.ConfigParser()
    exists = os.path.isfile(confPath)
    if exists:
        fp = open(confPath, "r")
        conf.read_file(fp)
    else:
        print("No config file found - creating a new file")
    return conf

def write_conf(confPath):
    exists = os.path.isfile(confPath)
    if exists:
        conf.write(open(confPath, "w"))
    else:
        path = base_conf_path
        dbutils.fs.mkdirs(path)
```

Code Management & Testing

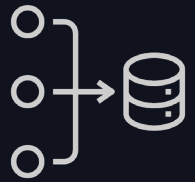


Code deployment

ML processes

Model deployment

ML Monitoring



Data Input

Quality

- Completeness
- Data Types
- Duplicates
- Outliers

Drift

- Change in distribution
- Change in trend
- Seasonal drift

Model Performance

Prediction Accuracy

- Accuracy of the live predictions
- Comparison with evaluation accuracy

Prediction Bias

- Systematic over or under predicting

Feature Importance

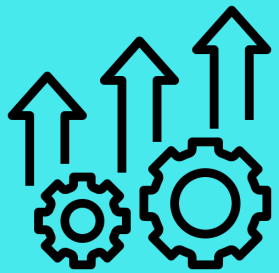
- Changes in feature importance as captured by Shap

Key Learnings

Key Business Learnings

Measure improvements vs a business process

10% reduction in MAPE, 100 hours in savings vs manual forecast



Work towards a transparent and explainable solution

Create trust and user confidence to increase adoption of solution

Invest more time in understanding business requirements

Work with a small stakeholder team to fully scope out project, PoC vs future releases

images: Flaticon.com

Key Technical Learnings

Measure improvements vs a benchmark

50% reduction in PoC completion time

Build for change and with maintenance in mind

To be able to balance together with improvements and innovation

Evaluate requirements against effort and cost

Provide a transparent view on time and cost to prioritise and challenge based on RoI

images: Flaticon.com

THANK YOU

Q & A