Configuration Manual

MSc Research Project
Data Analytics

Keith Scully
Student ID: X22186344

School of Computing
National College of Ireland

Supervisor: Mohammed Hasanuzzaman

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Keith Scully		
Student ID:	X22186344		
Programme:	Data Analytics	**Year:**	2024
Module:	MSc Research Project		
Supervisor:	Mohammed Hasanuzzaman		
Submission Due Date:	12th December 2024		
Project Title:	Investigating the Application of Tree-Based Machine Learning Techniques to Predict the Margin of Safety in Potential Stock Investments		
Word Count:	1,372	**Page Count:**	10

Signature: [illegible]

Date: 11/12/2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Keith Scully
Student ID: X22186344

1 Introduction

This manual describes the procedure for implementing the project. It includes hardware and software configurations along with detailed steps on how to execute the project code and re-producing the project outputs.

2 System Configuration

The following system configuration is required for this project, as described in the hardware and software subsections below, or at least a comparable system configuration that preserves compatibility of the necessary software and Python libraries.

2.1 Hardware Specification

The system on which this project was developed consisted of the following hardware specification:

CPU	12th Gen Intel Core i5-12400
GPU	n/a
RAM	32GB
Hard Disk	WD Black SN770 500GB M.2 SSD

2.2 Software Specification

The development system was using Windows 11 with Python v3.11.5. The project code was written using Jupyter Lab (v4.2.5) in the Google Chrome web browser. The Python libraries used in this project can be seen in the figure below.

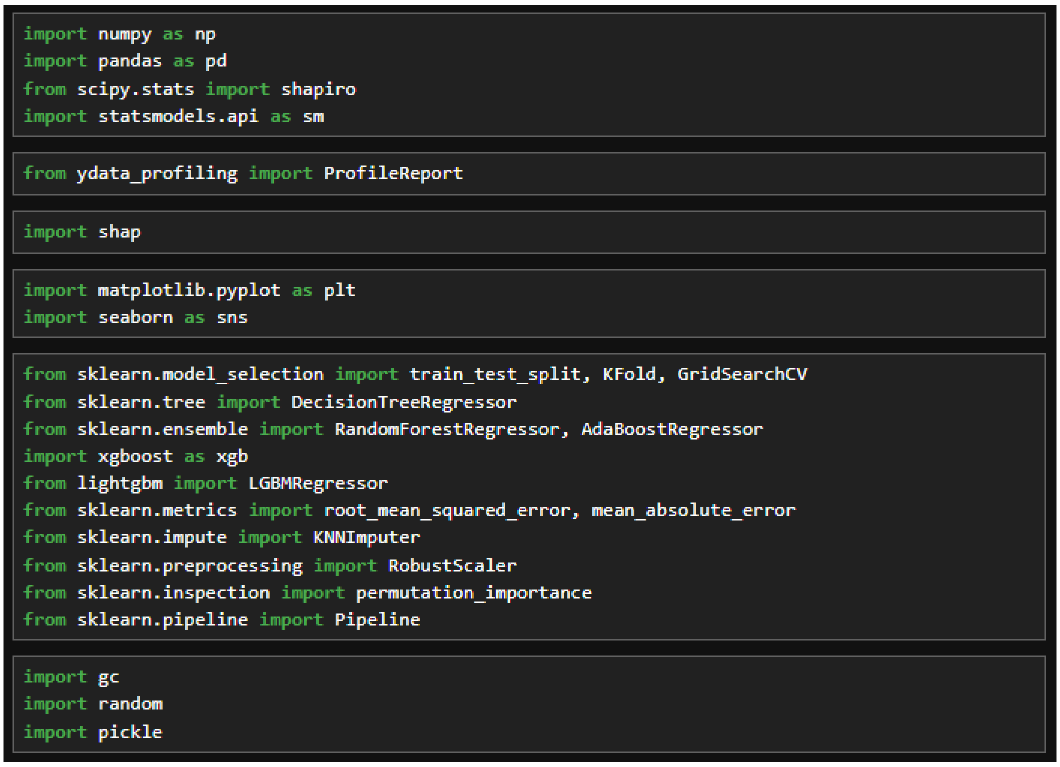

```python
import numpy as np
import pandas as pd
from scipy.stats import shapiro
import statsmodels.api as sm
```
```python
from ydata_profiling import ProfileReport
```
```python
import shap
```
```python
import matplotlib.pyplot as plt
import seaborn as sns
```
```python
from sklearn.model_selection import train_test_split, KFold, GridSearchCV
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, AdaBoostRegressor
import xgboost as xgb
from lightgbm import LGBMRegressor
from sklearn.metrics import root_mean_squared_error, mean_absolute_error
from sklearn.impute import KNNImputer
from sklearn.preprocessing import RobustScaler
from sklearn.inspection import permutation_importance
from sklearn.pipeline import Pipeline
```
```python
import gc
import random
import pickle
```

Figure 1: Required Python packages imported to the project.

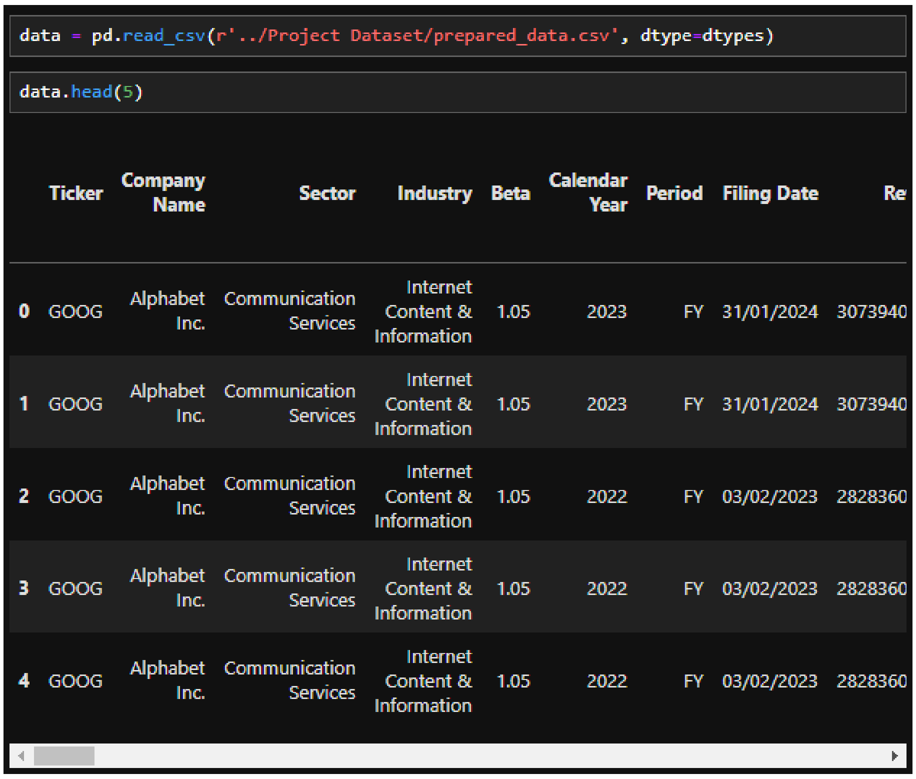
In the interest of ensuring compatibility, exact versions for all included libraries are provided as follows:

Library	**Version**
Numpy	2.0.1
Pandas	2.2.2
SciPy	1.13.1
Statsmodels	0.14.2
Ydata_Profiling	4.10.0
SHAP	0.46.0
Matplotlib	3.9.2
Seaborn	0.13.2
SciKit-Learn	1.5.1
XGBoost	2.1.1
LightGBM	4.5.0

The *gc*, *random* and *pickle* libraries are built-in libraries contained within the base Python installation.

3 Data Loading

The dataset for this project was built specifically for this project using data points from multiple sources, along with some fields which were manually calculated. The collected data is stored with the ‘prepared_data.csv’ file. This file can be loaded to the project code using the Pandas read_csv() function, as per Figure 2.



```
data = pd.read_csv(r'../Project Dataset/prepared_data.csv', dtype=dtypes)
```

```
data.head(5)
```

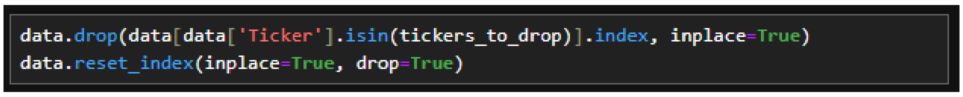
	Ticker	Company Name	Sector	Industry	Beta	Calendar Year	Period	Filing Date	Re
0	GOOG	Alphabet Inc.	Communication Services	Internet Content & Information	1.05	2023	FY	31/01/2024	3073940
1	GOOG	Alphabet Inc.	Communication Services	Internet Content & Information	1.05	2023	FY	31/01/2024	3073940
2	GOOG	Alphabet Inc.	Communication Services	Internet Content & Information	1.05	2022	FY	03/02/2023	2828360
3	GOOG	Alphabet Inc.	Communication Services	Internet Content & Information	1.05	2022	FY	03/02/2023	2828360
4	GOOG	Alphabet Inc.	Communication Services	Internet Content & Information	1.05	2022	FY	03/02/2023	2828360

Figure 2: Data loading using Pandas.

A Python dictionary called ‘dtypes’ is defined beforehand to specify that all variables in the data are loaded using the correct data type, thereby ensuring that subsequent code handles the data as expected.

4 Data Pre-Processing

The data pre-processing phase will remove some data that is not suitable for modelling, by defining a list of ticker symbols to drop contained in the ‘tickers_to_drop’ variable. These records are dropped from the dataframe and the dataframe index is reset using the code shown in Figure 3.

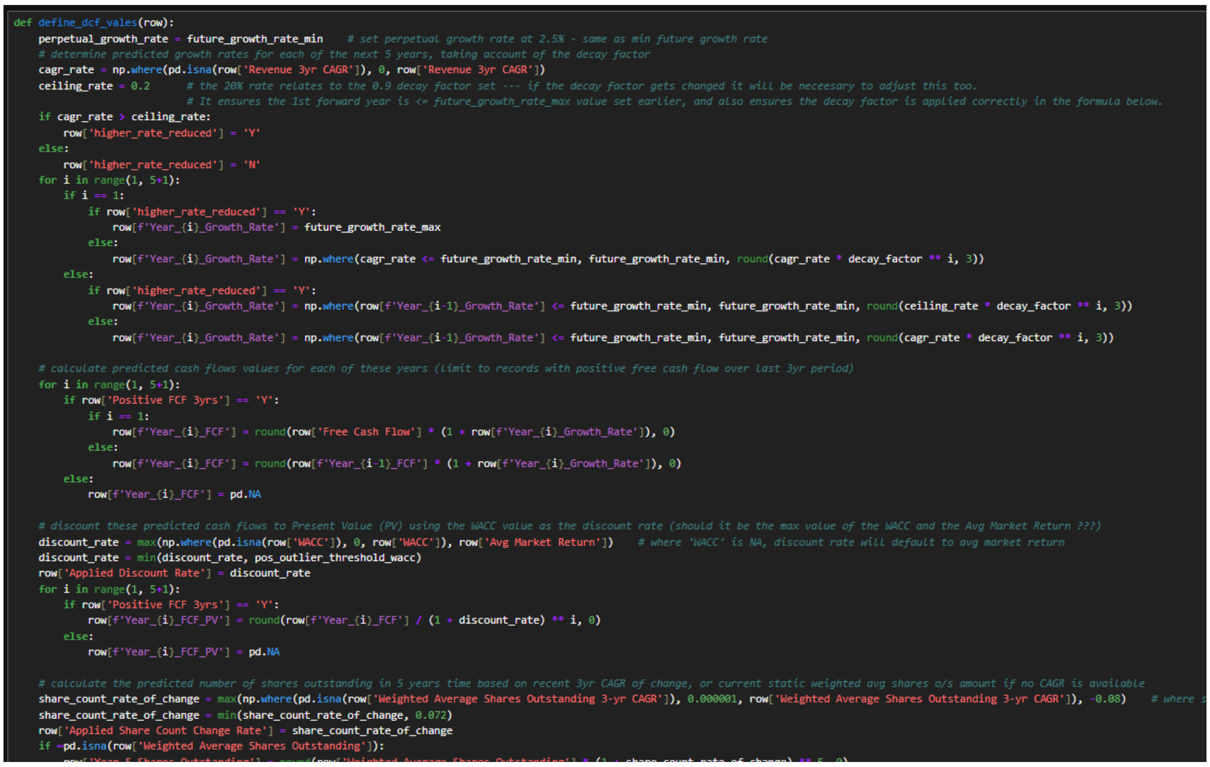


```
data.drop(data[data['Ticker'].isin(tickers_to_drop)].index, inplace=True)
data.reset_index(inplace=True, drop=True)
```

Figure 3: Dropping unwanted data.

The code can continue to be executed to repeat all data pre-processing tasks, as well as Exploratory Data Analysis (EDA). Considerable analysis is carried out to prepare for modelling the data to generate the target variable used later during the data mining phase. The process taken can be followed in the code and is supported by comments in Markdown format where necessary.

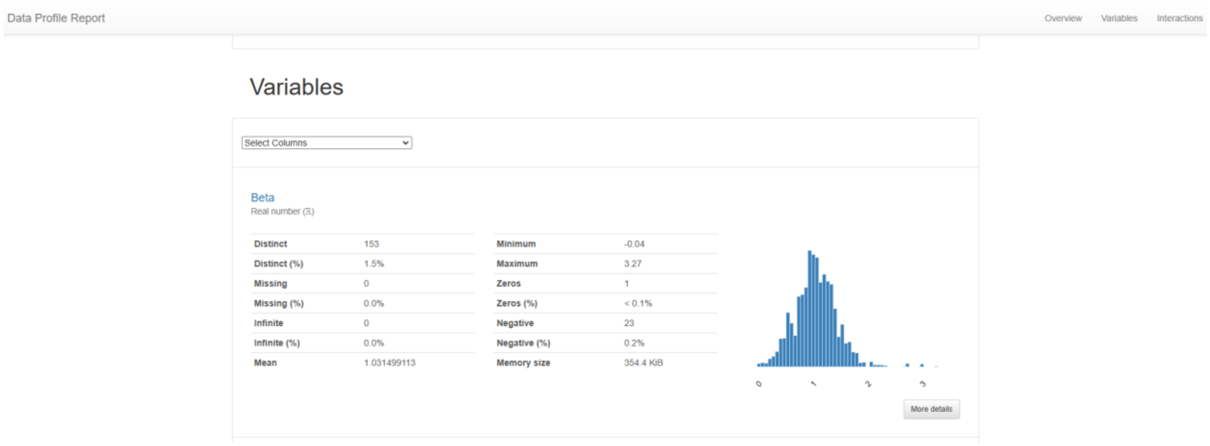
The modelling and generation of the target function is carried out by a complex function called ‘define_dcf_vales’, some of which can be seen in Figure 4, that when executed will define the target values in the dataframe.



```python
def define_dcf_vales(row):
    perpetual_growth_rate = future_growth_rate_min    # set perpetual growth rate at 2.5% - same as min future growth rate
    # determine predicted growth rates for each of the next 5 years, taking account of the decay factor
    cagr_rate = np.where(pd.isna(row['Revenue 3yr CAGR']), 0, row['Revenue 3yr CAGR'])
    ceiling_rate = 0.2      # the 20% rate relates to the 0.9 decay factor set --- if the decay factor gets changed it will be neceesary to adjust this too.
                            # It ensures the 1st forward year is <= future_growth_rate_max value set earlier, and also ensures the decay factor is applied correctly in the formula below.
    if cagr_rate > ceiling_rate:
        row['higher_rate_reduced'] = 'Y'
    else:
        row['higher_rate_reduced'] = 'N'
    for i in range(1, 5+1):
        if i == 1:
            if row['higher_rate_reduced'] == 'Y':
                row[f'Year_{i}_Growth_Rate'] = future_growth_rate_max
            else:
                row[f'Year_{i}_Growth_Rate'] = np.where(cagr_rate <= future_growth_rate_min, future_growth_rate_min, round(cagr_rate * decay_factor ** i, 3))
        else:
            if row['higher_rate_reduced'] == 'Y':
                row[f'Year_{i}_Growth_Rate'] = np.where(row[f'Year_{i-1}_Growth_Rate'] <= future_growth_rate_min, future_growth_rate_min, round(ceiling_rate * decay_factor ** i, 3))
            else:
                row[f'Year_{i}_Growth_Rate'] = np.where(row[f'Year_{i-1}_Growth_Rate'] <= future_growth_rate_min, future_growth_rate_min, round(cagr_rate * decay_factor ** i, 3))

    # calculate predicted cash flows values for each of these years (limit to records with positive free cash flow over last 3yr period)
    for i in range(1, 5+1):
        if row['Positive FCF 3yrs'] == 'Y':
            if i == 1:
                row[f'Year_{i}_FCF'] = round(row['Free Cash Flow'] * (1 + row[f'Year_{i}_Growth_Rate']), 0)
            else:
                row[f'Year_{i}_FCF'] = round(row[f'Year_{i-1}_FCF'] * (1 + row[f'Year_{i}_Growth_Rate']), 0)
        else:
            row[f'Year_{i}_FCF'] = pd.NA

    # discount these predicted cash flows to Present Value (PV) using the WACC value as the discount rate (should it be the max value of the WACC and the Avg Market Return ???)
    discount_rate = max(np.where(pd.isna(row['WACC']), 0, row['WACC']), row['Avg Market Return'])    # where 'WACC' is NA, discount rate will default to avg market return
    discount_rate = min(discount_rate, pos_outlier_threshold_wacc)
    row['Applied Discount Rate'] = discount_rate
    for i in range(1, 5+1):
        if row['Positive FCF 3yrs'] == 'Y':
            row[f'Year_{i}_FCF_PV'] = round(row[f'Year_{i}_FCF'] / (1 + discount_rate) ** i, 0)
        else:
            row[f'Year_{i}_FCF_PV'] = pd.NA

    # calculate the predicted number of shares outstanding in 5 years time based on recent 3yr CAGR of change, or current static weighted avg shares o/s amount if no CAGR is available
    share_count_rate_of_change = max(np.where(pd.isna(row['Weighted Average Shares Outstanding 3-yr CAGR']), 0.000001, row['Weighted Average Shares Outstanding 3-yr CAGR']), -0.08)    # where s
    share_count_rate_of_change = min(share_count_rate_of_change, 0.072)
    row['Applied Share Count Change Rate'] = share_count_rate_of_change
    if ~pd.isna(row['Weighted Average Shares Outstanding']):
        row['Year_5_Shares_Outstanding'] = round(row['Weighted Average Shares Outstanding'] * (1 + share_count_rate_of_change) ** 5, 0)
```

Figure 4: A partial view of the detailed function for generating the target variable.

During EDA activities in the code, the Ydata_Profiling library will produce a HMTL file in the code execution folder called ‘Data Profile Report.html’. Once opened in the browser this file will provide descriptive statistics of all variables in the dataset, with an example shown in Figure 5 below.



Variables

Beta
Real number (ℝ)

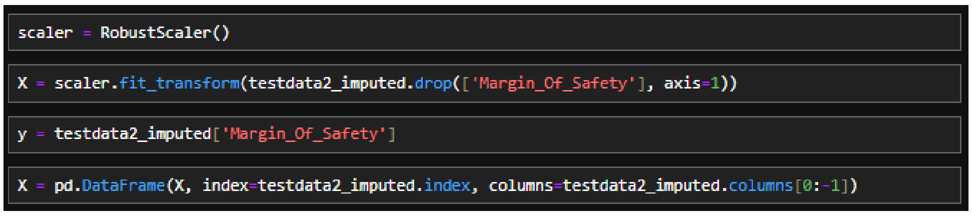
Distinct	153	Minimum	-0.04
Distinct (%)	1.5%	Maximum	3.27
Missing	0	Zeros	1
Missing (%)	0.0%	Zeros (%)	< 0.1%
Infinite	0	Negative	23
Infinite (%)	0.0%	Negative (%)	0.2%
Mean	1.031499113	Memory size	354.4 KiB

Figure 5: Descriptive statistics from ‘Data Profile Report.html’ file generated by the Ydata_Profiling library.

Extensive EDA activities are carried out through further code execution and supported by markdown comments as necessary.

5 Data Transformation

Data Transformation primarily involves scaling input features using the Robust Scaler from SciKit-Learn and can be executed using the code in Figure 6.

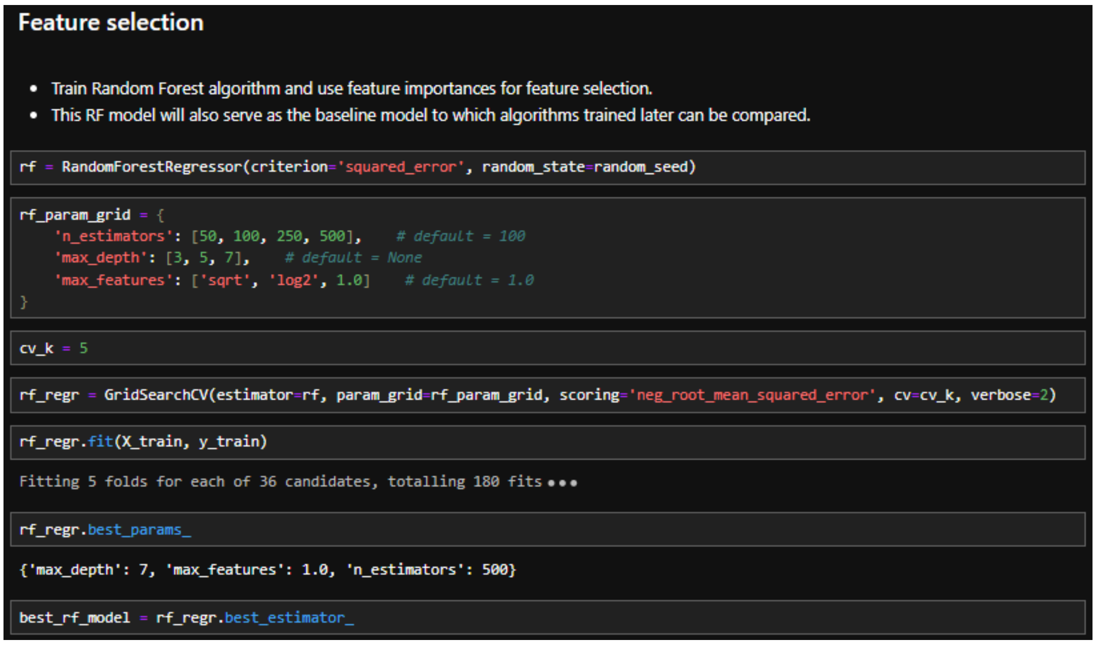


```
scaler = RobustScaler()
```
```
X = scaler.fit_transform(testdata2_imputed.drop(['Margin_Of_Safety'], axis=1))
```
```
y = testdata2_imputed['Margin_Of_Safety']
```
```
X = pd.DataFrame(X, index=testdata2_imputed.index, columns=testdata2_imputed.columns[0:-1])
```

Figure 6: Robust scaling applied to the input features.

6 Data Mining

Data Mining phase begins with training a baseline model that will is also utilised for feature selection using a Random Forest algorithm, as shown in Figure 7. Features are selected based on highest degree of importance, that are then used in training of all other model instances.



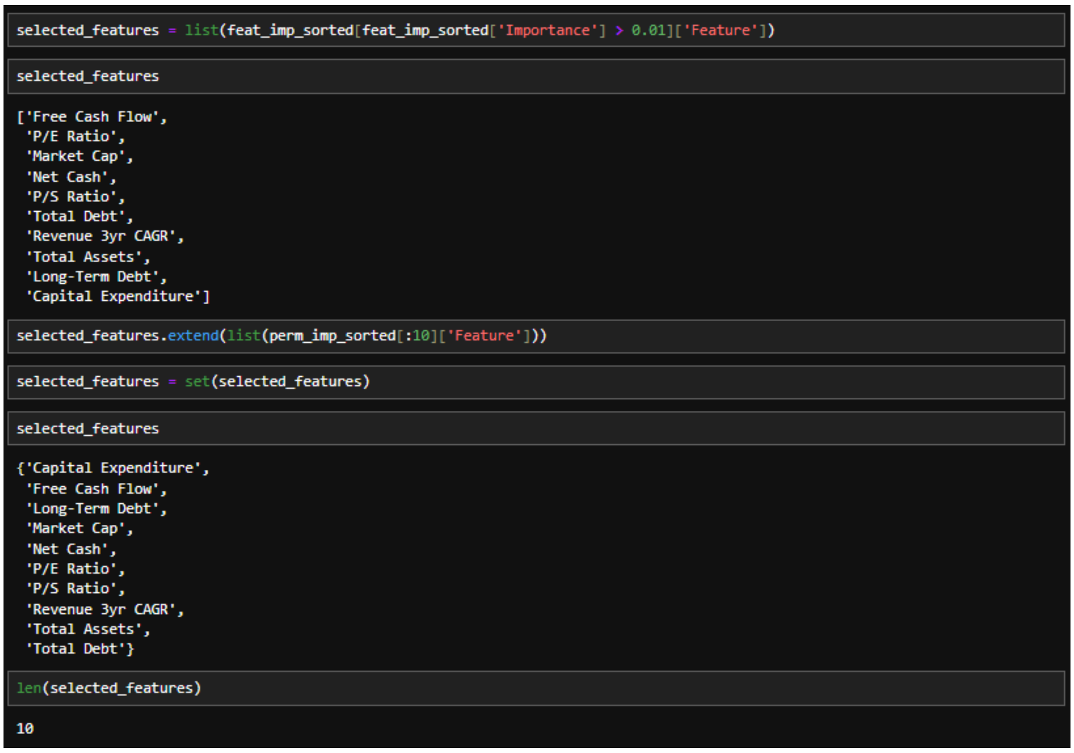
Feature selection

- Train Random Forest algorithm and use feature importances for feature selection.
- This RF model will also serve as the baseline model to which algorithms trained later can be compared.

```
rf = RandomForestRegressor(criterion='squared_error', random_state=random_seed)
```
```
rf_param_grid = {
    'n_estimators': [50, 100, 250, 500],    # default = 100
    'max_depth': [3, 5, 7],    # default = None
    'max_features': ['sqrt', 'log2', 1.0]    # default = 1.0
}
```
```
cv_k = 5
```
```
rf_regr = GridSearchCV(estimator=rf, param_grid=rf_param_grid, scoring='neg_root_mean_squared_error', cv=cv_k, verbose=2)
```
```
rf_regr.fit(X_train, y_train)
```
```
Fitting 5 folds for each of 36 candidates, totalling 180 fits •••
```
```
rf_regr.best_params_
```
```
{'max_depth': 7, 'max_features': 1.0, 'n_estimators': 500}
```
```
best_rf_model = rf_regr.best_estimator_
```

Figure 7: Training of baseline Random Forest model.

The features selected are a combination of features that had an importance value >= 0.1 which is then extended with any other features that had a permutation importance value of >=0.1, a step complete in the code shown in Figure 8.



```
selected_features = list(feat_imp_sorted[feat_imp_sorted['Importance'] > 0.01]['Feature'])
```

```
selected_features
```

```
['Free Cash Flow',
 'P/E Ratio',
 'Market Cap',
 'Net Cash',
 'P/S Ratio',
 'Total Debt',
 'Revenue 3yr CAGR',
 'Total Assets',
 'Long-Term Debt',
 'Capital Expenditure']
```

```
selected_features.extend(list(perm_imp_sorted[:10]['Feature']))
```

```
selected_features = set(selected_features)
```

```
selected_features
```

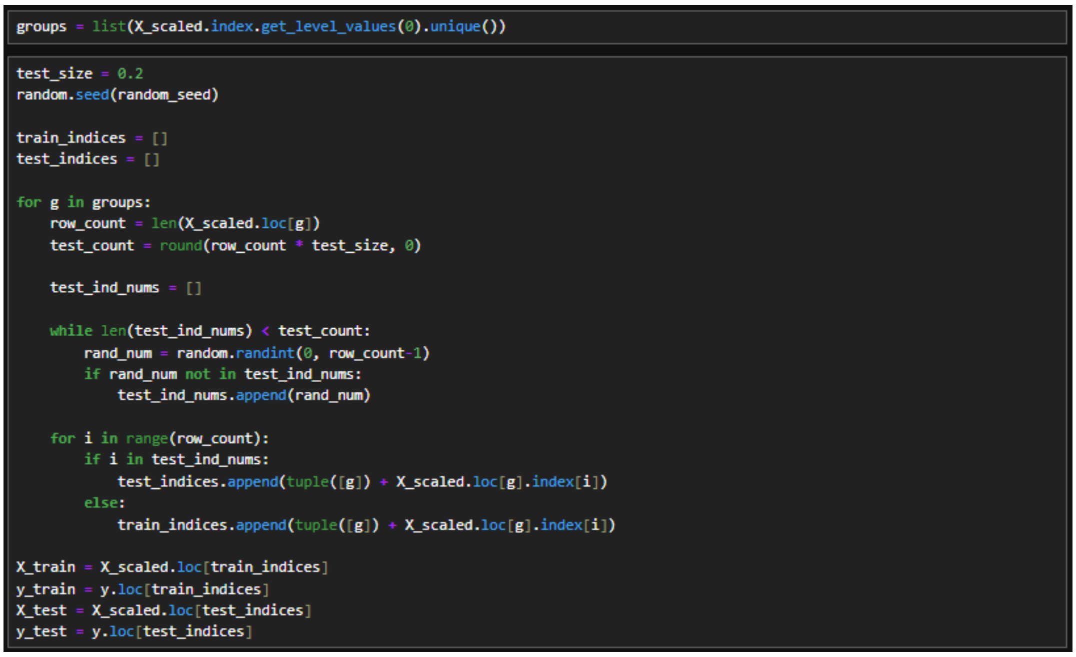
```
{'Capital Expenditure',
 'Free Cash Flow',
 'Long-Term Debt',
 'Market Cap',
 'Net Cash',
 'P/E Ratio',
 'P/S Ratio',
 'Revenue 3yr CAGR',
 'Total Assets',
 'Total Debt'}
```

```
len(selected_features)
```

```
10
```

Figure 8: Feature selection.

A customised train-test split occurs based on grouping of companies and business sectors, as shown in Figure 9.

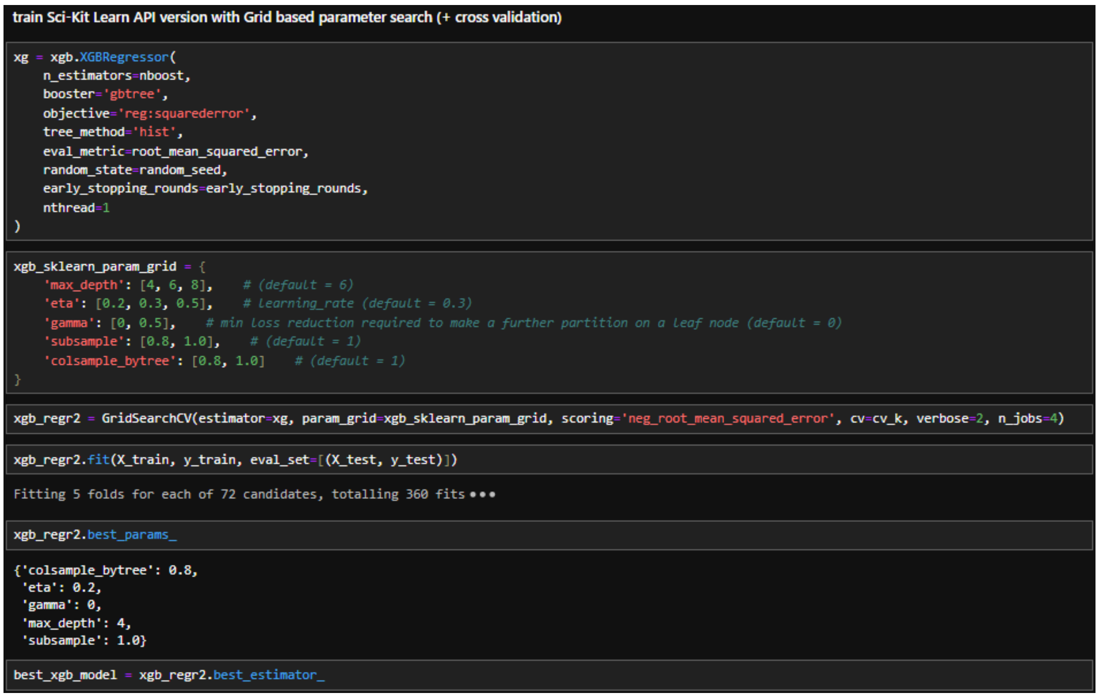


```
groups = list(X_scaled.index.get_level_values(0).unique())
```

```
test_size = 0.2
random.seed(random_seed)

train_indices = []
test_indices = []

for g in groups:
    row_count = len(X_scaled.loc[g])
    test_count = round(row_count * test_size, 0)

    test_ind_nums = []

    while len(test_ind_nums) < test_count:
        rand_num = random.randint(0, row_count-1)
        if rand_num not in test_ind_nums:
            test_ind_nums.append(rand_num)

    for i in range(row_count):
        if i in test_ind_nums:
            test_indices.append(tuple([g]) + X_scaled.loc[g].index[i])
        else:
            train_indices.append(tuple([g]) + X_scaled.loc[g].index[i])

X_train = X_scaled.loc[train_indices]
y_train = y.loc[train_indices]
X_test = X_scaled.loc[test_indices]
y_test = y.loc[test_indices]
```

Figure 9: Creation of custom train-test split.

Each model is trained using 5-fold cross validation and with hyperparameter optimisation using the grid search technique. The best model is then chosen based on its performance on the evaluation set. The code for the part of the process is shown in Figure 10 where the training of the XGBoost model occurs, and very similar steps are followed for each of the models trained.



train Sci-Kit Learn API version with Grid based parameter search (+ cross validation)

```
xg = xgb.XGBRegressor(
    n_estimators=nboost,
    booster='gbtree',
    objective='reg:squarederror',
    tree_method='hist',
    eval_metric=root_mean_squared_error,
    random_state=random_seed,
    early_stopping_rounds=early_stopping_rounds,
    nthread=1
)
```

```
xgb_sklearn_param_grid = {
    'max_depth': [4, 6, 8],    # (default = 6)
    'eta': [0.2, 0.3, 0.5],    # learning_rate (default = 0.3)
    'gamma': [0, 0.5],    # min loss reduction required to make a further partition on a leaf node (default = 0)
    'subsample': [0.8, 1.0],    # (default = 1)
    'colsample_bytree': [0.8, 1.0]    # (default = 1)
}
```

```
xgb_regr2 = GridSearchCV(estimator=xg, param_grid=xgb_sklearn_param_grid, scoring='neg_root_mean_squared_error', cv=cv_k, verbose=2, n_jobs=4)
```

```
xgb_regr2.fit(X_train, y_train, eval_set=[(X_test, y_test)])
```

```
Fitting 5 folds for each of 72 candidates, totalling 360 fits •••
```

```
xgb_regr2.best_params_
```

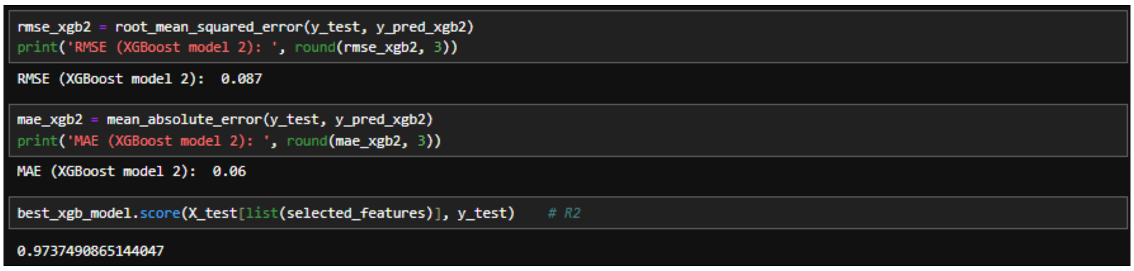
```
{'colsample_bytree': 0.8,
 'eta': 0.2,
 'gamma': 0,
 'max_depth': 4,
 'subsample': 1.0}
```

```
best_xgb_model = xgb_regr2.best_estimator_
```

Figure 10: XGBoost model training.

7 Evaluation

All trained models are evaluated using common RMSE, MAE and R^2 regression metrics, using code samples such as those in Figure 11.



```
rmse_xgb2 = root_mean_squared_error(y_test, y_pred_xgb2)
print('RMSE (XGBoost model 2): ', round(rmse_xgb2, 3))
```

```
RMSE (XGBoost model 2):  0.087
```

```
mae_xgb2 = mean_absolute_error(y_test, y_pred_xgb2)
print('MAE (XGBoost model 2): ', round(mae_xgb2, 3))
```

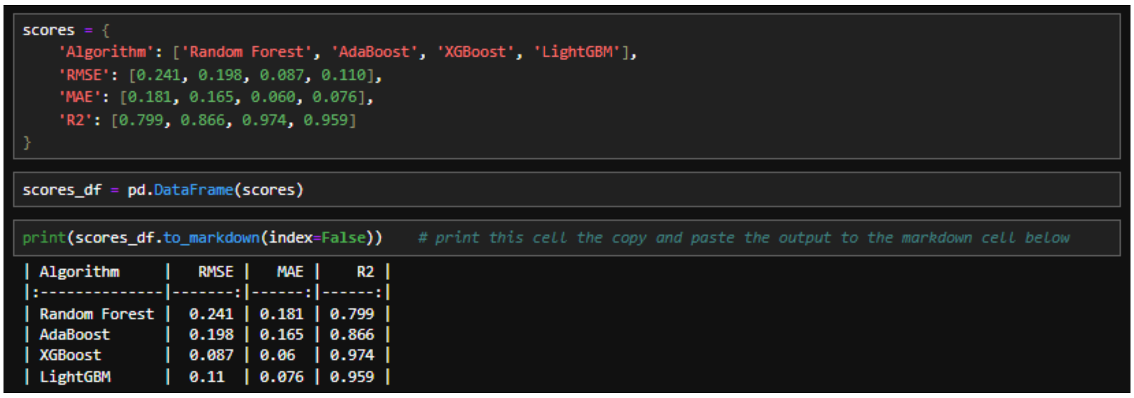
```
MAE (XGBoost model 2):  0.06
```

```
best_xgb_model.score(X_test[list(selected_features)], y_test)    # R2
```

```
0.9737490865144047
```

Figure 11: Determination of the XGBoost model's evaluation scores.

Scores are then compiled and stored in a new dataframe from which a markdown table is printed to be copied and used in any markdown cell for displaying the overall results table. This step is detailed in Figure 12.



```
scores = {
    'Algorithm': ['Random Forest', 'AdaBoost', 'XGBoost', 'LightGBM'],
    'RMSE': [0.241, 0.198, 0.087, 0.110],
    'MAE': [0.181, 0.165, 0.060, 0.076],
    'R2': [0.799, 0.866, 0.974, 0.959]
}
```

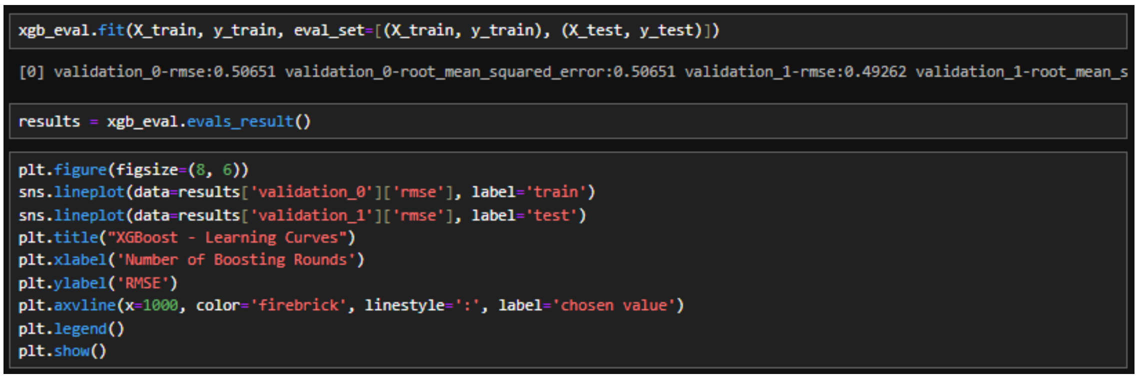
```
scores_df = pd.DataFrame(scores)
```

```
print(scores_df.to_markdown(index=False))    # print this cell the copy and paste the output to the markdown cell below
```

```
| Algorithm     |   RMSE |   MAE |    R2 |
|:--------------|-------:|------:|------:|
| Random Forest |  0.241 | 0.181 | 0.799 |
| AdaBoost      |  0.198 | 0.165 | 0.866 |
| XGBoost       |  0.087 | 0.06  | 0.974 |
| LightGBM      |  0.11  | 0.076 | 0.959 |
```

Figure 12: Collection of scores to produce markdown table.

A separate model is trained using the tuned hyperparameters of the final XGBoost model, but with an extreme value for the incremental training hyperparameter *n_estimators*, which provides sufficient scope for identifying any potential overfitting that may occur in the model. Evaluations are directed on both the train and test sets, with results extracted from this evaluation model to plot learning curves and support better understanding of the model fit. The training, extraction of evaluation results and plotting is carried out in the code shown in Figure 13.



```
xgb_eval.fit(X_train, y_train, eval_set=[(X_train, y_train), (X_test, y_test)])
```

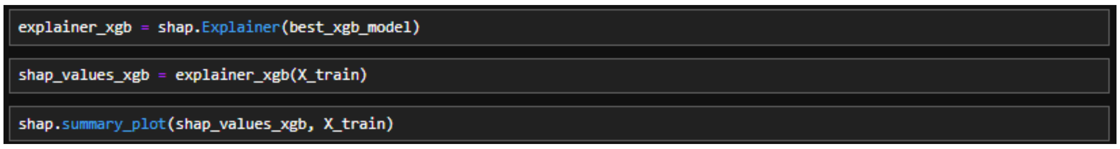
```
[0] validation_0-rmse:0.50651 validation_0-root_mean_squared_error:0.50651 validation_1-rmse:0.49262 validation_1-root_mean_s
```

```
results = xgb_eval.evals_result()
```

```
plt.figure(figsize=(8, 6))
sns.lineplot(data=results['validation_0']['rmse'], label='train')
sns.lineplot(data=results['validation_1']['rmse'], label='test')
plt.title("XGBoost - Learning Curves")
plt.xlabel('Number of Boosting Rounds')
plt.ylabel('RMSE')
plt.axvline(x=1000, color='firebrick', linestyle=':', label='chosen value')
plt.legend()
plt.show()
```

Figure 13: Evaluation model to determine characteristics of model fit.

SHAP values are calculated using a SHAP explainer object from which a bee-swarm summary plot is created to show how each feature has impacted predictions in the model, as shown in Figure 14.



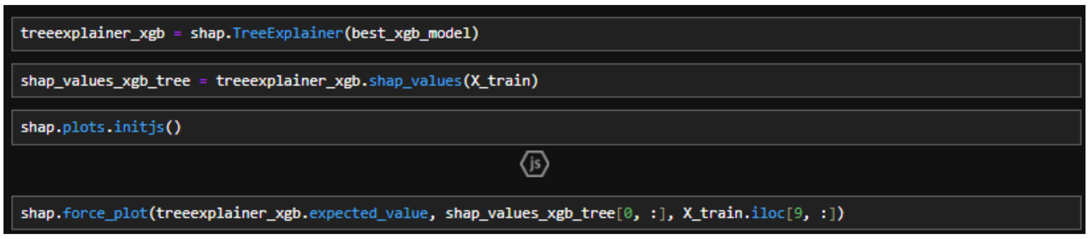
```
explainer_xgb = shap.Explainer(best_xgb_model)
```

```
shap_values_xgb = explainer_xgb(X_train)
```

```
shap.summary_plot(shap_values_xgb, X_train)
```

Figure 14: SHAP value extraction and generation of summary plot.

In addition, a tree explainer object is used to generated SHAP values for use in a force plot which can be utilised for any given observation in the dataset. The code for this is shown in Figure 15.



```
treeexplainer_xgb = shap.TreeExplainer(best_xgb_model)
```

```
shap_values_xgb_tree = treeexplainer_xgb.shap_values(X_train)
```

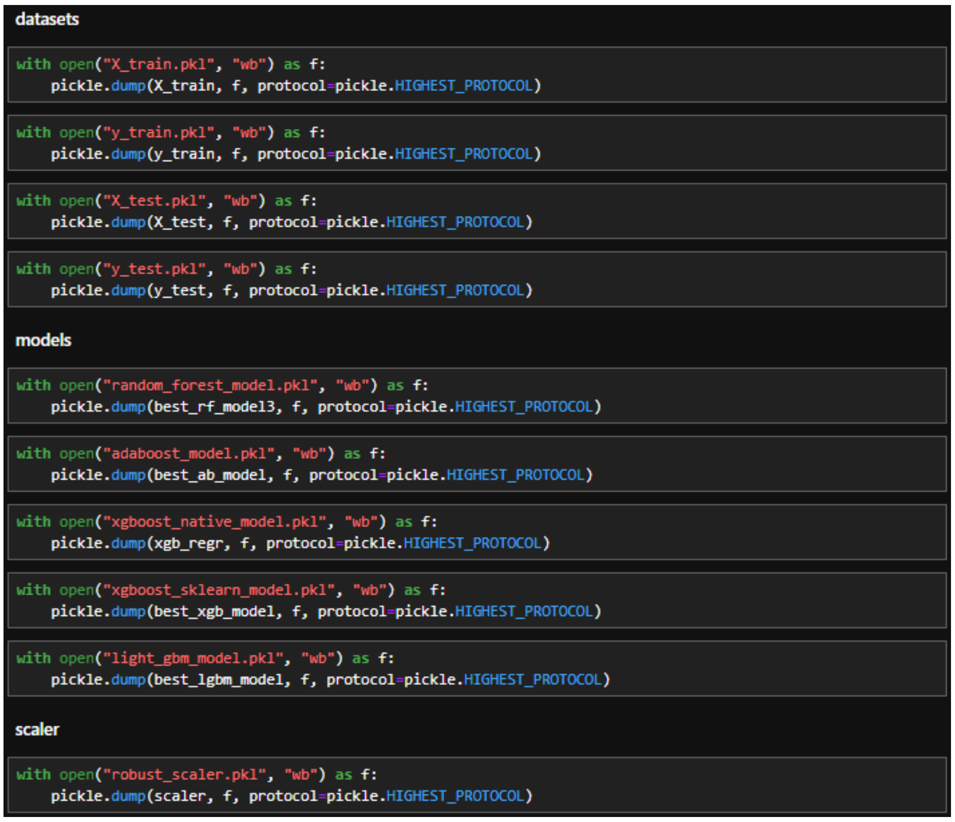
```
shap.plots.initjs()
```

```
shap.force_plot(treeexplainer_xgb.expected_value, shap_values_xgb_tree[0, :], X_train.iloc[9, :])
```

Figure 15: SHAP value extraction and generation of force plot.

8 Pickle Objects

Using Python's built-in *pickle* library, all datasets, models and the robust scaler object, are pickled and stored as independent files on the system for later use. This step is shown in Figure 16.



datasets

```
with open("X_train.pkl", "wb") as f:
    pickle.dump(X_train, f, protocol=pickle.HIGHEST_PROTOCOL)
```

```
with open("y_train.pkl", "wb") as f:
    pickle.dump(y_train, f, protocol=pickle.HIGHEST_PROTOCOL)
```

```
with open("X_test.pkl", "wb") as f:
    pickle.dump(X_test, f, protocol=pickle.HIGHEST_PROTOCOL)
```

```
with open("y_test.pkl", "wb") as f:
    pickle.dump(y_test, f, protocol=pickle.HIGHEST_PROTOCOL)
```

models

```
with open("random_forest_model.pkl", "wb") as f:
    pickle.dump(best_rf_model3, f, protocol=pickle.HIGHEST_PROTOCOL)
```

```
with open("adaboost_model.pkl", "wb") as f:
    pickle.dump(best_ab_model, f, protocol=pickle.HIGHEST_PROTOCOL)
```

```
with open("xgboost_native_model.pkl", "wb") as f:
    pickle.dump(xgb_regr, f, protocol=pickle.HIGHEST_PROTOCOL)
```

```
with open("xgboost_sklearn_model.pkl", "wb") as f:
    pickle.dump(best_xgb_model, f, protocol=pickle.HIGHEST_PROTOCOL)
```

```
with open("light_gbm_model.pkl", "wb") as f:
    pickle.dump(best_lgbm_model, f, protocol=pickle.HIGHEST_PROTOCOL)
```

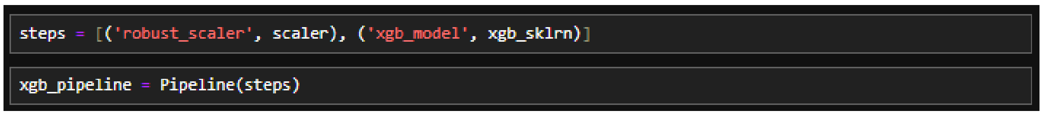
scaler

```
with open("robust_scaler.pkl", "wb") as f:
    pickle.dump(scaler, f, protocol=pickle.HIGHEST_PROTOCOL)
```

Figure 16: Saving and storing all datasets, models and transformers using the Python pickle library.

9 Pipeline Generation

A pipeline object is created using the pipeline module from SciKit-Learn which contains the fitted scaler and the pre-trained XGBoost model, which can be used for making predictions on any new data collected at a later point in time. This pipeline is generated using the code in Figure 17.

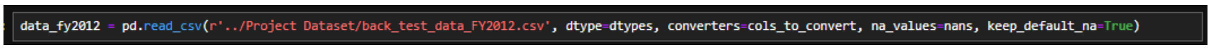


```python
steps = [('robust_scaler', scaler), ('xgb_model', xgb_sklrn)]
```
```python
xgb_pipeline = Pipeline(steps)
```

Figure 17: Generating new pipeline for predictions on new data.

10 Back-Testing

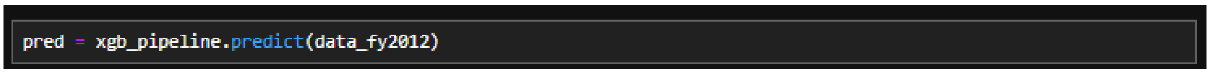
New data was collected particularly for back-testing the model on historical data previously unseen. This data is stored in the CSV with the filename ‘back_test_data_FY2012.csv’, which is loaded to the project to make prediction, as shown in Figure 18.



```python
data_fy2012 = pd.read_csv(r'../Project Dataset/back_test_data_FY2012.csv', dtype=dtypes, converters=cols_to_convert, na_values=nans, keep_default_na=True)
```

Figure 18: Reading new data for back-test predictions.

With the pipeline available, predictions can then be made using this new pipeline object on the new unseen data, as shown in Figure 19 below.



```python
pred = xgb_pipeline.predict(data_fy2012)
```

Figure 19: Making predictions using the newly created pipeline.

Following predictions made through the pipeline, the back-test dataset is sorted according to the predicted values and the top 20 ticker symbols are selected to form a theoretical portfolio, for which more new pricing data had to be collected. This data is collected and stored in the file ‘monthly gains.xlsx’ in which some final analysis is carried out to compare the performance of the theoretical portfolio against the wider market.

11 Conclusion

This manual provides all steps required to re-create this project using the provided code artifact and data files, using the Python programming environment specifications detailed in this manual, which may be executed using the Jupyter Lab notebook IDE for consistency with original project development.