Configuration Manual

MSc Research Project
Data Analytics

Sreelakshmi Chittazhi
Student ID: x22210466

School of Computing
National College of Ireland

Supervisor: Ahmed Makki

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Sreelakshmi Chittazhi
Student ID:	x22210466
Programme:	Data Analytics
Year:	2023-2024
Module:	MSc Research Project
Supervisor:	Ahmed Makki
Submission Due Date:	12/08/2024
Project Title:	Configuration Manual
Word Count:	419
Page Count:	8

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Sreelakshmi Chittazhi
Date:	12th August 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

Sreelakshmi Chittazhi
x22210466

1 Introduction

All the requirements that are necessary for this research has been included in this configuration manual. The software and hardware requirements as well as the code required for data importing, preprocessing, model building, and evaluation has also been included.

Section 2 discuss about the information about the environment used. The data collection and loading are described in section 3. The next section explains about the data preprocessing steps. Section 5 describes about the splitting of the data, model building and the evaluation.

2 Environment

2.1 Hardware Requirement

Detailed information about the hardware and software requirements as been shown in the table below.

Operating System	Windows 11
RAM	8 GB
Hard Disc	470 GB

Table 1: System Specifications

2.2 SoftwareRequirement

Programming Tools	Jupyter Notebook
Web Browser	Google Chrome
Other Required Software	Overleaf, Microsoft Word

Table 2: Software Details

3 Data collection and loading

This section explains the code for data manipulation and importing important libraries required for data loading, cleaning and building model. The data was collected from kaggle https://www.kaggle.com/competitions/predict-energy-behavior-of-prosumers/data

```python
# importing necessary libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.feature_selection import SelectKBest, mutual_info_regressi
import optuna
import lightgbm as lgb
from sklearn.model_selection import train_test_split, cross_val_score
import lightgbm as lgb
from sklearn.metrics import mean_squared_error , mean_absolute_error
from sklearn.ensemble import RandomForestRegressor
```

Figure 1: Importing libraries

```python
data = pd.read_csv('train.csv')

data.head(10)
```

```python
gas_price = pd.read_csv('gas_prices.csv')
```

```python
electricity_price_data = pd.read_csv('electricity_prices.csv')
```

```python
forecast_data = pd.read_csv('forecast_weather.csv')
```

Figure 2: Data loading

4 Data Preprocessing

In this section the data preprocessing steps and the code used to plot the charts, removing null values

```python
#check if there is any null values
data.isnull().sum()/data.shape[0]*100
```

```
county               0.00000
is_business          0.00000
product_type         0.00000
target               0.02616
is_consumption       0.00000
datetime             0.00000
data_block_id        0.00000
row_id               0.00000
prediction_unit_id   0.00000
dtype: float64
```

```python
#since the % of null value is negligible we ar deleteing it
data = data.dropna()
```

Figure 3: Null Values

```python
plt.figure(figsize=(7, 5))
sns.set_theme(style = "whitegrid")
ax = sns.countplot(data=data, x='is_consumption')

# Annotate the bars with the count values
for p in ax.patches:
    ax.annotate(f'{int(p.get_height())}',
                (p.get_x() + p.get_width() / 2., p.get_height()),
                ha='center', va='center', xytext=(0, 10),
                textcoords='offset points')

plt.title('Count Plot of is_consumption')
plt.xlabel('is_consumption')
plt.ylabel('Count')
plt.show()
```

Figure 4: Count plot

4.1 Feature Selection

Feature selection has been implemented through SelectKbest method. The data has been splitted x_sample and y_sample.

```python
# Assuming 'merged_data' is already defined
X_sample = data.drop(columns=['target'])

# Select only numeric columns
X_sample = X_sample.select_dtypes(include=[np.number])

y_sample = data['target']

selector = SelectKBest(score_func=mutual_info_regression, k=15)
selector.fit(X_sample, y_sample)

selected_indices = selector.get_support(indices=True)
selected_features = X_sample.columns[selected_indices]
print("Selected features:")
print(selected_features)
```

Figure 5: Feature selection

```python
feature_scores = selector.scores_[selected_indices]
# Create a DataFrame for the selected features and their scores
feature_scores_df = pd.DataFrame({'Feature': selected_features, 'Score': feature_scores})

# Sort the DataFrame by score
feature_scores_df = feature_scores_df.sort_values(by='Score', ascending=False)
```

```python
feature_scores_df
```

Figure 6: Feature selection

The feature_scores_df will show the features and score for each feature contributing to the target variable.

4.2 Feature Engineering

The variable such as hour, day, month, day of the week, day of the year has been extracted for the analysis.

```
data['hour'] =  data.datetime.dt.hour
data['day'] = data.datetime.dt.day
data['month'] = data.datetime.dt.month
data['day_of_week'] = data.datetime.dt.dayofweek
data['day_of_year'] = data.datetime.dt.dayofyear
```

Figure 7: Feature Engineering

After all the preprocessing all the dataset as been merged.

```
print(data.shape)
print(data.info())
```

```
(987546, 35)
<class 'pandas.core.frame.DataFrame'>
Int64Index: 987546 entries, 1488 to 991335
Data columns (total 35 columns):
 #   Column                             Non-Null Count   Dtype
---  ------                             --------------   -----
 0   county                             987546 non-null  int64
 1   is_business                        987546 non-null  int64
 2   product_type                       987546 non-null  int64
 3   target                             987546 non-null  float64
 4   is_consumption                     987546 non-null  int64
 5   datetime                           987546 non-null  datetime64[ns, UTC]
 6   data_block_id                      987546 non-null  int64
 7   row_id                             987546 non-null  int64
 8   prediction_unit_id                 987546 non-null  int64
 9   year                               987546 non-null  int64
 10  date                               987546 non-null  object
 11  10_metre_u_wind_component          987546 non-null  float64
 12  10_metre_v_wind_component          987546 non-null  float64
 13  cloudcover_high                    987546 non-null  float64
 14  cloudcover_low                     987546 non-null  float64
 15  cloudcover_mid                     987546 non-null  float64
 16  cloudcover_total                   987546 non-null  float64
 17  dewpoint                           987546 non-null  float64
 18  direct_solar_radiation             987546 non-null  float64
 19  snowfall                           987546 non-null  float64
 20  surface_solar_radiation_downwards  987546 non-null  float64
 21  temperature                        987546 non-null  float64
 22  total_precipitation                987546 non-null  float64
 23  forecast_date                      987546 non-null  datetime64[ns]
 24  lowest_price_per_mwh               987546 non-null  float64
 25  highest_price_per_mwh              987546 non-null  float64
 26  avg_price                          987546 non-null  float64
 27  hour                               987546 non-null  int64
 28  day                                987546 non-null  int64
 29  month                              987546 non-null  int64
 30  day_of_week                        987546 non-null  int64
 31  day_of_year                        987546 non-null  int64
 32  euros_per_mwh                      987546 non-null  float64
 33  eic_count                          987546 non-null  float64
 34  installed_capacity                 987546 non-null  float64
dtypes: datetime64[ns, UTC](1), datetime64[ns](1), float64(19), int64(13), object(1)
```

Figure 8: Merged Dataset

4.3 Hyper parameter tuning

Initially optuna library should be installed using pip method.

```python
# Define the objective function for Optuna
def objective(trial):
    param = {
        'n_estimators': trial.suggest_int('n_estimators', 19000, 21000),
        'learning_rate': trial.suggest_float('learning_rate', 0.05, 0.15),
        'num_leaves': trial.suggest_int('num_leaves', 50, 100),
        'max_depth': trial.suggest_int('max_depth', 15, 25),
        'subsample': trial.suggest_float('subsample', 0.7, 0.9)
    }

    lgb_reg = lgb.LGBMRegressor(**param, device='gpu')

    # Split the training data further for validation
    X_train_split, X_val_split, y_train_split, y_val_split = train_test_split(X_train, y_train, test_size=0.2, random_state=42)

    # Define the callbacks
    callbacks = [
        lgb.early_stopping(stopping_rounds=100, verbose=False),
        lgb.log_evaluation(period=100)
    ]

    # Train the model with the specified parameters
    lgb_reg.fit(X_train_split, y_train_split,
                eval_set=[(X_val_split, y_val_split)],
                callbacks=callbacks)

    # Predict on the validation set
    y_pred = lgb_reg.predict(X_val_split)

    # Calculate the mean squared error
    mse = mean_squared_error(y_val_split, y_pred)

    return mse

# Create an Optuna study to minimize the objective function
study = optuna.create_study(direction='minimize')

# Optimize the study over 10 trials
study.optimize(objective, n_trials=10)

# Retrieve and print the best hyperparameters
best_params = study.best_params
print("Best Hyperparameters:", best_params)
```

Figure 9: Hyperparameter tuning through Optuna

5 Data modelling and evaluation

```
x_train , x_test , y_train , y_test = train_test_split(data[cols] ,data['target'] , random_state = 42 , test_size = 0.20)

print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

Figure 10: Splitting the dataset as train and test

Figure 11 and 12 below shows the implementation Light GBM model and Random Forest Regressor.

```
callbacks = [
    lgb.early_stopping(stopping_rounds= 1000,  verbose=True),
    lgb.callback.log_evaluation(period=1000)
]

lgb_model = lgb.LGBMRegressor(  n_estimators = 19337 , max_depth = 22  , subsample = 0.8010925204759234 , learning_rate = 0.07972332158676076 , num_leaves = 77 )
lgb_model.fit(x_train, y_train , eval_set = [(x_test , y_test)] , callbacks = callbacks )
```

Figure 11: Light GBM

```
# Define the model with additional hyperparameters
rf_1 = RandomForestRegressor(
    n_estimators=30,
    max_depth=25,
    min_samples_split=10,
    min_samples_leaf=5,
    max_features='sqrt',
    bootstrap=True,
    oob_score=True,
    n_jobs=-1,
    random_state=42,
    verbose=1,
    ccp_alpha=0.01
)
```

```
rf_1.fit(x_train, y_train)
```

Figure 12: Random Forest Regressor

Next figure explain the stacking method, wherein the prediction are stacked together.

```
# Combine predictions as new features
stack_train = np.column_stack((lgb_train_preds, rf_1_train_preds))
stack_test = np.column_stack((lgb_test_preds, rf_1_test_preds))
```

Figure 13: Stacking the predictions

5.1 Linear Regression as meta model

```
# Train meta-model
from sklearn.linear_model import LinearRegression
meta_model = LinearRegression()
meta_model.fit(stack_train, y_train)
stacked_preds = meta_model.predict(stack_test)
```

```
# Evaluate the performance
mae_hybrid = mean_absolute_error(y_test, stacked_preds)
print(f"Mean Absolute Error with Linear Regression as meta-model: {mae_hybrid}")

# Compare with LightGBM model
mae_lgb = mean_absolute_error(y_test, lgb_test_preds)
print(f"Mean Absolute Error of LightGBM model: {mae_lgb}")

# Compare with Random Forest model
mae_rf = mean_absolute_error(y_test, rf_1_test_preds)
print(f"Mean Absolute Error of Random Forest model: {mae_rf}")
```

```
Mean Absolute Error with Linear Regression as meta-model: 18.20309618435062
Mean Absolute Error of LightGBM model: 18.154759848817246
Mean Absolute Error of Random Forest model: 29.958215438902055
```

Figure 14: Hybrid model_1

5.2 Ridge Regression as meta model

```
# # Train a new meta-model
from sklearn.linear_model import Ridge

meta_model_ridge = Ridge(alpha=1.0)
meta_model_ridge.fit(stack_train, y_train)
stacked_preds_ridge = meta_model_ridge.predict(stack_test)
```

```
mae_ridge =  mean_absolute_error(y_test, stacked_preds_ridge)
print(f"Mean Absolute Error with Ridge Regression as meta-model: {mae_ridge}")
# Compare with LightGBM model
mae_lgb = mean_absolute_error(y_test, lgb_test_preds)
print(f"Mean Absolute Error of LightGBM model: {mae_lgb}")

# Compare with Random Forest model
mae_rf = mean_absolute_error(y_test, rf_1_test_preds)
print(f"Mean Absolute Error of Random Forest model: {mae_rf}")
```

```
Mean Absolute Error with Ridge Regression as meta-model: 18.203096183577152
Mean Absolute Error of LightGBM model: 18.154759848817246
Mean Absolute Error of Random Forest model: 29.958215438902055
```

Figure 15: Hybrid model_2

5.3 Gradient Boosting as meta model

```
# # Train a new meta-model
from sklearn.ensemble import GradientBoostingRegressor

meta_model_gbr = GradientBoostingRegressor(n_estimators=10, learning_rate=0.1, random_state=42)
meta_model_gbr.fit(stack_train, y_train)
stacked_preds_gbr = meta_model_gbr.predict(stack_test)
```

```
# Evaluate the performance
mae_gradient = mean_absolute_error(y_test, stacked_preds_gbr)
print(f"Mean Absolute Error with Gradient Boosting as meta-model: {mae_gradient}")

# Compare with LightGBM model
mae_lgb = mean_absolute_error(y_test, lgb_test_preds)
print(f"Mean Absolute Error of LightGBM model: {mae_lgb}")

# Compare with Random Forest model
mae_rf = mean_absolute_error(y_test, rf_1_test_preds)
print(f"Mean Absolute Error of Random Forest model: {mae_rf}")
```

```
Mean Absolute Error with Gradient Boosting as meta-model: 114.9167329759491
Mean Absolute Error of LightGBM model: 18.154759848817246
Mean Absolute Error of Random Forest model: 29.958215438902055
```

Figure 16: Hybrid model_3