Configuration Manual

MSc Research Project
Data Analytics

Yugandhar Reddy Bana
Student ID: x22226991

School of Computing
National College of Ireland

Supervisor: Vladimir Milosavljevic

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Yugandhar Reddy Bana
Student ID:	X22226991
Programme:	Data Analytics
Year:	2024
Module:	MSc Research Project
Supervisor:	Vladimir Milosavljevic
Submission Due Date:	12/12/2024
Project Title:	Configuration Manual
Word Count:	1386
Page Count:	9

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: Yugandhar Reddy Bana

Date: 12/12/2024

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

Yugandhar Reddy Bana
X22226991

1 Introduction

This tutorial walks through all the steps involved in initializing and running a machine learning project on Google Colab, including hardware and software requirements, setting up the environment, integrating the Mapbox API, and a line-by-line explanation of the code that covers data preprocessing, model training and evaluation, and a few example predictions. This is with the view to helping the user successfully reproduce the workflow and obtain good predictive performance using the provided dataset and models.

2 Hardware Requirement

The project was implemented using a Windows 64 operating system with 16 GB of RAM. The details of the system specification have been highlighted in Figure 1. For this project, very high specifications are not necessary. A processor lower than an i7 would also work pretty much fine.

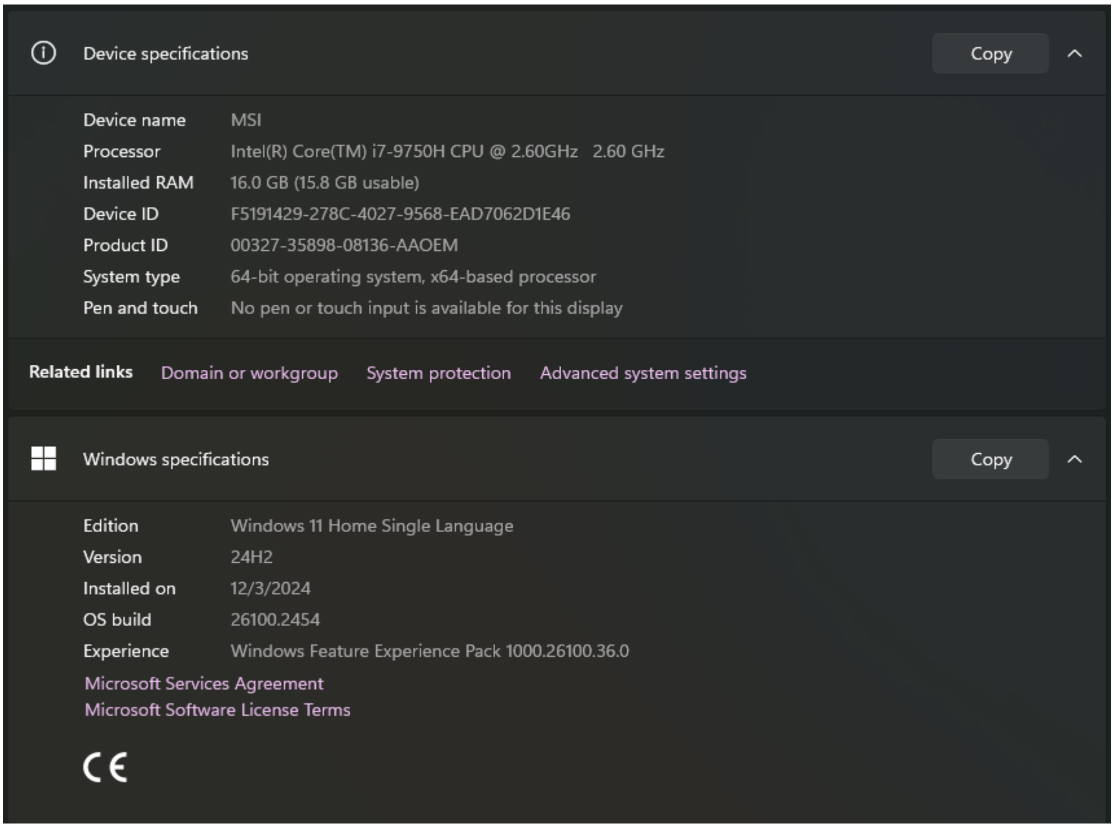

Figure 1: Hardware Configuration

3 Software Requirement

To complete this project, there are certain requirements for software that are needed: First, the use of a valid Google Account should be available to log into Google Drive and Colab. Google Colab is to be used as an IDE for the execution of the Jupyter Notebook. All necessary Python libraries have to be installed - pandas, numpy, scikit-learn, xgboost, lightgbm, matplotlib, and seaborn at a minimum. These libraries are fundamental to data manipulation, development of machine learning models, and visualization. Finally, one needs an updated internet browser recently updated version of Google Chrome or Mozilla Firefox-to access Google Colab and the interface of the Mapbox API. Ensure that all the software is updated for better functionality.

4 Environment Setup

The initial step in setting up the environment is to ensure proper organization of your files in Google Drive. Begin by creating a folder structure as follows:

4.1 Organizing Google Drive

- Navigate to the Google Drive.
- Sign in with the Google Account.
- Create a folder named Colab Notebooks (if it doesn't exists already).
- Inside the Colab Notebooks folder, create another folder called Research. This folder will have all the project related files, including the python code and the datasets required for the project. Place all the files accordingly into the Research folder.

4.2 Setting Up Google Colab

- Access Google Colab.
- Log in with your Google Account.
- Mount the Google Drive, run the following code at the beginning of the notebook to authenticate and grant access to the Google Drive.
- Inside the Colab Notebooks folder, create another folder called Research. This folder will have all the project related files, including the python code and the datasets required for the project.
- Update file paths in the notebook to point to the files inside to the Research Folder. For example:

```
[18] from google.colab import drive
     drive.mount('/content/drive')
```

Figure 2: Connecting to Google Drive

```
[26] # Reading the House Sold Dataset
     house_sold_data = pd.read_csv('/content/drive/My Drive/Colab Notebooks/Research/HouseSold.csv',encoding='unicode_escape')
```

Figure 3: File path to access to the dataset files

5 Mapbox API Configuration

The Mapbox API is a powerful tool that could be used to integrate geospatial data into your project. It offers geocoding, the process of converting addresses into coordinates; visualization of geographic data; and mapping. This project will use the Mapbox API to handle geocoding, which is necessary to transform address information into latitude and longitude coordinates for analysis. To get started with the Mapbox API, follow these steps:

5.1 Creating a Mapbox Account

- Visit Mapbox and click on Get Started For Free to Sign Up and create an account.
- Provide your email and create a password, or sign up using Google.

5.2 Generating an API Access Token

- Log in to your created Mapbox account.
- Navigate to the Token section under the Admin Section as shown in the Figure 4.
- Click "Create a Token" to generate the new token.
- Assign a meaningful name to your token.
- Set appropriate permissions for the token, for geocoding please ensure to select **SCOPES:LIST** under the Secret Scopes section.
- Click on Create Token and save the token securely as this token will be used in the code.
- We will get the free access for 100k records and after that there will be charges accordingly.

5.2 Integrate the API token into the project

- In the Colab Notebook replace the mapbox_api_token with the actual token that is being copied and stored while creating the token.

6 Code Walkthrough

The code provided in the project has several key steps that contribute to the overall functionality. Below is a detailed walkthrough of each major component:

- Importing necessary libraries is the first step to ensure all tools required for data manipulation, visualization, modelling and evaluation.

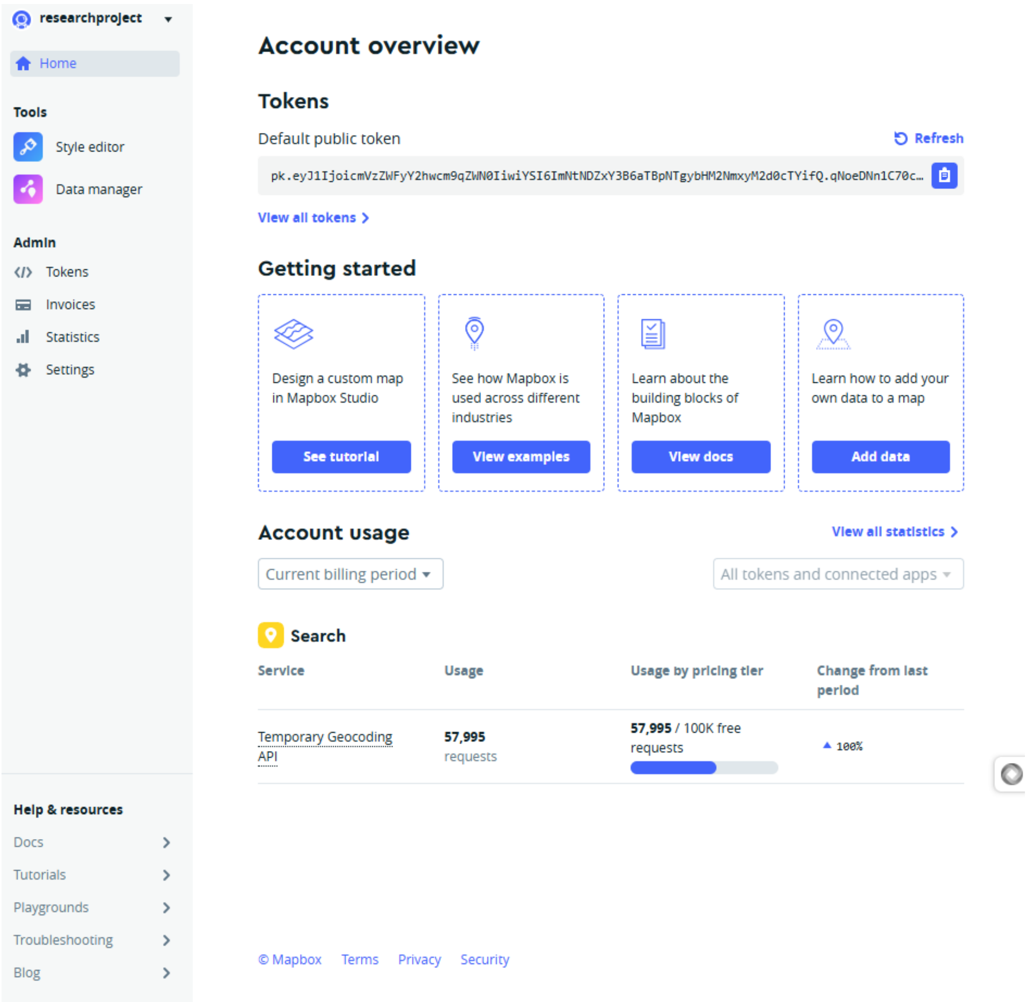

Figure 4: Interface of the Mapbox after login

Figure 5: MapBox API Token

- The datasets are loaded using pandas, and initial preprocessing is performed to handle missing values, outliers, and irrelevant columns. New features are engineered if necessary and all the three datasets have been merged accordingly.
- The data is divided into training and testing in 80-20 percent where 80% of the data is used for the testing and 20% of the data is used for the testing. This ensures the model can be evaluated on unseen data.
- Multiple machine learning models using the RandomizedSearchCV for hyperparamter optimization for training purpose.

7 Model Preparation and Evaluation

This section describes how machine learning models are prepared, trained, tuned, and evaluated using the dataset. Each model undergoes hyperparameter tuning, training, and evaluation using key metrics: R^2 Score, Mean Squared Error (MSE), and Mean Absolute Error (MAE).

```
# Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.feature_selection import RFE
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
import xgboost as xgb
import lightgbm as lgb
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import requests
import time
import folium
import warnings

# Ignore all warnings
warnings.filterwarnings('ignore')
```

Figure 6: Importing the Required Libraries

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6701 entries, 0 to 6700
Data columns (total 17 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Date of Sale (dd/mm/yyyy) 6701 non-null   object
 1   Address                   6701 non-null   object
 2   County                    6701 non-null   object
 3   Price                     6701 non-null   float64
 4   Not Full Market Price     6701 non-null   object
 5   VAT Exclusive             6701 non-null   object
 6   Description of Property   6701 non-null   object
 7   Sale_Year                 6701 non-null   int64
 8   Sale_Month                6701 non-null   int64
 9   Season                    6701 non-null   object
 10  Price_Level               6701 non-null   object
 11  Quarter                   6701 non-null   object
 12  longitude                 6701 non-null   float64
 13  latitude                  6701 non-null   float64
 14  eircode                   6701 non-null   object
 15  interest_rate             6701 non-null   float64
 16  Seasonal_Value            6701 non-null   float64
dtypes: float64(5), int64(2), object(10)
memory usage: 890.1+ KB
None
```

Figure 7: Final Data after Data Preprocessing and Data Cleaning

7.1 Random Forest Regressor

The results of the random forest regressor indicate a suitable model with the best overparameters determined by overparameterization: n_estimators = 500, min_samples_split = 2, min_samples_leaf = 1, max_features = 'log2'. and max_ allowance = no model 0.88 scores R^2, which means it explains approximately 88% of the target variable's variance. The MAE (Mean Absolute Error) of 28752 shows the average magnitude of the error. The sample prediction shows that the model is close to the original value; For example, the actual value of 426,872 was predicted to be 429,093. These results show that the random forest model provides accurate predictions with minimal variance. This makes it a reliable choice for dataset.

```python
# Random Forest Parameters
random_forest_params = {
    'n_estimators': [500],
    'max_features': ['log2'],
    'max_depth': [None],
    'min_samples_split': [2],
    'min_samples_leaf': [1]
}

# RandomizedSearchCV for Random Forest
rf_random_search = RandomizedSearchCV(
    estimator=RandomForestRegressor(random_state=42),
    param_distributions=random_forest_params,
    n_iter=50,
    cv=5,
    verbose=0,
    random_state=42,
    n_jobs=-1
)
rf_random_search.fit(X_train, y_train)
rf_best_model = rf_random_search.best_estimator_
rf_preds = rf_best_model.predict(X_test)

# Evaluate Random Forest
rf_r2 = r2_score(y_test, rf_preds)
rf_mse = mean_squared_error(y_test, rf_preds)
rf_mae = mean_absolute_error(y_test, rf_preds)
print(f"\nRandom Forest Results:\nBest Parameters: {rf_random_search.best_params_}")
print(f"R2 Score: {rf_r2:.4f} | Mean Squared Error: {rf_mse:.4f} | Mean Absolute Error: {rf_mae:.4f}")

# Display Sample Predictions
display_sample_predictions(y_test.reset_index(drop=True), rf_preds)
```

Figure 8: Random Forest Implementation

7.2 Gradient Boosting Regressor

Gradient Boosting Regressor performs well with best hyperparameters specified as n_estimators = 100, min_samples_split = 5, min_samples_leaf = 1, max_ allowance = 7 and Learning_rate = 0.1. The model received an R^2 score of 0.87, explaining 87% of the variance. In the target indicator variables MSE $2.1*10^{10}$ and Mean Absolute Error 30644. Reveal the size of the error Example predictions show that the model effectively captures the true value trend, such as predicting 440528 from a base value of 446940, although the performance is strong. But the measurement error is slightly higher compared to other models. Some models indicate that there may be room for further optimization.

```
[15] # Gradient Boosting Parameters
     gbr_params = {
         'n_estimators': [100],
         'learning_rate': [0.1],
         'max_depth': [7],
         'min_samples_split': [5],
         'min_samples_leaf': [1]
     }

     # RandomizedSearchCV for Gradient Boosting
     gbr_random_search = RandomizedSearchCV(
         estimator=GradientBoostingRegressor(random_state=42),
         param_distributions=gbr_params,
         n_iter=50,
         cv=5,
         verbose=0,
         random_state=42,
         n_jobs=-1
     )
     gbr_random_search.fit(X_train, y_train)
     gbr_best_model = gbr_random_search.best_estimator_
     gbr_preds = gbr_best_model.predict(X_test)

     # Evaluate Gradient Boosting
     gbr_r2 = r2_score(y_test, gbr_preds)
     gbr_mse = mean_squared_error(y_test, gbr_preds)
     gbr_mae = mean_absolute_error(y_test, gbr_preds)
     print(f"\nGradient Boosting Results:\nBest Parameters: {gbr_random_search.best_params_}")
     print(f"R2 Score: {gbr_r2:.4f} | Mean Squared Error: {gbr_mse:.4f} | Mean Absolute Error: {gbr_mae:.4f}")

     # Display Sample Predictions
     display_sample_predictions(y_test.reset_index(drop=True), gbr_preds)
```

Figure 9: Gradient Boosting Regressor

7.3 XGBoost Regerssor

XGBoost Regressor achieves excellent results with the best hyperparameters set to subsample = 1.0, n_estimators = 100, max_ledge = 10, Learning_rate = 0.05 and colsample_bytree = 1.0, with a maximum R^2 score of 0.87, which explains 87% of the variance in the target variable. The MSE of $2.1*10^{10}$ is nearly equivalent to the Gradient Boosting regressor and the MAE of 29,402 reflects the low error rate. An example prediction of 514,449 from the original value of 567,844 shows the accuracy of the model.

7.4 Decision Tree Regressor

The decision tree regressor achieves moderate performance with the best outlier parameters. min_samples_split = 10, min_samples_leaf = 4, and max_deep = 20. The model's R^2 score is 0.83, indicating that the target variable explains 83% of the variance. A sample forecast, such as 440,528 with the original value of 447,874, represents a reasonable but less accurate forecast. This model may not be as accurate as other models. But it can also be used as a basis for comparison.

7.5 LightGBM Regressor

LightGBM Regressor demonstrates excellent performance with appropriate hyperparameters: num_leaves = 50, n_estimators = 300, min_data_in_leaf = 10, max_ledge = 30, and learning_rate = 0.05, which explains 87% of the variance of the target variables. At MSE $2.2*10^{10}$ and MAE 31176 indicate competitive accuracy. Example predictions such as 514,449 compared to the original value of 574,712 confirm the reliability of the LightGBM model as an efficient and accurate choice. Ideal for large data sets.

```
# XGBoost Parameters
xgb_params = {
    'n_estimators': [100],
    'max_depth': [ 10],
    'learning_rate': [ 0.05],
    'subsample': [1.0],
    'colsample_bytree': [ 1.0]
}

# RandomizedSearchCV for XGBoost
xgb_random_search = RandomizedSearchCV(
    estimator=xgb.XGBRegressor(random_state=42),
    param_distributions=xgb_params,
    n_iter=50,
    cv=5,
    verbose=0,
    random_state=42,
    n_jobs=-1
)
xgb_random_search.fit(X_train, y_train)
xgb_best_model = xgb_random_search.best_estimator_
xgb_preds = xgb_best_model.predict(X_test)

# Evaluate XGBoost
xgb_r2 = r2_score(y_test, xgb_preds)
xgb_mse = mean_squared_error(y_test, xgb_preds)
xgb_mae = mean_absolute_error(y_test, xgb_preds)
print(f"\nXGBoost Results:\nBest Parameters: {xgb_random_search.best_params_}")
print(f"R2 Score: {xgb_r2:.4f} | Mean Squared Error: {xgb_mse:.4f} | Mean Absolute Error: {xgb_mae:.4f}")

# Display Sample Predictions
display_sample_predictions(y_test.reset_index(drop=True), xgb_preds)
```

Figure 10: XGBoost Regressor

```
[12] # Decision Tree Parameters
     dt_params = {
         'max_depth': [20],
         'min_samples_split': [10],
         'min_samples_leaf': [4]
     }

     # RandomizedSearchCV for Decision Tree
     dt_random_search = RandomizedSearchCV(
         estimator=DecisionTreeRegressor(random_state=42),
         param_distributions=dt_params,
         n_iter=50,
         cv=5,
         verbose=0,
         random_state=42,
         n_jobs=-1
     )
     dt_random_search.fit(X_train, y_train)
     dt_best_model = dt_random_search.best_estimator_
     dt_preds = dt_best_model.predict(X_test)

     # Evaluate Decision Tree
     dt_r2 = r2_score(y_test, dt_preds)
     dt_mse = mean_squared_error(y_test, dt_preds)
     dt_mae = mean_absolute_error(y_test, dt_preds)
     print(f"\nDecision Tree Results:\nBest Parameters: {dt_random_search.best_params_}")
     print(f"R2 Score: {dt_r2:.4f} | Mean Squared Error: {dt_mse:.4f} | Mean Absolute Error: {dt_mae:.4f}")

     # Display Sample Predictions
     display_sample_predictions(y_test.reset_index(drop=True), dt_preds)
```

Figure 11: Decision Tree Regressor

```
[17] # LightGBM Parameters
     lgb_params = {
         'n_estimators': [300],
         'max_depth': [30],
         'learning_rate': [ 0.05],
         'num_leaves': [50],
         'min_data_in_leaf': [10]
     }

     # RandomizedSearchCV for LightGBM
     lgb_random_search = RandomizedSearchCV(
         estimator=lgb.LGBMRegressor(random_state=42),
         param_distributions=lgb_params,
         n_iter=50,
         cv=5,
         verbose=0,
         random_state=42,
         n_jobs=-1
     )
     lgb_random_search.fit(X_train, y_train)
     lgb_best_model = lgb_random_search.best_estimator_
     lgb_preds = lgb_best_model.predict(X_test)

     # Evaluate LightGBM
     lgb_r2 = r2_score(y_test, lgb_preds)
     lgb_mse = mean_squared_error(y_test, lgb_preds)
     lgb_mae = mean_absolute_error(y_test, lgb_preds)
     print(f"\nLightGBM Results:\nBest Parameters: {lgb_random_search.best_params_}")
     print(f"R2 Score: {lgb_r2:.4f} | Mean Squared Error: {lgb_mse:.4f} | Mean Absolute Error: {lgb_mae:.4f}")

     # Display Sample Predictions
     display_sample_predictions(y_test.reset_index(drop=True), lgb_preds)
```

Figure 12: LightGBM Regressor