National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Mohamed Mubassir Hussain Hidayat Hussain
Student ID: X21227471

School of Computing
National College of Ireland

Supervisor: Abdul Shahid

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Mohamed Mubassir Hussain Hidayat Hussain
Student ID:	X21227471
Programme:	Data Analytics
Year:	2023
Module:	MSc Research Project
Supervisor:	Abdul Shahid
Submission Due Date:	31/01/2024
Project Title:	Configuration Manual
Word Count:	672
Page Count:	7

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:	Mohamed Mubassir Hussain Hidayat Hussain
Date:	31st January 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

Mohamed Mubassir Hussain Hidayat Hussain
X21227471

1 Introduction

This research project's Configuration manual provides all the information about the system used to perform all the tasks and the environment required to perform and execute the research project consisting of so many programming sections such as Pre- Processing, Data Integration and development as well as evaluation of machine learning models.

2 System Configuration

In this section we see the Operating System software as well as Hardware Configuration used to successfully execute the research project

2.1 Operating System

For this research project I have carried out this entire process in the Operating system Mac Operating system Ventura Version 13.1.

2.2 Hardware Configuration

Here for this research project the particular system with configurations used is shown in the below Table 1.

Name	Description
System	Macbook Pro
Storage	512 GB SSD
Processor	M1
RAM	16 GB

Table 1: Hardware Configurations

3 Environment

For this research project I have used Python as for development of this research project because of the fact that Python is very user friendly, easy to run as well as understand and also it has an extensive library which is very helpful. Below let's see the environment we have used for development and the usage of various libraries.

3.1 Jupyter Notebook

Jupyter Notebook is very easy to use offers a lot of features and moreover it serves as a one single where we could compute all our Data visualisations, Coding process and so on. The best feature here in Jupyter notebook is that it is a cross platform supported and has various programming languages support. The below Fig 1 shows an overview of Jupyter notebook Platform. [1]

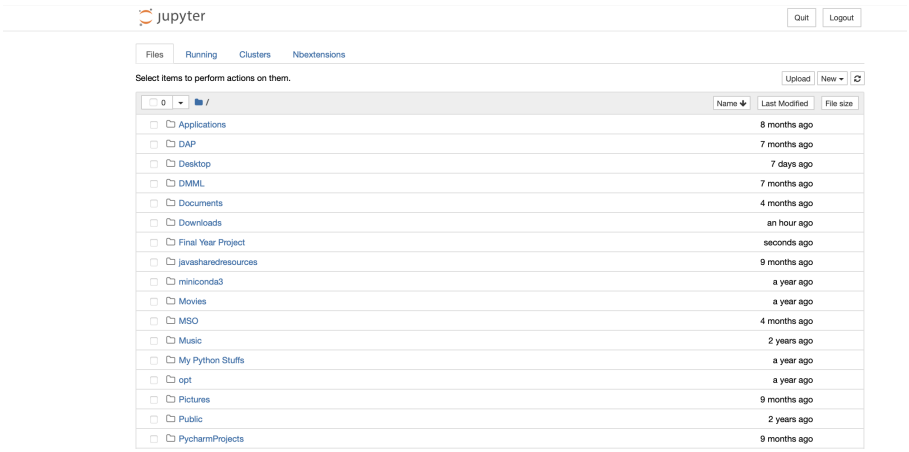

Figure 1: Jupyter Notebook

3.2 Python Libraries

Python has many libraries which are also called as Functions. Each library has its own use like to carry out tasks such as Data visualisation, Data modelling, Prediction and so on. Therefore for a particular task the relevant library was used to execute the research. As this research project was completely carried on using Jupyter Notebook it became a bit easy to use and install loads of python libraries which were essential for this research. The below Table 2 shows some of the libraries used and their versions for this research project.

Library	Version
Pandas	2.0.0
numpy	1.23.5
matplotlib	3.6.2
seaborn	0.12.2
tensorflow	2.15.0
scikit-learn	1.2.2
missingno	0.5.2
geopandas	0.14.0
surprise	1.1.3

Table 2: Python Libraries

[1] https://jupyter.org/

4 Data Source

For this research project a total of four data-sets have been used namely Attractions, Activities, Accommodations and Weather data-set. The first three data-sets have been fetched from the Failte Ireland Open Data platform which contains tourist related information's in the form of a CSV file. Then the weather data-set which contains an hourly weather data from 1989 to 2018 of various stations in the Ireland, It is obtained from Kaggle resource which shows that it has originally been sourced from Met Eireann an Irish National Meteorological Service provider. The below Fig 2 shows the structure of all the data-sets. [2] [3]

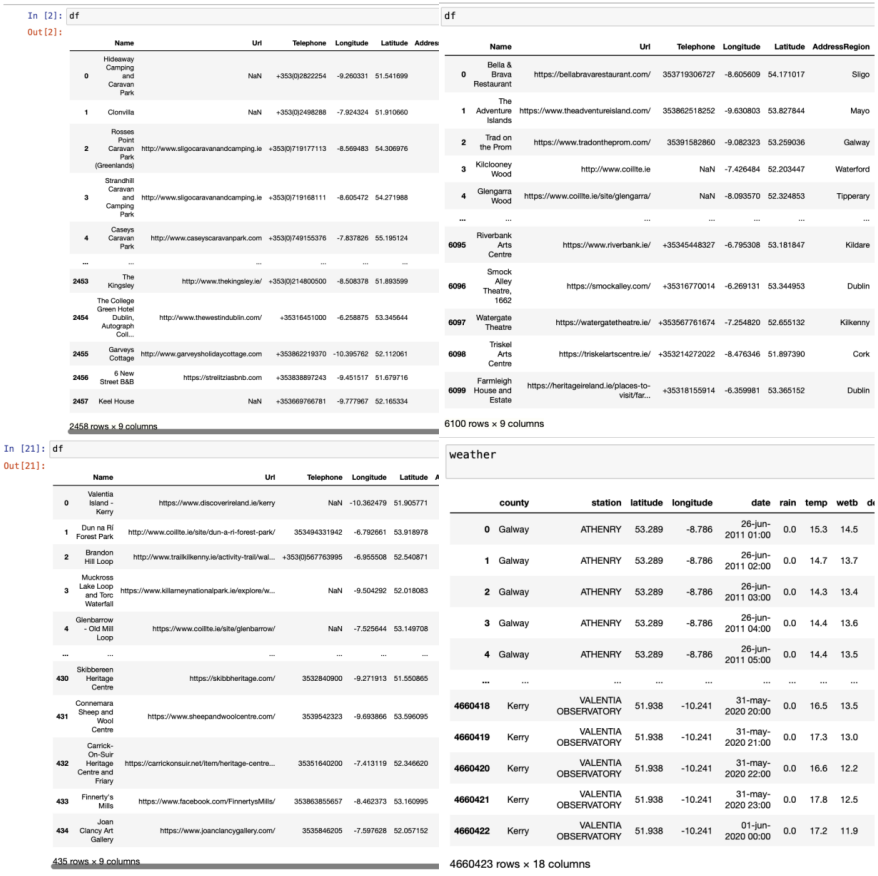

Figure 2: Data-set Overview

5 Code Implementation

In this section we will see the code implementation for both the Recommendation systems as well as the weather prediction model.

[2] https://www.failteireland.ie/Research-Insights/Open-data.aspx
[3] https://www.kaggle.com/datasets/conorrot/irish-weather-hourly-data/data

5.1 Implementation Of Recommendation systems and Weather Prediction Models

Performing Data-set Merging

Merging DataFrames:

```
In [41]: merged_df = pd.concat([accommodations, attractions, activities], ignore_index=True)

         # Save the merged dataset to a new CSV file
         merged_df.to_csv('merged_dataset.csv', index=False)

In [42]: merged_df
```

Out[42]:

	Name	Type	Country	County	Province	Town	Longitude	Latitude	Telephone	URL
0	Hideaway Camping and Caravan Park	Camping	Ireland	Cork	Munster	Skibbereen	-9.260331	51.541699	3532822254.0	NaN
1	Clonvilla	Camping	Ireland	Cork	Munster	Youghal	-7.924324	51.910660	3532498288.0	NaN
2	Rosses Point Caravan Park (Greenlands)	Camping	Ireland	Sligo	Connacht	Rosses Point	-8.569483	54.306976	353719177113.0	http://www.sligocaravanandcamping.ie
3	Strandhill Caravan and Camping Park	Camping	Ireland	Sligo	Connacht	Airport Road	-8.605472	54.271988	353719168111.0	http://www.sligocaravanandcamping.ie
4	Caseys Caravan Park	Camping	Ireland	Donegal	Ulster	Downings	-7.837826	55.195124	353749155376.0	http://www.caseyscaravanpark.com
...	...	...	...	...	...	...	...	...	...	...
8986	Riverbank Arts Centre	Learning	Ireland	Kildare	Leinster	Newbridge	-6.795308	53.181847	35345448327	https://www.riverbank.ie/
8987	Smock Alley Theatre, 1662	NaN	Ireland	Dublin	Leinster	Dublin City	-6.269131	53.344953	35316770014	https://smockalley.com/
8988	Watergate Theatre	Venue	Ireland	Kilkenny	Leinster	Kilkenny City	-7.254820	52.655132	353567761674	https://watergatetheatre.ie/
8989	Triskel Arts Centre	Craft	Ireland	Cork	Munster	Cork City	-8.476346	51.897390	353214272022	https://triskelartscentre.ie/
8990	Farmleigh House and Estate	Historic Houses and Castle	Ireland	Dublin	Leinster	Castleknock	-6.359981	53.365152	35318155914	https://heritageireland.ie/places-to-visit/far...

8991 rows × 10 columns

Figure 3: Data-set Merging

With the help of Geo pandas joining the data-sets.

Geospatial Matching and Merging:

Converts latitude and longitude columns in 'merged_df' and 'weather' DataFrames to GeoDataFrames, performs a nearest-neighbor search, merges based on the nearest points, and saves the result as 'result.csv'.

```
In [55]: import pandas as pd
         import geopandas as gpd
         from scipy.spatial import cKDTree

         # Assuming merged_df and weather are your DataFrames

         # Convert latitude and longitude columns to a GeoDataFrame
         gdf_merged = gpd.GeoDataFrame(
             merged_df,
             geometry=gpd.points_from_xy(merged_df['Longitude'], merged_df['Latitude']),
             crs='EPSG:4326',  # Assuming WGS84 coordinate system
         )

         gdf_weather = gpd.GeoDataFrame(
             weather,
             geometry=gpd.points_from_xy(weather['longitude'], weather['latitude']),
             crs='EPSG:4326',
         )

         # Create cKDTree for faster nearest-neighbor search
         tree_weather = cKDTree(gdf_weather.geometry.apply(lambda geom: (geom.x, geom.y)).tolist())

         # Find the nearest points
         gdf_merged['nearest_idx'] = gdf_merged.geometry.apply(lambda geom: tree_weather.query((geom.x, geom.y))[1])

         # Merge based on the nearest points
         result = pd.merge(gdf_merged, gdf_weather, left_on='nearest_idx', right_index=True, suffixes=('_merged', '_wea

         # Drop unnecessary columns
         result = result.drop(columns=['nearest_idx', 'geometry_weather'])

         # Save the cleaned and reordered DataFrame as "Attraction.csv"
         result.to_csv('result.csv', index=False)
```

Figure 4: Spatial Joining

Working model and Evaluation metrics of Content Based Recommendation and Regression Techniques.
A Content based recommendation model is executed in Fig 5 and then Evaluation matrix of recommendation systems in Fig 6. Now Data sampling in weather prediction data-set in Fig 7. followed by Feature selection using lasso and correlation techniques in Fig 8. Finally working model of regression techniques in Fig 9 and Fig 10.

```
In [27]: from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import linear_kernel
import pandas as pd

# Load your dataset
df = pd.read_csv('result.csv')

# Assuming df is your dataset
tfidf_vectorizer = TfidfVectorizer(stop_words='english')
df['Description'] = df['Name'] + ' ' + df['Type'] + ' ' + df['Town'] + ' ' + df['County'] + ' ' + df['season']
tfidf_matrix = tfidf_vectorizer.fit_transform(df['Description'])

# Compute the cosine similarity matrix
cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)

# Function to get recommendations based on content, county, and type
def content_recommendations(season, place_type, cosine_sim=cosine_sim):
    # Filter places based on season and type
    filtered_df = df[(df['season'] == season) & (df['Type'].isin(place_type))]

    # If there are no places matching the criteria, return an empty DataFrame
    if filtered_df.empty:
        return pd.DataFrame(columns=['Name', 'Type', 'Town', 'County', 'Longitude', 'Latitude', 'season','weat

    # Get indices of filtered places
    indices = filtered_df.index.tolist()

    # Calculate cosine similarity for filtered places
    sim_scores = cosine_sim[indices].sum(axis=0)

    # Enumerate the similarity scores
    sim_scores = list(enumerate(sim_scores))

    # Create a DataFrame to store similarity scores and corresponding indices
    similarity_df = pd.DataFrame(sim_scores, columns=['place_index', 'similarity'])

    # Merge the similarity DataFrame with the original DataFrame to get all places
    all_places = pd.merge(filtered_df, similarity_df, left_index=True, right_on='place_index', how='left')

    # Drop the temporary columns used for merging
    all_places.drop(['place_index', 'similarity'], axis=1, inplace=True)

    # Sort places by similarity if needed
    # all_places = all_places.sort_values(by='similarity', ascending=False)

    return all_places[['Name', 'Type', 'Town', 'County', 'Latitude', 'Longitude', 'season','weather_avg']]

# Example usage
season_input = "Winter"  # Replace with user input
place_type_input = ['Camping', 'B&B', 'Hotel', 'Hostel', 'Self Catering Accommodation', 'Food and Drink', 'Kay
 'Bird Watching', 'Craft', 'Activity Operator', 'Transport', 'Restaurant',
 'Swimming', 'Fishing', 'Shopping', 'Forest Park', 'Tracing Your Ancestors',
 'Sailing', 'Fitness and Leisure', 'Cooking', 'Climbing', 'Golf', 'Learning',
 'Venue', 'Pampering', 'Cruising', 'Photography', 'Movies', 'Horse Riding',
 'Agriculture', 'Race Course', 'Golf Course', 'Kitesurfing', 'Tour',
 'Museums and Attraction', 'Castle', 'Falconry', 'Island', 'Embarkation Point',
 'Natural Landscape', 'Church Abbey', 'Art Gallery', 'Public Sculpture',
 'Marina', 'Gardens', 'Literary Ireland', 'Public Park',
 'Historic Houses and Castle', 'Surfing', 'Spa', 'Zip Lining', 'Visitor Farm',
 'Gardening', 'Gaa', 'Beach', 'Ruins', 'Pitch And Putt', 'Adventure Park',
 'Zoos and Aquarium', 'Day Tour', 'Cinema', 'Food Shops', 'Fast Food',
 'Bike Rental', 'Cafe', 'B&B']

recommendations = content_recommendations(season=season_input, place_type=place_type_input)

# Save the recommendations to a CSV file
recommendations.to_csv("recommendation_data_season.csv", index=False)
```

Figure 5: Content Based Recommendation Systems

```
:  def evaluate_recommendations(actual_places, recommended_places):
       """
       Evaluate the quality of recommendations.

       Parameters:
       - actual_places: DataFrame containing the actual places the user likes or has visited.
       - recommended_places: DataFrame containing the recommended places.

       Returns:
       - Precision: The proportion of recommended places that the user actually likes.
       - Recall: The proportion of places that the user likes which were correctly recommended.
       """
       # Assuming 'Name' is a unique identifier for places
       actual_set = set(actual_places['Name'])
       recommended_set = set(recommended_places['Name'])

       # Calculate Precision
       precision = len(actual_set.intersection(recommended_set)) / len(recommended_set) if len(recommended_set) >

       # Calculate Recall
       recall = len(actual_set.intersection(recommended_set)) / len(actual_set) if len(actual_set) > 0 else 0

       return precision, recall

   # Example usage
   # Load the actual user preferences (you might need to collect this data)
   actual_preferences = pd.read_csv('result.csv')

   # Assuming 'recommendations' is the DataFrame containing your recommendations
   precision, recall = evaluate_recommendations(actual_preferences, recommendations)

   print(f'Precision: {precision:.2f}')
   print(f'Recall: {recall:.2f}')
```

Figure 6: Evaluation Metrics of Recommendation system

Data Cleaning:

Filtering and Sampling Weather Data, Dropping Unnecessary Columns

In [8]:
```
# Convert the 'date' column to a datetime object
weather['date'] = pd.to_datetime(weather['date'], format='%d-%b-%Y %H:%M')

# Filter data for 2010 to 2018
start_date = pd.to_datetime('2010-01-01')
end_date = pd.to_datetime('2018-12-31')
weather = weather[(weather['date'] >= start_date) & (weather['date'] <= end_date)]

# Calculate the number of data points per county
data_per_county = weather['county'].value_counts()

# Determine the target number of data points per county
total_target_rows = 200000
target_data_per_county = total_target_rows // len(data_per_county)

# Create a new DataFrame to store the sampled data
subset_df = pd.DataFrame()

# Sample data for each county
for county in data_per_county.index:
    count = data_per_county[county]
    if count >= target_data_per_county:
        sampled_data = weather[weather['county'] == county].sample(target_data_per_county, random_state=42)
        subset_df = pd.concat([subset_df, sampled_data])

# Reset the index of the resulting DataFrame
subset_df.reset_index(drop=True, inplace=True)

# Save the resulting subset as a CSV file
subset_df.to_csv('Cleaned Weather New.csv',index=False)
```

Figure 7: Data Filtering and Sampling of Weather Prediction Dataset

Feature Correlation Analysis for Temperature Prediction using Correlation matrix and Lasso Regression Feature Selection

In [27]:
```
# Assuming your DataFrame is named df
correlation_matrix = df_new.corr()
correlation_with_temp = correlation_matrix['temp'].sort_values(ascending=False)

# Select top N features (adjust N based on your preference)
top_features = correlation_with_temp[1:9].index.tolist()
print(top_features)
```

```
['wetb', 'dewpt', 'vappr', 'month', 'hour', 'wdsp', 'msl', 'year']
```

In [28]:
```
# Assuming 'temp' is your target variable
X = df_new[['wetb', 'dewpt', 'vappr', 'month', 'hour', 'wdsp', 'msl', 'year']]
y = df_new['temp']  # Assuming 'temp' is your target variable

# Standardize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Create a Lasso Regression model
alpha = 0.1  # You can adjust the regularization strength
model_lasso = Lasso(alpha=alpha)

# Fit the model to the scaled data
model_lasso.fit(X_scaled, y)

# Extract selected features
selected_features = X.columns[model_lasso.coef_ != 0]

print("Selected Features:", selected_features)
```

```
Selected Features: Index(['wetb', 'dewpt', 'vappr', 'hour', 'msl'], dtype='object')
```

Figure 8: Feature Correlation Analysis

Model Implementation

Multiple Linear Regression Model Training and Evaluation on Weather Data

In [30]:
```
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score

# Assuming 'temp' is your target variable
X = df_new[['wetb', 'dewpt', 'vappr', 'month','hour','wdsp','msl','year']]
y = df_new['temp']  # Assuming 'temp' is your target variable

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [31]:
```
# Create a Multiple Linear Regression model
model_mlr = LinearRegression()

# Fit the model to the training data
model_mlr.fit(X_train, y_train)
```

Out[31]:
```
LinearRegression
LinearRegression()
```

In [32]:
```
# Make predictions on the test set
y_pred_mlr = model_mlr.predict(X_test)

# Evaluate the model
mse_mlr = mean_squared_error(y_test, y_pred_mlr)
r2_mlr = r2_score(y_test, y_pred_mlr)

print(f'Mean Squared Error (Multiple Linear Regression): {mse_mlr}')
print(f'R2 Score (Multiple Linear Regression): {r2_mlr}')
```

Figure 9: Multi linear Regression Model Implementation

Random Forest Regression Model Training and Evaluation on Weather Data

:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score

# Assuming 'temp' is your target variable
X = df_new[['wetb', 'dewpt', 'vappr', 'month', 'hour','wdsp']]
y = df_new['temp']  # Assuming 'temp' is your target variable

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

:
```
# Create a Random Forest Regression model
model_rf = RandomForestRegressor(n_estimators=100, random_state=42)

# Fit the model to the training data
model_rf.fit(X_train, y_train)
```

:
```
RandomForestRegressor
RandomForestRegressor(random_state=42)
```

:
```
# Make predictions on the test set
y_pred_rf = model_rf.predict(X_test)

# Evaluate the model
mse_rf = mean_squared_error(y_test, y_pred_rf)
r2_rf = r2_score(y_test, y_pred_rf)

print(f'Mean Squared Error (Random Forest Regression): {mse_rf}')
print(f'R2 Score (Random Forest Regression): {r2_rf}')
```

Figure 10: Random Forest Model Implementation