Configuration Manual

MSc Research Project

Data Analytics

Junghyun Min

Student ID: x20103352

School of Computing

National College of Ireland

Supervisor: Dr. Catherine Mulwa

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Junghyun Min
Student ID:	X20103352
Programme:	Data Analytics **Year:** 2023
Module:	MSc Research Project
Lecturer:	Dr. Catherine Mulwa
Submission Due Date:	14/08/2023
Project Title:	Ensemble Stacking and Optimisation for Annual Revenue Prediction of Individual Airbnb Hosting: Italy
Word Count:	1323 **Page Count:** 18

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: ………

Date: 14th August 2023

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

Junghyun Min
Student ID: x20103352

1 Introduction

An implementation of the project to analyse Airbnb data in Italy and develop an annual profitability prediction model. This configuration manual encompasses System Configuration, Data Collection, Library Package Requirement, Data Preparation and Pre-processing, Model Preparation and Execution, and Evaluation.

2 System Configuration

The project was conducted on the local system with the hardware and software specifications as shown in Figure 1 and Figure 2.

2.1 Hardware Requirement

Device Specifications	
Device name	LAPTOP-M4B8V5NI
Processor	Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz 1.80 GHz
Installed RAM	8.00 GB
System type	64-bit operating system, x64-based processor
Window Specifications	
Edition	Windows 10 Home
Version	22H2
Installed on	01/10/2020
OS build	19045.3324
Experience	Windows Feature Experience Pack 1000.19041.1000.0

Figure 1: Hardware and Software Specifications

2.2 Software Requirement

```
import sys
import notebook

print("Python version: " + sys.version)
print("Jupyter notebook version: " + notebook.__version__)
```

executed in 42ms, finished 01:44:10 2023-08-13

```
Python version: 3.7.3 (default, Mar 27 2019, 17:13:21) [MSC v.1915 64 bit (AMD64)]
Jupyter notebook version: 5.7.8
```

Figure 2: Python and Notebook versions

2.2.1 Manual of Jupyter Notebook installation

1) Right-click the Anaconda installation file downloaded through the link and run it with administrator privileges.

[Free Download | Anaconda](https://www.anaconda.com/download/) (https://www.anaconda.com/download/)

2) Press the "Next" button to proceed with the installation, and tick the box when the screen below appears in Figure 3.

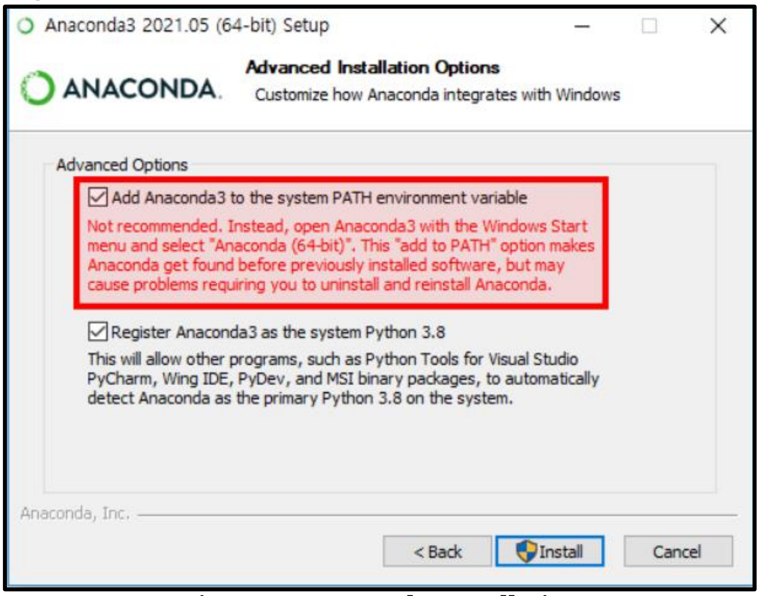

Figure 3: Anaconda Installation

3) After that, install it according to the default setting. (The path is also kept as C: drive)

4) Please make sure that both Anaconda Prompt and Jupiter Notebook are installed like Figure 4 and Figure 5 when you click the Windows icon on the taskbar.

Figure 4: Jupyter Notebook

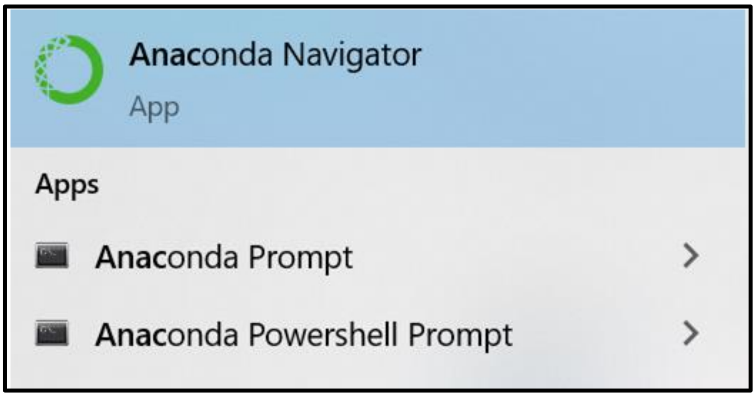

Figure 5: Anaconda Prompt and etc.

5) The installation has been successfully completed if the screen appears in the web browser as shown in Figure 6 after running the Jupiter Notebook as shown in Figure 4.

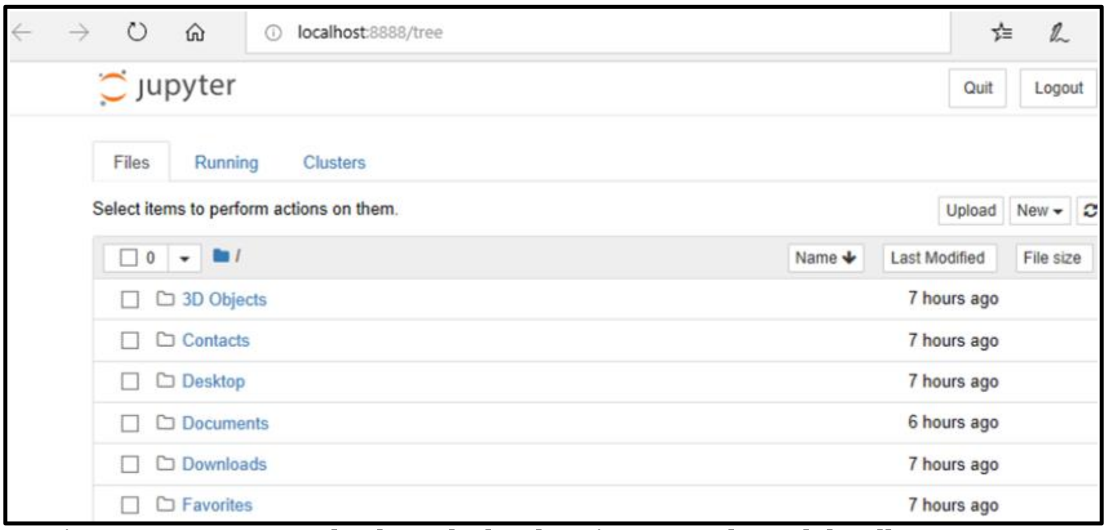

Figure 6: Jupyter Notebook on the local environment through localhost:8888/tree

3 Data Collection

The source of the original data was obtained from kaggle.com.

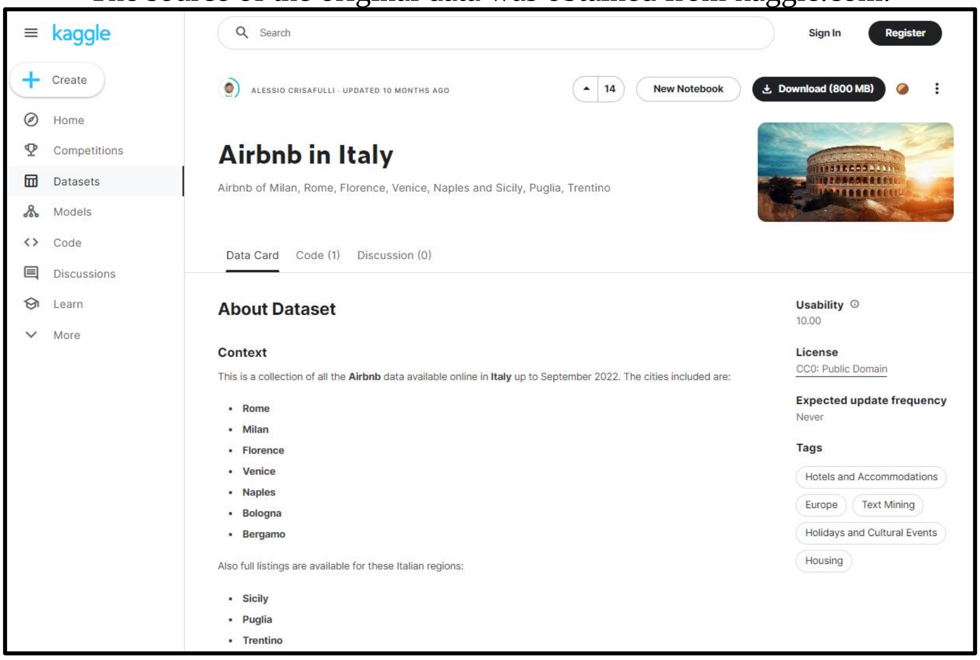

Figure 7: Data source from kaggle.com

The provided dataset through an open link is 'merged_file.csv' to start this project, which merged the data from all 10 cities as shown in Figure 8, due to the division of original data into separate 10 cities.

1. Download data on Google Drive with an open link.

2. Please click the link below and download a CSV file named 'merged_file.csv'.

https://drive.google.com/file/d/1rxWScCzr-SH2H55IC8xxijs6lXjkaPMZ/view?usp=drive_link

3. And then place the CSV file in the same folder where 1-3 Jupyter Notebook files in.

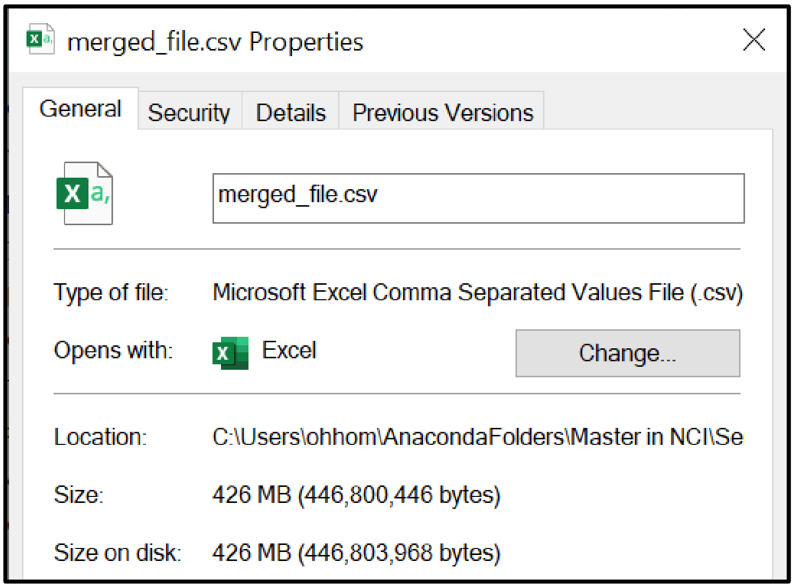

Figure 8: Data file Properties

4 Library Package Requirement

Like Figure 9, essential libraries must be imported before running all the other cells. If some libraries or packages have never been installed before, these installations also should be preceded with !pip command lines as shown in Figure 10.

```
import re
import math
from datetime import timedelta
# import random
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import missingno as msno
%matplotlib inline
from sklearn.preprocessing import LabelEncoder
# setting to see all the columns
pd.set_option('display.max_columns', None)
```

Figure 9: Import libraries for the first running Notebook file

```
## Unplock this cell If you need to install 'scikit-optimize' and 'deap'
# !pip install scikit-optimize
# !pip install deap
```

Figure 10: install scikit-optimize and deap for optimisation

Figure 9 indicates a screenshot of essential libraries to be installed for the '1.Preprocessing.ipynb' file, Whereas Figure 11 is for the '2.Modelling.ipynb' file.

```python
import math
import random
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from collections import Counter, OrderedDict

import warnings
warnings.filterwarnings("ignore", category=RuntimeWarning, message="divide by zero encountered in true_divide|invalid value encountered
pd.set_option('display.max_columns', None)
%matplotlib inline

# sklearn
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import mean_squared_error, r2_score

# Ensemble
import xgboost as xgb
import lightgbm as lgb
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor

# Optimisations
from skopt import BayesSearchCV
from deap import base, creator, tools, algorithms
```

Figure 11: Import libraries for the Second running Notebook file

5 Data Preparation and Pre-processing

```python
# read the file
df = pd.read_csv('merged_file.csv')
```

executed in 15.5s, finished 17:37:15 2023-08-13

```
C:₩ProgramData₩Anaconda3₩lib₩site-packages₩IPython₩core₩interactiveshell.py:3049: DtypeWarning: Columns (29) have mixed
types. Specify dtype option on import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

```python
# df # 181956 rows × 76 columns
```

executed in 11ms, finished 17:37:15 2023-08-13

```python
df.head(3)
```

executed in 203ms, finished 17:41:30 2023-08-13

	id	listing_url	scrape_id	last_scraped	source	name	description	neighborhood_overview	
0	15526	https://www.airbnb.com/rooms/15526	20220926182448	2022-09-26	previous scrape	Residenza PALAZZO lake view	Ideally located on the middle ridge of a panor...	One of the highlights of lake Iseo is Montisol...	https://a0.muscac
1	15542	https://www.airbnb.com/rooms/15542	20220926182448	2022-09-26	previous scrape	Suite PANORAMA facing the lake	Ideally located on the middle ridge of a panor...	One of the highlights of lake Iseo is Montisol...	https://a0.muscac

Figure 12: Data Import and Explore Dataframe

Import data as a dataframe in pandas, Figure 12, and explore the dataset such as the shape of df, data types, characteristics of features, and statistical analysis with describe() function as indicated in Figure 13.

```
df.describe()
```

executed in 811ms, finished 15:58:25 2023-08-13

	id	scrape_id	host_id	host_listings_count	host_total_listings_count	latitude	longitude	accommodates
count	1.819560e+05	1.819560e+05	1.819560e+05	181870.000000	181870.000000	181956.000000	181956.000000	181956.000000
mean	1.214052e+17	2.022092e+13	1.479840e+08	40.635954	65.244906	41.146021	13.692193	4.096677
std	2.503712e+17	6.533453e+06	1.467685e+08	169.450233	304.382593	2.986312	2.730557	2.215784
min	2.737000e+03	2.022091e+13	1.822000e+03	1.000000	1.000000	35.494150	9.025800	0.000000
25%	1.793897e+07	2.022091e+13	2.317709e+07	1.000000	1.000000	38.114800	12.230570	2.000000
50%	3.496080e+07	2.022092e+13	9.364056e+07	3.000000	3.000000	40.845370	13.271040	4.000000
75%	5.129850e+07	2.022093e+13	2.446785e+08	6.000000	7.000000	43.774370	15.288793	5.000000
max	7.257708e+17	2.022093e+13	4.809728e+08	1713.000000	20000.000000	46.524850	18.500770	16.000000

```
# object type is 37 that should be converted to neumeric for a regression model.
df.dtypes.value_counts()
```

executed in 29ms, finished 15:58:28 2023-08-13

```
object     37
float64    22
int64      17
dtype: int64
```

Figure 13: Statistical Analysis

Visualising null values can help to understand them easily and at a glance. Sorted feature with many null values in descending order and then use 'missingno' by the line of codes 'msno.bar()' function as described in Figure 14.

3.2 Visualisation of null values

```
# sorting columns in descending order
sorted_cols = df.isnull().sum().sort_values(ascending=False).index
```

executed in 598ms, finished 16:37:43 2023-08-13

```
# values as a bar chart
sorted_cols = df.isnull().sum().sort_values(ascending=False).index
msno.bar(df[sorted_cols], color='#20242c')
# Visualize the number of missing
plt.show()
```

Figure 14: Visualise Null Values using missingno tool

In Figure 15, 'yyyy-mm-dd' data type should be changed to 'to_datetime' for readable by pandas. Drop rows if they have NA in the 'host_since' column. Because one criterion that determines the size of the dataset is the 'host_since' feature. In the absence of the 'host_since' column, the measurement of hosting tenure and the calculation of profitability become challenging.

4.1 date format :'to_datetime' for readable by pandas

```
df['host_since'] = pd.to_datetime(df['host_since'])
df['last_scraped'] = pd.to_datetime(df['calendar_last_scraped'])
df['calendar_last_scraped'] = pd.to_datetime(df['calendar_last_scraped'])
df['first_review'] = pd.to_datetime(df['calendar_last_scraped'])
df['last_review'] = pd.to_datetime(df['calendar_last_scraped'])
```

executed in 776ms, finished 16:38:10 2023-08-13

4.2 Drop if NA in 'host_since' column

```
filtered_df = df.dropna(subset=['host_since'])
filtered_df
```

Figure 15: preprocessing of .to_datetime() and .dropna()

If numeric data contains special symbols such as $23.00 or 67%, these symbols need to be removed. Looking at maximum, mean and median statistical metrics, the presence of outliers and skewed distribution is observed in the column of 'price'. As 'price' is one of the key factors to create a 'profitability' column, this means that normalisation or similar work on 'profitability' should be applied(The last cell in Figure 16).

4.3 Removal of '$' symbols

```
filtered_df['price'] = filtered_df['price'].str.replace('$', '').str.replace(',', '').astype(float)
```

executed in 15ms, finished 18:08:34 2023-08-13

4.4 Removal of '%' symbols

```
filtered_df['host_response_rate'] = filtered_df['host_response_rate'].str.replace('%', '').str.replace(',', '').astype(float)
filtered_df['host_acceptance_rate'] = filtered_df['host_acceptance_rate'].str.replace('%', '').str.replace(',', '').astype(float)
```

executed in 19ms, finished 18:08:39 2023-08-13

```
filtered_df['price'].describe()
# The output indicates we will need to normalisation or similar work on 'profitability' column which would be created
```

executed in 29ms, finished 16:38:21 2023-08-13

```
count    181870.000000
mean        149.058680
std         690.484645
min           0.000000
25%          60.000000
50%          89.000000
75%         140.000000
max      100000.000000
Name: price, dtype: float64
```

[Figure 16: Remove $ and % symbols]

Incorrect listing counts should be modified in Figure 17. In instances where the same host has posted multiple hostings, the values of 'host_listings_count' and 'host_total_listings_count' are duplicated. Therefore, rectification of this issue is necessary. For instance in a dormitory room, if a host posts a listing for each bed in order to accommodate guests to full capacity, the hosting count for each bed is equivalently assigned as a product of the bed count.

```
selected_columns = ['last_scraped', 'host_since', 'host_listings_count', 'host_total_listings_count']
# Select the max row where host_total_listings_count has max
filtered_rows = df[df['host_total_listings_count'] == df['host_total_listings_count'].max()]
selected_rows = filtered_rows[selected_columns]
```

```
grouped_counts = filtered_df.groupby(['host_id', 'host_since']).size().reset_index(name='count')
merged_df = pd.merge(filtered_df, grouped_counts, on=['host_id', 'host_since'])
```

executed in 3.53s, finished 16:38:26 2023-08-13

```
# Checking if the rows really have same counting numbers and duplicated
duplicated_rows = merged_df[merged_df.duplicated(subset=['host_id', 'host_since'], keep=False)]
selected_columns = ['host_id', 'host_since', 'host_listings_count', 'host_total_listings_count']

# This work is important before calculating profitability
print("Find duplicated rows of a number of hosting counts based on 'host_id' and 'host_since':")
duplicated_rows[selected_columns]
```

executed in 598ms, finished 16:38:27 2023-08-13

Find duplicated rows of a number of hosting counts based on 'host_id' and 'host_since':

	host_id	host_since	host_listings_count	host_total_listings_count
0	60754	2009-12-07	5.0	5.0
1	60754	2009-12-07	5.0	5.0
2	60754	2009-12-07	5.0	5.0
3	60754	2009-12-07	5.0	5.0
4	60754	2009-12-07	5.0	5.0

```
# calculation of division by 'count' for actual value of listings_count and total_listings_count
grouped_counts = filtered_df.groupby(['host_id', 'host_since']).size().reset_index(name='count')
merged_df = pd.merge(filtered_df, grouped_counts, on=['host_id', 'host_since'])
merged_df['listings_count_divided'] = merged_df['host_listings_count'] / merged_df['count']
merged_df['total_listings_count_divided'] = merged_df['host_total_listings_count'] / merged_df['count']
```

Figure 17: Fix host listings, 'host_listings_count' and 'host_total_listings_count'

Now we can calculate annual profitability with create column 'total_listings_count_divided' This project does not follow the formula of profitability in the business field, its own calculation, like the percentage of the annual sales, is applied instead. Taking logarithm after division due to the skewed distribution. And then, change the values into percentages because the target column represents 'Profitability' as shown in Figure 18.

```
# from datetime import timedelta
one_year = timedelta(days=365)
aaa=merged_df['total_listings_count_divided']*merged_df['price']/((merged_df['calendar_last_scraped']-merged_df['host_since'])/one_year)
loglog = np.log(aaa)
# tolist
logl_list = loglog.tolist()
logl_array = np.array(logl_list)
normalized_list = (logl_array - logl_array.min()) / (logl_array.max() - logl_array.min()) * 100
normalized_list = np.round(normalized_list, 2)
```

Figure 18: Calculate by the own formula to create ‘profitability_by_numOfYears’

```
# host_verifications => list type
data['host_verifications'] = data['host_verifications'].str.strip("[]").str.replace("'", "").str.split(",₩s*")
# Extract unique values
unique_values = set(data['host_verifications'].explode().unique())
# One-Hot Encoding => Flatten the values in  [email', 'phone', 'work_email'
for value in ['email', 'phone', 'work_email']:
    data[value] = data['host_verifications'].apply(lambda x: int(value in x) if x else 0)
```

```
data["amenities"] = data["amenities"].apply(eval)

# Get unique items & counts
unique_values = data["amenities"].explode().value_counts().index.tolist()
unique_value_counts = data["amenities"].explode().value_counts().values

# Count number of amenities and store in a df
amen = {'Amenities':unique_values,'Counts':unique_value_counts}
amen_df = pd.DataFrame(amen)

# top 100 into a list
top_100_amen = amen_df['Amenities'].head(100).tolist()

# count how many amenities of top 100 list.
data["amenities"] = data["amenities"].apply(lambda x: sum(item in top_100_amen for item in x))
```

```
mapping = {
    'within an hour': 3,
    'within a few hours': 2,
    'within a day': 1,
    'a few days or more': 0
}
data['host_response_time'] = data['host_response_time'].map(mapping).fillna(-1)
```

```
# Create a LabelEncoder object
encoder = LabelEncoder()

# encoder convert the values of property_type column
data['property_type_encoded'] = encoder.fit_transform(data['property_type'])
```

Figure 19: Encoding values

In Figure 19 and Figure 20, For the features requiring encoding, applied various encoding methods to their specific characteristics, including one-hot encoding, mapping and LabelEncoder.

```python
# Extract unique values of 'bathroom' column
unique_values = data['bathrooms_text'].unique()

data['bathroom_num'] = data['bathrooms_text'].str.extract(r'(₩d+₩.₩d+|₩d+)').astype(float)
data['bathroom_str'] = data['bathrooms_text'].apply(lambda value: re.sub(r'[^a-zA-Z₩s]', '', value).strip() if pd.notnull(value) else '')
```

executed in 1.57s, finished 16:40:10 2023-08-13

```python
for value in data['bathroom_str'].unique():
    if value.startswith('shared'):
        data.loc[data['bathroom_str'] == value, 'bathroom_type'] = 0
    elif value.startswith('private'):
        data.loc[data['bathroom_str'] == value, 'bathroom_type'] = 1
    else:
        data.loc[data['bathroom_str'] == value, 'bathroom_type'] = -1
```

Figure 20: mapping the values for -1, 0, 1 with regular expression

Drop columns and fill null values with -1 as in Figure 21.

```python
df1 = data.drop(['listing_url', 'last_scraped', 'source', 'name', 'description', 'picture_url', 'host_url',
                 'host_name', 'host_since', 'host_location', 'host_thumbnail_url', 'host_picture_url',
                 'host_neighbourhood', 'host_verifications', 'host_has_profile_pic', 'host_identity_verified',
                 'neighbourhood', 'neighbourhood_group_cleansed',
                 'property_type', 'bathrooms', 'bathrooms_text', 'calendar_updated', 'calendar_last_scraped',
                 'first_review', 'last_review', 'bathroom_str'], axis=1)
```

```python
df2 = df1.drop(['id', 'scrape_id', 'host_id', 'calculated_host_listings_count', 'price',
               'calculated_host_listings_count_entire_homes', 'calculated_host_listings_count_private_rooms',
               'calculated_host_listings_count_shared_rooms', 'count', 'host_listings_count',
               'listings_count_divided', 'total_listings_count_divided'], axis=1)
```

```python
# fillna with a value of -1
df2.fillna(-1, inplace = True)
```

```python
df3 = df2.drop(['review_scores_communication', 'review_scores_checkin', 'review_scores_accuracy',
 'review_scores_value', 'review_scores_cleanliness', 'review_scores_location', 'availability_60',
'minimum_minimum_nights', 'maximum_minimum_nights', 'minimum_maximum_nights',
'maximum_maximum_nights', 'minimum_nights_avg_ntm', 'maximum_nights_avg_ntm'], axis=1)
```

Figure 21: Drop columns and fill null values with value of -1

7.4.1 Split and Save to 10 CSV files

```python
city_list = ['bergamo', 'roma', 'milano', 'sicilia', 'trentino', 'puglia', 'firenze', 'venezia', 'napoli', 'bologna']
cities = ['bergamo', 'roma', 'milano', 'sicilia', 'trentino', 'puglia', 'firenze', 'venezia', 'napoli', 'bologna']

city_dfs = {}  # Dictionary to store DataFrames for each city

# Select rows where 'city' column matches the current city and drop the 'city' column
for city in city_list:
    city_dfs[city] = df3[df3['city'] == city].drop('city', axis=1)

for city in cities:
    globals()[city + '_df'] = city_dfs[city]
    # Save each DataFrame as a CSV file
    globals()[city + '_df'].to_csv(city + '_df.csv', index=False)
```

Figure 22: Split the dataset into 10 cities for Modelling implementation

7.4.3.2 **Density histograms for all DataFrames, overlapped in one Figure.**

Compare the relative distribution of datasets at a glance.

```python
# Create density histograms for all DataFrames
for df in [bergamo_df, roma_df, milano_df, sicilia_df, trentino_df,
                                    puglia_df, firenze_df, venezia_df, napoli_df, bologna_df]:
    df['profitability_by_numOfYears'].hist(bins=100, density=True, alpha=0.5)

plt.xlabel('Annual revenue')
plt.ylabel('Density')
plt.title("Comparison of Profitability by Number of Years in All Cities")
plt.legend(labels=[f"DF{i+1}" for i in range(10)])
plt.show()
```

executed in 9.00s, finished 16:45:15 2023-08-13

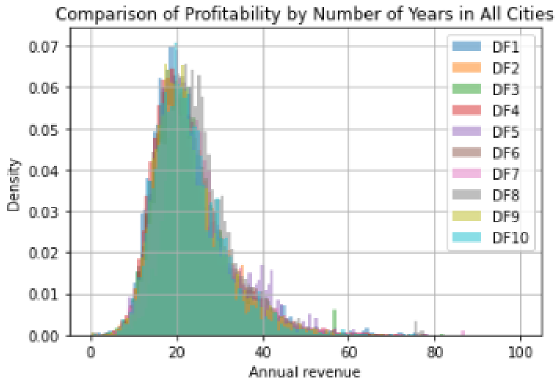

Figure 23: Distribution of "profitability_by_numOfYears" in 10 Italian city datasets

```python
import scipy.stats as stats

# one-way ANOVA
f_statistic, p_value = stats.f_oneway(bergamo_df['profitability_by_numOfYears'], roma_df['profitability_by_numOfYears'],
                                      milano_df['profitability_by_numOfYears'], sicilia_df['profitability_by_numOfYears'],
                                      trentino_df['profitability_by_numOfYears'], puglia_df['profitability_by_numOfYears'],
                                      firenze_df['profitability_by_numOfYears'], venezia_df['profitability_by_numOfYears'],
                                      napoli_df['profitability_by_numOfYears'], bologna_df['profitability_by_numOfYears'])

print("One-way ANOVA p-value:", round(p_value, 5))
```

executed in 33ms, finished 16:45:21 2023-08-13

```
One-way ANOVA p-value: 0.0
```

Figure 24: one-way ANOVA test

6 Model Preparation and Execution

6.1 Set pre-define Functions

In the second Jupyter Notebook file, "2.Modeling.ipynb," the data for 10 cities is required which was generated as a result of executing the first Jupyter Notebook file. When opening the second Jupyter Notebook, start by importing the necessary libraries. Next, pre-define functions to write code more practical and enable its straightforward utilisation.

Calculate Distance(latitude, longitude)

```
# Calulate long and lat from the center point of each city by haversine_distance()
def haversine_distance(lat1, lon1, lat2, lon2):
    # change the measure of lat and long through math.radians()
    lat1 = math.radians(lat1)
    lon1 = math.radians(lon1)
    lat2 = math.radians(lat2)
    lon2 = math.radians(lon2)

    # Distance between lons of the city center and the hosting location
    dlon = lon2 - lon1
    # Distance between lats of the city center and the hosting location
    dlat = lat2 - lat1

    # Applied Haversine Equation
    a = math.sin(dlat / 2) ** 2 + math.cos(lat1) * math.cos(lat2) * math.sin(dlon / 2) ** 2
    c = 2 * math.atan2(math.sqrt(a), math.sqrt(1 - a))
    distance = 6371 * c  # The radius(average distance to the centre of the Earth) is 6,371 km

    return distance
```

Figure 25: Distance calculator from the city centre to the hosting location

Label Encoder(neighbourhood_cleansed)

```
def nbhd_cleansed_encoder(city_df):
    # LabelEncoder class
    encoder = LabelEncoder()
    # replace into numeric in the column
    city_df['neighbourhood_cleansed_encoded'] = encoder.fit_transform(city_df['neighbourhood_cleansed'])
    # Drop the previous column
    city_df.drop('neighbourhood_cleansed', axis=1, inplace=True)
```

Figure 26: Hence the column requiring encoding only in the same city data, the values were encoded in the second Notebook file

Top 10 Important Features

- The `important_features` function is designed to extract the feature importances from a single model,
- and it may not directly apply to ensemble stacking models that combine multiple models.

```
def important_features(model, x_city):
    importance = model.feature_importances_
    # assign variable names
    variable_names = x_city.columns
    # Matching important features and variable_names
    importance_with_names = list(zip(variable_names, importance))
    importance_with_names_sorted = sorted(importance_with_names, key=lambda x: x[1], reverse=True)
    # Top 10 Important Features
    top_10_features_df = pd.DataFrame(importance_with_names_sorted[:10], columns=['Variable', 'Importance'])
    return top_10_features_df
```

Figure 27: Function to find the top 10 important factors in models

Running Model with Estimator effect graph

```
def effect_estimators(train_x, train_y, test_x):
    # Try different numbers of n_estimators - this will take a minute or so
    estimators = np.arange(10, 200, 20)
    scores = []
    best_score = -np.inf
    best_model = None

    for n in estimators:
        model.set_params(n_estimators=n, random_state=123)
        model.fit(train_x, train_y)
        score = model.score(train_x, train_y)
        scores.append(score)

        if score > best_score:
            best_score = score
            best_model = model

    plt.title("Effect of n_estimators")
    plt.xlabel("n_estimator")
    plt.ylabel("score")
    plt.plot(estimators, scores)

    return best_model, best_model.predict(test_x), best_score
```

Figure 28: 20 times running to find the best model and hyperparameters

Evaluation Tools

```
#https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics
def eval_metrics(y_test, y_pred, X_train):
    rmse = np.sqrt(mean_squared_error(y_test,y_pred))
    r2 = r2_score(y_test,y_pred)
    # Scikit-learn doesn't have adjusted r-square, hence custom code
    n = y_pred.shape[0]
    k = X_train.shape[1]
    adj_r_sq = 1 - (1 - r2)*(n-1)/(n-1-k)
    ae = np.abs(y_test - y_pred) / y_test
    filtered_ae = ae[np.isfinite(ae)]
    mae = np.mean(filtered_ae)
    accuracy = 100 - (mae * 100) # 100 - mape

    return round(adj_r_sq, 2), round(rmse, 2), round(mae, 2), round(accuracy, 2)
```

Figure 29: Evaluation function for 4 different metrics

bayes_search Parameter

```
bayes_search = {
    'n_estimators': (100, 1000),
    'learning_rate': (0.01, 1.0),
    'max_depth': (1, 16),
    'gamma': (0.01, 1.0),
}
```

Figure 30: Bayesian search setting

Functions for Genetic Algorithm

Evaluate Models

```
def evaluate_GA(idx, meta_features, test_y):
    # Convert hyperparameters to integers where necessary
    n_estimators = int(idx[0])
    max_depth = int(idx[2])

    # Create the optimized meta-learner with the given hyperparameters
    meta_learner = xgb.XGBRegressor(n_estimators=n_estimators, learning_rate=idx[1], max_depth=max_depth, g
amma=idx[3], random_state=123)

    # Fit the optimized meta-learner on the meta-features
    meta_learner.fit(meta_features, test_y)

    # Make predictions using the optimized meta-learner on the meta-features
    GA_pred = meta_learner.predict(meta_features)

    # Evaluate the ensemble model
    GA_rmse = mean_squared_error(test_y, GA_pred, squared=False)
    GA_ae = np.abs(test_y - GA_pred) / test_y
    filtered_GA_ae = GA_ae[np.isfinite(GA_ae)]
    GA_mae = np.mean(filtered_GA_ae)
    GA_accuracy = 100 - (GA_mae * 100)  # 100 - mape

    return round(GA_rmse, 2), round(GA_mae, 2), round(GA_accuracy, 2)
```

Figure 31: Evaluation function for 3 different metrics of Genetic Algorithm

DEAP and Parameter Tuning

```
def optimize_GA(meta_features, test_y):
    # Genetic Algorithm optimization using DEAP
    creator.create("FitnessMin", base.Fitness, weights=(-1.0, -0.1, -0.1))
    creator.create("idx", list, fitness=creator.FitnessMin)
    toolbox = base.Toolbox()
    toolbox.register("attr_float", np.random.uniform, 0.01, 1.0)
    toolbox.register("idx", tools.initRepeat, creator.idx, toolbox.attr_float, n=4)
    toolbox.register("population", tools.initRepeat, list, toolbox.idx)
    toolbox.register("mate", tools.cxTwoPoint)
    toolbox.register("mutate", tools.mutUniformInt, low=1, up=10, indpb=0.2)
    toolbox.register("select", tools.selTournament, tournsize=3)
    toolbox.register("evaluate", evaluate_GA, meta_features=meta_features, test_y=test_y)

    # Create the initial population
    population = toolbox.population(n=100)

    # Initialize a list to store the best fitness values over generations
    fitness_values = []
    rmse_values = []
    mae_values = []
    accuracy_values = []
    model = None
    best_accuracy = 0.0

    for gen in range(1, 21):  # Replace 21 with the number of generations you want to run
        population = algorithms.varAnd(population, toolbox, cxpb=0.5, mutpb=0.2)
        fitness_values = list(map(toolbox.evaluate, population))
        for ind, fit in zip(population, fitness_values):
            ind.fitness.values = fit
        best_idx = tools.selBest(population, k=1)[0]
        fitness_values.append(best_idx.fitness.values[0])
        rmse_values.append(evaluate_GA(best_idx, meta_features, test_y)[0])
        mae_values.append(evaluate_GA(best_idx, meta_features, test_y)[1])
        accuracy_values.append(evaluate_GA(best_idx, meta_features, test_y)[2])

        # Calculate the accuracy for the current best_idx
        current_accuracy = evaluate_GA(best_idx, meta_features, test_y)[2]

        # Store the best_idx if its accuracy is higher than the current best
        if current_accuracy > best_accuracy:
            best_accuracy = current_accuracy
            best_model = xgb.XGBRegressor(
                n_estimators=int(best_idx[0]),
                learning_rate=best_idx[1],
                max_depth=int(best_idx[2]),
                gamma=best_idx[3],
                random_state=123)

        # Print the best fitness value for each generation
        print(f"Generation {gen}, RMSE: {best_idx.fitness.values[0]}, MAE: {evaluate_GA(best_idx, meta_features, test_y)[1]},
Accuracy: {evaluate_GA(best_idx, meta_features, test_y)[2]}")

    # Plot the progress of optimization over generations
    gen = range(1, 21)
    plt.plot(gen, rmse_values, label='RMSE')
    plt.plot(gen, mae_values, label='MAE')
    plt.plot(gen, accuracy_values, label='Accuracy')
    plt.title("Genetic Algorithm Optimization Progress")
    plt.xlabel("Generation")
    plt.ylabel("Error / Accuracy Value")
    plt.legend()
    plt.show()

    return best_model
```

Figure 32: DEAP framework and Genetic Algorithm

6.2 Base Learners

For the base learners, the execution procedure is the same across the three methods, Figure 33, Figure 34, and Figure 35.

Random Forest

```
model = RandomForestRegressor()

# Use the effect_estimators function to get the best model, predictions, and best score
bergamo_rf_model, bergamo_rf_pred, bergamo_rf_score = effect_estimators(train_x_bergamo, train_y_bergamo, te
st_x_bergamo)
print("Best Score:", bergamo_rf_score)
print()

# Call the eval_metrics function and store the results in variables
bergamo_rf_adj_R, bergamo_rf_rmse, bergamo_rf_mae, bergamo_rf_accuracy = eval_metrics(test_y_bergamo, bergam
o_rf_pred, train_x_bergamo)

# Print the results
print("Adjusted R-squared:", bergamo_rf_adj_R)
print("RMSE:", bergamo_rf_rmse)
print("Mean Absolute Error:", bergamo_rf_mae)
print("Accuracy:", bergamo_rf_accuracy, "%")
```

Figure 33: Random Forest Modeling, Fitting, Prediction and Evaluation

Gradient Boosting

```
model = GradientBoostingRegressor()

# Use the effect_estimators function to get the best model, predictions, and best score
bergamo_gb_model, bergamo_gb_pred, bergamo_gb_score = effect_estimators(train_x_bergamo, train_y_bergamo, te
st_x_bergamo)
print("Best Score:", bergamo_gb_score)
print()

# Evaluation Tools
# Call the eval_metrics function and store the results in variables
bergamo_gb_adj_R, bergamo_gb_rmse, bergamo_gb_mae, bergamo_gb_accuracy = eval_metrics(test_y_bergamo, bergam
o_gb_pred, train_x_bergamo)

# Print the results
print("Adjusted R-squared:", bergamo_gb_adj_R)
print("RMSE:", bergamo_gb_rmse)
print("Mean Absolute Error:", bergamo_gb_mae)
print("Accuracy:", bergamo_gb_accuracy, "%")
```

Figure 34: Gradient Boosting Modeling, Fitting, Prediction and Evaluation

Light GBM

```
model = lgb.LGBMRegressor()

# Use the effect_estimators function to get the best model, predictions, and best score
bergamo_lgb_model, bergamo_lgb_pred, bergamo_lgb_score = effect_estimators(train_x_bergamo, train_y_bergamo,
test_x_bergamo)
print("Best Score:", bergamo_lgb_score)
print()

# Evaluation Tools
# Call the eval_metrics function and store the results in variables
bergamo_lgb_adj_R, bergamo_lgb_rmse, bergamo_lgb_mae, bergamo_lgb_accuracy = eval_metrics(test_y_bergamo, be
rgamo_lgb_pred, train_x_bergamo)

# Print the results
print("Adjusted R-squared:", bergamo_lgb_adj_R)
print("RMSE:", bergamo_lgb_rmse)
print("Mean Absolute Error:", bergamo_lgb_mae)
print("Accuracy:", bergamo_lgb_accuracy, "%")
```

Figure 33: LightGBM Modeling, Fitting, Prediction and Evaluation

6.3 Meta Learner

XGBoost, as a Metalearner and a single ensemble model, is consistent with the above execution method.

XGBoost

```
model = xgb.XGBRegressor()

# Use the effect_estimators function to get the best model, predictions, and best score
bergamo_xgb_model, bergamo_xgb_pred, bergamo_xgb_score = effect_estimators(train_x_bergamo, train_y_bergamo,
test_x_bergamo)
print("Best Score:", bergamo_xgb_score)
print()

# Evaluation Tools
# Call the eval_metrics function and store the results in variables
bergamo_xgb_adj_R, bergamo_xgb_rmse, bergamo_xgb_mae, bergamo_xgb_accuracy = eval_metrics(test_y_bergamo, be
rgamo_xgb_pred, train_x_bergamo)

# Print the results
print("Adjusted R-squared:", bergamo_xgb_adj_R)
print("RMSE:", bergamo_xgb_rmse)
print("Mean Absolute Error:", bergamo_xgb_mae)
print("Accuracy:", bergamo_xgb_accuracy, "%")
```

Figure 33: XGBoost Modeling, Fitting, Prediction and Evaluation

6.4 Ensemble Stacking

Ensemble stacking model is consistent with the above execution method, except for the use of meta-features which are stacked from base learners' predicted values.

Ensemble Stacking

```
model = xgb.XGBRegressor()
# Create a new feature matrix with base learners' predictions as meta-features
bergamo_meta_features = np.column_stack((bergamo_rf_pred, bergamo_gb_pred, bergamo_lgb_pred))

# Use the effect_estimators function to get the best model, predictions, and best score
bergamo_stack_model, bergamo_stack_pred, bergamo_stack_score = effect_estimators(bergamo_meta_features, test
_y_bergamo, bergamo_meta_features)
print("Best Score:", bergamo_stack_score)
print()

# Evaluation Tools
# Call the eval_metrics function and store the results in variables
bergamo_stack_adj_R, bergamo_stack_rmse, bergamo_stack_mae, bergamo_stack_accuracy = eval_metrics(test_y_ber
gamo, bergamo_stack_pred, bergamo_meta_features)
```

Figure 34: Ensemble stacking model

6.5 Bayesian Optimisation

Bayesian optimisation

```
# Meta Learner
bergamo_meta_learner = xgb.XGBRegressor(n_estimators=100, random_state=123)

# Perform Bayesian Search for the meta-learner
bergamo_bayes_opt = BayesSearchCV(bergamo_meta_learner, bayes_search, n_iter=10, cv=10,
                                  scoring="neg_mean_squared_error", verbose=4, random_state=123)

# Fit the optimized meta-learner on the meta-features
bergamo_bayes_opt.fit(bergamo_meta_features, test_y_bergamo)

# Make predictions using the meta-learner on the test data
bergamo_bayes_pred = bergamo_bayes_opt.predict(bergamo_meta_features)
```

Figure 35: Bayesian search on the stacking model

6.6 Genetic Algorithm

The stacking model incorporating Genetic Algorithm follows a straightforward execution approach, as all functionalities were assigned to the 'optimize_GA()' function in advance.

Genetic Algorithm

```
# optimize_GA function running
best_GAmodel_bergamo = optimize_GA(bergamo_meta_features, test_y_bergamo)
```

```
Generation 1, RMSE: 4.14, MAE: 0.16, Accuracy: 84.43
Generation 2, RMSE: 8.38, MAE: 0.32, Accuracy: 68.48
Generation 3, RMSE: 5.04, MAE: 0.17, Accuracy: 83.33
Generation 4, RMSE: 5.9, MAE: 0.19, Accuracy: 81.48
Generation 5, RMSE: 3.7, MAE: 0.14, Accuracy: 85.91
Generation 6, RMSE: 2.4, MAE: 0.09, Accuracy: 91.31
Generation 7, RMSE: 2.4, MAE: 0.09, Accuracy: 91.31
Generation 8, RMSE: 3.75, MAE: 0.14, Accuracy: 85.61
Generation 9, RMSE: 3.01, MAE: 0.11, Accuracy: 89.02
Generation 10, RMSE: 3.01, MAE: 0.11, Accuracy: 89.02
Generation 11, RMSE: 3.01, MAE: 0.11, Accuracy: 89.02
Generation 12, RMSE: 2.45, MAE: 0.09, Accuracy: 91.04
Generation 13, RMSE: 3.14, MAE: 0.12, Accuracy: 88.42
Generation 14, RMSE: 3.56, MAE: 0.13, Accuracy: 86.94
Generation 15, RMSE: 3.55, MAE: 0.12, Accuracy: 88.13
Generation 16, RMSE: 2.05, MAE: 0.07, Accuracy: 92.91
Generation 17, RMSE: 2.3, MAE: 0.09, Accuracy: 91.28
Generation 18, RMSE: 2.35, MAE: 0.09, Accuracy: 91.17
Generation 19, RMSE: 2.84, MAE: 0.11, Accuracy: 89.23
Generation 20, RMSE: 2.05, MAE: 0.07, Accuracy: 92.91
```

Figure 36: Iteration of GA model with 20 Generation runnings

```
# Use the best model
best_GAmodel_bergamo.fit(bergamo_meta_features, test_y_bergamo)
bergamo_GA_pred = best_GAmodel_bergamo.predict(bergamo_meta_features)
```

Figure 37: With best hyperparameters, Fitting and prediction for GA model

7 Evaluation

To evaluate the model's performance in Adjusted-R², RMSE, MAE, Accuracy is same as shown below.

```
# Evaluation Tools
# Call the eval_metrics function and store the results in variables
bergamo_stack_adj_R, bergamo_stack_rmse, bergamo_stack_mae, bergamo_stack_accuracy = eval_metrics(test_y_bergamo,
                                                                                    bergamo_stack_pred, bergamo_meta_features)

# Print the results
print("Adjusted R-squared:", bergamo_stack_adj_R)
print("RMSE:", bergamo_stack_rmse)
print("Mean Absolute Error:", bergamo_stack_mae)
print("Accuracy:", bergamo_stack_accuracy, "%")
```

```
Best Score: 0.9996393110774995

Adjusted R-squared: 1.0
RMSE: 0.16
Mean Absolute Error: 0.01
Accuracy: 99.41 %
```

Figure 38: Bayesian Optimisation

```
# Evaluation Tools
# Call the eval_metrics function and store the results in variables
bergamo_bayes_adj_R, bergamo_bayes_rmse, bergamo_bayes_mae, bergamo_bayes_accuracy = eval_metrics(test_y_bergamo,
                                                                                    bergamo_bayes_pred, bergamo_meta_features)

# Print the results
print("Adjusted R-squared:", bergamo_bayes_adj_R)
print("RMSE:", bergamo_bayes_rmse)
print("Mean Absolute Error:", bergamo_bayes_mae)
print("Accuracy:", bergamo_bayes_accuracy, "%")
```

```
Adjusted R-squared: 0.94
RMSE: 2.09
Mean Absolute Error: 0.08
Accuracy: 91.77 %
```

Figure 38: Bayesian Optimisation

```
# Use the best model
best_GAmodel_bergamo.fit(bergamo_meta_features, test_y_bergamo)
bergamo_GA_pred = best_GAmodel_bergamo.predict(bergamo_meta_features)
# Evaluation Tools
# Call the eval_metrics function and store the results in variables
bergamo_GA_adj_R, bergamo_GA_rmse, bergamo_GA_mae, bergamo_GA_accuracy = eval_metrics(test_y_bergamo,
                                                                        bergamo_GA_pred, bergamo_meta_features)
# Print the results
print("Adjusted R-squared:", bergamo_GA_adj_R)
print("RMSE:", bergamo_GA_rmse)
print("Mean Absolute Error:", bergamo_GA_mae)
print("Accuracy:", bergamo_GA_accuracy, "%")
```

```
Adjusted R-squared: 0.94
RMSE: 2.05
Mean Absolute Error: 0.07
Accuracy: 92.91 %
```

Figure 38: Genetic Algorithm

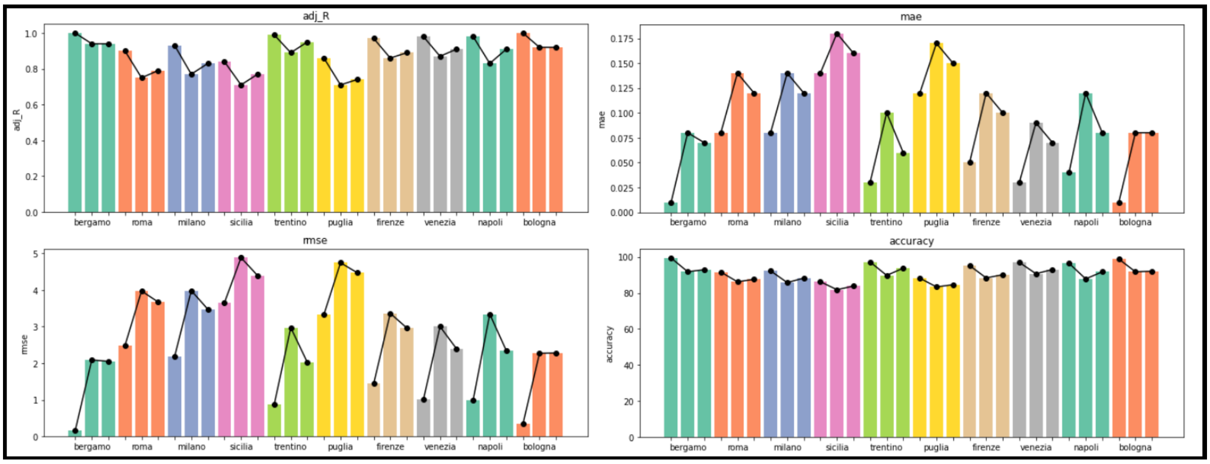

Figure 39: Comparison graphs on Stacking, BO and GA in 10 Datasets

References

Agarwal, A. (2020). Implementation of Genetic Algorithm/Evolutionary Algorithm in Python using DEAP framework. [online] Medium. Available at: https://aviral-agarwal.medium.com/implementation-of-genetic-algorithm-evolutionary-algorithm-in-python-using-deap-framework-c2d4bd247f70.

Yenigün, O. (2023). Step-by-Step Guide to Bayesian Optimization: A Python-based Approach. [online] Medium. Available at: https://medium.com/@okanyenigun/step-by-step-guide-to-bayesian-optimization-a-python-based-approach-3558985c6818 [Accessed 13 Aug. 2023].

www.kaggle.com. (n.d.). Airbnb in Italy. [online] Available at: https://www.kaggle.com/datasets/alessiocrisafulli/airbnb-italy [Accessed 13 Aug. 2023].