Group31_[redacted]

March 10, 2023

1 [redacted]: Machine Learning for Business 22/23

2 Table of Contents:

- Cleaning the Data to obtain 1,000 Original Instances that could be utilised in the process of Synthetic Data Generation
- Selecting 1,000 instances of Original Data

Question 6 Creating a Function for Synthetic Data Generation

- Utilising CTGAN to mimic the joint distribution of the original data

Question 7:

i) Generating Synthetic Data

ii) Assessing the Distributions

iii) Assessing the Correlations

- **Question 7 (A):** Generating 1,000 Synthetic Data Instances using the created Function
- **Question 7 (B):** Plotting the Distributions of Original VS Synthetic Data
- **Question 7 (C):** Correlations

i) Creating a Correlation Matrix for the Original Data

ii) Creating a Correlation Matrix for the Synthetic Data

iii) Creating a Correlation Matrix for the differences between the Original and Synthetic Data

Question 8: Building Predictive Models

- Selecting the Training and Testing Instances

i) 8 (A): 500 Original Data Training Instances

- Model 1 (A): Ridge Regression
- Model 2 (A): Decision Tree Regression with Tuned HyperParameters
- Model III (A): Neural Network Regression with tuned HuperParameters for the Original Data

ii) 8 (B): 500 Synthetic Training Instances

- Model I (B): Ridge Regression
- Model II (B): Decision Tree Regression with same HyperParameters that were tuned for the Original Data
- Model III (B): Neural Network Regression with the Same Hyperparameters as utilised for the Original Data

iii) 8 (C): 10,000 Synthetic Training Instances

- Generating 10,000 Instances of Synthetic Data utilising the same function
- Model I (C): Ridge Regression
- Model II (C): Decision Tree Regression with same HyperParameters that were tuned for the Original Data
- Model III (C): Neural Network Regression with same HyperParameters as Used for 8(A)

Question 9: Comparing the Models

References

[2]:
```
#LIBRARIES UTILISED THROUGHOUT THE ANALYSIS
import pandas as pd
import numpy as np
import random

import seaborn as sns
import matplotlib.pyplot as plt

#PART 1:
#to evaluate synthetic data using KL-divergence
#for KL divergence - Question 1
import scipy.stats
#for KL Divergence - Q2
from scipy.special import rel_entr


#LIBRARIES UTILISED SOLELY FOR PART 2:

#Question 6: Synthetic Data Generation
!pip install ctgan
!pip install --upgrade ctgan#upgrading the CTGAN Version
from ctgan import CTGAN

#Question 7:
from sklearn.model_selection import train_test_split
#Question 8:
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Ridge
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
import statsmodels.api as sm
from sklearn import linear_model
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import GridSearchCV
!pip install yellowbrick
from yellowbrick.regressor import PredictionError, ResidualsPlot
from sklearn.tree import plot_tree
from sklearn.metrics import mean_absolute_error


import tensorflow as tf
# set device placement to CPU-only
 #This will ensure that TensorFlow runs on the CPU instead of attempting to use␣
 ↪a GPU
tf.config.set_visible_devices([], 'GPU')
```

```
from tensorflow import keras
from keras import backend as K
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras import layers
#timing the model training
import time

#executing all codes in the cell
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = 'all'
```

```
/bin/bash: /opt/anaconda/envs/Python3/lib/libtinfo.so.6: no version information
available (required by /bin/bash)
```

```
         .::::.     .:::.
        …yy:    .yy.
        :.  .yy.    y.
             :y:   .:
             .yy  .:
              yy..:
              :y:.
              .y.
             .:.
        …:.
        :::.
```

```
• Project files and data should be stored in /project. This is shared among
everyone
  in the project.
• Personal files and configuration should be stored in /home/faculty.
• Files outside /project and /home/faculty will be lost when this server is
terminated.
• Create custom environments to setup your servers reproducibly.
```

```
bash: /opt/anaconda/envs/Python3/lib/libtinfo.so.6: no version information
available (required by bash)
Requirement already satisfied: ctgan in
```

```
/opt/anaconda/envs/Python3/lib/python3.9/site-packages (0.7.1)
Requirement already satisfied: rdt<2.0,>=1.3.0 in
/opt/anaconda/envs/Python3/lib/python3.9/site-packages (from ctgan) (1.3.0)
Requirement already satisfied: numpy<2,>=1.20.0 in
/opt/anaconda/envs/Python3/lib/python3.9/site-packages (from ctgan) (1.21.5)
Requirement already satisfied: pandas<2,>=1.1.3 in
/opt/anaconda/envs/Python3/lib/python3.9/site-packages (from ctgan) (1.5.1)
Requirement already satisfied: torch<2,>=1.8.0 in
/opt/anaconda/envs/Python3/lib/python3.9/site-packages (from ctgan) (1.13.1)
Requirement already satisfied: packaging<22,>=20 in
/opt/anaconda/envs/Python3/lib/python3.9/site-packages (from ctgan) (21.3)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in
/opt/anaconda/envs/Python3/lib/python3.9/site-packages (from
packaging<22,>=20->ctgan) (3.0.9)
Requirement already satisfied: python-dateutil>=2.8.1 in
/opt/anaconda/envs/Python3/lib/python3.9/site-packages (from
pandas<2,>=1.1.3->ctgan) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in
/opt/anaconda/envs/Python3/lib/python3.9/site-packages (from
pandas<2,>=1.1.3->ctgan) (2022.1)
Requirement already satisfied: scikit-learn<2,>=0.24 in
/opt/anaconda/envs/Python3/lib/python3.9/site-packages (from
rdt<2.0,>=1.3.0->ctgan) (1.0.2)
Requirement already satisfied: psutil<6,>=5.7 in
/opt/anaconda/envs/Python3/lib/python3.9/site-packages (from
rdt<2.0,>=1.3.0->ctgan) (5.9.0)
Requirement already satisfied: Faker>=10 in
/opt/anaconda/envs/Python3/lib/python3.9/site-packages (from
rdt<2.0,>=1.3.0->ctgan) (17.6.0)
Requirement already satisfied: scipy<2,>=1.5.4 in
/opt/anaconda/envs/Python3/lib/python3.9/site-packages (from
rdt<2.0,>=1.3.0->ctgan) (1.7.3)
Requirement already satisfied: nvidia-cuda-nvrtc-cu11==11.7.99 in
/opt/anaconda/envs/Python3/lib/python3.9/site-packages (from
torch<2,>=1.8.0->ctgan) (11.7.99)
Requirement already satisfied: nvidia-cublas-cu11==11.10.3.66 in
/opt/anaconda/envs/Python3/lib/python3.9/site-packages (from
torch<2,>=1.8.0->ctgan) (11.10.3.66)
Requirement already satisfied: nvidia-cuda-runtime-cu11==11.7.99 in
/opt/anaconda/envs/Python3/lib/python3.9/site-packages (from
torch<2,>=1.8.0->ctgan) (11.7.99)
Requirement already satisfied: nvidia-cudnn-cu11==8.5.0.96 in
/opt/anaconda/envs/Python3/lib/python3.9/site-packages (from
torch<2,>=1.8.0->ctgan) (8.5.0.96)
Requirement already satisfied: typing-extensions in
/opt/anaconda/envs/Python3/lib/python3.9/site-packages (from
torch<2,>=1.8.0->ctgan) (4.4.0)
Requirement already satisfied: setuptools in
```

```
/opt/anaconda/envs/Python3/lib/python3.9/site-packages (from nvidia-cublas-
cu11==11.10.3.66->torch<2,>=1.8.0->ctgan) (65.5.0)
Requirement already satisfied: wheel in
/opt/anaconda/envs/Python3/lib/python3.9/site-packages (from nvidia-cublas-
cu11==11.10.3.66->torch<2,>=1.8.0->ctgan) (0.37.1)
Requirement already satisfied: six>=1.5 in
/opt/anaconda/envs/Python3/lib/python3.9/site-packages (from python-
dateutil>=2.8.1->pandas<2,>=1.1.3->ctgan) (1.16.0)
Requirement already satisfied: joblib>=0.11 in
/opt/anaconda/envs/Python3/lib/python3.9/site-packages (from scikit-
learn<2,>=0.24->rdt<2.0,>=1.3.0->ctgan) (1.1.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/opt/anaconda/envs/Python3/lib/python3.9/site-packages (from scikit-
learn<2,>=0.24->rdt<2.0,>=1.3.0->ctgan) (2.2.0)
/bin/bash: /opt/anaconda/envs/Python3/lib/libtinfo.so.6: no version information
available (required by /bin/bash)
```

```
         .::::.     .::.
        …yy:    .yy.
        :.  .yy.    y.
             :y:   .:
             .yy  .:
              yy..:
              :y:.
              .y.
             .:.
        …:.
        :::.
```

- Project files and data should be stored in /project. This is shared among everyone
  in the project.
- Personal files and configuration should be stored in /home/faculty.
- Files outside /project and /home/faculty will be lost when this server is terminated.
- Create custom environments to setup your servers reproducibly.

```
bash: /opt/anaconda/envs/Python3/lib/libtinfo.so.6: no version information
available (required by bash)
ERROR: Invalid requirement: 'ctgan#upgrading'
/bin/bash: /opt/anaconda/envs/Python3/lib/libtinfo.so.6: no version
information available (required by /bin/bash)

         .::::.     .:::.
        …yy:    .yy.
        :.  .yy.    y.
             :y:   .:
             .yy  .:
              yy..:
              :y:.
              .y.
             .:.
        …:.
        :::.

• Project files and data should be stored in /project. This is shared among
everyone
  in the project.
• Personal files and configuration should be stored in /home/faculty.
• Files outside /project and /home/faculty will be lost when this server is
terminated.
• Create custom environments to setup your servers reproducibly.

bash: /opt/anaconda/envs/Python3/lib/libtinfo.so.6: no version information
available (required by bash)
Requirement already satisfied: yellowbrick in
/opt/anaconda/envs/Python3/lib/python3.9/site-packages (1.5)
Requirement already satisfied: matplotlib!=3.0.0,>=2.0.2 in
/opt/anaconda/envs/Python3/lib/python3.9/site-packages (from yellowbrick)
(3.5.3)
Requirement already satisfied: scipy>=1.0.0 in
/opt/anaconda/envs/Python3/lib/python3.9/site-packages (from yellowbrick)
(1.7.3)
Requirement already satisfied: scikit-learn>=1.0.0 in
```

```
/opt/anaconda/envs/Python3/lib/python3.9/site-packages (from yellowbrick)
(1.0.2)
Requirement already satisfied: numpy>=1.16.0 in
/opt/anaconda/envs/Python3/lib/python3.9/site-packages (from yellowbrick)
(1.21.5)
Requirement already satisfied: cycler>=0.10.0 in
/opt/anaconda/envs/Python3/lib/python3.9/site-packages (from yellowbrick)
(0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in
/opt/anaconda/envs/Python3/lib/python3.9/site-packages (from
matplotlib!=3.0.0,>=2.0.2->yellowbrick) (4.25.0)
Requirement already satisfied: kiwisolver>=1.0.1 in
/opt/anaconda/envs/Python3/lib/python3.9/site-packages (from
matplotlib!=3.0.0,>=2.0.2->yellowbrick) (1.4.2)
Requirement already satisfied: pillow>=6.2.0 in
/opt/anaconda/envs/Python3/lib/python3.9/site-packages (from
matplotlib!=3.0.0,>=2.0.2->yellowbrick) (9.2.0)
Requirement already satisfied: packaging>=20.0 in
/opt/anaconda/envs/Python3/lib/python3.9/site-packages (from
matplotlib!=3.0.0,>=2.0.2->yellowbrick) (21.3)
Requirement already satisfied: pyparsing>=2.2.1 in
/opt/anaconda/envs/Python3/lib/python3.9/site-packages (from
matplotlib!=3.0.0,>=2.0.2->yellowbrick) (3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in
/opt/anaconda/envs/Python3/lib/python3.9/site-packages (from
matplotlib!=3.0.0,>=2.0.2->yellowbrick) (2.8.2)
Requirement already satisfied: joblib>=0.11 in
/opt/anaconda/envs/Python3/lib/python3.9/site-packages (from scikit-
learn>=1.0.0->yellowbrick) (1.1.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/opt/anaconda/envs/Python3/lib/python3.9/site-packages (from scikit-
learn>=1.0.0->yellowbrick) (2.2.0)
Requirement already satisfied: six>=1.5 in
/opt/anaconda/envs/Python3/lib/python3.9/site-packages (from python-
dateutil>=2.7->matplotlib!=3.0.0,>=2.0.2->yellowbrick) (1.16.0)

2023-03-10 00:46:45.122668: W
tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load
dynamic library 'libcuda.so.1'; dlerror: libcuda.so.1: cannot open shared object
file: No such file or directory; LD_LIBRARY_PATH:
/opt/anaconda/envs/Python3/lib:
2023-03-10 00:46:45.122710: W
tensorflow/stream_executor/cuda/cuda_driver.cc:269] failed call to cuInit:
UNKNOWN ERROR (303)
2023-03-10 00:46:45.122730: I
tensorflow/stream_executor/cuda/cuda_diagnostics.cc:156] kernel driver does not
appear to be running on this host
(cube-d7aef6d2-e362-4f37-84f4-39aacc5b64ac-75dbc4d598-cplw2):
```

```
/proc/driver/nvidia/version does not exist
```

3 Part I: Generating synthetic data that matches the unseen real_test.csv

[3]:
```
# Loading the data as a pandas dataframe
df = pd.read_csv('real_train.csv')
```

[531]:
```
#printing the number of observations in the data
print('The Number of Observations is:', len(df))
```

```
The Number of Observations is: 100
```

[500]:
```
#checking the first 5 instances of the data
df.head()
```

[500]:

	x1	x2	x3
0	tpu	new	Glasgow
1	tpu	replacement	Birmingham
2	tpu	new	Birmingham
3	tpu	replacement	Glasgow
4	gpu	new	Glasgow

*Note that even upon setting a random seed in every code cell the synthetic data keeps changing upon every execution process due to the algorithm's stochastic nature. I.e., the algorithm is making random decisions during the data generation process, even upon the same random seed utilised, leading to slightly-differing results each time.

[101]:
```
#**setting a random seed to ensure that the generated synthetic data is the same
#every time the code is run
np.random.seed(8)
```

3.1 Question 1: Marginal Distribution

[88]:
```
#Step 1:
# Extracting the marginal distributions of the original data on each feature:
#x1, x2, and x3

#value_counts() -> counting the # of occurrences for each unique value
#in the feature specified

#normalize=True -> obtaining relative frequencies (normalising counts
#by dividing them by the total # samples)

#sort_index() -> sorting the resulting frequency counts in  the
#ascending order to align the probabilities with unique feature values
```

```
#The p1, p2, and p3 probability distributions  will be, consequently, utilised
#for synthetic data generation matching original data's marginal distributions.
p1 = df['x1'].value_counts(normalize=True).sort_index().values
p2 = df['x2'].value_counts(normalize=True).sort_index().values
p3 = df['x3'].value_counts(normalize=True).sort_index().values
```

[89]:
```
# Plotting histograms of the marginal distributions
fig, ax = plt.subplots(1, 3, figsize=(12, 4))
ax[0].hist(df['x1'], bins=len(p1), color='red', alpha=0.5, weights=np.
 ↪ones_like(df['x1']) / len(df['x1']))
ax[0].set_title('Marginal Distribution of x1')
ax[1].hist(df['x2'], bins=len(p2), color='green', alpha=0.5, weights=np.
 ↪ones_like(df['x2']) / len(df['x2']))
ax[1].set_title('Marginal Distribution of x2')
ax[2].hist(df['x3'], bins=len(p3), color='blue',alpha=0.5, weights=np.
 ↪ones_like(df['x3']) / len(df['x3']))
ax[2].set_title('Marginal Distribution of x3')

#Showing the Plot
plt.show();
```

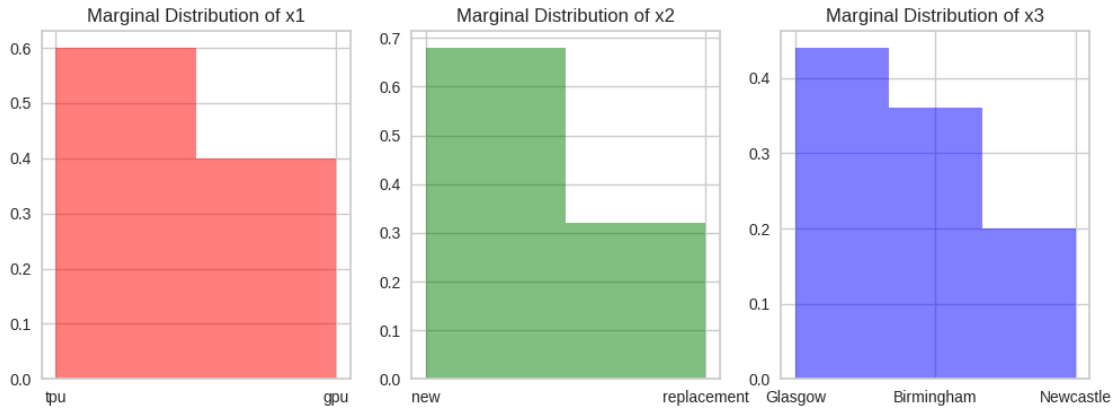

Figure 1:

The above figure depicts the marginal distribution of the x1, x2, x3 in the original data

[90]:
```
#Step 2:
#creating an empty 2D np array 'synthetic_data' for synthetic data with a shape␣
 ↪of (900,3)
# -> 900 rows and 3 columns (for x1, x2 and x3)
# This array will be then utilised to store synthetic data that we generate
#in the next step to match marginal distributions of the real DS.
#(it will be filled with synthetic data)
```

```
#Doing this is a more efficient and less computationally-expensive approach␣
 ↪than
# appending array rows one by one

n_rows = 900
#dtype='U25' -> for the array to store string values
#(Unicode strings with a maximum length of 25 chars)
synthetic_data = np.empty(shape = (n_rows, 3))
```

[92]:
```
# #Step 3:
#Generating synthetic data for x1 with the desired marginal distribution
#(1D np array)
x1_unique_values = df['x1'].unique()

np.random.seed(8)

#Generating a 1D np array q1 containing a random integers sample representing
#synthetic x1 values, with a probability distribution closely matching x2's␣
 ↪(real ds)
#marginal distribution

#utilising the np.random.choice() function
#to generate a random integers sample based on the p1 probability
#distribution (representing x1 feature's marginal distribution)

#1) using len(x1_unique_values) as the number of unique values
#(possible outcomes) of x1 since np.random.choice() function requires the
#p parameter to be a 1-D array-like obj having the same length as␣
 ↪x1_unique_values
#p1 array- probability distribution for each possible outcome

# len function -> specifying the elements' number in the population
#to randomly sample from

#2) size=n_rows -> sample size we want to generate (# rows in synthetic dataset)
#3)p=p2 -> probability distribution for each possible outcome,
#calculated based on each unique value's relative frequency in x1 real dataset␣
 ↪col.


#** the random seed that we set in the beginning of the code
#will ensure that random numbers generated for all the columns are always the␣
 ↪same
#every time we run the code

col1 = np.random.choice(len(x1_unique_values), size = n_rows, p = p1)
```

```
# Creating a dictionary to map integer values back to their corresponding␣
 ↪string values
#x1_dict = {i: val for i, val in enumerate(x1_unique_values)}

# Converting integer values back to their corresponding string values
#str_col1 = [x1_dict[val] for val in col1]

#setting the string values of the synthetic data generated for 'x2'
#as the second column of the synthetic_data array

synthetic_data[:, 0] = col1
```

[93]:
```
np.random.seed(8)
# Generating synthetic data for x2 with the desired marginal distribution
# createing a 1D np array of unique values in 'x2' column of the pd df..
x2_unique_values = df['x2'].unique()
col2 = np.random.choice(len(x2_unique_values), size = n_rows, p = p2)

# Creating a dictionary to map integer values back to their corresponding␣
 ↪string values
#x2_dict = {i: val for i, val in enumerate(x2_unique_values)}

# Converting integer values back to their corresponding string values
#str_col2 = [x2_dict[val] for val in col2]

#setting the string values of the synthetic data generated for 'x2'
#as the second column of the synthetic_data array
synthetic_data[:, 1] = col2
```

[94]:
```
np.random.seed(8)
# Generating synthetic data for x3 with the desired marginal distribution
x3_unique_values = df['x3'].unique()
col3 = np.random.choice(len(x3_unique_values), size = n_rows, p = p3)

# Creating a dictionary to map integer values back to their corresponding␣
 ↪string values
#x3_dict = {i: val for i, val in enumerate(x3_unique_values)}

# Converting integer values back to their corresponding string values
#str_col3 = [x3_dict[val] for val in col3]

#setting the string values of the synthetic data generated for 'x2'
#as the second column of the synthetic_data array
synthetic_data[:, 2] = col3
```

[95]:
```
# Converting synthetic data back to original string values
str_col1 = x1_unique_values[synthetic_data[:, 0].astype(int)]
str_col2 = x2_unique_values[synthetic_data[:, 1].astype(int)]
str_col3 = x3_unique_values[synthetic_data[:, 2].astype(int)]

# Combining columns into a single 2D np array
synthetic_data_str = np.column_stack((str_col1, str_col2, str_col3))
```

[96]:
```
#Step 4:
# Creating a pd df (synthetic_df) with the np array containing synthetic
#data we have created and feature names
#->setting the column names to be the same as in the original data and
#faciiltate comparability.
df_synth_marginal = pd.DataFrame(synthetic_data_str, columns = ['x1', 'x2',␣
 ↪'x3'])
df_synth_marginal
```

[96]:

	x1	x2	x3
0	gpu	new	Glasgow
1	gpu	replacement	Glasgow
2	gpu	replacement	Glasgow
3	gpu	new	Birmingham
4	gpu	new	Glasgow
..	…	…	…
895	tpu	new	Birmingham
896	gpu	new	Glasgow
897	gpu	new	Birmingham
898	gpu	replacement	Birmingham
899	gpu	replacement	Glasgow

[900 rows x 3 columns]

[51]:
```
#displaying the first 5 instances
df_synth_marginal.head()
```

[51]:

	x1	x2	x3
0	tpu	replacement	Birmingham
1	tpu	new	Glasgow
2	gpu	new	Birmingham
3	tpu	new	Glasgow
4	gpu	new	Birmingham

[52]:
```
print('The synthetic dataframe has {0} rows and {1} columns'.
 ↪format(len(df_synth_marginal),
                                                                  ␣
 ↪len(df_synth_marginal.columns)))
```

```
The synthetic dataframe has 900 rows and 3 columns
```

[97]:
```
p1_synth = df_synth_marginal['x1'].value_counts(normalize = True).sort_index().
 ↪values
p2_synth = df_synth_marginal['x2'].value_counts(normalize = True).sort_index().
 ↪values
p3_synth = df_synth_marginal['x3'].value_counts(normalize = True).sort_index().
 ↪values
```

[98]:
```
# Plotting histograms of the marginal distributions side by side
fig, axs = plt.subplots(2, 3, figsize=(16, 8))
axs[0, 0].hist(df['x1'], bins=len(p1), color='blue', alpha=0.5,
               weights=np.ones(len(df)) / len(df))
axs[0, 0].set_title('Marginal Distribution of x1')
axs[0, 1].hist(df['x2'], bins=len(p2), color='blue', alpha=0.5,
               weights=np.ones(len(df))/len(df))
axs[0, 1].set_title('Marginal Distribution of x2')
axs[0, 2].hist(df['x3'], bins=len(p3), color='blue', alpha=0.5,
               weights=np.ones(len(df)) / len(df))
axs[0, 2].set_title('Marginal Distribution of x3')

axs[1, 0].hist(df_synth_marginal['x1'], bins=len(p1_synth), color='red',␣
 ↪alpha=0.5,
               weights=np.ones(len(df_synth_marginal)) / len(df_synth_marginal))
axs[1, 0].set_title('Marginal Distribution of x1 (Synthetic Data)')
axs[1, 1].hist(df_synth_marginal['x2'], bins=len(p2_synth), color='red',
               alpha=0.5, weights=np.ones(len(df_synth_marginal)) /␣
 ↪len(df_synth_marginal))
axs[1, 1].set_title('Marginal Distribution of x2 (Synthetic Data)')
axs[1, 2].hist(df_synth_marginal['x3'], bins=len(p3_synth), color='red',␣
 ↪alpha=0.5,
               weights=np.ones(len(df_synth_marginal)) / len(df_synth_marginal))
axs[1, 2].set_title('Marginal Distribution of x3 (Synthetic Data)')

plt.show();
```

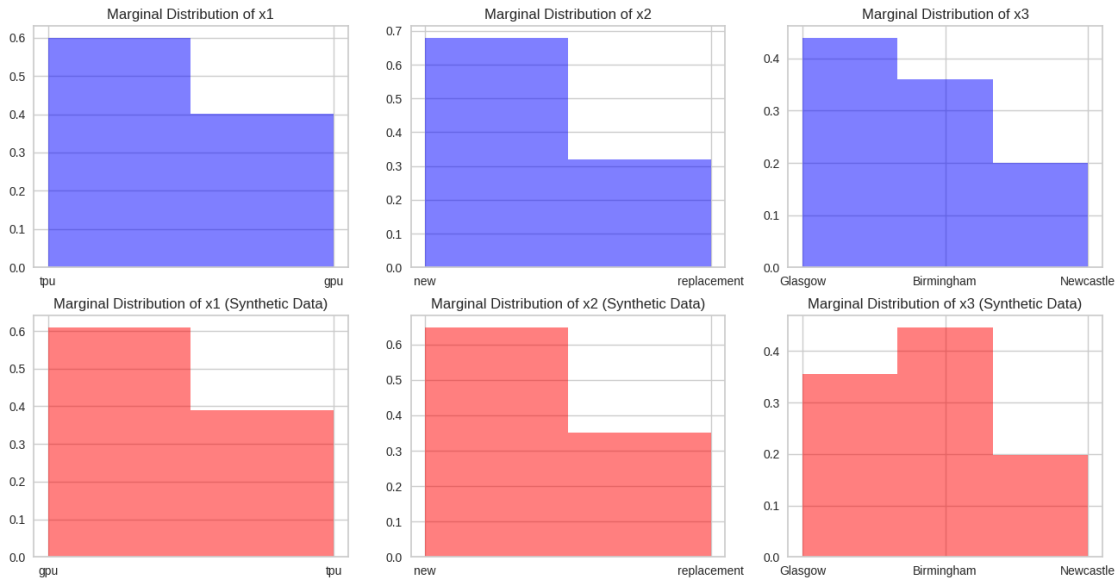

Figure 2: Marginal Distributions of Original VS Synthetic Data

The above figure shows the marginal distributions of the generated synthetic data of x1, x2, x3 and original data distributions. The distributions are generally similar to the distributions of the original data, hence, suggesting that the synthetic data is good. Meaning that the correlations and probabilities within the original data are closely replicated by the synthetic data. They are plotted together for comparison purposes.

[99]:
```
#Step 5:
#Exporting the synthetic data with feature names
#x1, x2, and x3 into a file called synthetic_marginal.csv
df_synth_marginal.to_csv('synthetic_marginal.csv')
```

3.1.1 *Evaluating the quality of our Synthetic Data for the Marginal Distrubution using KL-divergence*

[100]:
```
#Step 6: Evaluating the Quality of Synthetic Data using KL-Divergence

#Calculating KL-divergence between original and synthetic data's marginal␣
 ↪distributions

#Since KL-divergence measures the difference between 2 probability␣
 ↪distributions,
#it is calculated separately for each variable,
# since each variable has its own marginal distribution.

kl_div_1 = scipy.stats.entropy(p1, p1_synth)
kl_div_2 = scipy.stats.entropy(p2, p2_synth)
```

```
kl_div_3 = scipy.stats.entropy(p3, p3_synth)

print(f'KL-divergence for \'x1\': {kl_div_1}')
print(f'KL-divergence for \'x2\': {kl_div_2}')
print(f'KL-divergence for \'x3\': {kl_div_3}')
```

```
KL-divergence for 'x1': 0.08967198563172254
KL-divergence for 'x2': 0.0021551290808783166
KL-divergence for 'x3': 0.018340588704727732
```

3.2 Question 2: Joint Distribution

[544]:
```
#creating new data frame combs_df and dropping duplicates
combs_df = df.drop_duplicates()
combs_df
#Converting the dataframe into a numpy array
arr_combs = combs_df.to_numpy()
#converting arr_combs NumPy array into a list called combs_list.
combs_list = arr_combs.tolist()
print(combs_list)
```

[544]:

	x1	x2	x3
0	tpu	new	Glasgow
1	tpu	replacement	Birmingham
2	tpu	new	Birmingham
3	tpu	replacement	Glasgow
4	gpu	new	Glasgow
5	gpu	new	Newcastle
6	gpu	replacement	Newcastle
14	tpu	new	Newcastle
28	tpu	replacement	Newcastle
40	gpu	new	Birmingham
51	gpu	replacement	Birmingham
56	gpu	replacement	Glasgow

```
[['tpu', 'new', 'Glasgow'], ['tpu', 'replacement', 'Birmingham'], ['tpu', 'new',
'Birmingham'], ['tpu', 'replacement', 'Glasgow'], ['gpu', 'new', 'Glasgow'],
['gpu', 'new', 'Newcastle'], ['gpu', 'replacement', 'Newcastle'], ['tpu', 'new',
'Newcastle'], ['tpu', 'replacement', 'Newcastle'], ['gpu', 'new', 'Birmingham'],
['gpu', 'replacement', 'Birmingham'], ['gpu', 'replacement', 'Glasgow']]
```

[545]:
```
#converting the original DataFrame df into a NumPy array called realtrain_arr
#Each row of the DataFrame becomes a sub-array within the NumPy array
realtrain_arr = df.to_numpy()
#Converting the realtrain_arr NumPy array into a list called realtrain_list
realtrain_list = realtrain_arr.tolist()
#creating a list for joint probabilities
```

```
#Calculating the joint probability of combinations occurring in the original df
#Using a for loop
joint_probs = []
for i in combs_list:
    #getting the real joint probabilities of the categorical combinations
    joint_probs.append(realtrain_list.count(i)/len(df))
print(joint_probs)
```

```
[0.19, 0.12, 0.12, 0.06, 0.17, 0.06, 0.03, 0.08, 0.03, 0.06, 0.06, 0.02]
```

[546]:
```
np.random.seed(8)
#Generating synthetic data using numpy's random.choice function
#specifying number of samples to genenerate with 'size' parameter
#Specifying the probabilities of each categorical combination with the 'p'␣
 ↪parameter
synth_data = np.random.choice(len(arr_combs), size = 900, p = joint_probs)
#Indexing to select the rows corresponding to the synthetic data
prob_synth_joint = arr_combs[synth_data]
#creating a df from the synthetic data with column names 'x1', 'x2', and 'x3'
df_synth_joint = pd.DataFrame(prob_synth_joint, columns = ['x1', 'x2', 'x3'])

#displaying the synthetic df
display(df_synth_joint)
```

	x1	x2	x3
0	gpu	new	Birmingham
1	gpu	replacement	Birmingham
2	gpu	new	Birmingham
3	gpu	new	Glasgow
4	tpu	replacement	Birmingham
..	…	…	…
895	gpu	replacement	Glasgow
896	tpu	replacement	Newcastle
897	tpu	replacement	Glasgow
898	gpu	new	Glasgow
899	tpu	new	Glasgow

[900 rows x 3 columns]

[547]:
```
# Plotting joint distribution histograms for 'x1', 'x2', and 'x3'
fig, axs = plt.subplots(nrows=3, ncols=2, figsize=(12, 8))

for i, col in enumerate(['x1', 'x2', 'x3']):
    # Plotting the original data histogram
    axs[i][0].hist(df[col], bins=30, color='blue', alpha=0.5)
    axs[i][0].set_title(f'Original {col}')
    # Plotting the synthetic data histogram
```

```
    axs[i][1].hist(df_synth_joint[col], bins=30, color='red', alpha=0.5)
    axs[i][1].set_title(f'Synthetic {col}')

plt.tight_layout()

#Showing the Plot
plt.show();
```

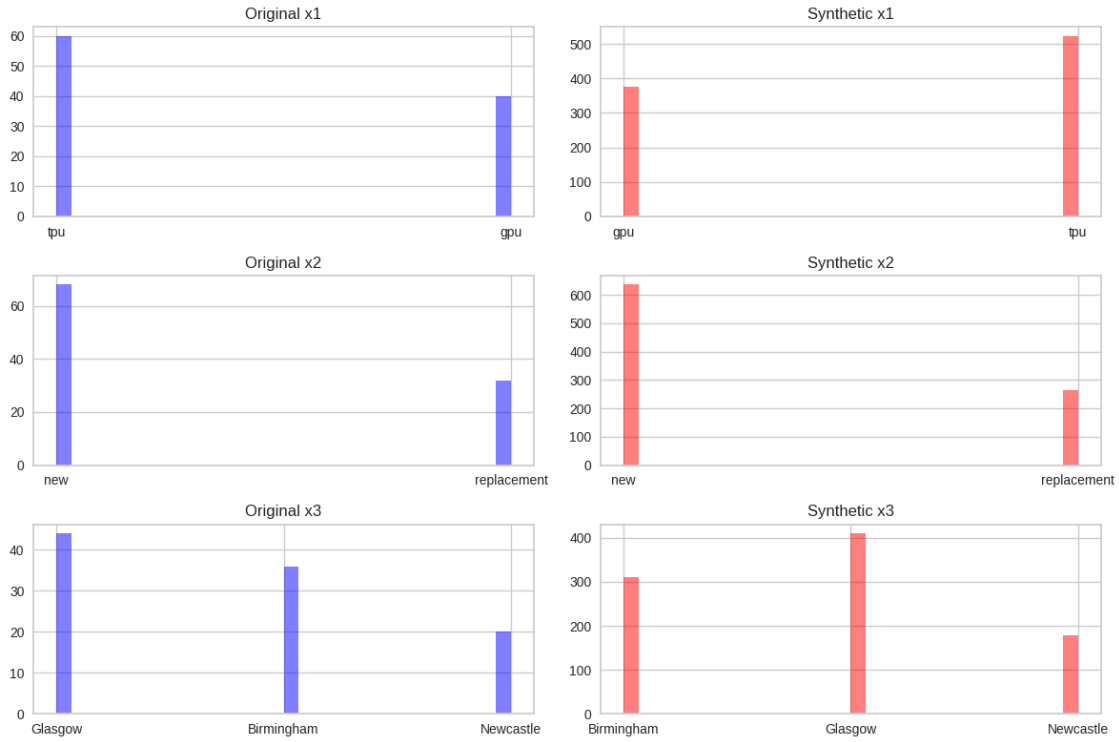

Figure 3: plotting the joint distributions. The graph demonstrates high similarity of the joint distributions

[672]:
```
## Exporting the synthetic data with feature names x1, x2, and x3 into a file␣
↪called synthetic_joint.csv
df_synth_joint.to_csv('synthetic_joint.csv')
```

3.2.1 *Evaluating the quality of our Synthetic Data for the Joint Distrubution using KL-divergence*

[673]:
```
#Setting the Random Seed
np.random.seed(8)

#Calculating the KL-divergence
#droping duplicate rows from df_synth_joint and assigning it to combs_synth
```

```
combs_synth = df_synth_joint.drop_duplicates()
#converting resulting dataframe to a numpy array and then list
array_combs_synth = combs_synth.to_numpy()
list_combs_synth = array_combs_synth.tolist()
#whole dataframe df_synth_joint is converted to a NumPy array and then to a list
array_whole_synth = df_synth_joint.to_numpy()
whole_list_synth = array_whole_synth.tolist()
#creating empty list synth_prob that will contain the synthetic probabilities
#loop is started over each element i in combs_list.
#For each i, the count of i in whole_list_synth is divided by 900
#(the total number of samples) and appended to synth_prob.
synth_prob = []
for i in combs_list:
    synth_prob.append(whole_list_synth.count(i)/900)

print(joint_probs)
print([round(i,2) for i in synth_prob])

#calculating the KL-divergence between the joint probabilities
#and the synthetic probabilities
#using the rel_entr function from the scipy library
#printing the KL Divergence
KL_divergence_joint = round(sum(rel_entr(synth_prob, joint_probs)), 5)

print(f'the KL-Divergence is: {KL_divergence_joint}')
```

```
[0.19, 0.12, 0.12, 0.06, 0.17, 0.06, 0.03, 0.08, 0.03, 0.06, 0.06, 0.02]
[0.18, 0.1, 0.14, 0.06, 0.19, 0.06, 0.03, 0.08, 0.03, 0.05, 0.06, 0.02]
the KL-Divergence is: 0.00607
```

3.3 Question 3: Marginal and Joint Distributions for Business Purposes

Self-supervised generative AI algorithms, producing realistic synthetic data are becoming increasingly appealing to businesses for privacy regulations, high acquisition costs and complexity (Moreno-Torres et al.,2012). Ideally, businesses desire synthetic data to mimic individual attributes' distribution, and all multivariate relationships combinations, for enhanced classification and prediction accuracy, which fail to be captured by marginal distributions(Emam,Hoptroff&Mosquera,2020). This requires comparing full joint empirical distributions(=) across synthetic and authentic data(Platzer&Reutterer, 2021). For instance, upon generating synthetic data for medical studies, joint distribution of gender, weight, age, and pre-existing medical conditions are crucial for predicting disease likelihood(ibid.).

However, the dimensionality curse(Bellman,1966) oftentimes makes synthetic data approximating joint distribution infeasible to use (Platzer&Reutterer,2021). Attribute values' cross-combinations surge exponentially with increasing attributes, creating high-dimensional data space with sparse available data(Platzer&Reutterer, 2021). While grouping and binning alleviates this issue for lower-level interactions, this underlying principle pertains at deeper levels. For instance, our data with 3 columns and 2-3 unique values/column comprised 12 combinations. Business datasets are typically

more intricate datasets, including more variables and attributes. For instance, a dataset with 50 variables would comprise 1,225 2-way and 19,600 3-way interactions, making generation unmanageable for time and computational power required(ibid.). Additionally, joint distribution synthetic data presents a privacy-utility trade-off (Stadler,Oprisanu&Troncoso, 2022). Requiring more data concerning variables' relationships, it, similarly, increases risks of personal information revealed, it is more prone to data privacy concerns(Emam Hoptroff&Mosquera, 2020).

Finally, joint distribution synthetic data is challenging for firms with limited data science and statistics knowledge. Additionally, accurate joint distribution synthetic data requires complex algorithms and data generation techniques, increasing business costs. Nevertheless, investment benefits from generation techniques could significantly outweigh the costs(Gartner,2022).

Thus, oftentimes, marginal distributions are utilised for multivariate distributions' analysis. Similarly, it is preferred upon businesses solely aiming to preserve each individual variable's distribution for data exploration or anomaly detection(Emam Hoptroff&Mosquera, 2020).

4 Part II: Developing a Business Application for Synthetic Data

4.1 Question 4: Business application for synthetic data (AirBnb)

Airbnb implements a peer-to-peer platform business model, earning commissions from bookings (Goel,2022). Thus, a prevalent business problem for Airbnb is hosts frequently struggling to select competitive listing pricings, leading to lower demand, leading to $352 million business loss in 2021(Curry,2023). On many e-commerce platforms where small sellers are facing a fixed inventory and limited selling time problem, price rigidity can be commonly observed(Pan et al.,2022). Hosts are required to know the value current renters attribute to their listings and future demand for their supply. A main driver for pricing dynamics is renters' willingness to pay(WTP). Thus, upon hosts failing to respond to changes in renters' WTP, both Airbnb and hosts incur revenue losses while leading to negative visitors' experience(ibid.).

By examining data on comparable properties, ML models can assist in predicting optimal listing prices(Assefa et al.,2020). However due to privacy issues, hosts may be reluctant to share real listing information. Thus, a possible solution for this issue would be generating synthetic data for ML models to utilise in predicting listing prices using Airbnb data.

Thus, synthetic data might be useful in generating data accurately reflecting elements affecting Airbnb pricings. By extracting values for the property features in the synthetic dataset, a well-fitted model to original data can be, subsequently, generated(Garcia et al.,2019). Thereby, instead of utilising hosts' actual data, a comparable dataset with similar statistical characteristics is constructed using synthetic data.The same column types as the original data, such as continuous variables like the number of bedrooms or bathrooms, categorical variables,including property type and neighbourhood, and an outcome variable such as listing price, can be used to create synthetic data. ML models can, consequently, be trained using this fictitious data without endangering hosts' privacy. This application would aid hosts to calculate ideal listing pricings, increasing Airbnb revenue by optimising the supply and demand's efficiency(Reynolds et al.,2019).

4.2 Question 5: Providing 1,000 Instances of 'Original' Data with 7 Columns

Utilising Cross-Sectional Data of Airbnb Listings in Amsterdam from July 1 to September 1, 2022. The Dataset was acquired from the Airbnb Database (Inside Airbnb, 2022).

i) Importing and Cleaning the Data to obtain 1,000 Original Instances that could be utilised in the process of Synthetic Data Generation

[674]:
```
df_airbnb = pd.read_csv('listings.csv')
print('Imported Dataset with {0} rows and {1} columns'.format(len(df_airbnb),
                                                              len(df_airbnb.
 ↪columns)))
```

```
Imported Dataset with 6893 rows and 75 columns
```

[553]:
```
#observing all the columns present in the dataset, their count and type
df_airbnb.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6893 entries, 0 to 6892
Data columns (total 75 columns):
 #   Column                                    Non-Null Count  Dtype
---  ------                                    --------------  -----
 0   id                                        6893 non-null   int64
 1   listing_url                               6893 non-null   object
 2   scrape_id                                 6893 non-null   int64
 3   last_scraped                              6893 non-null   object
 4   source                                    6893 non-null   object
 5   name                                      6893 non-null   object
 6   description                               6877 non-null   object
 7   neighborhood_overview                     4628 non-null   object
 8   picture_url                               6893 non-null   object
 9   host_id                                   6893 non-null   int64
 10  host_url                                  6893 non-null   object
 11  host_name                                 6893 non-null   object
 12  host_since                                6893 non-null   object
 13  host_location                             6122 non-null   object
 14  host_about                                4132 non-null   object
 15  host_response_time                        5236 non-null   object
 16  host_response_rate                        5236 non-null   object
 17  host_acceptance_rate                      6031 non-null   object
 18  host_is_superhost                         6885 non-null   object
 19  host_thumbnail_url                        6893 non-null   object
 20  host_picture_url                          6893 non-null   object
 21  host_neighbourhood                        3096 non-null   object
 22  host_listings_count                       6893 non-null   int64
 23  host_total_listings_count                 6893 non-null   int64
 24  host_verifications                        6893 non-null   object
 25  host_has_profile_pic                      6893 non-null   object
 26  host_identity_verified                    6893 non-null   object
 27  neighbourhood                             4628 non-null   object
 28  neighbourhood_cleansed                    6893 non-null   object
 29  neighbourhood_group_cleansed              0 non-null      float64
```

```
 30  latitude                                        6893 non-null   float64
 31  longitude                                       6893 non-null   float64
 32  property_type                                   6893 non-null   object
 33  room_type                                       6893 non-null   object
 34  accommodates                                    6893 non-null   int64
 35  bathrooms                                       0 non-null      float64
 36  bathrooms_text                                  6878 non-null   object
 37  bedrooms                                        6578 non-null   float64
 38  beds                                            6792 non-null   float64
 39  amenities                                       6893 non-null   object
 40  price                                           6893 non-null   object
 41  minimum_nights                                  6893 non-null   int64
 42  maximum_nights                                  6893 non-null   int64
 43  minimum_minimum_nights                          6891 non-null   float64
 44  maximum_minimum_nights                          6891 non-null   float64
 45  minimum_maximum_nights                          6891 non-null   float64
 46  maximum_maximum_nights                          6891 non-null   float64
 47  minimum_nights_avg_ntm                          6891 non-null   float64
 48  maximum_nights_avg_ntm                          6891 non-null   float64
 49  calendar_updated                                0 non-null      float64
 50  has_availability                                6893 non-null   object
 51  availability_30                                 6893 non-null   int64
 52  availability_60                                 6893 non-null   int64
 53  availability_90                                 6893 non-null   int64
 54  availability_365                                6893 non-null   int64
 55  calendar_last_scraped                           6893 non-null   object
 56  number_of_reviews                               6893 non-null   int64
 57  number_of_reviews_ltm                           6893 non-null   int64
 58  number_of_reviews_l30d                          6893 non-null   int64
 59  first_review                                    6247 non-null   object
 60  last_review                                     6247 non-null   object
 61  review_scores_rating                            6247 non-null   float64
 62  review_scores_accuracy                          6242 non-null   float64
 63  review_scores_cleanliness                       6242 non-null   float64
 64  review_scores_checkin                           6242 non-null   float64
 65  review_scores_communication                     6242 non-null   float64
 66  review_scores_location                          6242 non-null   float64
 67  review_scores_value                             6242 non-null   float64
 68  license                                         6449 non-null   object
 69  instant_bookable                                6893 non-null   object
 70  calculated_host_listings_count                  6893 non-null   int64
 71  calculated_host_listings_count_entire_homes     6893 non-null   int64
 72  calculated_host_listings_count_private_rooms    6893 non-null   int64
 73  calculated_host_listings_count_shared_rooms     6893 non-null   int64
 74  reviews_per_month                               6247 non-null   float64
dtypes: float64(21), int64(19), object(35)
memory usage: 3.9+ MB
```

[554]:
```
#observing all the columns that are predent in the dataset
df_airbnb.columns.tolist()
```

[554]:
```
['id',
 'listing_url',
 'scrape_id',
 'last_scraped',
 'source',
 'name',
 'description',
 'neighborhood_overview',
 'picture_url',
 'host_id',
 'host_url',
 'host_name',
 'host_since',
 'host_location',
 'host_about',
 'host_response_time',
 'host_response_rate',
 'host_acceptance_rate',
 'host_is_superhost',
 'host_thumbnail_url',
 'host_picture_url',
 'host_neighbourhood',
 'host_listings_count',
 'host_total_listings_count',
 'host_verifications',
 'host_has_profile_pic',
 'host_identity_verified',
 'neighbourhood',
 'neighbourhood_cleansed',
 'neighbourhood_group_cleansed',
 'latitude',
 'longitude',
 'property_type',
 'room_type',
 'accommodates',
 'bathrooms',
 'bathrooms_text',
 'bedrooms',
 'beds',
 'amenities',
 'price',
 'minimum_nights',
 'maximum_nights',
 'minimum_minimum_nights',
```

```
 'maximum_minimum_nights',
 'minimum_maximum_nights',
 'maximum_maximum_nights',
 'minimum_nights_avg_ntm',
 'maximum_nights_avg_ntm',
 'calendar_updated',
 'has_availability',
 'availability_30',
 'availability_60',
 'availability_90',
 'availability_365',
 'calendar_last_scraped',
 'number_of_reviews',
 'number_of_reviews_ltm',
 'number_of_reviews_l30d',
 'first_review',
 'last_review',
 'review_scores_rating',
 'review_scores_accuracy',
 'review_scores_cleanliness',
 'review_scores_checkin',
 'review_scores_communication',
 'review_scores_location',
 'review_scores_value',
 'license',
 'instant_bookable',
 'calculated_host_listings_count',
 'calculated_host_listings_count_entire_homes',
 'calculated_host_listings_count_private_rooms',
 'calculated_host_listings_count_shared_rooms',
 'reviews_per_month']
```

The 7 Features Selected for the Analysis:

1. neighbourhood_cleansed
2. availability_30
3. room_type
4. accommodates
5. beds
6. price
7. review_scores_rating

[675]:
```
#Assigning these features to df_1
df_1 = df_airbnb[['neighbourhood_cleansed', 'availability_30',
         'room_type', 'accommodates', 'beds',
         'price', 'review_scores_rating']]
print('Columns used in the analysis are:')
print(df_1.columns)
```

```
Columns used in the analysis are:
Index(['neighbourhood_cleansed', 'availability_30', 'room_type',
       'accommodates', 'beds', 'price', 'review_scores_rating'],
      dtype='object')
```

[556]:
```
df_1.head(10)
```

[556]:

	neighbourhood_cleansed	availability_30	room_type	accommodates	beds	price	review_scores_rating
0	Oostelijk Havengebied - Indische Buurt	1	Private room	2	2.0	$49.00	4.89
1	Centrum-Oost	0	Private room	2	1.0	$106.00	4.44
2	Centrum-West	0	Private room	2	1.0	$136.00	4.94
3	Centrum-West	4	Private room	2	1.0	$75.00	4.88
4	Centrum-Oost	0	Private room	1	1.0	$55.00	4.79
5	Centrum-Oost	0	Entire home/apt	4	2.0	$240.00	4.72
6	Centrum-West	1	Entire home/apt	3	2.0	$245.00	4.92
7	Centrum-West	3	Private room	2	1.0	$124.00	4.87
8	Centrum-Oost	0	Entire home/apt	2	1.0	$250.00	4.95
9	Zuid	0	Entire home/apt	4	2.0	$149.00	4.86

[581]:
```
df_1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6893 entries, 0 to 6892
Data columns (total 7 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   neighbourhood_cleansed  6893 non-null   object
 1   availability_30         6893 non-null   int64
 2   room_type               6893 non-null   object
 3   accommodates            6893 non-null   int64
 4   beds                    6792 non-null   float64
 5   price                   6893 non-null   object
 6   review_scores_rating    6247 non-null   float64
dtypes: float64(2), int64(2), object(3)
memory usage: 377.1+ KB
```

price had an 'object' data type with dollar and comma signs. Thus, it was transformed into a float value.

[1306]:
```
#checking if values >1000 are separated by commas
df_1.price[df_1.price.str.contains(',')]
```

[1306]:
```
37      $1,236.00
1400    $6,603.00
1679    $1,160.00
1773    $1,490.00
2244    $2,000.00
2372    $1,003.00
3013    $1,657.00
3090    $1,350.00
3134    $1,000.00
3429    $1,078.00
3925    $7,900.00
4016    $1,600.00
4273    $1,200.00
4912    $1,023.00
4936    $1,045.00
4971    $1,399.00
5105    $1,500.00
5191    $2,350.00
5271    $1,037.00
5275    $1,520.00
5276    $1,484.00
5639    $1,000.00
5819    $2,820.00
5842    $1,100.00
6169    $1,107.00
6200    $1,343.00
6306    $1,000.00
6800    $1,000.00
Name: price, dtype: object
```

[676]:
```
#price was a string with '$' and ',' if price was over $1,000
#Thus, we converted it into a float.
df_1['price'] = df_1['price'].str.replace('$', '', regex=False).str.
 ↪replace(',', '')
df_1['price']= df_1['price'].str.strip().astype(float)
```

```
/tmp/ipykernel_134/3730194115.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

```
  df_1['price'] = df_1['price'].str.replace('$', '',
regex=False).str.replace(',', '')
/tmp/ipykernel_134/3730194115.py:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_1['price']= df_1['price'].str.strip().astype(float)
```

[583]:
```
# renaming the column price so that it is easier for users to understand that
# the price is stated in dollars

#similarly, renaming 'neighbourhood_cleansed' to 'neighbourhood_group'
#since it actually refers to Amsterdam's Neighbourhood Groups
#this will make it more intuitive for the users
df_1 = df_1.rename(columns = {'price': 'price_dollars',
                          'neighbourhood_cleansed': 'neighbourhood_group'})
```

[1309]:
```
#re-chekcing the head after remaning
df_1.head()
```

[1309]:

	neighbourhood_group	availability_30	room_type	accommodates	beds	price_dollars	review_scores_rating
0	Oostelijk Havengebied - Indische Buurt	1	Private room	2	2.0	49.0	4.89
1	Centrum-Oost	0	Private room	2	1.0	106.0	4.44
2	Centrum-West	0	Private room	2	1.0	136.0	4.94
3	Centrum-West	4	Private room	2	1.0	75.0	4.88
4	Centrum-Oost	0	Private room	1	1.0	55.0	4.79

[1310]:
```
#checking descriptives
df_1.describe()
```

[1310]:

	availability_30	accommodates	beds	price_dollars
count	6893.000000	6893.000000	6792.000000	6893.000000
mean	3.430292	2.983461	1.944641	218.487306
std	6.189709	1.483226	1.617584	192.576328
min	0.000000	0.000000	1.000000	0.000000
25%	0.000000	2.000000	1.000000	125.000000
50%	0.000000	2.000000	1.000000	180.000000
75%	4.000000	4.000000	2.000000	264.000000

```
max          30.000000     16.000000     33.000000     7900.000000

       review_scores_rating
count           6247.000000
mean               4.812651
std                0.286552
min                0.000000
25%                4.750000
50%                4.890000
75%                5.000000
max                5.000000
```

[584]:
```
#Checking whether there are any duplicate rows
rows_duplicated = df_1[df_1.duplicated()]
rows_duplicated
print('Total of duplicate rows in the dataset (across all columns): ',
      df_1.duplicated().sum())
```

[584]:
```
         neighbourhood_group  availability_30        room_type  accommodates  \
804                Westerpark                0  Entire home/apt             2
947                  Oud-Oost                0  Entire home/apt             2
1055                Oud-Noord                0  Entire home/apt             2
1143                     Zuid                0  Entire home/apt             4
1374  De Baarsjes - Oud-West                0  Entire home/apt             2
…                         …              …                …           …
6595         Watergraafsmeer                0     Private room             2
6675         Watergraafsmeer                0     Private room             2
6679         Watergraafsmeer                0     Private room             2
6796              Noord-Oost                0  Entire home/apt             4
6870              Noord-Oost                0  Entire home/apt             2

      beds  price_dollars  review_scores_rating
804    1.0          150.0                  4.82
947    1.0          165.0                  4.95
1055   1.0           99.0                  4.88
1143   2.0          300.0                  5.00
1374   1.0          149.0                  5.00
…    …              …                      …
6595   2.0           71.0                   NaN
6675   2.0           71.0                   NaN
6679   2.0           71.0                   NaN
6796   3.0          150.0                  4.86
6870   1.0          150.0                  5.00

[251 rows x 7 columns]

Total of duplicate rows in the dataset (across all columns):  251
```

[585]:
```
#dropping duplicate values
df_1 = df_1.drop_duplicates(subset = None, keep = 'first',
                            ignore_index = False)
```

[8]:
```
# Re-checking whether all duplicates have been removed
print('Total of duplicate rows in the dataset (across all columns): ',
      df_1.duplicated().sum())
```

```
Total of duplicate rows in the dataset (across all columns):  0
```

[1315]:
```
#Checking the missing values for each dataset column
df_1.isna().sum()
```

[1315]:
```
neighbourhood_group       0
availability_30           0
room_type                 0
accommodates              0
beds                     98
price_dollars             0
review_scores_rating    624
dtype: int64
```

Missing values were deleted instead of replacing them with central central tendency measures to decrease the bias in the synthetic data generation process.

[587]:
```
#Deleting all missing values so that only real values are kept for creating␣
 ↪synthetic data
df_1.dropna(inplace = True)
```

[588]:
```
# Converting the 'beds' and 'price_dollars columns to integer type
#since there are solely integer values represented as floats
#so that full numbers are provided upon synthetic data creation
#after dropping NaN Values
df_1['beds'] = df_1['beds'].astype(int)
```

ii) Selecting 1,000 instances of Original Data

[589]:
```
#Before selecting 1,000 rows from the dataset for the creation of synthetic␣
 ↪data,
#the dataset was shuffled to ensure that no observations that were potentially
#following a pattern are affected upon the split, generating synthetic data
#that is less representative of certain features

#stating a random state so that the values are always the same every time we␣
 ↪run the analysis
#frac = 1 -> shuffling the entire dataset
shuffled_df = df_1.sample(frac = 1, random_state = 28)
```

```
#OR: Do it straight away
#df_used = df_1.sample(n = 1000, random_state = 28)
```

[590]:
```
#Selecting 1,000 rows for the further creation of synthetic data
#with a step of 2
df_used = shuffled_df[1000:3000:2]
#or: df_used = shuffled_df.head(1000)
df_used.shape
```

[590]: (1000, 7)

[566]:
```
#Checking the first 5 rows for the shuffled dataset
df_used.head()
```

[566]:

	neighbourhood_group	availability_30	room_type	accommodates	beds	price_dollars	review_scores_rating
5154	Zuid	22	Private room	2	1	371.0	4.33
3044	De Pijp - Rivierenbuurt	10	Entire home/apt	4	2	294.0	4.55
759	De Baarsjes - Oud-West	0	Entire home/apt	4	3	450.0	4.93
2588	Centrum-West	8	Entire home/apt	4	3	465.0	4.72
5001	Centrum-West	5	Entire home/apt	2	1	99.0	4.75

[43]:
```
#comparing the head of the shuffled dataset with the non-shuffled one
df_1.head()
# -> as can be seen, the shuffled dataset head has more variety of values
```

[43]:

	neighbourhood_group	availability_30	room_type	accommodates	beds	price_dollars	review_scores_rating
0	Oostelijk Havengebied - Indische Buurt	1	Private room	2	2	49.0	4.89
1	Centrum-Oost	0	Private room	2	1	106.0	4.44
2	Centrum-West	0	Private room	2	1	136.0	4.94
3	Centrum-West	4	Private room	2	1	75.0	4.88
4	Centrum-Oost	0	Private room	1	1	55.0	4.79

[44]:
```
#Checking the count and data types for all variables
df_used.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1000 entries, 5154 to 2965
Data columns (total 7 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   neighbourhood_group   1000 non-null   object
 1   availability_30       1000 non-null   int64
 2   room_type             1000 non-null   object
 3   accommodates          1000 non-null   int64
 4   beds                  1000 non-null   int64
 5   price_dollars         1000 non-null   float64
 6   review_scores_rating  1000 non-null   float64
dtypes: float64(2), int64(3), object(2)
memory usage: 62.5+ KB
```

4.3 Question 6 - Creating a Function for Synthetic Data Generation

Using CTGAN (Conditional Tabular GAN)

CTGAN is a type of Generative Adversarial Network (GAN) that is specifically designed to generate synthetic tabular data that is similar to the original data. It works by learning the joint probability distribution of the data, which allows it to generate new data points that are statistically similar to the original data (Chalé & Bastian, 2021). The CTGAN algorithm utilised in the function would model the joint distribution of the original inputted data and generate synthetic samples based on the joint distribution of that data.

1st Hyperparameter: #Epochs

Epochs determine the number of times the model is presented with the whole dataset during the training process. Upon setting a too low epochs number, the model might be unable to capture the underlying data structure, thus, underfitting. Conversely, a substantially high epochs number, the model could overfit to training data. Thus, upon increasing the epochs' number, the model's performance would keep improving until a certain extent, when it would start overfitting.

CTGAN's accuracy can be improved by setting a higher #epochs. Therefore, to better grasp the original data's distribution and correlations, we left the number of epochs to the CTGAN default of 300 (and, therefore, did not specified them in the function), which resulted in the best synthetic data quality.

It has to be noted that despite a higher number of epochs leading to better accuracy, it requires high computational power, leading to intricacies of creating synthetic data for a larger number of instances (e.g., 10,000 required in Question 8(C)).

Increasing the #rows increased the model's compexity, leading to loger training times and convergence issues.

Therefore, 300 epochs were chosen for the purpose of the assessment, which allowed performing the analysis while improving performance and not overfitting to the training data. Additionally, a

higher number of epochs set to the function led to inabilities to generate 10,000 instances on the CPU.

2nd Hyperparameter: Batch Size

The Batch Size is the number of instances rows fed to the model at once and is another Hyperparameter, impacting the Generator's performance.. in the CTgAN model, the batch size is to be perfectly divisible by the number of rows.

Benefits of increasing the batch size:

A higher batch size could increase model performance (higher convergence during training). Thus, increasing the batch size could result in higher convergence stability and less noisy gradient estimates. Leading to the generator's convergence improvement and smoother optimisation process, it could result in higher quality of the generated synthetic data. However, this was not the case for our data.

Disadvantages of increasing the batch size:

A larger batch size requires higher computational power due to higher memory required for larger batch size processing. Additionally, increasing the batch size could have both positive and negative effects on the synthetic data. The latter is resulting from the generator overfitting to the training data, which results in the synthetic data failing to capture the underlying structure of the data. This resulted upon increasing the batch size over 50.

Thus, given the abovementioned trade-offs, after experimenting with different parameters, the parameters that could be processed given the available resources while maximising the quality of synthetic data were selected.

[591]:
```
# Creating a function with 3 input arguments
#1)data -> the input data we want to utilise to generate synthetic data␣
 ↪instances
#2)n_rows -> number of rows we want to generate
#setting 1,000 as default, so that we can automatically apply the function in
#Question 7(A),
#only adding our input data



#2)random_seed=25 -> setting a default random state to ensure reproducibility␣
 ↪and consistency of the
#results (i.e., same synthetic data instances generated)

#This is crucial upon building predictive models, since it allows consistent and
#reproducble comparison of the different models' performance and hyperprameter␣
 ↪settings

#However, it should be noted that while setting a random seed ensures that
#the CTGAN model is initialized with the same random parameters and samples
#from the same learned distributions upon calling the function every time,
# some randomness in the sampling process will be still involved.
```

```
def create_synthetic_data(data, n_rows=1000, random_seed=25):
    np.random.seed(random_seed)
    ctgan = CTGAN(batch_size=50)
    #including categorical data
    ctgan.fit(data, discrete_columns=list(data.select_dtypes(include='object').
 ↪columns))
    synthetic_data = ctgan.sample(n_rows)

    for col in synthetic_data.select_dtypes(include='number').columns:
        #since our data contains solely positive values
        #taking the absolute values of the data to ensure only positive values␣
 ↪are returned
        #rounding to 2 decimal places (max decimal places present in our data)

        synthetic_data[col] = synthetic_data[col].abs().round(2).
 ↪astype('float32')

    return synthetic_data
```

4.4 Question 7: Generating Synthetic Data and Assessing the Correlations

4.4.1 Question 7 (A): Generating 1,000 Synthetic Data Instances using the created Function

[592]:
```
#Generating 1000 instances of Synthetic Data by applying the created function␣
 ↪to the original data
#(1000 instances used as a default for the 2nd argument (n_rows=1000))
#and the default random_seed=25 for the 3rd argument

synthetic_df_airbnb = create_synthetic_data(df_used)
```

[593]:
```
#checking that the generated synthetic data has the desired number of␣
 ↪rows(1000) and the same number
#of columns as the original dataset (7)
print('The Generated Synthetic Dataset has {0} rows and {1} columns'.
 ↪format(len(synthetic_df_airbnb),

 ↪len(synthetic_df_airbnb.columns)))
```

```
The Generated Synthetic Dataset has 1000 rows and 7 columns
```

[594]:
```
#checking the first 5 rows of the synthetic dataset
df_used.head()
```

[594]:

	neighbourhood_group	availability_30	room_type	accommodates	\
5154	Zuid	22	Private room	2	

```
3044  De Pijp - Rivierenbuurt                10  Entire home/apt             4
759    De Baarsjes - Oud-West                 0  Entire home/apt             4
2588              Centrum-West                8  Entire home/apt             4
5001              Centrum-West                5  Entire home/apt             2

      beds  price_dollars  review_scores_rating
5154     1          371.0                  4.33
3044     2          294.0                  4.55
759      3          450.0                  4.93
2588     3          465.0                  4.72
5001     1           99.0                  4.75
```

[595]:
```
#Comparing it with the first 5 rows of the original dataset
synthetic_df_airbnb.head()
```

[595]:
```
      neighbourhood_group  availability_30        room_type  accommodates  \
0            Centrum-Oost              0.0  Entire home/apt           2.0
1               Oud-Noord              0.0     Private room           4.0
2             Noord-West              12.0     Private room           3.0
3  Buitenveldert - Zuidas              0.0     Private room           4.0
4  De Baarsjes - Oud-West              0.0       Hotel room           2.0

   beds  price_dollars  review_scores_rating
0   1.0     555.750000                  4.83
1   1.0     107.690002                  4.71
2   2.0     150.139999                  4.61
3   2.0     451.529999                  5.01
4   1.0     220.740005                  4.44
```

4.4.2 Question 7 (B): Plotting the Distributions of Original VS Synthetic Data

[596]:
```
# Version 1: plotting distributions on the same Graph

def plot_distributions(original_data, synthetic_data):
    n_cols = original_data.shape[1]
    fig, axs = plt.subplots(nrows = 1, ncols = n_cols, figsize = (40, 7))

    for i in range(n_cols):
        axs[i].hist(original_data.iloc[:, i], alpha = 0.5, label = 'Original')
        axs[i].hist(synthetic_data.iloc[:, i], alpha = 0.5, label = 'Synthetic')
        axs[i].set_title(original_data.columns[i])
        axs[i].legend()

        # Rotating x-axis labels
        axs[i].tick_params(axis = 'x', labelrotation = 90)
    plt.show();
```

```
#Applyting the Function to the Original and Synthetic Data
plot_distributions(df_used, synthetic_df_airbnb)
```

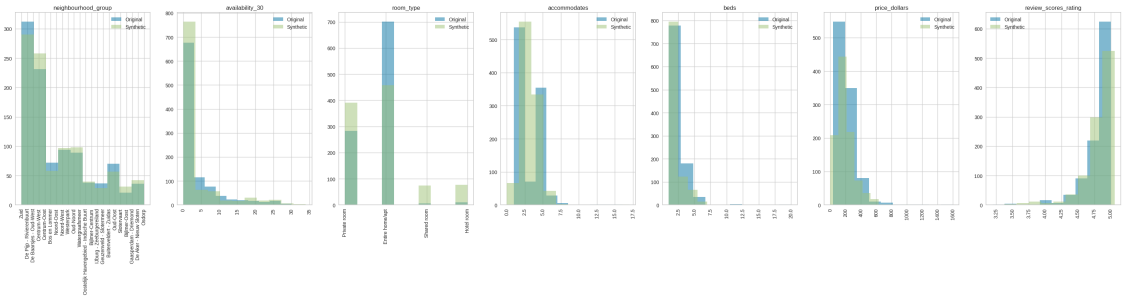

Figure 4: Comparison in the Original and Synthetic Data Joint DIstributions

[597]:
```
# Version 2: plotting distributions separately, side by side

#gridspec_kw = {'hspace': 2} - specifying the spacing between subplots
#taking a space of 2 since some labels were long
fig, ax = plt.subplots(ncols = 2, nrows = 7, figsize = (10, 40),
                       sharey = True, gridspec_kw = {'hspace': 2})

for i in range(0, 7):
    # plotting the distributions in the original data
    ax[i][0].hist(df_used.iloc[:, i], alpha = 0.5,
                  color = 'blue', label = 'Original')
    ax[i][0].set_title(df_used.columns[i])
    ax[i][0].tick_params(axis = 'x', labelrotation = 85)

    # plotting the distributions in the synthetic data
    ax[i][1].hist(synthetic_df_airbnb.iloc[:, i], alpha = 0.5,
                  color = 'red', label = 'Synthetic')
    ax[i][1].set_title(synthetic_df_airbnb.columns[i])
    ax[i][1].tick_params(axis = 'x', labelrotation = 85)


#Showing the Plot
plt.show();
```

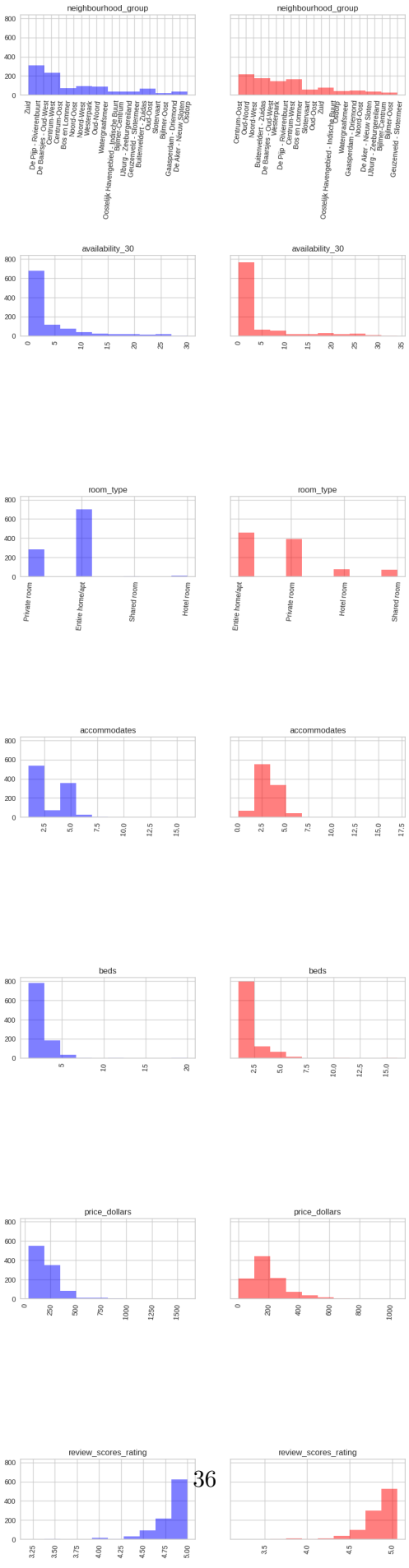
neighbourhood_group
availability_30
room_type
accommodates
beds
price_dollars
review_scores_rating

Figure 5: Distributions in Original and Synthetic Data The distribution plots of the original and synthetic data are depicted above. We can see that the distributions for 'neighbourhood_group', 'availability_30', and 'review_scores_rating'are generally similar in shape. However, the distribution plots for 'price_dollars', 'beds', 'accomodates', and 'room_type' have slight differences.

Moreover, major distribution difference as seen in specific categories such as 'Entire home/apt'could indicate a difference that the synthetic data is not capturing the same proportion of each category in the original data. The distribution difference could also indicate a difference in the relationships between the variable and other variables in the dataset.

4.4.3 Question 7 (C): Correlations

[239]:
```
def plot_correlations(original_data, synthetic_data):
    # Computing the correlation matrices for the original and synthetic data
    corr_original = df_used.corr()
    corr_synthetic = synthetic_df_airbnb.corr()

    # Computing the difference in correlations between the original and␣
 ↪synthetic data
    corr_diff = corr_synthetic - corr_original

    # Creating a figure with subplots for each heatmap
    fig, axs = plt.subplots(nrows = 1, ncols = 3, figsize = (15, 5))

    # Plotting the heatmap of original data correlations
    sns.heatmap(corr_original, cmap = "coolwarm", center = 0, annot = True, ax␣
 ↪= axs[0])
    axs[0].set_title("Correlations in Original Data")

    # Plotting the heatmap of synthetic data correlations
    sns.heatmap(corr_synthetic, cmap = "coolwarm", center = 0, annot = True, ax␣
 ↪= axs[1])
    axs[1].set_title("Correlations in Synthetic Data")

    # Plotting the heatmap of difference in correlations
    sns.heatmap(corr_diff, cmap = "coolwarm", center = 0, annot = True, ax =␣
 ↪axs[2])
    axs[2].set_title("Difference in Correlations")

    # Showing the plot
    plt.show();
plot_correlations(df_used, synthetic_df_airbnb)
```

```
/tmp/ipykernel_134/1976833037.py:3: FutureWarning: The default value of
```

```
numeric_only in DataFrame.corr is deprecated. In a future version, it will
default to False. Select only valid columns or specify the value of numeric_only
to silence this warning.
  corr_original = df_used.corr()
/tmp/ipykernel_134/1976833037.py:4: FutureWarning: The default value of
numeric_only in DataFrame.corr is deprecated. In a future version, it will
default to False. Select only valid columns or specify the value of numeric_only
to silence this warning.
  corr_synthetic = synthetic_df_airbnb.corr()
```

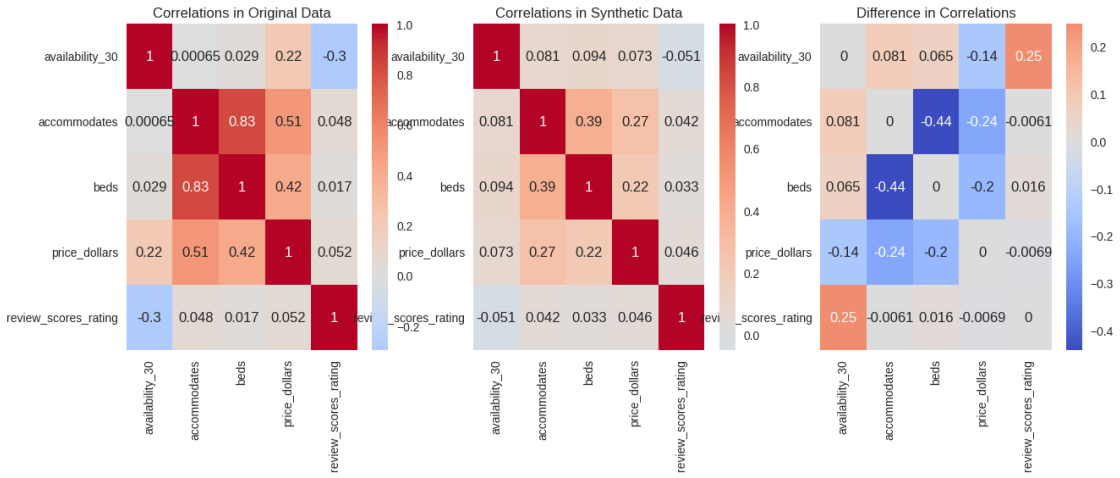

Figure 6: Correlations in Original and Synthetic Data, and Correlation Differences before OHE

To assess the similarity between the joint distributions of the original and synthetic data, we computed the correlation heatmaps for both the original and synthetic data before and after One-Hot-Encoding(OHE). We did so because OHE rresulted in 31 variables, intricating the evaluation process. Thus, the correlations between continuous variables will be evaluated first, proceeding with the correlations with the dummy variables' correlations after OHE.

We then visualized the difference between the correlations with another heatmap which indicates that the joint distributions of the original and synthetic data generally follow the same pattern. The correlation of the datasets are comparable with similar correlations, however there is a stark difference with the .44 decrease with beds and accommodates and a .25 increase with availability_30 and review_scores_rating. These larger differences may affect the overall results of our models when training with synthetic data.

The coefficients in the orginal data show stronger correlations, for example .83 for beds and accomodates but the synthetic data shows a weaker correlation of .39. Therefore, the function failed to capture the underlying structure of the data. This could be due to a number of reasons such as overfitting or the occurence of sampling variation.

Utilising One-Hot-Encoding to demonstrate Correlations with the Categorical Data

[598]:
```
# OHE for original data to perform the Correlation Matrix

# performing OHE (dummy encoding) on categorical data
#encoded_data = pd.get_dummies(df_used[['neighbourhood_group', 'room_type']],␣
 ↪drop_first = True)
encoded_data = pd.get_dummies(df_used[['neighbourhood_group', 'room_type']])

# merging the encoded data back into the original dataframe
df_used = pd.concat([df_used, encoded_data],
                    axis = 1).drop(df_used[['neighbourhood_group',
                                            'room_type']], axis = 1)

df_used.head()
```

[598]:

	availability_30	accommodates	beds	price_dollars	\
5154	22	2	1	371.0	
3044	10	4	2	294.0	
759	0	4	3	450.0	
2588	8	4	3	465.0	
5001	5	2	1	99.0	

	review_scores_rating	neighbourhood_group_Bijlmer-Centrum	\
5154	4.33	0	
3044	4.55	0	
759	4.93	0	
2588	4.72	0	
5001	4.75	0	

	neighbourhood_group_Bijlmer-Oost	neighbourhood_group_Bos en Lommer	\
5154	0	0	
3044	0	0	
759	0	0	
2588	0	0	
5001	0	0	

	neighbourhood_group_Buitenveldert - Zuidas	\
5154	0	
3044	0	
759	0	
2588	0	
5001	0	

	neighbourhood_group_Centrum-Oost	…	neighbourhood_group_Oud-Noord	\
5154	0	…	0	
3044	0	…	0	
759	0	…	0	
2588	0	…	0	

```
5001                                0  …                                   0

      neighbourhood_group_Oud-Oost  neighbourhood_group_Slotervaart  \
5154                             0                                0
3044                             0                                0
759                              0                                0
2588                             0                                0
5001                             0                                0

      neighbourhood_group_Watergraafsmeer  neighbourhood_group_Westerpark  \
5154                                    0                               0
3044                                    0                               0
759                                     0                               0
2588                                    0                               0
5001                                    0                               0

      neighbourhood_group_Zuid  room_type_Entire home/apt  \
5154                         1                          0
3044                         0                          1
759                          0                          1
2588                         0                          1
5001                         0                          1

      room_type_Hotel room  room_type_Private room  room_type_Shared room
5154                     0                       1                     0
3044                     0                       0                     0
759                      0                       0                     0
2588                     0                       0                     0
5001                     0                       0                     0

[5 rows x 31 columns]
```

[599]:
```
# OHE for synthetic data to perform the Correlation Matrix

# performing OHE (dummy encoding) on categorical data
encoded_synth = pd.get_dummies(synthetic_df_airbnb[['neighbourhood_group',
 ↪'room_type']])

# merge the encoded data back into the original dataframe
synthetic_df_airbnb = pd.concat([synthetic_df_airbnb, encoded_synth],
                                axis = 1).
 ↪drop(synthetic_df_airbnb[['neighbourhood_group',
                                                                 ␣
 ↪'room_type']], axis = 1)
synthetic_df_airbnb.head()
```

[599]:

	availability_30	accommodates	beds	price_dollars	review_scores_rating	\
0	0.0	2.0	1.0	555.750000	4.83	
1	0.0	4.0	1.0	107.690002	4.71	
2	12.0	3.0	2.0	150.139999	4.61	
3	0.0	4.0	2.0	451.529999	5.01	
4	0.0	2.0	1.0	220.740005	4.44	

	neighbourhood_group_Bijlmer-Centrum	neighbourhood_group_Bijlmer-Oost	\
0	0	0	
1	0	0	
2	0	0	
3	0	0	
4	0	0	

	neighbourhood_group_Bos en Lommer	\
0	0	
1	0	
2	0	
3	0	
4	0	

	neighbourhood_group_Buitenveldert - Zuidas	\
0	0	
1	0	
2	0	
3	1	
4	0	

	neighbourhood_group_Centrum-Oost	…	neighbourhood_group_Oud-Noord	\
0	1	…	0	
1	0	…	1	
2	0	…	0	
3	0	…	0	
4	0	…	0	

	neighbourhood_group_Oud-Oost	neighbourhood_group_Slotervaart	\
0	0	0	
1	0	0	
2	0	0	
3	0	0	
4	0	0	

	neighbourhood_group_Watergraafsmeer	neighbourhood_group_Westerpark	\
0	0	0	
1	0	0	
2	0	0	
3	0	0	

```
4                                   0                                   0

   neighbourhood_group_Zuid  room_type_Entire home/apt  room_type_Hotel room  \
0                         0                          1                     0
1                         0                          0                     0
2                         0                          0                     0
3                         0                          0                     0
4                         0                          0                     1

   room_type_Private room  room_type_Shared room
0                       0                     0
1                       1                     0
2                       1                     0
3                       1                     0
4                       0                     0

[5 rows x 31 columns]
```

i) Creating a Correlation Matrix for the Original Data

[600]:
```
# Calculating the correlation matrix for the original  data

# Computing the correlation matrix
original_corr = df_used.corr()

# Creating a mask to hide the upper triangle
mask = np.triu(np.ones_like(original_corr, dtype = bool))

# Plotting the correlation matrix
fig, ax = plt.subplots(figsize = (80, 70))
sns.heatmap(original_corr, cmap = 'coolwarm', annot = True, fmt = '.2f',
            square = True, mask = mask, cbar = False, ax = ax)
plt.title('Correlation Matrix of the Original Airbnb Data (Lower Triangle␣
 ↪Only)')
plt.show();
```

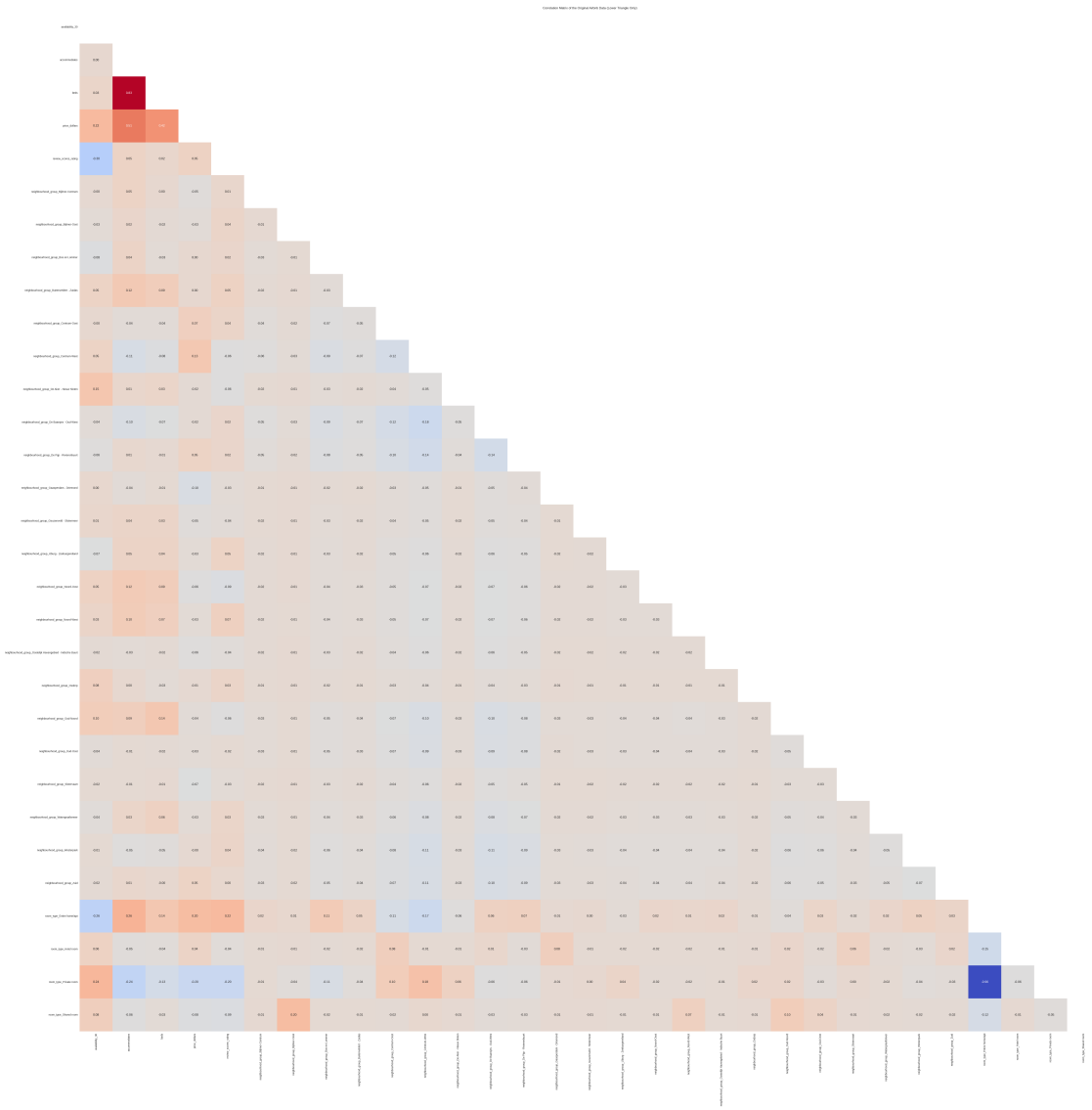

Figure 7: Correlation Heatmap for Original Data

ii) Creating a Correlation Matrix for the Synthetic Data

[601]:
```
# Calculating the correlation matrix for the synthetic data
synthetic_corr = synthetic_df_airbnb.corr()
# Creating a mask to hide the upper triangle
mask = np.triu(np.ones_like(synthetic_corr, dtype = bool))

# Plotting the synthetic data correlation matrix
fig, ax = plt.subplots(figsize = (80, 70))
sns.heatmap(synthetic_corr, cmap = 'coolwarm', annot = True,
            mask = mask, fmt = '.2f', square = True)
```

```
plt.title('Correlation Matrix of the Synthetic Airbnb Data')
plt.show();
```

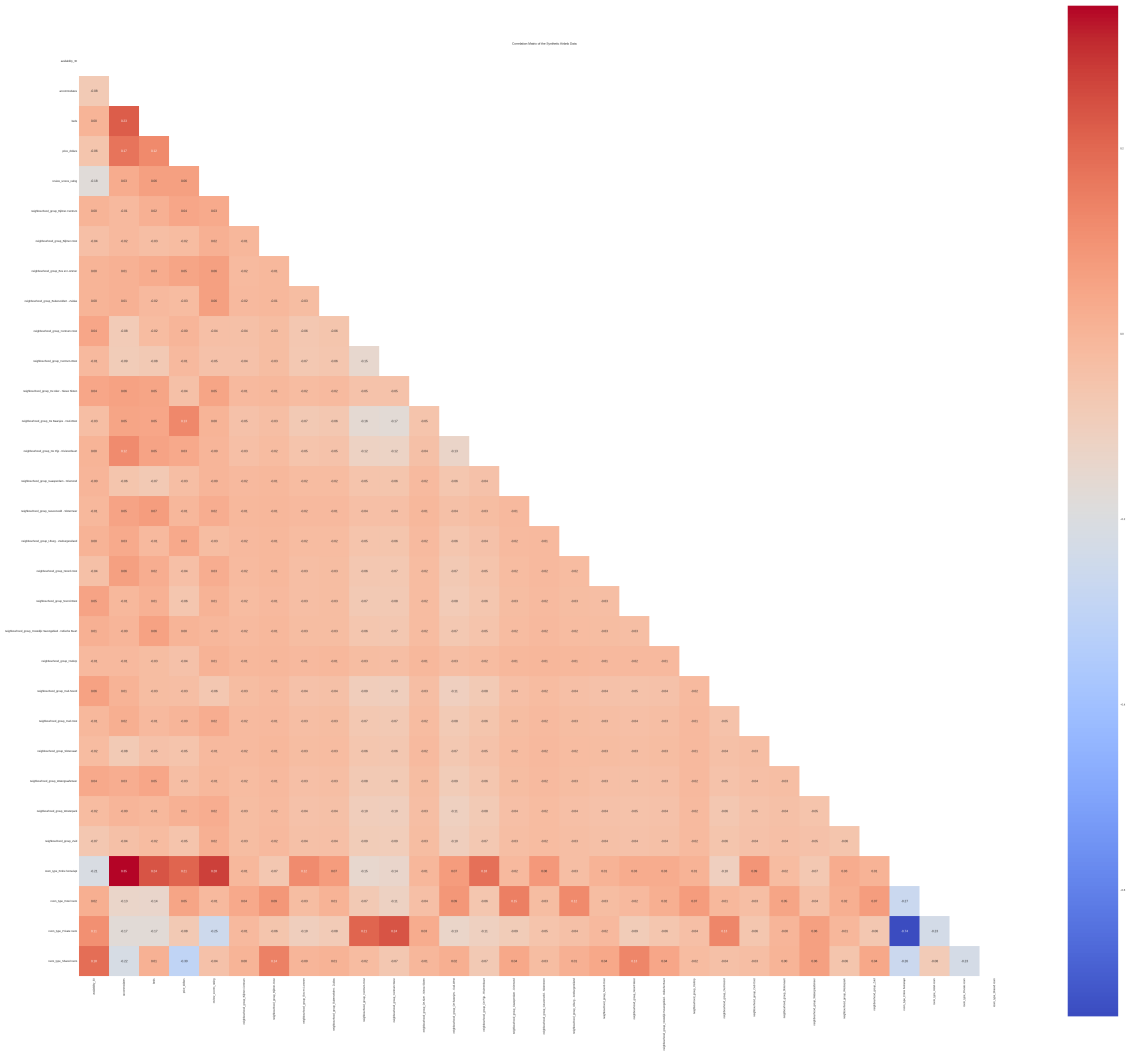

Figure 8: Correlation Heatmap for Synthetic Data

iii) Creating a Correlation Matrix for the differences between the Original and Synthetic Data

[604]:
```
# Assigning the difference between the original and synthetic data correlation␣
↪matrices
corr_diff = synthetic_corr - original_corr

# Creating a mask to hide the upper triangle
mask = np.triu(np.ones_like(corr_diff, dtype=bool))

# Plotting the difference in correlation matrices
```

```
fig, ax = plt.subplots(figsize = (80, 70))
sns.heatmap(corr_diff, cmap = 'coolwarm', annot = True, fmt = '.2f',
            square = True, mask = mask, cbar = False, ax = ax)
plt.title('Difference between the Correlation Matrices of Original and␣
  ↪Synthetic Airbnb Data (Lower Triangle Only)')
plt.show();
```

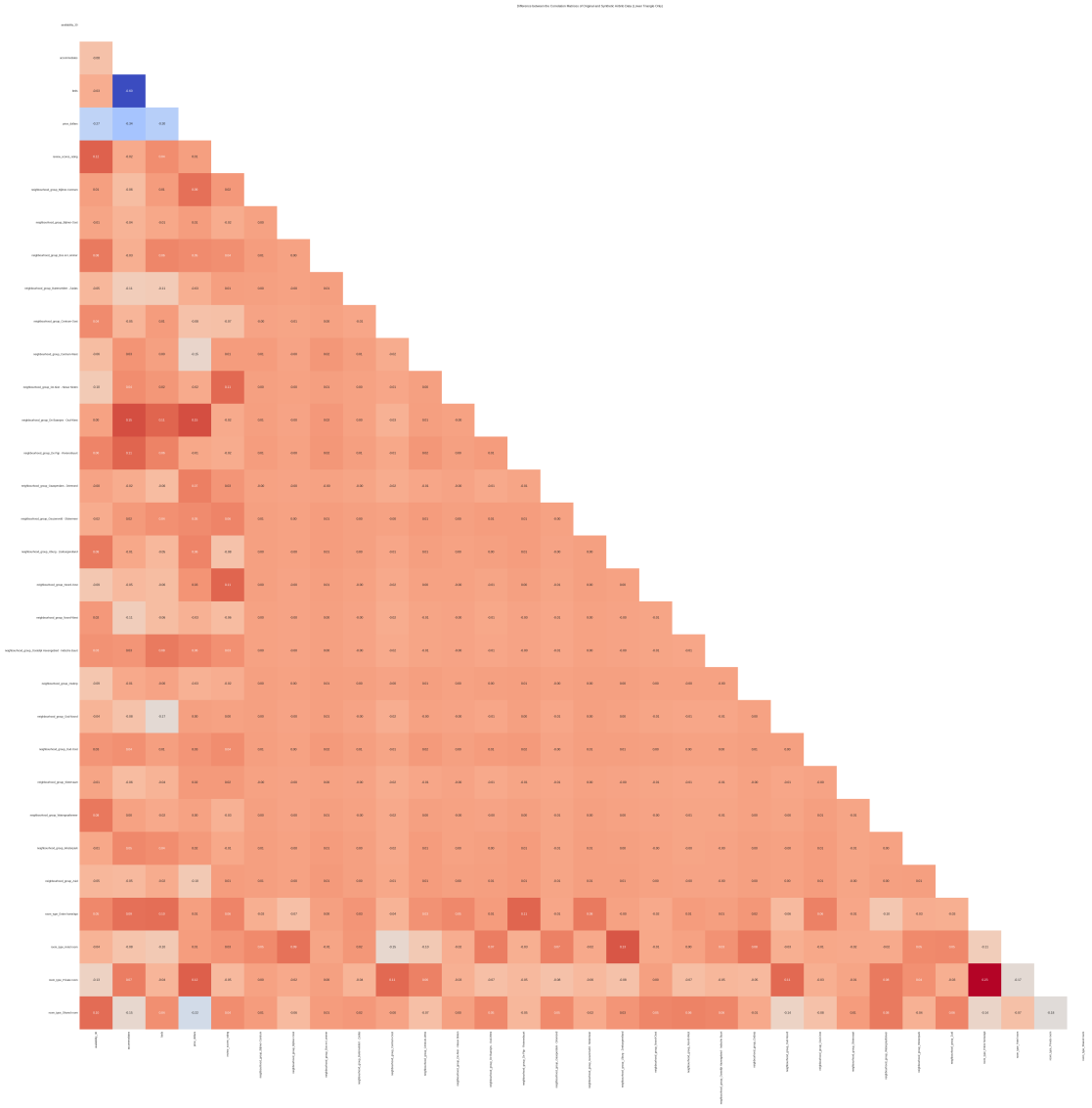

Figure 9: Correlation Differences

The highest correlation difference is beds and accomodates with a negative correlation of -.60. However, in the original and synthetic correlation heat maps this was the highest positive correlation. The difference in the correlations could be due to the fact that it failed to capture the associations between the variables. No other strong correlation differences were noticed after the

One-Hot-Encoding.

4.5 Question 8: Building Predictive Models

4.5.1 Models:

1) Ridge Regression

2) Decision Tree Regression

3) Neural Network Regression

4.5.2 Accuracy Metrics utilised for Model Comparison:

1) R-Squared(R2)/ Coefficient of Determination

2) Mean Squared Error (MSE)

3) Mean Absolute Error (MAE)

4.5.3 Sampling the Data for the Anlaysis

Steps:

1) Randomly Sampling 1,000 instances of the Original Data (Cleaned in Q5) to ensure no Crossover between the Training Data (A) and Testing Data (A-C)

2) One-Hot Encoding the Data before dividing it into the Training Set(A) and Teset Set (A-C) to avoid performing this operation twice(separately for the Training SEt (A) and Test Set (A-C))

3) Selecting 500 instances from the 1,000 instances of Randomly Sampled Data (in Step 1) for Training Set (A) and 500 instances for Test Set (A-C). Performing a train-test split utilising a random state to ensure results reproducibility.

4) Setting the IVs and DV for the Test Set (A-C) and saving the dataset to CSV

Step 1: Random Sampling from the Cleaned Data

[605]:
```
#randomly selecting 1,000 instances from the original dataframe to be used in␣
 ↪the analysis
#and then splitting it into 50/50 train and test set
#so that there are no instances that are accidentally attributed both to the␣
 ↪test set
#and train test(a)

#stating a random state so that the cases are always the same every time we run␣
 ↪the code
selected = df_1.sample(n = 1000, random_state = 25)
```

[606]:
```
print('The selected dataset has {0} rows and {1} columns'.format(len(selected),
                                                                 len(selected.
 ↪columns)))
```

```
The selected dataset has 1000 rows and 7 columns
```

Step 2: One-Hot-Encoding for the Sampled Data

Before Selecting the Training and Test Instances, OHE was performed on the 1,000 selected data to avoid performing this operation on the Training Set (A) and Test Set (A-C) separately.

[607]:
```
# OHE for the sampled data

# performing OHE (dummy encoding) on the categorical columns
encoded_data_sel = pd.get_dummies(selected[['neighbourhood_group',␣
 ↪'room_type']])

# merging the encoded data back into the original dataframe
selected = pd.concat([selected, encoded_data_sel],
                     axis = 1).drop(selected[['neighbourhood_group',
                                              'room_type']], axis = 1)
selected.head()
```

[607]:

	availability_30	accommodates	beds	price_dollars	\
4519	17	4	3	131.0	
2102	12	2	1	116.0	
2668	0	2	2	164.0	
1082	25	4	3	743.0	
431	0	2	1	190.0	

	review_scores_rating	neighbourhood_group_Bijlmer-Centrum	\
4519	4.38	0	
2102	4.82	0	
2668	5.00	0	
1082	5.00	0	
431	4.94	0	

	neighbourhood_group_Bijlmer-Oost	neighbourhood_group_Bos en Lommer	\
4519	0	0	
2102	0	0	
2668	0	0	
1082	0	0	
431	0	0	

	neighbourhood_group_Buitenveldert - Zuidas	\
4519	0	
2102	0	
2668	0	
1082	0	
431	0	

	neighbourhood_group_Centrum-Oost	…	neighbourhood_group_Oud-Noord	\
4519	0	…	1	
2102	0	…	0	

```
2668                                  0  …                                  0
1082                                  0  …                                  0
431                                   0  …                                  0

      neighbourhood_group_Oud-Oost  neighbourhood_group_Slotervaart  \
4519                             0                                0
2102                             0                                0
2668                             0                                0
1082                             0                                0
431                              0                                0

      neighbourhood_group_Watergraafsmeer  neighbourhood_group_Westerpark  \
4519                                    0                               0
2102                                    0                               0
2668                                    0                               0
1082                                    0                               0
431                                     0                               0

      neighbourhood_group_Zuid  room_type_Entire home/apt  \
4519                         0                          0
2102                         0                          0
2668                         0                          0
1082                         0                          1
431                          1                          1

      room_type_Hotel room  room_type_Private room  room_type_Shared room
4519                     0                       1                      0
2102                     0                       1                      0
2668                     0                       1                      0
1082                     0                       0                      0
431                      0                       0                      0

[5 rows x 31 columns]
```

[608]:
```
print('After performing One-Hot_encoding, the selected dataset has {0} rows and␣
 ↪{1} columns'.format(len(selected),
                                                                    len(selected.
 ↪columns)))
```

```
After performing One-Hot_encoding, the selected dataset has 1000 rows and 31
columns
```

Step 3: Selecting the Training Set for 8 (A) and Test Set (A-C)

[609]:
```
# Splitting the data into training set (A) and Test Set (all)

#setting a random state to ensure reproducibility of resuts and
#comparability of the models
```

```
#the dataset utilised for building a predictive model in 8(a) would be␣
 ↪df_train_a
#The predictive models will be tested on the Test Set
#Attributing 500 instances (50% of the selected dataframe)
#to the test set that would solely utilised to test our predictive models.
df_train_a, df_test_all = train_test_split(selected, test_size = 0.5,␣
 ↪random_state = 25)

print(' Training size 8 (A): {0} instaces\n Test size 8 (A-C): {1} instances'.
 ↪format(len(df_train_a),
                                                                  ␣
 ↪len(df_test_all)))
```

```
Training size 8 (A): 500 instaces
Test size 8 (A-C): 500 instances
```

Step 4: Selecting the Test Set Independent and Dependent Variables for 8 (A-C)

[610]:
```
print('The test dataset utilised to test all predictive models has {0} rows and␣
 ↪{1} columns'.format(len(df_test_all), len(df_test_all.columns)))
#31 columns - due to OHE

#setting the IV and DV (price_dollars) for test data
X_test = df_test_all.drop(columns = 'price_dollars', axis = 1).values
y_test = df_test_all.price_dollars.values

#saving the test set aside and saving it to CSV
df_test_all.to_csv('df_test_all.csv')
```

```
The test dataset utilised to test all predictive models has 500 rows and 31
columns
```

4.6 Question 8 (A): 500 Original Data Training Instances

[611]:
```
print('After Performing the One-Hot-Encoding, The Training Dataset (A) has {0}␣
 ↪rows and {1} columns'.format(len(df_train_a),
                                                    len(df_train_a.
 ↪columns)))
#31 columns - due to OHE
```

```
After Performing the One-Hot-Encoding, The Training Dataset (A) has 500 rows and
31 columns
```

4.6.1 Model 1 (A): Ridge Regression

4.6.2 Using Ridge Regression

In this case Ridge Regression was used to eliminate the overfitting problem, in which a complex model may fit the training data too closely and perform poorly on the data. Thus, using Ridge Regression will help us prevent overfitting by introducing a regularization penalty that discourages our model from being too complex.

Another reason we have chosen to use ridge regression is because we have a small sample size, in which the estimates of the regression coefficients may be noisy and unreliable. Ridge regression can help us stabilize the estimates by shrinking the coefficients towards zero and reducing the variability of the model.

Using Ridge Regression will allow us to retrieve better predicitive performance compared to using standard linear regression, especially due to the issues mentioned above.

[612]:
```
#Splitting the training data into the IV and DV (price_dollars)
X_train_a = df_train_a.drop(columns = 'price_dollars', axis = 1).values
y_train_a = df_train_a.price_dollars.values
```

[613]:
```
#Normalising IV and DV features to increase the Ridge Regression Accuracy
scaler = StandardScaler()

#Normalising IV and DV features in the Test Set to increase Model 1(Ridge␣
 ↪Regression)
#and Model 3(Neural Network Regression) accuracy

#renaming the variables so that they don't get scaled again upon several code␣
 ↪executions
X_train_norm_a = scaler.fit_transform(X_train_a)
X_test_norm = scaler.transform(X_test)

y_train_norm_a = scaler.fit_transform(y_train_a.reshape((-1,1)))
y_test_norm = scaler.transform(y_test.reshape((-1,1)))
```

[614]:
```
#Creating a Ridge Regression Object and Fitting the Training Data
#setting ridge parameter as alpha=1.0 to avoid shrinking coefficients
#and making the model less sensitive to the input data
#also to mitigate for potential overfitting or underfitting of the data
Ridge_model_a = Ridge(alpha = 1.0).fit(X_train_norm_a, y_train_norm_a)
#Demonstrating the coefficients of the Ridge Regression Model

pd.DataFrame(Ridge_model_a.coef_.flatten(),
             df_train_a.drop('price_dollars', axis = 1).columns,
             columns = ['Coefficient']).sort_values(by = 'Coefficient',␣
 ↪ascending = False).round(4)
```

[614]:

	Coefficient
accommodates	0.3015
availability_30	0.2671
neighbourhood_group_Centrum-Oost	0.1916
neighbourhood_group_Centrum-West	0.1787
beds	0.1740
room_type_Entire home/apt	0.1226
neighbourhood_group_Zuid	0.0790
neighbourhood_group_De Pijp - Rivierenbuurt	0.0658
review_scores_rating	0.0543
neighbourhood_group_Oud-Oost	0.0376
room_type_Hotel room	0.0249
neighbourhood_group_De Baarsjes - Oud-West	-0.0054
neighbourhood_group_De Aker - Nieuw Sloten	-0.0198
neighbourhood_group_Westerpark	-0.0356
neighbourhood_group_Geuzenveld - Slotermeer	-0.0386
neighbourhood_group_Oostelijk Havengebied - Ind…	-0.0390
neighbourhood_group_Bijlmer-Oost	-0.0441
room_type_Shared room	-0.0534
neighbourhood_group_Watergraafsmeer	-0.0553
neighbourhood_group_Buitenveldert - Zuidas	-0.0561
neighbourhood_group_Noord-Oost	-0.0669
neighbourhood_group_Bijlmer-Centrum	-0.0713
neighbourhood_group_Slotervaart	-0.0721
neighbourhood_group_Oud-Noord	-0.0788
neighbourhood_group_Bos en Lommer	-0.0822
neighbourhood_group_IJburg - Zeeburgereiland	-0.0856
neighbourhood_group_Osdorp	-0.0904
neighbourhood_group_Noord-West	-0.1070
room_type_Private room	-0.1207
neighbourhood_group_Gaasperdam - Driemond	-0.1239

[618]:
```
# Predicting y values on the training set utilising the trained model
y_pred_train_a = Ridge_model_a.predict(X_train_norm_a)

#Model Accuracy training set
# Evaluating the Model Performance on the training data using the R-squared
#(coefficient of determination) on the training set

R2_Ridge_train_a = Ridge_model_a.score(X_train_norm_a, y_train_norm_a)
#OR: #R2_train_a = r2_score(y_train_norm_a, y_pred_train_a)

# Printing the R2 value for the training set
print(f'The R-squared of the Ridge Regression(A) on training set:␣
 ↪{R2_Ridge_train_a * 100:.2f}%')
```

```
#Evaluating the Model Performance on the training data using the Mean Squared␣
 ↪Error
MSE_Ridge_train_a = mean_squared_error(y_train_norm_a, y_pred_train_a)
# Printing the MSE value on the training set
print(f'The MSE of the Ridge Regression(A) on training set: {MSE_Ridge_train_a:.
 ↪2f}')

# Computing the MAE on the test data
MAE_Ridge_train_a = mean_absolute_error(y_train_norm_a, y_pred_train_a)
print(f'The MAE of the Ridge Regression(A) on the test data is:␣
 ↪{MAE_Ridge_train_a:.2f}')
```

```
The R-squared of the Ridge Regression(A) on training set: 44.81%
The MSE of the Ridge Regression(A) on training set: 0.55
The MAE of the Ridge Regression(A) on the test data is: 0.52
```

[620]:
```
#Testing the Ridge Regression Model on the test set

# Calculating the predicted values
y_pred_test_a = Ridge_model_a.predict(X_test_norm)

# Evaluating the model on the testing data

#Computing the R2  and MSE on the test set using these predicted values

# Calculating the R2 on the test set
R2_Ridge_test_a = Ridge_model_a.score(X_test_norm, y_test_norm)
print(f'The R-squared of the Ridge Regression(A) on the test set:␣
 ↪{R2_Ridge_test_a * 100:.2f}%')

#Calculating the MSE on the test set
MSE_Ridge_test_a = mean_squared_error(y_test_norm, y_pred_test_a)
print(f'The MSE of the Ridge Regression(A)  on the test set: {MSE_Ridge_test_a:.
 ↪2f}')

# Calculating the MAE on the test data
MAE_Ridge_test_a = mean_absolute_error(y_test_norm, y_pred_test_a)
print(f'The MAE of the Ridge Regression(A) on the test data is:␣
 ↪{MAE_Ridge_test_a:.2f}')
```

```
The R-squared of the Ridge Regression(A) on the test set: 40.02%
The MSE of the Ridge Regression(A)  on the test set: 1.11
The MAE of the Ridge Regression(A) on the test data is: 0.61
```

[621]:
```
# Creating a scatterplot of actualVS predicted values
plt.scatter(y_test_norm, y_pred_test_a, color = "#6456DF")
```

```
# Adding the plot title and axis labels
plt.title('Actual vs Predicted Values')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')

# Adding a diagonal line for reference
plt.plot([y_test_norm.min(), y_test_norm.max()], [y_test_norm.min(),␣
 ↪y_test_norm.max()], 'k--', c = 'pink')

# Showing the plot
plt.show();
```

```
/tmp/ipykernel_134/3209032344.py:10: UserWarning: color is redundantly defined
by the 'color' keyword argument and the fmt string "k--" (-> color='k'). The
keyword argument will take precedence.
  plt.plot([y_test_norm.min(), y_test_norm.max()], [y_test_norm.min(),
y_test_norm.max()], 'k--', c = 'pink')
```

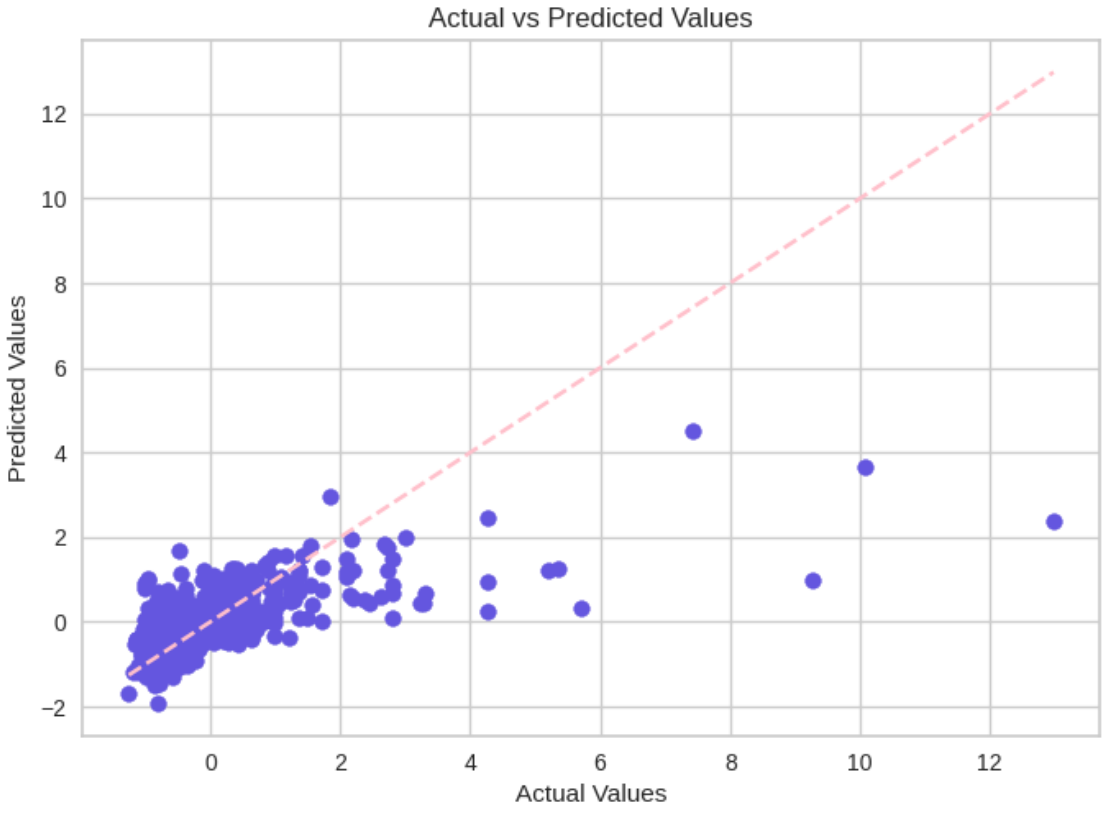

Figure 10: Acatterplot of Actual VS Predicted Values

The values follow a diagonal clustered pattern, however there are dispersed outliers. These outliers may be affecting the regression coefficients.

[622]:
```
# plotting the Residual Plot
#to evaluate teh differences between actual and predicted regression values

visualizer = ResidualsPlot(Ridge_model_a)
visualizer.fit(X_train_norm_a, y_train_norm_a)
#Evaluating model on validation data
visualizer.score(X_test_norm, y_test_norm)
#Poofing/drawing data
graph2 = visualizer.poof();
plt.show();
```

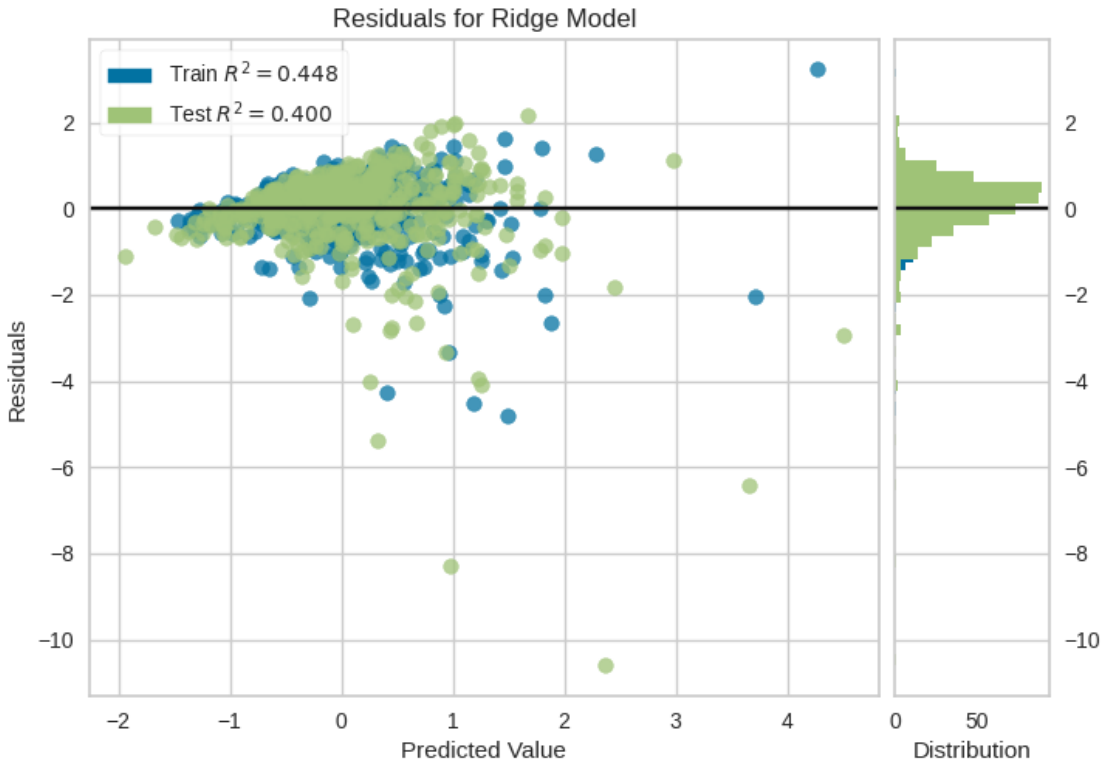

Figure 11: Residual Plot

Training and test data's residuals generally follow a similar pattern, mostly, coinciding. However, the blue points that are farther from from zero indicate underfitting on the training data. The model explained 40% variance in our DV (test set) which is similar to the training set(R2~44%). Therefore, the model fitted the test data fairly well.

4.6.3 Model 2 (A): Decision Tree Regression with Tuned HyperParameters

Not standardising the variables for the DT regressor for this model, the variables for the DT regressior are not standardized because decision trees are not sensitive to the scale of the features. Therefore, we don't need to standardize the variables when using a decision tree model.

In a decision tree, the algorithm splits the data based on the values of the features, not their absolute magnitudes, so the scale of the features doesn't affect the tree's structure or the splitting criteria. This is one of the advantages of decision trees, as it can handle features with different scales without any additional preprocessing.

GridSearchCV

Before creating the DT Regressor Model, the Grid Search was performed for HyperParameter tuning to find the best HyperParameters.

During the Grid Search, the pruning process in confucted with the HyperParameter values specified in the param_grid dictionnary for 3 hyparparameters ('max_depth', 'min_samples_leaf', and 'min_samples_split')/

It should be noted that few parameters were specified in the parameter grid to prevent the risk of overfitting on the training data, and these parameters were adjusted to the ones providing the best performance on the test data

[623]:
```
#finding the best hyperparameters for the DT

# Defining the parameter grid for the DT regressor
param_grid = {
    'max_depth': [3, 4, 5],
    'min_samples_leaf': [1, 2, 3],
    'min_samples_split': [2, 3, 4, 5]
}

# Creating a DT regressor object
dt = DecisionTreeRegressor()

# Creating a GridSearchCV object with 5-fold cross-validation
grid_search = GridSearchCV(dt, param_grid, cv = 5)

# Fitting the GridSearchCV object to the training data
grid_search.fit(X_train_a, y_train_a)

# Printing the best hyperparameters and corresponding R2 score obtained
#by the GridSearchCV algorithm during the cross-validation process on the␣
 ↪training data

#The R2 score printed with the grid_search.best_score_ statement is the best R2␣
 ↪score
#obtained by the GridSearchCV algorithm during the
#cross-validation process, which searches over the hyperparameter space␣
 ↪specified in param_grid
print('The Best Hyperparameters are: ', grid_search.best_params_)
print(' The Best R2 score on training data obtained by the GridSearchCV␣
 ↪algorithm during the cross-validation process is: ',
      grid_search.best_score_)
```

[623]:
```
GridSearchCV(cv=5, estimator=DecisionTreeRegressor(),
             param_grid={'max_depth': [3, 4, 5], 'min_samples_leaf': [1, 2, 3],
                         'min_samples_split': [2, 3, 4, 5]})
```

```
The Best Hyperparameters are:  {'max_depth': 4, 'min_samples_leaf': 2,
'min_samples_split': 2}
 The Best R2 score on training data obtained by the GridSearchCV algorithm
during the cross-validation process is:  0.1774766593756907
```

[624]:
```
# Creating a new DT regressor object with the best hyperparameters computed␣
 ↪above
best_dt_a = DecisionTreeRegressor(max_depth = grid_search.
 ↪best_params_['max_depth'],
                                  min_samples_leaf = grid_search.
 ↪best_params_['min_samples_leaf'],
                                  min_samples_split = grid_search.
 ↪best_params_['min_samples_split'])

# Fitting the DT regressor on the training data
best_dt_a.fit(X_train_a, y_train_a)
```

[624]:
```
DecisionTreeRegressor(max_depth=4, min_samples_leaf=2)
```

[625]:
```
#Demonstrating the DT Feature Importances
pd.DataFrame(best_dt_a.feature_importances_,
             df_train_a.drop('price_dollars', axis = 1).columns,
             columns=['Importances']).sort_values(by = 'Importances',
             ascending = False).round(4)
```

[625]:

	Importances
accommodates	0.4675
availability_30	0.2200
neighbourhood_group_De Pijp - Rivierenbuurt	0.1046
neighbourhood_group_Zuid	0.0815
neighbourhood_group_Centrum-Oost	0.0589
review_scores_rating	0.0260
room_type_Private room	0.0246
neighbourhood_group_Centrum-West	0.0169
room_type_Entire home/apt	0.0000
room_type_Hotel room	0.0000
neighbourhood_group_Westerpark	0.0000
neighbourhood_group_Oostelijk Havengebied - Ind…	0.0000
neighbourhood_group_Watergraafsmeer	0.0000
neighbourhood_group_Slotervaart	0.0000
neighbourhood_group_Oud-Oost	0.0000
neighbourhood_group_Oud-Noord	0.0000
neighbourhood_group_Osdorp	0.0000

```
neighbourhood_group_IJburg - Zeeburgereiland            0.0000
neighbourhood_group_Noord-West                          0.0000
neighbourhood_group_Noord-Oost                          0.0000
neighbourhood_group_Geuzenveld - Slotermeer             0.0000
neighbourhood_group_Gaasperdam - Driemond               0.0000
neighbourhood_group_De Baarsjes - Oud-West              0.0000
neighbourhood_group_De Aker - Nieuw Sloten              0.0000
neighbourhood_group_Buitenveldert - Zuidas              0.0000
neighbourhood_group_Bos en Lommer                       0.0000
neighbourhood_group_Bijlmer-Oost                        0.0000
neighbourhood_group_Bijlmer-Centrum                     0.0000
beds                                                    0.0000
room_type_Shared room                                   0.0000
```

[626]:
```python
# Predicting y values on the training set utilising the trained model
y_pred_train_a = best_dt_a.predict(X_train_a)

# Calculating the R-squared value on the training data
R2_DT_train_a = best_dt_a.score(X_train_a, y_train_a)
print(f'The R2 of the DT(A) on training set: {R2_DT_train_a * 100:.2f}%')

# Calculating the mean squared error on the training data
MSE_DT_train_a = mean_squared_error(y_train_a, y_pred_train_a)
print(f'The MSE of the DT(A) on training set: {MSE_DT_train_a:.2f}')


# Computing the Mean Absolute Error (MAE) on the training data
MAE_DT_train_a = mean_absolute_error(y_train_a, y_pred_train_a)
print(f'The MAE on of the DT(A) the training data is: {MAE_DT_train_a:.2f}')
```

```
The R2 of the DT(A) on training set: 45.50%
The MSE of the DT(A) on training set: 10332.63
The MAE on of the DT(A) the training data is: 72.94
```

[627]:
```python
# Evaluating the performance of the best_dt_a model on the test data
DT_predictions_test_a = best_dt_a.predict(X_test)

#R2 on the Test Set
R2_DT_test_a = best_dt_a.score(X_test, y_test)
print(f'The R2 of the DT(A) on the test set: {R2_DT_test_a * 100:.2f}%')

#MSE on the test set
MSE_DT_test_a = mean_squared_error(y_test, DT_predictions_test_a)
print(f'the MSE of the DT(A) on test set: {MSE_DT_test_a:.2f}')

# Computing the MAE on the test data
MAE_DT_test_a = mean_absolute_error(y_test, DT_predictions_test_a)
```

```
print(f'The MAE of the DT(A) on the test data is: {MAE_DT_test_a:.2f}')
```

```
The R2 of the DT(A) on the test set: 25.35%
the MSE of the DT(A) on test set: 26097.84
The MAE of the DT(A) on the test data is: 88.93
```

Visualising

[628]:
```
# Plotting the best_dt_a
plt.figure(figsize = (38,20))
plot_tree(best_dt_a, filled = True, feature_names = df_train_a.columns[:-1],␣
 ↪fontsize = 10)
plt.show();
```

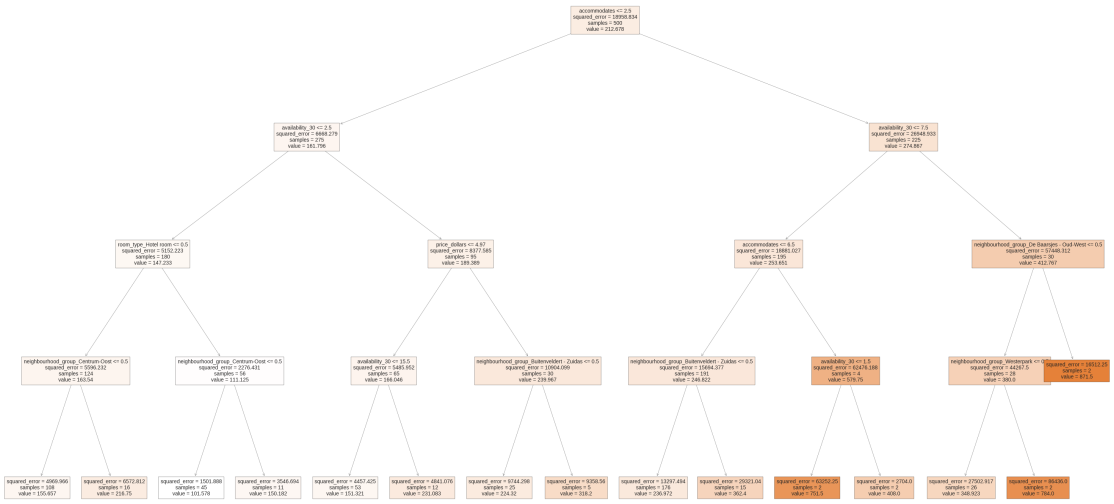

Figure 12: DT Feature Importances

The decision tree shows the root feature as accommodates spiltting off into the most important feature, availability_30 <= 2.5 and availability <= 7.5. The model then continues to split to create subsets of the data that are more homogeneous in terms of the target variable and make more accurate predictions.

[629]:
```
#Creating a scatterplot of actual VS predicted values (test set)
plt.scatter(y_test, DT_predictions_test_a, color = "#6456DF");
#plt.plot([3, 1000], [3, 1000], c='#f0439e', lw=2)
# Adding a diagonal line for reference
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', c =␣
 ↪'pink')
plt.xlabel('Actual (test set)');
plt.ylabel('Predictions');
```

```
plt.title("Scatterplot of actual vs predicted values")
plt.show();
```

```
/tmp/ipykernel_134/1451239848.py:5: UserWarning: color is redundantly defined by
the 'color' keyword argument and the fmt string "k--" (-> color='k'). The
keyword argument will take precedence.
  plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', c
= 'pink')
```

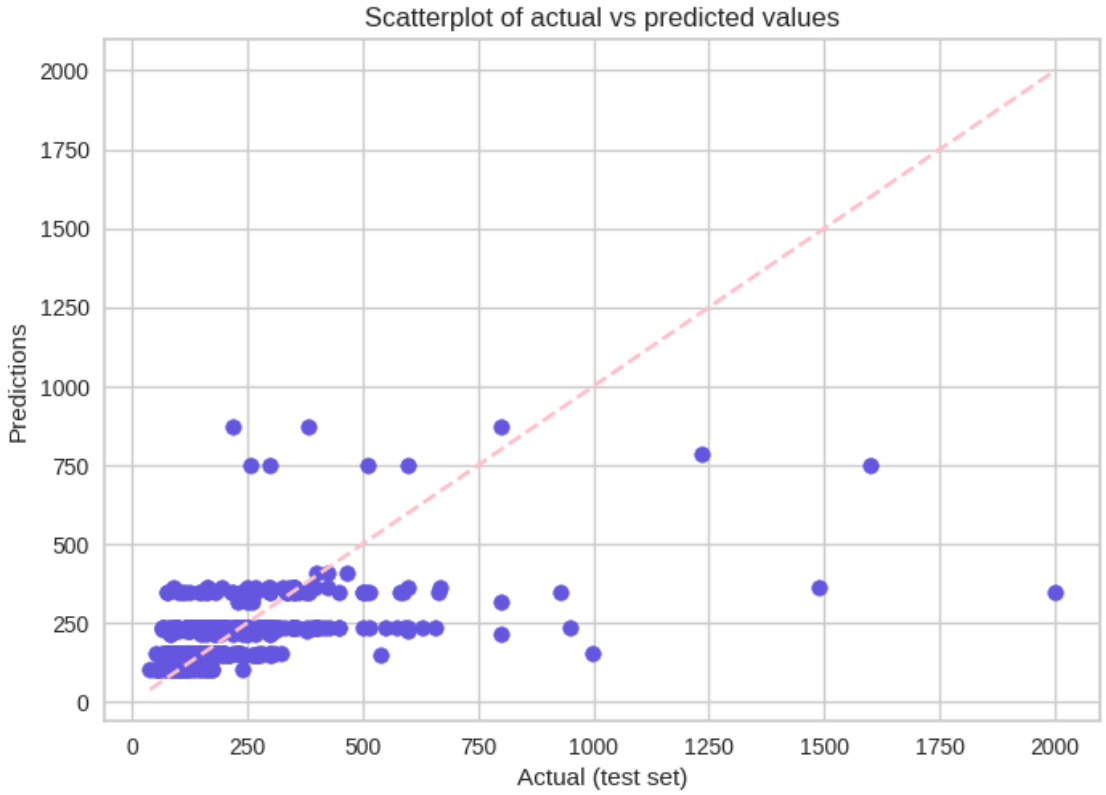

Figure 13: Scatterplot of Actual VS Predicted Values

The values generally follow a dispersed pattern, with groups of clusters. The points do not follow the diagonal line. This suggests that there is some structure in the data, but it is not a simple linear relationship. The presence of clusters indicates that there may be subgroups or patterns in the data that are driving the relationship between the variables. Thus, the model may need to use more complex decision rules to capture the relationship between the variables.

[630]:
```
#plotting the Prediction Error
# Instantiating the visualizer with the fitted DT model
#Alpha parameter - controling for scatterplot points' opacity
visualizer = PredictionError(best_dt_a, alpha = 0.5)
#The training data is fitted into the visualiser
```

```
visualizer.fit(X_train_a, y_train_a)
#Evaluating the model on the Test Data
visualizer.score(X_test, y_test)

# Displaying the visualiser
visualizer.poof();
```

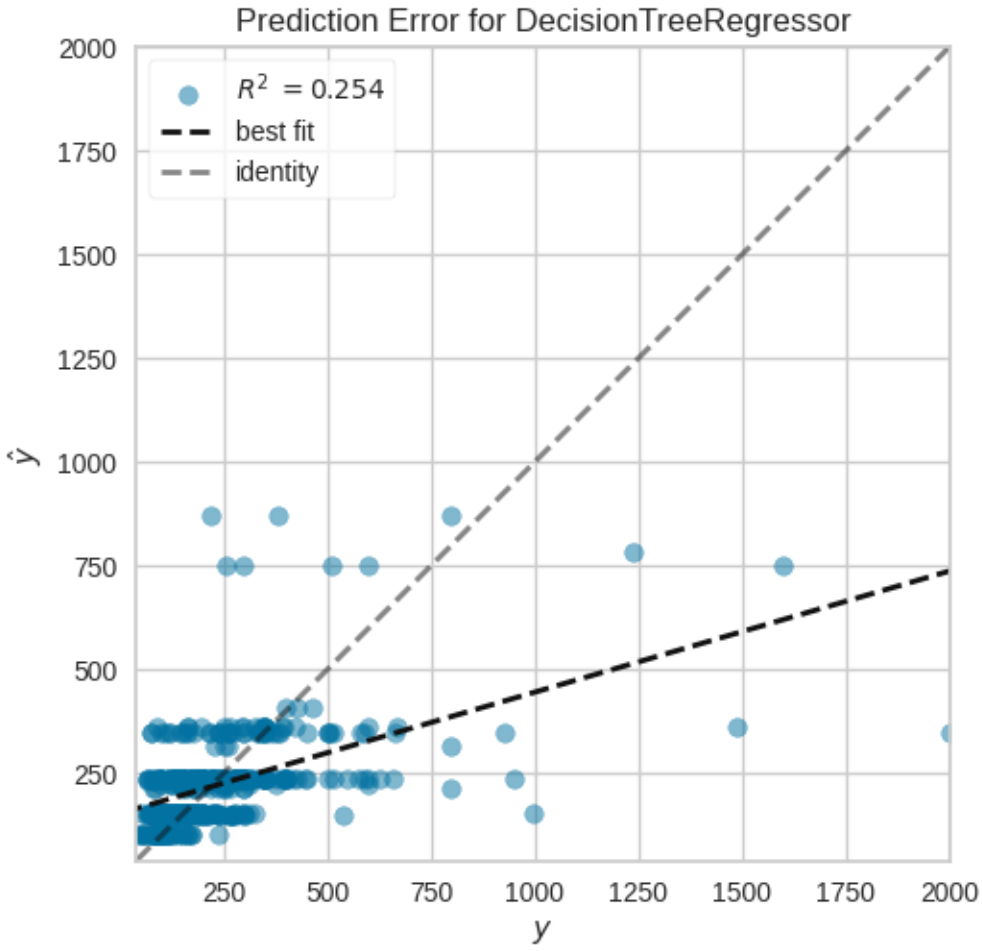

Figure 14: Prediction Error Plot

The identity line indicate a perfect model where predictions match actual values. The gradient of the line of best fit for this model is lower, thus indicating that our model predicts a lower than actual price. The degree of deviation from the identity line is quite high, suggesting that the accuracy of the model is quite low.

[201]:
```
#Plotting the Residuals plot
# Instantiating the visualizer with the fitted decision tree model
visualizer = ResidualsPlot(best_dt_a)
```

```
#Fitting training data into the visualiser
visualizer.fit(X_train_a, y_train_a)

# Evaluating model on validation data
visualizer.score(X_test, y_test)

# Poofing the data/ Displaying the plot
visualizer.poof();
```

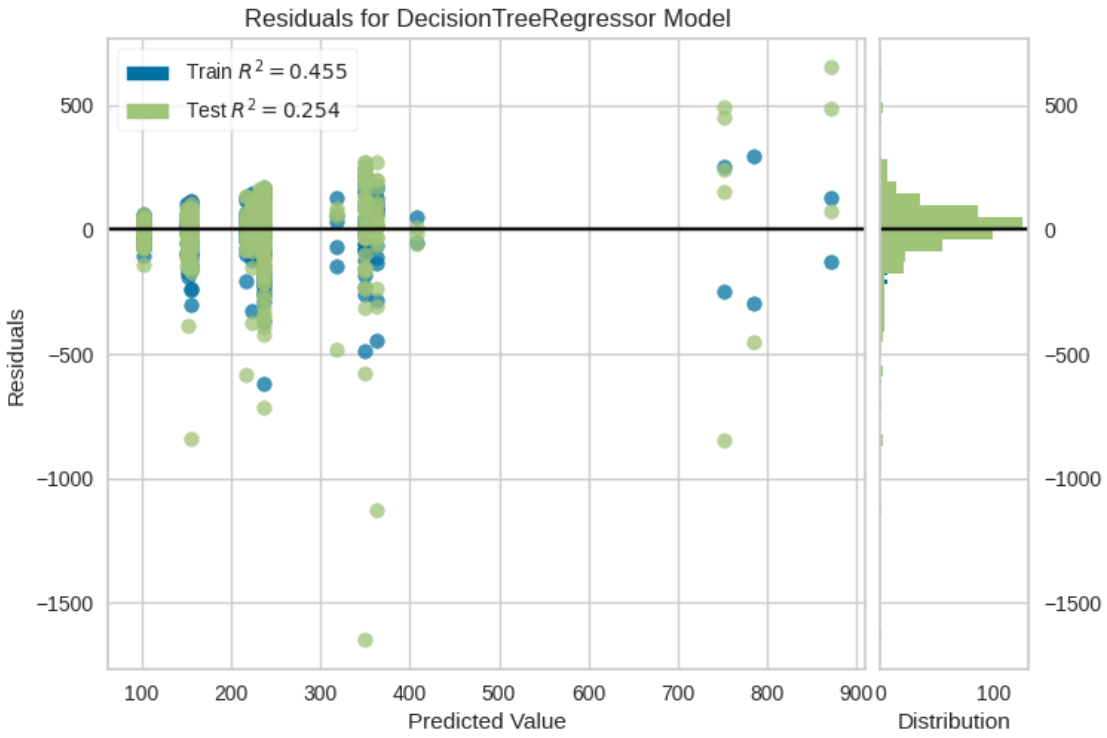

Figure 15:

The residuals plot above depicts clusters of points surrounding zero. This suggests that the model is not accurately capturing the relationships between the features and target variable. The clusters of points may indicate that the model is making consistent errors in predicting certain ranges of the target variable. The separate clusters may suggest that there are underlying patterns or relationships in the data that the model is not able to capture. This could be due to a number of factors such as non-linear relationships between features and the target variable, or missing features.

4.6.4 Model III (A): Neural Network Regression with tuned HuperParameters for the Original Data

Utilising the data normalised earlier to ensure faster model convergence and overcome the vanishing gradient issue

First, the function to calculate the R2 (coef of determination) for the neural network was defined. It will be utilised to evaluate the model's performance and compare it to the other 2 models (Model 1 and 2).

This will be the custom metric utilised upn compiling the model.

[631]:
```
#Defining the function to calculate the R2 score (coefficient of determination)␣
 ↪
#between the true and predicted values

def coeff_determination(y_true, y_pred):
    SS_res = K.sum(K.square(y_true-y_pred))
    SS_tot = K.sum(K.square(y_true - K.mean(y_true)))
    return (1 - SS_res/(SS_tot + K.epsilon()))
```

Early Stopping - regularisation technique

The Early Stopping regularisation technique was utilised to train the NN model. This would mitigate the model's overfitting risk, enhancing the model's performance(Vokhmina et al., 2015).

Adding Early stopping would ensure that the training process is stoppen upon decreasing performance on the validation set. For our code, the valisation loss would be monitored during the training prcess, making the function stop upon no improvement during 10 epoch and saving the model of highest performance in order to form predictions.

Additionally, early stopping aids to decrease computational power and time via stoppng the model at the point where performance would be unlikely to enhance.

Note that *several Regularisation techniques were utilised for the Neural Network model, which ensured the best performance given the data (Vokhmina et al., 2015)*

[632]:
```
# setting the random seed to ensure reproducibility of results
# setting the random seed for Numpy
np.random.seed(25)
# setting the random seed for Tensorflow
tf.random.set_seed(25)

# Defining the early stopping criteria

#monitor='val_loss' - monitoring the loss of the validation set during
#the training process.

#patience=10 - making the training stop upon no improvement in the
#validation set. throughout 10 epochs.

#verbose=1 - progress messages to be printed during training

#mode='min' - making the callack search for the minimum value of the val_loss

#restore_best_weights=True - saving the model of highest performance
```

```
#in order to form predictions.
early_stop = EarlyStopping(monitor='val_loss', patience=10,
                           verbose=1, mode='min', restore_best_weights=True)
```

Model Architecture

Utilising Keras's Deap Learning Framework (by utilising keras.Sequential())to Generate a sequential Neural Network Model model.

After OHE, our data consists of 31 features (i.e., 30IVs and 1 DV) input_shape=(30,) - shape of the input data to the layer, due to having 30 Independent Variables after performing OHE.

keras.layers.Dense(units=1) - the output neuron should represent the target variable the model attempts to predict based on the inputted features (i.e., 'price_dollars' in our case). Thus, we are specifying that the output layer should consist of 1 neuron and teh default, linear activation function.

Layers and Neurons in Each Layer:

Given the structured nature of Airbnb listings' data, solely dense layers were included instead of recurrent or convolutional layers. This would allow our model to grasp more intricate relationships between our independent variables and target variable.

The model's complexity was increased by adding multiple dense layers to enhance the model's learning of complex data representations. This is because each layer has a potential to capture various data features.

Additionally, since a lower number of hidden units/neurons decreases the model's capacity to prefectly fit the train set data, the number of hidden units was decreased in each subsequent layer. This reduced the ridk of overfitting by forcin the NN model to learn less intricate data representations.

However, given the trade-off of enhancing #layers and #neurons and considering the model's complexity and small dataset size (overgitting), early stopping (code cell above) and dropout layers were added after each layer to mitigate the overfitting risk.

Dropout - regularisation technique

Dropout layers with decreading dropout rates in each layer were added after each fully connected-layer. Similarly, decreasing the dropout rate can also reduce overfitting (Sung, 1998). Dropout works by randomly dropping out (setting to zero) some of the outputs of a layer during training. This can be seen as a form of regularization because it forces the model to learn more robust representations of the data. However, if the dropout rate is too high, the model may not learn enough from the data, which can result in underfitting.

Dropout sets certain outputs of the layer to 0 (randomly drops-out) during the training prpocess. Thus, similarly to #layers, the dropout rate was decreased throughout the NN to mitigate the overfitting risk of the NN and increasing its generalisation. Dropping out certain units of the train data, would allow for the model's learning of more robust data representation.

Additionally, dropout deareases #iterations required for the model convergence. Hence, utilising the dropout would increase the NN's traing's speed.

Activation Function

Different activation functions and their combinations were attempted for different layers, including tanh and sigmoid. However, adding solely the 'relu' activation for each layer resulted in a superior model performance.

Since our dataset has no negative values, standard *ReLU* activation function was utilised, and not the keras.layers.LeakyReLU() activation function (which can aid in proving model performance, allowing for a small non-zero gradient upon negative input)(Sung,1998).

[633]:
```
# defining the model architecture
NN_model_a = keras.Sequential([
    #1st fully-connected layer of the NN with 128 neurons,
    #input shape=30 -> features (30 IVs), and ReLU activation function.
    keras.layers.Dense(units=128, input_shape=(30,), activation='relu'),
    #adding the 1st dropout layer with a 0.5 dropout rate
    #after the 1st fully connected-layer
    keras.layers.Dropout(0.5),
    #2nd fully-connected layer with 64 hidden units/neurons
    #and the same activation function
    keras.layers.Dense(units=64, activation='relu'),
    #2nd dropout layer with the same dropout rate
    keras.layers.Dropout(0.5),
    #3rd fully-connected layer with 32 hidden units and the same activation␣
 ↪function
    keras.layers.Dense(units=32, activation='relu'),
    #3rd dropout layer with a lower dropout rate of 0.3
    keras.layers.Dropout(0.3),
    # 4th fully-connected layer with 16 hidden units and the same activation␣
 ↪function
    keras.layers.Dense(units=16, activation='relu'),
    #4th dropout layer with a lower dropout rate of 0.2
    keras.layers.Dropout(0.2),
    # 4th fully-connected layer with 8 hidden units and the same activation␣
 ↪function
    keras.layers.Dense(units=8, activation='relu'),
    #5th dropout layer with the same dropout rate of 0.2
    keras.layers.Dropout(0.2),
    # the NN's output layer with 1 unit (regression task)
    #with a linear activation function (default) in order to predict out␣
 ↪continuous
    #target variable ('price_dollars')
    keras.layers.Dense(units=1)
])
```

Compiling the model

1) Small datasets generally tend to be more prone to the overfitting risk, which is exacerbated with model complexity and high learning rate. Thus, before compiling the model, the learning rate was specified.

A higher learning rate would lead to higher convergence speed of the of the optimization proces, however, might result in the minimum loss function being overshooted (MSE in our case), leading to resulults' instability and inaccuracy.The small dataset size of 500 instances increased the ensitivity of the optimisation process to learning rate changes. Thus, after experimenting with various learning rate values the learning rate was decreased from the default 0.001 to 0.0001 to prevent the overfitting risk. Thus, decresing the learning rate resulted in minimum variation between the training and test values.

It should be noted that: While decreasing the learning rate even further could, potentially, result in superior performance, due to decreasing the learning rate requiring more epochs for the model training and smaller batch sizes, however this requires substantial computating power and could not be computed with the resources available. Thus, upon the trade-offs between the model performance and training time and power, the learning rate of 0.0001 was providing the best performance given the computational capabililities.

2) Similarly, since various optimisation algorithms result in different convergence speed and final outputs, various optimisers were experimented with, including SGD (Stochastic Gradient Descent), RMSprop and Adamax (Adaptive Moment Estimation), with the latter resulting in the poorest performance on the test set. However, the 'Adam' optimiser performed the best for the given data. Additionally, this optimisation algorithm possesses high computational efficiency, has easy implementation and is less sensitive to hyperparameter choice, making it one of the most commonly-used optimisers for various tasks.

3) Different Loss Functions were attempted, including MAE(mean absolute error) and MSLE(mean squared logarithmic error), which could potentiallhy result in the model better capturing outliers. However, the 'MSE'(Mean Squared Error) loss, one of the most frequent functions utilised in regression tasks, gave the best performance on the test data. Thus, we instruct the Optimiser to mnimise the MSE upon the training.

4) Finally, the metrics of evaluation were specified to be the same in the models 1 and 2. These evaluation metrics would be utilised in order to track the NN's performance

[634]:
```
# Compiling the model.
#decreasing the learning rate to 0.0001 (automatic = 0.001)
#to prevent overfitting
opt = keras.optimizers.Adam(learning_rate=0.0001)
NN_model_a.compile(optimizer=opt, loss='MSE',
                   metrics=['mse', 'mae', coeff_determination])
```

Model Training

1) Inputting the normalised traing data - the independent and target variables. They would be utilised to train the NN.

EPOCHS AND BATCH SIZE

the initial parameters for epochs and batch size were 64 epochs and 100 batch size. However, upon decreasing the learning rate the recommended parameters were applied. It is recommended to adjust these 2 parameters proportionally (ideally, by the factor of 10 for epochs)(Vokhmina et al., 2015). However, this would result in 640 epochs and batch size of 1 sample, requiring substantial computational power. Thus, 128 epochs and batch size of 5 were selected and provided teh optimal

performance.

2) epochs= 128 - similarly to Question 6, we are specifying the number of times the model should run through the entire dataset (128 times or 2 in the power of 7). The number of epochs were increased

A different number of epochs was attempted. However, increasing the number of epochs beyond the selected resulted in model overfitting on the training data (i.e., very good performance on training data and poorer performance on testing data), and 64 epochs provided the best performance on the test set.

3) Batch size=5 - specifying 5 samples to be propagated through the network at one particular time.

4) Validation split=0.2 - Instead of creating separate datasets for the validation set, Keras possesses an in-built parameter validation_split, which allows to split the trainings data into the training and validation sets directly. This saves time of manual computation while ensuring consistency of the split upon multiple executions. Additionally, for the purpose of the assessment, solely the test set is monitored. Thus, the validation split function is utilised for a rapid validation check.

Thus, we utilise this function to split the training set into 2 parts - training(80%) and validation(20%) sets. While this was unnecessary for the previous 2 models, Neural Networks possess an intricate parameter set, impacting the model performance. Thus, adding a validation split to the Neural Network by mitigating the overfitting risk. The Validation Set was utilised so as to control for the NN's performance evaluation during the training process and adjust the NN' models architecture and HuperParameters accordingly.

5) callbacks - specifying the early stopping as a callback. This would monitor the validation loss and terminate the training process upon no improvement in the model during the 10 epochs, as specified above.

[640]:
```
#timing the traninng process
#start time
start_time_NN_a =  time.time()

# Training the model with early stopping and epochs and batch size
#adjusted to the learning rate
#splitting the training set into 2 parts:
#training(80%) and validation(20%) of the training dataset
history = NN_model_a.fit(X_train_norm_a, y_train_norm_a,
                         epochs=128, batch_size=5,
                         validation_split=0.2, callbacks=[early_stop])

#end time
end_time_NN_a = time.time()
#calculating the difference
elapsed_time_NN_a = end_time_NN_a - start_time_NN_a

print(f"Elapsed time: {elapsed_time_NN_a:.2f} seconds")
```

```
Epoch 1/128
80/80 [==============================] - 0s 2ms/step - loss: 0.8269 - mse:
0.8269 - mae: 0.6267 - coeff_determination: -0.7795 - val_loss: 0.4430 -
val_mse: 0.4430 - val_mae: 0.5371 - val_coeff_determination: -0.1357
Epoch 2/128
80/80 [==============================] - 0s 2ms/step - loss: 0.7725 - mse:
0.7725 - mae: 0.6109 - coeff_determination: -0.7324 - val_loss: 0.4379 -
val_mse: 0.4379 - val_mae: 0.5331 - val_coeff_determination: -0.1207
Epoch 3/128
80/80 [==============================] - 0s 2ms/step - loss: 0.7683 - mse:
0.7683 - mae: 0.6112 - coeff_determination: -0.7365 - val_loss: 0.4369 -
val_mse: 0.4369 - val_mae: 0.5327 - val_coeff_determination: -0.1164
Epoch 4/128
80/80 [==============================] - 0s 2ms/step - loss: 0.8509 - mse:
0.8509 - mae: 0.6367 - coeff_determination: -0.6008 - val_loss: 0.4339 -
val_mse: 0.4339 - val_mae: 0.5272 - val_coeff_determination: -0.0940
Epoch 5/128
80/80 [==============================] - 0s 2ms/step - loss: 0.8433 - mse:
0.8433 - mae: 0.6324 - coeff_determination: -0.4256 - val_loss: 0.4367 -
val_mse: 0.4367 - val_mae: 0.5285 - val_coeff_determination: -0.0991
Epoch 6/128
80/80 [==============================] - 0s 2ms/step - loss: 0.7992 - mse:
0.7992 - mae: 0.6370 - coeff_determination: -8.3718 - val_loss: 0.4404 -
val_mse: 0.4404 - val_mae: 0.5333 - val_coeff_determination: -0.1163
Epoch 7/128
80/80 [==============================] - 0s 2ms/step - loss: 0.8970 - mse:
0.8970 - mae: 0.6534 - coeff_determination: -0.5494 - val_loss: 0.4403 -
val_mse: 0.4403 - val_mae: 0.5304 - val_coeff_determination: -0.1054
Epoch 8/128
80/80 [==============================] - 0s 2ms/step - loss: 0.8652 - mse:
0.8652 - mae: 0.6302 - coeff_determination: -1.1986 - val_loss: 0.4414 -
val_mse: 0.4414 - val_mae: 0.5297 - val_coeff_determination: -0.1027
Epoch 9/128
80/80 [==============================] - 0s 2ms/step - loss: 0.7096 - mse:
0.7096 - mae: 0.5790 - coeff_determination: -0.3887 - val_loss: 0.4404 -
val_mse: 0.4404 - val_mae: 0.5282 - val_coeff_determination: -0.0937
Epoch 10/128
80/80 [==============================] - 0s 2ms/step - loss: 0.8472 - mse:
0.8472 - mae: 0.6092 - coeff_determination: -0.4964 - val_loss: 0.4376 -
val_mse: 0.4376 - val_mae: 0.5269 - val_coeff_determination: -0.0865
Epoch 11/128
80/80 [==============================] - 0s 2ms/step - loss: 0.8068 - mse:
0.8068 - mae: 0.5921 - coeff_determination: -0.8576 - val_loss: 0.4383 -
val_mse: 0.4383 - val_mae: 0.5270 - val_coeff_determination: -0.0863
Epoch 12/128
80/80 [==============================] - 0s 2ms/step - loss: 0.7319 - mse:
0.7319 - mae: 0.5929 - coeff_determination: -0.3525 - val_loss: 0.4360 -
val_mse: 0.4360 - val_mae: 0.5269 - val_coeff_determination: -0.0853
```

```
Epoch 13/128
80/80 [==============================] - 0s 2ms/step - loss: 0.8439 - mse:
0.8439 - mae: 0.6177 - coeff_determination: -1.0049 - val_loss: 0.4394 -
val_mse: 0.4394 - val_mae: 0.5304 - val_coeff_determination: -0.0975
Epoch 14/128
43/80 [===============>...] - ETA: 0s - loss: 0.6470 - mse: 0.6470 -
mae: 0.6023 - coeff_determination: -0.8395Restoring model weights from the end
of the best epoch: 4.
80/80 [==============================] - 0s 2ms/step - loss: 0.8896 - mse:
0.8896 - mae: 0.6363 - coeff_determination: -0.5984 - val_loss: 0.4398 -
val_mse: 0.4398 - val_mae: 0.5322 - val_coeff_determination: -0.1058
Epoch 00014: early stopping
Elapsed time: 1.81 seconds
```

[641]:
```
# Making predictions on the training set
NN_train_predictions_a = NN_model_a.predict(X_train_norm_a)

# Evaluating the model on the training set
#utilising the evaluate() method is a method, specific to the Keras API
#for Neural Networks
train_loss_a, train_mse_a, train_mae_a, train_r2_a = NN_model_a.
 ↪evaluate(X_train_norm_a,

 ↪y_train_norm_a)

#saving the results to the variables to, consequently, compare the models
R2_NN_train_a = train_r2_a
MSE_NN_train_a = train_mse_a
MAE_NN_train_a = train_mae_a

#Printing the metrics on the training set
print(f'R2 of the NN (A) on the training set: {R2_NN_train_a * 100:.4f}%')
print(f'MSE of the NN (A) on the training set: {MSE_NN_train_a:.4f}')
print(f'MAE of the NN (A) on the training set: {MAE_NN_train_a:.4f}')
```

```
16/16 [==============================] - 0s 1ms/step - loss: 0.6675 - mse:
0.6675 - mae: 0.5824 - coeff_determination: 0.2791
R2 of the NN (A) on the training set: 27.9112%
MSE of the NN (A) on the training set: 0.6675
MAE of the NN (A) on the training set: 0.5824
```

[642]:
```
#Making predictions on the test set
NN_predictions_test_a = NN_model_a.predict(X_test_norm)

# Evaluating the performance of the NN(A) model on the test set
##utilising the evaluate() method is a method, specific to the Keras API
#for Neural Networks
```

```
test_loss_a, test_mse_a, test_mae_a, test_r2_a = NN_model_a.
 ↪evaluate(X_test_norm,
                                                                  ␣
 ↪y_test_norm)

#saving the results to the variables to, consequently, compare the models
R2_NN_test_a = test_r2_a
MSE_NN_test_a = test_mse_a
MAE_NN_test_a = test_mae_a

#Printing the metrics on the test set
print(f'R2 of the NN (A) on the test set:  {R2_NN_test_a * 100:.4f}%')
print(f'MSE of the NN (A) on the test set:   {MSE_NN_test_a:.4f}')
print(f'MAE of the NN (A) on the test set:  {MAE_NN_test_a:.4f}')
```

```
16/16 [==============================] - 0s 1ms/step - loss: 1.3814 - mse:
1.3814 - mae: 0.6878 - coeff_determination: 0.2057
R2 of the NN (A) on the test set:  20.5750%
MSE of the NN (A) on the test set:   1.3814
MAE of the NN (A) on the test set:  0.6878
```

Visualising

[643]:
```
# plotting the scatter plot
#** Note that for the Neural Network Regression, the axis of actual VS␣
 ↪Predicted Values
#are reversed comaring to Model 1 and 2
plt.scatter(NN_predictions_test_a, y_test_norm, color = '#6456DF')
# Adding a diagonal line for reference
plt.plot([y_test_norm.min(), y_test_norm.max()], [y_test_norm.min(),␣
 ↪y_test_norm.max()], 'k--', c = 'pink')
plt.xlabel('Predictions')
plt.ylabel('Actual (test set)')
plt.title('Scatterplot of actual vs predicted values')
plt.show();
```

```
/tmp/ipykernel_134/3681836100.py:6: UserWarning: color is redundantly defined by
the 'color' keyword argument and the fmt string "k--" (-> color='k'). The
keyword argument will take precedence.
  plt.plot([y_test_norm.min(), y_test_norm.max()], [y_test_norm.min(),
y_test_norm.max()], 'k--', c = 'pink')
```

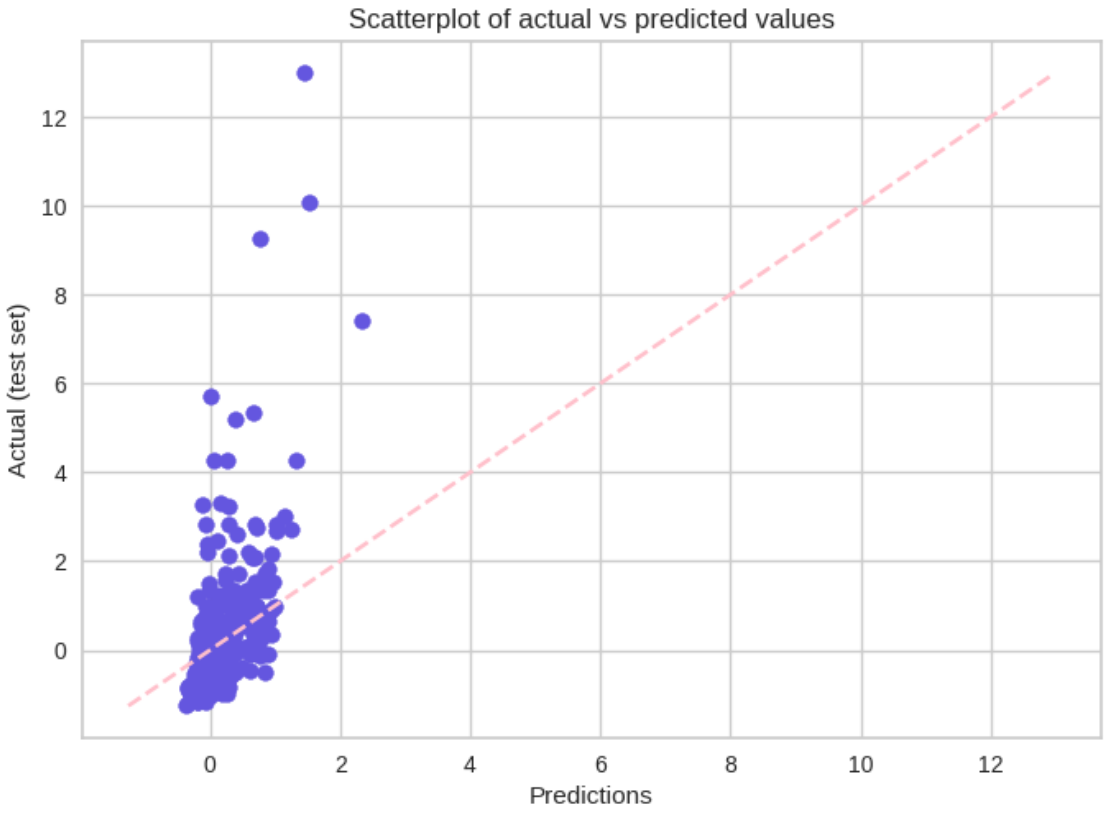

Figure 16: Scatterplotof Actual VS Predicted Values:

The values mostly follow a vertical pattern with many of the points clustered in one region. Outliers can be observed. The clustered pattern in one region indicates that the model is consistently making similar predictions for a range of input values.

4.7 Question 8 (B): 500 Synthetic Training Instances

[644]:
```
#Randomly sampling 500 out of 1,000 Synthetic Data Instances, generated in␣
↪Question 7(A)
#and stored in the synthetic_df_airbnb dataframe
synthetic_train = synthetic_df_airbnb.sample(n=500, random_state=8)
#Checking the fiest 5 rows of the sampled data
synthetic_train.head()
```

[644]:

	availability_30	accommodates	beds	price_dollars	review_scores_rating	\
908	0.0	3.0	10.0	71.070000	4.98	
265	34.0	1.0	2.0	0.880000	4.68	
627	1.0	2.0	2.0	122.379997	4.75	
327	2.0	5.0	2.0	181.309998	4.67	
244	0.0	4.0	2.0	268.720001	4.76	

```
     neighbourhood_group_Bijlmer-Centrum  neighbourhood_group_Bijlmer-Oost  \
908                                    0                                  0
265                                    0                                  0
627                                    0                                  0
327                                    0                                  0
244                                    0                                  0

     neighbourhood_group_Bos en Lommer  \
908                                  0
265                                  0
627                                  0
327                                  0
244                                  0

     neighbourhood_group_Buitenveldert - Zuidas  \
908                                           0
265                                           0
627                                           0
327                                           0
244                                           0

     neighbourhood_group_Centrum-Oost  …  neighbourhood_group_Oud-Noord  \
908                                 0  …                               0
265                                 0  …                               0
627                                 0  …                               1
327                                 0  …                               0
244                                 0  …                               0

     neighbourhood_group_Oud-Oost  neighbourhood_group_Slotervaart  \
908                             0                                0
265                             0                                0
627                             0                                0
327                             0                                0
244                             0                                0

     neighbourhood_group_Watergraafsmeer  neighbourhood_group_Westerpark  \
908                                    0                                0
265                                    0                                0
627                                    0                                0
327                                    0                                0
244                                    0                                0

     neighbourhood_group_Zuid  room_type_Entire home/apt  \
908                         0                          1
265                         0                          0
627                         0                          0
```

```
327                          0                          0
244                          0                          0

     room_type_Hotel room  room_type_Private room  room_type_Shared room
908                     0                      0                      0
265                     0                      0                      1
627                     0                      1                      0
327                     0                      1                      0
244                     0                      1                      0

[5 rows x 31 columns]
```

[645]:
```
print('After Performing One-Hot-Encoding, the Synthetic Training Dataset (B)␣
 ↪has {0} rows and {1} columns'.format(len(synthetic_train),
                                                                    ␣
 ↪len(synthetic_train.columns)))
```

```
After Performing One-Hot-Encoding, the Synthetic Training Dataset (B) has 500
rows and 31 columns
```

4.7.1 Model I (B): Ridge Regression

[646]:
```
#Normalising IV and DV features to increase the Ridge Regression Accuracy

#Splitting the training data into the IV and DV (price_dollars)
X_train_b = synthetic_train.drop(columns = 'price_dollars', axis = 1).values
y_train_b = synthetic_train.price_dollars.values

scaler = StandardScaler()

#renaming the variables so that they don't get scaled again upon several code␣
 ↪executions
X_train_norm_b = scaler.fit_transform(X_train_b)
y_train_norm_b = scaler.fit_transform(y_train_b.reshape((-1,1)))
```

[647]:
```
#Creating a Ridge Regression Object and Fitting the Training Data
Ridge_model_b = Ridge(alpha = 1.0).fit(X_train_norm_b, y_train_norm_b)
#Demonstrating the coefficients of the Ridge Regression Model

pd.DataFrame(Ridge_model_b.coef_.flatten(),
             synthetic_train.drop('price_dollars', axis = 1).columns,
             columns = ['Coefficient']).sort_values(by = 'Coefficient',
                                                    ascending = False).round(4)
```

[647]:
```
                                              Coefficient
room_type_Entire home/apt                          0.1144
neighbourhood_group_De Baarsjes - Oud-West         0.0949
```

```
accommodates                                          0.0882
neighbourhood_group_Centrum-Oost                      0.0595
room_type_Hotel room                                  0.0496
neighbourhood_group_Bos en Lommer                     0.0466
availability_30                                       0.0446
neighbourhood_group_Bijlmer-Oost                      0.0390
neighbourhood_group_Bijlmer-Centrum                   0.0199
neighbourhood_group_Centrum-West                      0.0187
beds                                                  0.0096
neighbourhood_group_Zuid                              0.0077
neighbourhood_group_Oud-Noord                         0.0065
neighbourhood_group_De Pijp - Rivierenbuurt           0.0039
neighbourhood_group_IJburg - Zeeburgereiland         -0.0040
neighbourhood_group_Oud-Oost                         -0.0155
neighbourhood_group_Noord-Oost                       -0.0165
neighbourhood_group_Oostelijk Havengebied - Ind…     -0.0179
neighbourhood_group_Watergraafsmeer                  -0.0183
room_type_Private room                               -0.0267
neighbourhood_group_Gaasperdam - Driemond            -0.0315
neighbourhood_group_Slotervaart                      -0.0353
review_scores_rating                                 -0.0368
neighbourhood_group_Westerpark                       -0.0392
neighbourhood_group_Geuzenveld - Slotermeer          -0.0512
neighbourhood_group_Noord-West                       -0.0515
neighbourhood_group_De Aker - Nieuw Sloten           -0.0621
neighbourhood_group_Osdorp                           -0.0667
neighbourhood_group_Buitenveldert - Zuidas           -0.1013
room_type_Shared room                                -0.2331
```

[650]:
```
# Predicting y values on the training set utilising the trained model
y_pred_train_b = Ridge_model_b.predict(X_train_norm_b)

#Model Accuracy training set
# Evaluating the Model Performance on the training data using the R-squared
#(coefficient of determination) on the training set

R2_Ridge_train_b = Ridge_model_b.score(X_train_norm_b, y_train_norm_b)
#OR:
#R2_train_a = r2_score(y_train_norm_b, y_pred_train_b)

# Printing the R2 value
print(f'R-squared of the Ridge Regression(B) on training set: {R2_Ridge_train_b␣
 ↪* 100:.2f}%')

#Evaluating the Model Performance on the training data using the Mean Squared␣
 ↪Error
MSE_Ridge_train_b = mean_squared_error(y_train_norm_b, y_pred_train_b)
```

```
# Printing the MSE value
print(f'MSE of the Ridge Regression(B) on training set: {MSE_Ridge_train_b:.
 ↪2f}')

# Calculating the MAE on the test data
MAE_Ridge_train_b = mean_absolute_error(y_test_norm, y_pred_test_b)
print(f'The MAE of the Ridge Regression(B) on the training data is:␣
 ↪{MAE_Ridge_train_b:.2f}')
```

```
R-squared of the Ridge Regression(B) on training set: 14.32%
MSE of the Ridge Regression(B) on training set: 0.86
The MAE of the Ridge Regression(B) on the training data is: 0.75
```

[651]:
```
#Testing the Ridge Regression model on the test set

# Calculating the predicted values
y_pred_Ridge_test_b = Ridge_model_b.predict(X_test_norm)

# Evaluating the model on the testing data

#Computing the R2  and MSE on the test set using these predicted values

# Calculating the R2 on the test set
R2_Ridge_test_b = Ridge_model_b.score(X_test_norm, y_test_norm)
print(f'R-squared of the Ridge Regression(B) on the test set: {R2_Ridge_test_b␣
 ↪* 100:.2f}%')

#Calculating the MSE on the test set
MSE_Ridge_test_b = mean_squared_error(y_test_norm, y_pred_Ridge_test_b)
print(f'MSE of the Ridge Regression(B) on the test set: {MSE_Ridge_test_b:.2f}')

# Calculating the MAE on the test data
MAE_Ridge_test_b = mean_absolute_error(y_test_norm, y_pred_Ridge_test_b)
print(f'The MAE of the Ridge Regression(B) on the test data is:
 ↪{MAE_Ridge_test_b:.2f}')
```

```
R-squared of the Ridge Regression(B) on the test set: 11.92%
MSE of the Ridge Regression(B) on the test set: 1.62
The MAE of the Ridge Regression(B) on the test data is:0.75
```

Visualising

[652]:
```
# Creating a scatterplot of actual VS predicted values for the Ridge Regression␣
 ↪(B)
plt.scatter(y_test_norm, y_pred_test_b, color = "#6456DF")

# Adding the plot title and axis labels
plt.title('Actual vs Predicted Values')
```

```
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')

# Adding a diagonal line for reference
plt.plot([y_test_norm.min(), y_test_norm.max()], [y_test_norm.min(),␣
 ↪y_test_norm.max()], 'k--', c = 'pink')

# Showing the plot
plt.show();
```

```
/tmp/ipykernel_134/3109736235.py:10: UserWarning: color is redundantly defined
by the 'color' keyword argument and the fmt string "k--" (-> color='k'). The
keyword argument will take precedence.
  plt.plot([y_test_norm.min(), y_test_norm.max()], [y_test_norm.min(),
y_test_norm.max()], 'k--', c = 'pink')
```

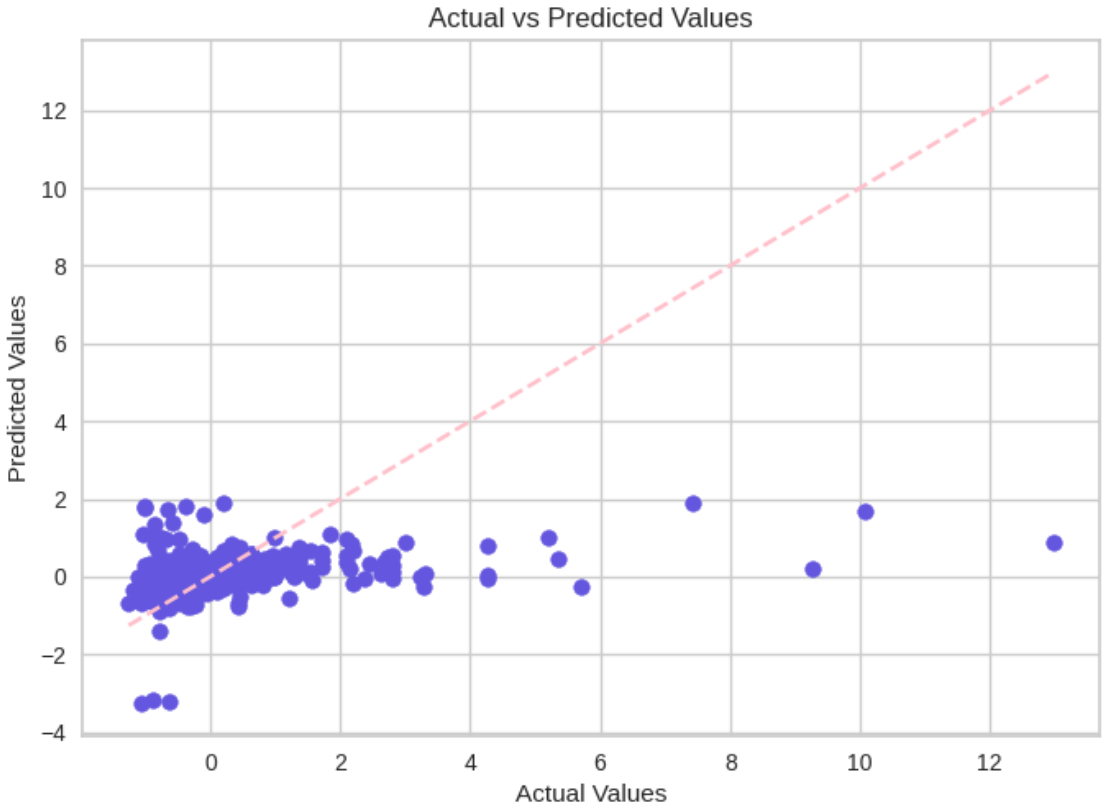

Figure 17: Scatterplot

The values do not follow a diagonal pattern and are generally clustered. Indicating poor performance of the ridge regression model on the data.

[653]:
```
#plotting the residuals
visualizer= ResidualsPlot(Ridge_model_b)
visualizer.fit(X_train_norm_b, y_train_norm_b)
#Evaluating model on validation data
visualizer.score(X_test_norm, y_test_norm)
#Poofing/drawing data
graph2 = visualizer.poof();
plt.show();
```

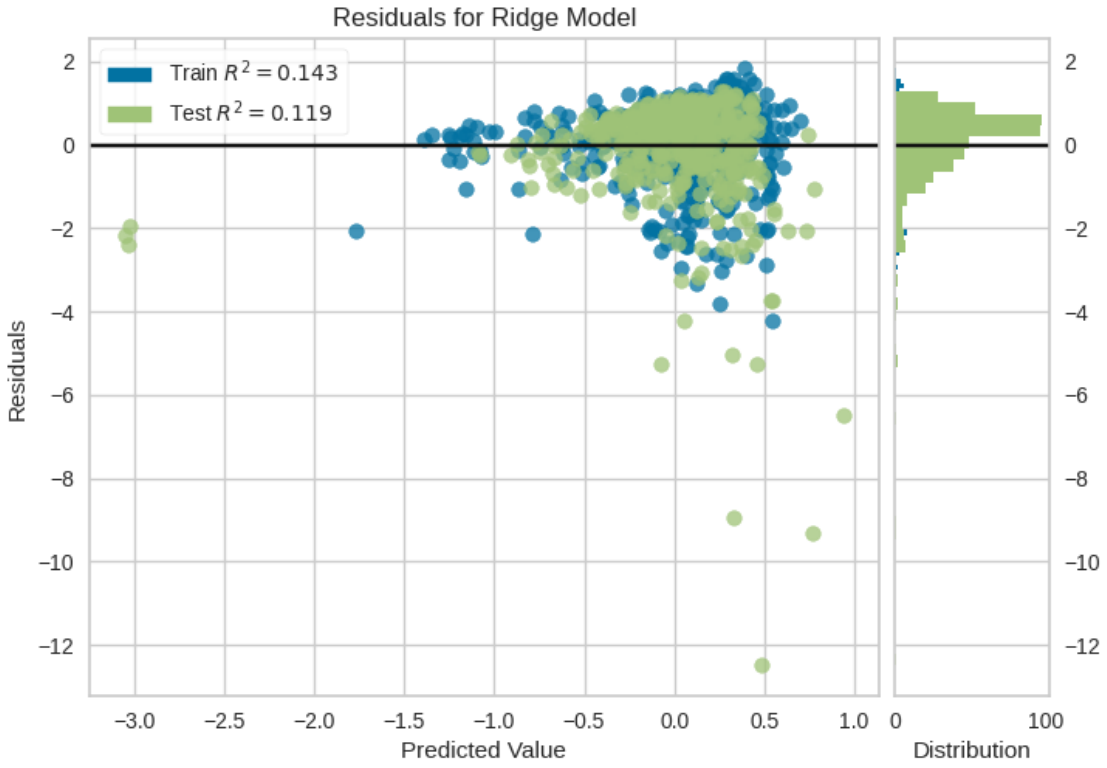

Figure 18: Residuals Plot

Training and test data's residuals generally follow a similar clustered pattern, mostly, overlapping within the same region. The model explained 11.05% variance in our DV (test set) which is similar to the training set(R2~14.32%). Therefore, suggesting the model fitted the test data fairly well but yielded poor performance.

4.7.2 Model II (B): Decision Tree Regression with same HyperParameters that were tuned for the Original Data

[654]:
```
#Creating a new DT regressor object with the best hyperparameters used fo 8(A)
DT_b = DecisionTreeRegressor(max_depth = grid_search.best_params_['max_depth'],
                             min_samples_leaf = grid_search.
↪best_params_['min_samples_leaf'],
```

```
                                    min_samples_split = grid_search.
 ↪best_params_['min_samples_split'])

# Fitting the DT regressor on the training data
DT_b.fit(X_train_b, y_train_b)

# Predicting the prices on the test data
y_pred_b = DT_b.predict(X_test)
#Demonstrating the DT Feature Importances
pd.DataFrame(DT_b.feature_importances_,
             synthetic_train.drop('price_dollars', axis = 1).columns,
             columns = ['Importances']).sort_values(by = 'Importances',␣
 ↪ascending = False).round(4)
```

[654]:
```
DecisionTreeRegressor(max_depth=4, min_samples_leaf=2)
```

[654]:

	Importances
room_type_Shared room	0.3932
review_scores_rating	0.3266
accommodates	0.1446
neighbourhood_group_De Baarsjes - Oud-West	0.0820
availability_30	0.0470
beds	0.0035
neighbourhood_group_Noord-West	0.0032
neighbourhood_group_Oostelijk Havengebied - Ind…	0.0000
room_type_Private room	0.0000
room_type_Hotel room	0.0000
room_type_Entire home/apt	0.0000
neighbourhood_group_Zuid	0.0000
neighbourhood_group_Westerpark	0.0000
neighbourhood_group_Watergraafsmeer	0.0000
neighbourhood_group_Slotervaart	0.0000
neighbourhood_group_Oud-Oost	0.0000
neighbourhood_group_Oud-Noord	0.0000
neighbourhood_group_Osdorp	0.0000
neighbourhood_group_Bijlmer-Oost	0.0000
neighbourhood_group_Bos en Lommer	0.0000
neighbourhood_group_Noord-Oost	0.0000
neighbourhood_group_Geuzenveld - Slotermeer	0.0000
neighbourhood_group_Gaasperdam - Driemond	0.0000
neighbourhood_group_De Pijp - Rivierenbuurt	0.0000
neighbourhood_group_Bijlmer-Centrum	0.0000
neighbourhood_group_De Aker - Nieuw Sloten	0.0000
neighbourhood_group_Centrum-West	0.0000
neighbourhood_group_Centrum-Oost	0.0000
neighbourhood_group_Buitenveldert - Zuidas	0.0000
neighbourhood_group_IJburg - Zeeburgereiland	0.0000

[657]:
```
# Predicting y values on the training set utilising the trained model
y_pred_train_b = DT_b.predict(X_train_b)

# Calculating the R-squared value on the training data
R2_DT_train_b = DT_b.score(X_train_b, y_train_b)
print(f'The R2 of the Decsion Tree(B) on training set: {R2_DT_train_b * 100:.
 ↪2f}%')
# Calculating the mean squared error on the training data
MSE_DT_train_b= mean_squared_error(y_train_b, y_pred_train_b)
print(f'The MSE of the Decsion Tree(B)  on training set: {MSE_DT_train_b:.2f}')

# Calculating the MAE on the test data
MAE_DT_train_b = mean_absolute_error(y_train_b, y_pred_train_b)
print(f'The MAE of the Decision Tree(B) on the training data is:␣
 ↪{MAE_DT_train_b:.2f}')
```

```
The R2 of the Decsion Tree(B) on training set: 18.20%
The MSE of the Decsion Tree(B)  on training set: 11505.71
The MAE of the Decision Tree(B) on the training data is: 79.26
```

[658]:
```
# Evaluating the performance of the best_dt_a model on the test data
DT_predictions_test_b = DT_b.predict(X_test)

R2_DT_test_b = DT_b.score(X_test, y_test)
print(f'The R2 of the Decsion Tree(B) on the test set: {R2_DT_test_b * 100:.
 ↪2f}%')
#MSE on test set
MSE_DT_test_b = mean_squared_error(y_test, DT_predictions_test_b)
print(f'The MSE of the Decsion Tree(B) on the test set: {MSE_DT_test_b:.2f}')

MAE_DT_test_b = mean_absolute_error(y_test, DT_predictions_test_b)
print(f'The MAE of the Decsion Tree(B) on the test data is:{MAE_DT_test_b:.2f}')
```

```
The R2 of the Decsion Tree(B) on the test set: 3.97%
The MSE of the Decsion Tree(B) on the test set: 33573.53
The MAE of the Decsion Tree(B) on the test data is:100.06
```

Visualising

[659]:
```
# Plotting the DT_b
plt.figure(figsize = (30,20))
plot_tree(DT_b, filled = True, feature_names = synthetic_train.columns[:-1],␣
 ↪fontsize = 10)
plt.show();
```

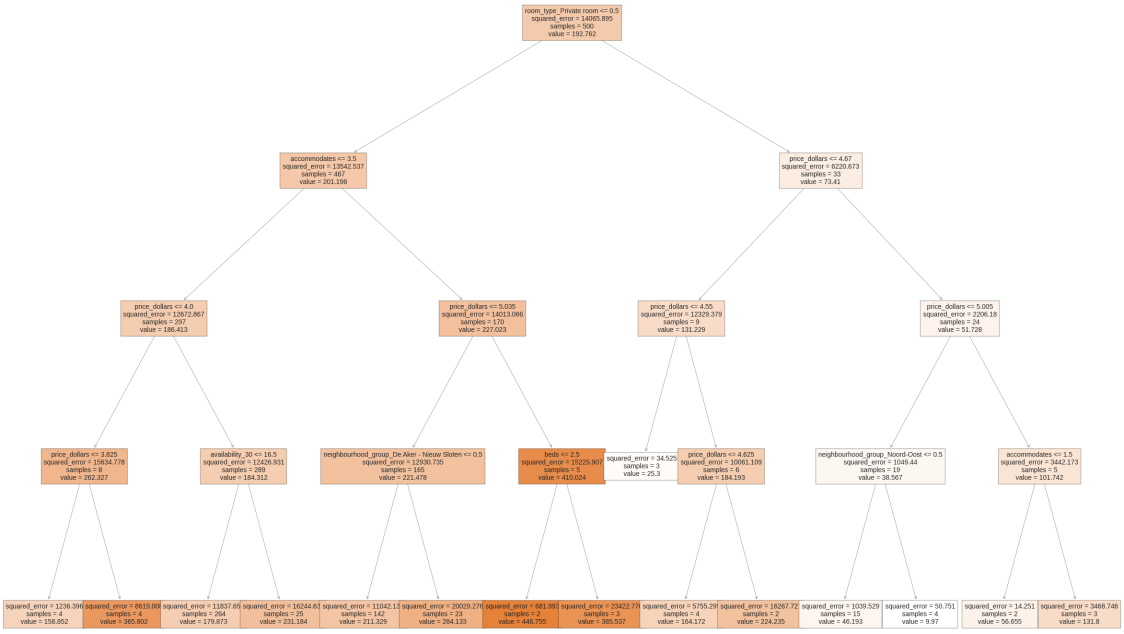

Figure 19: DT Plot

The decision tree shows the root feature as room_type_private_room spiltting off into the most important feature, beds <= 1.5 and neigbourhood_group_Centrum-West <= 0.5.

[660]:
```
#Creating a scatterplot of actual VS predicted values (test set)
plt.scatter(y_test, DT_predictions_test_b, color = "#6456DF");
#plt.plot([3, 1000], [3, 1000], c='#f0439e', lw=2)
# Adding a diagonal line for reference
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--',␣
 ↪c='pink')
plt.xlabel('Actual (test set)');
plt.ylabel('Predictions');
plt.title("Scatterplot of actual vs predicted values")
plt.show();
```

```
/tmp/ipykernel_134/1373118587.py:5: UserWarning: color is redundantly defined by
the 'color' keyword argument and the fmt string "k--" (-> color='k'). The
keyword argument will take precedence.
  plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--',
c='pink')
```

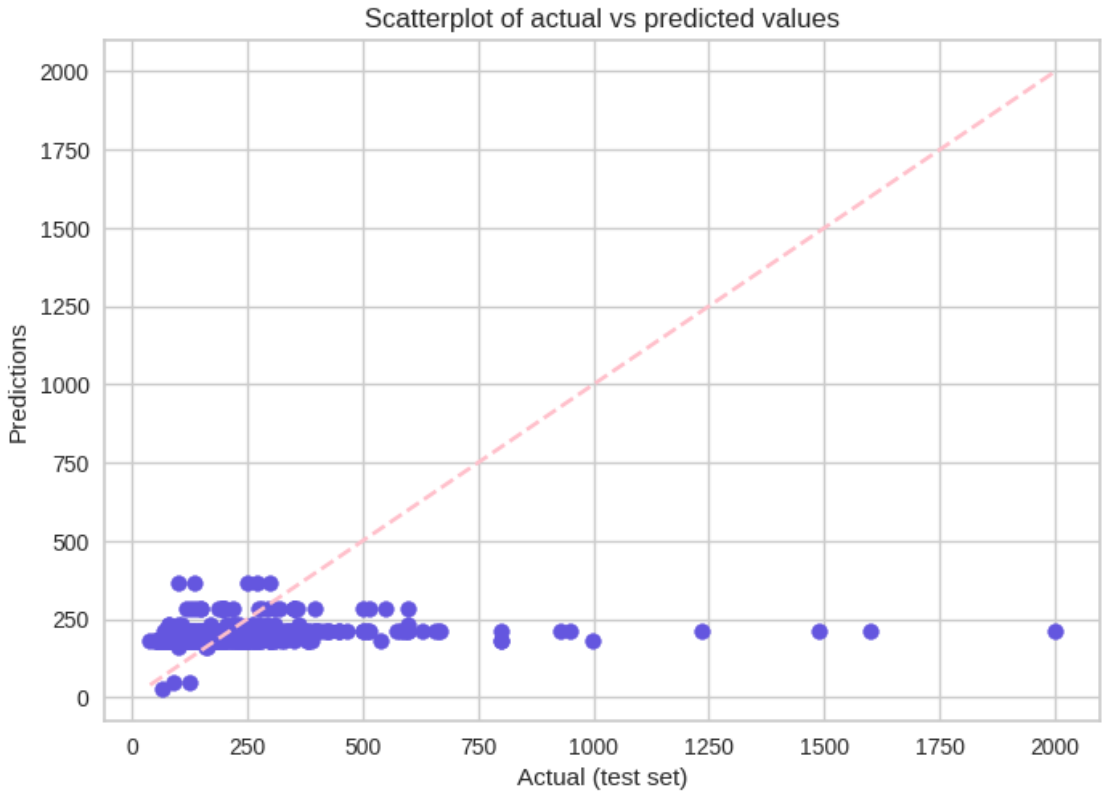

Figure 20: Scatterplot

The values do not follow a diagonal pattern and are generally clustered around one region. Indicating poor performance of the decision tree model.

[661]:
```
#Prediction Error
#Instantiating the visualizer with the fitted DT model
#Alpha parameter - controling for scatterplot points' opacity
visualizer = PredictionError(DT_b, alpha = 0.5)
#The training data is fitted into the visualiser
visualizer.fit(X_train_b, y_train_b)
#The model is evaluated on the test data
visualizer.score(X_test, y_test)

# Displaying the visualiser
visualizer.poof();
```

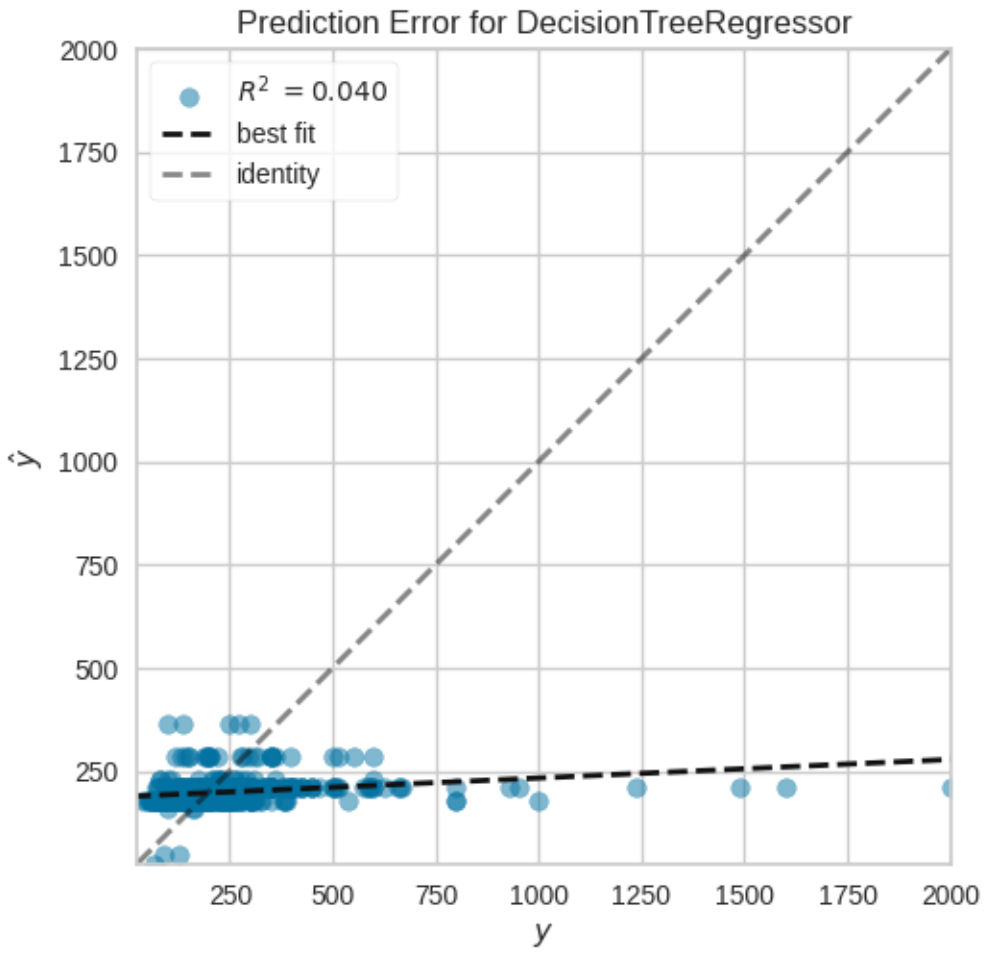

Figure 21:Prediction Error

There is a large deviation between the identity(actual) and the best fit line. The points are generally clustered around the line of best fit. The clustering of the points around the line of best fit suggests that the model is making more accurate predictions for some of the data points than for others.

[662]:
```
#Residuals
# Instantiating the visualizer with the fitted decision tree model
visualizer = ResidualsPlot(DT_b)

#Fitting training data into the visualiser
visualizer.fit(X_train_b, y_train_b)

# #Evaluating model on validation data
visualizer.score(X_test, y_test)

# Poofing the data/ Displaying the plot
visualizer.poof();
```

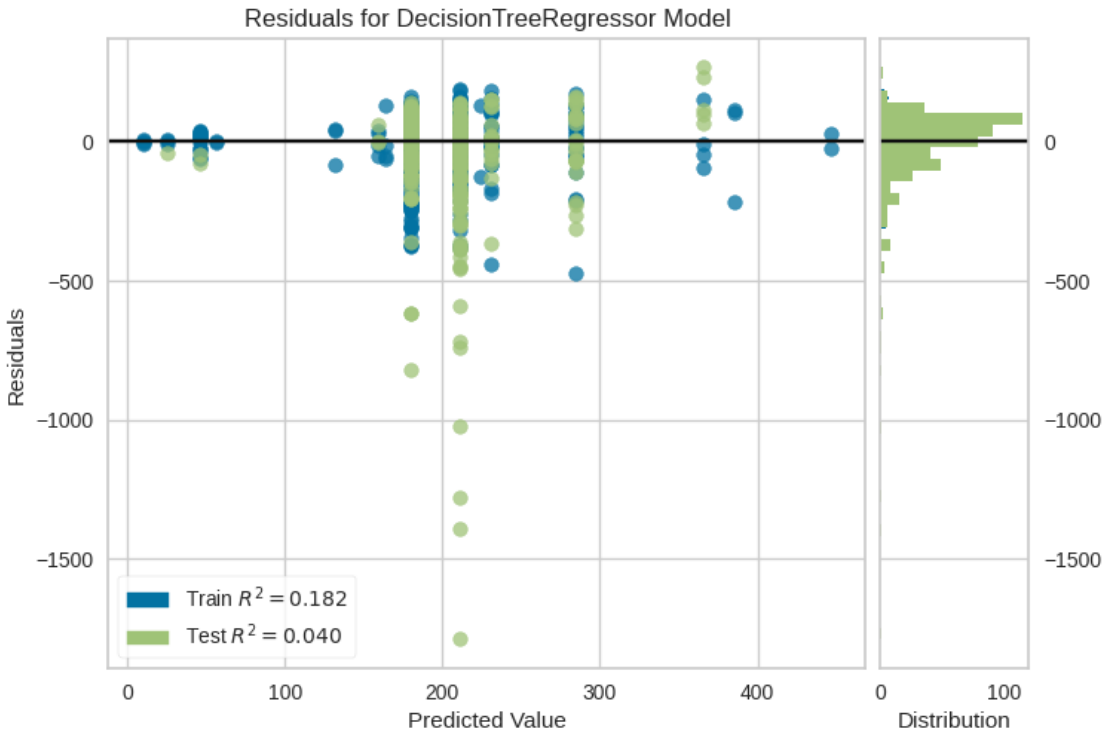

Figure 22: Residuals Plot

Training and test data's residuals generally follow a pattern of dispersed clusters and outliers away from zero. Meaning that the model is not accurately predicting the target variable, as there are significant differences between the predicted and actual values for many of the data points. The residuals away from zero indicates that there may be a bias in the model.

4.7.3 Model III (B): Neural Network Regression with the Same Hyperparameters as utilised for the Original Data

[664]:
```
# setting the same random seed to ensure reproducibility and comparability of␣
 ↪results
# setting the random seed for Numpy
np.random.seed(25)
# setting the random seed for Tensorflow
tf.random.set_seed(25)

# Defining the early stopping criteria
early_stop = EarlyStopping(monitor='val_loss', patience=10, verbose=1,
                           mode='min', restore_best_weights=True)

# defining the model architecture
NN_model_b = keras.Sequential([
    keras.layers.Dense(units=128, input_shape=(30,), activation='relu'),
```

```
    # adding a dropout layer with rate 0.5
    keras.layers.Dropout(0.5),
    keras.layers.Dense(units=64, activation='relu'),
    # adding a dropout layer with rate 0.5
    keras.layers.Dropout(0.5),
    keras.layers.Dense(units=32, activation='relu'),
    # adding a dropout layer with rate 0.3
    keras.layers.Dropout(0.3),
    keras.layers.Dense(units=16, activation='relu'),
    # adding a dropout layer with rate 0.2
    keras.layers.Dropout(0.2),
    keras.layers.Dense(units=8, activation='relu'),
    # adding a dropout layer with rate 0.2
    keras.layers.Dropout(0.2),
    keras.layers.Dense(units=1)
])

# Compiling the model with the 'adam' optimiser and MSE loss function
#to perform the regression task
#decreasing the learning rate to 0.02 (automatic = 0.001)
opt = keras.optimizers.Adam(learning_rate=0.0001)
NN_model_b.compile(optimizer=opt, loss='MSE', metrics=['mse', 'mae',␣
 ↪coeff_determination])
```

[665]:
```
#timing the traninng process
#start time
start_time_NN_b =  time.time()

# Training the model with early stopping and making 0.2 validation split
history = NN_model_b.fit(X_train_norm_b, y_train_norm_b,
                         epochs=128, batch_size=5,
                         validation_split=0.2, callbacks=[early_stop])


#end time
end_time_NN_b = time.time()
#calculating the difference
elapsed_time_NN_b = end_time_NN_b - start_time_NN_b

print(f"Elapsed time: {elapsed_time_NN_b:.2f} seconds")
```

```
Epoch 1/128
80/80 [==============================] - 1s 3ms/step - loss: 1.7847 - mse:
1.7847 - mae: 0.9925 - coeff_determination: -2.6411 - val_loss: 1.1270 -
val_mse: 1.1270 - val_mae: 0.8307 - val_coeff_determination: -0.6591
Epoch 2/128
80/80 [==============================] - 0s 2ms/step - loss: 1.5483 - mse:
```

```
1.5483 - mae: 0.9163 - coeff_determination: -2.7474 - val_loss: 1.1207 -
val_mse: 1.1207 - val_mae: 0.8340 - val_coeff_determination: -0.6630
Epoch 3/128
80/80 [==============================] - 0s 2ms/step - loss: 1.3355 - mse:
1.3355 - mae: 0.8766 - coeff_determination: -2.2231 - val_loss: 1.1200 -
val_mse: 1.1200 - val_mae: 0.8348 - val_coeff_determination: -0.6564
Epoch 4/128
80/80 [==============================] - 0s 2ms/step - loss: 1.4434 - mse:
1.4434 - mae: 0.9177 - coeff_determination: -2.2271 - val_loss: 1.1241 -
val_mse: 1.1241 - val_mae: 0.8371 - val_coeff_determination: -0.6578
Epoch 5/128
80/80 [==============================] - 0s 2ms/step - loss: 1.3266 - mse:
1.3266 - mae: 0.8860 - coeff_determination: -2.4145 - val_loss: 1.1232 -
val_mse: 1.1232 - val_mae: 0.8345 - val_coeff_determination: -0.6423
Epoch 6/128
80/80 [==============================] - 0s 2ms/step - loss: 1.0753 - mse:
1.0753 - mae: 0.7876 - coeff_determination: -1.0125 - val_loss: 1.1253 -
val_mse: 1.1253 - val_mae: 0.8355 - val_coeff_determination: -0.6450
Epoch 7/128
80/80 [==============================] - 0s 2ms/step - loss: 1.2627 - mse:
1.2627 - mae: 0.8468 - coeff_determination: -1.0829 - val_loss: 1.1203 -
val_mse: 1.1203 - val_mae: 0.8296 - val_coeff_determination: -0.6176
Epoch 8/128
80/80 [==============================] - 0s 2ms/step - loss: 1.1996 - mse:
1.1996 - mae: 0.8268 - coeff_determination: -1.3265 - val_loss: 1.1223 -
val_mse: 1.1223 - val_mae: 0.8298 - val_coeff_determination: -0.6160
Epoch 9/128
80/80 [==============================] - 0s 2ms/step - loss: 1.2481 - mse:
1.2481 - mae: 0.8399 - coeff_determination: -1.0053 - val_loss: 1.1243 -
val_mse: 1.1243 - val_mae: 0.8298 - val_coeff_determination: -0.6123
Epoch 10/128
80/80 [==============================] - 0s 2ms/step - loss: 1.1891 - mse:
1.1891 - mae: 0.8323 - coeff_determination: -1.3439 - val_loss: 1.1254 -
val_mse: 1.1254 - val_mae: 0.8292 - val_coeff_determination: -0.6040
Epoch 11/128
80/80 [==============================] - 0s 2ms/step - loss: 1.1753 - mse:
1.1753 - mae: 0.8031 - coeff_determination: -1.0799 - val_loss: 1.1259 -
val_mse: 1.1259 - val_mae: 0.8279 - val_coeff_determination: -0.5977
Epoch 12/128
80/80 [==============================] - 0s 2ms/step - loss: 1.0813 - mse:
1.0813 - mae: 0.8028 - coeff_determination: -1.5243 - val_loss: 1.1283 -
val_mse: 1.1283 - val_mae: 0.8281 - val_coeff_determination: -0.5985
Epoch 13/128
43/80 [===============>...] - ETA: 0s - loss: 1.0334 - mse: 1.0334 -
mae: 0.7835 - coeff_determination: -2.0698Restoring model weights from the end
of the best epoch: 3.
80/80 [==============================] - 0s 2ms/step - loss: 1.1232 - mse:
1.1232 - mae: 0.8139 - coeff_determination: -1.7031 - val_loss: 1.1286 -
```

```
val_mse: 1.1286 - val_mae: 0.8270 - val_coeff_determination: -0.5918
Epoch 00013: early stopping
Elapsed time: 2.33 seconds
```

[666]:
```
# Making predictions on the training set
NN_train_predictions_b = NN_model_b.predict(X_train_norm_b)

# Evaluating the model on the training set
train_loss_b, train_mse_b, train_mae_b, train_r2_b = NN_model_b.
 ↪evaluate(X_train_norm_b,

 ↪y_train_norm_b)

#saving the results to the variables to, consequently, compare the models
R2_NN_train_b = train_r2_b
MSE_NN_train_b = train_mse_b
MAE_NN_train_b = train_mae_b

#Printing the metrics on the training set
print(f'R2 of the NN (B) on the training set: {R2_NN_train_b * 100:.4f}%')
print(f'MSE of the NN (B) on the training set: {R2_NN_train_b :.4f}')
print(f'MAE of the NN (B) on the training set: {R2_NN_train_b :.4f}')
```

```
16/16 [==============================] - 0s 1ms/step - loss: 1.0252 - mse:
1.0252 - mae: 0.7896 - coeff_determination: -0.0975
R2 of the NN (B) on the training set: -9.7543%
MSE of the NN (B) on the training set: -0.0975
MAE of the NN (B) on the training set: -0.0975
```

[667]:
```
#Making predictions on the test set
NN_predictions_test_b = NN_model_b.predict(X_test_norm)

# Evaluating the performance of the best_dt_a model on the test set
test_loss_b, test_mse_b, test_mae_b, test_r2_b = NN_model_b.
 ↪evaluate(X_test_norm, y_test_norm)

#saving the results to the variables to, consequently, compare the models
R2_NN_test_b = test_r2_b
MSE_NN_test_b = test_mse_b
MAE_NN_test_b = test_mae_b

#Printing the metrics on the test set
print(f'R2 of the NN (B) on the test set:  {R2_NN_test_b  * 100:.4f}%')
print(f'MSE of the NN (B) on the test set:   {MSE_NN_test_b:.4f}')
print(f'MAE of the NN (B) on the test set:  {MAE_NN_test_b:.4f}')
```

```
16/16 [==============================] - 0s 1ms/step - loss: 1.8753 - mse:
1.8753 - mae: 0.8449 - coeff_determination: -0.0852
```

```
R2 of the NN (B) on the test set:  -8.5163%
MSE of the NN (B) on the test set:   1.8753
MAE of the NN (B) on the test set:  0.8449
```

Visualising

[668]:
```
# plotting the scatter plot
#** Note that for the Neural Network Regression, the axis of actual VS␣
 ↪Predicted Values
#are reversed comaring to Model 1 and 2
plt.scatter(NN_predictions_test_b, y_test_norm, color = '#6456DF')
# Adding a diagonal line for reference
plt.plot([y_test_norm.min(), y_test_norm.max()], [y_test_norm.min(),␣
 ↪y_test_norm.max()], 'k--', c = 'pink')
plt.xlabel('Predictions')
plt.ylabel('Actual (test set)')
plt.title('Scatterplot of actual vs predicted values')
plt.show();
```

```
/tmp/ipykernel_134/3819554391.py:6: UserWarning: color is redundantly defined by
the 'color' keyword argument and the fmt string "k--" (-> color='k'). The
keyword argument will take precedence.
  plt.plot([y_test_norm.min(), y_test_norm.max()], [y_test_norm.min(),
y_test_norm.max()], 'k--', c = 'pink')
```

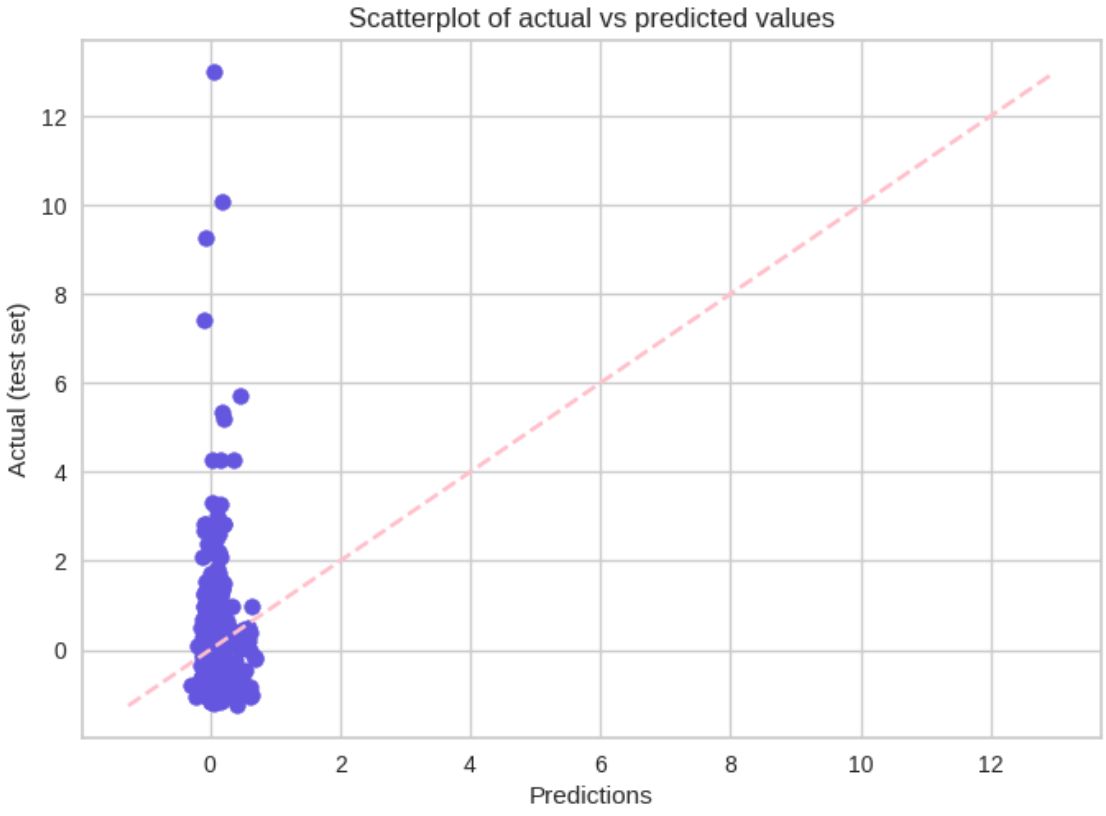

Figure 23: Scatterplot

The values mostly follow a vertical clustered pattern. Outliers can be observed. The clustered pattern suggests that the model is making similar predictions for a range of values.

4.8 Question 8 (C): 10,000 Synthetic Training Instances

Generating 10,000 synthetic data instances and preparing for the Predictive Models

[669]:
```
#Utilising the data that was not one-hot-encoded to generate 10,000 instances
#of synthetic data

df_used_no_OHE = shuffled_df[1000:3000:2]
```

[677]:
```
#random seed of 25 is specifiend as default in the create_synthetic_data␣
 ↪function
# Applying the 'create_synthetic_data' function created in Question 6 with␣
 ↪10,000 instances
synthetic_train_10k = create_synthetic_data(df_used_no_OHE, n_rows=10000)
```

[678]:
```
print('The synthetic training dataset (B) has {0} rows and {1} columns'.
 ↪format(len(synthetic_train_10k),

 ↪len(synthetic_train_10k.columns)))
```

```
The synthetic training dataset (B) has 10000 rows and 7 columns
```

[679]:
```
#Checking the first  rows in the created synthetic data
synthetic_train_10k.head()
```

[679]:

	neighbourhood_group	availability_30	room_type	accommodates	beds	price_dollars	review_scores_rating
0	Centrum-Oost	0.0	Entire home/apt	2.0	3.0	157.949997	5.00
1	Oud-Noord	18.0	Entire home/apt	4.0	2.0	272.489990	4.78
2	Centrum-West	0.0	Private room	2.0	1.0	139.080002	4.95
3	Buitenveldert - Zuidas	0.0	Entire home/apt	4.0	4.0	277.399994	5.00
4	Oud-Noord	0.0	Hotel room	2.0	1.0	214.960007	4.81

[680]:
```
# OHE For the 10,000 generated Synthetic Data on the categorical columns
encoded_data_s_10k = pd.get_dummies(synthetic_train_10k[['neighbourhood_group',
```

```
                                                        'room_type']])

# merging the encoded data back into the original dataframe
synthetic_train_10k = pd.concat([synthetic_train_10k, encoded_data_s_10k],
                    axis = 1).drop(synthetic_train_10k[['neighbourhood_group',
                                        'room_type']], axis = 1)

synthetic_train_10k.head()
```

[680]:

```
   availability_30  accommodates  beds  price_dollars  review_scores_rating  \
0              0.0           2.0   3.0     157.949997                  5.00
1             18.0           4.0   2.0     272.489990                  4.78
2              0.0           2.0   1.0     139.080002                  4.95
3              0.0           4.0   4.0     277.399994                  5.00
4              0.0           2.0   1.0     214.960007                  4.81

   neighbourhood_group_Bijlmer-Centrum  neighbourhood_group_Bijlmer-Oost  \
0                                    0                                 0
1                                    0                                 0
2                                    0                                 0
3                                    0                                 0
4                                    0                                 0

   neighbourhood_group_Bos en Lommer  \
0                                  0
1                                  0
2                                  0
3                                  0
4                                  0

   neighbourhood_group_Buitenveldert - Zuidas  \
0                                           0
1                                           0
2                                           0
3                                           1
4                                           0

   neighbourhood_group_Centrum-Oost  …  neighbourhood_group_Oud-Noord  \
0                                 1  …                              0
1                                 0  …                              1
2                                 0  …                              0
3                                 0  …                              0
4                                 0  …                              1

   neighbourhood_group_Oud-Oost  neighbourhood_group_Slotervaart  \
0                             0                                0
1                             0                                0
```

```
2                                0                                0
3                                0                                0
4                                0                                0

   neighbourhood_group_Watergraafsmeer  neighbourhood_group_Westerpark  \
0                                    0                               0
1                                    0                               0
2                                    0                               0
3                                    0                               0
4                                    0                               0

   neighbourhood_group_Zuid  room_type_Entire home/apt  room_type_Hotel room  \
0                         0                          1                     0
1                         0                          1                     0
2                         0                          0                     0
3                         0                          1                     0
4                         0                          0                     1

   room_type_Private room  room_type_Shared room
0                       0                     0
1                       0                     0
2                       1                     0
3                       0                     0
4                       0                     0

[5 rows x 31 columns]
```

Loading the Trainingand Test Data

[681]:
```
# loading the Train and Test set data
X_train_c= synthetic_train_10k.drop(columns = 'price_dollars', axis = 1).values
y_train_c = synthetic_train_10k.price_dollars.values
```

[682]:
```
# standardising the independent and target variables
scaler = StandardScaler()
X_train_norm_c = scaler.fit_transform(X_train_c)
y_train_norm_c = scaler.fit_transform(y_train_c.reshape((-1,1)))
```

4.8.1 Model I (C): Ridge Regression

[683]:
```
#Creating a Ridge Regression Object and Fitting the Training Data
Ridge_model_c = Ridge(alpha = 1.0).fit(X_train_norm_c, y_train_norm_c)

#Demonstrating the coefficients of the Ridge Regression Model
pd.DataFrame(Ridge_model_c.coef_.flatten(),
             synthetic_train_10k.drop('price_dollars', axis = 1).columns,
```

```
            columns = ['Coefficient']).sort_values(by = 'Coefficient',␣
↪ascending = False).round(4)
```

[683]:

	Coefficient
beds	0.1415
accommodates	0.1287
room_type_Entire home/apt	0.0557
neighbourhood_group_Centrum-Oost	0.0371
review_scores_rating	0.0305
neighbourhood_group_De Pijp - Rivierenbuurt	0.0234
neighbourhood_group_Centrum-West	0.0197
room_type_Private room	0.0186
room_type_Hotel room	0.0164
neighbourhood_group_Slotervaart	0.0159
neighbourhood_group_Zuid	0.0151
neighbourhood_group_De Baarsjes - Oud-West	0.0139
neighbourhood_group_Westerpark	0.0138
availability_30	0.0046
neighbourhood_group_Geuzenveld - Slotermeer	0.0046
neighbourhood_group_Bijlmer-Oost	-0.0014
neighbourhood_group_Gaasperdam - Driemond	-0.0034
neighbourhood_group_Buitenveldert - Zuidas	-0.0047
neighbourhood_group_Oud-Oost	-0.0075
neighbourhood_group_Bos en Lommer	-0.0155
neighbourhood_group_Watergraafsmeer	-0.0163
neighbourhood_group_De Aker - Nieuw Sloten	-0.0168
neighbourhood_group_IJburg - Zeeburgereiland	-0.0223
neighbourhood_group_Bijlmer-Centrum	-0.0228
neighbourhood_group_Noord-West	-0.0258
neighbourhood_group_Oostelijk Havengebied - Ind…	-0.0269
neighbourhood_group_Oud-Noord	-0.0269
neighbourhood_group_Osdorp	-0.0278
neighbourhood_group_Noord-Oost	-0.0286
room_type_Shared room	-0.1650

[684]:
```
# Predicting y values on the training set utilising the trained model
y_pred_train_c = Ridge_model_c.predict(X_train_norm_c)

#Model Accuracy training set
# Evaluating the Model Performance on the Training Data using the R-squared
R2_Ridge_train_c = Ridge_model_c.score(X_train_norm_c, y_train_norm_c)
# Printing the R2 value
print(f'R-squared of the Ridge Regression(C) on training set: {R2_Ridge_train_c␣
↪* 100:.2f}%')

#Evaluating the Model Performance on the training data using the Mean Squared␣
↪Error
```

```
MSE_Ridge_train_c = mean_squared_error(y_train_norm_c, y_pred_train_c)
# Printing the MSE value
print(f'The MSE of the Ridge Regression(C) on the training set:␣
 ↪{MSE_Ridge_train_c:.2f}')

MAE_Ridge_train_c = mean_absolute_error(y_train_c, y_pred_train_c)
print(f'The MAE of the Ridge Regression(C) on the training data is:␣
 ↪{MAE_Ridge_train_c:.2f}')
```

```
R-squared of the Ridge Regression(C) on training set: 11.88%
The MSE of the Ridge Regression(C) on the training set: 0.88
The MAE of the Ridge Regression(C) on the training data is: 217.13
```

[685]:
```
#Evaluating the Ridge Regression Model's Performance on the Test Set

# Calculating the predicted values
y_pred_Ridge_test_c = Ridge_model_c.predict(X_test_norm)

#Computing the R2  and MSE on the test set using these predicted values

# Calculating the R2 on the Test Set
R2_Ridge_test_c = Ridge_model_c.score(X_test_norm, y_test_norm)
print(f'The R-squared of the Ridge Regression(C) on the test set:␣
 ↪{R2_Ridge_test_c * 100:.2f}%')

#Calculating the MSE on the Test Set
MSE_Ridge_test_c = mean_squared_error(y_test_norm, y_pred_Ridge_test_c)
print(f'The M E of the Ridge Regression(C) on the test set: {MSE_Ridge_test_c:.
 ↪2f}')

MAE_Ridge_test_c = mean_absolute_error(y_test_norm, y_pred_Ridge_test_c)
print(f'The MAE of the Ridge Regression(C) on the test data is:␣
 ↪{MAE_Ridge_test_c:.2f}')
```

```
The R-squared of the Ridge Regression(C) on the test set: 18.05%
The M E of the Ridge Regression(C) on the test set: 1.51
The MAE of the Ridge Regression(C) on the test data is: 0.71
```

Visualising

[686]:
```
# Creating a scatterplot of actual vs predicted values
plt.scatter(y_test_norm, y_pred_Ridge_test_c, color = "#6456DF")

# Adding the plot title and axis labels
plt.title('Actual vs Predicted Values')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
```

```
# Adding a diagonal line for reference
plt.plot([y_test_norm.min(), y_test_norm.max()],
         [y_test_norm.min(), y_test_norm.max()], 'k--', c = 'pink')

# Showing the plot
plt.show();
```

```
/tmp/ipykernel_134/314357619.py:10: UserWarning: color is redundantly defined by
the 'color' keyword argument and the fmt string "k--" (-> color='k'). The
keyword argument will take precedence.
  plt.plot([y_test_norm.min(), y_test_norm.max()],
```

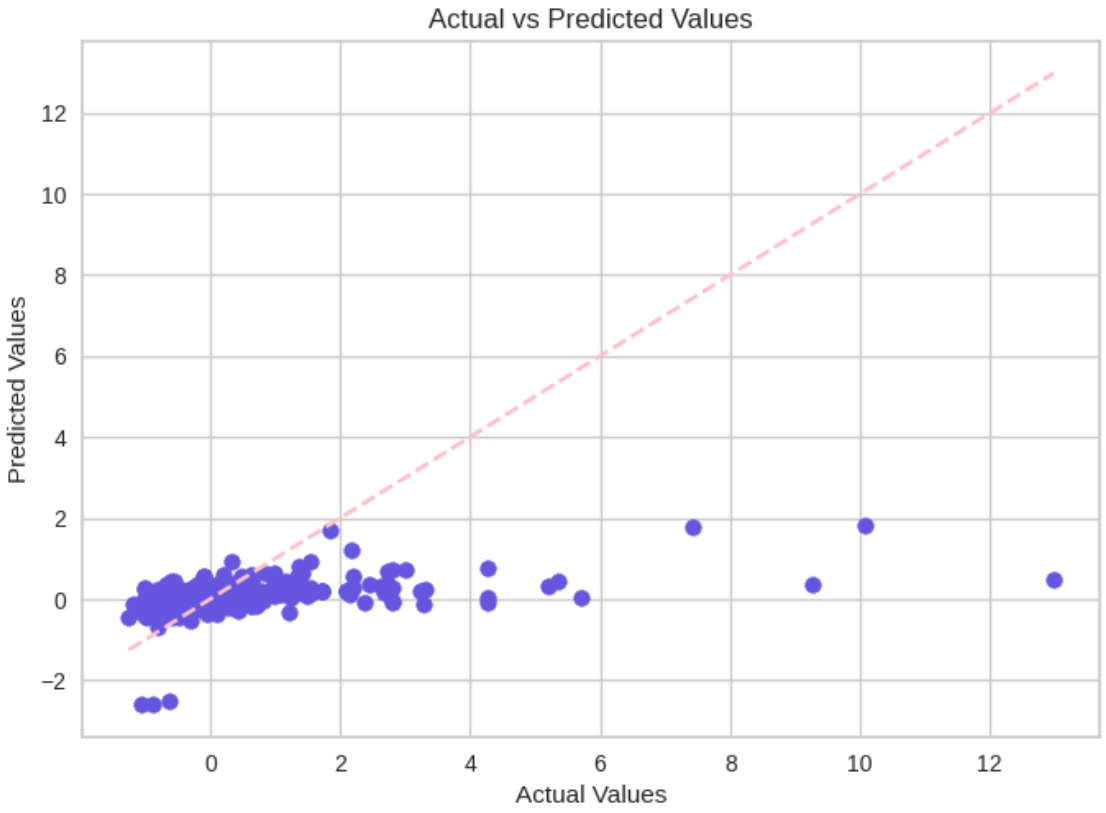

Figure 24: Scatterplot

The values do not follow a diagonal pattern and are generally clustered with a few outliers. Indicating poor performance of the ridge regression model on the data.

[245]:
```
#Residuals
visualizer = ResidualsPlot(Ridge_model_c)
visualizer.fit(X_train_norm_c, y_train_norm_c)
#Evaluating model on validation data
visualizer.score(X_test_norm, y_test_norm)
```

```
#Poofing/drawing data
graph3 = visualizer.poof();
plt.show();
```

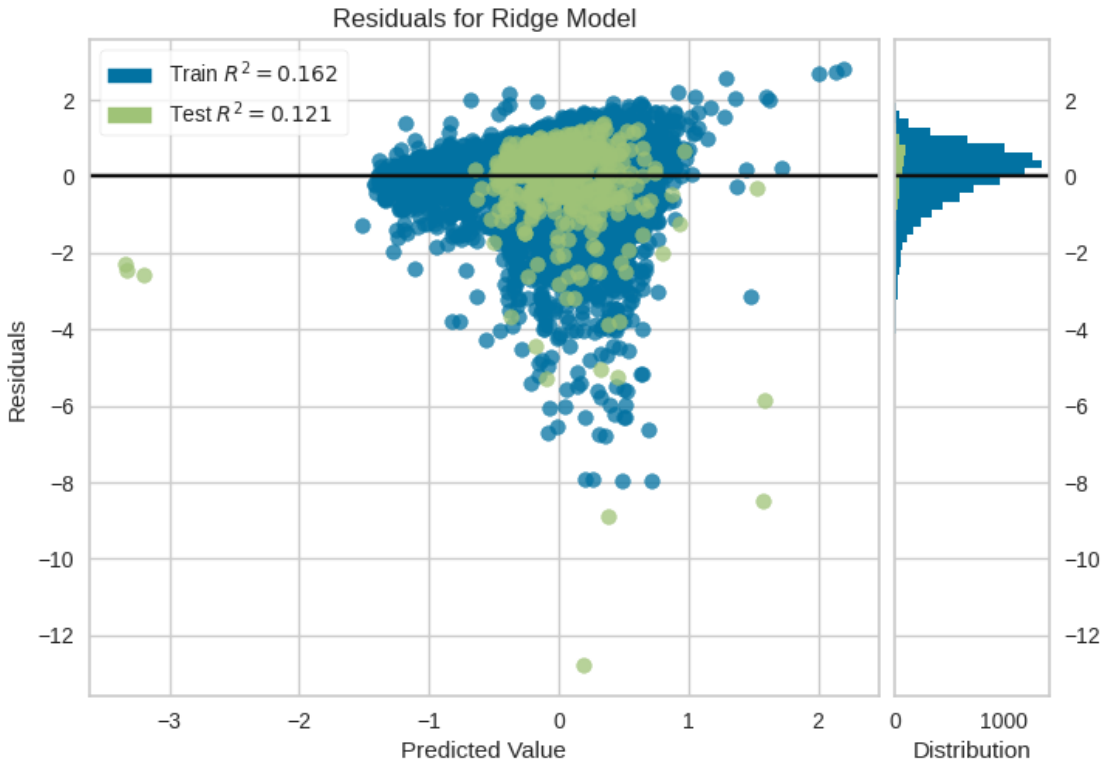

Figure 25: Residual Plot

The training and test data's residuals generally follow a similar pattern, mostly overlapping. The model explained 18.05% variance in our DV (test set) which is similar to the training set(R2~11.88%). Therefore, the model fitted the test poorly.

4.8.2 Model II (C): Decision Tree Regression with same HyperParameters that were tuned for the Original Data

[687]:
```
#Creating a new DT regressor object with the best hyperparameters used fo 8(A)
DT_c = DecisionTreeRegressor(max_depth = grid_search.best_params_['max_depth'],
                             min_samples_leaf = grid_search.
 ↪best_params_['min_samples_leaf'],
                             min_samples_split = grid_search.
 ↪best_params_['min_samples_split'])

# Fitting the DT regressor on the training data
DT_c.fit(X_train_c, y_train_c)
```

```
#Demonstrating the DT Feature Importances
pd.DataFrame(DT_c.feature_importances_,
             synthetic_train_10k.drop('price_dollars', axis = 1).columns,
             columns=['Importances']).sort_values(by = 'Importances',
                                                  ascending = False).round(4)
```

[687]:
```
DecisionTreeRegressor(max_depth=4, min_samples_leaf=2)
```

[687]:

	Importances
beds	0.4606
room_type_Shared room	0.3563
accommodates	0.1264
review_scores_rating	0.0233
availability_30	0.0200
neighbourhood_group_Centrum-West	0.0100
neighbourhood_group_Slotervaart	0.0034
neighbourhood_group_Bijlmer-Oost	0.0000
neighbourhood_group_Oostelijk Havengebied - Ind…	0.0000
room_type_Private room	0.0000
room_type_Hotel room	0.0000
room_type_Entire home/apt	0.0000
neighbourhood_group_Zuid	0.0000
neighbourhood_group_Westerpark	0.0000
neighbourhood_group_Watergraafsmeer	0.0000
neighbourhood_group_Oud-Oost	0.0000
neighbourhood_group_Oud-Noord	0.0000
neighbourhood_group_Osdorp	0.0000
neighbourhood_group_Noord-West	0.0000
neighbourhood_group_Bos en Lommer	0.0000
neighbourhood_group_Noord-Oost	0.0000
neighbourhood_group_Bijlmer-Centrum	0.0000
neighbourhood_group_Geuzenveld - Slotermeer	0.0000
neighbourhood_group_Gaasperdam - Driemond	0.0000
neighbourhood_group_De Pijp - Rivierenbuurt	0.0000
neighbourhood_group_De Baarsjes - Oud-West	0.0000
neighbourhood_group_De Aker - Nieuw Sloten	0.0000
neighbourhood_group_Centrum-Oost	0.0000
neighbourhood_group_Buitenveldert - Zuidas	0.0000
neighbourhood_group_IJburg - Zeeburgereiland	0.0000

[688]:
```
# Predicting y values on the training set utilising the trained model
DT_predictions_train_c = DT_c.predict(X_train_c)

# Calculating the R-squared value on the training data
R2_DT_train_c = DT_c.score(X_train_c, y_train_c)
print(f'The R2 of the Decision Tree(C) on training set: {R2_DT_train_c * 100:.
  ↪2f}%')
```

```
# Calculating the mean squared error on the training data
MSE_DT_train_c = mean_squared_error(y_train_c, DT_predictions_train_c)
print(f'The MSE of the Decision Tree(C) on the training set: {MSE_DT_train_c:.
 ↪2f}')

#MAE on training set
MAE_DT_train_c = mean_absolute_error(y_train_c, DT_predictions_train_c)
print(f'The  MAE of the Decision Tree(C) on the training data is:␣
 ↪{MAE_DT_train_c:.2f}')
```

```
The R2 of the Decision Tree(C) on training set: 12.22%
The MSE of the Decision Tree(C) on the training set: 15378.04
The  MAE of the Decision Tree(C) on the training data is: 85.67
```

[689]:
```
# Evaluating the performance of the DT_c model on the test data
DT_predictions_test_c = DT_c.predict(X_test)

R2_DT_test_c = DT_c.score(X_test, y_test)
print(f'The R2 of the Decision Tree(C) on the test set: {R2_DT_test_c * 100:.
 ↪2f}%')

#MSE on test set
MSE_DT_test_c = mean_squared_error(y_test, DT_predictions_test_c)
print(f'The MSE of the Decision Tree(C) on the test set: {MSE_DT_test_c:.2f}')

#MAE on test set
MAE_DT_test_c = mean_absolute_error(y_test, DT_predictions_test_c)
print(f'The MAE of the Decision Tree(C) on the test data is:{MAE_DT_test_c:.
 ↪2f}')
```

```
The R2 of the Decision Tree(C) on the test set: 11.97%
The MSE of the Decision Tree(C) on the test set: 30776.22
The MAE of the Decision Tree(C) on the test data is:101.61
```

Visualising

[690]:
```
# Plotting the DT_c
plt.figure(figsize = (30,20))
plot_tree(DT_c, filled = True, feature_names = synthetic_train_10k.columns[:
 ↪-1], fontsize = 10)
plt.show();
```

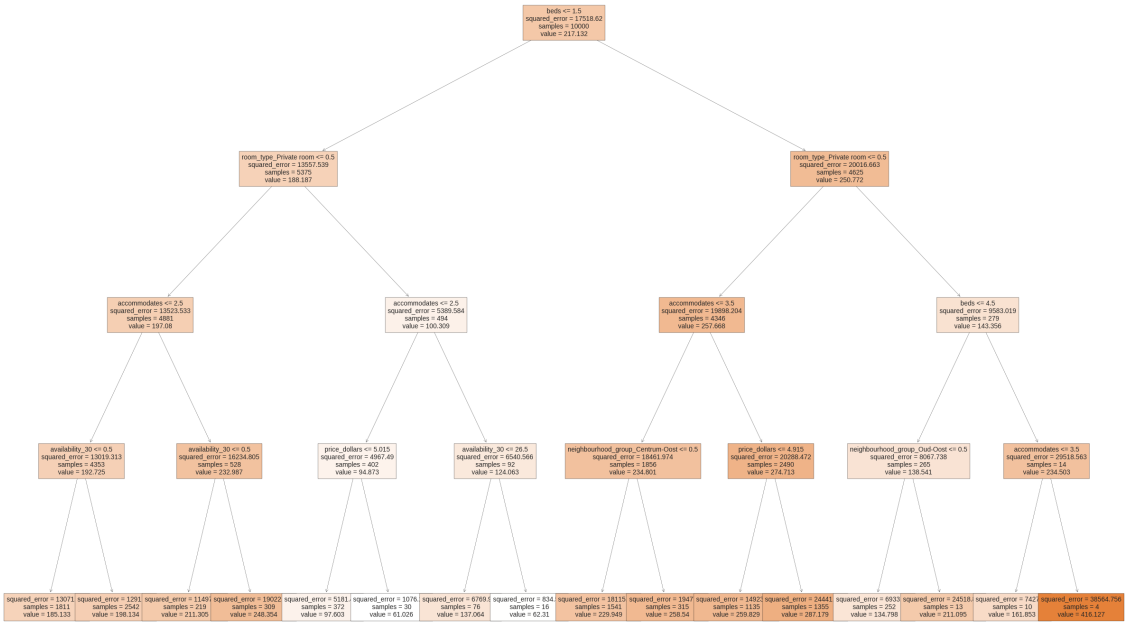

Figure 26: Decision Tree

The decision tree shows the root feature as beds spiltting off into the most important feature, room_type_Private room <= 0.5 and room_type_Private room <= 0.5.

[691]:
```
#Creating a scatterplot of actual VS predicted values (test set)
plt.scatter(y_test, DT_predictions_test_c, color = '#6456DF');
#plt.plot([3, 1000], [3, 1000], c='#f0439e', lw=2)
# Adding a diagonal line for reference
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--',␣
 ↪c='pink')
plt.xlabel('Actual (test set)');
plt.ylabel('Predictions');
plt.title("Scatterplot of actual vs predicted values")
plt.show();
```

```
/tmp/ipykernel_134/3762638976.py:5: UserWarning: color is redundantly defined by
the 'color' keyword argument and the fmt string "k--" (-> color='k'). The
keyword argument will take precedence.
  plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--',
c='pink')
```

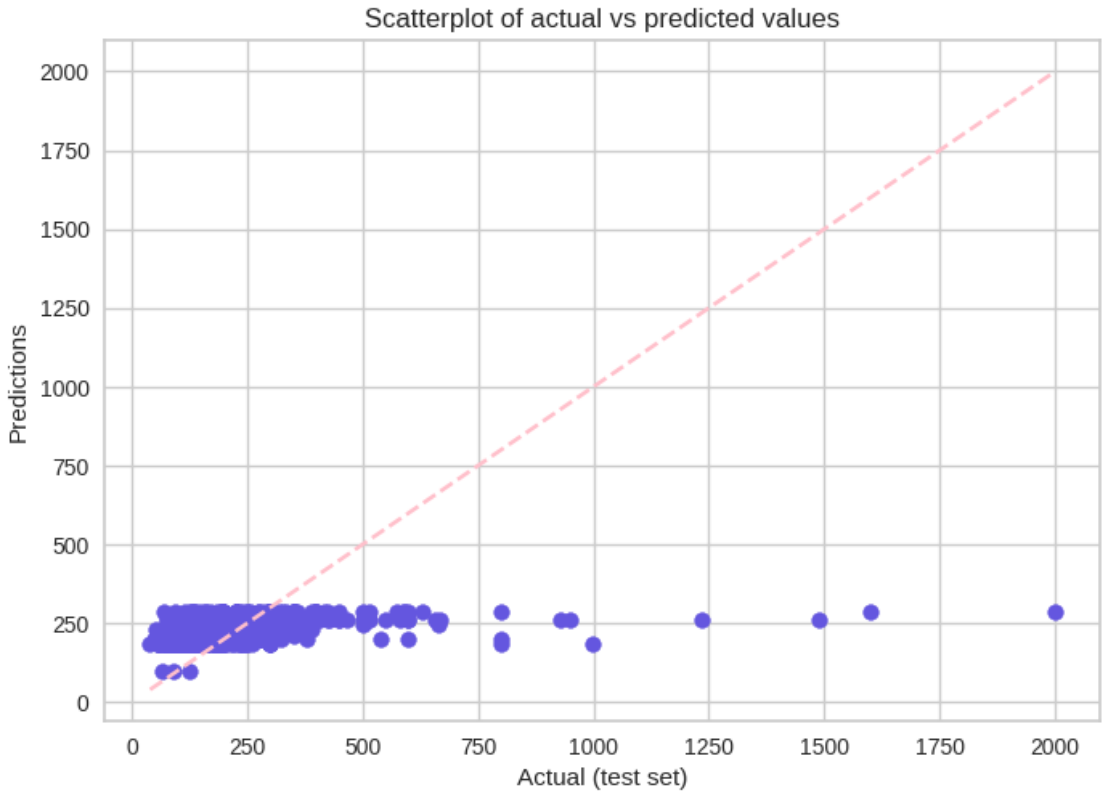

Figure 27: Scatterplot

The values do not follow a diagonal pattern and are generally clustered with a few outliers.

[251]:
```
#Prediction Error for DT_c
#Instantiating the visualizer with the fitted DT_c model
#Alpha parameter - controling for scatterplot points' opacity
visualizer = PredictionError(DT_c, alpha = 0.5)
#The training data is fitted into the visualiser
visualizer.fit(X_train_c, y_train_c)
#The model is evaluated on the test data
visualizer.score(X_test, y_test)

# Displaying the visualiser
visualizer.poof();
```

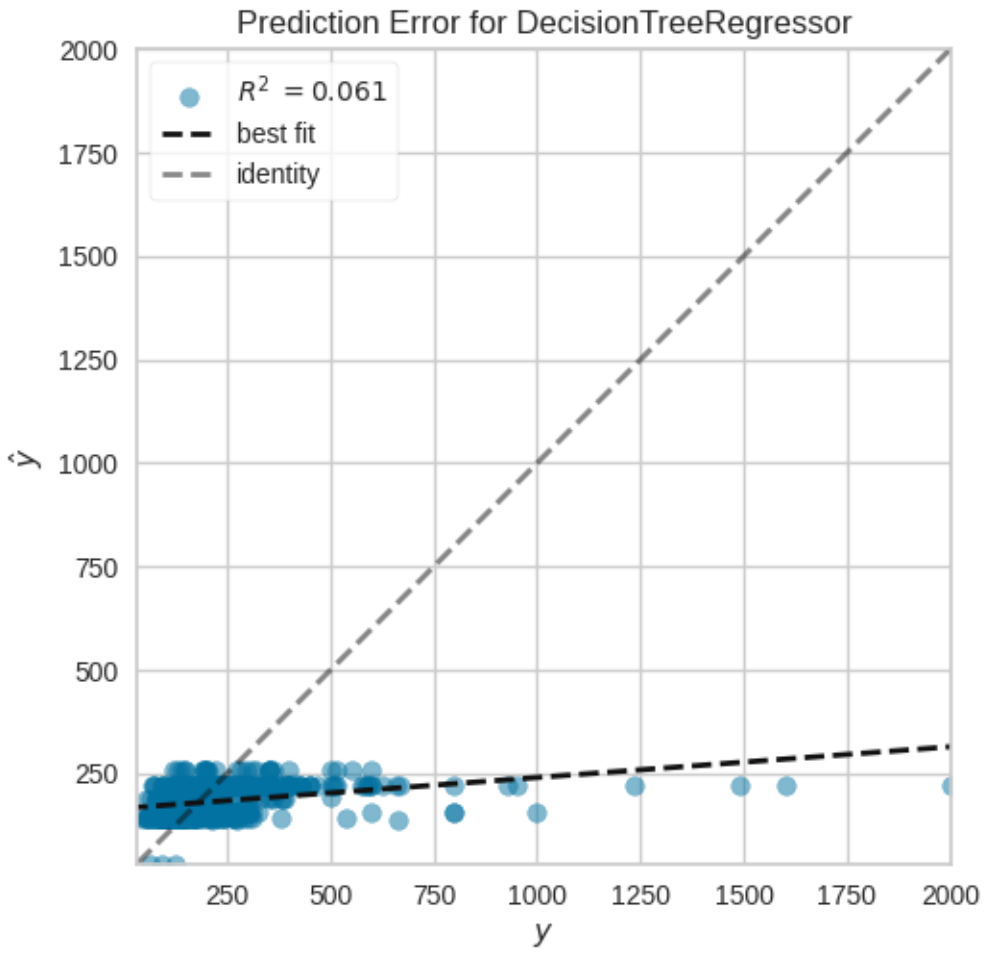

Figure 28: Prediction Error

There is a large difference between the identity line (actual data) and the best fit line. The points are generally clustered around the line of best fit, not the identity line. Thus, suggesting that the model is making more accurate predictions for some of the data points than for others.

[692]:
```
#Residuals for DT_c
# Instantiat   the visualizer with the fitted DT_c model
visualizer = ResidualsPlot(DT_c)

#Fitting the training data into the visualiser
visualizer.fit(X_train_c, y_train_c)

# Evaluating the model on the test data
visualizer.score(X_test, y_test)

# #Poofing/drawing data. Displaying the plot
visualizer.poof();
```

```
plt.show();
```

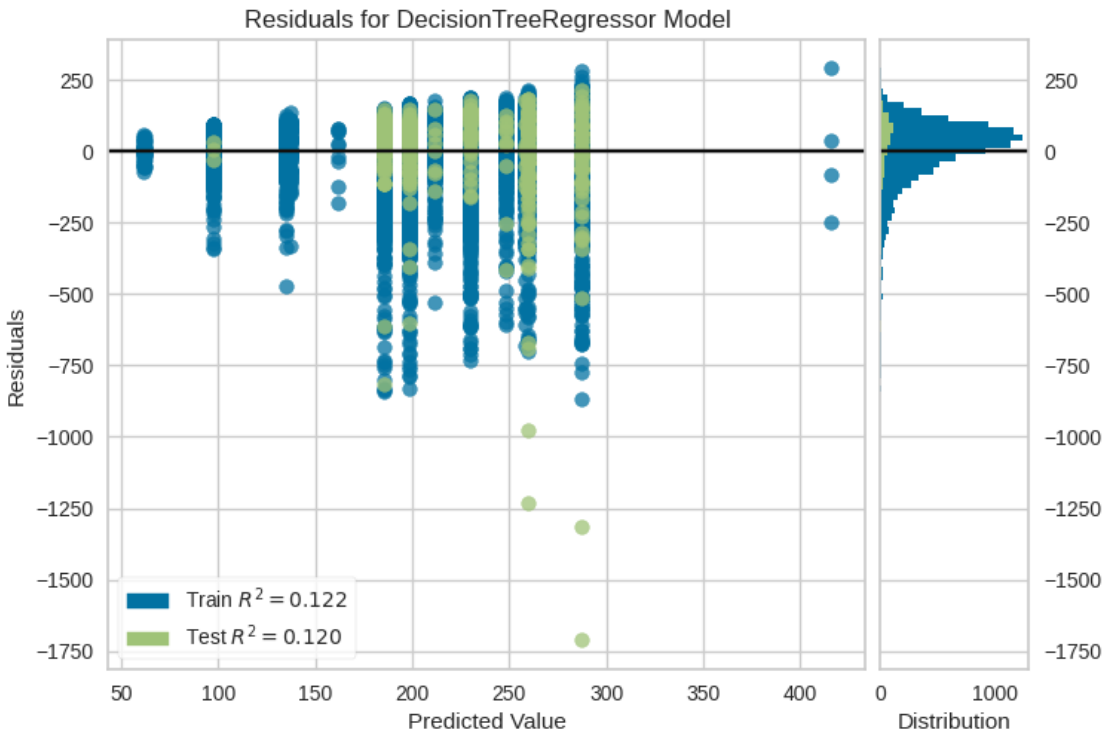

Figure 29: Residuals Plot

The training and test data residuals are generally in dispersed clusters. Many of these points are far from zero. Indicating that there is a difference between the actual and predicted data and there is also bias.

4.8.3 Model III (C): Neural Network Regression with same HyperParameters as Used for 8(A)

[693]:
```
# setting the same random seed to ensure reproducibility and comparability of␣
↪results
# random seed for Numpy
np.random.seed(25)
# random seed for Tensorflow
tf.random.set_seed(25)

# Defining the early stopping criteria
early_stop = EarlyStopping(monitor='val_loss', patience=10, verbose=1,
                           mode='min', restore_best_weights=True)

# defining the model architecture
NN_model_c = keras.Sequential([
```

```
    keras.layers.Dense(units=128, input_shape=(30,), activation='relu'),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(units=64, activation='relu'),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(units=32, activation='relu'),
    keras.layers.Dropout(0.3),
    keras.layers.Dense(units=16, activation='relu'),
    keras.layers.Dropout(0.2),
    keras.layers.Dense(units=8, activation='relu'),
    keras.layers.Dropout(0.2),
    keras.layers.Dense(units=1)
])

# Compiling the model. decreasing the learning rate to 0.02 (automatic = 0.001)
opt = keras.optimizers.Adam(learning_rate=0.0001)
NN_model_c.compile(optimizer=opt, loss='MSE', metrics=['mse', 'mae',␣
 ↪coeff_determination])
```

[694]:
```
#timing the traninng process
#start time
start_time_NN_c =  time.time()

# Training the model with early stopping
#taking the same a validation split
history = NN_model_c.fit(X_train_norm_c, y_train_norm_c,
                         epochs=128, batch_size=5,
                         validation_split=0.2, callbacks=[early_stop])



# Training the model with early stopping and making 0.2 validation split
history = NN_model_b.fit(X_train_norm_b, y_train_norm_b,
                         epochs=128, batch_size=5,
                         validation_split=0.2, callbacks=[early_stop])


#end time
end_time_NN_c = time.time()
#calculating the difference
elapsed_time_NN_c = end_time_NN_c - start_time_NN_c

print(f"Elapsed time: {elapsed_time_NN_c:.2f} seconds")
```

```
Epoch 1/128
1600/1600 [==============================] - 3s 1ms/step - loss: 1.2441 - mse:
1.2441 - mae: 0.7995 - coeff_determination: -2.3801 - val_loss: 0.9913 -
val_mse: 0.9913 - val_mae: 0.7238 - val_coeff_determination: -1.0277
```

```
Epoch 2/128
1600/1600 [==============================] - 2s 1ms/step - loss: 1.0401 - mse:
1.0401 - mae: 0.7291 - coeff_determination: -1.0395 - val_loss: 0.9942 -
val_mse: 0.9942 - val_mae: 0.7219 - val_coeff_determination: -1.0001
Epoch 3/128
1600/1600 [==============================] - 2s 1ms/step - loss: 1.0138 - mse:
1.0138 - mae: 0.7206 - coeff_determination: -1.0301 - val_loss: 0.9950 -
val_mse: 0.9950 - val_mae: 0.7213 - val_coeff_determination: -0.9918
Epoch 4/128
1600/1600 [==============================] - 2s 1ms/step - loss: 1.0045 - mse:
1.0045 - mae: 0.7137 - coeff_determination: -0.8666 - val_loss: 0.9951 -
val_mse: 0.9951 - val_mae: 0.7220 - val_coeff_determination: -0.9980
Epoch 5/128
1600/1600 [==============================] - 2s 1ms/step - loss: 0.9992 - mse:
0.9992 - mae: 0.7118 - coeff_determination: -0.9002 - val_loss: 0.9938 -
val_mse: 0.9938 - val_mae: 0.7217 - val_coeff_determination: -0.9982
Epoch 6/128
1600/1600 [==============================] - 2s 1ms/step - loss: 0.9846 - mse:
0.9846 - mae: 0.7053 - coeff_determination: -0.8230 - val_loss: 0.9895 -
val_mse: 0.9895 - val_mae: 0.7214 - val_coeff_determination: -1.0051
Epoch 7/128
1600/1600 [==============================] - 2s 1ms/step - loss: 0.9851 - mse:
0.9851 - mae: 0.7035 - coeff_determination: -0.8777 - val_loss: 0.9780 -
val_mse: 0.9780 - val_mae: 0.7172 - val_coeff_determination: -0.9760
Epoch 8/128
1600/1600 [==============================] - 2s 1ms/step - loss: 0.9867 - mse:
0.9867 - mae: 0.7051 - coeff_determination: -0.8226 - val_loss: 0.9771 -
val_mse: 0.9771 - val_mae: 0.7176 - val_coeff_determination: -0.9848
Epoch 9/128
1600/1600 [==============================] - 2s 1ms/step - loss: 0.9753 - mse:
0.9753 - mae: 0.7014 - coeff_determination: -0.8185 - val_loss: 0.9662 -
val_mse: 0.9662 - val_mae: 0.7126 - val_coeff_determination: -0.9512
Epoch 10/128
1600/1600 [==============================] - 2s 1ms/step - loss: 0.9752 - mse:
0.9752 - mae: 0.7011 - coeff_determination: -0.8185 - val_loss: 0.9556 -
val_mse: 0.9556 - val_mae: 0.7100 - val_coeff_determination: -0.9468
Epoch 11/128
1600/1600 [==============================] - 2s 1ms/step - loss: 0.9604 - mse:
0.9604 - mae: 0.6955 - coeff_determination: -0.9667 - val_loss: 0.9384 -
val_mse: 0.9384 - val_mae: 0.7019 - val_coeff_determination: -0.9131
Epoch 12/128
1600/1600 [==============================] - 2s 1ms/step - loss: 0.9643 - mse:
0.9643 - mae: 0.6976 - coeff_determination: -0.9632 - val_loss: 0.9353 -
val_mse: 0.9353 - val_mae: 0.6966 - val_coeff_determination: -0.8634
Epoch 13/128
1600/1600 [==============================] - 2s 1ms/step - loss: 0.9552 - mse:
0.9552 - mae: 0.6913 - coeff_determination: -1.6922 - val_loss: 0.9310 -
val_mse: 0.9310 - val_mae: 0.6983 - val_coeff_determination: -0.9033
```

```
Epoch 14/128
1600/1600 [==============================] - 2s 1ms/step - loss: 0.9512 - mse:
0.9512 - mae: 0.6908 - coeff_determination: -0.8455 - val_loss: 0.9254 -
val_mse: 0.9254 - val_mae: 0.6995 - val_coeff_determination: -0.9443
Epoch 15/128
1600/1600 [==============================] - 2s 1ms/step - loss: 0.9502 - mse:
0.9502 - mae: 0.6931 - coeff_determination: -1.0124 - val_loss: 0.9211 -
val_mse: 0.9211 - val_mae: 0.6948 - val_coeff_determination: -0.9047
Epoch 16/128
1600/1600 [==============================] - 2s 1ms/step - loss: 0.9420 - mse:
0.9420 - mae: 0.6875 - coeff_determination: -1.3311 - val_loss: 0.9185 -
val_mse: 0.9185 - val_mae: 0.6914 - val_coeff_determination: -0.8783
Epoch 17/128
1600/1600 [==============================] - 2s 1ms/step - loss: 0.9436 - mse:
0.9436 - mae: 0.6849 - coeff_determination: -0.7915 - val_loss: 0.9177 -
val_mse: 0.9177 - val_mae: 0.6934 - val_coeff_determination: -0.9017
Epoch 18/128
1600/1600 [==============================] - 2s 1ms/step - loss: 0.9456 - mse:
0.9456 - mae: 0.6877 - coeff_determination: -0.8997 - val_loss: 0.9180 -
val_mse: 0.9180 - val_mae: 0.6917 - val_coeff_determination: -0.8779
Epoch 19/128
1600/1600 [==============================] - 2s 1ms/step - loss: 0.9374 - mse:
0.9374 - mae: 0.6812 - coeff_determination: -0.8493 - val_loss: 0.9123 -
val_mse: 0.9123 - val_mae: 0.6901 - val_coeff_determination: -0.8824
Epoch 20/128
1600/1600 [==============================] - 2s 1ms/step - loss: 0.9361 - mse:
0.9361 - mae: 0.6818 - coeff_determination: -0.8068 - val_loss: 0.9083 -
val_mse: 0.9083 - val_mae: 0.6882 - val_coeff_determination: -0.8788
Epoch 21/128
1600/1600 [==============================] - 2s 1ms/step - loss: 0.9335 - mse:
0.9335 - mae: 0.6815 - coeff_determination: -0.8510 - val_loss: 0.9088 -
val_mse: 0.9088 - val_mae: 0.6899 - val_coeff_determination: -0.9008
Epoch 22/128
1600/1600 [==============================] - 2s 1ms/step - loss: 0.9307 - mse:
0.9307 - mae: 0.6809 - coeff_determination: -0.9940 - val_loss: 0.9056 -
val_mse: 0.9056 - val_mae: 0.6882 - val_coeff_determination: -0.8966
Epoch 23/128
1600/1600 [==============================] - 2s 1ms/step - loss: 0.9258 - mse:
0.9258 - mae: 0.6781 - coeff_determination: -0.8676 - val_loss: 0.9003 -
val_mse: 0.9003 - val_mae: 0.6848 - val_coeff_determination: -0.8796
Epoch 24/128
1600/1600 [==============================] - 2s 1ms/step - loss: 0.9279 - mse:
0.9279 - mae: 0.6794 - coeff_determination: -0.8351 - val_loss: 0.9030 -
val_mse: 0.9030 - val_mae: 0.6883 - val_coeff_determination: -0.9119
Epoch 25/128
1600/1600 [==============================] - 2s 1ms/step - loss: 0.9238 - mse:
0.9238 - mae: 0.6770 - coeff_determination: -0.9653 - val_loss: 0.9004 -
val_mse: 0.9004 - val_mae: 0.6863 - val_coeff_determination: -0.8986
```

```
Epoch 26/128
1600/1600 [==============================] - 2s 1ms/step - loss: 0.9213 - mse:
0.9213 - mae: 0.6745 - coeff_determination: -0.9580 - val_loss: 0.9022 -
val_mse: 0.9022 - val_mae: 0.6890 - val_coeff_determination: -0.9235
Epoch 27/128
1600/1600 [==============================] - 2s 1ms/step - loss: 0.9183 - mse:
0.9183 - mae: 0.6747 - coeff_determination: -0.9528 - val_loss: 0.8990 -
val_mse: 0.8990 - val_mae: 0.6859 - val_coeff_determination: -0.8991
Epoch 28/128
1600/1600 [==============================] - 2s 1ms/step - loss: 0.9288 - mse:
0.9288 - mae: 0.6790 - coeff_determination: -0.9441 - val_loss: 0.9016 -
val_mse: 0.9016 - val_mae: 0.6841 - val_coeff_determination: -0.8641
Epoch 29/128
1600/1600 [==============================] - 2s 1ms/step - loss: 0.9217 - mse:
0.9217 - mae: 0.6741 - coeff_determination: -0.8376 - val_loss: 0.8981 -
val_mse: 0.8981 - val_mae: 0.6832 - val_coeff_determination: -0.8730
Epoch 30/128
1600/1600 [==============================] - 2s 1ms/step - loss: 0.9212 - mse:
0.9212 - mae: 0.6726 - coeff_determination: -0.8279 - val_loss: 0.8966 -
val_mse: 0.8966 - val_mae: 0.6840 - val_coeff_determination: -0.8904
Epoch 31/128
1600/1600 [==============================] - 2s 1ms/step - loss: 0.9205 - mse:
0.9205 - mae: 0.6729 - coeff_determination: -0.7391 - val_loss: 0.8945 -
val_mse: 0.8945 - val_mae: 0.6814 - val_coeff_determination: -0.8729
Epoch 32/128
1600/1600 [==============================] - 2s 1ms/step - loss: 0.9114 - mse:
0.9114 - mae: 0.6694 - coeff_determination: -1.1492 - val_loss: 0.8956 -
val_mse: 0.8956 - val_mae: 0.6811 - val_coeff_determination: -0.8667
Epoch 33/128
1600/1600 [==============================] - 2s 1ms/step - loss: 0.9156 - mse:
0.9156 - mae: 0.6681 - coeff_determination: -0.6293 - val_loss: 0.8989 -
val_mse: 0.8989 - val_mae: 0.6867 - val_coeff_determination: -0.9193
Epoch 34/128
1600/1600 [==============================] - 2s 1ms/step - loss: 0.9103 - mse:
0.9103 - mae: 0.6720 - coeff_determination: -0.8065 - val_loss: 0.8905 -
val_mse: 0.8905 - val_mae: 0.6795 - val_coeff_determination: -0.8830
Epoch 35/128
1600/1600 [==============================] - 2s 1ms/step - loss: 0.9153 - mse:
0.9153 - mae: 0.6712 - coeff_determination: -0.7159 - val_loss: 0.8943 -
val_mse: 0.8943 - val_mae: 0.6817 - val_coeff_determination: -0.8912
Epoch 36/128
1600/1600 [==============================] - 2s 1ms/step - loss: 0.9060 - mse:
0.9060 - mae: 0.6679 - coeff_determination: -0.9602 - val_loss: 0.8923 -
val_mse: 0.8923 - val_mae: 0.6836 - val_coeff_determination: -0.9262
Epoch 37/128
1600/1600 [==============================] - 2s 1ms/step - loss: 0.9058 - mse:
0.9058 - mae: 0.6693 - coeff_determination: -0.7410 - val_loss: 0.8912 -
val_mse: 0.8912 - val_mae: 0.6781 - val_coeff_determination: -0.8633
```

```
Epoch 38/128
1600/1600 [==============================] - 2s 1ms/step - loss: 0.9062 - mse:
0.9062 - mae: 0.6651 - coeff_determination: -0.6836 - val_loss: 0.8943 -
val_mse: 0.8943 - val_mae: 0.6830 - val_coeff_determination: -0.9145
Epoch 39/128
1600/1600 [==============================] - 2s 1ms/step - loss: 0.9095 - mse:
0.9095 - mae: 0.6682 - coeff_determination: -1.3149 - val_loss: 0.8903 -
val_mse: 0.8903 - val_mae: 0.6810 - val_coeff_determination: -0.9097
Epoch 40/128
1600/1600 [==============================] - 2s 1ms/step - loss: 0.9074 - mse:
0.9074 - mae: 0.6680 - coeff_determination: -0.9824 - val_loss: 0.8876 -
val_mse: 0.8876 - val_mae: 0.6763 - val_coeff_determination: -0.8653
Epoch 41/128
1600/1600 [==============================] - 2s 1ms/step - loss: 0.9079 - mse:
0.9079 - mae: 0.6660 - coeff_determination: -0.8290 - val_loss: 0.8907 -
val_mse: 0.8907 - val_mae: 0.6809 - val_coeff_determination: -0.9147
Epoch 42/128
1600/1600 [==============================] - 2s 1ms/step - loss: 0.9055 - mse:
0.9055 - mae: 0.6660 - coeff_determination: -1.0100 - val_loss: 0.8892 -
val_mse: 0.8892 - val_mae: 0.6827 - val_coeff_determination: -0.9514
Epoch 43/128
1600/1600 [==============================] - 2s 1ms/step - loss: 0.9002 - mse:
0.9002 - mae: 0.6649 - coeff_determination: -1.1454 - val_loss: 0.8872 -
val_mse: 0.8872 - val_mae: 0.6831 - val_coeff_determination: -0.9720
Epoch 44/128
1600/1600 [==============================] - 2s 1ms/step - loss: 0.9047 - mse:
0.9047 - mae: 0.6675 - coeff_determination: -0.7980 - val_loss: 0.8872 -
val_mse: 0.8872 - val_mae: 0.6829 - val_coeff_determination: -0.9715
Epoch 45/128
1600/1600 [==============================] - 2s 1ms/step - loss: 0.9079 - mse:
0.9079 - mae: 0.6675 - coeff_determination: -0.7605 - val_loss: 0.8871 -
val_mse: 0.8871 - val_mae: 0.6803 - val_coeff_determination: -0.9283
Epoch 46/128
1600/1600 [==============================] - 2s 1ms/step - loss: 0.9008 - mse:
0.9008 - mae: 0.6661 - coeff_determination: -0.7832 - val_loss: 0.8870 -
val_mse: 0.8870 - val_mae: 0.6808 - val_coeff_determination: -0.9410
Epoch 47/128
1600/1600 [==============================] - 2s 1ms/step - loss: 0.9060 - mse:
0.9060 - mae: 0.6673 - coeff_determination: -0.8418 - val_loss: 0.8880 -
val_mse: 0.8880 - val_mae: 0.6788 - val_coeff_determination: -0.9127
Epoch 48/128
1600/1600 [==============================] - 2s 1ms/step - loss: 0.8992 - mse:
0.8992 - mae: 0.6614 - coeff_determination: -0.7129 - val_loss: 0.8843 -
val_mse: 0.8843 - val_mae: 0.6775 - val_coeff_determination: -0.9233
Epoch 49/128
1600/1600 [==============================] - 2s 1ms/step - loss: 0.9064 - mse:
0.9064 - mae: 0.6657 - coeff_determination: -1.1032 - val_loss: 0.8909 -
val_mse: 0.8909 - val_mae: 0.6822 - val_coeff_determination: -0.9398
```

```
Epoch 50/128
1600/1600 [==============================] - 2s 1ms/step - loss: 0.8995 - mse:
0.8995 - mae: 0.6631 - coeff_determination: -0.8503 - val_loss: 0.8886 -
val_mse: 0.8886 - val_mae: 0.6811 - val_coeff_determination: -0.9481
Epoch 51/128
1600/1600 [==============================] - 2s 1ms/step - loss: 0.9021 - mse:
0.9021 - mae: 0.6646 - coeff_determination: -0.7749 - val_loss: 0.8874 -
val_mse: 0.8874 - val_mae: 0.6815 - val_coeff_determination: -0.9623
Epoch 52/128
1600/1600 [==============================] - 2s 1ms/step - loss: 0.8979 - mse:
0.8979 - mae: 0.6628 - coeff_determination: -0.8148 - val_loss: 0.8866 -
val_mse: 0.8866 - val_mae: 0.6807 - val_coeff_determination: -0.9515
Epoch 53/128
1600/1600 [==============================] - 2s 1ms/step - loss: 0.8988 - mse:
0.8988 - mae: 0.6638 - coeff_determination: -0.6996 - val_loss: 0.8853 -
val_mse: 0.8853 - val_mae: 0.6802 - val_coeff_determination: -0.9535
Epoch 54/128
1600/1600 [==============================] - 2s 1ms/step - loss: 0.8975 - mse:
0.8975 - mae: 0.6631 - coeff_determination: -0.9808 - val_loss: 0.8859 -
val_mse: 0.8859 - val_mae: 0.6808 - val_coeff_determination: -0.9600
Epoch 55/128
1600/1600 [==============================] - 2s 1ms/step - loss: 0.8974 - mse:
0.8974 - mae: 0.6623 - coeff_determination: -0.8110 - val_loss: 0.8836 -
val_mse: 0.8836 - val_mae: 0.6749 - val_coeff_determination: -0.8955
Epoch 56/128
1600/1600 [==============================] - 2s 1ms/step - loss: 0.8973 - mse:
0.8973 - mae: 0.6623 - coeff_determination: -0.7862 - val_loss: 0.8895 -
val_mse: 0.8895 - val_mae: 0.6836 - val_coeff_determination: -0.9786
Epoch 57/128
1600/1600 [==============================] - 2s 1ms/step - loss: 0.8998 - mse:
0.8998 - mae: 0.6650 - coeff_determination: -0.6830 - val_loss: 0.8873 -
val_mse: 0.8873 - val_mae: 0.6809 - val_coeff_determination: -0.9578
Epoch 58/128
1600/1600 [==============================] - 2s 1ms/step - loss: 0.8969 - mse:
0.8969 - mae: 0.6608 - coeff_determination: -0.7073 - val_loss: 0.8846 -
val_mse: 0.8846 - val_mae: 0.6790 - val_coeff_determination: -0.9547
Epoch 59/128
1600/1600 [==============================] - 2s 1ms/step - loss: 0.9006 - mse:
0.9006 - mae: 0.6617 - coeff_determination: -0.8299 - val_loss: 0.8888 -
val_mse: 0.8888 - val_mae: 0.6812 - val_coeff_determination: -0.9513
Epoch 60/128
1600/1600 [==============================] - 2s 1ms/step - loss: 0.8967 - mse:
0.8967 - mae: 0.6623 - coeff_determination: -0.8097 - val_loss: 0.8849 -
val_mse: 0.8849 - val_mae: 0.6795 - val_coeff_determination: -0.9623
Epoch 61/128
1600/1600 [==============================] - 2s 1ms/step - loss: 0.8991 - mse:
0.8991 - mae: 0.6626 - coeff_determination: -0.7576 - val_loss: 0.8843 -
val_mse: 0.8843 - val_mae: 0.6783 - val_coeff_determination: -0.9472
```

```
Epoch 62/128
1600/1600 [==============================] - 2s 1ms/step - loss: 0.8978 - mse:
0.8978 - mae: 0.6640 - coeff_determination: -0.8048 - val_loss: 0.8867 -
val_mse: 0.8867 - val_mae: 0.6827 - val_coeff_determination: -0.9947
Epoch 63/128
1600/1600 [==============================] - 2s 1ms/step - loss: 0.9011 - mse:
0.9011 - mae: 0.6644 - coeff_determination: -0.8190 - val_loss: 0.8875 -
val_mse: 0.8875 - val_mae: 0.6824 - val_coeff_determination: -0.9838
Epoch 64/128
1600/1600 [==============================] - 2s 1ms/step - loss: 0.8942 - mse:
0.8942 - mae: 0.6610 - coeff_determination: -0.8332 - val_loss: 0.8859 -
val_mse: 0.8859 - val_mae: 0.6806 - val_coeff_determination: -0.9621
Epoch 65/128
1567/1600 [============================>.] - ETA: 0s - loss: 0.9072 - mse:
0.9072 - mae: 0.6660 - coeff_determination: -0.7925Restoring model weights from
the end of the best epoch: 55.
1600/1600 [==============================] - 2s 1ms/step - loss: 0.9009 - mse:
0.9009 - mae: 0.6645 - coeff_determination: -0.7887 - val_loss: 0.8867 -
val_mse: 0.8867 - val_mae: 0.6808 - val_coeff_determination: -0.9588
Epoch 00065: early stopping
Epoch 1/128
80/80 [==============================] - 0s 2ms/step - loss: 1.2817 - mse:
1.2817 - mae: 0.8545 - coeff_determination: -1.6039 - val_loss: 1.1259 -
val_mse: 1.1259 - val_mae: 0.8390 - val_coeff_determination: -0.6673
Epoch 2/128
80/80 [==============================] - 0s 2ms/step - loss: 1.2271 - mse:
1.2271 - mae: 0.8200 - coeff_determination: -1.7629 - val_loss: 1.1307 -
val_mse: 1.1307 - val_mae: 0.8400 - val_coeff_determination: -0.6671
Epoch 3/128
80/80 [==============================] - 0s 2ms/step - loss: 1.1103 - mse:
1.1103 - mae: 0.8027 - coeff_determination: -1.5675 - val_loss: 1.1290 -
val_mse: 1.1290 - val_mae: 0.8373 - val_coeff_determination: -0.6526
Epoch 4/128
80/80 [==============================] - 0s 2ms/step - loss: 1.2710 - mse:
1.2710 - mae: 0.8740 - coeff_determination: -1.8344 - val_loss: 1.1310 -
val_mse: 1.1310 - val_mae: 0.8366 - val_coeff_determination: -0.6453
Epoch 5/128
80/80 [==============================] - 0s 2ms/step - loss: 1.2164 - mse:
1.2164 - mae: 0.8570 - coeff_determination: -2.0265 - val_loss: 1.1287 -
val_mse: 1.1287 - val_mae: 0.8331 - val_coeff_determination: -0.6292
Epoch 6/128
80/80 [==============================] - 0s 2ms/step - loss: 1.0014 - mse:
1.0014 - mae: 0.7641 - coeff_determination: -0.8200 - val_loss: 1.1273 -
val_mse: 1.1273 - val_mae: 0.8320 - val_coeff_determination: -0.6242
Epoch 7/128
80/80 [==============================] - 0s 2ms/step - loss: 1.1746 - mse:
1.1746 - mae: 0.8201 - coeff_determination: -0.8812 - val_loss: 1.1226 -
val_mse: 1.1226 - val_mae: 0.8264 - val_coeff_determination: -0.5974
```

```
Epoch 8/128
80/80 [==============================] - 0s 2ms/step - loss: 1.1135 - mse:
1.1135 - mae: 0.7993 - coeff_determination: -1.1261 - val_loss: 1.1243 -
val_mse: 1.1243 - val_mae: 0.8267 - val_coeff_determination: -0.5963
Epoch 9/128
80/80 [==============================] - 0s 2ms/step - loss: 1.1656 - mse:
1.1656 - mae: 0.8124 - coeff_determination: -0.8079 - val_loss: 1.1273 -
val_mse: 1.1273 - val_mae: 0.8270 - val_coeff_determination: -0.5957
Epoch 10/128
80/80 [==============================] - 0s 2ms/step - loss: 1.1293 - mse:
1.1293 - mae: 0.8119 - coeff_determination: -1.1489 - val_loss: 1.1279 -
val_mse: 1.1279 - val_mae: 0.8263 - val_coeff_determination: -0.5927
Epoch 11/128
80/80 [==============================] - 0s 2ms/step - loss: 1.1145 - mse:
1.1145 - mae: 0.7841 - coeff_determination: -0.9267 - val_loss: 1.1287 -
val_mse: 1.1287 - val_mae: 0.8258 - val_coeff_determination: -0.5906
Epoch 12/128
80/80 [==============================] - 0s 2ms/step - loss: 1.0398 - mse:
1.0398 - mae: 0.7883 - coeff_determination: -1.3840 - val_loss: 1.1309 -
val_mse: 1.1309 - val_mae: 0.8263 - val_coeff_determination: -0.5925
Epoch 13/128
80/80 [==============================] - 0s 2ms/step - loss: 1.0744 - mse:
1.0744 - mae: 0.7936 - coeff_determination: -1.4808 - val_loss: 1.1305 -
val_mse: 1.1305 - val_mae: 0.8254 - val_coeff_determination: -0.5870
Epoch 14/128
80/80 [==============================] - 0s 2ms/step - loss: 1.0500 - mse:
1.0500 - mae: 0.7758 - coeff_determination: -0.6318 - val_loss: 1.1321 -
val_mse: 1.1321 - val_mae: 0.8254 - val_coeff_determination: -0.5878
Epoch 15/128
80/80 [==============================] - 0s 2ms/step - loss: 1.0792 - mse:
1.0792 - mae: 0.7852 - coeff_determination: -0.9698 - val_loss: 1.1332 -
val_mse: 1.1332 - val_mae: 0.8249 - val_coeff_determination: -0.5865
Epoch 16/128
80/80 [==============================] - 0s 2ms/step - loss: 1.0575 - mse:
1.0575 - mae: 0.7683 - coeff_determination: -0.6350 - val_loss: 1.1340 -
val_mse: 1.1340 - val_mae: 0.8248 - val_coeff_determination: -0.5866
Epoch 17/128
43/80 [===============>…] - ETA: 0s - loss: 0.8524 - mse: 0.8524 -
mae: 0.6974 - coeff_determination: -0.5323Restoring model weights from the end
of the best epoch: 7.
80/80 [==============================] - 0s 2ms/step - loss: 0.9832 - mse:
0.9832 - mae: 0.7501 - coeff_determination: -0.4901 - val_loss: 1.1343 -
val_mse: 1.1343 - val_mae: 0.8242 - val_coeff_determination: -0.5846
Epoch 00017: early stopping
Elapsed time: 148.53 seconds
```

[695]:
```
# Making predictions on the training set
NN_train_predictions_c = NN_model_c.predict(X_train_norm_c)

# Evaluating the model on the training set
train_loss_c, train_mse_c, train_mae_c, train_r2_c = NN_model_c.
 ↪evaluate(X_train_norm_c,

 ↪y_train_norm_c)

R2_NN_train_c = train_r2_c
MSE_NN_train_c = train_mse_c
MAE_NN_train_c = train_mae_c

#Printing the metrics on the training set
print(f'R2 of the NN (C) on the training set: {R2_NN_train_c * 100:.4f}%')
print(f'MSE of the NN (C) on the training set: {MSE_NN_train_c:.4f}')
print(f'MAE of the NN (C) on the training set: {MAE_NN_train_c:.4f}')
```

```
313/313 [==============================] - 0s 1ms/step - loss: 0.8913 - mse:
0.8913 - mae: 0.6745 - coeff_determination: 0.0808
R2 of the NN (C) on the training set: 8.0775%
MSE of the NN (C) on the training set: 0.8913
MAE of the NN (C) on the training set: 0.6745
```

[696]:
```
#Making predictions on the test set
NN_predictions_test_c = NN_model_c.predict(X_test_norm)

# Evaluating the performance of the best_dt_a model on the test set
test_loss_c, test_mse_c, test_mae_c, test_r2_c = NN_model_c.
 ↪evaluate(X_test_norm,

 ↪y_test_norm)

#saving the results to the variables to, consequently, compare the models
R2_NN_test_c = test_r2_c
MSE_NN_test_c = test_mse_c
MAE_NN_test_c = test_mae_c

#Printing the metrics on the test set
print(f'R2 of the NN (C) on the test set: {R2_NN_test_c * 100:.4f}%')
print(f'MSE of the NN (C) on the test set: {MSE_NN_test_c:.4f}')
print(f'MAE of the NN (C) on the test set: {MAE_NN_test_c :.4f}')
```

```
16/16 [==============================] - 0s 1ms/step - loss: 1.6427 - mse:
1.6427 - mae: 0.7464 - coeff_determination: 0.0904
R2 of the NN (C) on the test set: 9.0442%
MSE of the NN (C) on the test set: 1.6427
```

```
MAE of the NN (C) on the test set: 0.7464
```

Visualising

[697]:
```
# plotting the scatter plot
#** Note that for the Neural Network Regression, the axis of actual VS␣
 ↪Predicted Values
#are reversed comaring to Model 1 and 2
plt.scatter(NN_predictions_test_c, y_test_norm, color = '#6456DF')
# Adding a diagonal line for reference
plt.plot([y_test_norm.min(), y_test_norm.max()], [y_test_norm.min(),␣
 ↪y_test_norm.max()], 'k--', c = 'pink')
plt.xlabel('Predictions')
plt.ylabel('Actual (test set)')
plt.title('Scatterplot of actual vs predicted values')
plt.show();
```

```
/tmp/ipykernel_134/2508197739.py:6: UserWarning: color is redundantly defined by
the 'color' keyword argument and the fmt string "k--" (-> color='k'). The
keyword argument will take precedence.
  plt.plot([y_test_norm.min(), y_test_norm.max()], [y_test_norm.min(),
y_test_norm.max()], 'k--', c = 'pink')
```

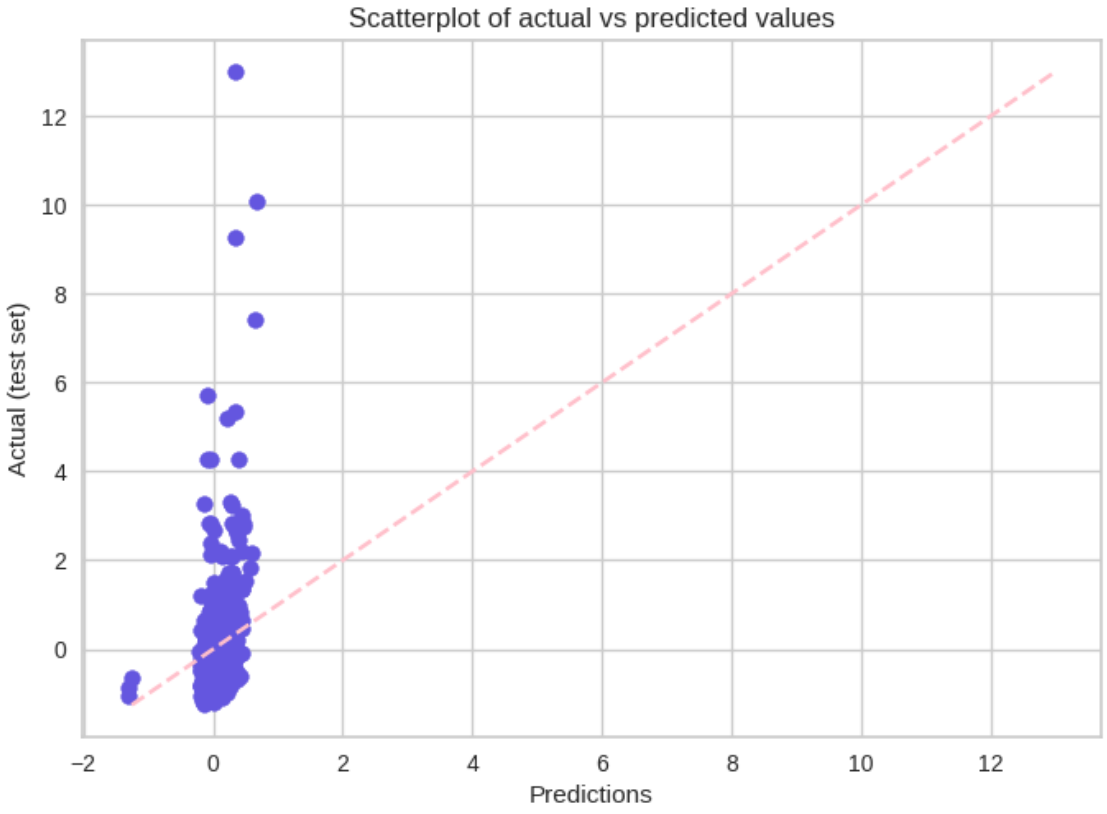

Figure 30: Scatterplot

In this figure, the values follow a vertical formation with a clustered pattern. Many outliers can be observed. The concentrated values suggest that the model is making similar predictions for a range of values.

4.9 Question 9: Comparing the Models' Performance

4.9.1 Model Results

MODEL 1: Ridge Regression

[698]:
```
#R2
print(f'Model 1A R2: {R2_Ridge_test_a * 100:.4f}%')
print(f'Model 1B R2: {R2_Ridge_test_b * 100:.4f}%')
print(f'Model 1C R2: {R2_Ridge_test_c * 100:.4f}%')
```

```
Model 1A R2: 40.0223%
Model 1B R2: 11.9231%
Model 1C R2: 18.0535%
```

[699]:
```
#MSE
print(f'Model 1A MSE: {MSE_Ridge_test_a:.4f}')
print(f'Model 1B MSE: {MSE_Ridge_test_b:.4f}')
print(f'Model 1C MSE: {MSE_Ridge_test_c:.4f}')
```

```
Model 1A MSE: 1.1060
Model 1B MSE: 1.6242
Model 1C MSE: 1.5111
```

[707]:
```
#MAE
print(f'Model 1A MAE: {MAE_Ridge_test_a:.4f}')
print(f'Model 1B MAE: {MAE_Ridge_test_b:.4f}')
print(f'Model 1C MAE: {MAE_Ridge_test_c:.4f}')
```

```
Model 1A MAE: 0.6092
Model 1B MAE: 0.7525
Model 1C MAE: 0.7059
```

MODEL 2: Decision Tree Regression

[701]:
```
#R2
print(f'Model 2A R2: {R2_DT_test_a * 100:.4f}%')
print(f'Model 2B R2: {R2_DT_test_b * 100:.4f}%')
print(f'Model 2C R2: {R2_DT_test_c * 100:.4f}%')
```

```
Model 2A R2: 25.3511%
Model 2B R2: 3.9680%
Model 2C R2: 11.9693%
```

[702]:
```
#MSE
print(f'Model 2A MSE: {MSE_DT_test_a:.2f}')
print(f'Model 2B MSE: {MSE_DT_test_b:.2f}')
print(f'Model 2C MSE: {MSE_DT_test_c:.2f}')
```

```
Model 2A MSE: 26097.84
Model 2B MSE: 33573.53
Model 2C MSE: 30776.22
```

[708]:
```
#MAE
print(f'Model 2A MAE: {MAE_DT_test_a:.4f}')
print(f'Model 2B MAE: {MAE_DT_test_b:.4f}')
print(f'Model 2C MAE: {MAE_DT_test_c:.4f}')
```

```
Model 2A MAE: 88.9317
Model 2B MAE: 100.0570
Model 2C MAE: 101.6121
```

MODEL 3: Neural Network Regression

[704]:
```
#R2
print(f'Model 3A R2: {R2_NN_test_a  * 100:.4f}%')
print(f'Model 3B R2: {R2_NN_test_b  * 100:.4f}%')
print(f'Model 3C R2: {R2_NN_test_c * 100:.4f}%')
```

```
Model 3A R2: 20.5750%
Model 3B R2: -8.5163%
Model 3C R2: 9.0442%
```

[705]:
```
#MSE
print(f'Model 3A MSE: {MSE_NN_test_a:.4f}')
print(f'Model 3B MSE: {MSE_NN_test_b:.4f}')
print(f'Model 3C MSE: {MSE_NN_test_c:.4f}')
```

```
Model 3A MSE: 1.3814
Model 3B MSE: 1.8753
Model 3C MSE: 1.6427
```

[706]:
```
#MAE
print(f'Model 3A MAE: {MAE_NN_test_a:.4f}')
print(f'Model 3B MAE: {MAE_NN_test_b:.4f}')
print(f'Model 3C MAE: {MAE_NN_test_c :.4f}')
```

```
Model 3A MAE: 0.6878
Model 3B MAE: 0.8449
Model 3C MAE: 0.7464
```

4.9.2 Model Comparison

Original data models outperformed synthetic data models as expected due to correlation differences. Although not explaining substantial DV variance, original data had higher prediction accuracy on all models with lower MAE&MSE

Ridge and DT Regression showed original data differed significantly from both synthetic datasets and explained more variance in 'price_dollars' (higher R2). Amid synthetic data, 10,000 instances performed better with higher R2 and slightly-lower MSE/MAE, while 500 instances performed worst across all models.

The Neural Network's(NN) R2 for 500 synthetic training instances was negative, failing to capture any original data's patterns, providing misleading prediction worse than price_dollars' mean value, not benefitting Airbnb and, potentially, compromising revenue. Nevertheless, NN's accuracy metrics substantially improved upon 10,000 instances. Thus, since NNs' performance increases with additional data, more synthetic data can be generated to enhance training data's coverage and diversity, improving models' generalisation(Sung,1998).

CTGAN's ability to capture joint distributions was hindered by few epochs number and low batch size(**computing constraints)(Vokhmina et al.,2015). To generate superior synthetic data, function parameters should be adjusted or another generation process attempted.Thus, high differences indicate inefficient original data replication(Chalé&Bastian, 2021).

Furthermore, despite additional instances aiding to improve synthetic data's performance, predicting listings' prices is highly-intricate even on original data, being highlydynamic and involving multiple interactions between features. Thus, additional variables and data points should be included upon generation, which if oftentimes unfeasible given available computational power(see Q4)(Sung,1998). Thus, including solely 7 features and 1,000 instances upon generation and CTGAN training made synthetic data inadequate to represent all underlying structures.

The abovementioned issues signal synthetic data's irrelevance to Airbnb for predicting optimal pricings. Instead, Airbnb could run predictive models on original data and compute average pricings given listings' parameters, displaying this information to hosts.Alternatively, ML algorithms,including federated learning could preserve hosts' privacy, while allowing larger dataset training(Thapa,Chamikara&Camtepe, 2021). Additionally, it would save costs and attract more commission revenues from augmented bookings.

Nevertheless, synthetic data could be useful for many other business applications, including learning analytics(Berg et al.,2016).

5 References

Assefa, S.A., Dervovic, D. and Mahfouz, M. (2020). "Generating synthetic data in Finance," Proceedings of the First ACM International Conference on AI in Finance [Preprint].

Bellman, R. (1966). "Dynamic Programming," Science, 153(3731), pp. 34–37. Available at: https://doi.org/10.1126/science.153.3731.34.

Berg, A.M., Mol, S.T. and Kismihók, G., Sclater, N. (2016). "The role of a reference synthetic data generator within the field of learning analytics.," Journal of Learning Analytics, 3(1).

Chale, M. and Bastian, N.D. (2021). "Challenges and opportunities for generative methods in the cyber domain," 2021 Winter Simulation Conference (WSC) [Preprint].

Curry, D. (2023). Airbnb revenue and Usage Statistics (2023), Airbnb Revenue and Usage Statistics (2023). Business of Apps. [online]. Available at: https://www.businessofapps.com/data/airbnb-statistics/ (Accessed: March 1, 2023).

El Emam, K.., Hoptroff, R.. and Mosquera , L. (2020). Practical synthetic data generation balancing privacy and the broad availability of data;balancing privacy and the broad availability of data. 1st edn. S.l., Sebastopol: O'REILLY MEDIA, INCORPORA.

Gartner. (2022). Is synthetic data the future of ai? Gartner. [online]. Available at: https://www.gartner.com/en/newsroom/press-releases/2022-06-22-is-synthetic-data-the-future-of-ai (Accessed: February 28, 2023).

Goel, A.S. (2022).How does airbnb make money: Business model, How does Airbnb make money | Business Model. The Strategy Story. [online]. Available at: https://thestrategystory.com/2022/03/29/how-does-airbnb-make-money-business-model/ (Accessed: March 9, 2023).

Inside Airbnb (2022). Get the Data, Inside Airbnb. [online]. Available at: http://insideairbnb.com/get-the-data/ (Accessed: February 15, 2022).

Krasheninnikova, E., García, J., Maestre, R. Fernández, F. (2019). "Reinforcement learning for pricing strategy optimization in the insurance industry," Engineering Applications of Artificial Intelligence, 80(2), pp. 8–19. Available at: https://doi.org/10.1016/j.engappai.2019.01.010.

Moreno-Torres, J.G., Raeder, T., Alaiz-Rodríguez, R., Chawla, N. and Herrera F.(2012) "A unifying view on dataset shift in classification," Pattern Recognition, 45(1), pp. 521–530. Available at: https://doi.org/10.1016/j.patcog.2011.06.019.

Pan, Q. and Wang, W. (2021). "Costly price adjustment and automated pricing: The case of airbnb," SSRN Electronic Journal [Preprint]. Available at: https://doi.org/10.2139/ssrn.4077985.

Platzer, M. and Reutterer, T. (2021). "Holdout-based empirical assessment of mixed-type synthetic data," Frontiers in Big Data, 4(2), pp. 7–18.

Reynolds, J., Ahmad, M.W., Rezgui, Y. and Hippolyte, J. (2019). "Operational Supply and demand optimisation of a multi-vector district energy system using artificial neural networks and a genetic algorithm," Applied Energy, 235, pp. 699–713.

Stadler, T., Oprisanu, B. and Troncoso, C. (2022) Synthetic data – Anonymisation Groundhog day - usenix, Synthetic Data – Anonymisation Groundhog Day. [online]. Available at: https://www.usenix.org/system/files/sec22summer_stadler.pdf (Accessed: February 28, 2023).

Sung, A.H. (1998). "Ranking importance of input parameters of neural networks," Expert Systems with Applications, 15(3-4), pp. 405–411. Available at: https://doi.org/10.1016/s0957-4174(98)00041-4.

Thapa, C., Chamikara, M.A. and Camtepe, S.A. (2021) "Advancements of federated learning towards Privacy Preservation: From federated learning to split learning," Federated Learning Systems, pp. 79–109.

Vokhmina, Y.V., Eskov, V.M., Gavrilenko, T.V. and Filatova, E. (2015). "Measuring order parameters based on Neural Network Technologies," Measurement Techniques, 58(4), pp. 462–466.