CCT College Dublin

Assessment Cover Page

Module Title:	Data Programming for Data Analytics, Statistics for Data Analysis, Data Preparation & Visulization, Machine Learning for Data Analytics
Assessment Title:	New House Prices in Ireland
Lecturer Name:	David McQuaid, John O'Sullivan, Sam Weiss, Dr. Muhammad Iqbal
Student Full Name:	Kagan Timur
Student Number:	2023044
Assessment Due Date:	14/04/2023
Date of Submission:	14/04/2023

Declaration

By submitting this assessment, I confirm that I have read the CCT policy on Academic Misconduct and understand the implications of submitting work that is not my own or does not appropriately reference material taken from a third party or other source. I declare it to be my own work and that all material from third parties has been appropriately referenced. I further confirm that this work has not previously been submitted for assessment by myself or someone else in CCT College Dublin or any other higher education institution.

New House Prices Research in Dublin

CONTENTS

Abstract

The main objective of this assignment is to compare the multiple linear regression model and decision tree regressor for predicting Dublin housing prices. The dataset includes national house prices in Ireland, and major cities like Dublin, Cork, Galway, Limerick, Waterford, and other areas in the country. I collected the data from government resources as a part of my analysis (Department of Housing, Local Government, and Heritage, 2016). It also includes a dataset which has the total population, and inflation(%) on yearly basis from the world data bank for my analysis (World Data Bank, 2015). Additionally, the dataset includes the highest mortgage rate on a yearly basis (Central Statistics Office, 2022). The process includes visualisation and analysis of real estate data using Python 3.9.13 in Jupyter Notebook, to predict yearly average sales, total population, inflation, and highest mortgage rate.

In order to select an appropriate prediction method, various regression models are being explored and compared. The methods under consideration include multiple linear regression, lasso and ridge regularization, as well as decision tree regressor with grid search hyperparameter tuning technique.

To evaluate and compare the performance of these methods, scatterplot and R^2 training and testing results are being utilized. These measures are being employed to assess how well each model fits the data and how accurately it can predict the outcome variable.

The scatterplot and R^2 results are being analysed to determine which method yields the most satisfactory fit for the given data set. This analytical process aids in selecting the best-suited method for predicting the outcome variable and drawing sound conclusions about the relationship between the variables under consideration.

The dataset in question ranges from 1997 to 2015, and the prediction model created is used from 1997 to 2014 to produce a prediction for 2015. By contrasting the anticipated value with the actual value, this strategy aims to assess the model's predictive power.

To choose the best model for the task at hand, other machine learning models are also being compared. This entails evaluating the effectiveness of various models using the proper evaluation metrics and procedures. The most crucial objective is to identify the model that generates the most reliable and accurate predictions for the provided data set after comparing the result with predictions and real life numbers.

1. DATA PREPARATION

```
import pandas as pd
✓ 3.4s
```

First import the pandas library as pd because it provides a wide range of functions and methods for data manipulation, plotting, merging, and analysing. *"The pandas library, under development since 2008, is intended to close the gap in the richness of available data analysis tools between Python, a general purpose systems and scientific computing language, and the numerous domain specific statistical computing platforms and database languages* (McKinney, 2011)*"*.

```
# Read the file into a DataFrame: df
data = pd.read_csv('https://opendata.housing.gov.ie/dataset/
✓ 0.5s
```

Python reads a CSV file from a remote server containing annual new house price data from the Central Statistics Office of Ireland, loads the file using the read_csv() method, and assigns the variable data before creating a data frame.

```
data.shape  # Check the shape of the DataFrame
✓ 0.0s
(78, 8)
```

The data frame represents 78 rows and 8 columns.

```
data.head() # first 5 rows
✓ 0.0s
```

	YEAR	Average New House Price	Unnamed: 2	Unnamed: 3	Unnamed: 4	Unnamed: 5	Unnamed: 6	Unnamed: 7
0	NaN	National	Dublin	Cork	Galway	Limerick	Waterford	Other Areas
1	1997.0	102,037	123,231	96,504	111,108	91,236	92,372	94,642
2	1998.0	124,368	159,558	112,166	118,917	104,327	108,789	116,639
3	1999.0	147,043	191,942	140,797	137,510	122,146	132,510	137,031
4	2000.0	166,155	216,433	164,535	160,972	146,838	145,087	154,141

The .head(n) function allows to retrieval of the first rows of a data frame, where n is an optional parameter that defaults to 5 if not specified.

```
data.tail() # last 5 rows
```
✓ 0.1s

	YEAR	Average New House Price	Unnamed: 2	Unnamed: 3	Unnamed: 4	Unnamed: 5	Unnamed: 6	Unnamed: 7
73	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
74	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
75	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
76	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
77	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

The .tail(n) function allows to retrieval of the last rows of a data frame.

```
data.sample(10) # 10 random rows
```
✓ 0.0s

	YEAR	Average New House Price	Unnamed: 2	Unnamed: 3	Unnamed: 4	Unnamed: 5	Unnamed: 6	Unnamed: 7
42	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
75	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
40	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
16	2012.0	223,580	292,004	235,446	221,159	213,458	179,716	210,180
31	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
7	2003.0	220,573	302,270	210,733	222,578	193,854	193,642	203,421
73	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
9	2005.0	272,034	386,089	264,719	274,745	226,773	245,315	255,730
25	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
69	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

It seems that some NaN values are present based on the insights drawn from the dataset. Furthermore, it would seem that the "Unnamed" columns' first row should contain the names of the other columns.

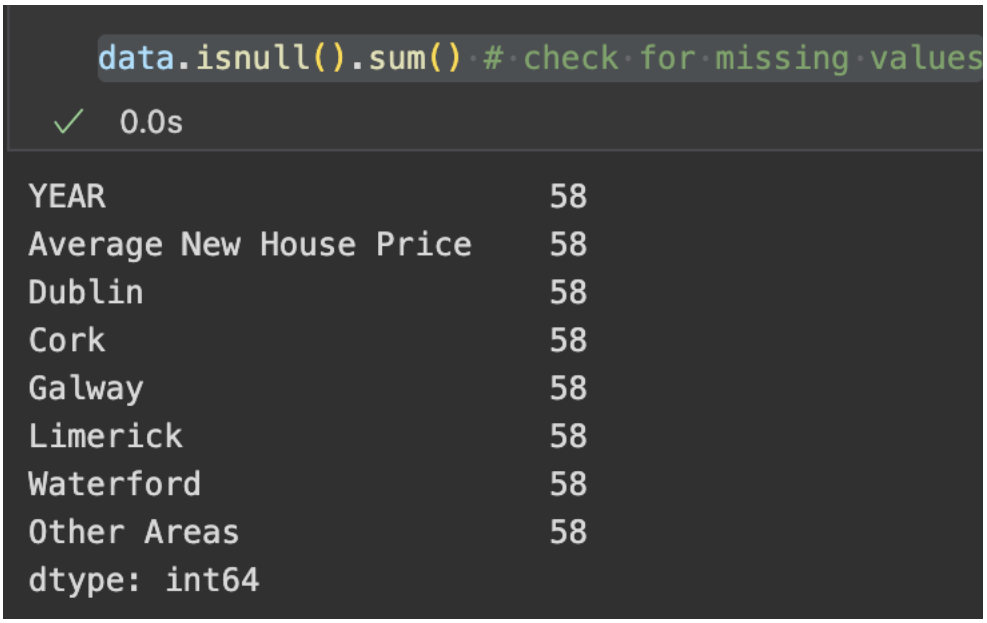
```
new_names = dict(zip(data.columns[2:], data.iloc[0, 2:]))

data = data.rename(columns=new_names)

data.head(1)
```
✓ 0.0s

	YEAR	Average New House Price	Dublin	Cork	Galway	Limerick	Waterford	Other Areas
0	NaN	National	Dublin	Cork	Galway	Limerick	Waterford	Other Areas

These lines of code create a new dictionary by combining the first row's values from the same column index with the columns' names starting at the second column index. This dictionary represents a mapping between the original column names and the desired column names.

The pandas .rename() function is then applied to the data frame with this dictionary as an argument to rename the columns. The first row of the data frame is assumed to contain the desired column names, but they were added as regular data rows. Therefore, the renaming operation corrects this by assigning the correct column names.

The first row, which carries the column names as values rather than column headings, is removed in this line of code by using the "drop" method on the "data" data frame. This is done to make sure the first row won't interfere with data analysis or modification. The variable "data" is given the resulting data frame as a new value.



The code data.isnull().sum() is used to calculate the number of missing values (NaN) in each column of the data frame. It first applies the isnull() method to the data frame, which returns a boolean data frame indicating whether each element of the data frame is missing or not. Then, the sum() method is applied to the boolean data frame, which calculates the number of True values (missing values) in each column.

The method of .dropna() drops rows with missing values.

```
data.info() # check the data types of the columns
✓ 0.0s
<class 'pandas.core.frame.DataFrame'>
Int64Index: 19 entries, 1 to 19
Data columns (total 8 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   YEAR                    19 non-null     float64
 1   Average New House Price  19 non-null     object
 2   Dublin                  19 non-null     object
 3   Cork                    19 non-null     object
 4   Galway                  19 non-null     object
 5   Limerick                19 non-null     object
 6   Waterford               19 non-null     object
 7   Other Areas             19 non-null     object
dtypes: float64(1), object(7)
memory usage: 1.3+ KB
```

After dropping the missing values, the data frame has 19 rows and 8 columns with no non-null values. However, due to the existing types of data, more processing is necessary before the data can be used for analysis and mathematical operations. For more accurate analysis, the remaining columns should be changed to floats and the "Year" column should be made into an integer.

```
data["YEAR"] = data['YEAR'].astype(int)
✓ 0.0s
```

The "astype" method from is used in this line of code to change the "YEAR" column in the "data" data frame into an integer data type. To ensure that the "YEAR" column values are displayed as integers, which is more appropriate for this sort of data, this conversion is carried out. The revised "data" data frame now has an integer-valued "YEAR" column.

```
# Convert columns to float type and replace commas with periods
cols_to_convert = ["Average New House Price", "Dublin ", "Cork", "Galway", "Limerick", "Waterford", "Other Areas"]
for col in cols_to_convert:
    data[col] = data[col].str.replace(",", ".").astype(float)

# Rename Dublin column to remove trailing space
data.rename(columns={'Dublin ': 'Dublin'}, inplace=True)
```

In order to enable calculations on the data, a “for” loop is created in this code block to change the specified columns' data type to "float" and replace the commas with "periods". Each selected column's commas are replaced with periods using the “str.replace()“ function, and each column's data type is changed to float using the “astype()” method. Using the “rename()” method and a dictionary, the column name "Dublin " is changed to "Dublin" because it has a trailing space. The generated data frame can now be applied to additional modelling and analysis.

```
data.info() # check the data types of the columns
✓ 0.1s
<class 'pandas.core.frame.DataFrame'>
Int64Index: 19 entries, 1 to 19
Data columns (total 8 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   YEAR                    19 non-null     int64
 1   Average New House Price  19 non-null     float64
 2   Dublin                  19 non-null     float64
 3   Cork                    19 non-null     float64
 4   Galway                  19 non-null     float64
 5   Limerick                19 non-null     float64
 6   Waterford               19 non-null     float64
 7   Other Areas             19 non-null     float64
dtypes: float64(7), int64(1)
memory usage: 1.3 KB
```

The data frame is examined using the.info() method once again to determine the data types of its columns.

1.2. Additional Dataset

```
data1 = pd.read_csv('IRL.csv')
✓ 0.0s
```

The 'IRL.csv' parameter is passed to the pd.read_csv() function, which reads the file and turns it into a data frame. Then, a data frame is allocated to the variable "data1" as a consequence.
“A simple multiplier is used to convert the national currencies of EMU members to euros. The following irrevocable euro conversion rate was adopted by the EU Council on January 1, 1999: 1 euro = 0.787564 Irish pound. Please note that historical data before 1999 are not actual euros and are not comparable or suitable for aggregation across countries. (World Data Bank, 2016)”

```
data1.head() # first 5 rows
```
✓ 0.4s

	YEAR	1997	1998	1999	2000	
0	Population, total	3.674171e+06	3.712696e+06	3.754786e+06	3.805174e+06	3.866
1	Interest rate spread (lending rate minus depos...	6.900000e+00	5.850000e+00	5.600000e+00	6.090000e+00	6.900
2	Inflation, consumer prices (annual %)	1.525605e+00	2.415518e+00	1.631924e+00	5.590717e+00	4.872

+ Code + Markdown

Transposing the dataset is a useful strategy to improve readability.

```
data1.head().T # check the rows of the data frame
```
✓ 0.0s

Outputs are collapsed ···

✓ `data1 = data1.transpose()` ···

	0	1	2
YEAR	Population, total	Interest rate spread (lending rate minus depos...	Inflation, consumer prices (annual %)
1997	3674171.0	6.9	1.525605
1998	3712696.0	5.85	2.415518
1999	3754786.0	5.6	1.631924
2000	3805174.0	6.09	5.590717

Following a review of the dataset's structure, the "data1" data frame will be overwritten using the transpose method to alter the dataset's shape. To accomplish this, use the assignment operator to assign the transposed data frame to the original variable called "data1". The method reverses all the columns and rows in the final data frame, making it easier to read for data analysis.

```
new_names = dict(zip(data1.columns[:], data1.iloc[0,:]))

data1 = data1.rename(columns=new_names)

data1.head()
```
✓ 0.0s Pyth

	Population, total	Interest rate spread (lending rate minus deposit rate, %)	Inflation, consumer prices (annual %)
YEAR	Population, total	Interest rate spread (lending rate minus depos...	Inflation, consumer prices (annual %)
1997	3674171.0	6.9	1.525605
1998	3712696.0	5.85	2.415518
1999	3754786.0	5.6	1.631924
2000	3805174.0	6.09	5.590717

The renaming of the columns in the data frame to more relevant names is the aim of the data preparation process, which includes this code block as an essential part.
In addition, it gives the columns of the data frame new names using a dictionary built based on the data frame's first row, making them more understandable and practical for use in future analyses.

```
df = data1.iloc[1:]  # drop the first row
✓ 0.0s

df # check the updated DataFrame
✓ 0.0s
```

	Population, total	Interest rate spread (lendi
1997	3674171.0	

The first row of the data frame is dropped in these two lines of code to remove inaccurate column names, and the modified data frame is checked to ensure the removal was successful.

```
df = df.rename_axis(None, axis=1) # rename the index column to YEAR
df.head() # first 5 rows
✓ 0.0s                                                    Python
```

	YEAR	Population, total	Interest rate spread (lending rate minus deposit rate, %)	Inflation, consumer prices (annual %)
0	1997	3674171.0	6.9	1.525605
1	1998	3712696.0	5.85	2.415518
2	1999	3754786.0	5.6	1.631924
3	2000	3805174.0	6.09	5.590717
4	2001	3866243.0	6.9	4.872905

The index column of the data frame is given the new name None in the first line of code, thus erasing the name of the index column. The second line of code uses the head method to display the first five rows to check the changed data frame.

```
df = df.reset_index().rename(columns={'index': 'YEAR'})

✓ 0.0s
```

In order to improve the readability and interpretation of the data, this code block renames the index column of a data frame to "YEAR" as part of the data preparation process.

```
df.info() # check the data types of the columns because we need to convert the YEAR column to integer typ
✓ 0.0s                                                    Python
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19 entries, 0 to 18
Data columns (total 4 columns):
 #   Column                                                Non-Null Count  Dtype
---  ------                                                --------------  -----
 0   YEAR                                                  19 non-null     object
 1   Population, total                                     19 non-null     object
 2   Interest rate spread (lending rate minus deposit rate, %)  19 non-null  object
 3   Inflation, consumer prices (annual %)                 19 non-null     object
dtypes: object(4)
memory usage: 736.0+ bytes
```

The column data types are currently objects, which makes them unsuitable for additional research. The other columns must be converted to floats because of their numerical values, and the "YEAR" column requires to be changed to an integer.

1.3. Merging the Disparate Datasets

```
df["YEAR"] = df['YEAR'].astype(int)
```
✓ 0.0s

```
merged_data = pd.merge(data, df, on='YEAR', how='left')
```
✓ 0.0s

```
merged_data.head()
```
✓ 0.0s Python

	YEAR	Average New House Price	Dublin	Cork	Galway	Limerick	Waterford	Other Areas	Population, total	Interest rate spread (lending rate minus deposit rate, %)	Inflation, consumer prices (annual %)
0	1997	102.037	123.231	96.504	111.108	91.236	92.372	94.642	3674171.0	6.9	1.525605
1	1998	124.368	159.558	112.166	118.917	104.327	108.789	116.639	3712696.0	5.85	2.415518
2	1999	147.043	191.942	140.797	137.510	122.146	132.510	137.031	3754786.0	5.6	1.631924
3	2000	166.155	216.433	164.535	160.972	146.838	145.087	154.141	3805174.0	6.09	5.590717
4	2001	181.146	252.192	175.372	171.578	154.515	157.767	167.493	3866243.0	6.9	4.872905

The first code modifies the "YEAR" column's type to enable it to combine with previously prepared data and provide more information to the analysis.

```
merged_data = merged_data.rename(columns={'Population, total': 'Total Population',
                                          'Interest rate spread (lending rate minus deposit rate, %)': 'H
                                          'Inflation, consumer prices (annual %)': 'Inflation %',
                                          'Average New House Price': 'Avg House Price in IE',
                                          'YEAR': 'Year',
                                          })
```
✓ 0.0s Python

Let's see our new data frame

```
merged_data.head()
```
✓ 0.0s Python

	Year	Avg House Price in IE	Dublin	Cork	Galway	Limerick	Waterford	Other Areas	Total Population	Highest Mortgage Rate %	Inflation %
0	1997	102.037	123.231	96.504	111.108	91.236	92.372	94.642	3674171.0	6.9	1.525605

The first block of code aims to modify the column names. With the use of a dictionary that converts the old column names to new column names, the code renames five columns in the merged_data data frame. The revised column titles, which are shown.

```
merged_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 19 entries, 0 to 18
Data columns (total 11 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   Year                    19 non-null     int64  
 1   Avg House Price in IE   19 non-null     float64
 2   Dublin                  19 non-null     float64
 3   Cork                    19 non-null     float64
 4   Galway                  19 non-null     float64
 5   Limerick                19 non-null     float64
 6   Waterford               19 non-null     float64
 7   Other Areas             19 non-null     float64
 8   Total Population        19 non-null     object 
 9   Highest Mortgage Rate %  19 non-null     object 
 10  Inflation %             19 non-null     object 
dtypes: float64(7), int64(1), object(3)
memory usage: 1.8+ KB
```

The data types of the variables in the merged data frame are checked using the "merged_data.info()" programmatic function. Make sure that all the variables are in a format that will allow for future analysis. Before some of the variables may be utilized in mathematical operations or modelling, they might need to be transformed into a different data type.

2. Insight

2.1. Overview of the Merged Dataset

After preparing the data frame for future analysis, let us take a quick look at it. The dataset comprises of 19 rows and 11 columns. Each row represents a year from 1997 to 2015, and each column represents a variable.

The first variable, "Year," denotes the year of the data point and is recorded on a nominal scale. The columns "Avg House Price in IE", "Dublin", "Cork", "Galway", "Limerick", and "Waterford" represent the current average house prices in their respective named cities in Ireland and are recorded on a ratio scale, signifying a true zero point.

The column "Other Areas" represents the average house price in other areas of Ireland, and is also recorded on a ratio scale. The column "Total Population" provides the current total population of Ireland and is recorded on a ratio scale as well.

The "Inflation %" column provides information on the overall price trend during the time period covered by the dataset and can be used to adjust nominal prices on a ratio scale for

inflation. The column "Highest Mortgage Rate %" representing the highest mortgage rate in Ireland, is recorded on an interval scale. The dataset provides facts on average property prices across Ireland's regions along with other economic factors.

```
# I am keeping the data for myself, so I will save it to a csv file on my desktop
# I will use the os module to get the path to my desktop
import os
file_name = 'merged_data.csv'
desktop_directory = os.path.join(os.path.expanduser('~'), 'Desktop')
output_file_path = os.path.join(desktop_directory, file_name)
merged_data.to_csv(output_file_path, index=False)
```

The code defines a file directory on the desktop for the combined data to be stored as a CSV file using the OS library. Then, it uses the output_file_path parameter passed to the to_csv() method on the merged_data data frame to store the data frame as a CSV file in the designated location on the desktop.

2.2. Importing Necessary Libraries

```
import warnings # import the warnings module
import seaborn as sns # import the seaborn module
import matplotlib.pyplot as plt # import the matplotlib module
warnings.filterwarnings("ignore") # ignore warnings
pd.set_option("display.max_rows",None) # display all rows
import matplotlib  # import the matplotlib module
import scipy.stats as stats # import the stats module from the scipy library
from scipy.stats import norm # import the norm module from the stats module of the scipy library
```

Seaborn is a data visualization tool that integrates with Pandas data structures, Matplotlib offers a variety of visualization options with an approachable syntax, and SciPy adds additional functionality for optimization, integration, and signal processing. NumPy is a core library used for multi-dimensional arrays and mathematical operations. SciPy's stats module includes the norm module for working with normal distributions as well as statistical functions, distributions, and tests. To make utilizing these libraries in Python code simpler, aliases are frequently used to import them.

```
# Let's describe the data to see the summary statistics of the data set
merged_data.describe()
```

Python

	Year	Avg House Price in IE	Dublin	Cork	Galway	Limerick	Waterford	Other Areas	Total Population	Highest Mortgage Rate %	Inflation %
count	19.000000	19.000000	19.000000	19.000000	19.000000	19.000000	19.000000	19.000000	1.900000e+01	19.000000	19.000000
mean	2006.000000	224.957632	309.492158	223.630263	215.932211	202.387632	195.253789	208.699947	4.231182e+06	4.901053	2.153413
std	5.627314	60.311751	97.607447	63.859395	56.306805	58.336559	57.672653	56.225711	3.658243e+05	1.040464	2.455865
min	1997.000000	102.037000	123.231000	96.504000	111.108000	91.236000	92.372000	94.642000	3.674171e+06	3.490000	-4.478103
25%	2001.500000	187.990500	255.786500	180.128000	179.386000	162.046000	155.909000	173.740500	3.899095e+06	4.180000	1.017160
50%	2006.000000	233.173000	302.270000	241.127000	221.159000	212.882000	193.642000	215.439000	4.273591e+06	4.420000	2.415518
75%	2010.500000	259.680500	362.367500	256.372500	241.690000	229.810000	226.176500	243.755500	4.570120e+06	5.725000	3.995972
max	2015.000000	320.788000	484.926000	326.765000	302.208000	290.670000	295.397000	299.884000	4.701957e+06	6.900000	5.590717

Before visualizing the dataset, it's essential to analyze central tendency and dispersion measurements, such as mean, standard deviation, maximum, and minimum values. The .describe() method calculates these statistics for all 19 variables. For instance, the average cost of a house in Ireland is 224.96 euros, while that in Dublin is 309.49 euros with a standard deviation of 97.61. The average inflation rate is 2.15 per cent, and the maximum mortgage rate is 4.90 per cent, with a standard deviation of 2.46 and 1.04, respectively. Negative values indicate deflation and positive values suggest inflation. The average house

price in Ireland may increase to a maximum of 320.79, and it may reach a high of 484.93 in Dublin.

```
# calculate probability of average house price in Dublin being less than 300,000 euros
prob_less_than_300k = norm.cdf(300, loc=mean, scale=std)
print(f"Probability of average house price in Dublin being less than 300,000 euros: {prob_less_than_300k:.2f}")

✓ 0.0s
Probability of average house price in Dublin being less than 300,000 euros: 0.46
```

The cumulative distribution function (cdf) of a normal distribution is used for calculating the probability of the average property price in Dublin will be less than 300,000 euros.

As a result, Dublin's average property price is likely to be less than 300,000 euros in 46% of cases. This shows that the average property price in Dublin will exceed 300,000 euros 54% of the time.

```
# calculate 95% confidence interval for average house price in Dublin
conf_int = norm.interval(0.95, loc=mean, scale=std)
print(f"95% confidence interval for average house price in Dublin: {conf_int}")

✓ 0.0s
95% confidence interval for average house price in Dublin: (118.185078128459, 500.79923766101473)
```

The norm function is used in the code to determine the Dublin's average home price's 95% confidence interval. The function returns a tuple containing the lower and upper bounds of the confidence interval and accepts as parameters the level of certainty, the average of the dataset, and the standard deviation. According to the tuple that results, there is a 95% probability that Dublin's average house price will be between 118,185 and 500,799 euros. Making choices regarding investments can be aided by this information. If an investor's target price is significantly below the lower bound of the confidence interval, they may decide against buying the property. Similarly, if the top limit of the confidence interval is far higher than their desired price, they can decide against investing in the property.

```
norm_prop = norm(loc=mean, scale=std)
norm_prop.sf(500.000) # calculate the survival function (1 - cdf) at x
print("Probability of house price in Dublin being greater than 500,000 : ", norm_prop.sf(500.000))
# The result of this code indicates that the probability of house price in Dublin being greater than 500,000

✓ 0.0s
Probability of house price in Dublin being greater than 500,000 :  0.025482420327711367
```

The code generates the mean and standard deviation for the normal distribution object norm_prop using the norm function from the scipy.stats package. The probability that the house will cost more than 500,000 is obtained by using the .sf() method on this object with a value of 500,000 as its input and computing the survival function (1 - cdf) at x.

Probability of house price in Dublin being greater than 500,000 euro is 0.025, which indicates a 2.5%.

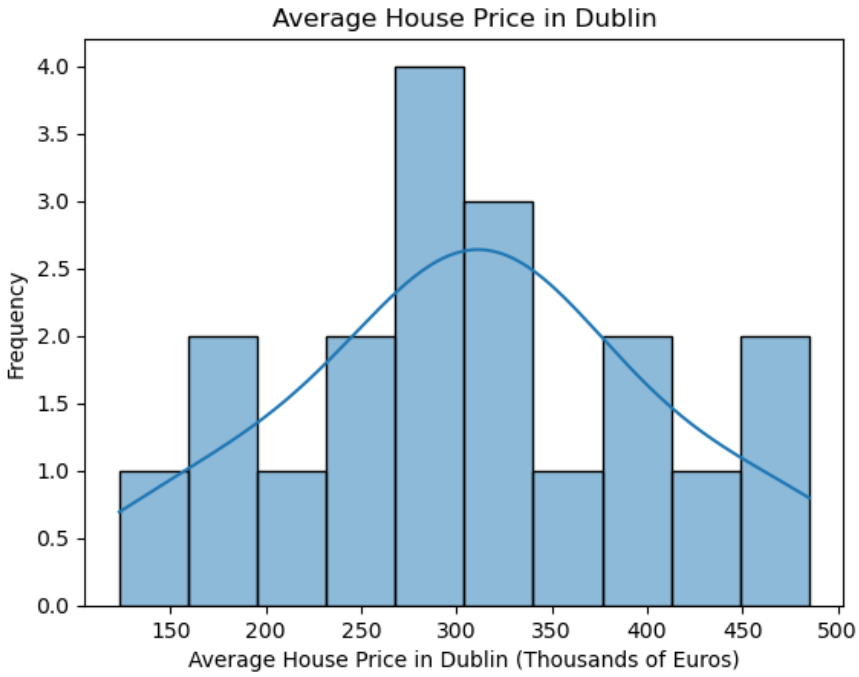
Investors and others with an interest in the Dublin house prices might find it helpful to know that there is a 3% chance that a house would cost more than 500,000 euros. Thinking about

the relatively low possibility that a residence with a 600,000 selling price will bring in more than 500,000 euros if a buyer is thinking about buying a new house. This can affect their choice to buy a house or engage in price negotiation. In contrast, if someone is looking to buy a house in Dublin, they might use this information to change their spending plan and look for properties in a more reasonable price range.

```
# plot histogram of average house prices in Dublin with seaborn and matplotlib
dublin=merged_data["Dublin"] # create a variable for the average house prices in
sns.histplot(dublin, bins=10, kde=True) # plot histogram with seaborn and set the
plt.title('Average House Price in Dublin') # I will name the title of the histogr
plt.xlabel('Average House Price in Dublin (Thousands of Euros)') # I will change
plt.ylabel('Frequency') # I will change the y-axis label to Frequency
plt.show() # display the histogram

✓ 0.6s
```

This code creates a histogram plot showing the Dublin average house price distribution.. The merged_data data frame's average house prices for Dublin are first stored in a variable called "dublin" that is created by the code. The histogram is then plotted using the Seaborn function "histplot," with the number of bins set to 10 and the "kde" option set to True, which also displays the kernel density estimation plot. The plot is then given a title, "Average House Price in Dublin," the x-axis label is changed to "Average House Price in Dublin (Thousands of Euros)," and the y-axis label is changed to "Frequency."
Finally, the plot is displayed using the 'show' method. This visualization can help us understand the distribution and frequency of average house prices in Dublin.

2.3. Normal Distribution of Dublin House Prices



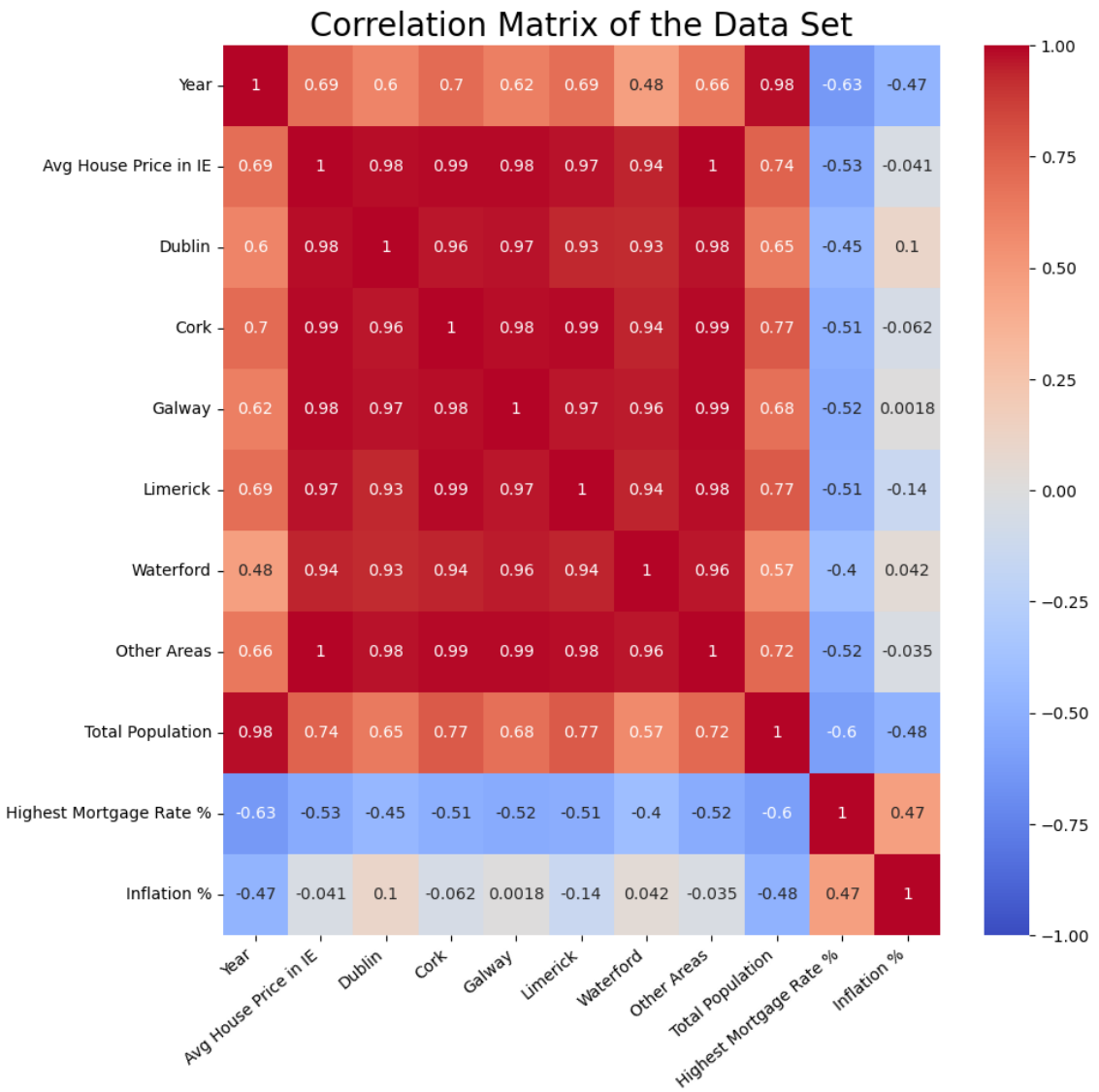
The histogram and kernel density indicate that the data is symmetrical about the mean, as seen in the distribution, with the majority of the data points clustering around the mean and

fewer points further away from the mean in either direction. A normal distribution suggests that most of the average house prices in Dublin revolve around a particular value, with few prices much higher or lower than this value.

For instance, it can give information on how prices are distributed in the housing market and assist individuals in making decisions that are better informed based on the probability of a particular price range will be seen.

3. Visualization

3.1. Heatmap Correlation Matrix



The heatmap examine more about the connections between variables and Dublin house prices. It is easier to determine any strong positive or negative correlations between the variables by using a heat map. In this situation, a heat map aids in clarifying the connections between average house prices in various cities and other factors such as total population,

highest mortgage rate, and inflation rate. This visualization might shed light on how these variables affect the value of houses across various regions.

Investigating the relationships between Dublin housing prices, total population, and the highest mortgage rate. It shows that the population and the mortgage rate have a positive correlation with Dublin property prices. On the other side, there was a negative correlation between the two. This suggests that while Dublin house prices tend to rise along with population growth and mortgage rates, they tend to fall along with inflation rates.

We may look at their correlation to learn more about how mortgage rates and inflation are related. The data indicates a negative association between the mortgage rate and inflation, which means that as inflation rises, mortgage rates tend to fall.
By altering the money supply and interest rates, central banks try to keep the economy stable. In order to reduce inflation, central banks usually raise interest rates. Mortgage rates typically rise due to rising interest rates. As a consequence, central banks typically drop interest rates during periods of low inflation, which lowers mortgage costs. Therefore, the dataset's negative association between mortgage rates and inflation may demonstrate how changes in monetary policy by the government have affected the housing market over time. *"Monetary policy was a string. You could pull on it to stop inflation but you could not push on it to halt recession. You could lead a horse to water but you could not make him drink.* (FRIEDMAN, 1995)*"*

Lastly, to investigate the relationship between the average house price in Ireland and inflation, population, and mortgage rate, we can examine their correlation. The dataset shows that the average house price in Ireland is positively correlated with population and mortgage rate, but negatively correlated with inflation as same as Dublin.

3.2. Line Charts

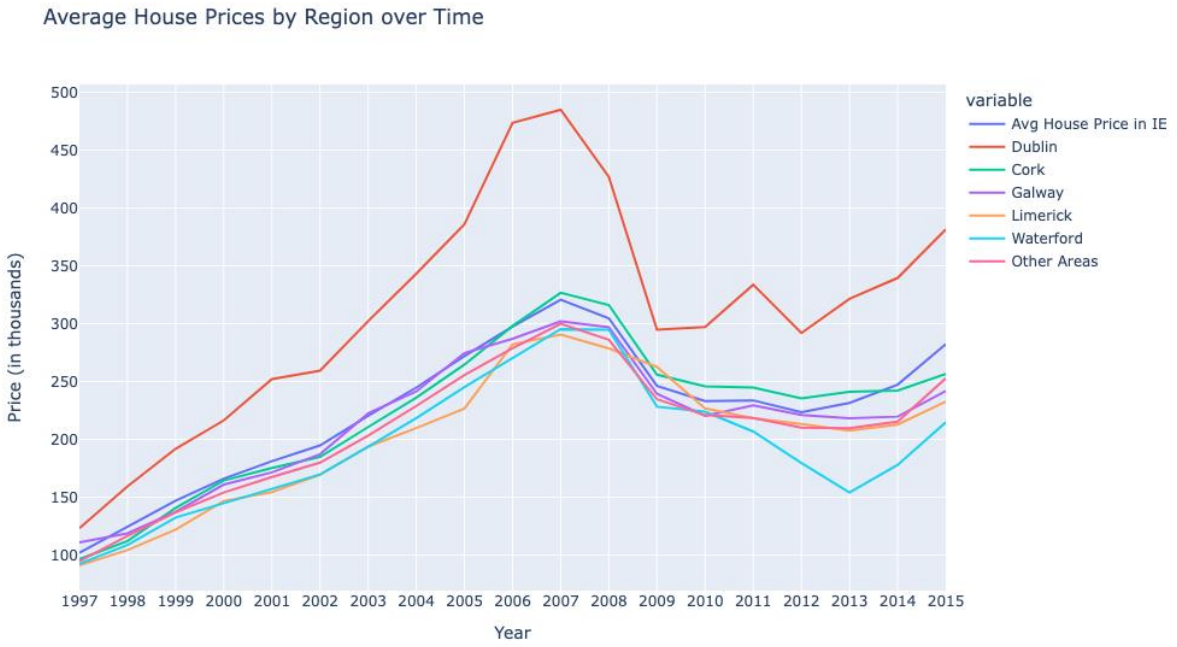
3.2.1. Average House Prices by Region Over Time

```
# Import the plotly.express module as px for interactive plots
import plotly.express as px

fig = px.line(merged_data, x='Year', y=['Avg House Price in IE', 'Dublin', 'Cork', 'Galway', 'Limerick', 'Waterford', 'Other Areas'],
              # x and y arguments are used to specify the columns to be used for the x and y axes
              title='Average House Prices by Region over Time') # I will set the title of the figure
fig.update_layout(xaxis_title='Year', yaxis_title='Price (in thousands)') # I will set the title of the x and y axis
# I will change the size of the figure
fig.update_layout(width=1000, height=600) # I will set the width and height of the figure to 1000 and 600 respectively
# I will change the x axis step to 1 year for better readability
fig.update_xaxes(dtick=1)


fig.show()
```

Using the Plotly Express module (GAUR, 2022), this code creates an interactive line chart that displays the property values over time in Ireland.

The 'fig' variable is assigned to the figure object that will be created by the plot. The "px.line" function's "x" and "y" arguments are used to indicate the columns to be used for the x and y axes, where the y-axis represents the average house values in Ireland and its various areas and the x-axis the years.

Using the 'title' argument, the figure's title is modified to "Average House Prices by Region over Time." The 'xaxis_title' and 'yaxis_title' variables, respectively, are used to set the x-axis and y-axis labels.



The chart demonstrates an overall increase trend in real estate prices from 1997 to 20015 across all regions. We can see that the cost of housing is generally higher in Dublin than the rest of Ireland. Dublin's housing market experienced a rapid rise in prices from 2001 to 2006, followed by a more gradual rise up to 2007. The line then exhibits a sudden decline until 2011, a little increase until 2012, and then a steady ascent once more until 2015. Similar trends are visible in Ireland's other areas, however with less extreme spikes and decrease.

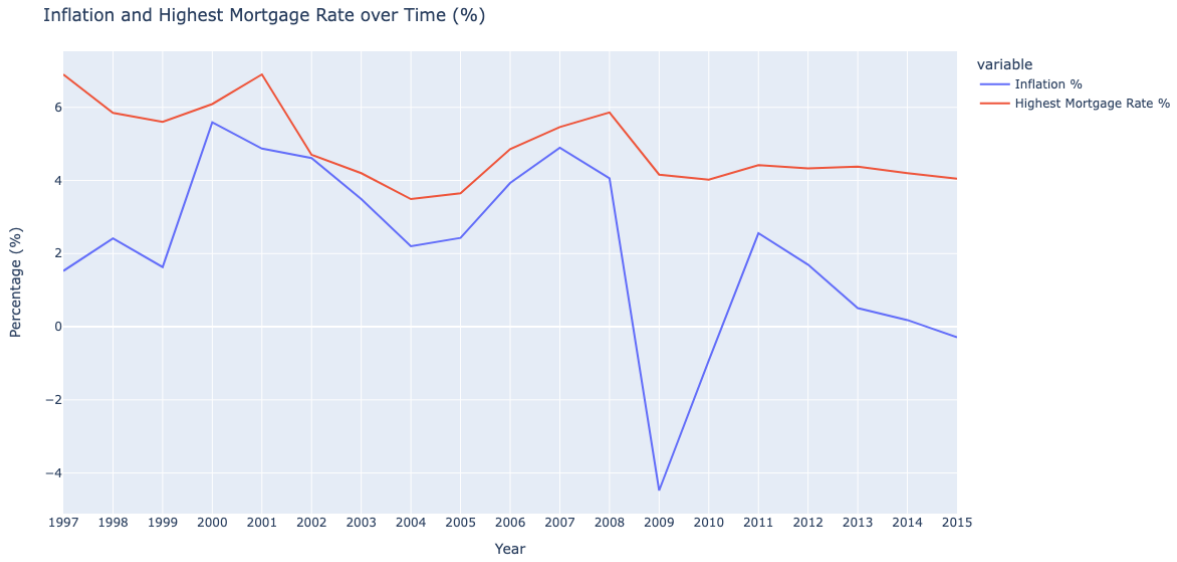
All regions saw a key turning point in 2007, when the average house price started to decline. This trend lasted until 2012, when the average house price was at its lowest. Limerick, Waterford, and other areas saw relatively lesser falls in housing prices, while Dublin, Cork, and Galway saw the biggest drops.

Except for Waterford and Other Areas, where prices continued to drop, all areas' average house prices showed signs of recovery from 2012 to 2015.

3.2.2.Inflation and Highest Mortgage Rate Over Time

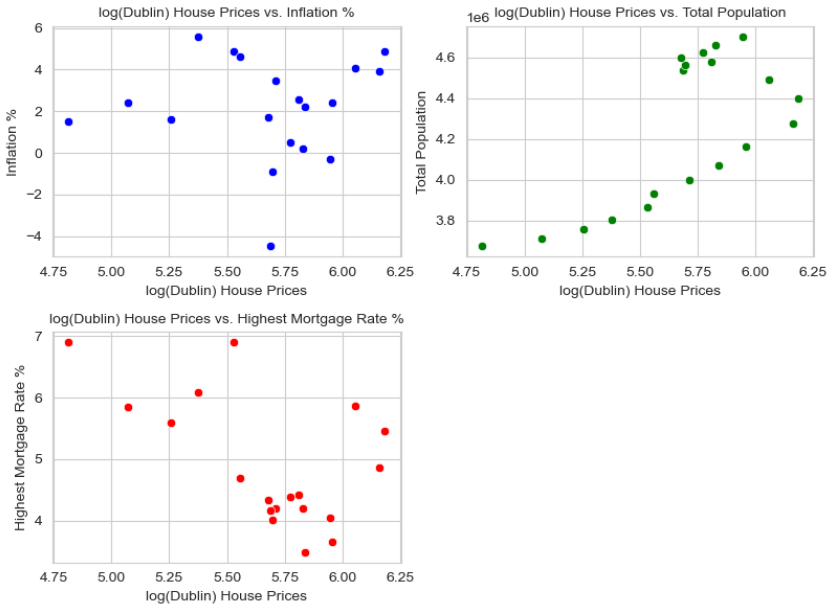
```
import plotly.express as px

fig = px.line(merged_data, x='Year', y=['Inflation %', 'Highest Mortgage Rate %']) # create the line chart
fig.update_layout(title='Inflation and Highest Mortgage Rate over Time (%)', xaxis_title='Year', yaxis_title='Percentage (%)')
fig.update_layout(width=1200, height=600)
# I will change the x axis step to 1
fig.update_xaxes(dtick=1)
fig.show()
```

The 'px.line' function is used to build a line chart after importing the Plotly Express library as px. The 'Inflation%' and 'Highest Mortgage Rate%' columns are plotted against the 'Year' column on the x-axis of the chart, which is based on the'merged_data' dataframe.



High inflation and mortgage rates coexisted in the years 1997 to 2000, indicating their combined impact on Irish property prices.
Mortgage rates have decreased while inflation rates changed between 2001 and 2007, suggesting that other variables, such as growth in the economy or changes in the labour market, may have had a greater impact on the dynamics of the housing market during this time.
Mortgage rates and inflation both rise in 2008, possibly as a result of the world financial crisis, highlighting the effects of external economic forces on their connection and the housing market.
Following the financial crisis, the housing market has been impacted since 2009 by low inflation and mortgage rates, as well as potential government initiatives and market adjustments.

3.3. Scatterplot



The linearity of the relationship between Dublin house prices and financial factors was improved by taking the logarithmic value of the former and creating scatterplots to compare their linear relationships. This enables us to gain a deeper insight into the relationships among Dublin house prices, inflation, total population, and the highest mortgage rate.

Some understanding of the dynamics of their correlations can be obtained from the scatter plots of Dublin real estate prices against economic factors. The scatter plot of Dublin home prices compared to inflation reveals a weak positive linear relationship, suggesting that inflation tends to rise slightly as Dublin home prices rise. However, there are a lot of variations in the relationship and it is not particularly strong. The Dublin house prices vs. highest mortgage rate scatter plot reveal a weakly negative linear relationship between the two variables, however, the Dublin house prices vs. total population has the same result. In general, the relationships between Dublin house prices and economic factors are weak.

4. Machine Learning

In order to create precise predictive models for the typical home price in Dublin, the research project currently applies a variety of sophisticated modelling techniques, such as Multiple Linear Regression, Lasso (L1 regularization), Ridge (L2 regularization), and decision tree regression. The basis for building these models is data spanning the years 1997 through 2014.

These several modelling techniques were chosen because of their unique advantages and capacities for managing various facets of the data. Numerous independent variables can be accommodated via multiple regression, problematic multicollinearity and overfitting issues can be addressed by Lasso and Ridge regularization techniques, and decision tree regression provides a clear, non-parametric option. The study makes sure that the best hyperparameters are found and used by using GridSearchCV, which maximizes model performance.

After these initial models are created, a thorough evaluation process is carried out to see how well they work. Finding the most accurate and dependable model among them is the main objective. This study intends to make a substantial contribution to the understanding of Dublin's housing market dynamics by utilizing these sophisticated modelling approaches and meticulously examining their results. It also aims to offer useful information for making educated decisions.

In order to estimate the relationship between housing prices and total population, inflation, and mortgage rate linear regression is used as a fundamental technique. It provides interpretability and analytical effectiveness, enabling fast analysis of the data's underlying trends.

When feature selection is required or desirable, lasso regression employs L1 regularization to encourage model sparsity. The most important features are automatically determined via lasso regression by setting some coefficients to zero. This trait aids in building a parsimonious model that concentrates on the main predictors of housing prices.

Ridge regression is helpful when the features, like Dublin and Total Population, are multicollinear. The penalty term, which is added to the loss function by reducing the magnitude of the coefficients, reduces multicollinearity. As a result, ridge regression improves model generalization and prediction accuracy by avoiding overfitting.

4.1. Importing Necessary Libraries

```
# Importing the necessary libraries
from sklearn.model_selection import GridSearchCV # Importing GridSearchCV for hyperparameter tuning
from sklearn.linear_model import Lasso # Importing Lasso for linear regression with L1 regularization
from sklearn.linear_model import Ridge # Importing Ridge for linear regression with L2 regularization
from sklearn.tree import DecisionTreeRegressor # Importing DecisionTreeRegressor for decision tree regression
from sklearn.linear_model import LinearRegression # Importing LinearRegression for linear regression
from sklearn.model_selection import train_test_split  # Importing train_test_split for splitting the data into training and testing sets
from sklearn.metrics import r2_score # Importing r2_score for calculating the R-squared value
✓  0.0s
```

The Python library Scikit-learn (sklearn) is used to train and test regression models. Hyperparameter tuning is done with GridSearchCV, linear regression with L1 and L2 regularization is done with Lasso and Ridge, decision tree regression is done with DecisionTreeRegressor, and plain linear regression is done with LinearRegression. The data

is separated between training and testing sets using the train_test_split function. The R-squared value, a gauge of the regression model's precision, is computed using the r2_score package.

4.2. Project Management Framework

The dataset is manageable because the job of predicting house values using the total population, the mortgage rate, and inflation is clearly defined and contains few variables. The CRISP-DM framework is appropriate for this particular problem.

CRISP-DM is adaptable and places a strong emphasis on comprehending the business challenge and coordinating data mining with project goals. It enables iterative modifications, ensuring the model stays accurate and relevant when new facts come to light and market circumstances alter.

Project objectives, such as displaying housing values, are set up during the Business Understanding phase. The following processes involve gathering data, preprocessing it, and using it to create and test models. The selected model is assessed, put into use, and its performance is tracked.

As a result of its emphasis on understanding the issue, coordinating attempts, and continuous improvement, CRISP-DM is highly suited for predicting real estate values using the total population, the mortgage rate, and inflation.

4.3. Multiple Linear Regression

```
# Select relevant features and target variable for Dublin from 1997 to 2014
dublin_data = merged_data.loc[(merged_data['Year'] >= 1997) & (merged_data['Year'] <= 2014), ['Dublin', 'Year', 'Total Population', 'Hi

# Split the data into training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(dublin_data.loc[:, ['Year', 'Total Population', 'Highest Mortgage Rate %', 'Inflati

# Train the model
from sklearn.linear_model import LinearRegression
linear_model = LinearRegression() #
linear_model.fit(X_train, y_train)

# Calculate R^2 using sklearn's r2_score function on the training and testing sets
from sklearn.metrics import r2_score
train_r2 = r2_score(y_train, linear_model.predict(X_train))
test_r2 = r2_score(y_test, linear_model.predict(X_test))
print("R^2 score on training set:", train_r2)
print("R^2 score on testing set:", test_r2)

✓ 0.0s                                                                    Python
R^2 score on training set: 0.7093187766081941
R^2 score on testing set: 0.7731555325463024
```

After importing the necessary libraries for machine learning, with the ".loc" technique, the code first chooses the target variable and pertinent attributes from a dataset for the city of Dublin that spans the years 1997 to 2014. A new data frame called dublin_data is subsequently created and contains these chosen features and the target variable.

After that, the train_test_split function from the sklearn.model_selection module divides the dublin_data data frame into training and testing sets. The dependent variable (Dublin) is

stored in y_train and y_test, whereas the independent variables (Year, Total Population, Highest Mortgage Rate%, and Inflation%) are placed in x_train and x_test

The LinearRegression class from the sklearn.linear_model module is then used to build a linear regression model, which is subsequently trained on the training set using the fit() function.

The r2_score() method from the sklearn.metrics module is then used by the code to get the R^2 score for both the training and testing sets. How well the model fits the data is indicated by the R^2 rating. Using the print() function, the resulting R^2 values for the training and testing sets are displayed.

With an R^2 value of 0.71 on the training set and 0.77 on the testing set, the model appears to match the data quite well and accurate, according to the results. If the training score is higher than the testing score, the model may not have been trained for a sufficient amount of time. The model could benefit from extra training cycles. It is a sign that the dataset is too simple for additional adjustments, yet the model is too simple to properly capture the patterns in the data.

Although the current linear regression result seems sufficient for future examination, it is worth investigating whether regularization approaches could enhance the model's functionality. Lasso and Ridge regression are two examples of regularization methods that can be used to avoid overfitting and possibly increase the prediction ability of a model.

The model may still be overfitting to the training data because these scores were obtained without using any regularization approaches. In addition, Lasso and Ridge regularization with the gridsearchcv hyperparameter tuning is a useful technique to improve the model's output.

4.4. Hyperparameter Tuning for Lasso (L1 Regularization)

```
# Define the hyperparameter grid to search
param_grid = {
    'alpha': [0.001, 0.01, 0.1, 1, 10, 100],
}

# Create a Lasso model
lasso = Lasso()

# Use GridSearchCV to search for the best hyperparameters
grid_search = GridSearchCV(estimator=lasso, param_grid=param_grid, cv=5, scoring='neg_mean_squared_error', n_jobs=-1)
grid_search.fit(X_train, y_train)

# Print the best hyperparameters and corresponding score
print("Best hyperparameters: ", grid_search.best_params_)
print("Best score: ", -grid_search.best_score_)
```

✓ 0.1s Python

```
Best hyperparameters:  {'alpha': 1}
Best score:  3694.9459243732686
```

This code block generates a Lasso Regression model, provides a grid search across the hyperparameter spectrum using 5-fold cross-validation (cv=5), and searches through a range of alpha values using the param_grid dictionary. This implies that the data will be separated into five subsets, the model will be trained and evaluated five times, and each subset will be

used as a validation set just once. The performance of the model will then be estimated by averaging the findings.

The cross-validation performance metric is the negative mean squared error. Using the fit() function on the grid_search object and the training data, the grid search and optimum hyperparameters are determined. The best hyperparameters and related scores are sent in the output using the grid_search object's best_params_ and best_score_ attributes.

GridSearchCV() will attempt to maximize the negative MSE, which is similar to minimizing the MSE, by utilizing negative MSE as the performance metric. This is because GridSearchCV() is built to maximize the score that its scoring function returns.

4.5. Lasso (L1 Regularization)

```
# Train the model using Lasso Regression
lasso_model = Lasso(alpha=1)  # Adjust the alpha according to the best hyperparameters
lasso_model.fit(X_train, y_train)

# Calculate R^2 using sklearn's r2_score function on the training and testing sets
train_r2 = r2_score(y_train, lasso_model.predict(X_train))
test_r2 = r2_score(y_test, lasso_model.predict(X_test))
print("R^2 score on training set:", train_r2)
print("R^2 score on testing set:", test_r2)
```

✓ 0.0s

```
R^2 score on training set: 0.7085058723776207
R^2 score on testing set: 0.7581753156011952
```

The optimum alpha value is used to train a Lasso Regression model after doing the grid search to identify the optimal hyperparameters. The performance of the model is then assessed using the R-squared scores on the training and testing sets. The testing set's R^2 score of 0.758 is lesser than the score the linear regression model attained in a previous phase, proving that the Lasso Regression model provides a worst fit for the data.

4.6. Hyperparameter Tuning for Ridge(L2 Regularization)

```
# Perform hyperparameter tuning using GridSearchCV
# Create an instance of the Ridge class
ridge = Ridge()
# estimator is the model to be used for hyperparameter tuning
# param_grid is the dictionary of hyperparameters to be used for tuning
# cv is the number of folds to be used for cross-validation
# scoring is the metric to be used for evaluating the model
# verbose is the level of verbosity
# n_jobs is the number of jobs to be run in parallel
grid_search = GridSearchCV(estimator=ridge, param_grid=param_grid, cv=5, scoring='neg_mean_squared_error', verbose=2, n_jobs=-1)

# Fit the model
grid_search.fit(X_train, y_train)

# Print the best hyperparameters and corresponding score
print("Best hyperparameters: ", grid_search.best_params_)
print("Best score: ", -grid_search.best_score_)

# Train the final model on the training set with the best hyperparameters
ridge_model = grid_search.best_estimator_
ridge_model.fit(X_train, y_train)
✓ 0.1s
```

This program uses GridSearchCV with 5-fold cross-validation and negative mean squared error as the performance indicator to identify the optimum hyperparameters for a Ridge regression. The final model is trained on the complete training set using the best hyperparameters, and the best hyperparameters are printed along with the matching score. The outcome reveals that 1, with a mean squared error of 3694.94, is the ideal value for the regularization strength hyperparameter (alpha). The optimal value for the regularization strength hyperparameter (alpha) is 1.

4.7. Ridge (L2 Regularization)

```
# Train the model using Ridge Regression
ridge_model = Ridge(alpha=1)  # Adjust the alpha according the GridsearchCV
ridge_model.fit(X_train, y_train)

# Calculate R^2 using sklearn's r2_score function on the training and testing sets
train_r2 = r2_score(y_train, ridge_model.predict(X_train))
test_r2 = r2_score(y_test, ridge_model.predict(X_test))
print("R^2 score on training set:", train_r2)
print("R^2 score on testing set:", test_r2)

✓ 0.0s
R^2 score on training set: 0.7056400959494376
R^2 score on testing set: 0.7424473987359418
```

The R^2 scores for the training and testing sets are then calculated using the Ridge regression model on the training set with an alpha value of 1. The console prints the R^2 score of 0.74, the result shows that the model has a decent ability to predict the testing set. The testing result is shown to be closer to the training result than that of the other models, indicating a good fit for

the Ridge regression model. In fact, the Ridge regression model yields the least accurate result among the other regression models applied. Moreover, the Ridge regression model has been widely used in various fields such as finance, economics, and engineering due to its robustness and interpretability. Its effectiveness in handling multicollinearity and overfitting has made it a good choice for the dataset.
The R^2 test score is gradually approaching the training set, but its performance declines after applying both regularization methods. The dataset I'm currently analyzing is too small to justify altering the parameters of the training and test sets.

4.8. Decision Tree Regressor

According to their respective correlation coefficients, the variables' non-linear relationships are depicted by the scatterplot. Particularly, it seems that there is a weak or non-linear relationship between the highest mortgage rate, total population and Dublin house prices.

The code applies a decision tree model to detect nonlinear relationships between the features and target variables. As a result, it may be inferred that decision tree models may be more suited for data that exhibits non-linear connections rather than linear correlation. To ensure optimal performance of the decision tree model, the hyperparameters will be tuned using GridSearchCV prior to its application.

```
# Define the hyperparameter grid for the grid search
param_grid = {
    'max_depth': [None] + list(range(1, 11)),
    'min_samples_split': range(2, 21),
    'min_samples_leaf': range(1, 21),
}

# Initialize the Decision Tree Regressor
tree_model = DecisionTreeRegressor(random_state=42)

# Create the GridSearchCV object
grid_search = GridSearchCV(
    estimator=tree_model,
    param_grid=param_grid,
    cv=5,
    scoring='r2',
    n_jobs=-1,
    verbose=1,
)

# Perform the grid search
grid_search.fit(X_train, y_train)

# Print the best hyperparameters and corresponding score
print("Best hyperparameters: ", grid_search.best_params_)
```

✓ 5.8s

```
Fitting 5 folds for each of 4180 candidates, totalling 20900 fits
Best hyperparameters:  {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 13}
```

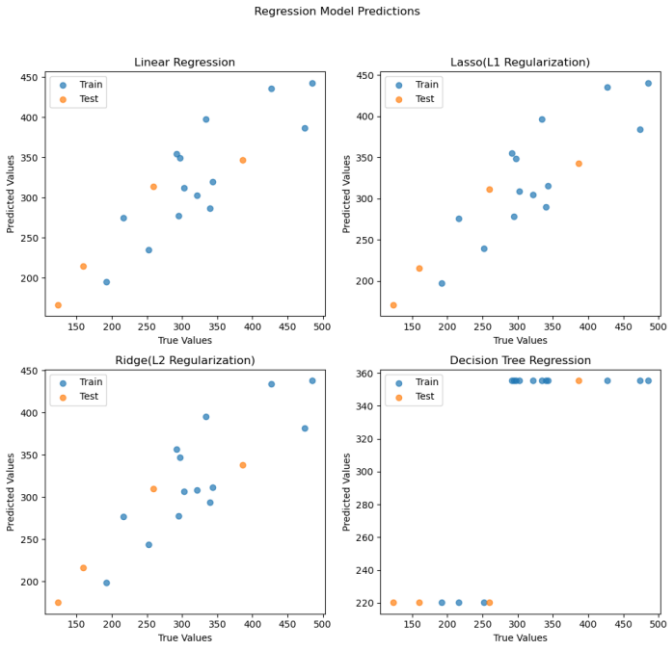
The grid search's most effective hyperparameters are:
min_samples_leaf: 1
min_samples_split: 13
max_depth: None

Based on the greatest R^2 score a measurement of how well the model fits the data. These hyperparameters were chosen by the grid search as the most effective ones.

```
# Train the model using Decision Tree Regressor
# Best hyperparameters:  {'max_depth': None, 'min_samples_leaf': 4, 'min_samples_split': 2}
tree_model = DecisionTreeRegressor(max_depth=None, min_samples_leaf=1, min_samples_split=13, random_state=42
tree_model.fit(X_train, y_train)

# Calculate R^2 using sklearn's r2_score function on the training and testing sets
train_r2 = r2_score(y_train, tree_model.predict(X_train))
test_r2 = r2_score(y_test, tree_model.predict(X_test))
print("Decision Tree R^2 score on training set:", train_r2)
print("Decision Tree R^2 score on testing set:", test_r2)

✓ 0.0s

Decision Tree R^2 score on training set: 0.44593618319089556
Decision Tree R^2 score on testing set: 0.6259176117954808
```

The Decision Tree Regressor model is trained using this code and the optimal hyperparameters are identified by GridSearchCV on the training data. The target variable is then predicted using the model, and the R^2 score is calculated. The results demonstrate that the Decision Tree Regressor model has an R^2 score of 0.4459 on the training set and 0.6259 on the testing set, indicating that the model is likely overfitting the training set but has a modest predictive power on the testing set. While the R^2 score may not be satisfactory, a decision tree can still be a valuable tool for handling the sharp price increases and non-linear relationships between variables in the dataset.

4.9. Comparison of the Machine Learning Models



The performance of four different regression models in predicting house prices is contrasted in this graph. For higher-priced homes, the first model, a simple Linear Regression model, is found to perform well with little variance.

The plot also demonstrates that, in general, the Lasso Regression model performs similarly to the Linear regression and Ridge Regression models, but that, as a result of its unique regularization method, it exhibits greater variation for higher-priced homes.

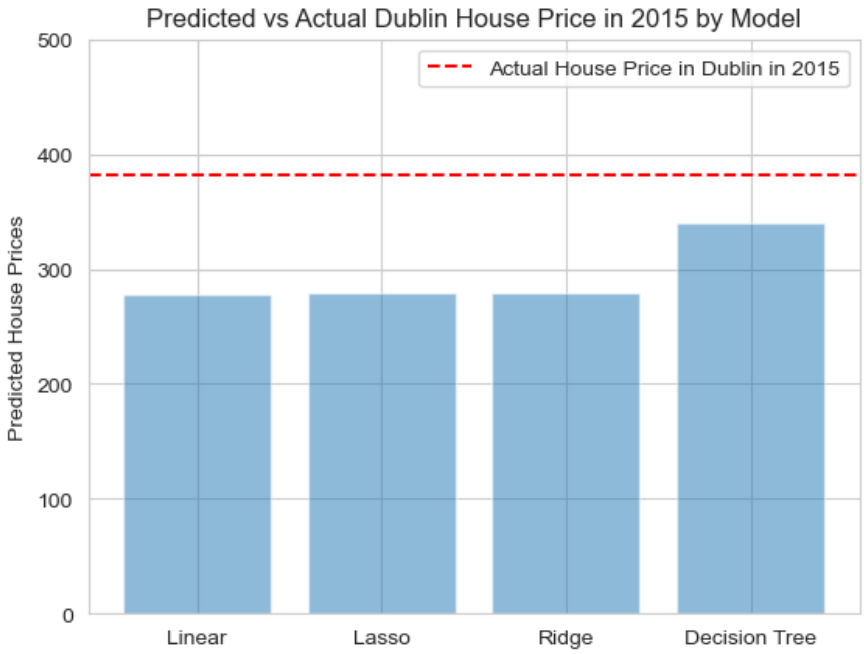
In order to avoid overfitting, the third model is a Ridge Regression model, a kind of Linear Regression. The plot shows that regularization causes the ridge regression model to perform better for more expensive residences than the linear regression model.

The fourth model presented is the Decision Tree Regressor model, which is a non-linear model capable of capturing non-linear relationships between input and output variables. The plotted results demonstrate that the decision tree model has greater deviation than the other models for higher-priced homes, but greater precision for lower-priced homes, due to its ability to capture non-linear associations.

The Decision Tree Regression model creates a tree with a few splits, resulting in two different regions with distinct predicted values. The presence of two horizontal lines indicates that the model has created two distinct levels of predictions. Unlike the other models that rely on linear combinations of input features, the Decision Tree makes predictions based on the structure of the tree itself. The data set is not large enough to make the more complex calculations.

4.10. Comparison of Predicted Outcomes to Actual Market Prices

As previously mentioned, the data from 1997 to 2014 is utilized to predict the average house prices in Dublin for 2015 in the machine learning models. These models are then applied and their output is compared to the actual average house price in Dublin for 2015 which we already have in the dataset. In order to determine which model provides the most accurate prediction.



The decision tree has the closest result due to it captures more complex and non-linear relationships between the features and the target variable. The other models, such as linear regression, lasso regression, and ridge regression, are linear models that may not capture

these relationships as effectively. It's essential to emphasize how small the dataset in consideration is.

Due to its capacity to manage multicollinearity and avoid overfitting, ridge regression may, in some circumstances, be preferable to linear and lasso regression. This happens when there is a high correlation between two or more independent variables in the dataset, which can cause unstable results in linear regression.

However, it is essential to note that decision trees can sometimes overfit the data, meaning they perform well on the training set but not as well on new, unseen data. In this particular example, the decision tree happened to provide the most accurate prediction for 2015.

Bibliography

FRIEDMAN, M., 1995. *The Role of Monetary Policy.* London: © Macmillan Publishers Limited 1995.

GAUR, D., 2022. *Kaggle.* [Online]
Available at: https://www.kaggle.com/code/durgancegaur/a-guide-to-any-classification-problem
[Accessed 2023].

McKinney, W., 2011. pandas: a Foundational Python Library for Data Analysis and Statistics. *pandas: a Foundational Python Library for Data Analysis and Statistics,* pp. 1-9.

World Data Bank, W. D. I., 2016. *World Data Bank.* [Online]
Available at:
https://databank.worldbank.org/reports.aspx?source=2&series=FP.CPI.TOTL.ZG&country=IRL
[Accessed 2023].