VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

School of Computer Science and Engineering

VIT Chennai

Vandalur - Kelambakkam Road, Chennai - 600 127

Final Review Report

Programme: B.Tech – CSE with AI & ML

Course: CSE2004

Slot: D2

Faculty: M. PREMALATHA

Component: J

HOUSES PRICE PREDICTION

Team Member(s):

PIYUSH MUDGAL (20BAI1156)
PRITHVI M R (20BAI1146)
DESHARAJU SAI ABHISHEK (20BAI1115)

Abstract

House prices increase every year, so there is a need for a system to predict house prices in the future. House price prediction can help the developer determine the selling price of a house and can help the customer to arrange the right time to purchase a house. There are many factors that influence the price of a house some of which include their location, size and proximity to areas of interest/ convenience. So, with the advent of data mining and machine learning algorithms we aim to develop a model capable of predicting the prices of houses based on a particular dataset including features such as population, median income, and median house prices for each block group in a city.

Introduction

1. Data Set Description

Housing Info:

Column	Non-Null	Count	Data type
Longitude	20640	Non-null	float64
Latitude	20640	Non-null	float64
Housing_median_age	20640	Non-null	float64
Total_rooms	20640	Non-null	float64
Total_bedrooms	20433	Non-null	float64
Population	20640	Non-null	float64
Households	20640	Non-null	float64
Median_income	20640	Non-null	float64
Median_house_value	20640	Non-null	float64
Ocean_proximity	20640	Non-null	object

dataypes: float64(9), object(1)
memory usage: 1.6+ MB

There are 20,640 instances in the dataset. Note that the total_bedrooms attribute has only 20,433 non-zero values, which means 207 districts do not contain values.
All attributes are numeric except for the ocean_proximity field. Its type is an object, so it can contain any type of Python object. We can find out which categories exist in that column and how many districts belong to each category by using the value_counts() method:
housing.ocean_proximity.value_counts()

Dataset is taken from housing.csv file

Link:
https://raw.githubusercontent.com/ageron/handson-ml/master/datasets/housing/housing.csv

Normalization and ER diagram

- **Functional dependencies**

{longitude, latitude, housing_median_age, total_rooms, total_bedrooms, population, households, median_income, ocean_proximity} → { median_house_value}

{total_bedrooms, population} → {housing_median_age}

{median_income} → {longitude, latitude}

{longitude, latitude} → {ocean_proximity}

- **Minimal Cover (after removing redundancies)**

{longitude, latitude, housing_median_age, total_bedrooms, population, households} ⟶ { median_house_value}

{total_bedrooms, population}⟶{total_rooms}

{housing_median_age} ⟶{median_income)

{longitude, latitude} ⟶{ocean_proximity}

- **Normalization to 2NF**

Table 1: **Attributes: [population, total_bedrooms, total_rooms]**
Functional dependencies: {total_bedrooms, population }⟶{total_rooms}

Table 2: **Attributes: [housing_median_age, median_income]**
Functional dependencies: {housing_median_age} ⟶{median_income)

Table 3: **Attributes: [latitude, longitude, ocean_proximity]**
Functional dependencies: {longitude, latitude} ⟶{ocean_proximity}

Table 4: **Attributes: [longitude, latitude, housing_median_age, total_bedrooms, population, households, median_house_value]**
Functional dependencies: {longitude, latitude, housing_median_age, total_bedrooms, population, households}⟶ { median_house_value}

- **Normalization to 3NF & BCNF**

Table 1: **Attributes: [population, total_bedrooms, total_rooms]**
Functional dependencies: {total_bedrooms, population }⟶{total_rooms}

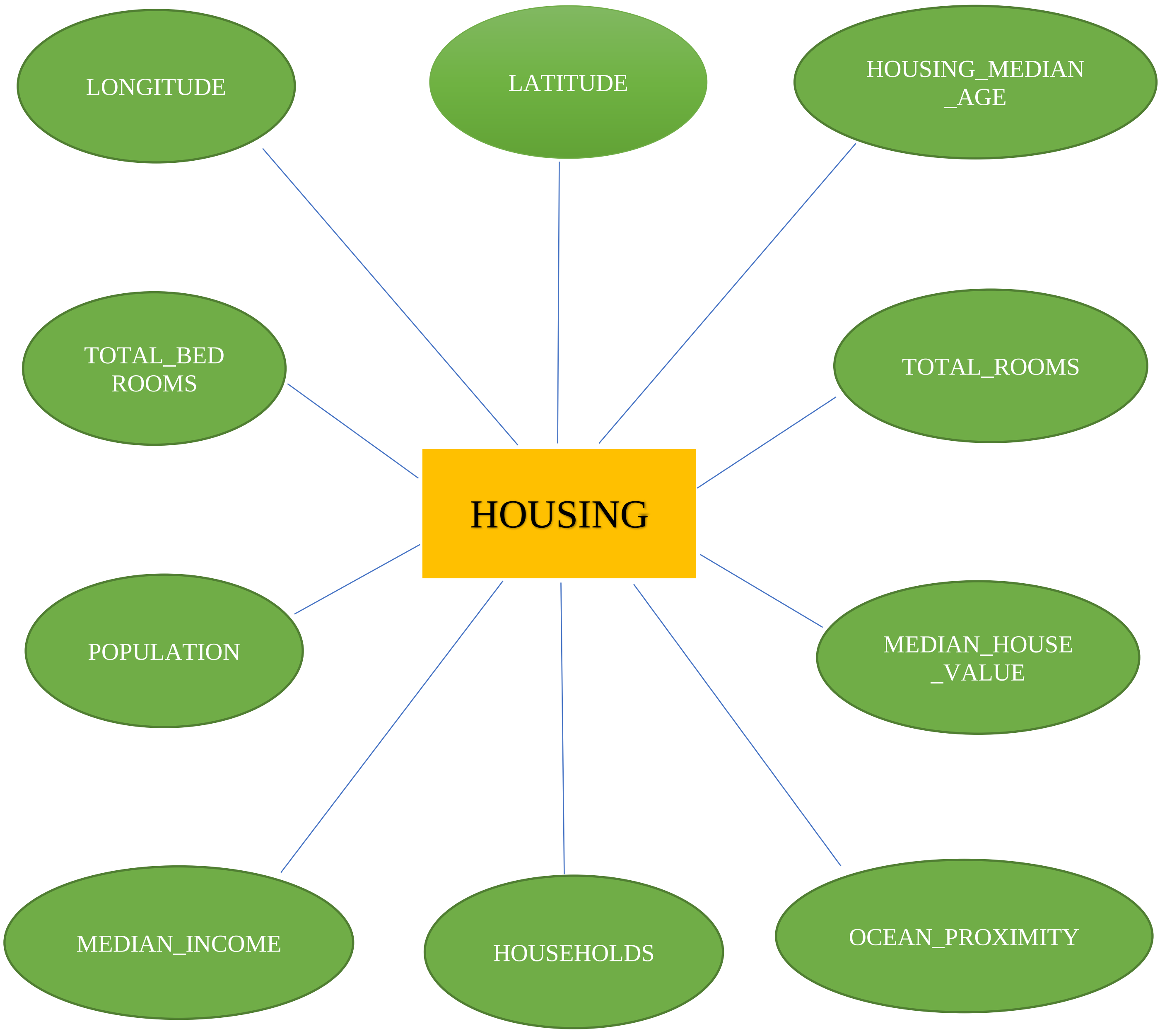
Table 2: **Attributes: [housing_median_age, median_income]**
Functional dependencies: {housing_median_age} ⟶{median_income)

Table 3: **Attributes: [latitude, longitude, ocean_proximity]**
Functional dependencies: {longitude, latitude} ⟶{ocean_proximity}

Table 4: **Attributes: [longitude, latitude, housing_median_age, total_bedrooms, population, households, median_house_value]**

Functional dependencies: {longitude, latitude, housing_median_age, total_bedrooms, population, households} → { median_house_value}

ER DIAGRAM:



Methodology and Algorithm used

Methodology:

We've retrieved the data set, ordered it and polished it. We've dealt with the missing values in the data set. We've separated the price column in the data set so we could work on the rest of the data set and find some suitable relation for predicting the price. We've split the dataset into two parts, training set and testing set. 70% of the original data is given to training set and 30% to testing set. We will then apply linear and polynomial regressions on the training set. We'll find the relation in the data and then apply it on the testing set and predict the value of house prices. We will also find the accuracy of out result.

Algorithms used:

1) Linear Regression:

Linear regression is a basic and commonly used type of predictive analysis. The overall idea of regression is to examine two things: (1) does a set of predictor variables do a good job in predicting an outcome (dependent) variable? (2) Which variables in particular are significant predictors of the outcome variable, and in what way do they–indicated by the magnitude and sign of the beta estimates–impact the outcome variable? These regression estimates are used to explain the relationship between one dependent variable and one or more independent variables. The simplest form of the regression equation with one dependent and one independent variable is defined by the formula $y = c + b*x$, where y = estimated dependent variable score, c = constant, b = regression coefficient, and x = score on the independent variable.

2) Polynomial Regression

Polynomial Regression is a regression algorithm that models the relationship between a dependent(y) and independent variable(x) as nth degree polynomial. It is a linear model with some modification in order to increase the accuracy. The dataset used in Polynomial regression for training is of non-linear nature. It makes use of a linear regression model to fit the complicated and non-linear functions and datasets. Hence, in Polynomial regression, the original features are converted into polynomial features of required degree (2,3,..,n) and then modelled using a linear model.

Implementation

- LINEAR REGRESSION ALGORITHM:

Packages used:

- For data visualization and interpretation:
  - Pandas
  - Matplotlib (Modules: pyplot)
  - NumPy

- For training, test sets and linear regression algorithm
  - Sklearn (Modules: model_selection ⇒ train_test_split, model_selection ⇒ StratifiedShuffleSplit, LinearRegression, mean_squared_error)

- For data preparation
  - OneHotEncoder
  - StandardScaler
  - SimpleImputer
  - ColumnTransformer
  - Pipeline

Function used:

- .head()
- .info()
- .valuecounts()
- .hist()
- .show()

Code and implementation:

```
import pandas as pd
housing = pd.read_csv("housing.csv")
housing.head()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0	NEAR BAY
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0	NEAR BAY
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0	NEAR BAY
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	341300.0	NEAR BAY
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	342200.0	NEAR BAY

```
housing.shape
```

```
(20640, 10)
```

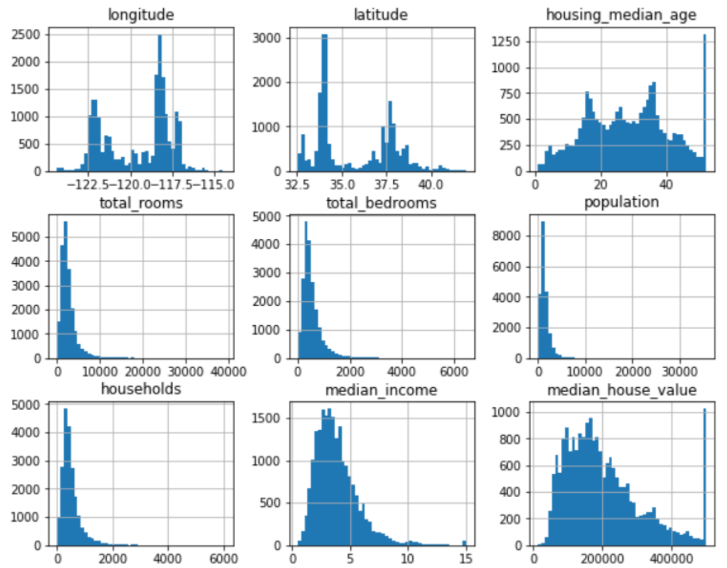
```
housing.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   longitude           20640 non-null  float64
 1   latitude            20640 non-null  float64
 2   housing_median_age  20640 non-null  float64
 3   total_rooms         20640 non-null  float64
 4   total_bedrooms      20433 non-null  float64
 5   population          20640 non-null  float64
 6   households          20640 non-null  float64
 7   median_income       20640 non-null  float64
 8   median_house_value  20640 non-null  float64
 9   ocean_proximity     20640 non-null  object
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

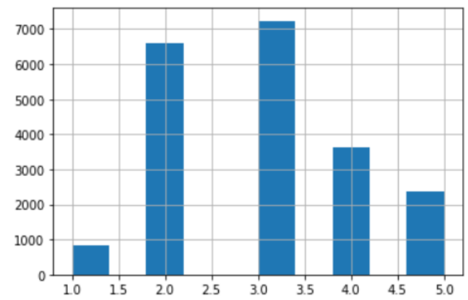
```
housing.ocean_proximity.value_counts()
```

```
<1H OCEAN     9136
INLAND        6551
NEAR OCEAN    2658
NEAR BAY      2290
ISLAND           5
Name: ocean_proximity, dtype: int64
```

```
import matplotlib.pyplot as plt
housing.hist(bins=50, figsize=(10, 8))
plt.show()
```




```
import numpy as np
housing['income_cat'] = pd.cut(housing['median_income'], bins=[0., 1.5, 3.0, 4.5, 6., np.inf], labels=[1, 2, 3, 4, 5])
housing['income_cat'].hist()
plt.show()
```

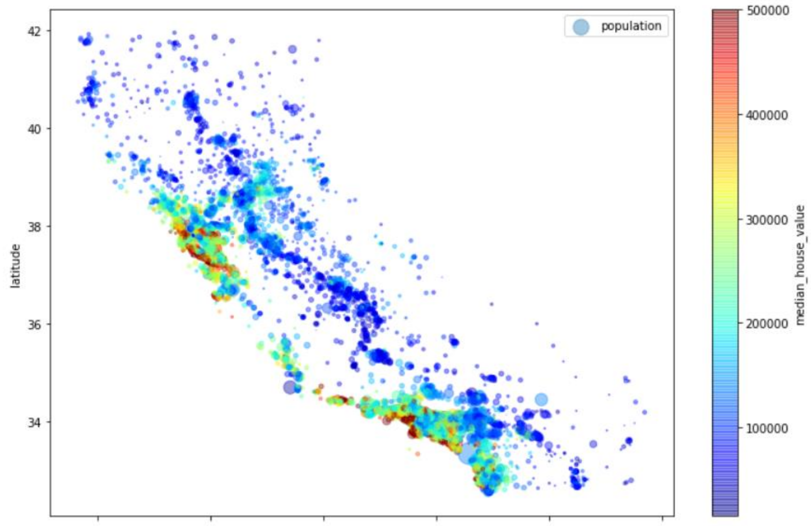


```
from sklearn.model_selection import StratifiedShuffleSplit
split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
for train_index, test_index in split.split(housing, housing["income_cat"]):
    strat_train_set = housing.loc[train_index]
    strat_test_set = housing.loc[test_index]
print(strat_test_set['income_cat'].value_counts()*100 / len(strat_test_set))
```

```
3    35.053295
2    31.879845
4    17.635659
5    11.458333
1     3.972868
Name: income_cat, dtype: float64
```

```
for set_ in (strat_train_set, strat_test_set):
    set_.drop('income_cat', axis=1, inplace=True)
housing = strat_train_set.copy()
```

```
housing.plot(kind='scatter', x='longitude', y='latitude', alpha=0.4, s=housing['population']/100, label='population',
figsize=(12, 8), c='median_house_value', cmap=plt.get_cmap('jet'), colorbar=True)
plt.legend()
plt.show()
```




```
corr_matrix = housing.corr()
print(corr_matrix.median_house_value.sort_values(ascending=False))
```

```
median_house_value    1.000000
median_income         0.687160
total_rooms           0.135097
housing_median_age    0.114110
households            0.064506
total_bedrooms        0.047689
population           -0.026920
longitude            -0.047432
latitude             -0.142724
Name: median_house_value, dtype: float64
```

```python
# Data Preparation
housing = strat_train_set.drop("median_house_value", axis=1)      #X variable
housing_labels = strat_train_set["median_house_value"].copy()     #Y variable


housing_num = housing.drop("ocean_proximity", axis=1)

from sklearn.base import BaseEstimator, TransformerMixin

# column index
rooms_ix, bedrooms_ix, population_ix, households_ix = 3, 4, 5, 6

class CombinedAttributesAdder(BaseEstimator, TransformerMixin):
    def __init__(self, add_bedrooms_per_room=True): # no *args or **kargs
        self.add_bedrooms_per_room = add_bedrooms_per_room
    def fit(self, X, y=None):
        return self  # nothing else to do
    def transform(self, X):
        rooms_per_household =X[:, rooms_ix] / X[:, households_ix]
        population_per_household = X[:, population_ix] / X[:, households_ix]
        if self.add_bedrooms_per_room:
            bedrooms_per_room = X[:, bedrooms_ix] / X[:, rooms_ix]
            return np.c_[X, rooms_per_household, population_per_household,
                         bedrooms_per_room]
        else:
            return np.c_[X, rooms_per_household, population_per_household]
```

```python
from sklearn.preprocessing import OneHotEncoder
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
num_pipeline = Pipeline([
    ('imputer',SimpleImputer(strategy="median")),
    ('attribs_adder', CombinedAttributesAdder()),
    ('std_scaler', StandardScaler()),
])

from sklearn.compose import ColumnTransformer
num_attribs = list(housing_num)
cat_attribs = ["ocean_proximity"]
full_pipeline = ColumnTransformer([
    ("num", num_pipeline, num_attribs),
    ("cat", OneHotEncoder(), cat_attribs),
])
housing_prepared = full_pipeline.fit_transform(housing)
```

```python
from sklearn.linear_model import LinearRegression
lin_reg = LinearRegression()
lin_reg.fit(housing_prepared, housing_labels)

data = housing.iloc[:5]
labels = housing_labels.iloc[:5]
data_preparation = full_pipeline.transform(data)
print("Predictions: ", lin_reg.predict(data_preparation))
```

```
Predictions:  [210644.60459286 317768.80697211 210956.43331178  59218.98886849
 189747.55849879]
```

```python
from sklearn.metrics import mean_squared_error
print("The mean square error for linear regression model:",end=" ")
print(mean_squared_error(y_true = housing_labels, y_pred = lin_reg.predict(housing_prepared)))
```

```
The mean square error for linear regression model: 4709829587.971121
```

- POLYNOMIAL REGRESSION ALGORITHM:

Packages used:

- Sklearn (Modules: r2_score, PolnomialFeatures, LinearRegression)

Code and implementation:

```
[ ] from sklearn.metrics import r2_score
    print("The r2 score for linear regression model:",end=" ")
    print(r2_score(y_true = housing_labels, y_pred = lin_reg.predict(housing_prepared)))
```

```
The r2 score for linear regression model: 0.6481624842804428
```

```
from sklearn.preprocessing import PolynomialFeatures

poly = PolynomialFeatures(degree = 2)
X_poly = poly.fit_transform(housing_prepared)

poly.fit(X_poly, housing_labels)
lin2 = LinearRegression()
lin2.fit(X_poly, housing_labels)
```

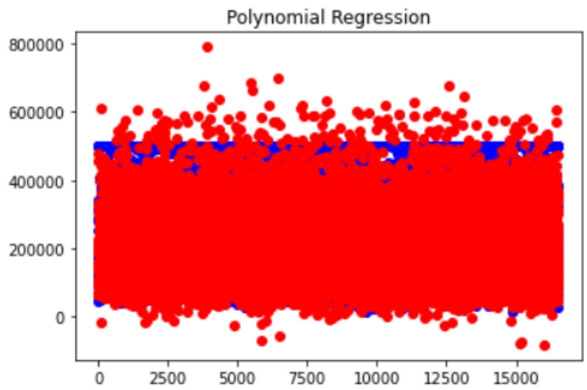
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

```
[ ] x_val = []
    for i in range(1, 16513):
      x_val.append(i)

    plt.scatter(x_val, housing_labels, color = 'blue')

    plt.scatter(x_val, lin2.predict(X_poly), color = 'red')
    plt.title('Polynomial Regression')
```

```
Text(0.5, 1.0, 'Polynomial Regression')
```




```
[ ] r2_SCORE=[]
    for i in range(2,6):
        poly=PolynomialFeatures(degree=i)
        I_poly=poly.fit_transform(housing_prepared)
        lini=LinearRegression()
        lini.fit(I_poly,housing_labels)
        print("The r2 score for Polynomial Regression of degree:",i,end=" ")
        r=r2_score(y_true=housing_labels, y_pred=lini.predict(I_poly))
        r2_SCORE.append(r)
        print('%.4f'%r)
```

```
The r2 score for Polynomial Regression of degree: 2 0.7272
The r2 score for Polynomial Regression of degree: 3 0.7993
The r2 score for Polynomial Regression of degree: 4 0.8526
The r2 score for Polynomial Regression of degree: 5 0.9202
```

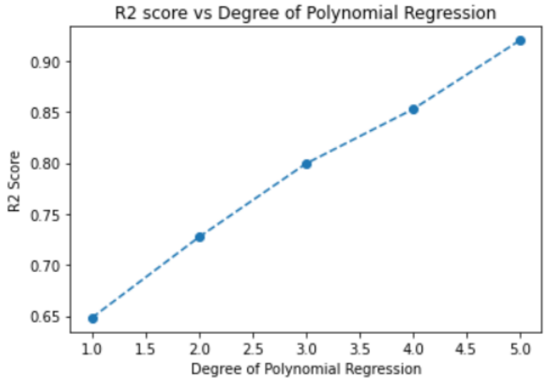
```
[ ] y_pt=r2_SCORE
    y_pt.insert(0,r2_score(y_true = housing_labels, y_pred = lin_reg.predict(housing_prepared)))
    x_pt=[1,2,3,4,5]
```

```
plt.plot(x_pt, y_pt,linestyle='dashed', marker='o')

# naming the x axis
plt.xlabel('Degree of Polynomial Regression')
# naming the y axis
plt.ylabel('R2 Score')

# giving a title to my graph
plt.title('R2 score vs Degree of Polynomial Regression')
```

```
Text(0.5, 1.0, 'R2 score vs Degree of Polynomial Regression')
```



Results and Discussion

We've found out the R2 scores for the algorithms implemented
Linear Regression:
Polynomial Regression
For degree = 2: 0.7272
For degree = 3: 0.7993
For degree = 4: 0.8526
For degree = 5: 0.9202

R-squared is a statistical measure of how close the data are to the fitted regression line. It is also known as the coefficient of determination, or the coefficient of multiple determination for multiple regression.

From the R2 scores of the different algorithms we can tell which algorithm is the most accurate for our dataset and model,
Polynomial Regression of degree 5 having the maximum value of R2 score is therefore the most accurate, but it also has high time complexity
Polynomial Regression of degree 4 was very close whilst having relatively low time complexity

Conclusion

Therefore, with this project, we have developed and demonstrated a linear regression model and a polynomial regression model up to degree 5 for the prediction of housing prices.

We have also shown that the polynomial regression algorithm is superior in terms of accuracy of the predicted prices to the linear model.