

PYTHON PROJECT
ON
California Housing Price Prediction

Submitted

By

UTPALA MOHAPATRA

PROJECT DESCRIPTION

Background of Problem Statement :

The US Census Bureau has published California Census Data which has 10 types of metrics such as the population, median income, median housing price, and so on for each block group in California. The dataset also serves as an input for project scoping and tries to specify the functional and nonfunctional requirements for it.

Problem Objective :

The project aims at building a model of housing prices to predict median house values in California using the provided dataset. This model should learn from the data and be able to predict the median housing price in any district, given all the other metrics.

Districts or block groups are the smallest geographical units for which the US Census Bureau publishes sample data (a block group typically has a population of 600 to 3,000 people). There are 20,640 districts in the project dataset.

Domain: Finance and Housing

DATA DICTIONARY

Field	Description
longitude	(signed numeric - float) : Longitude value for the block in California, USA
latitude	(numeric - float) : Latitude value for the block in California, USA
housing_median_age	(numeric - int) : Median age of the house in the block
total_rooms	(numeric - int) : Count of the total number of rooms (excluding bedrooms) in all houses in the block
total_bedrooms	(numeric - float) : Count of the total number of bedrooms in all houses in the block
population	(numeric - int) : Count of the total number of population in the block
households	(numeric - int) : Count of the total number of households in the block
median_income	(numeric - float) : Median of the total household income of all the houses in the block
ocean_proximity	(numeric - categorical) : Type of the landscape of the block [Unique Values : 'NEAR BAY', '<1H OCEAN', 'INLAND', 'NEAR OCEAN', 'ISLAND']
median_house_value	(numeric - int) : Median of the household prices of all the houses in the block

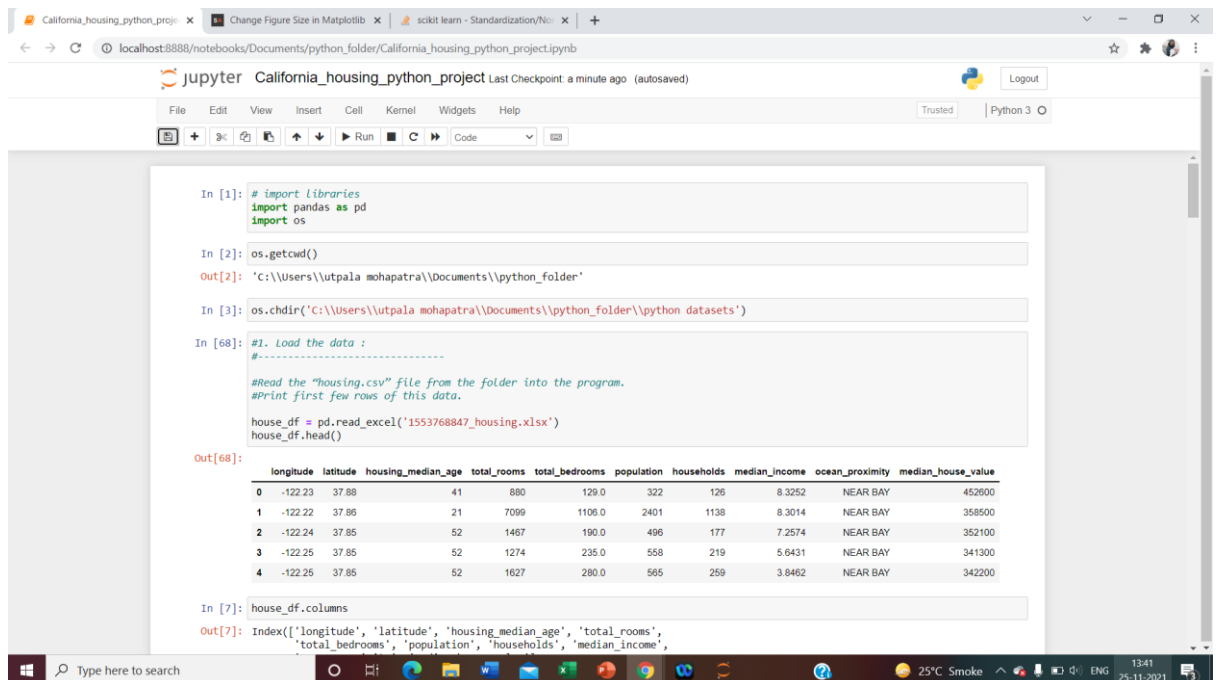
Dataset Size : 20640 rows x 10 columns

ANALYSIS OF TASK

Problem statement – 1>

Load the data :

- Read the “**housing.csv**” file from the folder into the program.
- Print first few rows of this data.



The screenshot shows a Jupyter Notebook window titled "California_housing_python_project". The notebook contains the following code and output:

```
In [1]: # import libraries
import pandas as pd
import os

In [2]: os.getcwd()
Out[2]: 'c:\\Users\\utpala mohapatra\\Documents\\python_folder'
```

```
In [3]: os.chdir('c:\\Users\\utpala mohapatra\\Documents\\python_folder\\python_datasets')
```

```
In [68]: #1. Load the data :
#-----
#Read the "housing.csv" file from the folder into the program.
#Print first few rows of this data.

house_df = pd.read_excel('1553768847_housing.xlsx')
house_df.head()
```

The output of the last cell is a pandas DataFrame with 10 columns: longitude, latitude, housing_median_age, total_rooms, total_bedrooms, population, households, median_income, ocean_proximity, and median_house_value. The first five rows of data are displayed:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	ocean_proximity	median_house_value
0	-122.23	37.88	41	880	129.0	322	126	8.3252	NEAR BAY	452600
1	-122.22	37.86	21	7099	1106.0	2401	1138	8.3014	NEAR BAY	358500
2	-122.24	37.85	52	1467	190.0	496	177	7.2574	NEAR BAY	352100
3	-122.25	37.85	52	1274	235.0	558	219	5.6431	NEAR BAY	341300
4	-122.25	37.85	52	1627	280.0	565	259	3.8462	NEAR BAY	342200

```
In [7]: house_df.columns
Out[7]: Index(['longitude', 'latitude', 'housing_median_age', 'total_rooms',
            'total_bedrooms', 'population', 'households', 'median_income',
```

Problem statement -2>

. Handle missing values :

- Fill the missing values with the mean of the respective column.

The screenshot shows a Jupyter Notebook interface with the following code and output:

```
In [8]: #2. Handle missing values :  
#-----  
#checking for missing values  
house_df.isnull().any(axis=1).sum()  
  
Out[8]: 207  
  
In [9]: # If there is any missing value , replace it with the column mean.  
house_df_means = house_df.mean()  
house_df = house_df.fillna(house_df_means)  
house_df  
  
Out[9]:
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	ocean_proximity	median_house_value
0	-122.23	37.88	41	880	129.0	322	126	8.3252	NEAR BAY	452600
1	-122.22	37.86	21	7099	1106.0	2401	1138	8.3014	NEAR BAY	358500
2	-122.24	37.85	52	1467	190.0	496	177	7.2574	NEAR BAY	352100
3	-122.25	37.85	52	1274	235.0	558	219	5.6431	NEAR BAY	341300
4	-122.25	37.85	52	1627	280.0	565	259	3.8462	NEAR BAY	342200
...
20635	-121.09	39.48	25	1665	374.0	845	330	1.5603	INLAND	78100
20636	-121.21	39.49	18	697	150.0	356	114	2.5568	INLAND	77100
20637	-121.22	39.43	17	2254	485.0	1007	433	1.7000	INLAND	92300
20638	-121.32	39.43	18	1860	409.0	741	349	1.8672	INLAND	84700
20639	-121.24	39.37	16	2785	616.0	1387	530	2.3886	INLAND	89400

20640 rows x 10 columns

```
In [20]: #3. Encode categorical data :  
#-----
```

Problem statement -3>

Encode categorical data :

- Convert categorical column in the dataset to numerical data.
- Extract input (X) and output (Y) data from the dataset.

The screenshot shows a Jupyter Notebook interface with the following code and output:

```
In [20]: #3. Encode categorical data :  
#-----  
#check the types of categorical variable  
house_df.ocean_proximity.unique()  
  
Out[20]: array(['NEAR BAY', '<1H OCEAN', 'INLAND', 'NEAR OCEAN', 'ISLAND'],  
      dtype=object)  
  
In [11]: # create new numeric variable for ocean_proximity  
house_df['ocean_proximity_num']=house_df.ocean_proximity.map({'NEAR BAY':0, '<1H OCEAN':1, 'INLAND':2, 'NEAR OCEAN':3, 'ISLAND':4})  
house_df  
  
Out[11]:
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	ocean_proximity	median_house_value
0	-122.23	37.88	41	880	129.0	322	126	8.3252	NEAR BAY	452600
1	-122.22	37.86	21	7099	1106.0	2401	1138	8.3014	NEAR BAY	358500
2	-122.24	37.85	52	1467	190.0	496	177	7.2574	NEAR BAY	352100
3	-122.25	37.85	52	1274	235.0	558	219	5.6431	NEAR BAY	341300
4	-122.25	37.85	52	1627	280.0	565	259	3.8462	NEAR BAY	342200
...
20635	-121.09	39.48	25	1665	374.0	845	330	1.5603	INLAND	78100
20636	-121.21	39.49	18	697	150.0	356	114	2.5568	INLAND	77100
20637	-121.22	39.43	17	2254	485.0	1007	433	1.7000	INLAND	92300
20638	-121.32	39.43	18	1860	409.0	741	349	1.8672	INLAND	84700
20639	-121.24	39.37	16	2785	616.0	1387	530	2.3886	INLAND	89400

20640 rows x 11 columns

The screenshot shows a Jupyter Notebook interface with the following content:

```
In [12]: x = pd.DataFrame(house_df.drop(['median_house_value', 'ocean_proximity'], axis=1))
x.head(10)
```

Out[12]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	ocean_proximity_num
0	-122.23	37.88	41	880	129.0	322	126	8.3252	0
1	-122.22	37.86	21	7099	1106.0	2401	1138	8.3014	0
2	-122.24	37.85	52	1467	190.0	496	177	7.2574	0
3	-122.25	37.85	52	1274	235.0	558	219	5.6431	0
4	-122.25	37.85	52	1627	280.0	565	259	3.8462	0
5	-122.25	37.85	52	919	213.0	413	193	4.0368	0
6	-122.25	37.84	52	2535	489.0	1094	514	3.6591	0
7	-122.25	37.84	52	3104	687.0	1157	647	3.1200	0
8	-122.26	37.84	42	2555	665.0	1206	595	2.0804	0
9	-122.25	37.84	52	3549	707.0	1551	714	3.6912	0

```
In [13]: y = house_df['median_house_value']
y
```

Out[13]:

```
0      452600
1     3585000
2     352100
3     341300
4     342200
...
20635     78100
20636     77100
20637     92300
20638     84700
20639     89400
Name: median_house_value, Length: 20640, dtype: int64
```

Problem statement -4>

Split the dataset :

- Split the data into 80% training dataset and 20% test dataset.

The screenshot shows a Jupyter Notebook interface with the following content:

```
In [15]: #4. Split the dataset :
#-----
#Split the data into 80% training dataset and 20% test dataset.
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=.20)
```

In [16]: x_train.shape

Out[16]: (16512, 9)

In [17]: y_train.shape

Out[17]: (16512,)

In [18]: x_test.shape

Out[18]: (4128, 9)

In [19]: y_test.shape

Out[19]: (4128,)

In [21]: x_train.isnull().any(axis=1).sum()

Out[21]: 0

In [24]: x_test.isnull().any(axis=1).sum()

Out[24]: 0

In [80]: #5. Standardize data :

Problem statement -5>

Standardize data :

- Standardize training and test datasets.

A screenshot of a Jupyter Notebook interface. The browser tabs at the top include 'California_housing_python_proj...', 'Change Figure Size in Matplotlib', and 'scikit learn - Standardization/No...'. The address bar shows 'localhost:8888/notebooks/Documents/python_folder/Caliornia_housing_python_project.ipynb'. The notebook title is 'California_housing_python_project' with a 'Last Checkpoint: 5 minutes ago (autosaved)' status. The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations, running cells, and code execution. The code area shows two cells. The first cell, 'In [80]:', contains code for standardizing data using sklearn's StandardScaler. It imports preprocessing, creates a StandardScaler, fits it on x_train, and transforms both x_train and x_test. The output, 'Out[80]:', displays a 2D array of standardized features. The second cell, 'In [82]:', is a comment indicating the next step: 'Perform Linear Regression'.

```
In [80]: #5. Standardize data :
#-----
#Standardize training and test datasets.

from sklearn import preprocessing
train_scaler = preprocessing.StandardScaler().fit(x_train)
x_train_scaled = train_scaler.transform(x_train)
x_test_scaled = train_scaler.transform(x_test)

Out[80]: array([[ 1.15249792, -0.84026056, -1.31758597, ..., -0.8925193 ,
                 -0.60357393,  0.63657238],
                [-1.19195459,  0.75973767,  0.50483201, ..., -0.85319078,
                 1.20093205, -0.53523176],
                [ 0.25961069,  0.21545675, -1.55518831, ...,  0.17983825,
                 0.07725795,  0.63657238],
                ...,
                [-0.67318212,  1.84360744,  1.53364217, ..., -0.92922591,
                 -1.0483037 ,  0.63657238],
                [-0.19930342,  0.37967944, -0.84238128, ..., -0.73258333,
                 -0.88990608,  0.63657238],
                [-0.59337097,  1.39316806, -1.1591844 , ..., -0.1636308 ,
                 -0.67524873,  0.63657238]])

In [82]: #6. Perform Linear Regression :
#-----
#Perform Linear Regression on training data.

lm_house = LinearRegression().fit(x_train_scaled,y_train)

In [81]: print(house_df.columns,lm_house.coef_)
print(lm_house.intercept_)
```

Problem statement -6>

Perform Linear Regression :

- Perform Linear Regression on training data.
- Predict output for test dataset using the fitted model.
- Print root mean squared error (RMSE) from Linear Regression.

A screenshot of a Jupyter Notebook interface, continuing from the previous one. The browser tabs and address bar are the same. The notebook title is 'California_housing_python_project' with a 'Last Checkpoint: 6 minutes ago (autosaved)' status. The code area shows three cells. The first cell, 'In [82]:', contains the same code as the previous notebook for fitting a LinearRegression model. The second cell, 'In [81]:', prints the columns of the house dataset and the coefficients of the fitted model. The output shows the column names and a long list of numerical coefficients. The third cell, 'In [84]:', contains code for predicting house prices for the test dataset using the fitted model. The output, 'Out[84]:', displays a 1D array of predicted prices. The fourth cell, 'In [85]:', contains code for calculating the root mean squared error (RMSE) using sklearn's mean_squared_error function and the math library's sqrt function. The output, '68167.11862510696', is shown at the bottom.

```
In [82]: #6. Perform Linear Regression :
#-----
#Perform Linear Regression on training data.

lm_house = LinearRegression().fit(x_train_scaled,y_train)

In [81]: print(house_df.columns,lm_house.coef_)
print(lm_house.intercept_)

Index(['longitude', 'latitude', 'housing_median_age', 'total_rooms',
       'total_bedrooms', 'population', 'households', 'median_income',
       'ocean_proximity', 'median_house_value'],
      dtype='object') [-83185.52054642 -88823.95175485 13993.06753142 -14718.49542039
       37076.88130873 -44114.9577891 26463.2705696 75482.82626531
       -3019.08042945]
206921.81274224748

In [84]: #Predict output for test dataset using the fitted model.

pred_house_price = lm_house.predict(x_test_scaled)
pred_house_price

Out[84]: array([[112048.24441979, 330161.02135485, 138606.42026473, ...,
                43864.31676901,  98421.17311607,  72529.66350603]])

In [85]: #Print root mean squared error (RMSE) from Linear Regression.

from sklearn.metrics import mean_squared_error
from math import sqrt
print(sqrt(mean_squared_error(y_test,pred_house_price)))

68167.11862510696
```

Problem statement -7>

Bonus exercise: Perform Linear Regression with one independent variable :

- Extract just the median_income column from the independent variables (from **X_train** and **X_test**).
- Perform Linear Regression to predict housing values based on **median_income**.
- Predict output for test dataset using the fitted model.

```
California_housing_python_project x Change Figure Size in Matplotlib x scikit learn - Standardization/No x +
localhost:8888/notebooks/Documents/python_folder/California_housing_python_project.ipynb
jupyter California_housing_python_project Last Checkpoint: 6 minutes ago (autosaved)
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3
In [46]: #7. Bonus exercise: Perform Linear Regression with one independent variable :
#Extract just the median_income column from the independent variables
feature_col=['median_income']
x= house_df[feature_col]
y=house_df['median_house_value']
X_train,X_test,V_train,V_test= train_test_split(x,y,test_size=0.2)

In [48]: #Perform Linear Regression to predict housing values based on median_income.
lm= LinearRegression()
lm.fit(X_train,V_train)
print(lm.coef_)
print(lm.intercept_)

[41740.10981192]
45547.76349347978

In [49]: #Predict output for test dataset using the fitted model.
pred_house_price1 = lm.predict(X_test)
pred_house_price1

Out[49]: array([322514.26215048, 162941.8223395 , 160333.06547626, ...,
242323.16317981, 214553.46812194, 183377.78010342])

In [66]: #Plot the fitted model for training data as well as for test data to check if the fitted model satisfies the test data.
import matplotlib.pyplot as plt

plt.figure(figsize=(10,10))
plt.plot(X_test,pred_house_price1,'r')
plt.scatter(X_train,V_train,color='blue')
plt.title('Training & Test ')
plt.xlabel('Median Income')
plt.ylabel('Median House Value')
```

- Plot the fitted model for training data as well as for test data to check if the fitted model satisfies the test data.

