California Housing Price Prediction

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

Analysis Tasks to be performed:

1. Build a model of housing prices to predict median house values in California using the provided dataset.
2. Train the model to learn from the data to predict the median housing price in any district, given all the other metrics.
3. Predict housing prices based on median_income and plot the regression chart for it.

1. Load the data :

In [316…

```
# Read the housing.csv file from the folder into the program
import numpy as np
import pandas as pd

df_housing=pd.read_excel('./dataset/housing.xlsx')
```

In [317…

```
# Print first few rows of this data

df_housing.head()
```

Out[317…

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	hous
0	-122.23	37.88	41	880	129.0	322	
1	-122.22	37.86	21	7099	1106.0	2401	
2	-122.24	37.85	52	1467	190.0	496	
3	-122.25	37.85	52	1274	235.0	558	
4	-122.25	37.85	52	1627	280.0	565	

In [318…

```
df_housing.shape
```

Out[318…

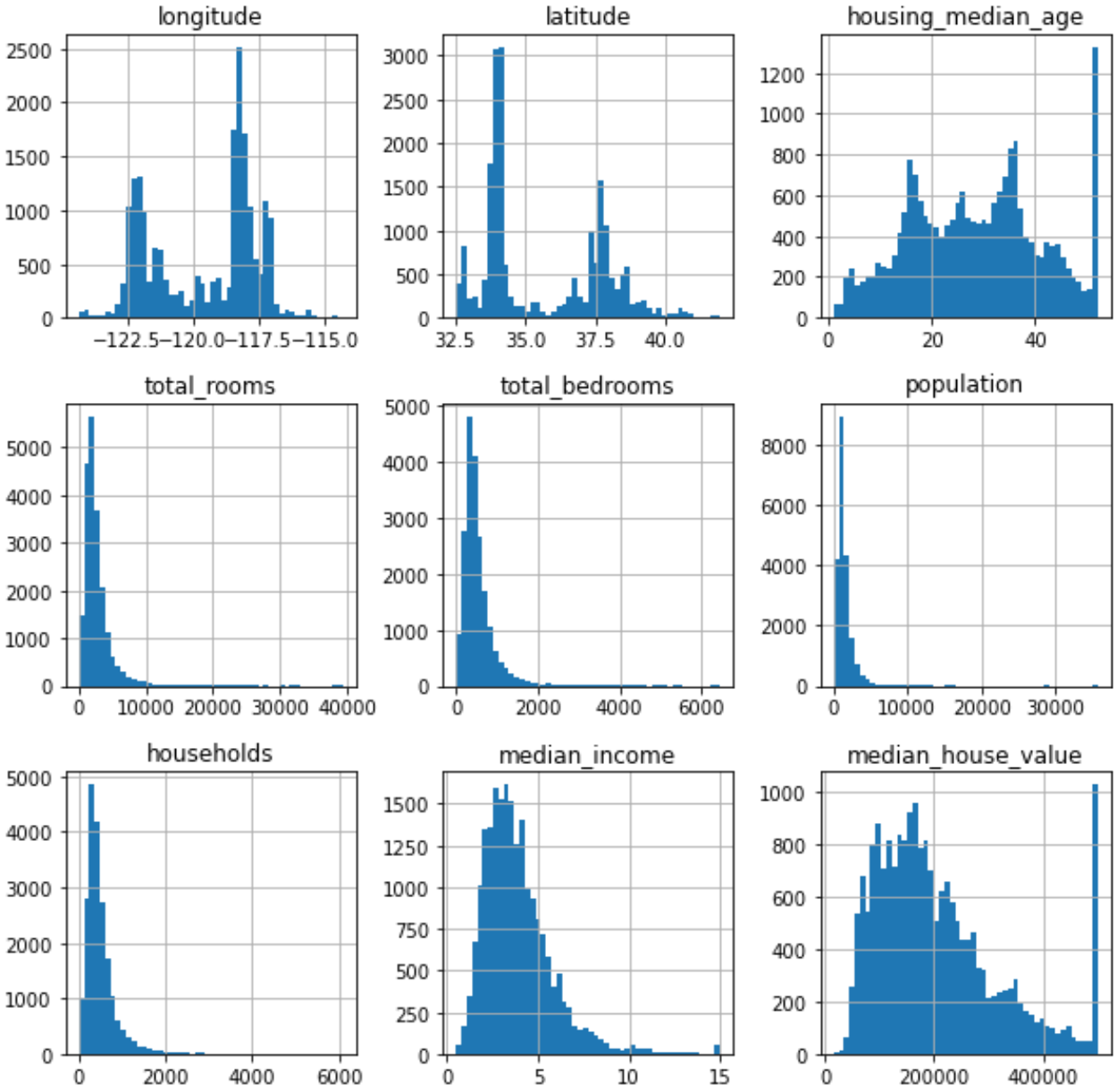
```
(20640, 10)
```

In [319…

```
df_housing.hist(bins=50,figsize=(10,10))
```

Out[319…

```
array([[<AxesSubplot:title={'center':'longitude'}>,
        <AxesSubplot:title={'center':'latitude'}>,
        <AxesSubplot:title={'center':'housing_median_age'}>],
       [<AxesSubplot:title={'center':'total_rooms'}>,
        <AxesSubplot:title={'center':'total_bedrooms'}>,
        <AxesSubplot:title={'center':'population'}>],
       [<AxesSubplot:title={'center':'households'}>,
        <AxesSubplot:title={'center':'median_income'}>,
        <AxesSubplot:title={'center':'median_house_value'}>]],
      dtype=object)
```



In [320…

```
# Extract input (X) and output (Y) data from the dataset

Y=df_housing['median_house_value']
X=df_housing.drop('median_house_value', axis=1)

print(Y.shape)
print(X.shape)
```

```
(20640,)
(20640, 9)
```

Handle missing values

In [321…

```
X.isnull().sum()
```

Out[321…
```
longitude               0
latitude                0
housing_median_age      0
total_rooms             0
total_bedrooms        207
population              0
households              0
median_income           0
ocean_proximity         0
dtype: int64
```

In [322…
```
# Fill the missing values with the mean of the respective column.
from sklearn.impute import SimpleImputer

imp=SimpleImputer(missing_values=np.nan,strategy='mean')
X['total_bedrooms']=imp.fit_transform(X[['total_bedrooms']])
```

In [323…
```
X.isnull().sum()
```

Out[323…
```
longitude             0
latitude              0
housing_median_age    0
total_rooms           0
total_bedrooms        0
population            0
households            0
median_income         0
ocean_proximity       0
dtype: int64
```

Encode categorical data

In [324…
```
# Convert categorical column in the dataset to numerical data.
```

In [325…
```
# find column with non numerical data
X_isreal=X.applymap(np.isreal)

X_isreal.columns[X_isreal.isin([False]).any()]
```

Out[325…
```
Index(['ocean_proximity'], dtype='object')
```

In [326…
```
X['ocean_proximity'].value_counts()
```

Out[326…
```
<1H OCEAN     9136
INLAND        6551
NEAR OCEAN    2658
NEAR BAY      2290
ISLAND           5
Name: ocean_proximity, dtype: int64
```

In [327…
```
# substitute the island with near bay

X['ocean_proximity']=np.where(X['ocean_proximity']=='ISLAND','NEAR BAY',X[
```

In [328…
```
X['ocean_proximity'].value_counts()
```

Out[328…
```
<1H OCEAN     9136
INLAND        6551
NEAR OCEAN    2658
NEAR BAY      2295
Name: ocean_proximity, dtype: int64
```

In [329…
```
# categorical values into dummy var

dum=pd.get_dummies(X['ocean_proximity'])
dum
```

Out[329…

	<1H OCEAN	INLAND	NEAR BAY	NEAR OCEAN
0	0	0	1	0
1	0	0	1	0
2	0	0	1	0
3	0	0	1	0
4	0	0	1	0
...	...	...	...	...
20635	0	1	0	0
20636	0	1	0	0
20637	0	1	0	0
20638	0	1	0	0
20639	0	1	0	0

20640 rows × 4 columns

In [330…
```
X.drop('ocean_proximity',axis=1,inplace=True)
```

In [331…
```
X=pd.concat([X,dum],axis=1)
```

In [332…
```
X.head()
```

Out[332...

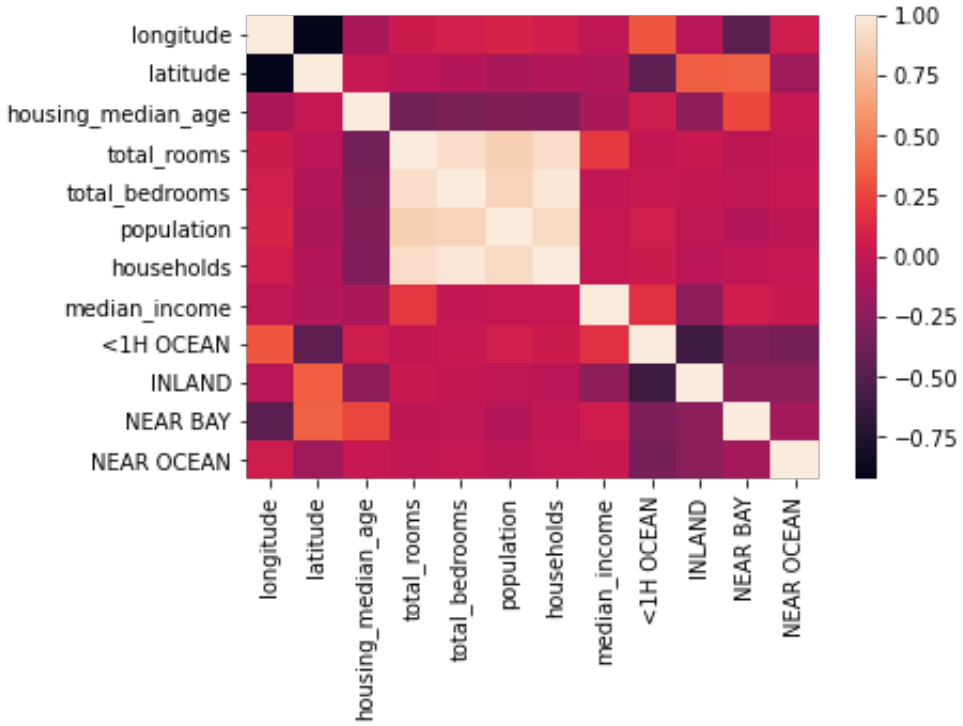
	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	hous
0	-122.23	37.88	41	880	129.0	322	
1	-122.22	37.86	21	7099	1106.0	2401	
2	-122.24	37.85	52	1467	190.0	496	
3	-122.25	37.85	52	1274	235.0	558	
4	-122.25	37.85	52	1627	280.0	565	

In [333...

```
import seaborn as sns

sns.heatmap(X.corr())
```

Out[333...

```
<AxesSubplot:>
```



Split the dataset

In [334...

```
# 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=.2,random_stat

print('train shape', x_train.shape)
print('test shape', x_test.shape)
```

```
train shape (16512, 12)
test shape (4128, 12)
```

Standardize data

In [335…

```
# standardize training and test dataset

from sklearn.preprocessing import StandardScaler

scaler=StandardScaler()
x_train_sc=scaler.fit_transform(x_train)
x_test_sc=scaler.transform(x_test)
```

Perform Linear Regression

In [336…

```
# Perform Linear Regression on training data.

from sklearn.linear_model import LinearRegression

reg=LinearRegression(fit_intercept=True).fit(x_train_sc,y_train)

reg.score(x_test_sc,y_test)
```

Out[336… 0.6375071818640171

In [ ]:

In [337…

```
print('columns',X.columns)
print()
print('coef',reg.coef_)
```

```
columns Index(['longitude', 'latitude', 'housing_median_age', 'total_rooms'
,
       'total_bedrooms', 'population', 'households', 'median_income',
       '<1H OCEAN', 'INLAND', 'NEAR BAY', 'NEAR OCEAN'],
      dtype='object')

coef [-53475.37167331 -54138.55533858  13463.34149805 -11904.01486862
  32981.79863478 -43234.00673208  26951.63483258  74322.84872906
   5833.49711216 -12713.85207835   3201.3265078    5949.44994858]
```

In [338…

```
# Predict output for test dataset using the fitted model.

pred=reg.predict(x_test_sc)
```

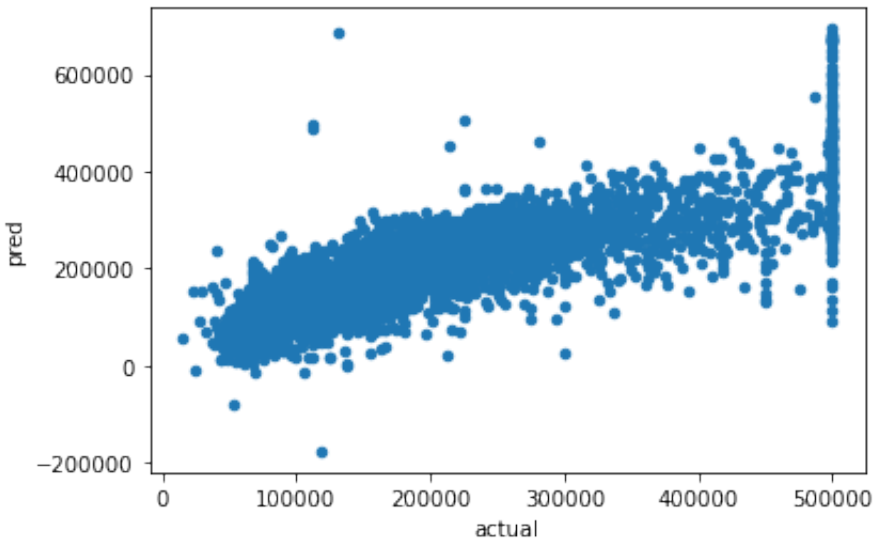
In [339…

```
# Print root mean squared error (RMSE) from Linear Regression.

from sklearn.metrics import mean_squared_error

rmse=mean_squared_error(y_test,pred,squared=True)
print(rmse)
```

```
4754785104.058765
```

In [340…
```
pred_df=pd.DataFrame({'actual':y_test, 'pred':pred})


pred_df.plot(x='actual',y='pred',kind='scatter')
```

Out[340…
```
<AxesSubplot:xlabel='actual', ylabel='pred'>
```



Bonus exercise: Perform Linear Regression with one independent variable

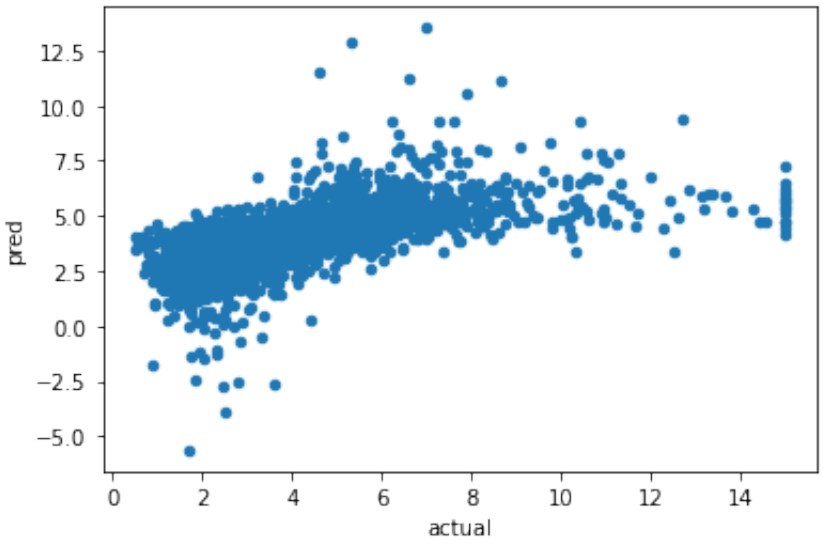
In [307…
```
# Extract just the median_income column from the independent variables (fr

y_train_mi,y_test_mi=x_train['median_income'],x_test['median_income']
x_train_mi,x_test_mi=x_train.drop('median_income',axis=1),x_test.drop('medi
```

In [308…
```
# Perform Linear Regression to predict housing values based on median_inco

scaler_mi=StandardScaler()
x_train_mi_sc=scaler_mi.fit_transform(x_train_mi)
x_test_mi_sc=scaler_mi.transform(x_test_mi)

reg=LinearRegression(fit_intercept=True).fit(x_train_mi_sc,y_train_mi)
```

In [309…
```
# Predict output for test dataset using the fitted model.

pred_mi=reg.predict(x_test_mi_sc)
```

In [310…
```
# Plot the fitted model for training data as well as for test data to chec

pred_mi_df=pd.DataFrame({'actual':y_test_mi, 'pred':pred_mi})


pred_mi_df.plot(x='actual',y='pred',kind='scatter')
```

Out[310… <AxesSubplot:xlabel='actual', ylabel='pred'>



In [341…

```
rmse_mib=mean_squared_error(y_test_mi,pred_mi,squared=True)
print(rmse_mib)
```

```
2.1642383976000295
```

In [ ]: