```python
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# Importing California Housing Prices Dataset

from pathlib import Path
import tarfile
import urllib.request

def load_housing_data():
    tarball_path = Path("datasets/housing.tgz")
    if not tarball_path.is_file():
        Path("datasets").mkdir(parents=True, exist_ok=True)
        url = "https://github.com/ageron/data/raw/main/housing.tgz"
        urllib.request.urlretrieve(url, tarball_path)
        with tarfile.open(tarball_path) as housing_tarball:
            housing_tarball.extractall(path="datasets")
    return pd.read_csv(Path("datasets/housing/housing.csv"))

housing = load_housing_data()

housing
```

```
       longitude  latitude  housing_median_age  total_rooms  
total_bedrooms  \
0        -122.23     37.88                41.0        880.0   
129.0   
1        -122.22     37.86                21.0       7099.0   
1106.0   
2        -122.24     37.85                52.0       1467.0   
190.0   
3        -122.25     37.85                52.0       1274.0   
235.0   
4        -122.25     37.85                52.0       1627.0   
280.0   
...          ...       ...                 ...          ...   
...   
20635    -121.09     39.48                25.0       1665.0   
374.0   
20636    -121.21     39.49                18.0        697.0   
150.0   
20637    -121.22     39.43                17.0       2254.0   
485.0   
20638    -121.32     39.43                18.0       1860.0   
409.0   
20639    -121.24     39.37                16.0       2785.0   
616.0   
```

```
       population  households  median_income  median_house_value  \
0           322.0       126.0         8.3252            452600.0
1          2401.0      1138.0         8.3014            358500.0
2           496.0       177.0         7.2574            352100.0
3           558.0       219.0         5.6431            341300.0
4           565.0       259.0         3.8462            342200.0
...           ...         ...            ...                 ...
20635       845.0       330.0         1.5603             78100.0
20636       356.0       114.0         2.5568             77100.0
20637      1007.0       433.0         1.7000             92300.0
20638       741.0       349.0         1.8672             84700.0
20639      1387.0       530.0         2.3886             89400.0

      ocean_proximity
0            NEAR BAY
1            NEAR BAY
2            NEAR BAY
3            NEAR BAY
4            NEAR BAY
...               ...
20635          INLAND
20636          INLAND
20637          INLAND
20638          INLAND
20639          INLAND

[20640 rows x 10 columns]

                                                        Data
Preprocessing

df=housing.copy()

housing.head()

   longitude  latitude  housing_median_age  total_rooms
total_bedrooms  \
0    -122.23     37.88                41.0        880.0
129.0
1    -122.22     37.86                21.0       7099.0
1106.0
2    -122.24     37.85                52.0       1467.0
190.0
3    -122.25     37.85                52.0       1274.0
235.0
4    -122.25     37.85                52.0       1627.0
280.0

   population  households  median_income  median_house_value
ocean_proximity
```

```
0       322.0       126.0         8.3252            452600.0
NEAR BAY
1      2401.0      1138.0         8.3014            358500.0
NEAR BAY
2       496.0       177.0         7.2574            352100.0
NEAR BAY
3       558.0       219.0         5.6431            341300.0
NEAR BAY
4       565.0       259.0         3.8462            342200.0
NEAR BAY
```

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

```
longitude               0
latitude                0
housing_median_age      0
total_rooms             0
total_bedrooms        207
population              0
households              0
median_income           0
median_house_value      0
ocean_proximity         0
dtype: int64
```

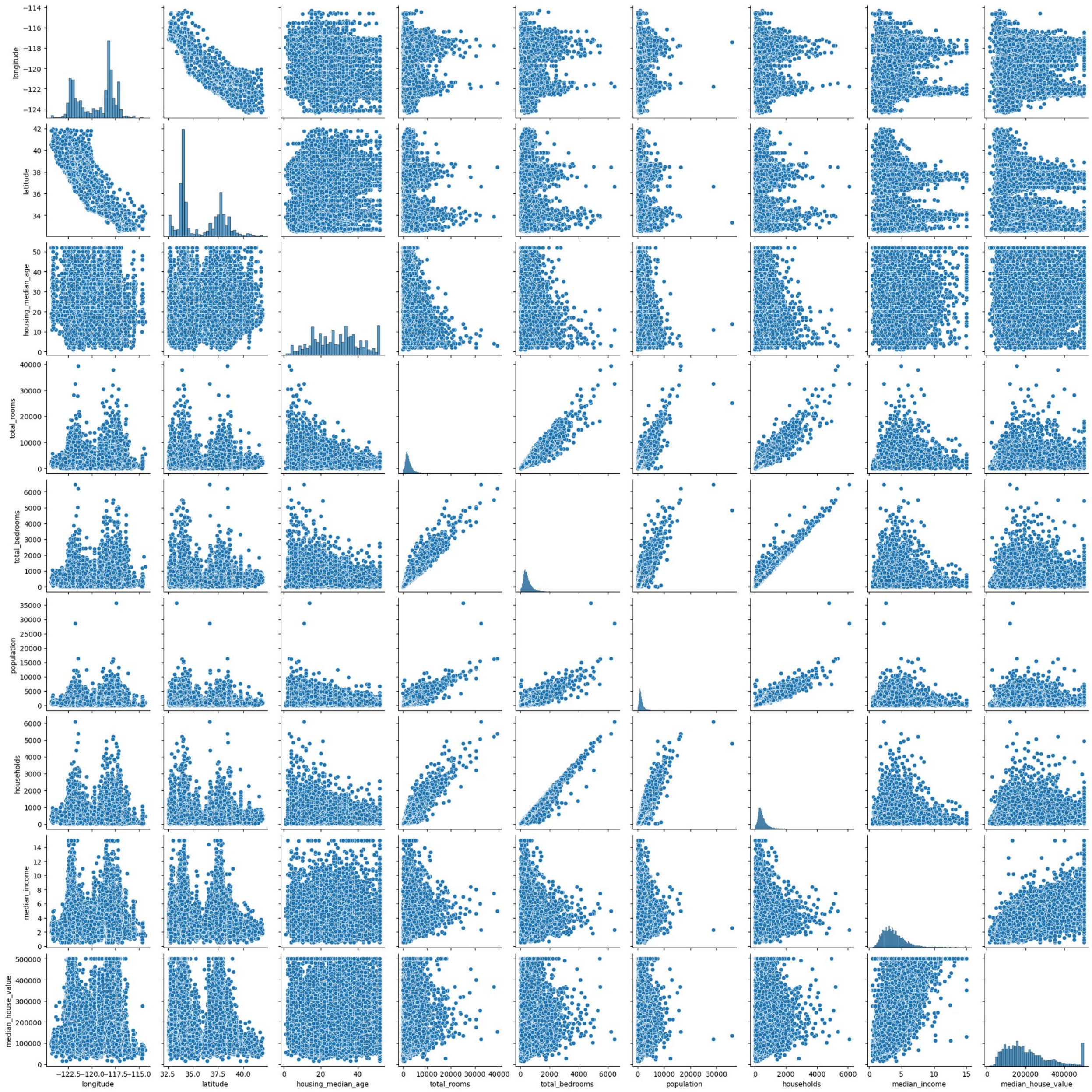
total_bedrooms contains 207 missing value we have to impute them before training machine learning model

```
housing.describe()
```

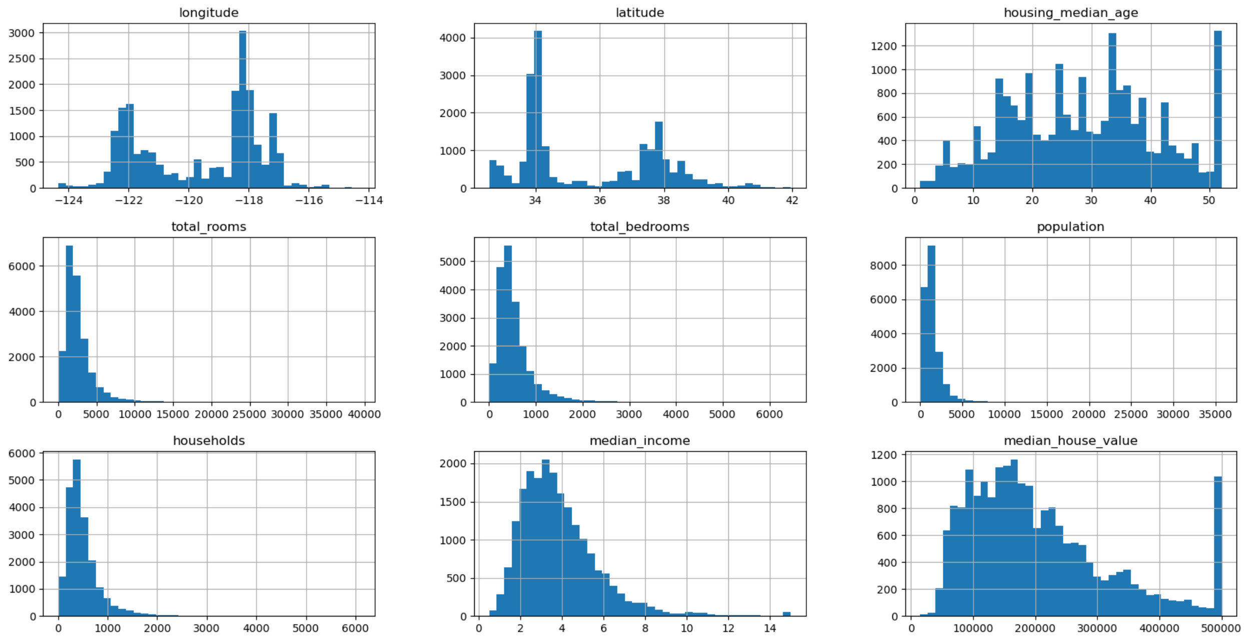
```
          longitude      latitude  housing_median_age   total_rooms  \
count  20640.000000  20640.000000        20640.000000  20640.000000
mean    -119.569704     35.631861           28.639486   2635.763081
std        2.003532      2.135952           12.585558   2181.615252
min     -124.350000     32.540000            1.000000      2.000000
25%     -121.800000     33.930000           18.000000   1447.750000
50%     -118.490000     34.260000           29.000000   2127.000000
75%     -118.010000     37.710000           37.000000   3148.000000
max     -114.310000     41.950000           52.000000  39320.000000

       total_bedrooms    population    households  median_income  \
count    20433.000000  20640.000000  20640.000000   20640.000000
mean       537.870553   1425.476744    499.539680       3.870671
std        421.385070   1132.462122    382.329753       1.899822
min          1.000000      3.000000      1.000000       0.499900
25%        296.000000    787.000000    280.000000       2.563400
50%        435.000000   1166.000000    409.000000       3.534800
75%        647.000000   1725.000000    605.000000       4.743250
max       6445.000000  35682.000000   6082.000000      15.000100

       median_house_value
count        20640.000000
mean        206855.816909
std         115395.615874
min          14999.000000
25%         119600.000000
50%         179700.000000
75%         264725.000000
max         500001.000000
```

```
sns.pairplot(housing)
```

```
<seaborn.axisgrid.PairGrid at 0x14e40eed0>
```



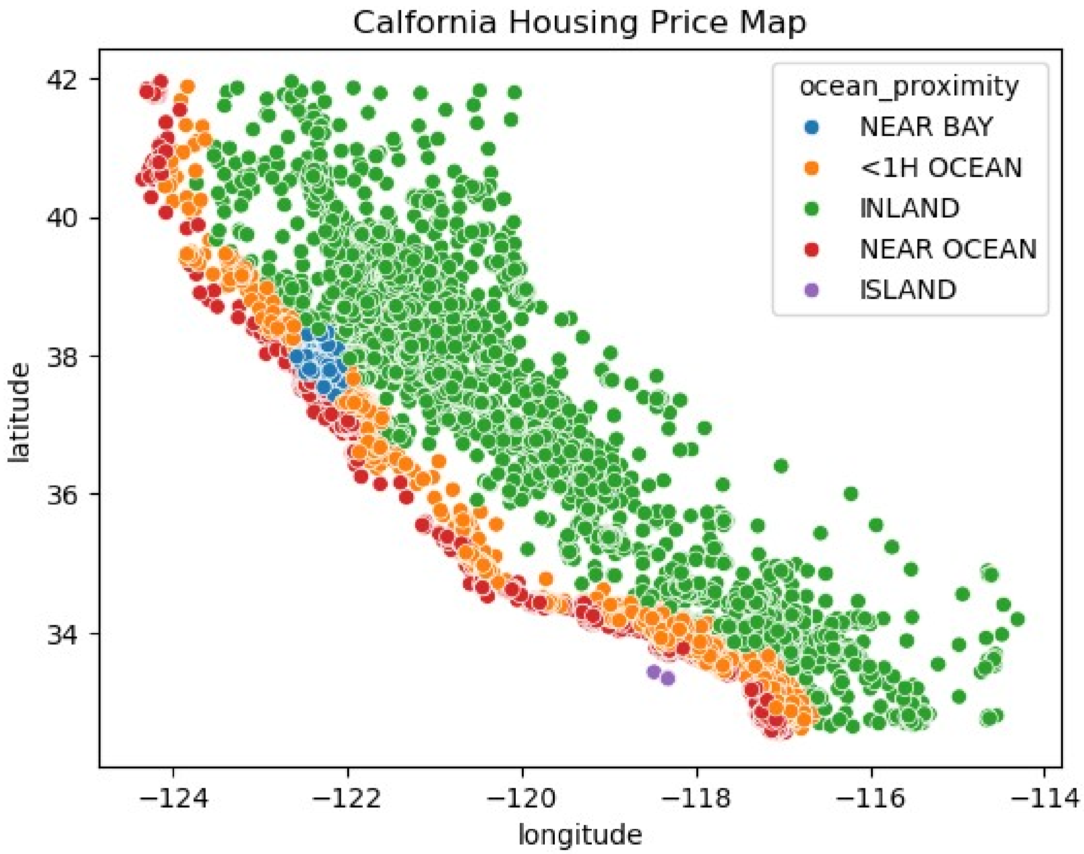

```
%matplotlib inline
housing.hist(bins=40,figsize=(20,10))
plt.show()
```



total_rooms,total_bedrooms and household are right tailed data to make them normal distributed we can apply natural log transformation or log base 10 transformation on them

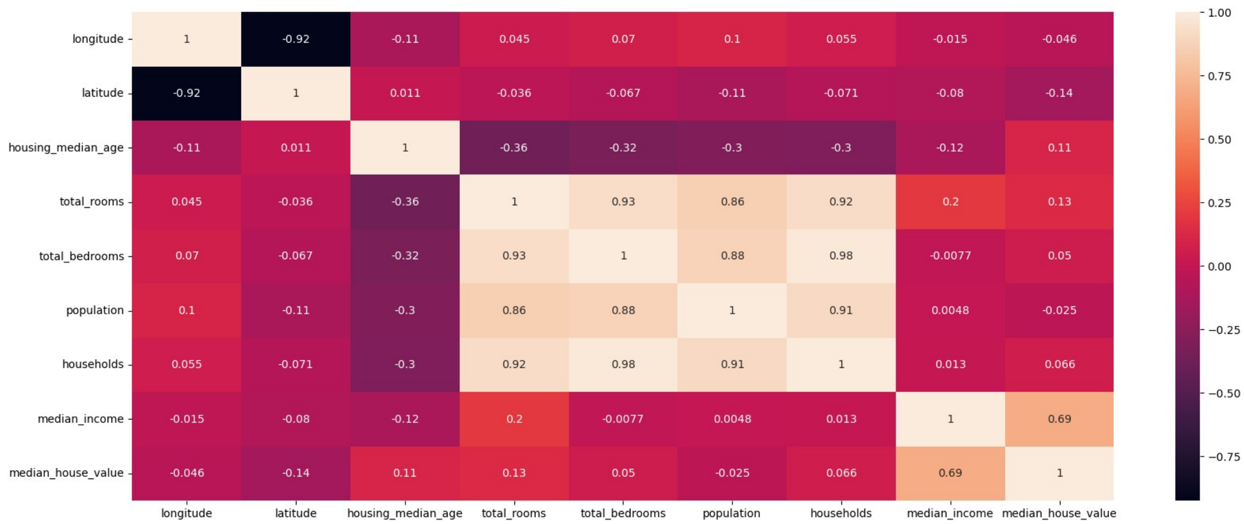
median_house_age has almost normal distribution and normal_house_value is little bit right tailed where we can apply natural log to make it normal distributed

```
# let's see how actual calfornia dataset looks like
sns.scatterplot(x=housing.longitude,y=housing.latitude,hue=housing['ocean_proximity'])
plt.title('Calfornia Housing Price Map')
plt.show()
```




```
# checking the correlation between different features
corr=housing.drop(columns=['ocean_proximity']).corr()
corr

                    longitude  latitude  housing_median_age
total_rooms  \
longitude            1.000000 -0.924664           -0.108197
0.044568
latitude            -0.924664  1.000000            0.011173    -
0.036100
housing_median_age  -0.108197  0.011173            1.000000    -
0.361262
total_rooms          0.044568 -0.036100           -0.361262
1.000000
total_bedrooms       0.069608 -0.066983           -0.320451
0.930380
population           0.099773 -0.108785           -0.296244
0.857126
households           0.055310 -0.071035           -0.302916
0.918484
median_income       -0.015176 -0.079809           -0.119034
0.198050
median_house_value  -0.045967 -0.144160            0.105623
```

```
0.134153

                    total_bedrooms  population  households  
median_income  \
longitude                 0.069608    0.099773    0.055310       -
0.015176
latitude                 -0.066983   -0.108785   -0.071035       -
0.079809
housing_median_age       -0.320451   -0.296244   -0.302916       -
0.119034
total_rooms               0.930380    0.857126    0.918484       
0.198050
total_bedrooms            1.000000    0.877747    0.979728       -
0.007723
population                0.877747    1.000000    0.907222       
0.004834
households                0.979728    0.907222    1.000000       
0.013033
median_income            -0.007723    0.004834    0.013033       
1.000000
median_house_value        0.049686   -0.024650    0.065843       
0.688075

                    median_house_value
longitude                    -0.045967
latitude                     -0.144160
housing_median_age            0.105623
total_rooms                   0.134153
total_bedrooms                0.049686
population                   -0.024650
households                    0.065843
median_income                 0.688075
median_house_value            1.000000

plt.figure(figsize=(20,8))
sns.heatmap(corr,annot=True)
plt.show()
```



From above visualization we can conclude that some features are heighly correlated with each other which cause multicollinarity and as we know some machine learning algorithms doesn't work well with multicollinarity.

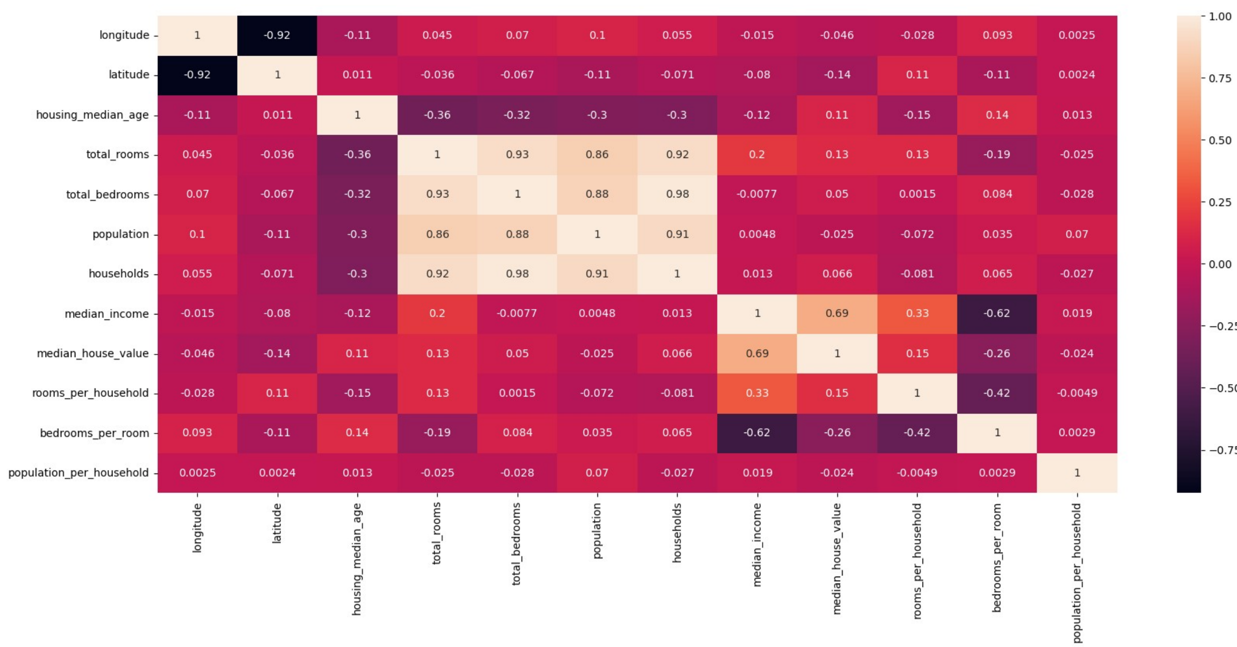
```
# To get rid of multicollinearity we try some new features in our
dataset

housing["rooms_per_household"] =
housing["total_rooms"]/housing["households"]
housing["bedrooms_per_room"] =
housing["total_bedrooms"]/housing["total_rooms"]
housing["population_per_household"]=housing["population"]/housing["hou
seholds"]

corr1=housing.drop(columns=['ocean_proximity']).corr()
corr1
```

```
                          longitude  latitude  housing_median_age  \
longitude                  1.000000 -0.924664           -0.108197
latitude                  -0.924664  1.000000            0.011173
housing_median_age        -0.108197  0.011173            1.000000
total_rooms                0.044568 -0.036100           -0.361262
total_bedrooms             0.069608 -0.066983           -0.320451
population                 0.099773 -0.108785           -0.296244
households                 0.055310 -0.071035           -0.302916
median_income             -0.015176 -0.079809           -0.119034
median_house_value        -0.045967 -0.144160            0.105623
rooms_per_household       -0.027540  0.106389           -0.153277
bedrooms_per_room          0.092657 -0.113815            0.136089
population_per_household   0.002476  0.002366            0.013191

                          total_rooms  total_bedrooms  population
```

```
households  \
longitude                     0.044568        0.069608    0.099773
0.055310
latitude                     -0.036100       -0.066983   -0.108785   -
0.071035
housing_median_age           -0.361262       -0.320451   -0.296244   -
0.302916
total_rooms                   1.000000        0.930380    0.857126
0.918484
total_bedrooms                0.930380        1.000000    0.877747
0.979728
population                    0.857126        0.877747    1.000000
0.907222
households                    0.918484        0.979728    0.907222
1.000000
median_income                 0.198050       -0.007723    0.004834
0.013033
median_house_value            0.134153        0.049686   -0.024650
0.065843
rooms_per_household           0.133798        0.001538   -0.072213   -
0.080598
bedrooms_per_room            -0.187900        0.084238    0.035319
0.065087
population_per_household     -0.024581       -0.028355    0.069863   -
0.027309

                          median_income  median_house_value  \
longitude                     -0.015176           -0.045967
latitude                      -0.079809           -0.144160
housing_median_age            -0.119034            0.105623
total_rooms                    0.198050            0.134153
total_bedrooms                -0.007723            0.049686
population                     0.004834           -0.024650
households                     0.013033            0.065843
median_income                  1.000000            0.688075
median_house_value             0.688075            1.000000
rooms_per_household            0.326895            0.151948
bedrooms_per_room             -0.615661           -0.255880
population_per_household       0.018766           -0.023737

                          rooms_per_household  bedrooms_per_room  \
longitude                           -0.027540           0.092657
latitude                             0.106389          -0.113815
housing_median_age                  -0.153277           0.136089
total_rooms                          0.133798          -0.187900
total_bedrooms                       0.001538           0.084238
population                          -0.072213           0.035319
households                          -0.080598           0.065087
median_income                        0.326895          -0.615661
```

```
median_house_value                     0.151948           -0.255880
rooms_per_household                    1.000000           -0.416952
bedrooms_per_room                     -0.416952            1.000000
population_per_household              -0.004852            0.002938

                          population_per_household
longitude                                 0.002476
latitude                                  0.002366
housing_median_age                        0.013191
total_rooms                              -0.024581
total_bedrooms                           -0.028355
population                                0.069863
households                               -0.027309
median_income                             0.018766
median_house_value                       -0.023737
rooms_per_household                      -0.004852
bedrooms_per_room                         0.002938
population_per_household                  1.000000
```

```python
plt.figure(figsize=(20,8))
sns.heatmap(corr1,annot=True)
plt.show()
```



Now we can drop these columns from housing and Let's go to next step where we make our dataset fully ready to feed into machine learning algorithm

```python
housing.drop(columns=['total_rooms','total_bedrooms','population','households'],inplace=True)

housing.head()
```

```
   longitude  latitude  housing_median_age  median_income
median_house_value  \
0    -122.23     37.88                41.0         8.3252
452600.0
1    -122.22     37.86                21.0         8.3014
358500.0
2    -122.24     37.85                52.0         7.2574
352100.0
3    -122.25     37.85                52.0         5.6431
341300.0
4    -122.25     37.85                52.0         3.8462
342200.0

  ocean_proximity  rooms_per_household  bedrooms_per_room  \
0        NEAR BAY             6.984127           0.146591
1        NEAR BAY             6.238137           0.155797
2        NEAR BAY             8.288136           0.129516
3        NEAR BAY             5.817352           0.184458
4        NEAR BAY             6.281853           0.172096

   population_per_household
0                  2.555556
1                  2.109842
2                  2.802260
3                  2.547945
4                  2.181467
```

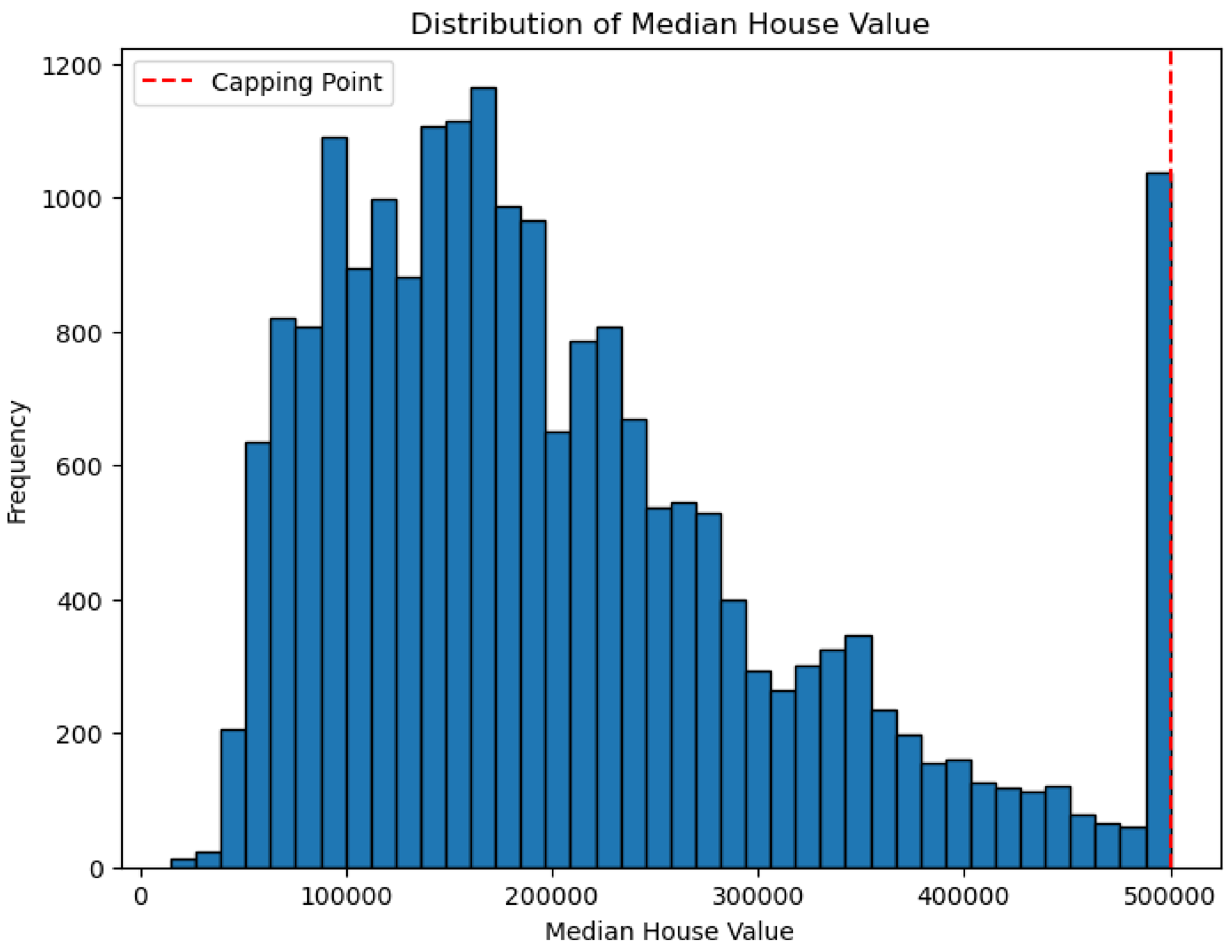
```
housing[housing['median_house_value']>=500000]
```

```
       longitude  latitude  housing_median_age  median_income  \
89       -122.27     37.80                52.0         1.2434
103      -118.47     33.99                24.0         2.9750
105      -118.50     33.97                29.0         5.1280
107      -118.39     34.08                27.0         3.8088
132      -122.34     37.55                44.0         6.9533
...          ...       ...                 ...            ...
20494    -118.12     34.13                52.0        11.7060
20500    -121.93     37.66                24.0         8.3337
20511    -122.05     37.31                25.0         9.2298
20515    -117.36     33.17                24.0         2.3182
20535    -122.43     37.80                52.0         4.5399

       median_house_value ocean_proximity  rooms_per_household  \
89               500001.0        NEAR BAY             2.929412
103              500001.0       <1H OCEAN             3.456731
105              500001.0       <1H OCEAN             3.932471
107              500001.0       <1H OCEAN             4.345395
132              500001.0      NEAR OCEAN             7.608025
...                   ...             ...                  ...
20494            500001.0       <1H OCEAN             8.975535
```

```
20500          500001.0      <1H OCEAN            7.915000
20511          500001.0      <1H OCEAN            7.237676
20515          500001.0     NEAR OCEAN            5.574932
20535          500001.0       NEAR BAY            4.898601

       bedrooms_per_room  population_per_household
89              0.313253                  4.658824
103             0.315716                  1.598558
105             0.295214                  1.662356
107             0.258895                  1.753289
132             0.133063                  2.601852
...                  ...                       ...
20494           0.116184                  2.981651
20500           0.133923                  2.702500
20511           0.130868                  2.790493
20515           0.216031                  2.212534
20535           0.221984                  1.667832

[992 rows x 9 columns]
```

```
capped_values = housing[housing['median_house_value'] >= 500001]
print(f"Capped values: {len(capped_values)} out of {len(housing)}
({len(capped_values)/len(housing)*100:.2f}%)")


plt.figure(figsize=(8, 6))
plt.hist(housing['median_house_value'], bins=40, edgecolor='black')
plt.axvline(x=500001, color='r', linestyle='--', label='Capping
Point')
plt.xlabel('Median House Value')
plt.ylabel('Frequency')
plt.title('Distribution of Median House Value')
plt.legend()
plt.show()
```

```
Capped values: 965 out of 20640 (4.68%)
```




```
housing.shape
```

```
(20640, 9)
```

```
housing=housing.sample(20640)

# If you are curious to know why i do this let me explain, I do this
to make data more random so during train test spilt data spreed
properly into x_train,x_test

housing
```

```
       longitude  latitude  housing_median_age  median_income  \
3478     -118.16     33.88                30.0         2.9779
18741    -122.67     38.43                17.0         3.2813
15576    -118.39     34.23                43.0         2.1518
1398     -121.64     36.66                24.0         5.2285
1363     -117.00     32.67                16.0         6.6143
...          ...       ...                 ...            ...
1613     -118.33     33.96                42.0         2.3000
19104    -117.06     32.76                38.0         3.2188
```

```
11074    -120.67     38.76                35.0         2.1682
10334    -119.00     35.39                51.0         2.8295
1244     -122.33     37.55                51.0         9.3694

       median_house_value ocean_proximity  rooms_per_household  \
3478             169500.0       <1H OCEAN             4.422977
18741            202700.0       <1H OCEAN             4.980149
15576            161600.0       <1H OCEAN             3.728125
1398             248100.0       <1H OCEAN             5.932710
1363             264100.0      NEAR OCEAN             7.386139
...                   ...             ...                  ...
1613             189200.0       <1H OCEAN             5.285266
19104            150500.0      NEAR OCEAN             5.571942
11074            138100.0          INLAND             5.260000
10334             72100.0          INLAND             4.576667
1244             500001.0      NEAR OCEAN             8.300971

       bedrooms_per_room  population_per_household
3478            0.234947                  3.083551
18741           0.199302                  2.220844
15576           0.250629                  3.700000
1398            0.159420                  2.740187
1363            0.137176                  3.306931
...                  ...                       ...
1613            0.214116                  2.310345
19104           0.185926                  2.287770
11074           0.191540                  2.650000
10334           0.206846                  2.160000
1244            0.129435                  2.815534

[20640 rows x 9 columns]
```

```
housing.describe()
```

```
          longitude      latitude  housing_median_age
median_income  \
count  20640.000000  20640.000000        20640.000000   20640.000000

mean    -119.569704     35.631861           28.639486       3.870671

std        2.003532      2.135952           12.585558       1.899822

min     -124.350000     32.540000            1.000000       0.499900

25%     -121.800000     33.930000           18.000000       2.563400

50%     -118.490000     34.260000           29.000000       3.534800

75%     -118.010000     37.710000           37.000000       4.743250

max     -114.310000     41.950000           52.000000      15.000100
```

```
       median_house_value  rooms_per_household  bedrooms_per_room  \
count        20640.000000         20640.000000       20433.000000
mean        206855.816909             5.429000           0.213039
std         115395.615874             2.474173           0.057983
min          14999.000000             0.846154           0.100000
25%         119600.000000             4.440716           0.175427
50%         179700.000000             5.229129           0.203162
75%         264725.000000             6.052381           0.239821
max         500001.000000           141.909091           1.000000

       population_per_household
count              20640.000000
mean                   3.070655
std                   10.386050
min                    0.692308
25%                    2.429741
50%                    2.818116
75%                    3.282261
max                 1243.333333
```

```python
from sklearn.model_selection import train_test_split
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import FunctionTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import LabelBinarizer
from sklearn.pipeline import FeatureUnion

x_train,x_test,y_train,y_test=train_test_split(housing.drop(columns=['
median_house_value']),housing['median_house_value'],test_size=0.20,ran
dom_state=42)

x_train.head()
```

```
       longitude  latitude  housing_median_age  median_income
ocean_proximity  \
16892    -118.29     33.90                27.0         1.7714
<1H OCEAN
8306     -122.31     38.31                32.0         3.8796
NEAR BAY
5385     -119.29     34.26                32.0         3.6007
NEAR OCEAN
15958    -118.11     33.83                36.0         4.2703
<1H OCEAN
7781     -117.11     32.67                52.0         1.4844
```

```
NEAR OCEAN

       rooms_per_household  bedrooms_per_room
population_per_household
16892             2.532500           0.388944
2.667500
8306              5.765101           0.177726
2.621924
5385              5.491667           0.231866
2.240000
15958             5.966555           0.169843
3.224080
7781              3.943662           0.253571
3.056338
```

```
x_test.head()
```

```
       longitude  latitude  housing_median_age  median_income
ocean_proximity  \
11890    -117.89     34.07                35.0         3.9808
<1H OCEAN
14814    -121.95     37.94                21.0         6.8642
INLAND
11959    -117.40     33.95                32.0         2.4408
INLAND
4180     -118.08     33.76                27.0         2.0952
<1H OCEAN
1915     -117.37     34.12                32.0         3.8398
INLAND

       rooms_per_household  bedrooms_per_room
population_per_household
11890             5.309963           0.181376
2.966790
14814             7.315545           0.130352
3.058005
11959             4.457207           0.248105
2.148649
4180              3.412903           0.300567
1.245161
1915              6.230469           0.178056
3.152344
```

```
x_test['ocean_proximity'].value_counts()
```

```
ocean_proximity
<1H OCEAN     1878
INLAND        1296
NEAR OCEAN     514
NEAR BAY       438
```

```
ISLAND           2
Name: count, dtype: int64
```

```
num_attribute=list(housing.drop(columns=['ocean_proximity','median_hou
se_value']))

num_attribute
```

```
['longitude',
 'latitude',
 'housing_median_age',
 'median_income',
 'rooms_per_household',
 'bedrooms_per_room',
 'population_per_household']
```

```
cat_attribute=['ocean_proximity']


pipeline1=Pipeline(steps=[
    ('imputer',SimpleImputer(strategy='median')),
    ('Scaler',StandardScaler())

],verbose=True)


pipeline2=Pipeline(steps=[
    ('Label
Encoder',OneHotEncoder(sparse_output=False,drop='first',handle_unknown
='ignore'))
])

preprocessor=ColumnTransformer(transformers=[
    ('Pipeline 1 For Numerical Columns ',pipeline1,num_attribute),
    ('Encoder For Cateogrical Columns',pipeline2,cat_attribute)
])

preprocessor
```

```
ColumnTransformer(transformers=[('Pipeline 1 For Numerical Columns ',
                                 Pipeline(steps=[('imputer',

SimpleImputer(strategy='median')),
                                                 ('Scaler',
StandardScaler())],
                                          verbose=True),
                                 ['longitude', 'latitude',
'housing_median_age',
                                  'median_income',
```

```
'rooms_per_household',
                                  'bedrooms_per_room',
                                  'population_per_household']),
                                ('Encoder For Cateogrical Columns',
                                 Pipeline(steps=[('Label Encoder',

OneHotEncoder(drop='first',

handle_unknown='ignore',

sparse_output=False))]),
                                 ['ocean_proximity'])])
```

```
x_train.shape
```

```
(16512, 8)
```

```
x_train_transformed=preprocessor.fit_transform(x_train)
```

```
[Pipeline] ........... (step 1 of 2) Processing imputer, total=   0.0s
[Pipeline] ............ (step 2 of 2) Processing Scaler, total=   0.0s
```

```
x_train_transformed.shape
```

```
(16512, 11)
```

```
x_test.shape
```

```
(4128, 8)
```

```
x_test_transformed=preprocessor.fit_transform(x_test)
```

```
[Pipeline] ........... (step 1 of 2) Processing imputer, total=   0.0s
[Pipeline] ............ (step 2 of 2) Processing Scaler, total=   0.0s
```

```
x_test_transformed.shape
```

```
(4128, 11)
```

```
x_test['ocean_proximity'].value_counts()
```

```
ocean_proximity
<1H OCEAN     1878
INLAND        1296
NEAR OCEAN     514
NEAR BAY       438
ISLAND           2
Name: count, dtype: int64
```

```
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
```

```
from sklearn.svm import SVR
from sklearn.metrics import
accuracy_score,mean_absolute_error,mean_squared_error,mean_absolute_pe
rcentage_error,root_mean_squared_error
from sklearn.model_selection import cross_val_score
from sklearn.metrics import r2_score, explained_variance_score
```

```
                                                First Model as Linear
Regression
```

```
lr=LinearRegression()
lr.fit(x_train_transformed,y_train)
```

```
LinearRegression()
```

```
y_pred_lr=lr.predict(x_test_transformed)
```

```
print('Percentage
Error',mean_absolute_percentage_error(y_test,y_pred_lr))
print('Mean Square Error',mean_squared_error(y_test,y_pred_lr))
print('RMSE',root_mean_squared_error(y_test,y_pred_lr))
print('R2 Score',r2_score(y_test,y_pred_lr))
print('Variance Score',explained_variance_score(y_test,y_pred_lr))
```

```
Percentage Error 0.3005917558525927
Mean Square Error 5201881955.40867
RMSE 72124.0733417676
R2 Score 0.6094009740883701
Variance Score 0.6094055604744889
```

```
cv_score_lr=cross_val_score(lr,x_train_transformed,y_train,cv=10,scori
ng='neg_mean_squared_error')
```

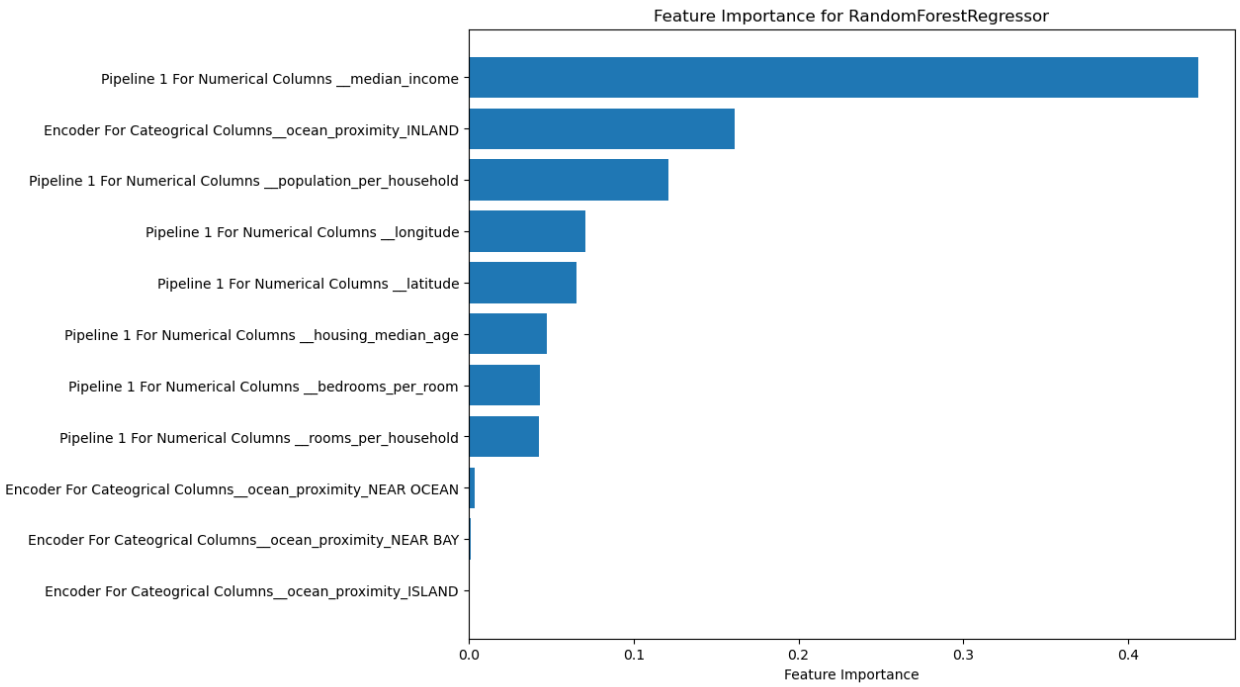
```
rmse_lr=np.sqrt(-cv_score_lr)
print('CV Scores',rmse_lr)
print('RMSE',rmse_lr.mean())
```

```
CV Scores [69241.34483476 69685.83662162 70012.47076207 69417.97070133
 73225.12136675 74577.41502072 70239.71996077 71187.93527516
 69418.59489156 70293.43849952]
RMSE 70729.98479342667
```

Conclusion : LinearRegression Model doesn't perform well it goes Underfitting and it fail to learn the pattern cause it give us 27% of mean_absolute_percentage_error and RMSE as 70804 after doing 10 Cross validation and its r2 score is 0.63.

```
                                      Second Model as
RandomForestRegressor
```

```
rfr=RandomForestRegressor(n_estimators=100,max_features=8)

rfr.fit(x_train_transformed,y_train)

RandomForestRegressor(max_features=8)

y_pred_rfr=rfr.predict(x_test_transformed)

print('Percentage
Error',mean_absolute_percentage_error(y_test,y_pred_rfr))
print('Mean Square Error',mean_squared_error(y_test,y_pred_rfr))
print('RMSE',root_mean_squared_error(y_test,y_pred_rfr))
print('R2 Score',r2_score(y_test,y_pred_rfr))
print('Variance Score',explained_variance_score(y_test,y_pred_rfr))

Percentage Error 0.18226256099572993
Mean Square Error 2495179025.639212
RMSE 49951.76699216167
R2 Score 0.8126419428113997
Variance Score 0.8135118348123949

# This line of Code take some time to run

cv_score_rfr=cross_val_score(rfr,x_train_transformed,y_train,cv=10,sco
ring='neg_mean_squared_error')

rmse_rfr=np.sqrt(-cv_score_rfr)
print('CV Scores',rmse_rfr)
print('RMSE',rmse_rfr.mean())

CV Scores [46038.38201923 48815.91194992 46998.38371322 49356.48951925
 51874.76664592 49718.64121897 50531.29819724 48803.80872544
 47660.84540778 47217.64201541]
RMSE 48701.61694123798

importances = rfr.feature_importances_
feature_names = preprocessor.get_feature_names_out()
sorted_indices = importances.argsort()

plt.figure(figsize=(10, 8))
plt.barh(feature_names[sorted_indices], importances[sorted_indices])
plt.xlabel('Feature Importance')
plt.title('Feature Importance for RandomForestRegressor')
plt.show()
```



Random forest Regressor perform well than Linear regression and it give us RMSE of 48094 which is less than of Linear regression.we confirm it by doing cross validation and well R2 score is 0.81 but explained_variance_score is 0.81

After doing Cross validation we can confirm that RandomForesetRegressor perform better than Linear regressor

```
                                             Thrid Model as SVM
```

```
svr=SVR(kernel='linear')

svr.fit(x_train_transformed,y_train)

SVR(kernel='linear')

y_pred_svr=svr.predict(x_test_transformed)

print('Percentage
Error',mean_absolute_percentage_error(y_test,y_pred_svr))
print('Mean Square Error',mean_squared_error(y_test,y_pred_svr))
print('RMSE',root_mean_squared_error(y_test,y_pred_svr))
print('R2 Score',r2_score(y_test,y_pred_svr))
print('Variance Score',explained_variance_score(y_test,y_pred_svr))

Percentage Error 0.4935222025503083
Mean Square Error 12384196095.115314
RMSE 111284.30300413133
```

```
R2 Score 0.07009521305628585
Variance Score 0.12391550820848884
```

SVM perfrom more worst than Linear Regressor

```
                                                Fourth Model as
XGboost

from xgboost import XGBRegressor
xgb=XGBRegressor()
xgb.fit(x_train_transformed,y_train)

XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None,
early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None,
feature_types=None,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None,
max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=None, max_leaves=None,
             min_child_weight=None, missing=nan,
monotone_constraints=None,
             multi_strategy=None, n_estimators=None, n_jobs=None,
             num_parallel_tree=None, random_state=None, ...)

xgb_pred=xgb.predict(x_test_transformed)
xgb_pred

array([208091.19, 277352.66, 102060.79, ..., 164733.56, 407450.44,
       336372.7 ], dtype=float32)


print('Percentage
Error',mean_absolute_percentage_error(y_test,xgb_pred))
print('Mean Square Error',mean_squared_error(y_test,xgb_pred))
print('RMSE',root_mean_squared_error(y_test,xgb_pred))
print('R2 Score',r2_score(y_test,xgb_pred))
print('Variance Score',explained_variance_score(y_test,xgb_pred))

Percentage Error 0.17544368021484072
Mean Square Error 2332399337.424746
RMSE 48294.92041017095
R2 Score 0.824864747596245
Variance Score 0.8251776029971587

cv_score_xgb=cross_val_score(xgb,x_train_transformed,y_train,cv=10,sco
ring='neg_mean_squared_error')
```

```
rmse_xgb=np.sqrt(-cv_score_xgb)
print('CV Scores',rmse_xgb)
print('RMSE',rmse_xgb.mean())
```

```
CV Scores [43575.99177336 46011.69672152 45598.91499307 46970.51804133
 49974.68446794 46303.37663823 50087.88041028 46612.5573549
 44922.44872763 44134.51657066]
RMSE 46419.25856989343
```

Xgboost give us better result than any other algorithms which we use before even it is better than RandomForest Regressor.Xgboost give almost 45943 RMSE with 17% of mean_absolute_percentage_error and its R2 score is about 0.82 with explained Variance score of 0.82 so we choose our final model as XGboost and try to find best parameter for it

```
from sklearn.model_selection import GridSearchCV

parameter=[{'n_estimators':[60,80,100,50],'max_depth':
[None,5,10,15],'booster':['gbtree','dart']}]


grid_cv=GridSearchCV(estimator=xgb,param_grid=parameter,cv=10,n_jobs=-
1,verbose=3,scoring='neg_mean_squared_error')

grid_cv.fit(x_train_transformed,y_train)
```

```
Fitting 10 folds for each of 32 candidates, totalling 320 fits
```

```
GridSearchCV(cv=10,
             estimator=XGBRegressor(base_score=None, booster=None,
                                    callbacks=None,
colsample_bylevel=None,
                                    colsample_bynode=None,
                                    colsample_bytree=None,
device=None,
                                    early_stopping_rounds=None,
                                    enable_categorical=False,
eval_metric=None,
                                    feature_types=None, gamma=None,
                                    grow_policy=None,
importance_type=None,
                                    interaction_constraints=None,
                                    learning_rate=None,...
                                    max_cat_to_onehot=None,
max_delta_step=None,
                                    max_depth=None, max_leaves=None,
                                    min_child_weight=None,
missing=nan,
                                    monotone_constraints=None,
                                    multi_strategy=None,
n_estimators=None,
```

```
                                    n_jobs=None,
num_parallel_tree=None,
                                    random_state=None, ...),
             n_jobs=-1,
             param_grid=[{'booster': ['gbtree', 'dart'],
                          'max_depth': [None, 5, 10, 15],
                          'n_estimators': [60, 80, 100, 50]}],
             scoring='neg_mean_squared_error', verbose=3)
```

```
grid_cv.best_params_
```

```
{'booster': 'gbtree', 'max_depth': None, 'n_estimators': 100}
```

```
print('Best RMSE Score',np.sqrt(-grid_cv.best_score_))
```

```
Best RMSE Score 46465.37840520108
```

```
XGB_Model=grid_cv.best_estimator_
```

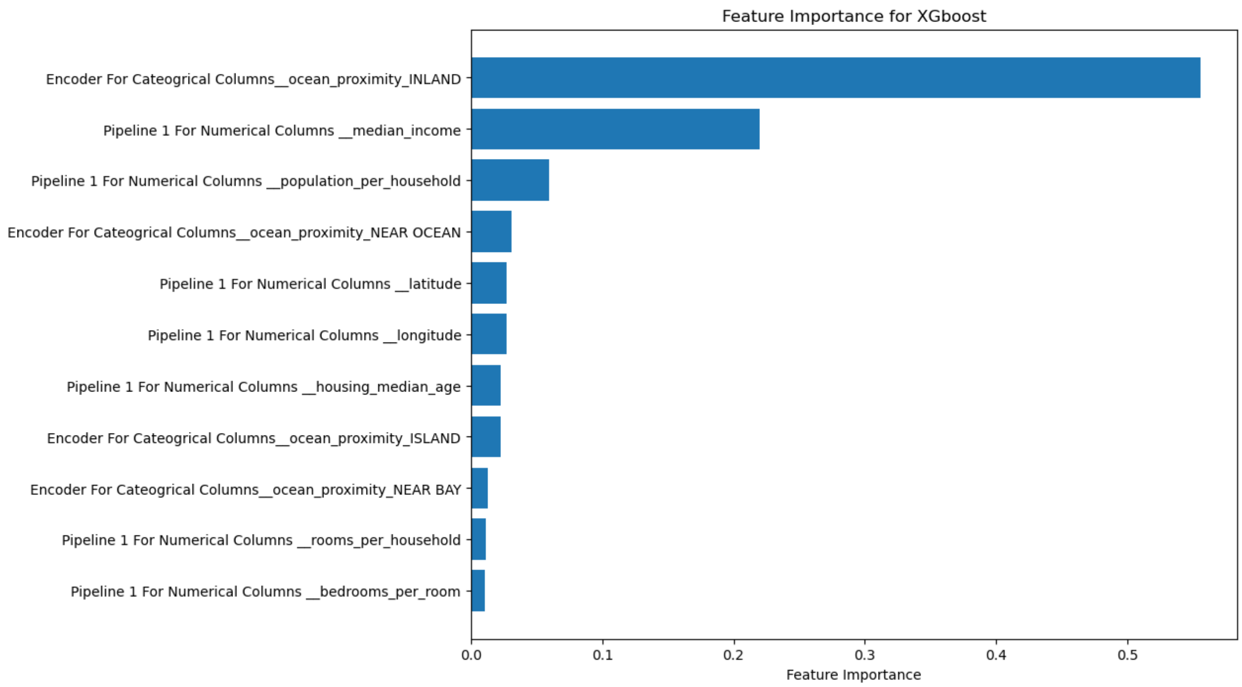
```
XGB_Model
```

```
XGBRegressor(base_score=None, booster='gbtree', callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None,
early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None,
feature_types=None,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None,
max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=None, max_leaves=None,
             min_child_weight=None, missing=nan,
monotone_constraints=None,
             multi_strategy=None, n_estimators=100, n_jobs=None,
             num_parallel_tree=None, random_state=None, ...)
```

```
XGB_Model.feature_importances_
```

```
array([0.02714383, 0.02721211, 0.0229763 , 0.21967393, 0.01127589,
       0.01047511, 0.05931564, 0.5555109 , 0.02286633, 0.01276142,
       0.03078858], dtype=float32)
```

```
importances = XGB_Model.feature_importances_
feature_names = preprocessor.get_feature_names_out()
sorted_indices = importances.argsort()

plt.figure(figsize=(10, 8))
plt.barh(feature_names[sorted_indices], importances[sorted_indices])
plt.xlabel('Feature Importance')
plt.title('Feature Importance for XGboost')
plt.show()
```




```
XGB_Model.fit(x_train_transformed,y_train)

XGBRegressor(base_score=None, booster='gbtree', callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None,
early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None,
feature_types=None,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None,
max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=None, max_leaves=None,
             min_child_weight=None, missing=nan,
monotone_constraints=None,
             multi_strategy=None, n_estimators=100, n_jobs=None,
             num_parallel_tree=None, random_state=None, ...)

new_y_pred_XGB=XGB_Model.predict(x_test_transformed)

print('Percentage
Error',mean_absolute_percentage_error(y_test,new_y_pred_XGB))
print('Mean Square Error',mean_squared_error(y_test,new_y_pred_XGB))
print('RMSE',root_mean_squared_error(y_test,new_y_pred_XGB))
print('R2 Score',r2_score(y_test,new_y_pred_XGB))
print('Variance
Score',explained_variance_score(y_test,new_y_pred_XGB))
```

```
Percentage Error 0.17544368021484072
Mean Square Error 2332399337.424746
RMSE 48294.92041017095
R2 Score 0.824864747596245
Variance Score 0.8251776029971587
```

```
cv_score_xgb_model=cross_val_score(XGB_Model,x_train_transformed,y_tra
in,cv=10,scoring='neg_mean_squared_error')

rmse_xgb_model=np.sqrt(-cv_score_xgb_model)
print('CV Scores',rmse_xgb_model)
print('RMSE',rmse_xgb_model.mean())
```

```
CV Scores [43575.99177336 46011.69672152 45598.91499307 46970.51804133
 49974.68446794 46303.37663823 50087.88041028 46612.5573549
 44922.44872763 44134.51657066]
RMSE 46419.25856989343
```

After doing GridSearchCV we find out best estimators we are able to get the best result which we can get by reducing mean_absolute_percenatge_error to 17% and final RMSE approx to 46419 with r2 score of 0.82 and also explained varwhich is better than any other model which we used before

```
Final_Model=Pipeline(steps=[
    ('Pre Processing',preprocessor),
    ('Random Forest Regressor',XGB_Model),
])

Final_Model
```

```
Pipeline(steps=[('Pre Processing',
                 ColumnTransformer(transformers=[('Pipeline 1 For
Numerical '
                                                  'Columns ',

Pipeline(steps=[('imputer',

SimpleImputer(strategy='median')),

('Scaler',

StandardScaler())],

verbose=True),
                                                  ['longitude',
'latitude',

'housing_median_age',
                                                   'median_income',
```

```
'rooms_per_household',

'bedrooms_per_room',

'population_per_household']),
                                                            ('Encoder For Cat...
                              feature_types=None, gamma=None,
grow_policy=None,
                              importance_type=None,
                              interaction_constraints=None,
learning_rate=None,
                              max_bin=None, max_cat_threshold=None,
                              max_cat_to_onehot=None,
max_delta_step=None,
                              max_depth=None, max_leaves=None,
                              min_child_weight=None, missing=nan,
                              monotone_constraints=None,
multi_strategy=None,
                              n_estimators=100, n_jobs=None,
                              num_parallel_tree=None,
random_state=None, ...))])

Final_Model.fit(x_train,y_train)

[Pipeline] ........... (step 1 of 2) Processing imputer, total=   0.0s
[Pipeline] ............ (step 2 of 2) Processing Scaler, total=   0.0s

Pipeline(steps=[('Pre Processing',
                 ColumnTransformer(transformers=[('Pipeline 1 For
Numerical '
                                                             'Columns ',

Pipeline(steps=[('imputer',

SimpleImputer(strategy='median')),

('Scaler',

StandardScaler())],

verbose=True),
                                                             ['longitude',
'latitude',

'housing_median_age',
                                                              'median_income',

'rooms_per_household',

'bedrooms_per_room',
```

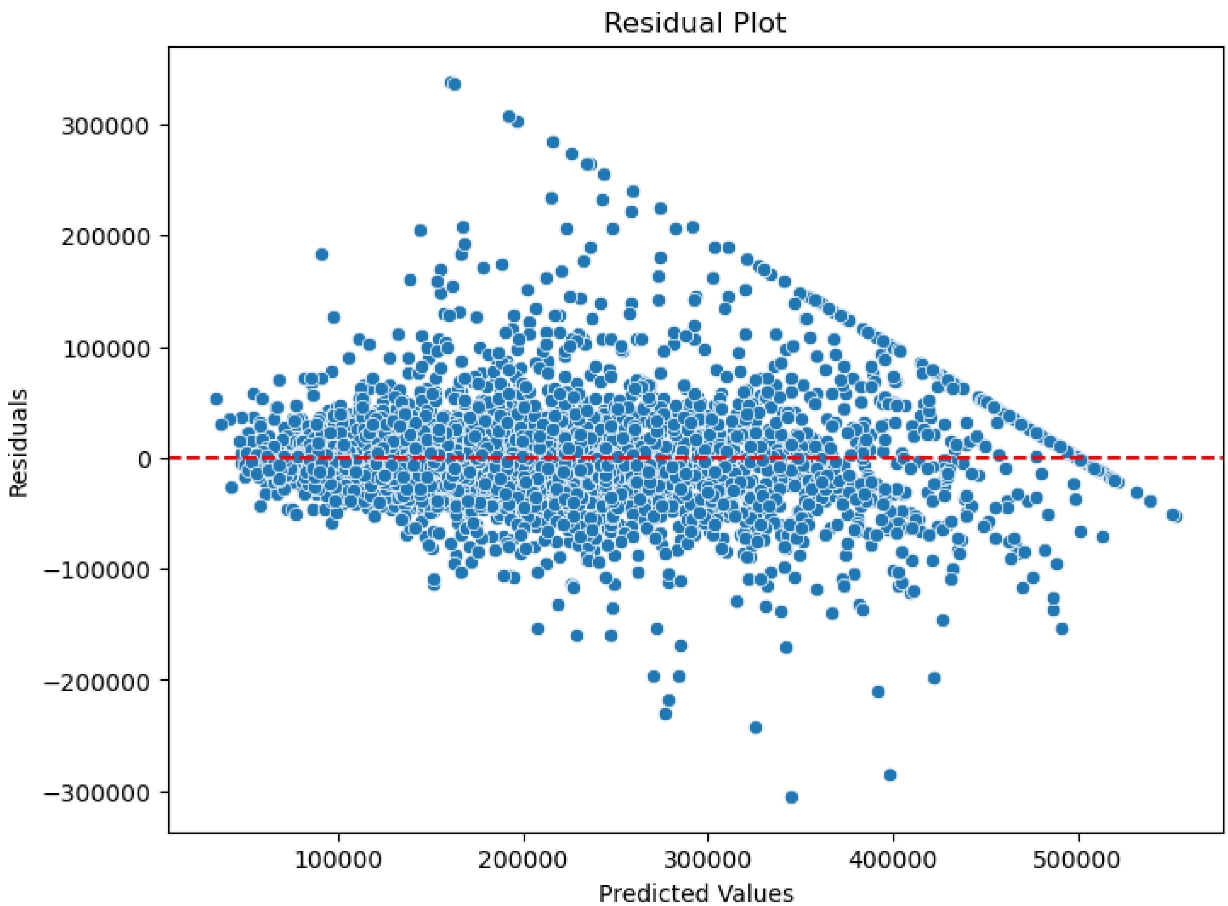
```
'population_per_household']),
                                                  ('Encoder For Cat...
                              feature_types=None, gamma=None,
grow_policy=None,
                              importance_type=None,
                              interaction_constraints=None,
learning_rate=None,
                              max_bin=None, max_cat_threshold=None,
                              max_cat_to_onehot=None,
max_delta_step=None,
                              max_depth=None, max_leaves=None,
                              min_child_weight=None, missing=nan,
                              monotone_constraints=None,
multi_strategy=None,
                              n_estimators=100, n_jobs=None,
                              num_parallel_tree=None,
random_state=None, ...))])
```

```
Final_pred_using_xgb=Final_Model.predict(x_test)
```

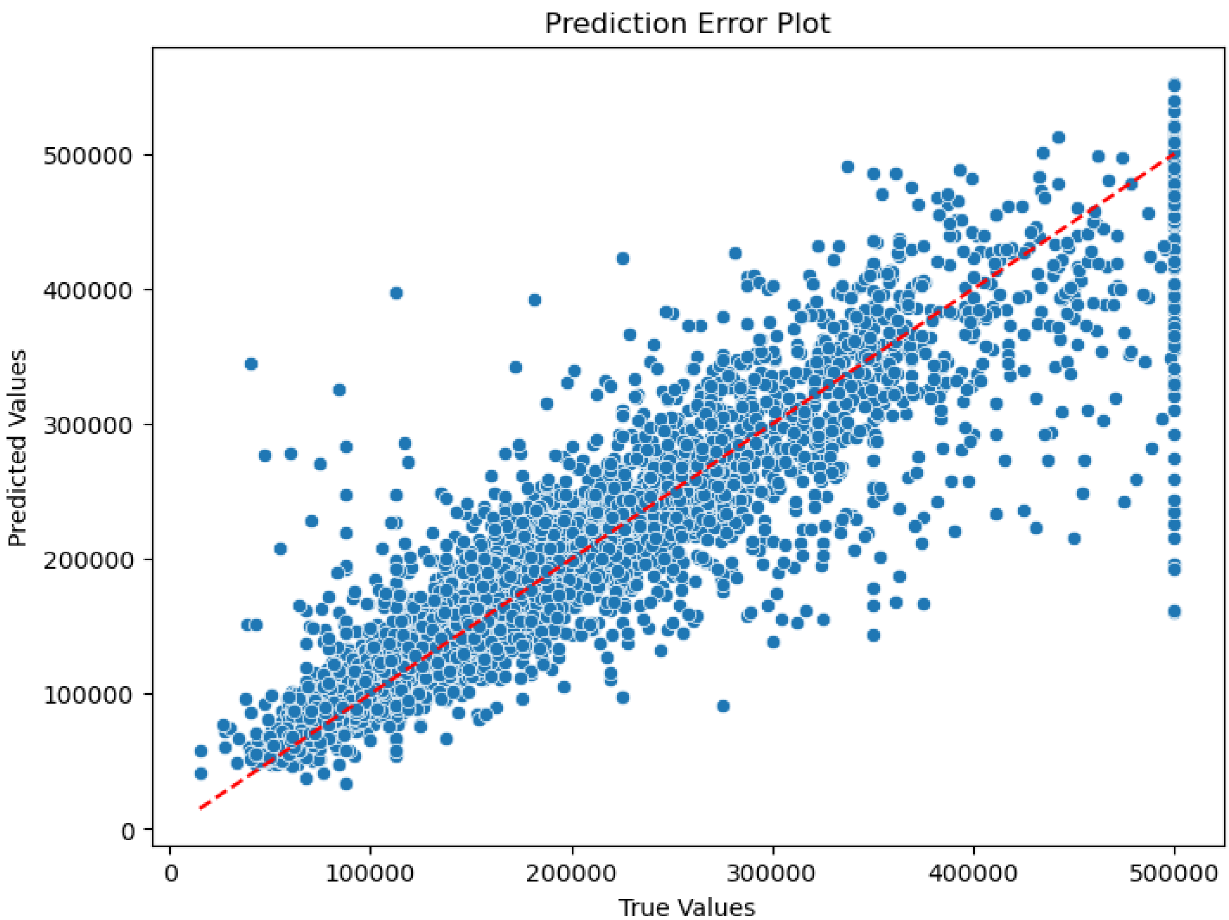
```
print('Final Model RMSE
',root_mean_squared_error(y_test,Final_pred_using_xgb))
```

```
Final Model RMSE  45608.15638318089
```

```
residuals = y_test - Final_pred_using_xgb
plt.figure(figsize=(8, 6))
sns.scatterplot(x=Final_pred_using_xgb, y=residuals)
plt.axhline(y=0, color='r', linestyle='--')
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.title('Residual Plot')
plt.show()
```




```
plt.figure(figsize=(8, 6))
sns.scatterplot(x=y_test, y=Final_pred_using_xgb)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()],
'r--')
plt.xlabel('True Values')
plt.ylabel('Predicted Values')
plt.title('Prediction Error Plot')
plt.show()
```




```
import pickle

with open('Final Model For California Housing Price','wb') as f:
    pickle.dump(Final_Model,f)

with open('Final Model For California Housing Price','rb') as a:
    model=pickle.load(a)


model.predict(pd.DataFrame(np.array([-117.28,32.74,33.0,2.7515,'NEAR
OCEAN',4.235772,0.266795,1.814024]).reshape(1,8),columns=housing.drop(
columns=['median_house_value']).columns))

array([276127.06], dtype=float32)



housing

       longitude  latitude  housing_median_age  median_income  \
3478     -118.16     33.88                30.0         2.9779
18741    -122.67     38.43                17.0         3.2813
```

```
15576    -118.39     34.23                     43.0         2.1518
1398     -121.64     36.66                     24.0         5.2285
1363     -117.00     32.67                     16.0         6.6143
...          ...       ...                      ...            ...
1613     -118.33     33.96                     42.0         2.3000
19104    -117.06     32.76                     38.0         3.2188
11074    -120.67     38.76                     35.0         2.1682
10334    -119.00     35.39                     51.0         2.8295
1244     -122.33     37.55                     51.0         9.3694

       median_house_value ocean_proximity  rooms_per_household  \
3478             169500.0       <1H OCEAN             4.422977
18741            202700.0       <1H OCEAN             4.980149
15576            161600.0       <1H OCEAN             3.728125
1398             248100.0       <1H OCEAN             5.932710
1363             264100.0      NEAR OCEAN             7.386139
...                   ...             ...                  ...
1613             189200.0       <1H OCEAN             5.285266
19104            150500.0      NEAR OCEAN             5.571942
11074            138100.0          INLAND             5.260000
10334             72100.0          INLAND             4.576667
1244             500001.0      NEAR OCEAN             8.300971

       bedrooms_per_room  population_per_household
3478            0.234947                  3.083551
18741           0.199302                  2.220844
15576           0.250629                  3.700000
1398            0.159420                  2.740187
1363            0.137176                  3.306931
...                  ...                       ...
1613            0.214116                  2.310345
19104           0.185926                  2.287770
11074           0.191540                  2.650000
10334           0.206846                  2.160000
1244            0.129435                  2.815534

[20640 rows x 9 columns]
```