```
# import libraries
import pandas as pd  #data processing, CSV file I/O
import numpy as np # linear algebra
import matplotlib.pyplot as plt #creates plot
import seaborn as sns

# For ignoring warning
import warnings
warnings.filterwarnings("ignore")

#Download csv file into our Notebook
#Default separartor is comma
data=pd.read_csv("California_Houses.csv")

#Exploring and cleaning data
#look at the dataset within each district(block)
data

       Median_House_Value  Median_Income  Median_Age  Tot_Rooms
Tot_Bedrooms  \
0                452600.0         8.3252          41        880
129
1                358500.0         8.3014          21       7099
1106
2                352100.0         7.2574          52       1467
190
3                341300.0         5.6431          52       1274
235
4                342200.0         3.8462          52       1627
280
...                   ...            ...         ...        ...
...
20635             78100.0         1.5603          25       1665
374
20636             77100.0         2.5568          18        697
150
20637             92300.0         1.7000          17       2254
485
20638             84700.0         1.8672          18       1860
409
20639             89400.0         2.3886          16       2785
616

       Population  Households  Latitude  Longitude  Distance_to_coast
\
0             322         126     37.88    -122.23        9263.040773

1            2401        1138     37.86    -122.22       10225.733072

2             496         177     37.85    -122.24        8259.085109
```

```
3               558           219      37.85    -122.25         7768.086571
4               565           259      37.85    -122.25         7768.086571
...             ...           ...        ...        ...                 ...
20635           845           330      39.48    -121.09       162031.481121
20636           356           114      39.49    -121.21       160445.433537
20637          1007           433      39.43    -121.22       153754.341182
20638           741           349      39.43    -121.32       152005.022239
20639          1387           530      39.37    -121.24       146866.196892

       Distance_to_LA  Distance_to_SanDiego  Distance_to_SanJose  \
0        556529.158342         735501.806984         67432.517001
1        554279.850069         733236.884360         65049.908574
2        554610.717069         733525.682937         64867.289833
3        555194.266086         734095.290744         65287.138412
4        555194.266086         734095.290744         65287.138412
...                ...                   ...                  ...
20635    654530.186299         830631.543047        248510.058162
20636    659747.068444         836245.915229        246849.888948
20637    654042.214020         830699.573163        240172.220489
20638    657698.007703         834672.461887        238193.865909
20639    648723.337126         825569.179028        233282.769063

       Distance_to_SanFrancisco
0                  21250.213767
1                  20880.600400
2                  18811.487450
3                  18031.047568
4                  18031.047568
...                         ...
20635             222619.890417
20636             218314.424634
20637             212097.936232
20638             207923.199166
20639             205473.376575

[20640 rows x 14 columns]
```

```
#No. of (Rows ,Columns)
data.shape
```

```
(20640, 14)
```

```
#Checking for Duplicates
data.duplicated().sum()

0

#Preprocessing
#1-Check if data has any null values
data.info()
#if there is any nan-> not a number
#Drop null values and save: data.dropna(inplace=True)

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 14 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Median_House_Value        20640 non-null  float64
 1   Median_Income             20640 non-null  float64
 2   Median_Age                20640 non-null  int64  
 3   Tot_Rooms                 20640 non-null  int64  
 4   Tot_Bedrooms              20640 non-null  int64  
 5   Population                20640 non-null  int64  
 6   Households                20640 non-null  int64  
 7   Latitude                  20640 non-null  float64
 8   Longitude                 20640 non-null  float64
 9   Distance_to_coast         20640 non-null  float64
 10  Distance_to_LA            20640 non-null  float64
 11  Distance_to_SanDiego      20640 non-null  float64
 12  Distance_to_SanJose       20640 non-null  float64
 13  Distance_to_SanFrancisco  20640 non-null  float64
dtypes: float64(9), int64(5)
memory usage: 2.2 MB

#OR Checking for null values
data.isnull().sum()

Median_House_Value          0
Median_Income               0
Median_Age                  0
Tot_Rooms                   0
Tot_Bedrooms                0
Population                  0
Households                  0
Latitude                    0
Longitude                   0
Distance_to_coast           0
Distance_to_LA              0
Distance_to_SanDiego        0
Distance_to_SanJose         0
```

```
Distance_to_SanFrancisco    0
dtype: int64

#the dataset contains 20,640 samples and 14 features;

#all features are numerical features encoded as floating number or int
;

#there is no missing values.

#describtive statistics of data
data.describe()

       Median_House_Value  Median_Income    Median_Age
Tot_Rooms  \
count        20640.000000   20640.000000  20640.000000  20640.000000

mean        206855.816909       3.870671     28.639486   2635.763081

std         115395.615874       1.899822     12.585558   2181.615252

min          14999.000000       0.499900      1.000000      2.000000

25%         119600.000000       2.563400     18.000000   1447.750000

50%         179700.000000       3.534800     29.000000   2127.000000

75%         264725.000000       4.743250     37.000000   3148.000000

max         500001.000000      15.000100     52.000000  39320.000000


       Tot_Bedrooms    Population    Households      Latitude
Longitude  \
count  20640.000000  20640.000000  20640.000000  20640.000000
20640.000000
mean     537.898014   1425.476744    499.539680     35.631861    -
119.569704
std      421.247906   1132.462122    382.329753      2.135952
2.003532
min        1.000000      3.000000      1.000000     32.540000    -
124.350000
25%      295.000000    787.000000    280.000000     33.930000    -
121.800000
50%      435.000000   1166.000000    409.000000     34.260000    -
118.490000
75%      647.000000   1725.000000    605.000000     37.710000    -
118.010000
max     6445.000000  35682.000000   6082.000000     41.950000    -
114.310000
```

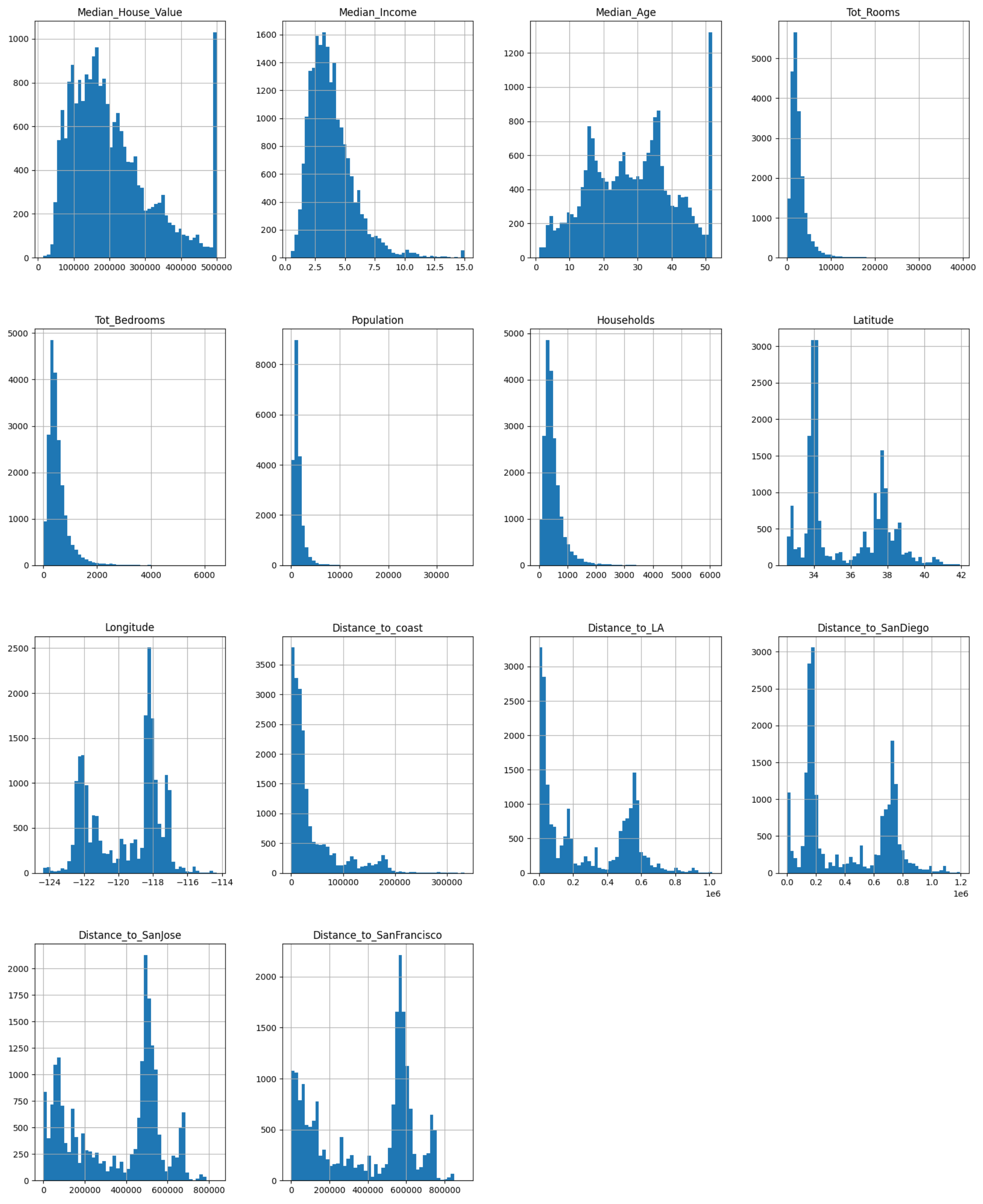
```
       Distance_to_coast  Distance_to_LA  Distance_to_SanDiego  \
count       20640.000000    2.064000e+04          2.064000e+04
mean        40509.264883    2.694220e+05          3.981649e+05
std         49140.039160    2.477324e+05          2.894006e+05
min           120.676447    4.205891e+02          4.849180e+02
25%          9079.756762    3.211125e+04          1.594264e+05
50%         20522.019101    1.736675e+05          2.147398e+05
75%         49830.414479    5.271562e+05          7.057954e+05
max        333804.686371    1.018260e+06          1.196919e+06

       Distance_to_SanJose  Distance_to_SanFrancisco
count         20640.000000              20640.000000
mean         349187.551219             386688.422291
std          217149.875026             250122.192316
min             569.448118                456.141313
25%          113119.928682             117395.477505
50%          459758.877000             526546.661701
75%          516946.490963             584552.007907
max          836762.678210             903627.663298
```

```
#Let's check the distribution of the Target variable.
data['Median_House_Value'].value_counts()
```

```
Median_House_Value
500001.0    965
137500.0    122
162500.0    117
112500.0    103
187500.0     93
           ...
359200.0      1
54900.0       1
377600.0      1
81200.0       1
47000.0       1
Name: count, Length: 3842, dtype: int64
```

```
data.hist(bins=50,figsize=(20,25))#No of bins is No of chuncks to
split data into
```

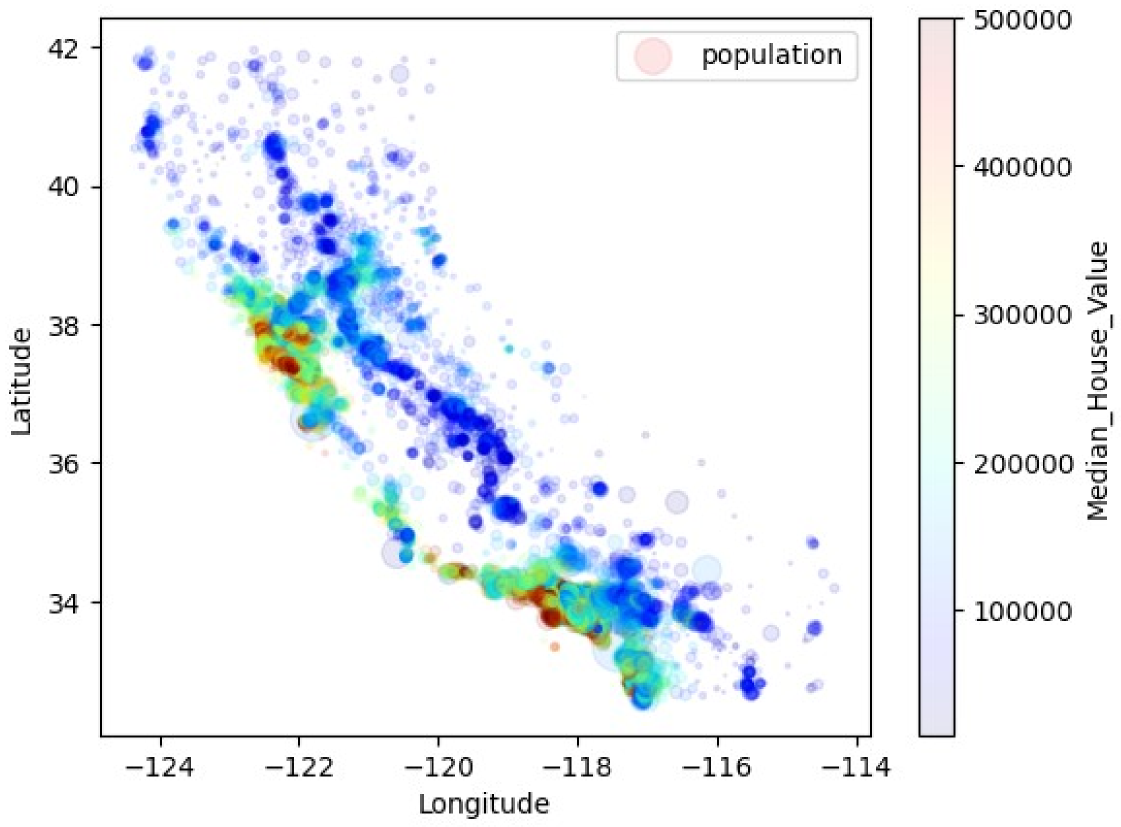
```
array([[<Axes: title={'center': 'Median_House_Value'}>,
        <Axes: title={'center': 'Median_Income'}>,
        <Axes: title={'center': 'Median_Age'}>,
        <Axes: title={'center': 'Tot_Rooms'}>],
       [<Axes: title={'center': 'Tot_Bedrooms'}>,
        <Axes: title={'center': 'Population'}>,
        <Axes: title={'center': 'Households'}>,
        <Axes: title={'center': 'Latitude'}>],
       [<Axes: title={'center': 'Longitude'}>,
        <Axes: title={'center': 'Distance_to_coast'}>,
        <Axes: title={'center': 'Distance_to_LA'}>,
        <Axes: title={'center': 'Distance_to_SanDiego'}>],
       [<Axes: title={'center': 'Distance_to_SanJose'}>,
        <Axes: title={'center': 'Distance_to_SanFrancisco'}>, <Axes:
>,
        <Axes: >]], dtype=object)
```



```
#Median house value has outliers at the very right where some houses
that price much much
#higher than the rest of the population & they are all maxed out at
500K$
```

```
#Means every house with at least 500k$ is labeled as 500k$
#Also in MEdian house age all houses age above 50 years are grouped
together

#plot the geographical features -> Map of California
data.plot(kind="scatter",x="Longitude",y="Latitude",alpha=0.1,
         s=data["Population"]/100,label="population",
         c="Median_House_Value",cmap=plt.get_cmap("jet"))
#alpha is used for transparency thus when points are laid over each
other they are darker
#darker points houses near cost are more expensive
#sized of dots represents the population of the district
#color corresponds to house value
```

```
<Axes: xlabel='Longitude', ylabel='Latitude'>
```



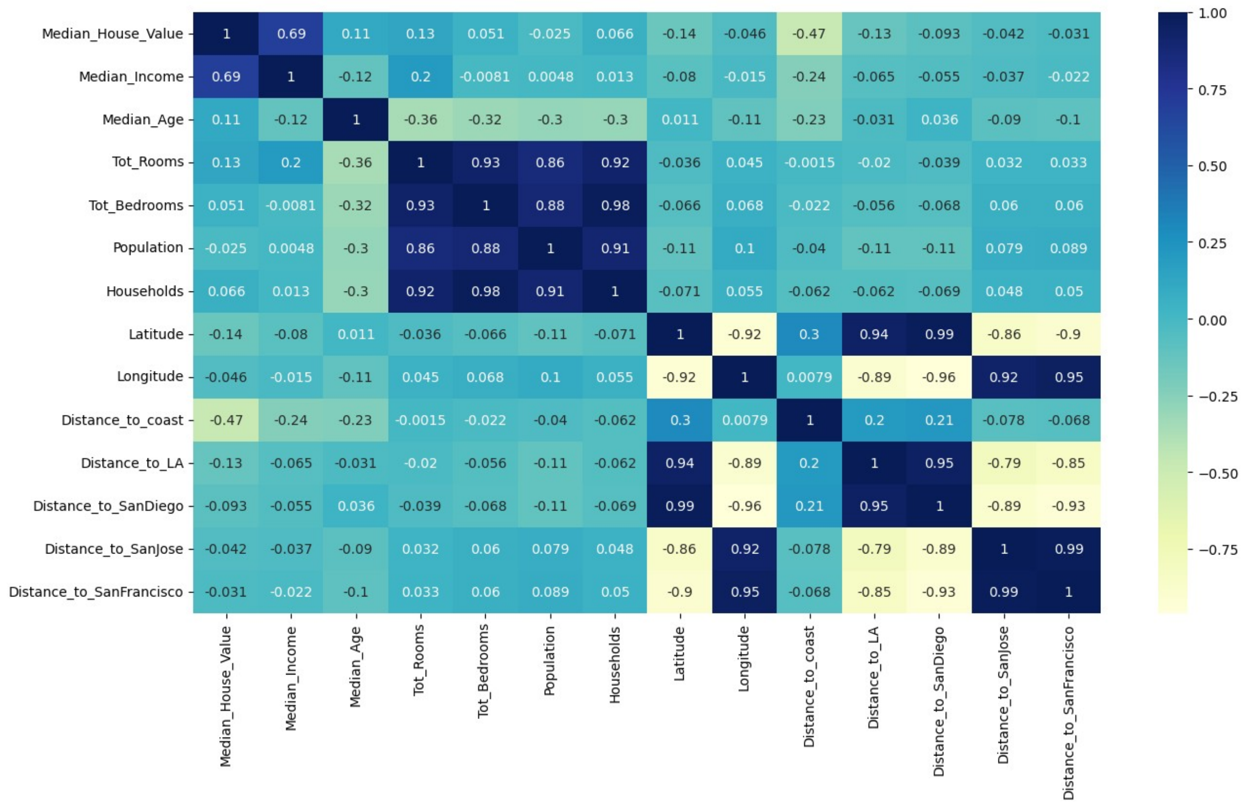
```
#create correlation matrix in a table corr_matrix->shows how closely
related 2 variables are
#or how much one variable changes as the other one changes
#correlation of a variable with itself=1 highest possible value of
correlation
corr_matrix=data.corr()
#correlation Rage [-1,+1]
#corr=-1: exactly the opposite
```

```
#corr=0: No relation
#corr1:exactly the same
# if close to one or -1 they are closely related/
corr_matrix["Median_House_Value"].sort_values(ascending=False)
#Specify column and sort correlations in descending order
```

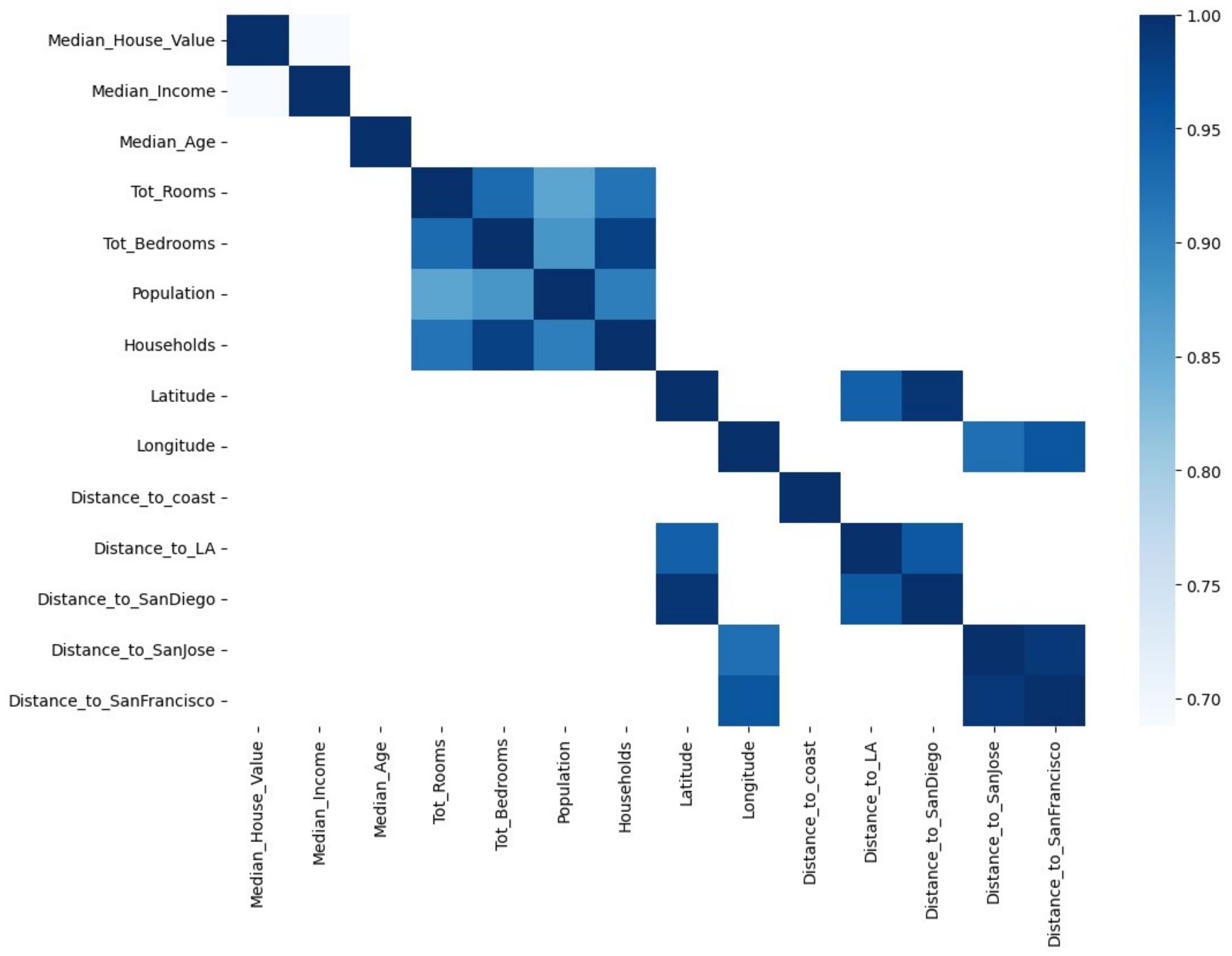
```
Median_House_Value           1.000000
Median_Income                0.688075
Tot_Rooms                    0.134153
Median_Age                   0.105623
Households                   0.065843
Tot_Bedrooms                 0.050594
Population                  -0.024650
Distance_to_SanFrancisco    -0.030559
Distance_to_SanJose         -0.041590
Longitude                   -0.045967
Distance_to_SanDiego        -0.092510
Distance_to_LA              -0.130678
Latitude                    -0.144160
Distance_to_coast           -0.469350
Name: Median_House_Value, dtype: float64
```

```
#plot heatmap of correlation
plt.figure(figsize=(15,8))
sns.heatmap(corr_matrix,annot=True, cmap="YlGnBu")
```

```
<Axes: >
```



```
#plot heatmap for correlated data above 50%
kot = corr_matrix[corr_matrix>=.50]
plt.figure(figsize=(12,8))
sns.heatmap(kot, cmap="Blues")

<Axes: >
```



```
 #feature engineering: Combine features to generate new interesting
features
#Exp: -Total no. of bedrooms
#     -Total no. of rooms
#New feature: The number of bedrooms per room
data['Badroom_Ratio']=data['Tot_Bedrooms']/data['Tot_Rooms']
#New feature: How many rooms per household
# data['Household_Ratio']=data['Tot_Rooms']/data['Households']
# data['Rooms_Population']=data['Tot_Rooms']*data['Population']
# data['Bedrooms_Population']=data['Tot_Bedrooms']*data['Population']
# data['Bedrooms_Households']=data['Tot_Bedrooms']*data['Households']
# data['Population_Households']=data['Population']*data['Households']
data['Value_Income']=data['Median_House_Value']*data['Median_Income']
data['Latitude_LA']=data['Latitude']*data['Distance_to_LA']
data['Latitude_SanDiego']=data['Latitude']*data['Distance_to_SanDiego'
]
data['Longitude_SanJose']=data['Longitude']*data['Distance_to_SanJose'
]
data['Longitude_SanFrancisco']=data['Longitude']*data['Distance_to_San
Francisco']
```

```
data['LA_SanDiego']=data['Distance_to_LA']*data['Distance_to_SanDiego']
]
data['SanJose_SanFrancisco']=data['Distance_to_SanJose']*data['Distanc
e_to_SanFrancisco']


#To remove skewness in data , +1 is used to avoid log(0)!!!
data['Tot_Rooms']=np.log(data['Tot_Rooms']+1)
data['Tot_Bedrooms']=np.log(data['Tot_Bedrooms']+1)
data['Population']=np.log(data['Population']+1)
data['Households']=np.log(data['Households']+1)
data['Distance_to_coast']=np.log(data['Distance_to_coast']+1)

#Creating features and labels dataset
#x is the features dataset without target(label), Drop the target -
>Median_House_Value
x=data.drop(['Median_House_Value'],axis =1 ) # axis=1 to drop column
#y is only the target (label) dataset
y=data['Median_House_Value']

x

       Median_Income  Median_Age  Tot_Rooms  Tot_Bedrooms  Population
\
0             8.3252          41   6.781058      4.867534    5.777652

1             8.3014          21   8.867850      7.009409    7.784057

2             7.2574          52   7.291656      5.252273    6.208590

3             5.6431          52   7.150701      5.463832    6.326149

4             3.8462          52   7.395108      5.638355    6.338594

...              ...         ...        ...           ...         ...

20635         1.5603          25   7.418181      5.926926    6.740519

20636         2.5568          18   6.548219      5.017280    5.877736

20637         1.7000          17   7.720905      6.186209    6.915723

20638         1.8672          18   7.528869      6.016157    6.609349

20639         2.3886          16   7.932362      6.424869    7.235619


       Households  Latitude  Longitude  Distance_to_coast
Distance_to_LA  \
0        4.844187     37.88    -122.23           9.133896
556529.158342
```

```
1        7.037906     37.86    -122.22           9.232760
554279.850069
2        5.181784     37.85    -122.24           9.019190
554610.717069
3        5.393628     37.85    -122.25           8.957908
555194.266086
4        5.560682     37.85    -122.25           8.957908
555194.266086
...           ...       ...       ...                ...
...
20635    5.802118     39.48    -121.09          11.995552
654530.186299
20636    4.744932     39.49    -121.21          11.985715
659747.068444
20637    6.073045     39.43    -121.22          11.943118
654042.214020
20638    5.857933     39.43    -121.32          11.931675
657698.007703
20639    6.274762     39.37    -121.24          11.897284
648723.337126

       ...  Distance_to_SanJose  Distance_to_SanFrancisco
Badroom_Ratio  \
0      ...         67432.517001              21250.213767
0.146591
1      ...         65049.908574              20880.600400
0.155797
2      ...         64867.289833              18811.487450
0.129516
3      ...         65287.138412              18031.047568
0.184458
4      ...         65287.138412              18031.047568
0.172096
...    ...                  ...                       ...             .
..
20635  ...        248510.058162             222619.890417
0.224625
20636  ...        246849.888948             218314.424634
0.215208
20637  ...        240172.220489             212097.936232
0.215173
20638  ...        238193.865909             207923.199166
0.219892
20639  ...        233282.769063             205473.376575
0.221185

       Value_Income   Latitude_LA  Latitude_SanDiego
Longitude_SanJose  \
0        3767985.52  2.108132e+07       2.786081e+07      -
```

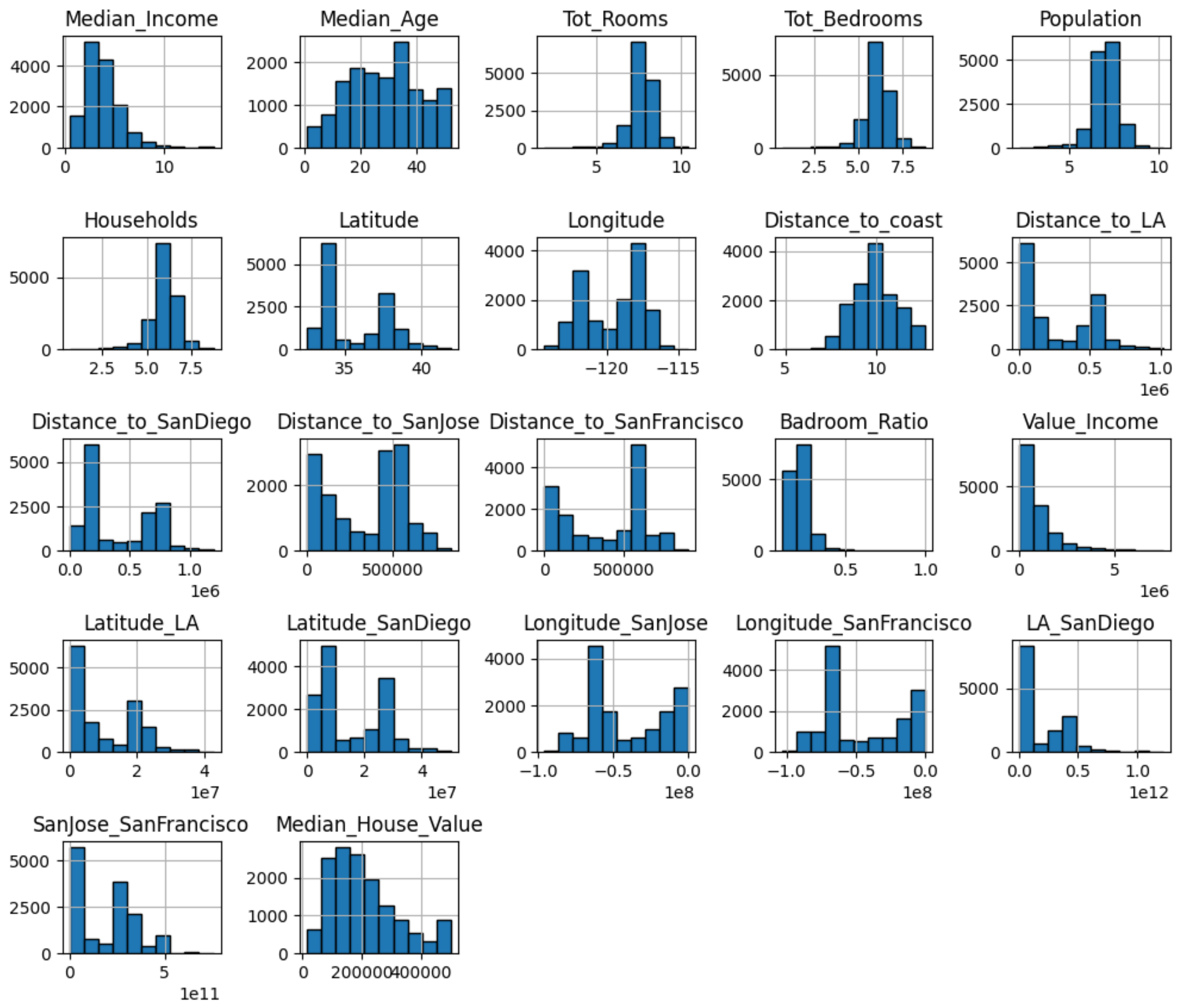
```
8.242277e+06
1        2976051.90  2.098504e+07       2.776035e+07      -
7.950400e+06
2        2555330.54  2.099202e+07       2.776395e+07      -
7.929378e+06
3        1925990.03  2.101410e+07       2.778551e+07      -
7.981353e+06
4        1316169.64  2.101410e+07       2.778551e+07      -
7.981353e+06
...             ...           ...                ...                ..
.
20635     121859.43  2.584085e+07       3.279333e+07      -
3.009208e+07
20636     197129.28  2.605341e+07       3.302335e+07      -
2.992068e+07
20637     156910.00  2.578888e+07       3.275448e+07      -
2.911368e+07
20638     158151.84  2.593303e+07       3.291114e+07      -
2.889768e+07
20639     213540.84  2.554024e+07       3.250266e+07      -
2.828320e+07

       Longitude_SanFrancisco   LA_SanDiego  SanJose_SanFrancisco
0                -2.597414e+06  4.093282e+11          1.432955e+09
1                -2.552027e+06  4.064184e+11          1.358281e+09
2                -2.299516e+06  4.068212e+11          1.220250e+09
3                -2.204296e+06  4.075655e+11          1.177195e+09
4                -2.204296e+06  4.075655e+11          1.177195e+09
...                        ...           ...                   ...
20635            -2.695704e+07  5.436734e+11          5.532328e+10
20636            -2.646189e+07  5.517108e+11          5.389089e+10
20637            -2.571051e+07  5.433126e+11          5.094003e+10
20638            -2.522524e+07  5.489624e+11          4.952603e+10
20639            -2.491159e+07  5.355660e+11          4.793340e+10

[20640 rows x 21 columns]

y

0        452600.0
1        358500.0
2        352100.0
3        341300.0
4        342200.0
           ...
20635     78100.0
20636     77100.0
20637     92300.0
20638     84700.0
```

```
20639     89400.0
Name: Median_House_Value, Length: 20640, dtype: float64
```

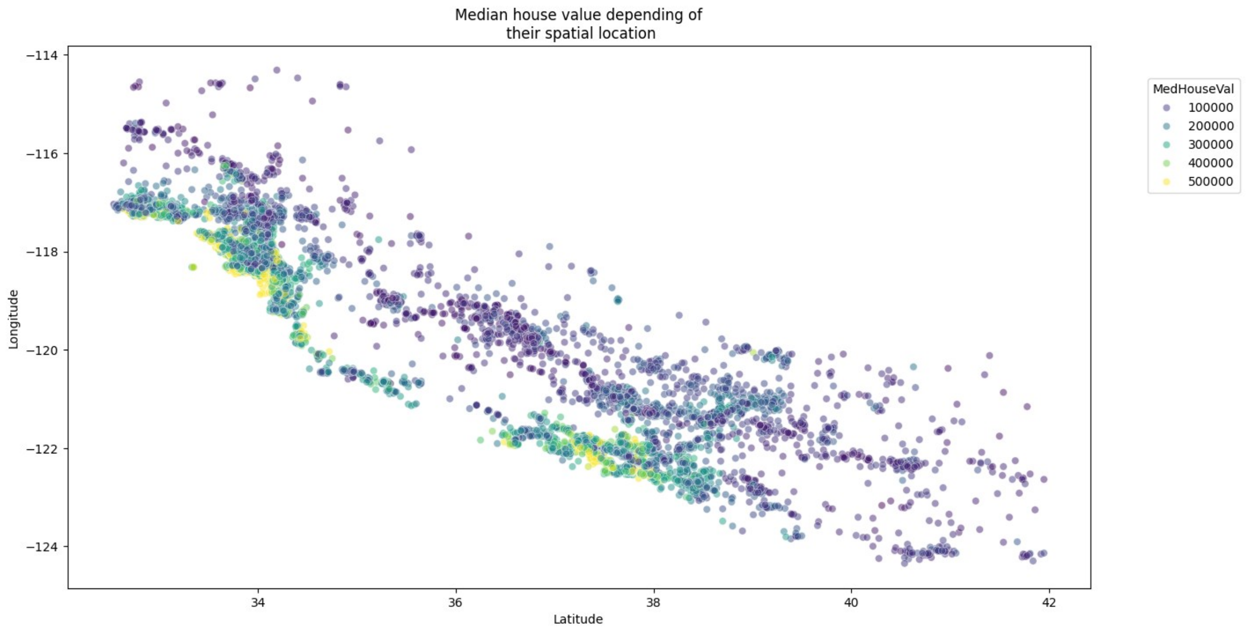
```
from sklearn.model_selection import train_test_split

# set aside 15% of train and test data for evaluation
x_train, x_test, y_train, y_test = train_test_split(x, y,
    test_size=0.3, shuffle = True, random_state = 1900)
# Use the same function above for the validation set
x_test, x_val, y_test, y_val = train_test_split(x_test, y_test,
    test_size=0.5, random_state= 8) # 0.3 x 0.5= 0.15

#Data Spliting
print("x_train shape: {}".format(x_train.shape))
print("y_train shape: {}".format(y_train.shape))
print("x_val shape: {}".format(x_val.shape))
print("y val shape: {}".format(y_val.shape))
print("x_test shape: {}".format(x_test.shape))
print("y_test shape: {}".format(y_test.shape))
```

```
x_train shape: (14448, 21)
y_train shape: (14448,)
x_val shape: (3096, 21)
y val shape: (3096,)
x_test shape: (3096, 21)
y_test shape: (3096,)
```

```
#use only train data
#to get again the combined data frame but only for the training data
train_data=x_train.join(y_train)

# Data Exploration
#To get a histogram of individual features in Training set
train_data.hist(figsize=(12,10),edgecolor="black")
plt.subplots_adjust(hspace=0.8, wspace=0.5)
```



```
plt.figure(figsize=(15,8))
sns.scatterplot(
    data=train_data, x="Latitude",y="Longitude",
hue="Median_House_Value" ,palette="viridis",
    alpha=0.5
)
plt.legend(title="MedHouseVal", bbox_to_anchor=(1.05, 0.95),
loc="upper left")
_ = plt.title("Median house value depending of\n their spatial
location")
```



```
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
#linear regressing is called ordinary least squares-> OLS
OLS= Pipeline([
#Scaling input  No need to scale output
    ('scaler', StandardScaler()),
    ('regressor', LinearRegression())
])
OLS.fit(x_train,y_train)
```

```
Pipeline(steps=[('scaler', StandardScaler()),
                ('regressor', LinearRegression())])
```

```
# Display intercept and coefficients of the OLS model and R^2
print("Intercept is " + str(OLS['regressor'].intercept_))
print("The set of coefficients are " + str(OLS['regressor'].coef_))
#R^2 is the measure of performance of the OLS regression
print("The R-Squared value is " + str(OLS.score (x_train,y_train)))
```

```
Intercept is 207809.14168050993
The set of coefficients are [  -58070.26907658    -4782.89824866
71598.85914909   -51330.44625947
   -40704.57555972    24227.22027286  -315641.77479945
38271.41081192
   -13564.84107437  -145035.58914444 -1041973.07345917
2894705.00070248
 -1210452.59847144     8928.02964488   128176.14191814
64062.65017811
  1288718.83829814  2819391.9075079  -1160209.33682804
62609.68754738
  -157230.57494066]
The R-Squared value is 0.848097947453569
```

```
#getting the best Hyperparameter for Ridge regression
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import Ridge
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
# set aside 15% of train and test data for evaluation
x_train, x_test, y_train, y_test = train_test_split(x, y,
    test_size=0.3, shuffle = True, random_state = 1900)
# Use the same function above for the validation set
x_test, x_val, y_test, y_val = train_test_split(x_test, y_test,
test_size=0.5, random_state= 8) # 0.3 x 0.5= 0.15
errors =[]
#we will use our validation set
values = [1e-15, 1e-10, 1e-18, 1e-13, 1e-12, 1, 5, 10, 20, 30, 35, 40,
45, 50, 55, 100]
for i in range(len(values)):
    ridge= Pipeline([
#Scaling input  No need to scale output
    ('scaler', StandardScaler()),
    ('regressor', Ridge(alpha=values[i]))
    ])
    ridge.fit(x_val,y_val)
    prediction = ridge.predict(x_val)
    MSE=mean_squared_error(y_val,prediction)
    errors.append(MSE)
index_of_best_alpha = np.array(errors).argmin()
best_alpha = values[index_of_best_alpha]
print("the best hyperparameter of Ridge regression: "+str(best_alpha))

ridge= Pipeline([
#Scaling input  No need to scale output
    ('scaler', StandardScaler()),
    ('ridge_regressor', Ridge(alpha=best_alpha))
])
ridge.fit(x_train,y_train)
# print(ridge.score(x_train,y_train))
```

```
the best hyperparameter of Ridge regression: 1e-15
```

```
Pipeline(steps=[('scaler', StandardScaler()),
                ('ridge_regressor', Ridge(alpha=1e-15))])
```

```
#applying Ridge regression with another way to get best hyperparameter
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV

parameters = {'alpha':[1e-15,1e-10,1e-18,1e-13,1e-
12,1,5,10,20,30,35,40,45,50,55,100]}
#chooseing best Hyperparameter by GridSearchCV
ridge_regressor=GridSearchCV(Ridge(),parameters,cv=5)
# ridge_regressor.fit(x_train,y_train)
ridge= Pipeline([
#Scaling input  No need to scale output
    ('scaler', StandardScaler()),
    ('ridge_regressor', ridge_regressor)
])
#compare the Hyperparameter values between the 2 diffrent methods
ridge.fit(x_train,y_train)
if(ridge['ridge_regressor'].best_params_['alpha']==best_alpha):
    print("As we can see the Hyperparameter of the two methods have
the same value which is: "+str(best_alpha))
else:
    print("As we can see the Hyperparameter of the two methods are
diffrent \n method-1= "+str(best_alpha)+"\n method-2=
"+str(ridge['ridge_regressor'].best_params_['alpha']))


print(" How ever the diffrence are very small and both of them  got
the same score: "+str(ridge.score(x_train,y_train)))
```

```
As we can see the Hyperparameter of the two methods are diffrent
 method-1= 1e-15
 method-2= 1e-10
 How ever the diffrence are very small and both of them  got the same
score: 0.848097947453569
```

```
#getting the best Hyperparameter for Lasso regression
#we will use our validation setonly this time because it is the
requierd method
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import Lasso
errors =[]
values = [1e-15, 1e-10, 1e-18, 1e-13, 1e-12, 1, 5, 10, 20, 30, 35, 40,
45, 50, 55, 100]

for i in range(len(values)):
    lasso = Lasso(alpha=values[i])
    lasso.fit(x_val,y_val)
    prediction = lasso.predict(x_val)
    MSE=mean_squared_error(y_val,prediction)
    errors.append(MSE)
index_of_best_alpha = np.array(errors).argmin()
best_alpha_l = values[index_of_best_alpha]
```

```
print("the best hyperparameter of lasso regression: 
"+str(best_alpha_l))
lasso= Pipeline([
#Scaling input  No need to scale output
    ('scaler', StandardScaler()),
    ('ridge_regressor', Lasso(alpha=best_alpha_l))
])
lasso.fit(x_train,y_train)
```

```
the best hyperparameter of lasso regression: 1e-12
```

```
Pipeline(steps=[('scaler', StandardScaler()),
                ('ridge_regressor', Lasso(alpha=1e-12))])
```

```
 from sklearn.metrics import mean_squared_error
#predicting with OLS
y_pred=OLS.predict(x_val)#The predicted values based on validate 
dataset
MSE=mean_squared_error(y_val,y_pred)
MSE
print("Mean square error: " + str( MSE))
```

```
Mean square error: 1844847809.5235841
```

```
y_pred
```

```
array([248695.33350802,  84759.68034878, 156303.63855013, ...,
       184375.74545651, 233194.41810921, 459402.60582285])
```

```
RMSE= np.sqrt(MSE)
print("Root mean square error: " + str( RMSE))
```

```
Root mean square error: 42951.69157930319
```

```
from sklearn.metrics import mean_absolute_error as MAE
# calculate MAE
error = MAE(y_val,y_pred)

# display
print("Mean absolute error : " + str(error))
```

```
Mean absolute error : 29667.89139684592
```

```
y_pred_R=ridge.predict(x_val)#The predicted values based on validate 
dataset
#predicting with OLS
MSE_R=mean_squared_error(y_val,y_pred_R)
MSE_R
print("Mean square error for Ridge Regression: " + str( MSE_R))
```

```
Mean square error for Ridge Regression: 1844847809.4254832
```

```
RMSE_R= np.sqrt(MSE_R)
print("Root mean square error for Ridge Regression: " + str( RMSE_R))
```

```
Root mean square error for Ridge Regression: 42951.6915781612
```

```
from sklearn.metrics import mean_absolute_error as MAE
# calculate MAE
error = MAE(y_val,y_pred_R)

# display
print("Mean absolute error : " + str(error))
```

```
Mean absolute error : 29667.891399970013
```

```
y_pred_L=lasso.predict(x_val)#The predicted values based on validate
dataset
#predicting with OLS
MSE_L=mean_squared_error(y_val,y_pred_L)
MSE_L
print("Mean square error for Lasso Regression: " + str( MSE_L))
```

```
Mean square error for Lasso Regression: 1903693311.5767913
```

```
RMSE_L= np.sqrt(MSE_L)
print("Root mean square error for Lasso Regression: " + str( RMSE_L))
```

```
Root mean square error for Lasso Regression: 43631.3340568082
```

```
from sklearn.metrics import mean_absolute_error as MAE
# calculate MAE
error = MAE(y_val,y_pred_L)

# display
print("Mean absolute error for Lasso regression : " + str(error))
```

```
Mean absolute error for Lasso regression : 30304.374774314463
```

```
Performance=pd.DataFrame({'PREDICTIONS':y_pred,'ACTUAL VALUES':y_val})
#creating new column error
Performance['Error']=Performance['ACTUAL VALUES']-
Performance['PREDICTIONS']
Performance.head()
```

```
         PREDICTIONS  ACTUAL VALUES         Error
17909  248695.333508       247600.0  -1095.333508
2451    84759.680349        67000.0 -17759.680349
15005  156303.638550       133700.0 -22603.638550
9346   275307.111164       316300.0  40992.888836
3543   350063.426898       357100.0   7036.573102
```

```
#preparing data for plotting :
Performance.reset_index(drop=True,inplace=True)
#new column index-> No. of observations
Performance.reset_index(inplace=True)
Performance.head()
```

```
   index    PREDICTIONS  ACTUAL VALUES         Error
0      0  248695.333508       247600.0  -1095.333508
1      1   84759.680349        67000.0 -17759.680349
2      2  156303.638550       133700.0 -22603.638550
3      3  275307.111164       316300.0  40992.888836
4      4  350063.426898       357100.0   7036.573102
```

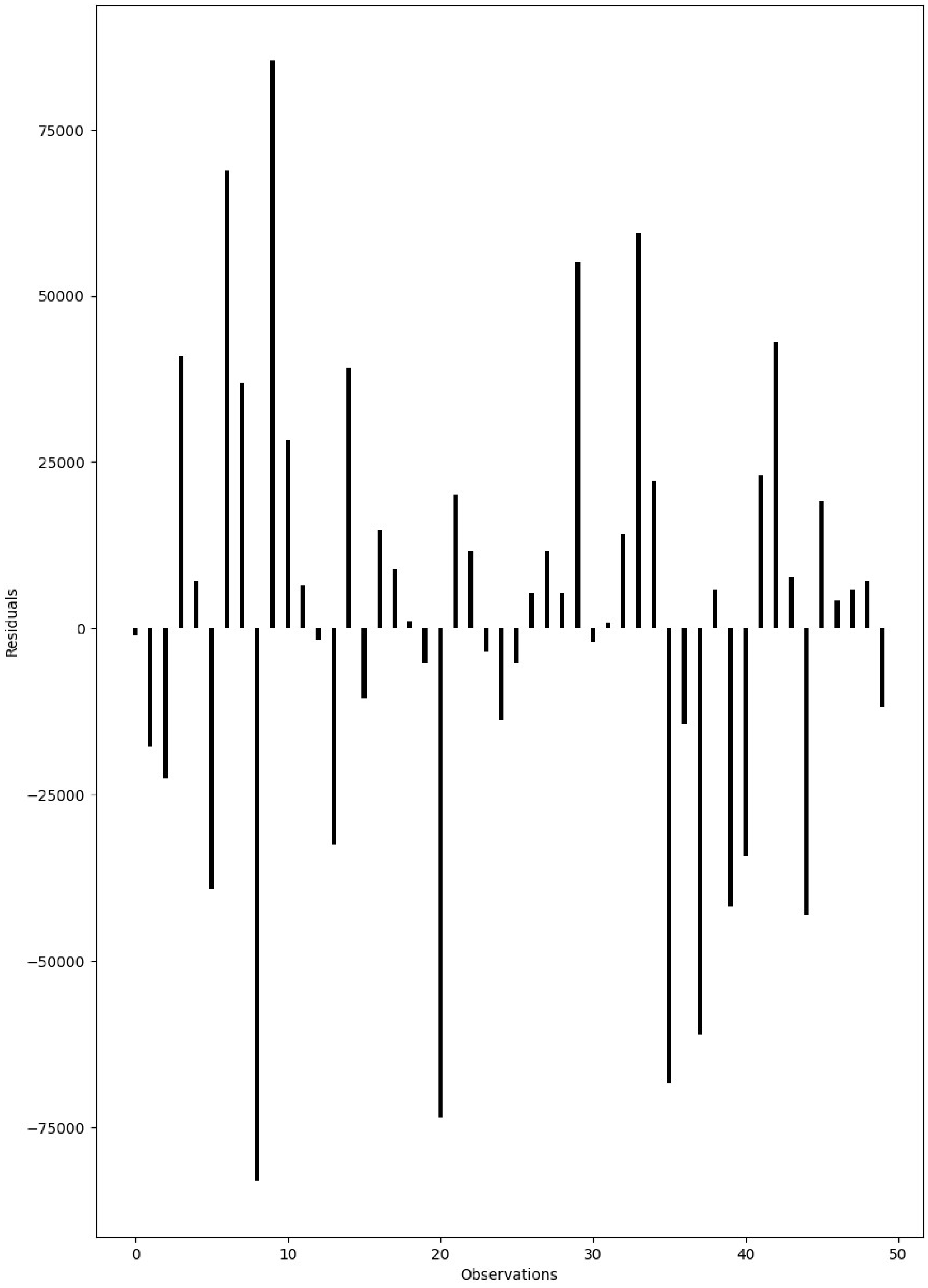
```
Performance_l=pd.DataFrame({'PREDICTIONS_l':y_pred_L,'ACTUAL
VALUES_l':y_val})
#creating new column error
Performance_l['Error_l']=Performance_l['ACTUAL VALUES_l']-
Performance_l['PREDICTIONS_l']

Performance_r=pd.DataFrame({'PREDICTIONS_r':y_pred_R,'ACTUAL
VALUES_r':y_val})
#creating new column error
Performance_r['Error_l']=Performance_r['ACTUAL VALUES_r']-
Performance_r['PREDICTIONS_r']

#preparing data for plotting :
Performance_l.reset_index(drop=True,inplace=True)
#new column index-> No. of observations
Performance_l.reset_index(inplace=True)
Performance_l.head()
```

```
   index  PREDICTIONS_l  ACTUAL VALUES_l       Error_l
0      0  244352.327026         247600.0   3247.672974
1      1   95647.921289          67000.0 -28647.921289
2      2  154470.592652         133700.0 -20770.592652
3      3  277081.777510         316300.0  39218.222490
4      4  349393.830291         357100.0   7706.169709
```

```
#preparing data for plotting :
Performance_r.reset_index(drop=True,inplace=True)
#new column index-> No. of observations
Performance_r.reset_index(inplace=True)
Performance_r.head()
```

```
   index  PREDICTIONS_r  ACTUAL VALUES_r       Error_l
0      0  248695.333571         247600.0  -1095.333571
1      1   84759.680540          67000.0 -17759.680540
2      2  156303.638538         133700.0 -22603.638538
3      3  275307.111338         316300.0  40992.888662
4      4  350063.426986         357100.0   7036.573014
```

```
#plot the residuals(errors)
fig=plt.figure(figsize=(10,15))
#plot bar chart x-axis-> index , y-axis->error
plt.bar('index','Error',data=Performance[:50],color='black',width=0.3)
#data=Performance[:50] this means getting data from the performance
dataset's first 50 observaions to be clear
plt.xlabel("Observations")
plt.ylabel("Residuals")
plt.show()
#Error is positive means underestimating: Prediction lesser than the
actual value
#Error is negative means overestimating: Prediction higher than the
actual value
```



75000
50000
25000
0
−25000
−50000
−75000
Residuals
0
10
20
30
40
50
Observations

```
import statsmodels.api as sm #specialized library for statisticals
model Displays results better than sklearn
x_train=sm.add_constant(x_train)
#add constant =1 o train data set this important to get statsmodel
linear regression with intercept correctly identical to sklearn
x_train.head()
```

```
       const  Median_Income  Median_Age  Tot_Rooms  Tot_Bedrooms
Population  \
6791     1.0         3.1250          44   6.960348      5.529429
6.848005
2888     1.0         1.8319          36   7.271704      5.855072
6.961296
3402     1.0         6.2037          32   6.405228      4.691348
5.752573
9330     1.0         6.1359          16   4.615121      3.044522
3.828641
12520    1.0         1.1458          48   6.995766      6.001415
6.831954

       Households  Latitude  Longitude  Distance_to_coast  ...  \
6791     5.549076     34.08    -118.15          10.413730  ...
2888     5.834811     35.39    -118.99          11.710204  ...
3402     4.736198     34.27    -118.35          10.359508  ...
9330     3.258097     37.96    -122.50           6.738164  ...
12520    5.820083     38.55    -121.47          10.888868  ...

       Distance_to_SanJose  Distance_to_SanFrancisco  Badroom_Ratio  \
6791         495140.641637             563175.694111       0.238367
2888         338152.763741             405760.086533       0.242003
3402         467129.274368             535164.210208       0.178808
9330          87624.405549              21546.591499       0.200000
12520        140050.130972             120454.322106       0.369386

       Value_Income   Latitude_LA  Latitude_SanDiego
Longitude_SanJose  \
6791      642500.00  3.123455e+05       6.042801e+06      -
5.850087e+07
2888      101487.26  5.791362e+06       1.209503e+07      -
4.023680e+07
3402     1274239.98  8.949827e+05       7.025104e+06      -
5.528475e+07
9330     1303878.75  2.197829e+07       2.876226e+07      -
1.073399e+07
12520      74935.32  2.226577e+07       2.915958e+07      -
1.701189e+07

       Longitude_SanFrancisco   LA_SanDiego  SanJose_SanFrancisco
6791            -6.653921e+07  1.625078e+09          2.788512e+11
2888            -4.828139e+07  5.592763e+10          1.372089e+11
```

```
3402            -6.333668e+07  5.353517e+09          2.499909e+11
9330            -2.639457e+06  4.386969e+11          1.888007e+09
12520           -1.463159e+07  4.368880e+11          1.686964e+10

[5 rows x 22 columns]
```

```
#sm.OLS(label dataset,features dataset that includes constant
value).fit
nicerOLS=sm.OLS(y_train,x_train).fit()
nicerOLS.summary()
#comment: Constant is now the intercept
```

```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results

======================================================================
========
Dep. Variable:     Median_House_Value   R-squared:
0.848
Model:                            OLS   Adj. R-squared:
0.848
Method:                 Least Squares   F-statistic:
4026.
Date:                Wed, 01 Nov 2023   Prob (F-statistic):
0.00
Time:                        23:21:17   Log-Likelihood:            -
1.7535e+05
No. Observations:               14448   AIC:
3.507e+05
Df Residuals:                   14427   BIC:
3.509e+05
Df Model:                          20

Covariance Type:            nonrobust

======================================================================
======================
                               coef    std err          t      P>|t|
[0.025      0.975]
----------------------------------------------------------------------
----------------------
const                     -600.3963     65.349     -9.187      0.000
-728.489    -472.303
Median_Income            -3.018e+04    644.097    -46.849      0.000
-3.14e+04   -2.89e+04
Median_Age                -385.3993     37.300    -10.332      0.000
-458.512    -312.287
Tot_Rooms                 9.559e+04   8545.582     11.185      0.000
7.88e+04    1.12e+05
```

```
Tot_Bedrooms              -7.085e+04   8646.634     -8.194      0.000
-8.78e+04   -5.39e+04
Population                -5.528e+04   1518.440    -36.407      0.000
-5.83e+04   -5.23e+04
Households                  3.36e+04   3029.159     11.094      0.000
2.77e+04    3.95e+04
Latitude                   -7.75e+04   8528.332     -9.087      0.000
-9.42e+04   -6.08e+04
Longitude                 -2.464e+04   2380.125    -10.353      0.000
-2.93e+04      -2e+04
Distance_to_coast         -1.169e+04    822.861    -14.207      0.000
-1.33e+04   -1.01e+04
Distance_to_LA               -2.4097      0.821     -2.936      0.003
-4.018      -0.801
Distance_to_SanDiego         -1.8097      0.478     -3.789      0.000
-2.746      -0.873
Distance_to_SanJose          12.7554      4.398      2.900      0.004
4.134      21.377
Distance_to_SanFrancisco     -4.0410      5.046     -0.801      0.423
-13.931       5.849
Badroom_Ratio              1.529e+05   3.15e+04      4.849      0.000
9.11e+04    2.15e+05
Value_Income                  0.1252      0.001    122.801      0.000
0.123       0.127
Latitude_LA                   0.0623      0.025      2.490      0.013
0.013       0.111
Latitude_SanDiego             0.0581      0.014      4.065      0.000
0.030       0.086
Longitude_SanJose             0.1063      0.036      2.961      0.003
0.036       0.177
Longitude_SanFrancisco       -0.0332      0.041     -0.804      0.422
-0.114       0.048
LA_SanDiego                -3.09e-07   1.54e-07     -2.011      0.044
-6.1e-07   -7.77e-09
SanJose_SanFrancisco      -3.997e-07   1.05e-07     -3.790      0.000
-6.06e-07   -1.93e-07
======================================================================
========
Omnibus:                     1955.250   Durbin-Watson:
2.005
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
26176.619
Skew:                          -0.072   Prob(JB):
0.00
Kurtosis:                       9.593   Cond. No.
3.24e+15
======================================================================
========
Notes:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
[2] The condition number is large, 3.24e+15. This might indicate that
there are
strong multicollinearity or other numerical problems.
"""

test_data=x_test.join(y_test)
test_data

       Median_Income  Median_Age  Tot_Rooms  Tot_Bedrooms  Population
\
9660          2.4519          22   7.661527      6.070738    6.721426

3033          3.2500          22   6.892642      5.176150    6.113682

3039          4.8266          13   8.418036      6.570883    7.682943

16586         3.4821          16   7.709757      6.129050    7.090077

16389         3.5735          25   7.584773      5.894403    6.943122

...              ...         ...        ...           ...         ...

12755         3.0086          27   7.773174      6.287859    6.761573

16068         3.7361          48   7.687997      6.040255    6.948897

16681         4.4033          15   8.668884      6.829794    7.751475

15478         3.5839           5   8.466110      6.870053    7.910957

9510          3.8958          33   7.070724      5.356586    6.272877


       Households  Latitude  Longitude  Distance_to_coast
Distance_to_LA  \
9660     5.749393     41.31    -121.18          12.393601
847236.324358
3033     5.141664     35.39    -119.11          11.671335
168509.993269
3039     6.510258     35.37    -119.12          11.650626
166991.195552
16586    6.100319     37.75    -121.44          10.458260
501862.376668
16389    5.891644     38.05    -121.25          10.090755
520163.946494
...           ...       ...        ...                ...
...
12755    6.115892     38.61    -121.38          11.039695
```

```
579364.293044
16068    5.940171     37.75    -122.49           8.083626
561420.922514
16681    6.740519     35.13    -120.56           9.100515
243545.558562
15478    6.809039     33.16    -117.15           9.745623
141779.071052
9510     5.384495     39.13    -123.23          10.652182
718742.255629

       ...  Distance_to_SanFrancisco  Badroom_Ratio  Value_Income  \
9660   ...             407551.836660       0.203390     140984.25
3033   ...             397712.344960       0.178862     288925.00
3039   ...             398563.297627       0.157499     705648.92
16586  ...              87181.726267       0.205473     594046.26
16389  ...             108081.355786       0.184037     381649.80
...    ...                       ...            ...           ...
12755  ...             130700.464955       0.226105     381791.34
16068  ...               5808.154770       0.192114    1196299.22
16681  ...             338278.378469       0.158817    1178323.08
15478  ...             701151.886604       0.202526     568048.15
9510   ...             166049.131134       0.179422     560995.20

        Latitude_LA  Latitude_SanDiego  Longitude_SanJose  \
9660   3.499933e+07       4.212630e+07      -5.406774e+07
3033   5.963569e+06       1.229253e+07      -3.930244e+07
3039   5.906479e+06       1.223568e+07      -3.940261e+07
16586  1.894530e+07       2.571797e+07      -7.406693e+06
16389  1.979224e+07       2.660674e+07      -1.182640e+07
...             ...                ...                ...
12755  2.236926e+07       2.926442e+07      -1.804714e+07
16068  2.119364e+07       2.792438e+07      -8.569656e+06
16681  8.555755e+06       1.450052e+07      -3.289148e+07
15478  4.701394e+06       1.638460e+06      -7.416967e+07
9510   2.812438e+07       3.513969e+07      -2.848990e+07

       Longitude_SanFrancisco   LA_SanDiego  SanJose_SanFrancisco  \
9660             -4.938713e+07  8.639781e+11          1.818403e+11
3033             -4.737152e+07  5.853107e+10          1.312322e+11
3039             -4.747686e+07  5.776789e+10          1.318371e+11
16586            -1.058735e+07  3.419041e+11          5.317262e+09
16389            -1.310486e+07  3.637284e+11          1.054197e+10
...                        ...           ...                   ...
12755            -1.586442e+07  4.391287e+11          1.943294e+10
16068            -7.114409e+05  4.152935e+11          4.063506e+08
16681            -4.078284e+07  1.005276e+11          9.228995e+10
15478            -8.213994e+07  7.005408e+09          4.439112e+11
9510             -2.046223e+07  6.454479e+11          3.838937e+10

       Median_House_Value
```

```
9660             57500.0
3033             88900.0
3039            146200.0
16586           170600.0
16389           106800.0
...                  ...
12755           126900.0
16068           320200.0
16681           267600.0
15478           158500.0
9510            144000.0

[3096 rows x 22 columns]
```

```
OLS.score(x_test,y_test)
```

```
0.8589384353527918
```

```
ridge.score(x_test,y_test)
```

```
0.8589384353632443
```

```
lasso.score(x_test,y_test)
```

```
0.8566134110824559
```