California Housing Price Prediction

Description

**Background of Problem Statement :**

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

**Problem Objective :**

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

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

**Domain**: Finance and Housing

**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** :

* Read the “**housing.csv**” file from the folder into the program.
* Print first few rows of this data.
* Extract input (X) and output (Y) data from the dataset.

2. **Handle missing values** :

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

3. **Encode categorical data** :

* Convert categorical column in the dataset to numerical data.

4. **Split the dataset** :

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

5. **Standardize data** :

* Standardize training and test datasets.

6. **Perform Linear Regression** :

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

            [ HINT: Import **mean\_squared\_error** from **sklearn.metrics** ]

7. **Perform Decision Tree Regression** :

* Perform Decision Tree Regression on training data.
* Predict output for test dataset using the fitted model.
* Print root mean squared error from Decision Tree Regression.

8. **Perform Random Forest Regression** :

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

9. **Bonus exercise: Perform Linear Regression with one independent variable** :

* Extract just the median\_income column from the independent variables (from **X\_train** and **X\_test**).
* Perform Linear Regression to predict housing values based on **median\_income**.
* Predict output for test dataset using the fitted model.
* Plot the fitted model for training data as well as for test data to check if the fitted model satisfies the test data.

Dataset Description :

|  |  |
| --- | --- |
| Field | Description |
| longitude | (signed numeric - float) : Longitude value for the block in California, USA |
| latitude | (numeric - float ) : Latitude value for the block in California, USA |
| housing\_median\_age | (numeric - int ) : Median age of the house in the block |
| total\_rooms | (numeric - int ) : Count of the total number of rooms (excluding bedrooms) in all houses in the block |
| total\_bedrooms | (numeric - float ) : Count of the total number of bedrooms in all houses in the block |
| population | (numeric - int ) : Count of the total number of population in the block |
| households | (numeric - int ) : Count of the total number of households in the block |
| median\_income | (numeric - float ) : Median of the total household income of all the houses in the block |
| ocean\_proximity | (numeric - categorical ) : Type of the landscape of the block [ Unique Values : 'NEAR BAY', '<1H OCEAN', 'INLAND', 'NEAR OCEAN', 'ISLAND'  ] |
| median\_house\_value | (numeric - int ) : Median of the household prices of all the houses in the block |

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

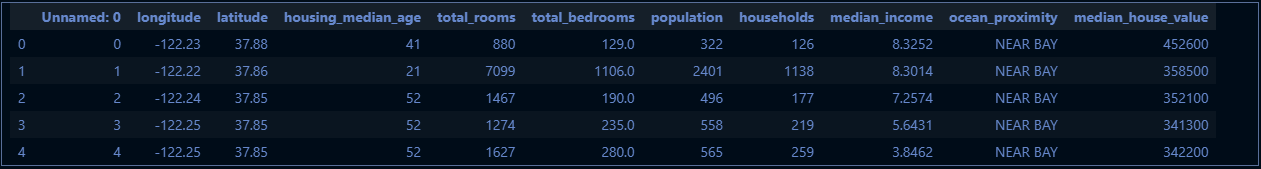
%matplotlib inline

readFile=pd.read\_excel('1553768847\_housing.xlsx')

readFile.to\_csv('housing.csv')

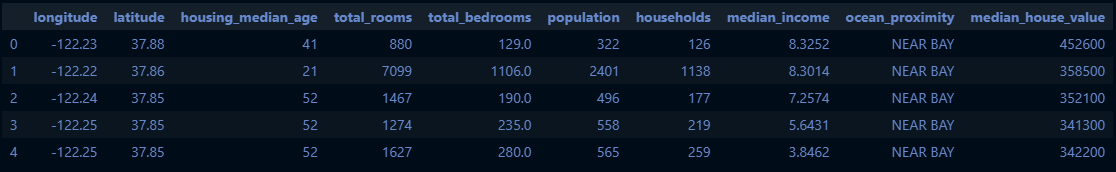
df=pd.read\_csv('housing.csv')

df.head()

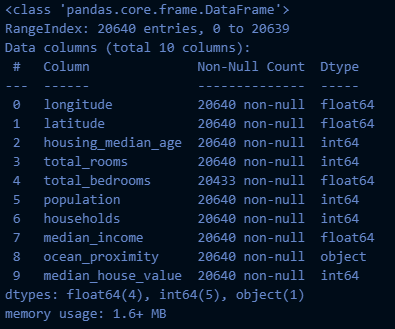
****

df.drop('Unnamed: 0',*axis*=1,*inplace*=True)

df.head()



df.info()



df['ocean\_proximity'].value\_counts()

<1H OCEAN 9136

INLAND 6551

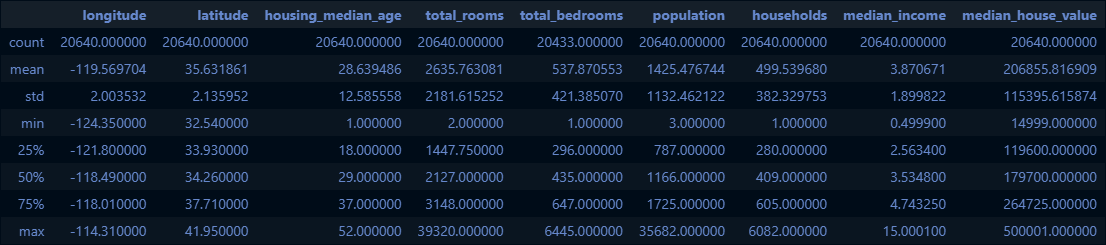
NEAR OCEAN 2658

NEAR BAY 2290

ISLAND 5

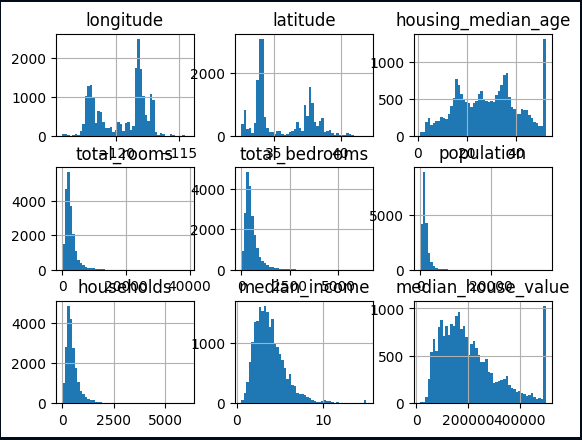
Name: ocean\_proximity, dtype: int64

df.describe()



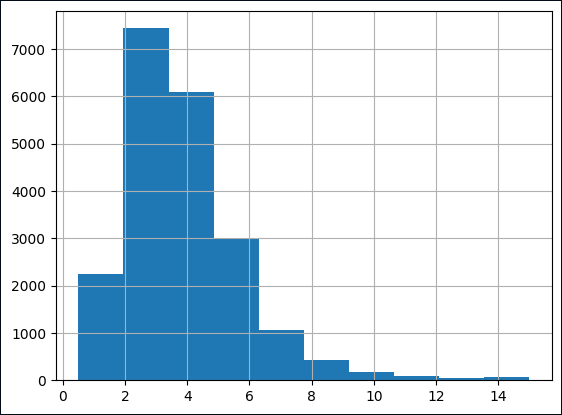
df.hist(*bins*=50)

plt.show()



# median income looks like an imp feature

df['median\_income'].hist()



#dividing the income category to limit the number income category

df['income\_cat']=np.ceil(df['median\_income'])

#putting eerything above 5th category as 5th category

df['income\_cat'].where(df['income\_cat']<5, *other*=5.0, *inplace*=True)

from sklearn.model\_selection import StratifiedShuffleSplit

split=StratifiedShuffleSplit(*n\_splits*=1, *test\_size*=0.2, *random\_state*=29)

for trainIndex, testIndex in split.split(df, df['income\_cat']):

    stratTrainSet=df.loc[trainIndex]

    startTestSet=df.loc[testIndex]

df['income\_cat'].value\_counts()/len(df)

5.0 0.391812

4.0 0.249516

3.0 0.239632

2.0 0.111337

1.0 0.007703

Name: income\_cat, dtype: float64

startTestSet['income\_cat'].value\_counts()/len(startTestSet)

5.0 0.391715

4.0 0.249516

3.0 0.239583

2.0 0.111434

1.0 0.007752

Name: income\_cat, dtype: float64

as seen above the proportions are maintained in the test set using stratified sampling

[why stratified?] : because the feature-space are less and also because its a mid-sized dataset & we don't want to miss out any class

#experimenting: with random sampling now

from sklearn.model\_selection import train\_test\_split

trainSet, testSet=train\_test\_split(df, *test\_size*=0.2, *random\_state*=29)

*def* income\_cat\_proportions(*data*):

    return *data*['income\_cat'].value\_counts()/len(*data*)

comparingProps=pd.DataFrame({

    'Overall Props':income\_cat\_proportions(df),

    'Random':income\_cat\_proportions(testSet),

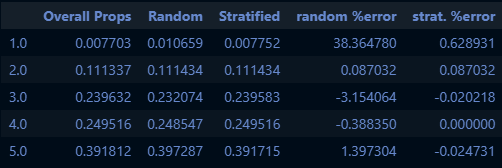
    'Stratified':income\_cat\_proportions(startTestSet)

}).sort\_index()

comparingProps['random %error']=100\*comparingProps['Random']/comparingProps['Overall Props']-100

comparingProps['strat. %error']=100\*comparingProps['Stratified']/comparingProps['Overall Props']-100

comparingProps



for items in (stratTrainSet,startTestSet):

    items.drop('income\_cat',*axis*=1,*inplace*=True)

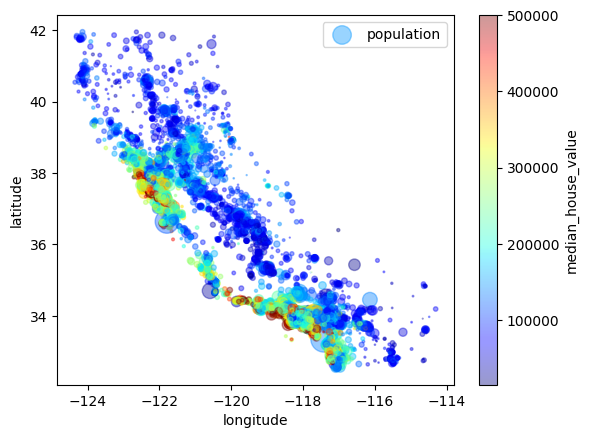
df=stratTrainSet.copy()

df.plot(*kind*='scatter',*x*='longitude',*y*='latitude',*alpha*=0.4,

*s*=df['population']/100,*label*='population',

*c*='median\_house\_value',*cmap*=plt.get\_cmap('jet'),*sharex*=False)

plt.legend()



import matplotlib.image as mpimg

ax=df.plot(*kind*='scatter',*x*='longitude',*y*='latitude',*alpha*=0.4,

*s*=df['population']/100,*label*='population',

*c*='median\_house\_value',*cmap*=plt.get\_cmap('jet'),*sharex*=False)

calImg=mpimg.imread('california.png')

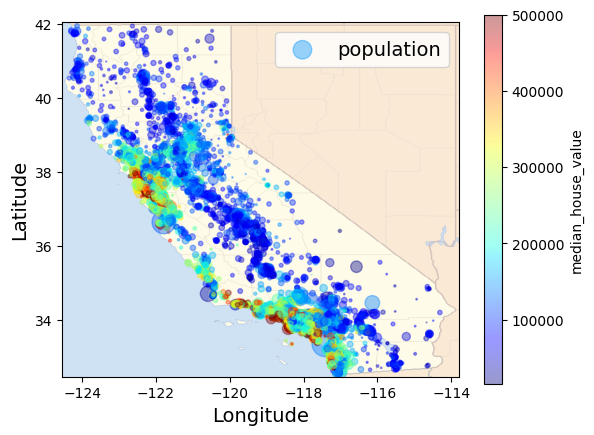
plt.imshow(calImg,*extent*=[-124.55,-113.8,32.45,42.05],*alpha*=0.5,*cmap*=plt.get\_cmap('jet'))

plt.xlabel('Longitude',*fontsize*=14)

plt.ylabel('Latitude',*fontsize*=14)

plt.legend(*fontsize*=14)

plt.show()



Looking or Correlations

(Pearson's Distance Correlation equation)

corrMatrix=df.corr()

corrMatrix['median\_house\_value'].sort\_values(*ascending*=False)

median\_house\_value 1.000000

median\_income 0.688886

total\_rooms 0.138943

housing\_median\_age 0.107011

households 0.066467

total\_bedrooms 0.050577

population -0.023908

longitude -0.040800

latitude -0.149529

Name: median\_house\_value, dtype: float64

#other approach it to use the scatter plot in a A vs B fashiion

#problem with this is that(for N features, there will be N^2 plots)

impAtrbutes=['median\_house\_value','median\_income','total\_rooms','housing\_median\_age']

from pandas.plotting import scatter\_matrix

scatter\_matrix(df[impAtrbutes],*figsize*=(12,8))

array([[<AxesSubplot:xlabel='median\_house\_value', ylabel='median\_house\_value'>,

<AxesSubplot:xlabel='median\_income', ylabel='median\_house\_value'>,

<AxesSubplot:xlabel='total\_rooms', ylabel='median\_house\_value'>,

<AxesSubplot:xlabel='housing\_median\_age', ylabel='median\_house\_value'>],

[<AxesSubplot:xlabel='median\_house\_value', ylabel='median\_income'>,

<AxesSubplot:xlabel='median\_income', ylabel='median\_income'>,

<AxesSubplot:xlabel='total\_rooms', ylabel='median\_income'>,

<AxesSubplot:xlabel='housing\_median\_age', ylabel='median\_income'>],

[<AxesSubplot:xlabel='median\_house\_value', ylabel='total\_rooms'>,

<AxesSubplot:xlabel='median\_income', ylabel='total\_rooms'>,

<AxesSubplot:xlabel='total\_rooms', ylabel='total\_rooms'>,

<AxesSubplot:xlabel='housing\_median\_age', ylabel='total\_rooms'>],

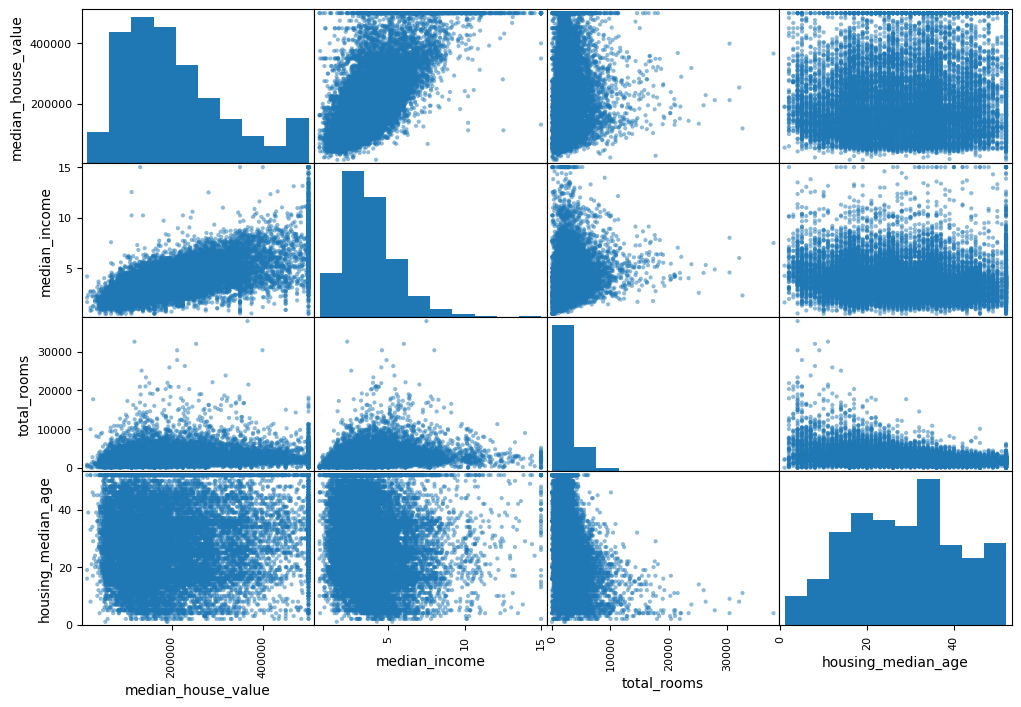
[<AxesSubplot:xlabel='median\_house\_value', ylabel='housing\_median\_age'>,

<AxesSubplot:xlabel='median\_income', ylabel='housing\_median\_age'>,

<AxesSubplot:xlabel='total\_rooms', ylabel='housing\_median\_age'>,

<AxesSubplot:xlabel='housing\_median\_age', ylabel='housing\_median\_age'>]],

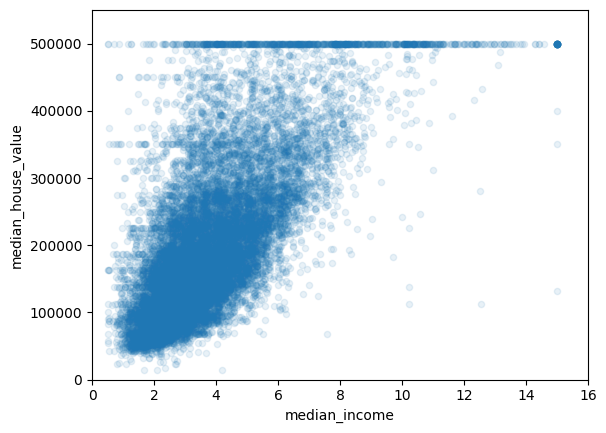
dtype=object)



df.plot(*kind*='scatter',*x*='median\_income',*y*='median\_house\_value',*alpha*=0.1)

plt.axis([0,16,0,550000])

(0.0, 16.0, 0.0, 550000.0)



Feature Engineering

df['bedrooms\_per\_room']=df['total\_bedrooms']/df['total\_rooms']

df['population\_per\_household']=df['population']/df['households']

df['rooms\_per\_household']=df['total\_rooms']/df['households']

corrMatrix=df.corr()

corrMatrix['median\_house\_value'].sort\_values(*ascending*=False)

median\_house\_value 1.000000

median\_income 0.688886

rooms\_per\_household 0.145112

total\_rooms 0.138943

housing\_median\_age 0.107011

households 0.066467

total\_bedrooms 0.050577

population\_per\_household -0.021886

population -0.023908

longitude -0.040800

latitude -0.149529

bedrooms\_per\_room -0.262760

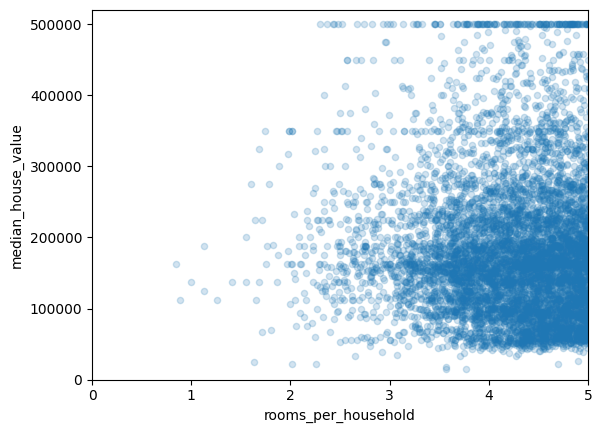
Name: median\_house\_value, dtype: float64

Observation: the ne bedrooms\_per\_room is highly correlated but in a reciprocative ay to the median\_house\_value, So the houses with lesser bedroom/room ratio will tend to be more expensive.

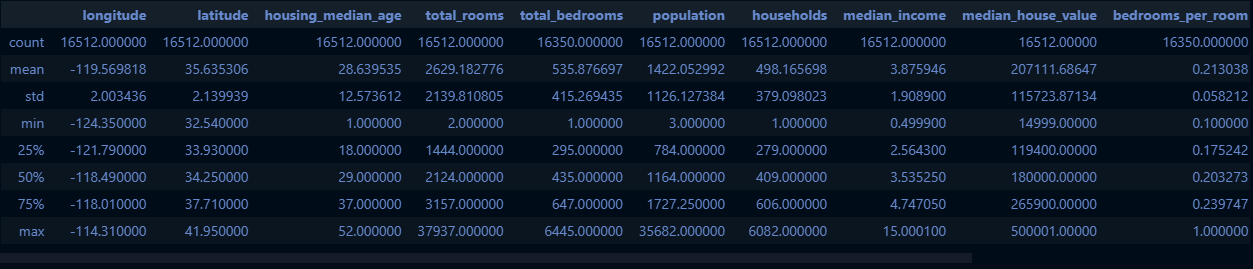
df.plot(*kind*='scatter',*x*='rooms\_per\_household',*y*='median\_house\_value',*alpha*=0.2)

plt.axis([0,5,0,520000])

plt.show()



df.describe()



Preparing the data for ML algos

df=stratTrainSet.drop('median\_house\_value',*axis*=1)

dfLabels=stratTrainSet['median\_house\_value'].copy()

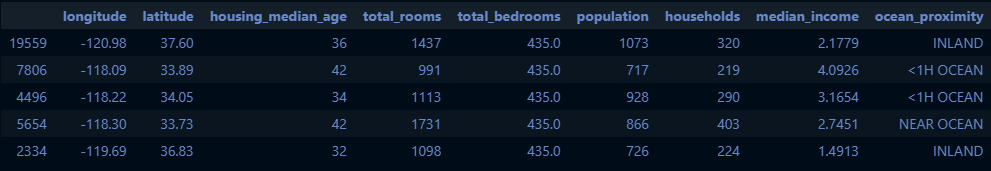
#when calculating imputing value on your own

sampleIncompleteRows=df[df.isnull().any(*axis*=1)].head()

median=df['total\_bedrooms'].median()

sampleIncompleteRows['total\_bedrooms'].fillna(median,*inplace*=True)

sampleIncompleteRows



#When using Scikit-Learns's Imputer class

from sklearn.impute import SimpleImputer

imputer=SimpleImputer(*strategy*='median')

dfNum=df.drop('ocean\_proximity',*axis*=1)

imputer.fit(dfNum)

SimpleImputer(strategy='median')

#Imputer basically computes across all the attributes,

# so if you wanna see this across all the attributes!

imputer.statistics\_

array([-118.49 , 34.25 , 29. , 2124. , 435. ,

1164. , 409. , 3.53525])

dfNum.median().values

array([-118.49 , 34.25 , 29. , 2124. , 435. ,

1164. , 409. , 3.53525])

x=imputer.transform(dfNum)

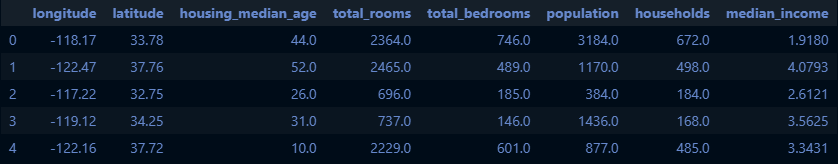
dftr=pd.DataFrame(x, *columns*=dfNum.columns)

#cross check for missing value

dftr[dftr.isnull().any(*axis*=1)]



dftr.head()



Handling categorical values

dfcat=df['ocean\_proximity']

dfcat.head(10)

8252 NEAR OCEAN

15985 NEAR BAY

14353 NEAR OCEAN

20324 NEAR OCEAN

636 NEAR BAY

7866 <1H OCEAN

16697 <1H OCEAN

5396 <1H OCEAN

14627 NEAR OCEAN

7985 <1H OCEAN

Name: ocean\_proximity, dtype: object

#using pandas's on factorize() method to convert them into categorical features

dfcatEncoded, dfCategories=dfcat.factorize()

dfcatEncoded[:10]

array([0, 1, 0, 0, 1, 2, 2, 2, 0, 2], dtype=int64)

dfCategories

Index(['NEAR OCEAN', 'NEAR BAY', '<1H OCEAN', 'INLAND', 'ISLAND'], dtype='object')

#using Scikit-Learn's OneHotEncoder

from sklearn.preprocessing import OneHotEncoder

encoder=OneHotEncoder()

dfcat1Hot=encoder.fit\_transform(dfcatEncoded.reshape(1,-1))

dfcat1Hot

<1x16512 sparse matrix of type '<class 'numpy.float64'>'

with 16512 stored elements in Compressed Sparse Row format>

#since 1 hot encoder returns a sparse matrix, need to change it to

#it to a dense array

dfcat1Hot.toarray()

Custom Transformations

from sklearn.base import BaseEstimator, TransformerMixin

rooms\_ix, bedrooms\_ix, population\_ix, household\_ix = 3, 4, 5, 6

*class* CombinedAttributesAdder(BaseEstimator, TransformerMixin):

*def* \_\_init\_\_(*self*, *add\_bedrooms\_per\_room* = True):

*self*.add\_bedrooms\_per\_room = *add\_bedrooms\_per\_room*

*def* fit(*self*, *X*, *y*=None):

        return *self* # nothing to do here

*def* transform(*self*, *X*, *y*=None):

        rooms\_per\_household = *X*[:, rooms\_ix] / *X*[:, household\_ix]

        population\_per\_household = *X*[:, population\_ix] / *X*[:, household\_ix]

        if *self*.add\_bedrooms\_per\_room:

            bedrooms\_per\_room = *X*[:, bedrooms\_ix] / *X*[:, rooms\_ix]

            return np.c\_[*X*, rooms\_per\_household, population\_per\_household, bedrooms\_per\_room]

        else:

            return np.c\_[*X*, rooms\_per\_household, population\_per\_household]

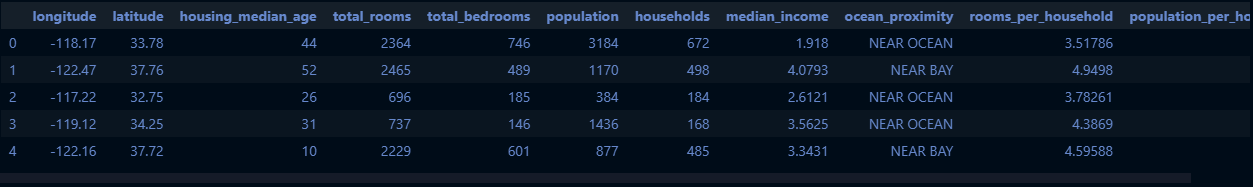
attr\_adder = CombinedAttributesAdder(*add\_bedrooms\_per\_room*=False)

housing\_extra\_attribs = attr\_adder.transform(df.values)

housing\_extra\_attribs = pd.DataFrame(housing\_extra\_attribs, *columns*=list(df.columns)+["rooms\_per\_household",

                                                                                           "population\_per\_household"])

housing\_extra\_attribs.head()



## Setting up Pipeline for all the preprocessings

from sklearn.pipeline import Pipeline

from sklearn.preprocessing import StandardScaler

num\_pipeline = Pipeline([

    ("imputer", SimpleImputer(*strategy*="median")),

    ("attribs\_adder", CombinedAttributesAdder()),

    ("std\_scaler", StandardScaler())

])

housing\_num\_tr = num\_pipeline.fit\_transform(dfNum)

housing\_num\_tr

array([[ 0.69873008, -0.86701651, 1.22167997, ..., -0.73330055,

0.14483584, 1.61597814],

[-1.447648 , 0.99290573, 1.85795236, ..., -0.18789943,

-0.06398979, -0.24145193],

[ 1.17292989, -1.34835317, -0.20993289, ..., -0.63246142,

-0.08693301, 0.82725701],

...,

[ 0.60389012, -0.71747503, 1.85795236, ..., -0.26644093,

-0.08428764, 0.05353069],

[ 0.77360373, -0.74551405, -0.13039885, ..., 0.12330867,

0.04687834, -0.57263353],

[ 1.16294673, -1.32498732, 0.02866925, ..., -0.54785369,

-0.11247573, 0.75516148]])

*class* DataFrameSelector(BaseEstimator, TransformerMixin):

*def* \_\_init\_\_(*self*, *attribute\_names*):

*self*.attibute\_names = *attribute\_names*

*def* fit(*self*, *X*, *y*=None):

        return *self* # do nothing

*def* transform(*self*, *X*, *y*=None):

        return *X*[*self*.attibute\_names].values

# complete Pipeline

num\_attribs = list(dfNum.columns)

cat\_attribs = ["ocean\_proximity"]

num\_pipeline = Pipeline([

    ("selector", DataFrameSelector(num\_attribs)),

    ("imputer", SimpleImputer(*strategy*="median")),

    ("attribs\_adder", CombinedAttributesAdder()),

    ("std\_scaler", StandardScaler())

])

cat\_pipeline =Pipeline([

    ("selector", DataFrameSelector(cat\_attribs)),

    ("cat\_encoder", OneHotEncoder(*sparse*=False))

])

from sklearn.pipeline import FeatureUnion

full\_pipeline = FeatureUnion(*transformer\_list*=[

    ('num\_pipeline', num\_pipeline),

    ('cat\_pipeline', cat\_pipeline)

])

housing\_prepared = full\_pipeline.fit\_transform(df)

housing\_prepared

array([[ 0.69873008, -0.86701651, 1.22167997, ..., 0. ,

0. , 1. ],

[-1.447648 , 0.99290573, 1.85795236, ..., 0. ,

1. , 0. ],

[ 1.17292989, -1.34835317, -0.20993289, ..., 0. ,

0. , 1. ],

...,

[ 0.60389012, -0.71747503, 1.85795236, ..., 0. ,

0. , 0. ],

[ 0.77360373, -0.74551405, -0.13039885, ..., 0. ,

0. , 0. ],

[ 1.16294673, -1.32498732, 0.02866925, ..., 0. ,

0. , 1. ]])

##Selecting & Training Models

from sklearn.linear\_model import LinearRegression

lin\_reg = LinearRegression()

lin\_reg.fit(housing\_prepared, dfLabels)

LinearRegression()

# trying the full pipeline on a few training instances

some\_data = df.iloc[:5]

some\_labels = dfLabels.iloc[:5]

some\_data\_prepared = full\_pipeline.transform(some\_data)

print("Prediction: ", lin\_reg.predict(some\_data\_prepared))

print("Actual Labels: ", list(some\_labels))

Prediction: [137518.96213291 276982.04670464 187447.39591053 188060.79338636

218216.25795436]

Actual Labels: [147500, 306700, 125000, 194100, 137500]

from sklearn.metrics import mean\_squared\_error

housing\_predictions = lin\_reg.predict(housing\_prepared)

lin\_mse = mean\_squared\_error(dfLabels, housing\_predictions)

lin\_rmse = np.sqrt(lin\_mse)

lin\_rmse

68553.83107772926

from sklearn.tree import DecisionTreeRegressor

tree\_reg = DecisionTreeRegressor()

tree\_reg.fit(housing\_prepared, dfLabels)

DecisionTreeRegressor()

housing\_predictions = tree\_reg.predict(housing\_prepared)

tree\_mse = mean\_squared\_error(dfLabels, housing\_predictions)

tree\_rmse = np.sqrt(tree\_mse)

tree\_rmse

0.0

##Cross Validation

from sklearn.model\_selection import cross\_val\_score

scores = cross\_val\_score(tree\_reg, housing\_prepared, dfLabels, *cv*=10, *scoring*="neg\_mean\_squared\_error")

tree\_rmse\_scores = np.sqrt(-scores)

*def* display\_scores(*scores*):

    print("scores: ", *scores*)

    print("mean: ", *scores*.mean())

    print("std deviation: ", *scores*.std())

display\_scores(tree\_rmse\_scores)

scores: [70698.92714492 71259.6606053 67931.77802544 71863.5482105

75649.6764588 71384.33536566 71537.14821151 70476.95536812

73628.79977501 67422.29962176]

mean: 71185.31287870223

std deviation: 2280.662150502712

lin\_scores = cross\_val\_score(lin\_reg, housing\_prepared, dfLabels, *cv*=10, *scoring*="neg\_mean\_squared\_error")

lin\_rmse\_scores = np.sqrt(-lin\_scores)

display\_scores(lin\_rmse\_scores)

scores: [69475.53714737 73077.11070203 66093.12404915 68452.18631484

68250.44881654 69607.74820596 67898.76059539 67769.37140143

70520.07131809 67226.74503153]

mean: 68837.11035823292

std deviation: 1855.9923048358821

from sklearn.ensemble import RandomForestRegressor

forest\_reg = RandomForestRegressor(*random\_state*=29)

forest\_reg.fit(housing\_prepared, dfLabels)

RandomForestRegressor(random\_state=29)

housing\_pred = forest\_reg.predict(housing\_prepared)

forest\_scores = cross\_val\_score(lin\_reg, housing\_prepared, dfLabels, *cv*=10, *scoring*="neg\_mean\_squared\_error")

forest\_rmse\_scores = np.sqrt(-forest\_scores)

display\_scores(forest\_rmse\_scores)

scores: [69475.53714737 73077.11070203 66093.12404915 68452.18631484

68250.44881654 69607.74820596 67898.76059539 67769.37140143

70520.07131809 67226.74503153]

mean: 68837.11035823292

std deviation: 1855.9923048358821

##Fine Tuning Model

from sklearn.model\_selection import GridSearchCV

param\_grid = [

    {'n\_estimators': [3, 10, 30], 'max\_features': [2, 4, 6, 8]},

    {'bootstrap': [False], 'n\_estimators': [3, 10], 'max\_features': [2, 3, 4]}

]

rf\_reg = RandomForestRegressor()

grid\_search = GridSearchCV(rf\_reg, param\_grid, *cv*=5, *scoring*="neg\_mean\_squared\_error")

grid\_search.fit(housing\_prepared, dfLabels)

GridSearchCV(cv=5, estimator=RandomForestRegressor(),

param\_grid=[{'max\_features': [2, 4, 6, 8],

'n\_estimators': [3, 10, 30]},

{'bootstrap': [False], 'max\_features': [2, 3, 4],

'n\_estimators': [3, 10]}],

scoring='neg\_mean\_squared\_error')

# to get the best combination of hyperparameters

grid\_search.best\_params\_

{'max\_features': 6, 'n\_estimators': 30}

# to get the best estimators directly

grid\_search.best\_estimator\_

RandomForestRegressor(max\_features=6, n\_estimators=30)

cv\_res = grid\_search.cv\_results\_

for mean\_score, params in zip(cv\_res["mean\_test\_score"], cv\_res["params"]):

    print(np.sqrt(-mean\_score), params)

64734.404653588914 {'max\_features': 2, 'n\_estimators': 3}

55430.99110297355 {'max\_features': 2, 'n\_estimators': 10}

52512.942398883526 {'max\_features': 2, 'n\_estimators': 30}

59890.12082324734 {'max\_features': 4, 'n\_estimators': 3}

52368.5437300643 {'max\_features': 4, 'n\_estimators': 10}

50300.40157565945 {'max\_features': 4, 'n\_estimators': 30}

58456.522503967826 {'max\_features': 6, 'n\_estimators': 3}

52466.51011602238 {'max\_features': 6, 'n\_estimators': 10}

49933.37944466548 {'max\_features': 6, 'n\_estimators': 30}

58770.68674914 {'max\_features': 8, 'n\_estimators': 3}

52045.0128349212 {'max\_features': 8, 'n\_estimators': 10}

50398.00700031304 {'max\_features': 8, 'n\_estimators': 30}

62267.88941437042 {'bootstrap': False, 'max\_features': 2, 'n\_estimators': 3}

54396.00864435611 {'bootstrap': False, 'max\_features': 2, 'n\_estimators': 10}

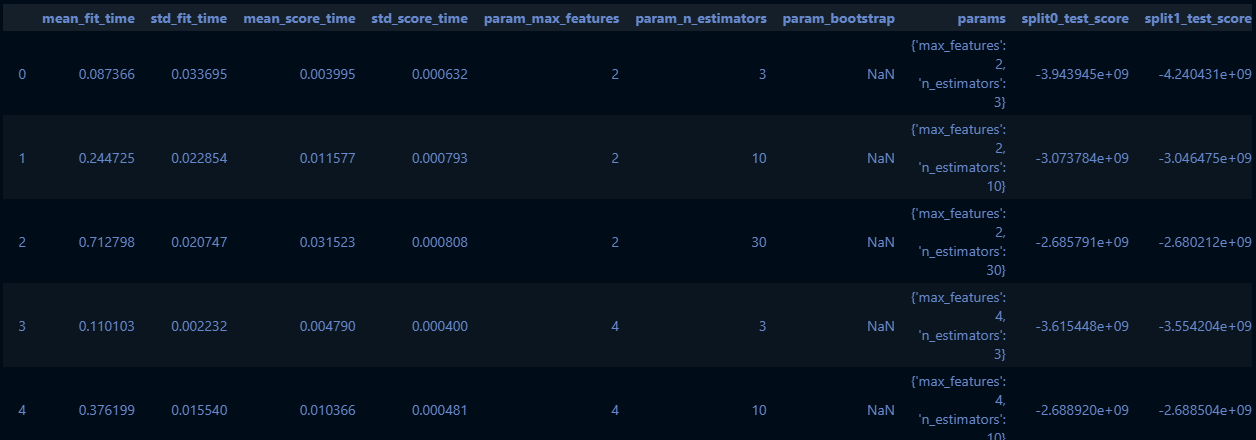
59542.86080849918 {'bootstrap': False, 'max\_features': 3, 'n\_estimators': 3}

53014.23717448815 {'bootstrap': False, 'max\_features': 3, 'n\_estimators': 10}

58285.004273205894 {'bootstrap': False, 'max\_features': 4, 'n\_estimators': 3}

52090.67392576495 {'bootstrap': False, 'max\_features': 4, 'n\_estimators': 10}

pd.DataFrame(grid\_search.cv\_results\_)



from sklearn.model\_selection import RandomizedSearchCV

from scipy.stats import randint

params\_distibs = {

    'n\_estimators': randint(*low*=1, *high*=200),

    'max\_features': randint(*low*=1, *high*=8),

}

rf\_reg = RandomForestRegressor(*random\_state*=29)

rnd\_search = RandomizedSearchCV(rf\_reg, *param\_distributions*=params\_distibs, *n\_iter*=10,

*cv*=5, *scoring*="neg\_mean\_squared\_error", *random\_state*=29)

rnd\_search.fit(housing\_prepared, dfLabels)

RandomizedSearchCV(cv=5, estimator=RandomForestRegressor(random\_state=29),

param\_distributions={'max\_features': <scipy.stats.\_distn\_infrastructure.rv\_frozen object at 0x0000019257A75198>,

'n\_estimators': <scipy.stats.\_distn\_infrastructure.rv\_frozen object at 0x0000019257A756D8>},

random\_state=29, scoring='neg\_mean\_squared\_error')

cvres = rnd\_search.cv\_results\_

for mean\_score, params in zip(cvres["mean\_test\_score"], cvres["params"]):

    print(np.sqrt(-mean\_score), params)

49460.90418316759 {'max\_features': 6, 'n\_estimators': 116}

50345.89723680561 {'max\_features': 5, 'n\_estimators': 35}

54430.71181834486 {'max\_features': 1, 'n\_estimators': 97}

51593.2492404671 {'max\_features': 2, 'n\_estimators': 114}

54448.606511279846 {'max\_features': 1, 'n\_estimators': 98}

49465.0294588746 {'max\_features': 7, 'n\_estimators': 95}

54228.15380865881 {'max\_features': 1, 'n\_estimators': 156}

49414.42736688848 {'max\_features': 6, 'n\_estimators': 149}

49359.28565539728 {'max\_features': 7, 'n\_estimators': 152}

54219.66631779875 {'max\_features': 1, 'n\_estimators': 165}

feature\_importances = grid\_search.best\_estimator\_.feature\_importances\_

feature\_importances

array([7.62365844e-02, 7.17182103e-02, 4.10034831e-02, 1.73576248e-02,

1.62534623e-02, 1.70999319e-02, 1.52072315e-02, 3.07877048e-01,

7.91097946e-02, 1.08367213e-01, 7.53742423e-02, 1.42386074e-02,

1.49155218e-01, 2.28925783e-05, 3.45159351e-03, 7.52686288e-03])

extra\_attribs = ["rooms\_per\_hhold", "pop\_per\_hhold", "bedrooms\_per\_room"]

cat\_encoder = cat\_pipeline.named\_steps["cat\_encoder"]

cat\_one\_hot\_attribs = list(cat\_encoder.categories\_[0])

attributes = num\_attribs + extra\_attribs + cat\_one\_hot\_attribs

sorted(zip(feature\_importances, attributes), *reverse*=True)

[(0.307877048062816, 'median\_income'),

(0.14915521766440826, 'INLAND'),

(0.10836721279805103, 'pop\_per\_hhold'),

(0.07910979457376982, 'rooms\_per\_hhold'),

(0.07623658442996901, 'longitude'),

(0.07537424225496162, 'bedrooms\_per\_room'),

(0.07171821025092595, 'latitude'),

(0.041003483111690185, 'housing\_median\_age'),

(0.0173576248057527, 'total\_rooms'),

(0.01709993188270225, 'population'),

(0.016253462295576872, 'total\_bedrooms'),

(0.015207231487153187, 'households'),

(0.014238607413183477, '<1H OCEAN'),

(0.007526862880319198, 'NEAR OCEAN'),

(0.003451593510468284, 'NEAR BAY'),

(2.2892578252163994e-05, 'ISLAND')]

final\_model = grid\_search.best\_estimator\_

X\_test = startTestSet.drop("median\_house\_value", *axis*=1)

y\_test = startTestSet["median\_house\_value"].copy()

X\_test\_prepared = full\_pipeline.transform(X\_test)

final\_predictions = final\_model.predict(X\_test\_prepared)

final\_mse = mean\_squared\_error(y\_test, final\_predictions)

final\_rmse = np.sqrt(final\_mse)