**Code**

#!/usr/bin/env python

# coding: utf-8

## Ansari Real Estate - Price Predictor

# In[1]:

import pandas as pd

# In[2]:

housing = pd.read\_csv("data.csv")

# In[3]:

housing.head()

# In[4]:

housing.info()

# In[5]:

housing['CHAS'].value\_counts()

# In[6]:

housing.describe()

# In[7]:

get\_ipython().run\_line\_magic('matplotlib', 'inline')

# In[8]:

import matplotlib.pyplot as plt

housing.hist(bins=50, figsize=(20, 15))

## Train-Test Splitting

# In[9]:

# For learning purpose

import numpy as np

def split\_train\_test(data, test\_ratio):

    np.random.seed(42)

    shuffled = np.random.permutation(len(data))

    print(shuffled)

    test\_set\_size = int(len(data) \* test\_ratio)

    test\_indices = shuffled[:test\_set\_size]

    train\_indices = shuffled[test\_set\_size:]

    return data.iloc[train\_indices], data.iloc[test\_indices]

# In[10]:

# train\_set, test\_set = split\_train\_test(housing, 0.2)

# In[11]:

print(f"Rows in train set: {len(train\_set)}\nRows in test set: {len(test\_set)}\n")

# In[12]:

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

train\_set, test\_set  = train\_test\_split(housing, test\_size=0.2, random\_state=42)

print(f"Rows in train set: {len(train\_set)}\nRows in test set: {len(test\_set)}\n")

# In[13]:

from sklearn.model\_selection import StratifiedShuffleSplit

split = StratifiedShuffleSplit(n\_splits=1, test\_size=0.2, random\_state=42)

for train\_index, test\_index in split.split(housing, housing['CHAS']):

    strat\_train\_set = housing.loc[train\_index]

    strat\_test\_set = housing.loc[test\_index]

# In[14]:

strat\_test\_set['CHAS'].value\_counts()

# In[15]:

strat\_train\_set['CHAS'].value\_counts()

# In[16]:

# 95/7

# In[17]:

# 376/28

# In[18]:

housing = strat\_train\_set.copy()

## Looking for Correlations

# In[19]:

corr\_matrix = housing.corr()

corr\_matrix['MEDV'].sort\_values(ascending=False)

# In[20]:

# from pandas.plotting import scatter\_matrix

# attributes = ["MEDV", "RM", "ZN", "LSTAT"]

# scatter\_matrix(housing[attributes], figsize = (12,8))

# In[21]:

housing.plot(kind="scatter", x="RM", y="MEDV", alpha=0.8)

## Trying out Attribute combinations

# In[22]:

housing["TAXRM"] = housing['TAX']/housing['RM']

# In[23]:

housing.head()

# In[24]:

corr\_matrix = housing.corr()

corr\_matrix['MEDV'].sort\_values(ascending=False)

# In[25]:

housing.plot(kind="scatter", x="TAXRM", y="MEDV", alpha=0.8)

# In[26]:

housing = strat\_train\_set.drop("MEDV", axis=1)

housing\_labels = strat\_train\_set["MEDV"].copy()

# In[28]:

a = housing.dropna(subset=["RM"]) #Option 1

a.shape

# In[29]:

housing.drop("RM", axis=1).shape

# In[30]:

median = housing["RM"].median() # Compute median for Option 3

# In[31]:

housing["RM"].fillna(median)

# In[32]:

housing.shape

# In[33]:

housing.describe() # before we started filling missing attributes

# In[34]:

from sklearn.impute import SimpleImputer

imputer = SimpleImputer(strategy="median")

imputer.fit(housing)

# In[35]:

imputer.statistics\_

# In[36]:

X = imputer.transform(housing)

# In[37]:

housing\_tr = pd.DataFrame(X, columns=housing.columns)

# In[38]:

housing\_tr.describe()

## Creating a Pipeline

# In[39]:

from sklearn.pipeline import Pipeline

from sklearn.preprocessing import StandardScaler

my\_pipeline = Pipeline([

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

    #     ..... add as many as you want in your pipeline

    ('std\_scaler', StandardScaler()),

])

# In[40]:

housing\_num\_tr = my\_pipeline.fit\_transform(housing)

# In[41]:

housing\_num\_tr.shape

# ## Selecting a desired model for Dragon Real Estates

# In[42]:

from sklearn.linear\_model import LinearRegression

from sklearn.tree import DecisionTreeRegressor

from sklearn.ensemble import RandomForestRegressor

# model = LinearRegression()

# model = DecisionTreeRegressor()

model = RandomForestRegressor()

model.fit(housing\_num\_tr, housing\_labels)

# In[43]:

some\_data = housing.iloc[:5]

# In[44]:

some\_labels = housing\_labels.iloc[:5]

# In[45]:

prepared\_data = my\_pipeline.transform(some\_data)

# In[46]:

model.predict(prepared\_data)

# In[47]:

list(some\_labels)

# ## Evaluating the model

# In[48]:

from sklearn.metrics import mean\_squared\_error

housing\_predictions = model.predict(housing\_num\_tr)

mse = mean\_squared\_error(housing\_labels, housing\_predictions)

rmse = np.sqrt(mse)

# In[49]:

rmse

# ## Using better evaluation technique - Cross Validation

# In[50]:

# 1 2 3 4 5 6 7 8 9 10

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

scores = cross\_val\_score(model, housing\_num\_tr, housing\_labels, scoring="neg\_mean\_squared\_error", cv=10)

rmse\_scores = np.sqrt(-scores)

# In[51]:

rmse\_scores

# In[52]:

def print\_scores(scores):

    print("Scores:", scores)

    print("Mean: ", scores.mean())

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

# In[53]:

print\_scores(rmse\_scores)

# ## Saving the model

# In[54]:

from joblib import dump, load

dump(model, 'Dragon.joblib')

# ## Testing the model on test data

# In[55]:

X\_test = strat\_test\_set.drop("MEDV", axis=1)

Y\_test = strat\_test\_set["MEDV"].copy()

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

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

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

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

# print(final\_predictions, list(Y\_test))

# In[56]:

final\_rmse

# In[57]:

import matplotlib.pyplot as plt

housing.hist(bins=50, figsize=(20, 15))

**Model Usage:**

#!/usr/bin/env python

# coding: utf-8

# In[1]:

from joblib import dump, load

import numpy as np

model = load('Ansari.joblib')

features = np.array([[-5.43942006, 4.12628155, -1.6165014, -0.67288841, -1.42262747,

       -11.44443979304, -49.31238772,  7.61111401, -26.0016879 , -0.5778192 ,

       -0.97491834,  0.41164221, -66.86091034]])

model.predict(features)