# -\*- coding: utf-8 -\*-

"""

Created on Fri Sep 18 19:38:49 2022

@author: SABIHA

"""

import pandas as pd

housing = pd.read\_csv("F:\datasets\housing\_boston.csv")

housing.head()

housing.info ()

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

housing.describe()

#%matplotlib inline

# # For plotting histogram

import matplotlib.pyplot as plt

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

#TEST TRAIN SPLITTING

# 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]

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")

#for actual split with equal no. of test pattern and train pattern ,so that like in chas we get proper 0 & 1 to both test and train set

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]

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

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

#Check both train and test ratio

95/7

376/28

# 95/7

#Output 13.571428571428571

#376/28

#Output: 13.428571428571429

#LOOKING FOR CORRELATION

corr\_matrix = housing.corr()

corr\_matrix

# GRAPHS ANALYSIS

from pandas.plotting import scatter\_matrix

attributes=["medv","rm","zn","lstat"]

scatter\_matrix(housing[attributes],figsize=(10,6))

corr\_matrix

housing.plot(kind="scatter", x="rm", y="medv", alpha=0.8)#most suitable relation we get

housing.plot(kind="scatter", x="zn", y="medv", alpha=0.8)

housing.plot(kind="scatter",x="chas", y="medv", alpha=0.8)

housing.plot(kind="scatter",x="crim",y="medv", alpha= 0.8)

housing.plot(kind="scatter",x="indus",y="medv", alpha= 0.8)

housing.plot(kind="scatter",x="lstat",y="medv", alpha= 0.8)

#histogram plot

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

import seaborn as sns

sns.distplot(housing['chas'])

print("The skewness of chas is {}".format(housing['chas'].skew()))

sns.distplot(housing['rm'])

print("The skewness of rm is {}".format(housing['rm'].skew()))

sns.distplot(housing['crim'])

print("The skewness of scatter is {}".format(housing['crim'].skew()))

#heat maplot

import matplotlib.pyplot as plt #plotting, visualizing

from sklearn import model\_selection

import seaborn as sns

corr\_matrix = housing.corr()

fig=plt.figure(figsize=(12,9))

sns.heatmap(corr\_matrix,vmax=.8,square=True)

#TRYING OUT ATTRIBUTE COMBINATION

housing.head()

corr\_matrix = housing.corr()

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

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

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

housing\_labels

#MISSING ATTRIBUTES

# To take care of missing attributes, you have three options:

# 1. Get rid of the missing data points

# 2. Get rid of the whole attribute

# 3. Set the value to some value(0, mean or median)

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

a.shape

# Note that the original housing dataframe will remain unchanged

housing.drop("rm", axis=1).shape # Option 2

# Note that there is no RM column and also note that the original housing dataframe will remain unchanged

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

median

housing["rm"].fillna(median) # Option 3

# Note that the original housing dataframe will remain unchanged

housing.shape

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

from sklearn.impute import SimpleImputer

imputer = SimpleImputer(strategy="median")

imputer.fit(housing)

imputer.statistics\_

X = imputer.transform(housing)

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

housing\_tr.describe

#Scikit-learn Design¶

#Primarily, three types of objects

#1)Estimators - It estimates some parameter based on a dataset. Eg. imputer. It has a fit method and transform method. Fit method - Fits the dataset and calculates internal parameters

#2)Transformers - transform method takes input and returns output based on the learnings from fit(). It also has a convenience function called fit\_transform() which fits and then transforms.

#3)Predictors - LinearRegression model is an example of predictor. fit() and predict() are two common functions. It also gives score() function which will evaluate the predictions.

#Feature Scaling

#Primarily, two types of feature scaling methods:

#1)Min-max scaling (Normalization) (value - min)/(max - min) Sklearn provides a class called MinMaxScaler for this

#2)Standardization (value - mean)/std Sklearn provides a class called StandardScaler for this

#CREATING PIPELINE

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()),

])

Code

Dragon Real Estate - Price Predictor

import pandas as pd

housing = pd.read\_csv("F:\datasets\housing\_boston.csv")

housing.head()

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

#0 471

#1 35

housing.describe()

%matplotlib inline

# # For plotting histogram

# import matplotlib.pyplot as plt

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

Train-Test Splitting

# 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]

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

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

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")

#for actual split with equal no. of test pattern and train pattern ,so that like in chas we get proper 0 #& 1 to both test and train set

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]

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

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

# 95/7

# 376/28

housing = strat\_train\_set.copy()

#LOOKING FOR CORRELATION

corr\_matrix = housing.corr()

​

from pandas.plotting import scatter\_matrix

attributes=["medv","rm","zn","lstat"]

scatter\_matrix(housing[attributes],figsize=(10,6))

# from pandas.plotting import scatter\_matrix

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

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

corr\_matrix

#GRAPH ANALYSIS

housing.plot(kind="scatter", x="rm", y="medv", alpha=0.8)

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

import seaborn as sns

sns.distplot(housing['chas'])

print("The skewness of chas is {}".format(housing['chas'].skew()))

​

​

The skewness of chas is 3.404265772962613

sns.distplot(housing['rm'])

print("The skewness of rm is {}".format(housing['rm'].skew()))

​

​

The skewness of rm is 0.326670699060074

sns.distplot(housing['crim'])

print("The skewness of scatter is {}".format(housing['crim'].skew()))

​

​

The skewness of scatter is 4.649931610435271

#heat maplot

import matplotlib.pyplot as plt#plotting, visvualizing

from sklearn import model\_selection

​

​

import seaborn as sns

corr\_matrix = housing.corr()

fig=plt.figure(figsize=(12,9))

sns.heatmap(corr\_matrix,vmax=.8,square=True)

#Trying out Attribute combinations

housing["tax"] = housing['tax']/housing['rm']

housing.head()

corr\_matrix = housing.corr()

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

housing.plot(kind="scatter", x="tax", y="medv", alpha=0.8)

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

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

Missing Attributes

# To take care of missing attributes, you have three options:

# 1. Get rid of the missing data points

# 2. Get rid of the whole attribute

# 3. Set the value to some value(0, mean or median)

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

a.shape

# Note that the original housing dataframe will remain unchanged

(404, 13)

housing.drop("rm", axis=1).shape # Option 2

# Note that there is no RM column and also note that the original housing dataframe will remain unchanged

(404, 12)

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

housing["rm"].fillna(median) # Option 3

# Note that the original housing dataframe will remain unchanged

housing["rm"].fillna(median) # Option 3

housing.shape

housing.shape

(404, 13)

# before we started filling missing attributes

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

from sklearn.impute import SimpleImputer

imputer = SimpleImputer(strategy="median")

imputer.fit(housing)

from sklearn.impute import SimpleImputer

imputer = SimpleImputer(strategy="median")

imputer.fit(housing)

SimpleImputer(copy=True, fill\_value=None, missing\_values=nan,

strategy='median', verbose=0)

imputer.statistics\_

imputer.statistics\_

X = imputer.transform(housing)

X = imputer.transform(housing)

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

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

housing\_tr.describe()

Scikit-learn Design

Primarily, three types of objects

Estimators - It estimates some parameter based on a dataset. Eg. imputer. It has a fit method and transform method. Fit method - Fits the dataset and calculates internal parameters

Transformers - transform method takes input and returns output based on the learnings from fit(). It also has a convenience function called fit\_transform() which fits and then transforms.

Predictors - LinearRegression model is an example of predictor. fit() and predict() are two common functions. It also gives score() function which will evaluate the predictions.

Feature Scaling

Primarily, two types of feature scaling methods:

Min-max scaling (Normalization) (value - min)/(max - min) Sklearn provides a class called MinMaxScaler for this

Standardization (value - mean)/std Sklearn provides a class called StandardScaler for this

Creating a Pipeline

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()),

])

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()),

])

​

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

housing\_num\_tr.shape

#Selecting a desired model for Dragon Real Estates

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)

some\_data = housing.iloc[:5]

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

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

model.predict(prepared\_data)

list(some\_labels)

#Evaluating the model

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)

rmse

#Using better evaluation technique - Cross Validation¶

# 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)

rmse\_scores

def print\_scores(scores):

print("Scores:", scores)

print("Mean: ", scores.mean())

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

print\_scores(rmse\_scores)

#Testing the model on test data

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)

final\_rmse

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

prepared\_data[0]