Exercise 1:

APPLY CENTRAL TENDENCY AND VARIABILITY ON GIVEN DATASET

Aim: Apply central tendency and variability on given dataset

Software Required: Google Co Lab, Jupyter notebook, Kaggle .

Theory:
Descriptive Statistics: These are calculated values that summarize aspects of a data set. First, we will focus on measures of central tendency, which as the name suggests, are statistics that aim to estimate the center of the data set around which (usually) most of the data values occur. Then, we will focus on measures of variability. These statistics reflect how spread out the data is in relation to other data points or the center of the data set. Measures of central tendency and variability require the data to have interval or ratio scales. After this notebook you will know:

- how to calculate sample mean, median, and mode and what each of these measures of central tendency tells you.
- how to calculate sample variance and standard deviation.
- how to calculate the z scores and determine the probability of a given z-score using a z-table.
- how to calculate standard error of a sample mean.
- how to calculate confidence intervals using a z-score for a given confidence level (α level).

Measures of Central Tendency

The following definitions of mean, median, and mode state in words how each is calculated.

Mean: The mean of a data set is the average value. Is that circular? Let me state this in the form of an algorithm. To find the mean of a data set first sum up all of the values in the data set then divide by the number of data points or examples in the data set. When we find or know the mean of an entire population we call that mean a parameter of the population and in is assigned the symbol μ. When the mean is of a sample it is a statistic of the sample ans is assigned the symbol $\bar{x}$.

Median: The median of a data set is simply the data point that occurs exactly in the middle of the data set. To find the median first the data needs to be sorted and listed in ascending or descending order. Then if there is an odd number of data points in the set the median is in the exact middle of this list. If there is an even number of data points in the list then the median is the average of the two middle values in the list.

Mode: The mode is the value in a data set that is repeated the most often. Below the mean, median, and mode are calculated for a dummy data set.

Measures of Variability

The following definitions of variance and standard deviation state in words how each is calculated.

Variance: The variance of a data set is found by first finding the deviation of each element in the data set from the mean. These deviations are squared and then added together. (Why are they squared?) Finally, the sum of squared deviations is normalized by the number of elements in the population, N, for a population variance or the number of element in the sample minus one, N-1, for the sample variance. (Why is a sample variance normalized by N-1?)
Standard Deviation: Once the variance is in hand, standard deviation is eacy to find. It is simply the square root of the variance. The symbol for population standard deviation is σ while the simple for sample standard deviation is s or sd.

Program :

```
import numpy as np
x= np.array([56, 31 ,56, 8 , 32])
print(" mean value of array x", np.mean(x))
print(" mean value of array x", x.mean())
print(" x median",np.median(x))
#print(" x median",x.median())


import numpy as np
x= np.array([56,31,56,8,32])
x1=np.sort(x)
print("Original unsorted values are     ",x)
print("Arguments of unsorted array are ",np.argsort(x))
print("sorted ascedning order values are",x1)
print("Arguments of sorted array are    ",np.argsort(x1))

import numpy as np
scores= np.array([89,73,84,91,87,77,94,67])
print(" score  median:",np.median(scores))
print("argument position for minimum value in unsorted array ",np.argmin(x))
print("argument position for maximum value in unsorted array ",np.argmax(x))
print("Average value ",np.average(x))
print("sum            ",np.sum(x))
print("Maximum  value  ",np.max(x))
print("minimum  ",np.min(x))

import numpy as np
y= np.array([56,31,0,56,0,8,88,0,32])
print(" Places where Non Zero vlaues are there ", np.nonzero(y))

import numpy as np
import matplotlib.pyplot as plt
y= np.array([650,450,275,350,387,575,555,649,361])
print("mean is   :",np.mean(y))
print("median is :",np.median(y))
range=np.max(y)- np.min(y)
print("range is :",range)
plt.bar(y,y,color = 'b')
plt.bar(np.mean(y),y,color = 'g',width=5)
plt.bar(np.median(y),y,color='r',width=5)
plt.bar(range,y,color = 'm',width =4)
plt.show()


import numpy as np
import matplotlib.pyplot as plt
y= np.array([650,450,275,350,387,575,555,649,361,1500,1578,1600])
print("mean is   :",np.mean(y))
print("median is :",np.median(y))
range=np.max(y)- np.min(y)
print("range is :",range)
plt.bar(y,y,color = 'b')
plt.bar(np.mean(y),y,color = 'g',width=10)
plt.bar(np.median(y),y,color='r',width=12)
plt.bar(range,y,color = 'm',width =14)
plt.show()
```

```
import numpy as np
import matplotlib.pyplot as plt
y= np.array([85, 87, 89, 90, 91, 91, 91, 92, 93, 93, 94, 96 ])
plt.boxplot(y)
plt.show()
def iqr(x):
    return np.percentile(x,75) - np.percentile(x,25)
number = [85, 87, 89, 90, 91, 91, 91, 92, 93, 93, 94, 96]
print(np.sort(number))
print("Q1:", np.percentile(number,25))
print("Q2:", np.percentile(number,50))
print("Q3:", np.percentile(number,75))
print("IQR Range :", iqr(number))
```

```
import numpy as np
import matplotlib.pyplot as plt

def iqr(x):
    return np.percentile(x,75) - np.percentile(x,25)
number = [25,75, 77, 79, 80, 81, 82, 83, 85, 85, 86, 86, 99]
print(np.sort(number))
print("Q1:", np.percentile(number,25))
print("Q2:", np.percentile(number,50))
print("Q3:", np.percentile(number,75))
print("IQR Range :", iqr(number))
plt.boxplot(number)
plt.show()
```

```
from numpy import random
x = random.normal(loc=0, scale=2, size=(3, 4))
print(x)
```

```
from numpy import random
import matplotlib.pyplot as plt
import seaborn as sns
sns.distplot(random.normal(size=1000), hist=False)
plt.show()
```

```
import matplotlib.pyplot as plt
import numpy as np
import scipy.stats as stats
import math

import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm
# Plot between -10 and 10 with .001 steps.
x_axis = np.arange(-10, 10, 0.001)
# Mean = 0, SD = 2.
plt.plot(x_axis, norm.pdf(x_axis,0,2))
plt.show()



import numpy as np
s = np.random.poisson(5, 10000)

import matplotlib.pyplot as plt
s = np.random.poisson(lam=(100., 500.), size=(100, 2))
count, bins, ignored = plt.hist(s, 14, density=True)
plt.show()


n, p = 10, .5  # number of trials, probability of each trial
s = np.random.binomial(n, p, 1000)
print(s)


sum(np.random.binomial(9, 0.1, 20000) == 0)/20000.

# Draw samples from the distribution:
import numpy as np
s = np.random.poisson(5, 10000)

# Display histogram of the sample:
import matplotlib.pyplot as plt
count, bins, ignored = plt.hist(s, 14, density=True)
plt.show()


import numpy as np
import matplotlib.pyplot as plt
data = np.random.randn(100, 5)
plt.boxplot(data)
plt.show()

# Draw samples from the distribution
import numpy as np
```

```
mu, sigma = 0, 0.1  # mean and standard deviation
s = np.random.normal(mu, sigma, 1000)

#Verify the mean and the variance
abs(mu - np.mean(s)) < 0.01    #True

#Display the histogram of the samples, along with the probability density function:

import matplotlib.pyplot as plt
count, bins, ignored = plt.hist(s, 30)#, normed=True)
plt.plot(bins, 1/(sigma * np.sqrt(2 * np.pi)) * np.exp( - (bins -
 mu)**2 / (2 * sigma**2) ), linewidth=2, color='r')
plt.show()


from numpy import random
import matplotlib.pyplot as plt
import seaborn as sns

sns.distplot(random.binomial(n=10, p=0.5, size=1000), hist=True, kde=False)
plt.show()


from numpy import random
import matplotlib.pyplot as plt
import seaborn as sns
sns.distplot(random.normal(loc=50, scale=5, size=1000), hist=False, label='normal')
sns.distplot(random.binomial(n=100, p=0.5, size=1000), hist=False, label='binomial')
plt.show()

import numpy as np
data1= np.array([75, 69, 56, 46, 47, 79, 92, 97, 89, 88,
        36, 96, 105, 32, 116, 101, 79, 93, 91, 112])
print("mean absolute deviation:")
np.mean(np.absolute(data1 - np.mean(data1)))


import pandas as pd

data = [75, 69, 56, 46, 47, 79, 92, 97, 89, 88,
        36, 96, 105, 32, 116, 101, 79, 93, 91, 112]
# Creating data frame of the given data
df = pd.DataFrame(data)
 # Absolute mean deviation
df.mad()    # mad() is mean absolute deviation function
```

```
import numpy as np
data_50 = np.array([10,11,13,15,18,20,22,23,23,30,
                     35,36,38,41,44,44,48,49,50,50,
                     55,56,57,60,63,64,68,69,70,73,
                     75,76,76,78,80,81,82,83,83,85,
                     87,89,90,94,96,99,101,102,103,107])
print(data_50)
print("15% of data values:",np.percentile(data_50,15))
print("25% of data values:",np.percentile(data_50,25,interpolation='lower'))
print("50% of data values:",np.percentile(data_50,50,interpolation='linear'))
print("75% of data values:",np.percentile(data_50,75,interpolation='higher'))


a = np.array([[10, 7, 4], [3, 2, 1],[22 , 55 ,44 ],[15, 16, 11],[30, 20,40]])
print(a)
print("50% :",np.percentile(a, 50))

import matplotlib.pyplot as plt
a = np.arange(4)
p = np.linspace(0, 100, 6001)
ax = plt.gca()
lines = [
    ('linear', None),
    ('higher', '--'),
    ('lower', '--'),
    ('nearest', '-.'),
    ('midpoint', '-.'),
]
for interpolation, style in lines:
    ax.plot(
        p, np.percentile(a, p, interpolation=interpolation),
        label=interpolation, linestyle=style)
ax.set(
    title='Interpolation methods for list: ' + str(a),
    xlabel='Percentile',
    ylabel='List item returned',
    yticks=a)
ax.legend()
plt.show()

a = np.array([[10, 7, 4], [3, 2, 1],[22 , 55 ,44 ],[15, 16, 11],[30, 20,40]])
print(a)
print("variance of array across columns:",np.var(a,axis=0))
print("standard deviaion across columns:",np.std(a,axis=0))
print("variance  of  array  across  rows  :",np.var(a,axis=1))
print("standard deviaion across rows  :",np.std(a,axis=1))
```

```
b =np.array([2,3,4,5,6,7,8,9])
print("variance:",np.var(b))
print("std     :",np.std(b))

x = np.array([[0, 2], [1, 1], [2, 0]]).T
print(x)
np.cov(x)

import pandas as pd
df = pd.DataFrame([(1, 2), (0, 3), (2, 0), (1, 1)],
                  columns=['dogs', 'cats'])
df

df.cov()



import numpy as np
x = np.arange(10, 20)
print(x)
y = np.array([2, 1, 4, 5, 8, 12, 18, 25, 96, 48])
print(y)
print()
r = np.corrcoef(x, y)
print(r)



import numpy as np
import scipy.stats
x = np.arange(10, 20)
y = np.array([2, 1, 4, 5, 8, 12, 18, 25, 96, 48])
print(scipy.stats.pearsonr(x, y))    # Pearson's r
print(scipy.stats.spearmanr(x, y))   # Spearman's rho
print(scipy.stats.kendalltau(x, y))  # Kendall's tau
```

Result: The probability parameters are
Mean =
Median=
Mode=
Variance=
Standard Deviation=

Exercise 2:

PERFORM EXPLORATORY DATA ANALYSIS ON GIVEN DATASET

Aim: Perform EDA on given dataset and prepare dataset to train and test ML model

Software Required: Google Co Lab, Jupyter notebook, Kaggle.

Dataset: **Toyota.csv**

Theory:

Exploratory Data Analysis (EDA) is exploring the data and discovering the insights Understand data, getting to know the context of data Getting to know the variables and their relationships Data Preprocessing (Handling missing values) Data visualization

Data Pre-processing includes the following:

- Detecting and Handling Missing values
- Type conversion
- Detecting and Treating Outliers
- Feature and Target attribute Selection
- Scaling
- Dimensionality Reduction
- Training & Test set splitting

Program :

```
import pandas as pd
import io
toyotadf=pd.read_csv('/kaggle/input/toyota-
dataset/Toyota.csv',index_col=0,na_values=["??","????"])

df = toyotadf.copy()
toyotadf.shape
```

Detecting and handling Missing Values

```
toyotadf.info()

toyotadf.isnull().sum()

toyotadf['HP'].unique()
```

```
import numpy as np
np.unique(toyotadf['KM'])


toyotadf['Doors'].unique()

toyotadf['Automatic'].unique()

toyotadf['MetColor'].unique()

toyotadf.info()
```

Now KM and HP columns are identified as float

```
toyotadf['MetColor'] = toyotadf['MetColor'].astype('O')
toyotadf['Automatic'] = toyotadf['Automatic'].astype('object')


# Alternate....
toyotadf[['MetColor','Automatic']] = toyotadf[['MetColor','Automatic']].astype
('object')

toyotadf['Doors'].unique()


toyotadf['Doors'].replace('three',3,inplace=True)
toyotadf['Doors'].replace('four',4,inplace=True)
toyotadf['Doors'].replace('five',5,inplace=True)


toyotadf['Doors'] = toyotadf['Doors'].astype('int')

toyotadf['Doors'].unique()

toyotadf.info()
```

Let's focus on NULL values....

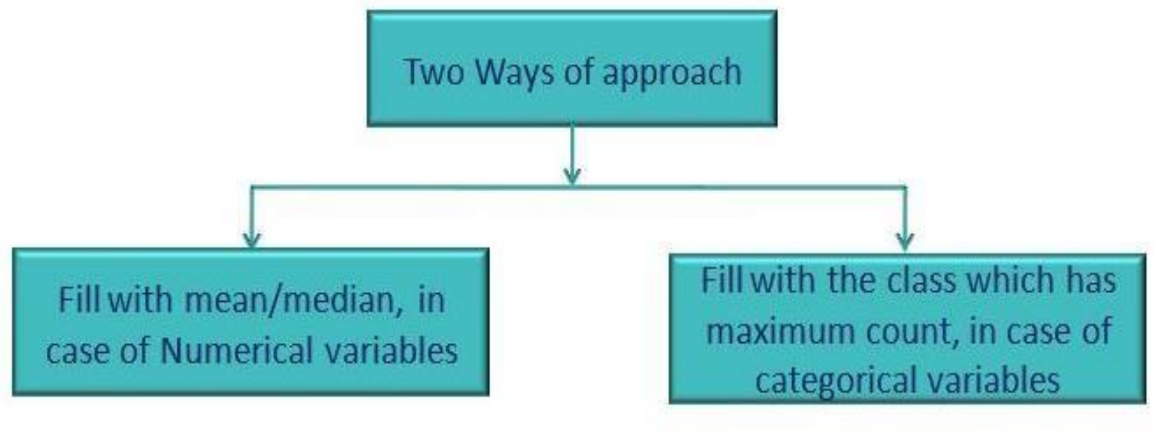
```
# Total no.of NULL values
toyotadf.isnull().sum().sum()
```

B. Whether to Handle or Remove Missing values????

Imputing Missing Values

List item
List item

Different approaches to fill the missing values


Two Ways of approach
Fill with mean/median, in case of Numerical variables
Fill with the class which has maximum count, in case of categorical variables

'Age', 'KM', and 'HP' are numeric... Whehter to fill with mean of median value??

```
toyotadf.describe()
```

Thumb Rule: for normal distribution use mean value for imputing; median for skewed distribution

```
toyotadf.describe(exclude=['int','float'])

toyotadf.describe(include='O')

toyotadf['Age'].tail(10)

toyotadf['Age'].mean()

toyotadf['Age'].fillna(toyotadf['Age'].mean(), inplace=True)


toyotadf['Age'].tail(10)

toyotadf['KM'].head(10)

toyotadf['KM'].fillna(toyotadf['KM'].median(), inplace=True)
toyotadf['KM'].median()
```

```
toyotadf['KM'].head(10)

toyotadf['FuelType'].value_counts()


toyotadf['FuelType'].mode() # most frequently occured value


toyotadf['MetColor'].value_counts()

toyotadf['MetColor'].mode()
```

`MetColor of category-1 is most frequently occuring`

```
toyotadf['FuelType'].fillna(toyotadf['FuelType'].value_counts().index[0], inplace=Tr
ue)
toyotadf['MetColor'].fillna(toyotadf['MetColor'].mode().index[0], inplace=True)


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

Missing Values Treated Successfully!!!

C. EDA for Numerical & Categorical Variables

- **Frequency tables**
- **Two-way table**
- **Two-way table-joint probability**
- **Two-way table-marginal probability**
- **Two-way table-conditional probability**
- **Correlation**

pandas.crosstab()

Computes a simple cross-tabulation of one, two or more factors

By default computes a frequency table of the factors Attributes:
Index - values to group by in the rows
columns - Values to group by in the columns
values - Array of values to aggregate according to the factors. Requires aggfunc be specified

Margins - Add row/column margins (subtotals)
dropna - Do not include columns whose entries are all NaN
normalize - Normalize by dividing all values by the sum of values
If passed 'all' or True, will normalize over all values
If passed 'index' will normalize over each row
If passed 'columns' will normalize over each column
If margins is True, will also normalize margin value

```
# frequency distribution of 'FuelType'

pd.crosstab(index=toyotadf['FuelType'],columns='count')
```

Most of the cars have Petrol as FuelType

```
# frequecny distribution of gearbox types w.r.t fueltype

pd.crosstab(index=toyotadf['Automatic'],
            columns=toyotadf['FuelType'],
            dropna=True)
```

The cars with Automatic gear box doen't have FuelType either CNG or Diesel; using only Petrol as FuelType

Most of the cars using Petrol and Manual gear box

Two-way Tabel - Joint Probability

gives Likelihood of two independent events happening at the same

```
pd.crosstab(index = toyotadf['Automatic'],
            columns = toyotadf['FuelType'],
            dropna = True,
            normalize = True) # normalize=True; will divide all the values by
                              # sum of values
```

Joint probability of Automatic gear box and Fueltype CNG is 0.01

The probability is high for the cars with Manual gearbox and pocessing Petrol as Fueltype - 0.833

There is NO probability for the cars having Automatic gear box and CNG or Diesel as Fueltype - 0.00 and 0.000

Two-way Tabel - Marginal Probability

probability of the occurance of single event

set margins=True to get marginal probability vlaues;

we will get row sum and column sum of the table values

```
pd.crosstab(index = toyotadf['Automatic'],
            columns = toyotadf['FuelType'],
            dropna = True,
            normalize = True)
```

Given the manual gear box, the probability of getting the fuel type CNG is 0.011, Diesel is 0.1 and Petrol is 0.88

For any manual gearbox the probability of fuel type can be Petrol

Automatic gearbox cars pocesses only Petrol as Fueltype

```
pd.crosstab(index = toyotadf['Automatic'],
            columns = toyotadf['FuelType'],
            margins = True,
            dropna = True,
            normalize ='columns')
```

D. Outliers

```
import warnings
warnings.filterwarnings('ignore')

import seaborn as sns
sns.boxplot(toyotadf['Price'])

toyotadf['Price'].sort_values()


sns.distplot(toyotadf['Price'])


sns.boxplot('Age', data=toyotadf)


sns.boxplot('HP',data=toyotadf)


sns.boxplot(toyotadf['KM'])


sns.boxplot(toyotadf['CC'])
sns.boxplot(toyotadf['Weight'])
```

Price, CC, Age, Weight, KM and HP are the attributes which are having Outliers

Using Statistical methods
Consider the data points for detection Outliers

```
data = [11,12,24,11,14,12,15,11,12,13,12,13,14,102,
        12,11,13,14,107,15,11,12,14,108,11,14,16]
len(data)
>>27
```

Outlier detection using Number Standard Deviation method

```
import numpy as np
lower_limit = np.mean(data) - 2* np.std(data)
upper_limit = np.mean(data) + 2* np.std(data)
print(lower_limit, upper_limit)
>>-34.87723139841871 81.84019436138166

outliers = []
def detect_outliers(data):
  for i in data:
    if (i<lower_limit or i>upper_limit):
      outliers.append(i)
  return outliers

outliers = detect_outliers(data)
outliers
>>[102, 107, 108]
```

Outlier detection using Z-Score method

$$Z = \frac{x - \mu}{\sigma}$$

```
import numpy as np
outliers = []
def detect_outliers(data):
  threshold = 2.0
  mean = np.mean(data)
  std = np.std(data)

  for i in data:
    z_score = np.abs((i - mean)/std)
    if np.ceil(z_score) > threshold:
      outliers.append(i)
  return outliers
outliers_pts = detect_outliers(data)
outliers_pts
```

E. Outlier Detection using Inter Quartile Range method

```
q1,q3 = np.percentile(data,[25,75])
print('Quartile 1: ',q1)
print('Quartile 2: ',q3)
iqr = q3 - q1
print('Inter Quartile Range: ',iqr)
```

```
>>Quartile 1:  12.0
Quartile 2:  14.5
Inter Quartile Range:  2.5


lower_bound = q1 - (iqr*1.5)
upper_bound = q3 + (iqr*1.5)
print(lower_bound)
print(upper_bound)


outliers = []
def detect_outliers(data):
  for i in data:
    if (i < lower_bound or i > upper_bound):
      outliers.append(i)
  return outliers

outliers = detect_outliers(data)
outliers

df=toyotadf.copy()
```

Result:

The Exploratory data analysis, Detecting and Handling Missing values, Type conversion, Detecting and Treating Outliers, Feature and Target attribute Selection, Scaling, Dimensionality Reduction, Training & Test set splitting are done on a given dataset.

Exercise 3:

BUILDING A LINEAR REGRESSION MODEL

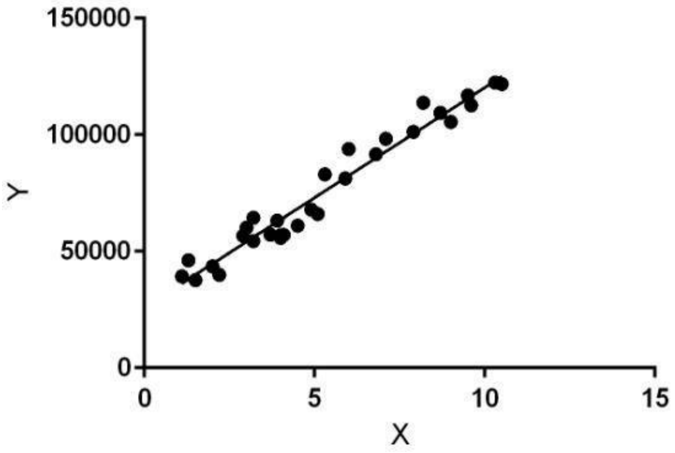
Aim: Calculate the MSE by Building a linear regression model using python on given dataset with and without data handling

a. Prepare the data for ML model
b. Splitting Training data and Test data.
c. Evaluate the model (intercept and slope).
d. Visualize the training set and testing set using Matplotlib, Seaborn.
e. predicting the test set result
f. compare actual output values with predicted values

Dataset: https://www.kaggle.com/camnugent/california-housing-prices

Software Required: Google Co Lab, Jupyter notebook, Kaggle .

Theory:
Linear Regression is a machine learning algorithm based on supervised learning. It performs a regression task. Regression models a target prediction value based on independent variables. It is mostly used for finding out the relationship between variables and forecasting. Different regression models differ based on – the kind of relationship between dependent and independent variables they are considering, and the number of independent variables getting used.



Linear regression performs the task to predict a dependent variable value (y) based on a given independent variable (x). So, this regression technique finds out a linear relationship between x (input) and y(output). Hence, the name is Linear Regression.
In the figure above, X (input) is the work experience and Y (output) is the salary of a person. The regression line is the best fit line for our model.

Hypothesis function for Linear Regression :

$$y = \theta_1 + \theta_2 . x$$

While training the model we are given :
x: input training data (univariate – one input variable(parameter))
y: labels to data (supervised learning)

When training the model – it fits the best line to predict the value of y for a given value of x. The model gets the best regression fit line by finding the best θ1 and θ2 values.
θ1: intercept
θ2: coefficient of x

Once we find the best θ1 and θ2 values, we get the best fit line. So when we are finally using our model for prediction, it will predict the value of y for the input value of x.

Program:

```
# Import libraries
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import os

#Read dataset into pandas dataframe
data = pd.read_csv('/kaggle/input/housing/housing.csv')
data

a.Prepare the data for ML model

data.shape

data.columns

data.describe()[['median_income','median_house_value']]

Finding Null Values & Outliers

data.isnull().sum()


# A. Data Visualizations

data.median_income.hist()

data.median_house_value.hist()

import matplotlib.pyplot as plt
plt.boxplot(data.median_income)
plt.boxplot(data.median_house_value)

plt.scatter(data.median_income,data.median_house_value)

data.corr()[['median_income','median_house_value']]

#data.median_income.values.reshape(-1,1)
x=data.median_income.values.reshape(-1,1)
x

y=data.median_house_value.values.reshape(-1,1)
y
```

# b. Splitting train and test datasets

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.2)
x_train.shape,x_test.shape,y_train.shape,y_test.shape
```

i) Building model using sklearn

```
from sklearn.linear_model import LinearRegression
lm=LinearRegression()

lm.fit(x_train,y_train)
```

# c. Evaluate the model (intercept and slope).

```
lm.coef_,lm.intercept_

y_pred= lm.intercept_+lm.coef_*x_train
y_pred
```

# e. Predicting the test set result

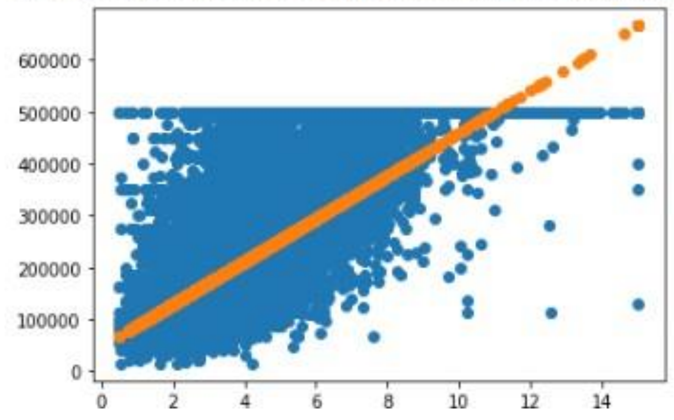
```
#Prediction for test dataset
y_pred=lm.predict(x_test)
y_pred


from sklearn.metrics import mean_squared_error, mean_absolute_error
mean_absolute_error(y_test,y_pred),np.sqrt(mean_squared_error(y_test,y_pred))
```

d.Visualize the training set and testing set using Matplotlib, Seaborn.

```
plt.scatter(x,y)
plt.scatter(x_test,y_pred)
```

```
<matplotlib.collections.PathCollection at 0x7f86cbfc42d0>
```



ii) Data Handling- Visualization of Data without outliers

```
plt.hist(x[x<14])

data1=data[data.median_income<data.median_income.quantile(0.99)]

data1.median_income.hist()

data1=data[data.median_income<data.median_income.quantile(0.92)]
data1=data1[data1.median_house_value<data1.median_house_value.quantile(0.
92)]

data1.median_income.hist()

data.shape,data1.shape

plt.boxplot(data1.median_income)

plt.boxplot(data1.median_house_value)

plt.scatter(data1.median_income,data1.median_house_value)

data1.columns

x=data1[['median_income','households','housing_median_age']]
y=data1.median_house_value.values.reshape(-1,1)

x.max(),x.min(),y.max(),y.min()
```

iii) Building model after Data Handling for outliers

```
from sklearn.model_selection import  train_test_split
x_train,x_test, y_train,y_test = train_test_split(x,y,test_size=0.2)
x_train.shape,y_train.shape, x_test.shape,y_test.shape

x_train.max(),y_train.max(), x_test.max(),y_test.max()
```

Building model using sklearn

```
from sklearn.linear_model import LinearRegression
lm=LinearRegression()
lm.fit(x_train,y_train)

# Display the parameters of model
lm.get_params()

# c. Evaluate the model (intercept and slope) after Data handling
for outliers

# Display the coefficients of input features
lm.coef_

# Display the intercept
lm.intercept_
```

e. Predicting the test set result

```
#prediction for test dataset
y_test_pred=lm.predict(x_test)
mean_absolute_error(y_test,y_test_pred)
```

f. compare actual output values with predicted values

```
#Predict the house value for given data of median_income, households and
median_age of house
newval=pd.DataFrame({'income':[4,6,3],'households':[5,4,2],'houseage':[20
,12,30]})
lm.predict(newval)
```

Result: The Mean square error for a given dataset is

MSE before Data Handling:

MSE after Data Handling: