Experiment 1 :

Write a python program to Compute Central Tendency Measures : Mean, Median , Mode ; Measure of Dispersion : variance, Standard Deviation.

Aim :- To Compute central tendency measures (mean median , mode) and the measures of dispersion (variance , standard deviation) for a given data set.

Program :-

```
import statistics
def central_tendency_and_dispersion (data):
if not data:
    return "The data list is empty"
mean = Statistics.mode (data)
median = Statistics. median (data)
try:
    mode = Statistics.mode (data)
 except Statistics. StatisticsError:
    mode = "No Unique mode"
Variance = Statistics. Variance (data)
Standard_deviation = Statistics. Stdev(data)
return {
    "Mean" : mean,
    "Median": median,
    "Mode" : mode,
  "Variance"; variance,
"Standard Deviation": standard_deviation
}
data = [1,2,2,3,4,5,5,5,6,6,7,8,9]
results = Central_tendency_and_dispersion(data)
for measure, value in results.items():
    print(f"{measure}: {values}")
```

Output :-

```
 Mean : 4.846153846153846
 Median:  5
 Mode  :  5
 Variance: 5.8076923076923075
Standard Deviation: 2.4099154150493l4
```

Experiment 2: Study of python Basic Libraries such as Statistics, Math, Numpy and Scipy

Aim :- To understand and explore basic python libraries such as Statistics, Math, numpy & scipy.

Program:

```
import math
import Statistics
import numpy as np
from Scipy import Stats
sqrt-25 = math.sqrt(25)
print("square root of 25 is :", sqrt-25)
factorial-5 = math.factorial(5)
print("Factorial of 5 is ", factorial-5)
Sine-90 = math.Sin(math.radians(90))
print(" Sine of 90 degrees is :", Sine-90)
data = [1,2,2,3,4,5,5,6,8,9,10]
mean = Statistics.mean(data)
print(" Mean of data is:", mean)
mode = Statistics.mode(data)
print("Mode of data is :", mode)
array = np.array ([1,2,3,4,5])
sum-array = np.sum(array)
print("sum of array elements is :", sum-array)
mean-array = np.mean(array)
```

```
print ("Mean of array elements is :", mean_array)
std_array = np.std(array)
print ("standard deviation of array elements is :" std_array)
data = [2, 8, 5, 7, 10, 12, 18, 5, 5]
Skewness = stats.skew(data)
print ("skewness of data is :", skewness)
kurto sis = stats.kurtosis(data)
Print ("kurtosis of data is :", kurtosis)
t_stat, P_value = stats.ttest_1samp(data, 10)
Print ("T-statistic is :", t_stat)
Print ("P_value is :", p_value)
```

Output :-

```
Square root of 25 is : 5.0
Factorial of 5 is : 120
Sine of 90 degrees is : 1.0
Mean of data is : 5.2
Median of data is : 5.0
Mode of data is 5
Sum of array elements is : 15
mean of array elements is : 3.0
Standard deviation of array elements : 1.4142135623730
                                       951.
Skewness of data is : 0.9302669073823 68
kurtosis of data is : 0.14000472589792112
T-statistics is : -1.2510864843424485
P-value is : 0.24624961912944497
```

Experiment - 3

Study of python libraries for ML application such as pandas and Matplotlib.

Aim:- To understand and explore libraries commonly used in machine learning applications, namely pandas and Matplotlib.

Program:-

```
import pandas as pd
import matplotlib.pyplot as plt
data = {
   'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eva'],
   'Age': [24, 27, 22, 32, 29]
   'Salary': [7000, 8000, 65000, 12000, 95000]
 }
df = pd.DataFrame(data)
print ("DataFrame:")
print (df)
print ("\n Descriptive Statistics:")
print( df.describe())
df ['YearsExperience'] = [2,5,1,8,6]
print( "\n Dataframe after adding a new Column:")
print (df)
high-Salary = df[df['Salary'] >80000]
print ( "\n Rows where Salary is greater than
                                   8000...)
print( high-Salary)
Plt.figure (figsize = (10,6))
Plt.Scatter (df['Age'], df['Salary'], color = 'blue',
label = 'Salary')
Plt.title ('Age vs Salary )
Plt. Xlabel ('Age')
Plt. ylabel ('Salary')
Plt. legend ( )
plt. grid (True)
Plt. show ( )
Plt. figure (figsize = (10,6))
bar-width = 0.35
index = range (len(df))
Plt.bar (index, df['Age'], bar-width ,color='b',
          label = 'Age')
Plt. bar ( [i+ bar-width for i in index], df['
   YearsExperience'], bar_width , color='r',
 label = 'YearsExperience')
Plt. xlabel ('person')
Plt. ylabel ( 'values')
Plt. title (' Age and Years of Experience')
Plt. xticks ([i+ bar_width/2 for i in index],
                      df['Name'])
Plt. legend ( )
Plt. tight_layout ( )
Plt. show ( )
```

Output :-

```
Dataframe :
        Name     Age     Salary
  0     Alice    24      70000
  1     Bob      27      80000
  2     charlie  22      65000
  3     David    32      12000
  4     Eva      29      95000
Descriptive statistics:
              Age            Salary
  Count   5.000000       5.000000
  mean    26.800000      86000.000000
  std     3.962323       22192.341021
  min     22.000000      65000.000000
  25%     24.000000      70000.000000
  50%     27.000000      80000.000000
  75%     29.000000      95000.000000
  max     32.000000      120000.000000
Dataframe after adding a new column
        Name     Age   Salary   YearsExperience
  0     Alice    24    70000        2
  1     Bob      27    80000        5
  2     Charlie  22    65000        1
  3     David    32    120000       8
  4     Eva      29    95000        6
```

Rows where Salary is greater than 8000:

	Name	Age	Salary	Years Experience
3	David	32	120000	8
4	Eva	29	95000	6

Experience 4 :- Write a python program to implement Simple linear Regression.

Aim : To implement and understand simple linear Regression, a fundamental machine learning algorithm for predicting a continuous target variable based on one independent variable.

Program :-

```
import pandas as pd
import numpy as np
from sklearn import linear_model
import matplotlib.pyplot as plt
df = pd.read_csv('homeprices.csv')
df
%matplotlib inline
plt.xlabel('area')
plt.ylabel('price')
plt.scatter(df.area, df.price, color='red',
                              market='+')
new_df = df.drop("price", axis='columns')
new_df
price = df.price
price
reg = linear_model.LinearRegression()
reg.fit(new_df, price)
reg.predict([[3300]])
reg.coef_
reg.intercept_
```

Experiment - 5: Implementation of Multiple Linear Regression using Scikit - learn

Aim:- To implement multiple linear Regression a supervised learning algorithm for predicting a continuous target variables

Program:-

```
import pandas as pd
import numpy as np
from sklearn import linear_model
!pip install word2number
from word2number import w2n
d = pd.read_csv("hiring.csv")
d
d.experience = d.experience.fillna("zero")
d
d.experience = d.experience.apply(w2n.word-to
                                   -num)
d
import math
median_test_score = math.floor(d['test_score
                  (out of 10)'].mean())
median_test_scored['test_score(out of 10)']
    = d['test_score(out of 10)'].fillna(median
 -~~com~~_test_score)
d
reg = linear_model.LinearRegression()
reg.fit(d[['experience', 'test_score(out of 10)',
      'interview_score (out of 10)']].d['Salary(₹)']])
LinearRegression()
reg.predict([[2,9,6]])
```

Experiment - 6 : Implementation of Decision tree using Sklearn and its parameter tuning.

Objective : To implement Decision Tree algorithm for classification or regression tasks using Scikit - learn and perform parameter tuning to improve model performance.

Program

```
import pandas as pd
df = pd.read_csv ("salaries.csv")
df.head ()
df ['company'].nunique()
pd.value_counts (df.company)
df ['job'].nunique ()
Pd.value_Counts (df.job)
df ['degree'].nunique()
Pd.value_counts (df.degree)
result = df.dtypes
result
inputs = df.drops ('Salary_more_then_100K',
              axis = 'columns')
target = df ['Salary_more_then_100k]
df.describe()
from Sklearn.preprocessing import LabelEncounter
le_company = LabelEncounter()
```

```
le_job = LabelEncounter( )
le_degree = LabelEncoder( )
inputs['company_n'] = le_company.fit_transf
                        -orm (inputs ['company'])
inputs ['job_n'] = le_job.fit_transform (inputs
                                       ['job'])
inputs ['degree_n'] = le_degree.fit_transform
                     (inputs ['~~job~~ degree'])
inputs_n = inputs.drop ([ 'company', 'job', 'degree'],
          axis = 'columns')
inputs_n
 target
 from sklearn import tree
 model = tree.DecisionTreeClassifier( )
 model.fit (inputs_n, target)
 model.get_params( )
 model.Score (inputs_n, target)
 model.predict ([[2,1,0]])
 model.predict ([[2,1,1]])
 model.predict ([[2,2,1]])
```