Practical No.: 01

Aim:- Discretive Satistcs

1. Mean

import numpy as np

print("Rutuja Kale, 14")

data=np.array([[24,23,23,24,24,22],[6.9,5.65,6.53,6.28,6.35,5.73],[112.5,84,98,102.5,83,84.5]])  
  
print(data)  
  
print("Mean of entire data: ",np.mean(data))  
  
print("Mean of each row: ", np.mean(data,axis=1))

1. Median

import numpy as np  
import statistics as s  
print("Rutuja Kale, 14")

data=[1,2,4,5,6,76,8,45]  
print("Data:",data)  
  
*#Median without python package*s1=sorted(data)  
print("Sorted data",s1)  
n=len(data)  
  
if n % 2 == 0:  
 median1 = s1[n//2]  
 median2 = s1[n//2 - 1]  
 median = (median1 + median2)/2  
else:  
 median = s1[n//2]  
print("Median is: " + str(median))

*#Median using python packages  
  
#Numpy*m3 = np.median(data)  
print("Median is:",m3)  
  
*#Statistics*m4 = np.median(data)  
print("Median is:",m4)

1. Mode

import statistics as s  
print("Rutuja Kale, 14")

data = [10, 20, 30, 30, 40, 40, 40, 50, 50, 60]  
print("data:",data)  
  
m2=s.mode(data)  
print("Mode of given set of data values is",m2)

1. Correlation

import statistics as s  
import numpy as np  
import pandas as pd  
import seaborn as sns  
import matplotlib.pyplot as plt  
print("Rutuja Kale, 14")

df=pd.DataFrame({'TCS':[100,120,130,90,115],'Wipro':[50,55,70,80,85]})  
print("Data:",df)  
  
*#Calculating Corelation*c=df.corr()  
print("Correlation of a given data:",c)  
  
sns.heatmap(c,annot=True)  
plt.show()

1. Standard Deviation

import statistics as s  
print("Rutuja Kale, 14")

*#for standard Deviation*x=[1,2,3,4,5,6]  
y=[2,4,6,8,10]  
  
print("Data of x: ",x,"\n")  
print("Data of y: ",y,"\n")  
  
a=s.stdev(x)  
b=s.stdev(y)  
print("Standard Deviation of x: ",a,"\n")  
print("Standard Deviation of y: ",b,"\n")  
  
*#for mean deviation*c=s.mean(x)  
d=s.mean(y)  
e=s.pstdev(x)  
f=s.pstdev(y)  
  
print("Mean of x: ",c,"\n")  
print("Mean of y: ",d,"\n")  
print("Mean Deviation of x: ",e,"\n")  
print("Mean Deviation of y: ",f,"\n")

1. Variance

import statistics as s  
import pandas as pd  
print("Rutuja Kale, 14")

df=pd.DataFrame({'TCS':[100,120,130,90,115],'Wipro':[50,55,70,80,85]})  
print("Data:\n",df,"\n")  
  
*#Calculatig Variance*var=df.var()  
print("Variance of data \n",var)

Pract 2 Hypothesis Testing

1. One Way

import pandas as pd  
from scipy.stats import chi2\_contingency *#used when you want to work wi th chi-square*print("Rutuja kale,14")  
df = pd.read\_csv('diabetes (1).csv')  
df.columns = ['Pregnancies','Glucose','BloodPressure','SkinThickness',' Insulin','BMI','DiabetesPedigreeFunction','Age','Outcome']  
df.head()  
df.sample(5)  
df.info()  
df.describe()  
def Age\_Final(df):  
 if df['Age']<= 40:  
 return 1  
 elif df['Age'] > 40:  
 return 2  
 else:  
 return 'indifferent'  
df['Age\_New'] = df.apply(Age\_Final,axis=1)  
df.head()  
*#Dropping the Columns*df\_new = df.drop(columns = ['Pregnancies','Glucose','BloodPressure','SkinThickness','BMI','DiabetesPedigreeFunction','Age'])  
df\_new.head()  
*#Hypothesis Testing*stats,p,dof,expected = chi2\_contingency(df\_new)  
print("Statistic:",stats)  
print("p-value:",p)  
print("Degree of Freedom:",dof)  
prob = 0.95 *# 95%*alpha = 1.0 - prob  
print("Alpha Value:",alpha)  
if p <= alpha:  
 print("Rejecting the Null Hypothesis")  
else:  
 print("Accepting the Null Hypothesis")

B)

Practical No.: 03 Correlation And Regression

a. Linear Regression

import pandas as pd  
import numpy as np  
  
print("Rutuja Kale,14")  
x=np.array([1,2,4,5,6,7,8,10])  
y=np.array([2,4,6,7,9,11,12,14])  
  
x\_mean=np.mean(x)  
y\_mean=np.mean(y)  
  
n=np.sum((x-x\_mean)\*(y-y\_mean))  
d=np.sum((x-x\_mean)\*\*2)  
  
b=n/d  
a=y\_mean-(b\*x\_mean)  
  
print("Beta",b)  
print("Alpha",a)

b.List Square Method

import numpy as np  
  
print("Rututja Kale,14")  
def list\_square(x,y):  
 n=len(x)  
 sum\_x=np.sum(x)  
 sum\_y = np.sum(y)  
  
 sum\_xy=np.sum(x\*y)  
 sum\_x2=np.sum(x\*\*2)  
  
 a=(sum\_y\*sum\_x2-sum\_x\*sum\_xy)/(n\*sum\_x2-sum\_x\*\*2)  
 b=(n\*sum\_xy-sum\_x\*sum\_y)/(n\*sum\_x2-sum\_x\*\*2)  
 return a,b  
  
x=np.array([1,2,4,6,7])  
y=np.array([1,3,3,5,4])  
  
a,b=list\_square(x,y)  
print(a,b)

c.Karl Pearson's coefficient

import pandas as pd  
import numpy as np  
  
print("Rutuja Kale,14")  
df=pd.DataFrame({'x':[1,2,3,4,5],'y':[2,4,5,4,5]})  
  
covar= df['x'].cov(df['y'])  
  
var\_x = df['x'].var()  
var\_y = df ['y'].var()  
  
r=covar/np.sqrt(var\_x \* var\_y)  
print (r)

d.Spearman’s Rank Correlation

import numpy as np  
  
print("Rutuja Kale,14")  
def spearman\_corr(x,y):  
 x\_rank= np.argsort(x).argsort()  
 y\_rank= np.argsort(y).argsort()  
 d\_rank=x\_rank-y\_rank  
 d\_rank\_square=np.square(d\_rank)  
 d1 = np.sum(d\_rank\_square)  
 n=len(x)  
 d2=n\*(n\*\*2-1)  
 d3=6\*(d1/d2)  
 d4=1-d3  
 return d4  
  
  
x=[56,75,45,71]  
y=[66,70,40,60]  
r=spearman\_corr(x,y)  
print(r)

Practical No.: 04 Anova

1. One Way

import pandas as pd  
import seaborn as sns  
from scipy.stats import f\_oneway  
import urllib  
  
print("Rutuja Kale,14")  
  
*# Load iris.csv dataset from Github*iris = sns.load\_dataset("iris")  
  
*# Split dataset by species and replace NaN values with 0*setosa = iris[iris["species"] == "setosa"]["petal\_length"].fillna(0)  
versicolor = iris[iris["species"] == "versicolor"]["petal\_length"].fillna(0)  
virginica = iris[iris["species"] == "virginica"]["petal\_length"].fillna(0)  
  
*# Define null and alternative hypotheses*null\_hypothesis = "The mean petal length is equal across all three species of iris (setosa, versicolor, and virginica)."  
alt\_hypothesis = "The mean petal length differs across at least one of the three species of iris."  
  
*# Perform one-way ANOVA test*f\_stat, p\_value = f\_oneway(setosa, versicolor, virginica)  
  
*# Print results*print("Null hypothesis:", null\_hypothesis)  
print("Alternative hypothesis:", alt\_hypothesis)  
print("F-value:", f\_stat)  
print("P-value:", p\_value)  
  
formatted\_num = '{:.100f}'.format(p\_value)  
print(formatted\_num)  
  
*# Define significance level*alpha = 0.05  
  
*# Determine whether to accept or reject null hypothesis*if p\_value < alpha:  
 print("We reject the null hypothesis.")  
else:  
 print("We accept the null hypothesis.")  
  
*# Visualize results using a heatmap*sns.heatmap(iris.corr(), annot=True, cmap="YlGnBu")

1. Two Way

import pandas as pd  
import seaborn as sns  
from statsmodels.formula.api import ols  
from statsmodels.stats.anova import anova\_lm  
import matplotlib.pyplot as plt  
  
print("Rutuja Kale,14")  
*# Load iris.csv dataset from Github*iris = sns.load\_dataset("iris")  
  
*# Replace NaN values with 0*iris.fillna(0, inplace=True)  
  
*# Define null and alternative hypotheses*null\_hypothesis = "There is no significant interaction between the species and the petal width in determining petal length."  
alt\_hypothesis = "There is a significant interaction between the species and the petal width in determining petal length."  
  
*# Fit the model with the formula*model = ols('petal\_length ~ species + petal\_width + species:petal\_width', data=iris).fit()  
  
*# Perform two-way ANOVA test*anova\_table = anova\_lm(model, typ=2)  
  
*# Print results*print("Null hypothesis:", null\_hypothesis)  
print("Alternative hypothesis:", alt\_hypothesis)  
print(anova\_table)  
  
*# Define significance level*alpha = 0.05  
  
*# Determine whether to accept or reject null hypothesis*if anova\_table['PR(>F)'][2] < alpha:  
 print("We reject the null hypothesis.")  
else:  
 print("We accept the null hypothesis.")  
  
*# Visualize results using a heatmap*plt.figure(figsize=(10, 8)) *# set the size of the plot*sns.heatmap(iris.groupby(["species", "petal\_width"]).mean()["petal\_length"].unstack(), annot=True, cmap="YlGnBu")  
plt.show()

Practical 5: Forecasting:-

print("Rutuja Kale,14")

import pandas as pd

# importing the dataset

df = pd.read\_csv("sales.csv")

# printing

print(df.isnull().sum())

#print(df)

import matplotlib.pyplot as plt

# setting the size

plt.figure(figsize=(23,7))

# plotting the graph

plt.plot(df.Month, df.Sales\_Value)

plt.show()

# Importing the modules

from statsmodels.graphics.tsaplots import plot\_acf, plot\_pacf

# fixing the size

plt.rcParams.update({'figure.figsize':(9,7), 'figure.dpi':120})

# Original Series

fig, axes = plt.subplots(2, 2, sharex=True)

axes[0, 0].plot(df.Sales\_Value); axes[0, 0].set\_title('Original Series')

plot\_acf(df.Sales\_Value, ax=axes[0, 1])

# 1st Differencing to make stationary time series data

axes[1, 0].plot(df.Sales\_Value.diff()); axes[1, 0].set\_title('1st Order Differencing')

plot\_acf(df.Sales\_Value.diff().dropna(), ax=axes[1, 1])

plt.show()

plt.show()

#importing modules

#Parial auto correlation

from statsmodels.graphics.tsaplots import plot\_acf, plot\_pacf

# PACF plot of 1st differenced series

plt.rcParams.update({'figure.figsize':(9,3), 'figure.dpi':120})

# fixing the axis

fig, axes = plt.subplots(1, 2, sharex=True)

# plotting on differen axis

axes[0].plot(df.Sales\_Value.diff()); axes[0].set\_title('1st Differencing')

axes[1].set(ylim=(0,5))

# plotting partial autocorrelation function

plot\_pacf(df.Sales\_Value.diff().dropna(), ax=axes[1])

plt.show()

# importing the ARIMA model

from statsmodels.tsa.arima.model import ARIMA

from statsmodels.graphics.tsaplots import plot\_predict

import statsmodels

# 1,1,1 ( arima p d q )

model = ARIMA(df.Sales\_Value, order=(1,1,1))

# Training arima modeling

model\_fit = model.fit()

# arima model results

#model\_fit.plot\_predict(dyanmic=False)

plot\_predict(model\_fit)

plt.show()

Practical 6:- Time Series

# time series

import pandas as pd

import matplotlib.pyplot as plt

print(“Rutuja Kale,14”)

# Load data

data = pd.read\_csv('time\_series\_data.csv', parse\_dates=['date'], index\_col='date')

# Plot the time series

plt.plot(data)

plt.xlabel('Date')

plt.ylabel('Value')

plt.title('Time Series Data')

plt.show()

# Compute rolling mean and standard deviation

rolling\_mean = data.rolling(window=2).mean()

rolling\_std = data.rolling(window=2).std()

print ("rolling\_mean")

# Plot the rolling statistics

plt.plot(data, label='Original')

plt.plot(rolling\_mean, label='Rolling Mean')

plt.plot(rolling\_std, label='Rolling Std')

plt.legend()

plt.title('Rolling Mean and Standard Deviation')

plt.show()

# Perform Dickey-Fuller test for stationarity

from statsmodels.tsa.stattools import adfuller

result = adfuller(data['value'])

print('ADF Statistic:', result[0])

print('p-value:', result[1])

print('Critical Values:', result[4])

Practical 7 Markover:

#MARKOV CHAIN

#Example 1:

a)

import random

print("Rutuja Kale,14")

# Define transition probabilities for the Markov chain

transition\_probabilities = {

    'sunny': {'sunny': 0.8, 'rainy': 0.2},

    'rainy': {'sunny': 0.4, 'rainy': 0.6}

}

# Define the initial state

current\_state = 'rainy'

# Simulate the Markov chain

for i in range(5):

    print(current\_state)

    next\_states = list(transition\_probabilities[current\_state].keys())

    probabilities = list(transition\_probabilities[current\_state].values())

    current\_state = random.choices(next\_states, probabilities)[0]

b)

print("Rutuja Kale,14")

import random

# Define the transition matrix

transition\_matrix = {

    'A': {'A': 0.5, 'B': 0.5},

    'B': {'A': 0.3, 'B': 0.7}

}

# Define the starting state

current\_state = 'A'

# Generate a chain of 10 states

chain = []

for i in range(10):

    chain.append(current\_state)

    next\_states = list(transition\_matrix[current\_state].keys())

    probabilities = list(transition\_matrix[current\_state].values())

    current\_state = random.choices(next\_states, probabilities)[0]

# Print the resulting chain

print(chain)

Practical 8 Probability:

import math

# Probability of acceptance

p = 0.3

# Number of trials (students applying)

n = 5

# Probability of at most 2 being accepted

prob = 0

for i in range(3):

    # Calculate probability of i students being accepted

    comb = math.comb(n, i)

    prob\_i = comb \* (p \* i) \* ((1 - p) \* (n - i))

    prob += prob\_i

print("Probability of at most 2 students being accepted:", prob)

import math

# Probability of winning a single game

p = 0.5

# Number of games in the series

n = 7

# Probability of World Series lasting i games

for i in range(4, n+1):

    # Calculate probability of i games being played

    comb = math.comb(n, i)

    prob\_i = comb \* (p \* i) \* ((1 - p) \* (n - i))

    print("Probability of World Series lasting", i, "games:", prob\_i)

import math

# Probability of making a free throw

p = 0.7

# Number of free throws attempted

n = 5

# Probability of making the third free throw on the fifth shot

comb = math.comb(n-1, 2) # number of combinations where the third free throw is made on the fifth shot

prob = comb \* (p \* 3) \* ((1 - p) \* 2)

print("Probability of making the third free throw on the fifth shot:", prob)

#using the hypergeometric distribution

import math

# Total number of cards in the deck

N = 52

# Number of hearts in the deck

n\_hearts = 13

# Number of cards drawn

n\_drawn = 5

# Number of ways to draw 2 or fewer hearts

n\_ways = 0

for k in range(3):

    n\_ways += math.comb(n\_hearts, k) \* math.comb(N - n\_hearts, n\_drawn - k)

# Total number of ways to draw 5 cards

n\_total\_ways = math.comb(N, n\_drawn)

# Probability of obtaining 2 or fewer hearts

prob = n\_ways / n\_total\_ways

print("Probability of obtaining 2 or fewer hearts:", prob)

# Total number of cards in the deck

N = 52

# Number of cards to draw from each suit

n\_spades = 1

n\_hearts = 1

n\_diamonds = 1

n\_clubs = 2

# Total number of cards to draw

n\_drawn = n\_spades + n\_hearts + n\_diamonds + n\_clubs

# Probability of drawing a spade

p\_spade = 1 / 4

# Probability of drawing a heart

p\_heart = 1 / 4

# Probability of drawing a diamond

p\_diamond = 1 / 4

# Probability of drawing a club

p\_club = 1 / 4

# Probability of drawing 1 spade, 1 heart, 1 diamond, and 2 clubs

prob = (p\_spade \* n\_spades) \* (p\_heart \* n\_hearts) \* (p\_diamond \* n\_diamonds) \* (p\_club \* n\_clubs)

print("Probability of drawing 1 spade, 1 heart, 1 diamond, and 2 clubs:", prob)

#Poisson distribution

#The average number of homes sold by the Acme Realty company is 2 homes per day. What is the probability that exactly 3 homes will be sold tomorrow?

import math

# Average number of homes sold per day

lam = 2

# Number of homes sold tomorrow

k = 3

# Probability of selling k homes tomorrow

prob = (lam \*\* k) \* math.exp(-lam) / math.factorial(k)

print("Probability of selling exactly 3 homes tomorrow:", prob)