1)

```
# -*- coding: utf-8 -*-
"""
Created on Fri Oct  1 12:13:49 2021

@author: admin
"""

import numpy as np
X=np.array([[1,2,3],[3,2,1],[9,10,7]])
print(X)
Y=np.array([[1,2,3],[3,2,1],[9,10,7]]).T
print(X)
A=np.array([[0]*3])
print(A)
print(np.zeros((4,8)))
print(np.ones(4))
print(X)
print(Y)
print(X+Y)
print(np.shape(X))
I=np.array([[10,20,30,40,50,60,70]])
print(I)
print(I[0,1],I[0,3],I[0,6])
print(np.shape(I))
print(I.T)
V=np.array([[1,2,3,4,5],[6,7,8,9,10],[11,12,13,14,15],[16,17,18,19,20]])
print(V)
print(np.shape(V))
print(V.T)
print(V[:,0])
print(V[0,:])
print(np.zeros(7))
print(np.ones(3))
```

2)

```
import numpy as np
# 1. Define and print a 6 dimentional vector
X=np.array([[1,2,3,4,5,6]])
print(X)

# 2. Print the transpose of the above vector
print(X.T)

# 3. Define two non square matrices such that they can be mulplied.
X=np.array([[1,2],[3,4],[5,6]])
Y=np.array([[1,2,3],[4,5,6]])

# 4. Print the shape of the above matrices\
print(np.shape(X), np.shape(Y))
```

```
# 5. Print the product of above two matrices (do so without using the inbuilt functions).
Z=np.array([np.zeros(3)]*3)
for i in range(len(X)):
    for j in range(len(Y[1])):
        for k in range(len(Y)):
            Z[i][j] += X[i][k] * Y[k][j]
print(Z)

# 6. Define two non square matrices of same order and print their sum.
A=np.array([[1,2,3],[4,5,6]])
B=np.array([[-1,-2,-3],[-4,-5,-6]])
print(A+B)

# 7. Define a square matrix A.
A=np.array([[7,2,4],[4,9,6],[7,8,9]])

# 8. Print the transpose of A.
print(A.T)

# 9. Print the identity matrix of the above order I.
I=np.array([[1,0,0],[0,1,0],[0,0,1]])
print(I)

# 10. Verify A.I = I.A for matrix multiplication.
X=A@I
print("A.I = ",X)
Y=I@A
print("I.A = ",Y)
print(" Therefore, A.I = I.A")

# 11. Define another square matrix of the same order as A.
B=np.array([[2,5,7],[3,6,3],[0,1,9]])

# 12. Print the product of the matrices as matrix multiplication
print(A@B)

# 13. Print the product of the matrices by element wise multiplication
print(np.multiply(A,B))

# 14. Calculate and print the inverse of A. (Use linalg)
d=np.linalg.det(A)
print("Determinant = ",d)
if d!=0:
    print("Inverse of A = ",np.linalg.inv(A))
else:
    print("Inverse does not exist")
```

3)

```
# -*- coding: utf-8 -*-
"""
Created on Thu Oct 28 12:17:01 2021
```

```
@author: admin
"""

import pandas as pd
import numpy as np
import os
import seaborn as sns
import matplotlib.pyplot as plt

os.chdir("C:/Users/Lohith/Documents")
iris = pd.read_csv('Iris.csv')
print(iris.head())
print(iris.describe())
sns.countplot(x='Species', data = iris)
plt.show()
#sns.scatterplot('SepalLengthCm','SepalWidthCm', hue='Species',data = iris)
#plt.show()
#sns.pairplot(iris.drop(['Id'],axis =1), hue= 'Species', height= 2)
#plt.show()
#sns.heatmap(iris.corr(), data = iris)
#plt.show()
#x= iris.corr(method= 'pearson')
#print(x)
#sns.heatmap(iris.corr(method='pearson').drop(['Id'],axis=1).drop(['Id'],axis=0))
```

4)

```
# -*- coding: utf-8 -*-
"""
Created on Fri Oct 29 11:28:26 2021

@author: admin
"""
# EDA and linear regression for two pair of variables
import pandas as pd
import numpy as np
import os
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import linear_model
from sklearn import datasets
import sklearn



os.chdir("C:/Users/Lohith/Documents")
mtcars=pd.read_csv('CarPrice_Assignment.csv')
#print(mtcars.describe())
mtcars.info()

# 1. EDA and visualisation
```

```
#print(mtcars.describe())

#sns.countplot('doornumber',data=mtcars)
#plt.show()

#plt.hist('cylindernumber',data = mtcars)
#plt.show()

#x= mtcars.corr(method= 'pearson')
#print(x)

#sns.heatmap(mtcars.corr(method='pearson').drop(['car_ID','symboling'],axis=1).drop(['car_ID','symboling'],axis=0),
data=mtcars)
#sns.show()

#df = pd.DataFrame(mtcars,columns=['cylindernumber','horsepower'])
#plt.bar(df['cylindernumber'], df['horsepower'])
#plt.title('Cylinder number vs Horsepower', fontsize=14)
#plt.xlabel('CYlinder Number', fontsize=14)
#plt.ylabel('Horse Power', fontsize=14)
#plt.show()


#sns.pairplot(mtcars)
#plt.show()

#sns.boxplot(y='compressionratio',x='fueltype',data=mtcars)
#plt.show()


#2. Regression on one variable
#(a) Regression on one variable for negative correlation
#X=mtcars[['highwaympg']]
#Y=mtcars[['horsepower']]
#reg=linear_model.LinearRegression()
#reg.fit(X,Y)
#print(reg.coef_)
#sns.regplot(X,Y)
#plt.show()

#(b) Regression on one variable for positive correlation
X=mtcars[['wheelbase']]
Y=mtcars[['carlength']]
reg=linear_model.LinearRegression()
reg.fit(X,Y)
print(reg.coef_)
print(reg.intercept_) #IMP
sns.regplot(X,Y)
plt.show()

#(c) Regression on one variable with no correlation
#X=mtcars[['stroke']]
#Y=mtcars[['price']]
#reg=linear_model.LinearRegression()
#reg.fit(X,Y)
```

```
#print(reg.coef_)
#sns.regplot(X,Y)
#plt.show()



#3. Regression on multiple variables
X=mtcars[['horsepower','curbweight']]
Y=mtcars[['price']]
reg=linear_model.LinearRegression()
reg.fit(X,Y)
print(reg.coef_)

# complete credit to the internet for the below code
df2 = pd.DataFrame(mtcars,columns=['horsepower','curbweight','price'])
import statsmodels.formula.api as smf
model = smf.ols(formula='price ~ horsepower + curbweight', data=df2)
results_formula = model.fit()
results_formula.params


## Prepare the data for Visualization

x_surf, y_surf = np.meshgrid(np.linspace(df2.horsepower.min(), df2.horsepower.max(), 100),np.linspace(df2.curbweight.min(), df2.curbweight.max(), 100))
onlyX = pd.DataFrame({'horsepower': x_surf.ravel(), 'curbweight': y_surf.ravel()})
fittedY=results_formula.predict(exog=onlyX)



## convert the predicted result in an array
fittedY=np.array(fittedY)




# Visualize the Data for Multiple Linear Regression

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(df2['horsepower'],df2['curbweight'],df2['price'],c='red', marker='o', alpha=0.5)
ax.plot_surface(x_surf,y_surf,fittedY.reshape(x_surf.shape), color='b', alpha=0.3)
ax.set_xlabel('Horsepower')
ax.set_ylabel('Curbweight')
ax.set_zlabel('Price')
plt.show()
```

5)

```
getwd()
data=read.csv("CarPrice_Assignment.csv")
mean(data$curbweight)
# H0: Mean curbweight = 2550
# H1: Mean curbweight > 2550
t.test(data$curbweight,mu=2550,alternative ='two.sided',conf.level=0.95)
```

6)

```
getwd()
setwd("C:/Users/admin/Documents")
mydata=read.csv("WHO_data.csv")
View(mydata)
mydatamod=mydata[(mydata$Country=="India"),]
View(mydatamod)
plot(mydatamod$Cumulative_cases,mydatamod$Cumulative_deaths) #, xlab="Covid cases", ylab="Covid deaths", main="Cases vs. Deaths")
```

7)

```
# -*- coding: utf-8 -*-
"""
Created on Mon Jan  3 08:07:10 2022

@author: admin
"""

import pandas as pd
from sklearn.datasets import load_iris
from factor_analyzer import FactorAnalyzer
from factor_analyzer.factor_analyzer import calculate_bartlett_sphericity
from factor_analyzer.factor_analyzer import calculate_kmo
import matplotlib.pyplot as plt
import os
import seaborn as sns
import numpy as np

os.chdir("C:/Users/Lohith/Documents")
df=pd.read_csv('CarPrice_Assignment_FA.csv')
df.info()
df.drop(['car_ID','CarName'],axis=1,inplace=True)
df.info()
# Converting the categorical data into continous was done manually using FIND AND REPLACE in MS Excel.

# Checking the correlation
#x= df.corr(method= 'pearson')
#print(x)
#sns.heatmap(df.corr(method='pearson'),data=df)
#plt.show()
# Bartlett□ s test
#chi_square_value,p_value=calculate_bartlett_sphericity(df)
#print(chi_square_value, p_value)
# Not sure how to interpret this.


# Kaiser-Meyer-Olkin (KMO) Test
```

```
kmo_all,kmo_model=calculate_kmo(df)
print(kmo_model)
# KMO values range between 0 and 1. Value of KMO less than 0.5 is considered inadequate.
# The overall KMO for our data is 0.78, which is pretty good.
# This value indicates that we can proceed with our planned factor analysis.


#Choosing the number of factors
# Create factor analysis object and perform factor analysis
#fa = FactorAnalyzer()
#fa.analyze(df, 25, rotation=None)
#Check Eigenvalues
#ev, v = fa.get_eigenvalues()
#print(ev)

fa = FactorAnalyzer()
fa.fit(df)
eigen_values, vectors = fa.get_eigenvalues()
print(vectors)
# 3 eigen values are greater than 1 therefore,
# NUMBER OF FACTORS = 3


# Create scree plot using matplotlib
plt.scatter(range(1,df.shape[1]+1),vectors)
plt.plot(range(1,df.shape[1]+1),vectors)
plt.title('Scree Plot')
plt.xlabel('Factors')
plt.ylabel('Eigenvalue')
plt.grid()
plt.show()
# It is understandable from the scree plot that the number of factors 3 or 4.

# Create factor analysis object and perform factor analysis
fa = FactorAnalyzer()
fa.set_params(n_factors=6, rotation='varimax')
fa.fit(df)
loadings = fa.loadings_
print(loadings)


# Get variance of each factors
print(fa.get_factor_variance())
# It is in the below format
#                    Factor 1       Factor2       Factor3
# SS Loadings
# Proportion Var
# Cummulative Var

# Total 58% cumulative Variance is explained by the 3 factors.
```

8)

```
# -*- coding: utf-8 -*-
```

```
"""
Created on Fri Jan 7 07:15:19 2022

@author: admin
"""


import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import os
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix


os.chdir("C:/Users/Lohith/Documents")
data=pd.read_csv('Telecom_Data.csv')


data.info()


# regressor variables
x = data.iloc[:, 0:20].values
#print(x)

# regressed variables
y = data.iloc[:, 20].values
#print(y)


xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size = 0.25, random_state = 0)




classifier = LogisticRegression(random_state = 0)
classifier.fit(xtrain, ytrain)

#y_pred = classifier.predict(xtest)

#cm = confusion_matrix(ytest, y_pred)

#print ("Confusion Matrix : \n", cm)
```

9)

```
# -*- coding: utf-8 -*-
"""
Created on Fri Jan 28 13:08:09 2022

@author: admin
```

```
"""


from sklearn.datasets import load_iris
from sklearn.cluster import AgglomerativeClustering
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import dendrogram, linkage

data = load_iris()
df = data.data
print(df)
df = df[:,:]
Z = linkage(df, method = "ward")
dendro = dendrogram(Z)
plt.title('Dendogram')
plt.ylabel('Euclidean distance')
plt.show()
ac = AgglomerativeClustering(n_clusters=3, affinity="euclidean", linkage="ward")

labels = ac.fit_predict(df)
plt.figure(figsize = (8,5))
plt.scatter(df[labels == 0,0] , df[labels == 0,1], c= 'red')
plt.scatter(df[labels == 1,0] , df[labels == 1,1], c= 'blue')
plt.scatter(df[labels == 2,0] , df[labels == 2,1], c= 'green')
plt.scatter(df[labels == 3,0] , df[labels == 3,1], c= 'black')
plt.scatter(df[labels == 4,0] , df[labels == 4,1], c= 'orange')
plt.show()
```