Slip 1

Create 'Position_Salaries' Data set. Build a linear regression model by identifying independent and target variable. Split the variables into training and testing sets. then divide the training and testing sets into a 7:3 ratio, respectively and print them. Build a simple linear regression model.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
dataset = pd.read_csv("Salary.csv")
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:,1].values
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=1/3,
random_state=0)
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)
y_pred = regressor.predict(X_test)
print(y_pred)
import matplotlib.pyplot as plt
plt.scatter(X_test, y_train, color = 'red')
plt.plot(X_train, regressor.predict(X_train), color='blue')
plt.title('Salary vs Experience (Test set)')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.show()
```

Slip 2

Create ‘Salary’ Data set . Build a linear regression model by identifying independent and target variable. Split the variables into training and testing sets and print them. Build a simple linear regression model for predicting purchases.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
dataset = pd.read_csv("Salary.csv")
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:,1].values
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=1/3,
random_state=0)
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)
y_pred = regressor.predict(X_test)
print(y_pred)
import matplotlib.pyplot as plt
plt.scatter(X_test , y_train , color = 'red')
plt.plot(X_train , regressor.predict(X_train) , color = 'green')
plt.title("Salary vs Purchases")
plt.xlabel('Purchases')
plt.ylabel('Salary')
plt.show()
```

Slip 3

Create 'User' Data set having 5 columns namely: User ID, Gender, Age, Estimated Salary and Purchased. Build a logistic regression model that can predict whether on the given parameter a person will buy a car or not.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
data = pd.read_csv('suv_data.csv')
print(data.head(10))
data.info
print("Number of Customers " , len(data))
Gender = pd.get_dummies(data['Gender'] , drop_first = True)
print(Gender.head(5))
data = pd.concat([data , Gender] , axis = 1)
print(data.head(5))
#Dropping User ID and gender column
data.drop(['User ID' , 'Gender' ]  ,axis = 1 , inplace = True)
print(data.head(5))
X = data.drop('Purchased'  , axis = 1)
y = data['Purchased']
#Train and Test Data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,
random_state = 1)
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
model = LogisticRegression(solver = 'liblinear')
```

```
model.fit(X_train,y_train)
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
intercept_scaling=1, max_iter=100, multi_class='warn',n_jobs=None, penalty='l2',
random_state=None, solver='liblinear',tol=0.0001, verbose=0, warm_start=False)
predictions = model.predict(X_test)
print(predictions)
print(classification_report(y_test, predictions))
print("Confusion Matrix: \n",confusion_matrix(y_test, predictions))
print("Accuracy: ",accuracy_score(y_test, predictions))
plt.figure(figsize = (5,5))
sns.distplot(data[data['Purchased']==1]['Age'])
data['EstimatedSalary'].plot.hist()
data['Age'].plot.hist()
sns.countplot(x='Purchased', data = data)
plt.figure(figsize = (20,10))
sns.barplot(x=data['Age'],y=data['Purchased'])
```

Slip 4

Build a simple linear regression model for Fish Species Weight Prediction

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import linear_model
from sklearn.model_selection import train_test_split
data = pd.read_csv('Fish.csv')
print(data.head())
data.isnull().sum()
data.rename(columns={'Length1':'VerticalLen','Length2':'DiagonalLen','Length3':'C
rossLen'},inplace = True)
data.sample(5)
data.shape
data.info()
data.Species.value_counts()
data_sp = data.Species.value_counts()
data_sp = pd.DataFrame(data_sp)
data_sp.T
new_data = data.drop([40])
print("New dimension of the dataset is :-" , new_data.shape)
print(new_data.head())
new_data2 = new_data.drop(['VerticalLen', 'DiagonalLen', 'CrossLen'], axis =1) #
Can also use axis = 'columns'
print('New dimension of dataset is= ', new_data2.shape)
new_data2.head()
sns.boxplot(x = new_data2['Weight'])
plt.title("Outlier Detection based on weight")
def outlier_detection(dataframe):
    Q1 = dataframe.quantile(0.25)
    Q3 = dataframe.quantile(0.75)
    IQR = Q3-Q1
    upper_end = Q3 + 1.5 * IQR
```

```
    lower_end = Q1 - 1.5 * IQR
    Outlier = dataframe[(dataframe>upper_end)| (dataframe<lower_end)]
    return Outlier
outlier_detection(new_data2['Weight'])
sns.boxplot(x = new_data2['Height'])
plt.title("Outlier detection based on height ")
sns.boxplot(x = new_data['Width'])
plt.title("Outlier Detection based on Width")
data3 = new_data2.drop([142 , 143 , 144])
data3.shape
data3.describe().T
X = data3[['Height', 'Width']]
X.head()
y = data3['Weight']
y.head()
X_train,X_test, y_train, y_test = train_test_split(X, y, test_size =0.2, random_state
= 42)
print('X_train dimension= ', X_train.shape)
print('X_test dimension= ', X_test.shape)
print('y_train dimension= ', y_train.shape)
print('y_train dimension= ', y_test.shape)
model = linear_model.LinearRegression()
model.fit(X_train,y_train)
print('coef:' , model.coef_)
print('Intercept:' ,model.intercept_)
print('Score is :' , model.score(X_test , y_test)
predictedWeight = pd.DataFrame(model.predict(X_test), columns=['Predicted
Weight'])
actualWeight = pd.DataFrame(y_test)
actualWeight = actualWeight.reset_index(drop=True) # Drop the index so that we
can concat it, to create new dataframe
df_actual_vs_predicted = pd.concat([actualWeight,predictedWeight],axis =1)
df_actual_vs_predicted.T
def Visualize():
    plt.scatter(X_test['Width'], y_test , color= 'red' , label = 'Actual Weight')
```

```
   plt.scatter(X_test['Width']  , model.predict(X_test) , color = 'green' , label =
'Predicted Weight' )
   plt.xlabel('Width')
   plt.ylabel('Weight')
   plt.rcParams["figure.figsize"] = (10,6)
   plt.title('Actual vs Predicted weight for Test Data')
   plt.legend()
   plt.show()
Visualize()
sns.distplot((y_test-model.predict(X_test)))
plt.rcParams["figure.figsize"] = (10,6) # Custom figure size in inches
plt.title("Histogram of Residuals")
```

Slip 5

Use the iris dataset. Write a Python program to view some basic statistical details like percentile, mean, std etc. of the species of 'Iris-setosa', 'Iris-versicolor' an'Iris-virginica'. Apply logistic regression on the dataset to identify different species (setosa, versicolor, verginica) of Iris flowers given just 4 features: sepal and petal lengths and widths.. Find the accuracy of the model.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.datasets import load_iris
data = sns.load_dataset("iris")
print(data.head())
X = data.iloc[: , :-1]
y = data.iloc[:, -1]
#Split the data 80% on training data and 20% on test data
X_train , X_test , y_train ,y_test = train_test_split(X , y , test_size = 0.2 ,
random_state= 42)
model = LogisticRegression()
model.fit(X_train , y_train)
prediction = model.predict(X_test)
print(prediction)
print()
print(classification_report(y_test , prediction))
print(accuracy_score(y_test , prediction))
def Visualize_iris_dataset():
   plt.xlabel('Features')
   plt.ylabel('Species')
```

```
    pltX = data.loc[: , 'sepal_length']
    pltY = data.loc[: , 'species']
    plt.scatter(pltX , pltY , color = 'blue' , label = 'sepal_length')
    pltX = data.loc[:,'sepal_width']
    pltY = data.loc[: , 'species']
    plt.scatter(pltX , pltY ,color = 'green' ,label='sepal_width')
    pltX = data.loc[:,'petal_length']
    pltY = data.loc[:, 'species']
    plt.scatter(pltX , pltY ,color = 'red' , label='petal_length')
    pltX= data.loc[: , 'petal_width']
    pltY = data.loc[: , 'species']
    plt.scatter(pltX , pltY ,color ='black' , label = 'petal_width')
    plt.legend(loc = 4 , prop={'size':8})
    plt.show()
Visualize_iris_dataset()
```

Slip 6

Create the following dataset in python & Convert the categorical values into numeric format.Apply the apriori algorithm on the above dataset to generate the frequent itemsets and association rules. Repeat the process with different min_sup value.

```
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules

dataset = [

      ["Bread" , "Milk"] ,
      ["Bread" , "Diaper" ,"Beer" ,"Eggs"] ,
      ["Milk" ,"Diaper" ,"Bread" ,"Coke"] ,
      ["Bread" ,"Milk" , "Diaper" ,"Beer"],
      ["Bread" , "Milk" ,"Diaper" , "Coke"],
]
te = TransactionEncoder()
te_array = te.fit(dataset).transform(dataset)
df = pd.DataFrame(te_array , columns = te.columns_)
#Result after Preprocessing
print("Result after Preprocessing")
print(df)
frequent_itemsets_ap = apriori(df ,min_support=0.01 ,use_colnames=True)
print("\n Results after Applying  apriori Alogorithm")
print(frequent_itemsets_ap)
rules_ap = association_rules(frequent_itemsets_ap , metric="confidence" ,
min_threshold=0.8)
frequent_itemsets_ap['length'] = frequent_itemsets_ap['itemsets'].apply(lambda
x: len(x))
print("\n Frequent 2 Item Sets")
```

```
print(frequent_itemsets_ap[frequent_itemsets_ap['length']>=2])
print("\n Frequent 3 Item sets")
print(frequent_itemsets_ap[frequent_itemsets_ap['length']>=3])
print("\n Frequent 4 Item sets")
print(frequent_itemsets_ap[frequent_itemsets_ap['length']>=4])
```

Slip 7

Download the Market basket dataset. Write a python program to read the dataset and display its information. Preprocess the data (drop null values etc.) Convert the categorical values into numeric format. Apply the apriori algorithm on the above dataset to generate the frequent itemsets and association rules.

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns
from csv import reader
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori, association_rules
groceries = []
with open('groceries.csv', 'r') as read_obj:
   csv_reader = reader(read_obj)
   for row in csv_reader:
      groceries.append(row)
items = set(sum(groceries, []))
df = pd.DataFrame(columns=items)
print(df)
# fitting the list and converting the transactions to true and false
encoder = TransactionEncoder()
transactions = encoder.fit(groceries).transform(groceries)
transactions = transactions.astype('int')
df = pd.DataFrame(transactions, columns=encoder.columns_)
df.head()
df.shape
frequent_itemsets = apriori(df, min_support=0.02, use_colnames=True)
frequent_itemsets['length'] = frequent_itemsets['itemsets'].apply(lambda x:
len(x))
frequent_itemsets
frequent_itemsets = frequent_itemsets.sort_values(by='support',
ascending=False)
print(frequent_itemsets)
```

```
# finding top 5 items with minimum support of 2%
frequent_itemsets[ (frequent_itemsets['length'] == 1) &
                  (frequent_itemsets['support'] >= 0.02)][0:5]
# finding itemsets having length 2 and minimum support of 2%
frequent_itemsets[(frequent_itemsets['length'] == 2) &
                 (frequent_itemsets['support'] >= 0.02)]
rules = association_rules(frequent_itemsets, metric='support',
min_threshold=0.02)
rules
rules[(rules['support'] >= 0.02) &
     (rules['lift'] > 1.0)]
```

Slip 8

Download the groceries dataset. Write a python program to read the dataset and display its information. Preprocess the data (drop null values etc.) Convert the categorical values into numeric format. Apply the apriori algorithm on the above dataset to generate the frequent itemsets and association rules.

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns
from csv import reader
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori, association_rules
# reading the dataset
groceries = []
with open('groceries.csv', 'r') as read_obj:
   csv_reader = reader(read_obj)
   for row in csv_reader:
      groceries.append(row)
items = set(sum(groceries, []))
df = pd.DataFrame(columns=items)
print(df)
# fitting the list and converting the transactions to true and false
encoder = TransactionEncoder()
transactions = encoder.fit(groceries).transform(groceries)
# converting the true and false to 1 and 0
transactions = transactions.astype('int')
df = pd.DataFrame(transactions, columns=encoder.columns_)
# viewing the first few rows of the dataframe
df.head()
df.shape
# applying the apriori algorithm
frequent_itemsets = apriori(df, min_support=0.02, use_colnames=True)
frequent_itemsets['length'] = frequent_itemsets['itemsets'].apply(lambda x:
len(x))
```

```
frequent_itemsets
frequent_itemsets = frequent_itemsets.sort_values(by='support',
ascending=False)
print(frequent_itemsets)
# finding top 5 items with minimum support of 2%
frequent_itemsets[ (frequent_itemsets['length'] == 1) &
                  (frequent_itemsets['support'] >= 0.02)][0:5]
# finding itemsets having length 2 and minimum support of 2%
frequent_itemsets[(frequent_itemsets['length'] == 2) &
                 (frequent_itemsets['support'] >= 0.02)]
# finding top 10 association rules with minimum support of 2%
rules = association_rules(frequent_itemsets, metric='support',
min_threshold=0.02)
rules
rules[(rules['support'] >= 0.02) &
    (rules['lift'] > 1.0)]
```

Slip 9

Create your own transactions dataset and apply the above process on your dataset.

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns
from csv import reader
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori, association_rules
groceries = []
with open('groceries.csv', 'r') as read_obj:
   csv_reader = reader(read_obj)
   for row in csv_reader:
      groceries.append(row)
items = set(sum(groceries, []))
df = pd.DataFrame(columns=items)
print(df)
encoder = TransactionEncoder()
transactions = encoder.fit(groceries).transform(groceries)
transactions = transactions.astype('int')
df = pd.DataFrame(transactions, columns=encoder.columns_)
df.head()
df.shape
frequent_itemsets = apriori(df, min_support=0.02, use_colnames=True)
frequent_itemsets['length'] = frequent_itemsets['itemsets'].apply(lambda x:
len(x))
frequent_itemsets
frequent_itemsets = frequent_itemsets.sort_values(by='support',
ascending=False)
print(frequent_itemsets)
frequent_itemsets[ (frequent_itemsets['length'] == 1) &
(frequent_itemsets['support'] >= 0.02) ][0:5]
frequent_itemsets[(frequent_itemsets['length'] == 2) &
```

```
            (frequent_itemsets['support'] >= 0.02)]
rules = association_rules(frequent_itemsets, metric='support',
min_threshold=0.02)
rules
rules[(rules['support'] >= 0.02) &
    (rules['lift'] > 1.0)]
```

Slip 10

Create the following dataset in python & Convert the categorical values into numeric format.Apply the apriori algorithm on the above dataset to generate the frequent itemsets and association rules. Repeat the process with different min_sup value

```
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules
dataset = [

      ["Eggs", "Milk", "Bread"]
      ["Eggs", "Apple"],
      ["Milk", "Bread",],
      ["Apple", "Milk"],
      ["Milk", "Apple", "Bread"],
]

te = TransactionEncoder()
te_array = te.fit(dataset).transform(dataset)
df = pd.DataFrame(te_array , columns = te.columns_)
print("Result after Preprocessing")
print(df)
frequent_itemsets_ap = apriori(df ,min_support=0.01 ,use_colnames=True)
print("\n Results after Applying  apriori Alogorithm")
print(frequent_itemsets_ap)
rules_ap = association_rules(frequent_itemsets_ap , metric="confidence" ,
min_threshold=0.8)
frequent_itemsets_ap['length'] = frequent_itemsets_ap['itemsets'].apply(lambda
x: len(x))
print("\n Frequent 2 Item Sets")
print(frequent_itemsets_ap[frequent_itemsets_ap['length']>=2])
print("\n Frequent 3 Item sets")
print(frequent_itemsets_ap[frequent_itemsets_ap['length']>=3])
```

```
print("\n Frequent 4 Item sets")
print(frequent_itemsets_ap[frequent_itemsets_ap['length']>=4])
```

Slip 11

Create the following dataset in python & Convert the categorical values into numeric format.Apply the apriori algorithm on the above dataset to generate the frequent itemsets and associationrules. Repeat the process with different min_sup values

```
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules
dataset = [

    ["butter", "bread", "Milk"],

    ["butter", "flour", "Milk", "Sagar"],
    ["butter", "eggs", "milk", "salt"],
    ["eggs"],
    ["butter", "flour", "milk", "Salt"],
]
te = TransactionEncoder()
te_array = te.fit(dataset).transform(dataset)
df = pd.DataFrame(te_array , columns = te.columns_)

print("Result after Preprocessing")
print(df)
frequent_itemsets_ap = apriori(df ,min_support=0.01 ,use_colnames=True)
print("\n Results after Applying  apriori Alogorithm")
print(frequent_itemsets_ap)
rules_ap = association_rules(frequent_itemsets_ap , metric="confidence" ,
min_threshold=0.8)
frequent_itemsets_ap['length'] = frequent_itemsets_ap['itemsets'].apply(lambda
x: len(x))
print("\n Frequent 2 Item Sets")
print(frequent_itemsets_ap[frequent_itemsets_ap['length']>=2])
print("\n Frequent 3 Item sets")
```

```
print(frequent_itemsets_ap[frequent_itemsets_ap['length']>=3])
print("\n Frequent 4 Item sets")
print(frequent_itemsets_ap[frequent_itemsets_ap['length']>=4])
```

Slip 12

Create 'heights-and-weights' Data set . Build a linear regression model by identifying independent and target variable. Split the variables into training and testing sets and print them. Build a simple linear regression model for predicting purchases.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
data = pd.read_csv("HeightWeight.csv")
print(data.head())
data.describe()
data.info()
height_values = data["Height(Inches)"].values
weight_values = data["Weight(Pounds)"].values
plt.scatter(weight_values, height_values)
weight_vector = weight_values.reshape(-1,1)
x_train, x_test, y_train, y_test = train_test_split(weight_vector, height_values,
train_size=.8, test_size=.2)
lm = LinearRegression()
lm.fit(x_train, y_train)
y_predict = lm.predict(x_test)
print(f"Train accuracy {round(lm.score(x_train,y_train)*100,2)} %")
print(f"Test accuracy {round(lm.score(x_test,y_test)*100,2)} %")
plt.scatter(x_train,y_train,color='red')
plt.plot(x_test,y_predict)
plt.xlabel("Weight (Pounds)")
plt.ylabel("Height (Inches)")
plt.title("Trained Height Weight Data")
plt.plot
```

Slip 13

Download nursery dataset from UCI. Build a linear regression model by identifying independent and target variable. Split the variables into training and testing sets and print them. Build a simple linear regression model for predicting purchases

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
df = pd.read_csv('nursery_dataset.csv')
df = df.rename(columns={'final evaluation': 'final'})
L = len(df.index)
def print_counts(df):
    for x in df.columns:
        for i, y in zip(df[x].value_counts().index, df[x].value_counts()):
            i = str(i)
            s = f'{i:14} {y/L:4.2f} {y:4d}'
            print(s)
        print('\n')
print_counts(df)
for x in df.drop(['health', 'final'], axis=1).columns:
    lst = list(df[x].value_counts().index)
    dic = {k:i+1 for i, k in enumerate(lst)}
    df[x].replace(dic, inplace=True)
print_counts(df)
dic1 = {'recommended': 2,
        'priority': 3,
        'not_recom': 1}
dic2 = {'not_recom':1,
        'priority':4,
        'spec_prior':5,
```

```
        'very_recom':3,
        'recommend':2}
df['health'].replace(dic1, inplace=True)
df['final'].replace(dic2, inplace=True)
print_counts(df)
ind = (df.loc[:, 'final'] == 2) | (df.loc[:, 'final'] == 3)
df_23 = df[ind].reset_index(drop=True)
df = df[~ind].reset_index(drop=True)
dic3 = {4: 2, 5: 3}
df['final'].replace(dic3, inplace=True)
X = df.iloc[:, :-1]
y = df.iloc[:, -1]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, stratify=y)
model = LinearRegression()
model.fit(X_train, y_train)
yp_train = model.predict(X_train)
yp_test = model.predict(X_test)
print((y_train, yp_train))
X_23, y_23 = df_23.iloc[:,:-1], df_23.iloc[:,-1]
y_pred_23 = model.predict(X_23)
print(y_pred_23)
plt.figure(figsize=(12, 6))
sns.heatmap(df.corr(), annot=True, fmt='.2f');
```

Slip 14

Create the following dataset in python & Convert the categorical values into numeric format.Apply the apriori algorithm on the above dataset to generate the frequent itemsets and association rules. Repeat the process with different min_sup values

```
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules
#Create Dataset
dataset = [

    ["Apple" , "Mango" , "Banana"] ,

    ["Mango" , "Banana" , "Cabbage" , "Carrots"] ,
    ["Mango" ,"banana" , "Carrots" ] ,
    ["Mango" , "Carrots"],

]

#Convert the list to dataframe with boolean Vlaues
te = TransactionEncoder()
te_array = te.fit(dataset).transform(dataset)
df = pd.DataFrame(te_array , columns = te.columns_)
#Result after Preprocessing
print("Result after Preprocessing")
print(df)
#Find the frequently occuring itemsets using Apriori Algorithm:-
frequent_itemsets_ap = apriori(df ,min_support=0.01 ,use_colnames=True)
print("\n Results after Applying  apriori Alogorithm")
print(frequent_itemsets_ap)
rules_ap = association_rules(frequent_itemsets_ap , metric="confidence" ,
min_threshold=0.8)
frequent_itemsets_ap['length'] = frequent_itemsets_ap['itemsets'].apply(lambda
x: len(x))
```

```
print("\n Frequent 2 Item Sets")
print(frequent_itemsets_ap[frequent_itemsets_ap['length']>=2])
print("\n Frequent 3 Item sets")
print(frequent_itemsets_ap[frequent_itemsets_ap['length']>=3])
print("\n Frequent 4 Item sets")
print(frequent_itemsets_ap[frequent_itemsets_ap['length']>=4])
```

Slip 15

Create the following dataset in python & Convert the categorical values into numeric format.Apply the apriori algorithm on the above dataset to generate the frequent itemsets and association rules. Repeat the process with different min_sup values

```
  import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules
dataset = {

      'Company':['Tata' , 'MG', 'KIA' , 'Hyundai'] ,
      'Model': ['Nexon' , 'Altos' ,'Seltos' , 'Creta'],
      'Year' : [2017 , 2021 , 2019 , 2015]
}
df = pd.DataFrame(dataset, index=['0',
                        '1',
                        '2',
                        '3'])

print(df)
te = TransactionEncoder()
te_array = te.fit(df).transform(df)
df = pd.DataFrame(te_array , columns = te.columns_)
print("Result after Preprocessing")
print(df)
frequent_itemsets_ap = apriori(df ,min_support=0.01 ,use_colnames=True)
print("\n Results after Applying  apriori Alogorithm")
print(frequent_itemsets_ap)
rules_ap = association_rules(frequent_itemsets_ap , metric="confidence" ,
min_threshold=0.8)
frequent_itemsets_ap['length'] = frequent_itemsets_ap['itemsets'].apply(lambda
x: len(x))
print("\n Frequent 2 Item Sets")
```

```
print(frequent_itemsets_ap[frequent_itemsets_ap['length']>=2])
print("\n Frequent 3 Item sets")
print(frequent_itemsets_ap[frequent_itemsets_ap['length']>=3])
print("\n Frequent 4 Item sets")
print(frequent_itemsets_ap[frequent_itemsets_ap['length']>=4])
```

Slip 16

Consider any text paragraph. Preprocess the text to remove any special characters and digits. Generate the summary using extractive summarization process

```
import bs4 as bs
import urllib.request
import re
import nltk
nltk.download('punkt')
nltk.download('stopwords')
scraped_data =
urllib.request.urlopen('https://en.wikipedia.org/wiki/Severe_acute_respiratory_s
yndrome_coronavirus_2')
article = scraped_data.read()
parsed_article = bs.BeautifulSoup(article,'lxml')
paragraphs = parsed_article.find_all('p')
article_text = ""
for p in paragraphs:
   article_text += p.text
# Removing Square Brackets and Extra Spaces
article_text = re.sub(r'\[[0-9]*\]', ' ', article_text)
article_text = re.sub(r'\s+', ' ', article_text)
# Removing special characters and digits
formatted_article_text = re.sub('[^a-zA-Z]', ' ', article_text )
formatted_article_text = re.sub(r'\s+', ' ', formatted_article_text)
sentence_list = nltk.sent_tokenize(article_text)
stopwords = nltk.corpus.stopwords.words('english')
word_frequencies = {}
for word in nltk.word_tokenize(formatted_article_text):
   if word not in stopwords:
      if word not in word_frequencies.keys():
         word_frequencies[word] = 1
      else:
         word_frequencies[word] += 1
```

```
    maximum_frequncy = max(word_frequencies.values())
for word in word_frequencies.keys():
    word_frequencies[word] = (word_frequencies[word]/maximum_frequncy)
    sentence_scores = {}
for sent in sentence_list:
    for word in nltk.word_tokenize(sent.lower()):
        if word in word_frequencies.keys():
            if len(sent.split(' ')) < 30:
                if sent not in sentence_scores.keys():
                    sentence_scores[sent] = word_frequencies[word]
                else:
                    sentence_scores[sent] += word_frequencies[word]
import heapq
summary_sentences = heapq.nlargest(7, sentence_scores,
key=sentence_scores.get)

summary = ' '.join(summary_sentences)
print(summary)
```

Slip 17

Consider text paragraph.*So, keep working. Keep striving. Never give up. Fall down seven times, get up eight. Ease is a greater threat to progress than hardship. Ease is a greater threat to progress than hardship. So, keep moving, keep growing, keep learning. See you at work.*Preprocess the text to remove any special characters and digits. Generate the summary using extractive summarization process.

```
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize, sent_tokenize
text = """ keep working. Keep striving. Never give up. Fall down seven times, get
up eight. Ease is a greater threat to progress than hardship. Ease is a greater
threat to progress than
hardship. So, keep moving, keep growing, keep learning. See you at work"""
stopWords = set(stopwords.words("english"))
words = word_tokenize(text)
freqTable = dict()
for word in words:
    word = word.lower()
    if word in stopWords:
        continue
    if word in freqTable:
        freqTable[word] += 1
    else:
        freqTable[word] = 1
sentences = sent_tokenize(text)
sentenceValue = dict()
for sentence in sentences:
    for word, freq in freqTable.items():
        if word in sentence.lower():
            if sentence in sentenceValue:
                sentenceValue[sentence] += freq
            else:
                sentenceValue[sentence] = freq
```

```
sumValues = 0
for sentence in sentenceValue:
   sumValues += sentenceValue[sentence]
average = int(sumValues / len(sentenceValue))
summary = ''
for sentence in sentences:
   if (sentence in sentenceValue) and (sentenceValue[sentence] > (1.2 * average)):
      summary += " " + sentence
print(summary)
```

Slip 18

Consider any text paragraph. Remove the stopwords. Tokenize the paragraph to extract words and sentences. Calculate the word frequency distribution and plot the frequencies. Plot the wordcloud of the text.

```
import warnings
warnings.filterwarnings('ignore')
import numpy as np
import pandas as pd
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize, sent_tokenize
import string
import collections
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator
import matplotlib.cm as cm
import matplotlib.pyplot as plt
text = """ keep working. Keep striving. Never give up. Fall down seven times, get
up eight. Ease is a greater threat to progress than hardship. Ease is a greater
threat to progress than
hardship. So, keep moving, keep growing, keep learning. See you at work"""
stopWords = set(stopwords.words("english"))
words = word_tokenize(text)
freqTable = dict()
for word in words:
    word = word.lower()
    if word in stopWords:
        continue
    if word in freqTable:
        freqTable[word] += 1
    else:
        freqTable[word] = 1
sentences = sent_tokenize(text)
sentenceValue = dict()
for sentence in sentences:
```

```
    for word, freq in freqTable.items():
        if word in sentence.lower():
            if sentence in sentenceValue:
                sentenceValue[sentence] += freq
            else:
                sentenceValue[sentence] = freq
sumValues = 0
for sentence in sentenceValue:
    sumValues += sentenceValue[sentence]
wordcloud_spam = WordCloud(background_color="white").generate(text)
plt.figure(figsize = (20,20))
plt.imshow(wordcloud_spam, interpolation='bilinear')
plt.axis("off")
plt.show()
```

Slip 19

Download the movie_review.csv dataset from Kaggle by using the following link https://www.kaggle.com/nltkdata/movie review/version/3?select=movie_review.csv to perform sentiment analysis on above dataset and create a word cloud

```
import numpy as np
import pandas as pd
import re
import string
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem.snowball import SnowballStemmer
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator
import matplotlib.cm as cm
import matplotlib.pyplot as plt
data = pd.read_csv('movie_review.csv')
print(data.head())
data.isnull()
data.dtypes
data.shape
stemmer  = SnowballStemmer(language='english')
def preprocessing(phrase):
    lower = [phrase.lower() for phrase in phrase]
    no_punct = [text.translate(str.maketrans('','',string.punctuation)) for text in
lower]
    stem = [stemmer.stem(i) for i in no_punct]
    join = ["".join(text) for text in stem]
    return join
label = data['text']
data1 = preprocessing(data["tag"])
print(data1)
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer= TfidfVectorizer(stop_words = 'english')
```

```
data2  = vectorizer.fit_transform(data1)
#tf_x_test = vectorizer.transform(test_final)
print(data2)
from sklearn.svm import LinearSVC
clf = LinearSVC(random_state=0)
clf.fit(data2,label)
print(clf)
y_pred=clf.predict(data2)
print(y_pred)

text = """
      films adapted from comic books have had plenty of success , whether they're about superheroes ( batman , superman , spawn ) , or geared toward kids ( casper ) or the arthouse crowd ( ghost world ) , but there's never really been a comic book like from hell before .for starters , it was created by alan moore ( and eddie campbell ) , who brought the medium to a whole new level in the mid '80s with a 12-part series called the watchmen .to say moore and campbell thoroughly researched the subject of jack the ripper would be like saying michael jackson is starting to look a little odd .the book ( or " graphic novel , " if you will ) is over 500 pages long and includes nearly 30 more that consist of nothing but footnotes .in other words , don't dismiss this film because of its source.if you can get past the whole comic book thing , you might find another stumbling block in from hell's directors , albert and allen hughes .getting the hughes brothers to direct this seems almost as ludicrous as casting carrot top in , well , anything , but riddle me this : who better to direct a film that's set in the ghetto and features really violent street crime than the mad geniuses behind menace ii society ?
the ghetto in question is , of course , whitechapel in 1888 london's east end .
"""
wordcloud_spam = WordCloud(background_color="white").generate(text)

# Lines 2 - 5
plt.figure(figsize = (20,20))
plt.imshow(wordcloud_spam, interpolation='bilinear')
plt.axis("off")
plt.show()
```

Slip 20

Consider text paragraph.*"""Hello all, Welcome to Python Programming Academy. Python Programming Academy is a nice platform to learn new programming skills. It is difficult to get enrolled in this Academy."""*Remove the stopwords.

```
import warnings
warnings.filterwarnings('ignore')
#loading all necessary libraries
import numpy as np
import pandas as pd
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize, sent_tokenize
import string
import collections
text = """ Hello all, Welcome to Python Programming Academy.
Python Programming Academy is a nice platform to learn new programming skills.
It is difficult to get enrolled in this Academy."""
# Tokenizing the text
stopWords = set(stopwords.words("english"))
words = word_tokenize(text)
print(words)
```

Slip 21

Build a simple linear regression model for User Data.

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
x= np.array([2.4,5.0,1.5,3.8,8.7,3.6,1.2,8.1,2.5,5,1.6,1.6,2.4,3.9,5.4])
y = np.array([2.1,4.7,1.7,3.6,8.7,3.2,1.0,8.0,2.4,6,1.1,1.3,2.4,3.9,4.8])
n = np.size(x)
X_train , x_test , Y_train , y_test = train_test_split(x  , y , test_size = 0.2 ,
random_state = 1)
print(X_train.shape)
print(Y_train.shape)
print(x_test.shape)
print(y_test.shape)
regressor = LinearRegression()
X_train = X_train.reshape(-1,1)
regressor.fit(X_train , Y_train)
print(regressor.intercept_)
print(regressor.coef_)
x_test = x_test.reshape(-1,1)
y_pred = regressor.predict(x_test)
print(y_pred)
```

Slip 22

Consider any text paragraph. Remove the stopwords.

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

import numpy as np
import pandas as pd
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize , sent_tokenize
text = """
      Hello  , Hi everyone It's John your lovely data scientists  If you want any
help   , feel free
      to reach me . Okaayyy Thank you


"""
StopWords = set(stopwords.words("english"))
words = word_tokenize(text)
print(words)
print(StopWords)
```

Slip 23

Consider any text paragraph. Preprocess the text to remove any special characters and digits.

```
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize, sent_tokenize

text =""" keep working. Keep striving. Never give up. Fall down seven times, get up eight. Ease is a greater  threat to progress  than hardship. Ease is a greater threat to progress than hardship. So, keep moving, keep growing, keep learning. See you at work"""
stopWords = set(stopwords.words("english"))

words = word_tokenize(text)

freqTable = dict()
for word in words:
    word = word.lower()
    if word in stopWords:
        continue
    if word in freqTable:
        freqTable[word] += 1
    else:
        freqTable[word] = 1

sentences = sent_tokenize(text)
sentenceValue = dict()

for sentence in sentences:
    for word, freq in freqTable.items():
        if word in sentence.lower():
            if sentence in sentenceValue:
                sentenceValue[sentence] += freq
            else:
```

```
            sentenceValue[sentence] = freq
sumValues = 0
for sentence in sentenceValue:
   sumValues += sentenceValue[sentence]

average = int(sumValues / len(sentenceValue))
summary = ''
for sentence in sentences:
   if (sentence in sentenceValue) and (sentenceValue[sentence] > (1.2 * average)):
      summary += " " + sentence
print(summary)
```

Slip 24

Consider the following dataset : https://www.kaggle.com/datasets/datasnaek/youtube-new?select=INvideos.csv
Write a Python script for the following :
i. Read the dataset and perform data cleaning operations on it.
ii. Find the total views, total likes, total dislikes and comment count.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import re
import plotly.express as px
import plotly.graph_objects as go
df_videos = pd.read_csv('INvideos.csv', error_bad_lines=False)
df_videos.head()
df_videos.shape
df_videos.columns
df_videos.isnull().sum()
new_df =
df_videos['video_id'].value_counts().rename_axis('video_id').head(10).reset_inde
x(name='counts')
fig = px.bar(new_df, x="video_id", y="counts")
fig.show()
df1 =pd.DataFrame(df_videos.channel_title.value_counts())
df1.columns=['times channel got trenidng']
df1.head(6)
df_channel
=pd.DataFrame(df_videos.groupby(by=['channel_title'])['views'].mean()).sort_val
ues(by='views',ascending=False)
df_channel.head(10).plot(kind='bar');
plt.title('Most viewed channels');
sns.regplot(data=df_videos,x='views',y='likes')
plt.title('Regression plot for views & likes')
sns.regplot(data=df_videos,x='views',y='dislikes')
plt.title('Regression plot for views & dislikes')
```

Slip 25

Consider the following dataset : https://www.kaggle.com/datasets/seungguini/youtube-comments-for-covid19-relatedvideos?select=covid_2021_1.csv
Write a Python script for the following :
i. Read the dataset and perform data cleaning operations on it.
ii. ii. Tokenize the comments in words. iii. Perform sentiment analysis and find the percentage of positive, negative and neutral comments..

```
from mpl_toolkits.mplot3d import Axes3D
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import numpy as np
import os
import pandas as pd
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize, sent_tokenize
nRowsRead = 1000
df1 = pd.read_csv('youtube_comments_coronavirus.csv')
nRow, nCol = df1.shape
print(f'There are {nRow} rows and {nCol} columns')
df1.head(5)
df1.isnull().sum()
df1.shape
to_drop = ['url' , 'title' ,'views' , 'likes', 'dislikes']
df1.drop(to_drop, inplace=True, axis=1)
print(df1.head())
data = df1['comment_text'].map(lambda x: x.lstrip('+-').rstrip('aAbBcC'))
print(data.head())
text = """
Everyone who reads this message may your mother lives a 100 years.
Just imagine, years from now kids are gonna watch this in class and take a test on this pandemic
Who else started manually breathing while watching the part of the video talking and showing lungs See you in 10 years when it gets recommended to you!!! When I get the vaccine I hope I would get the War Hammer titan.
```

Canâ€™t wait for this to be all over Jet lock airplane travellerscoronavirus without air plane hospital becoming airplanes to spread corona virus
I have had covid for 12 days and my dad has had it for 17 days I hope we get well soon and I hope all of you guys that are struggling yk with the virus get well soon
This virus has really taught me to appreciate what you and the people you have because you never know when youâ€™ll see them againhere we are 2021 less then a year latter! HUMANITY is extraordinaryMy grandma suffers from corona and is ICU, I would ask everyone to pray for her fast recovery
ðŸ˜¢Thestart Everyone be like: COVER YOUR SNEEZE
The festive dime phylogenetically flower because arch longitudinally seal absent a cold playroom. sweet, sick spleen
Corona" is the Spanish, Catalan, Hungarian, and Italian word for "crown".prayers to everyone who got covid-19
Who ever reading this - I pray for you and your family's good health.ÐŸÐ¾Ð»Ñ,Ð¾Ñ€Ñ‹ Ð½ÐµÐ´ÐµÐ»Ð¸ Ñ,ÐµÐ¼Ð¿ÐµÑ€Ð°Ñ,ÑƒÑ€Ð°, Ð½ÐµÐ´ÐµÐ»ÑŽ Ð´Ð¸Ð°Ñ€ÐµÑ , Ñ,Ñ€Ð¸ Ð´Ð½Ñ Ð´Ð°Ð²Ð»ÐµÐ½Ð¸Ðµ Ð¸ ÐºÑ€Ð¾Ð²ÑŒ Ð¸Ð· Ð´ÐµÑ ÐµÐ½, Ð¿Ð¾Ñ,Ð¾Ð¼ Ð½ÐµÐ´ÐµÐ»Ñ ÐºÐ°Ñˆ Ð»Ñ Ð»Ð°Ð±Ð¾Ñ Ñ,Ð¸.
My mother is experiencing time to time fever,extreme weakness,headaches,mild cough,chills and back pain... I'm confused n scared about her plz pray for her recovery and healthâ ¤ï¸
That's life and you never know what tomorrow brings !!
Every vacation we swolling corona virus germs in flight zuzubi matter 6 months problem
Dear 2019, Iâ€™m sorry Iâ€™ve ever complained about youðŸ˜ðŸ˜,
99.5% live after infection of C19 you end up living
It was predicted that coronavirus would end by the end of this year so buckle up
Some people just die from this so sad found out wow god bless them and those people going through losing there fam man wow.
I hate having to isolate for this long. I got it then 2 weeks later my kids got it. I have been inside for 3 weeks. It's really affecting my mental health.
The lonely objective ipsilaterally mine because invention eventually dream amongst a abortive pants. hurried, fine zinc
Ã“timo vÃdeo, Ã³tima explicaÃ§Ã£o Simplesmente,ameiiiâ ¤ï¸
This channel is PERFECT ABSOLUTELY PERFECT, other than the graphics and amazing animations, this thing has been helpful to know! Thanks to you... no all of

nucleus medical media i found out the perfect video to share to my classmates at home. Keep up thew good work, stay safe and stay strong!
Thanks So Much Dr Otor for your wonderful change in my life with your Herbal Medicine curing my Herpes illness I love you so much Dr Otor
RIP to all that have died from Covid 19. We will never forget this terrible tragedy.
ã‚³ãƒãƒŠäººé¡žæ»…ã ³ã‚‹ã€€ã€€ç—…åŽŸå¾®ç”Ÿç‰©å·»ã è¿”ã —
how could you say that is sars cov2? when it hasn't been isolated or photographed? are you parroting the theories?
My body does more work then me, itâ€™s fighting wars while I sit on my couch
"""
stopWords = set(stopwords.words("english"))
words = word_tokenize(text)
print(words)

Slip 26

Consider text paragraph. """Hello all, Welcome to Python Programming Academy. Python Programming Academy is a nice platform to learn new programming skills. It is difficult to get enrolled in this Academy.""" Preprocess the text to remove any special characters and digits. Generate the summary using extractive summarization process.

```
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize, sent_tokenize
text = """ Hello all, Welcome to Python Programming Academy. Python
Programming Academy
is a nice platform to learn new programming skills. It is difficult to get enrolled in
this Academy."""
StopWords = set(stopwords.words("english"))
words = word_tokenize(text)
print(words)
freqtable = dict()
for word in words:
   word = word.lower()
   if word in StopWords:
      continue
   if word in freqtable:
      freqtable[word]+=1
   else:
      freqtable[word] = 1
sentences = sent_tokenize(text)
sentenceValue = dict()

for sentence in sentences:
   for word, freq in freqtable.items():
      if word in sentence.lower():
         if sentence in sentenceValue:
            sentenceValue[sentence] += freq
         else:
```

```
            sentenceValue[sentence] = freq
sumValues = 0
for sentence in sentenceValue:
   sumValues += sentenceValue[sentence]
print(sentences)
average = int(sumValues / len(sentenceValue))
summary = ''
for sentence in sentences:
   if(sentence in sentenceValue) and (sentenceValue[sentence]>(1.2*average)):
      summary +=" "+sentence
print(summary)
```

Slip 27

Create your own transactions dataset and apply the above process on your dataset

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns
from csv import reader
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori, association_rules
groceries = []

with open('groceries.csv', 'r') as read_obj:
   csv_reader = reader(read_obj)
   for row in csv_reader:
      groceries.append(row)

items = set(sum(groceries, []))
df = pd.DataFrame(columns=items)
print(df)
encoder = TransactionEncoder()
transactions = encoder.fit(groceries).transform(groceries)
transactions = transactions.astype('int')
df = pd.DataFrame(transactions, columns=encoder.columns_)
df.head()
df.shape
frequent_itemsets = apriori(df, min_support=0.02, use_colnames=True)
frequent_itemsets['length'] = frequent_itemsets['itemsets'].apply(lambda x:
len(x))
frequent_itemsets
frequent_itemsets = frequent_itemsets.sort_values(by='support',
ascending=False)
```

```
print(frequent_itemsets)
frequent_itemsets[ (frequent_itemsets['length'] == 1) &
                  (frequent_itemsets['support'] >= 0.02) ][0:5]
frequent_itemsets[(frequent_itemsets['length'] == 2) &
                 (frequent_itemsets['support'] >= 0.02)]
rules = association_rules(frequent_itemsets, metric='support',
min_threshold=0.02)
rules
rules[(rules['support'] >= 0.02) &
     (rules['lift'] > 1.0)]
```

Slip 28

Build a simple linear regression model for Car Dataset.

```
import warnings
warnings.filterwarnings('ignore')
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
Scaler = StandardScaler()
cars = pd.read_csv('CarPrice_Assignment.csv')
cars.head()
cars.shape
cars.describe()
cars.info()
cars.loc[cars.duplicated()]
cars.columns
plt.figure(figsize=(20,8))
plt.subplot(1,2,1)
plt.title('Car Price Distribution Plot')
sns.distplot(cars.price)
plt.subplot(1,2,2)
plt.title('Car Price Spread')
sns.boxplot(y=cars.price)
plt.show()
plt.figure(figsize=(25, 6))
df =
pd.DataFrame(cars.groupby(['horsepower'])['price'].mean().sort_values(ascendin
g = False))
df.plot.bar()
plt.title('HorsePower vs Average Price')
plt.show()
```

```
df = pd.DataFrame(cars.groupby(['stroke'])['price'].mean().sort_values(ascending
= False))
df.plot.bar()
plt.title('Stroke vs Average Price')
plt.show()
df =
pd.DataFrame(cars.groupby(['enginesize'])['price'].mean().sort_values(ascending
= False))
df.plot.bar()
plt.title('Engine_size vs Average Price')
plt.show()
cars = pd.read_csv('CarPrice_Assignment.csv')
cars.head()
X = cars.iloc[:, :-1].values
y = cars.iloc[:,1].values
x =np.array(cars['price'])
y =np.array(cars['horsepower'])
print(X)
print(y)
print(x)
print(y)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=1/3,
random_state=0)
X_train.shape
y_train.shape
X_train= X_train.reshape(-1 , 1)
y_train = y_train.reshape(-1 , 1)
X_test = X_test.reshape(-1 , 1)
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)
y_pred = regressor.predict(X_test)

print(y_pred)
```

Slip 29

Build a logistic regression model for Student Score Dataset.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
lc = LabelEncoder()
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
df = pd.read_csv('StudentsPerformance.csv')
print(df.head())
df.shape
df.describe()
df["mean_Score"] = ((df["math score"]+df["reading score"]+df["writing
score"])/3).round()
df.head()
df['gender'].value_counts()
df['gender'] = lc.fit_transform(df['gender'])
df['race/ethnicity'] = lc.fit_transform(df['race/ethnicity'])
df['parental level of education'] = lc.fit_transform(df['parental level of
education'])
df['lunch'] = lc.fit_transform(df['lunch'])
df['test preparation course'] = lc.fit_transform(df['test preparation course'])
df.head()
df['test preparation course'].value_counts()
df = df.drop(['math score', 'writing score', 'reading score'],axis = 1)
df.head()
```

```
y = df['mean_Score']
x = df.drop(['mean_Score'], axis  = 1)
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size = 0.2, random_state
= 0)
model = LogisticRegression(solver='liblinear', random_state=0)
print(model)
model.fit(x_train, y_train)
predictions = model.predict(x_test)
print(predictions)
difference = abs(predictions - y_test)
print(difference)
difference.mean()
labels = ['None', 'Completed']
colors = ['blue', 'gold']
plt.pie(df['test preparation course'].value_counts() , labels = labels, colors =
colors)
sns.countplot(x = df['gender'], hue = df['race/ethnicity'])
plt.figure(figsize = (12,6))
sns.pairplot(df)
plt.show()
plt.figure(figsize = (12,6))
sns.heatmap(df.corr())
plt.show()
```

Slip 30

Create the dataset . transactions = [['eggs', 'milk','bread'], ['eggs', 'apple'], ['milk', 'bread'], ['apple', 'milk'], ['milk', 'apple', 'bread']] .
Convert the categorical values into numeric format.Apply the apriori algorithm on the above dataset to generate the frequent itemsets and association rules.

```
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules
dataset = [

    ["eggs" , "milk" , "bread"] ,
    ["eggs" , "apple"] ,
    ["milk" ,"bread" ] ,
    ["Apple" ,"milk"],
    ["Milk" , "apple" ,"bread"],
]
te = TransactionEncoder()
te_array = te.fit(dataset).transform(dataset)
df = pd.DataFrame(te_array , columns = te.columns_)
print("Result after Preprocessing")
print(df)
frequent_itemsets_ap = apriori(df ,min_support=0.01 ,use_colnames=True)
print("\n Results after Applying  apriori Alogorithm")
print(frequent_itemsets_ap)
rules_ap = association_rules(frequent_itemsets_ap , metric="confidence" ,
min_threshold=0.8)
frequent_itemsets_ap['length'] = frequent_itemsets_ap['itemsets'].apply(lambda
x: len(x))
print("\n Frequent 2 Item Sets")
print(frequent_itemsets_ap[frequent_itemsets_ap['length']>=2])
print("\n Frequent 3 Item sets")
print(frequent_itemsets_ap[frequent_itemsets_ap['length']>=3])
print("\n Frequent 4 Item sets")
print(frequent_itemsets_ap[frequent_itemsets_ap['length']>=4])
```