----------LAB 01 (DATA CLEANING)----------

#import libraries

```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
```

#load data

```
raw_data=pd.read_csv("loan.csv")
```

#display data

```
raw_data
```

#dsiplay first 5 rows

```
raw_data.head(5)
```

 # get the num of rows & columns

```
raw_data.shape
```

#get data types of the columns

```
raw_data.dtypes
```

#explore a relevent column(describe)

```
raw_data['loan_amnt'].describe()
```

#drop columns

```
raw_data= raw_data.drop(['zip_code', 'policy_code', 'application_type', 'last_credit_pull_d',
'verification_status', 'pymnt_plan', 'funded_amnt_inv', 'sub_grade', 'out_prncp',
'out_prncp_inv', 'total_pymnt_inv', 'total_pymnt', 'total_pymnt_inv', 'total_rec_prncp',
'total_rec_int', 'total_rec_late_fee', 'recoveries', 'collection_recovery_fee', 'last_pymnt_d',
'last_pymnt_amnt', 'initial_list_status'], axis =1)
```

#check whther they removed

```
raw_data
```

#remove null value columns

```
col_num=0
TotalObjects = raw_data.shape[0]
print ("Column\t\t\t\t\t Null Values%")
for x in raw_data:
    nullCount = raw_data[x].isnull().sum();
    nullPercent = nullCount*100 / (TotalObjects)
    if nullCount > 0 and nullPercent > 20 :
        col_num=col_num+1
        raw_data.drop(x, axis=1,inplace=True)
    print(str(x)+"\t\t\t\t\t "+str(nullPercent))
print ("A total of "+str(col_num)+" deleted !")
```

raw_data.shape

```
# replace with mean and unknown
raw_data['emp_title'].fillna('Unknown',inplace = True)
raw_data['dti'].fillna(0,inplace=True)
raw_data['revol_util'].fillna(raw_data['revol_util'].mean(),inplace = True)

#return unquie values in data set
pd.unique(raw_data['emp_length'].values)


def CalculateEmployeeLength(year):
    if year == '< 1 year':
        return 0.5
    elif year == '10+ years':
        return 10
    else:
        yr=str(year)
        return yr.rstrip(' years')

# In[17]:
raw_data['emp_length']=raw_data['emp_length'].apply(CalculateEmployeeLength)

# In[18]:
raw_data
```

#visualize data

```
def CalculateLoanRanges(value):
    if value <= 5000:
        return '5K and Below'
    if value > 5000 and value <= 10000:
        return '5K-10K'
    if value > 10000 and value <= 15000:
        return '10K-15K'
    if value > 15000 and value <= 20000:
        return '15K-20K'
    if value > 20000 and value <= 25000:
        return '20K-25K'
    if value > 25000 and value <= 30000:
        return '25K-30K'
    if value > 30000 :
        return '30K and Above'
    return 'Other'
```

#call the function

```
loan_ranges = raw_data['loan_amnt'].apply(CalculateLoanRanges)
```

#count how many loan amount in each range

```
loan_ranges.value_counts()
```

#draw a pie chart

```
f = plt.figure()
loan_ranges.value_counts().plot.pie(autopct='%1.0f%%',)
plt.title('Pie Chart of Loan Amount')
```

#bar chart

```
pur = raw_data['purpose'].value_counts()
pur.plot(kind='bar')
```

#save cleaned data into new CSV file

```
raw_data.to_csv('cleaned_loans2007.csv', index=False,encoding='utf-8')
```

----------LAB 02 (Association Rule mining) ----------

 # Import libraries

```
import pandas as pd
from apyori import apriori
```

#import data set

```
store_data = pd.read_csv("store_data.csv", header=None)
```

#show the number of data you want

```
display(store_data.head(6))
```

#display num of transactions , maximum number of item in transcation

```
store_data.shape
```

#convert the data frame into a list

```
records = []
for i in range(1, 7501):
  records.append([str(store_data.values[i, j]) for j in range(0, 20)])
```

records

#apply the apriori algorithm ,

 # 2; you should have atleast 2

#lift ; confidence /support

```
association_rules = apriori(records, min_support=0.0045, min_confidence=0.2,
 min_lift=3, min_length=2)
```

#make the rules as a list

```
association_results = list(association_rules)
```

#view the result

```
association_results
```

----------LAB 03 (Clustering) ----------

#import libraries

```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
```

```
#data visualize based on matplot
import seaborn as sns
sns.set()
from sklearn.cluster import KMeans

#raw_data ; any variable name we select
#pd ; pandas object
raw_data=pd.read_csv("Countries.csv")

# describe columns, get description about columns
raw_data['Latitude'].describe()

# show first five raws of the data set
raw_data.head()

#plot the data
#matplotlib.pyplot.scatter(x, y)
#we import both matplot and pyplot as plt
# use names we defined
plt.scatter(raw_data['Longitude'],raw_data['Latitude'])

#extract data to a new data frame
# : means all the records in our dataset
# column 1 and 3 only
# give NumOfColumns + 1
new_data = raw_data.iloc[:,1:3]
```

#view new data frame

```
new_data
```

#clustering

#initialize a clustering

```
kmeans = KMeans(2)
```

#performe clustering

#give the correct data set

```
kmeans.fit(new_data)
```

#extract the result

```
identified_clusters = kmeans.fit_predict(new_data)
identified_clusters
```

#how many clusters

#length

```
len(identified_clusters)
```

#copy oiginal data set into country_cluster data set

#copy original data set

```
country_cluster = raw_data.copy()
```

#view the data set

```
country_cluster
```

#add a new column into data set and assign values to that columan

```
country_cluster['ClusterNo'] = identified_clusters
```

#view the data again

```
country_cluster
```

#plot the data

#take the correct data set

#plot clusterNo as 3rd parameter

#cmap rainbow; view thr graph more attractively

```
plt.scatter(country_cluster['Longitude'],country_cluster['Latitude'],c =
country_cluster['ClusterNo'],cmap = 'rainbow')
```

#find the optimal No of clusters this data set have

#see the WCSS values

```
wcss=[]
for i in range(1,11):
   kmeans = KMeans(i)
   kmeans.fit(new_data)
   wcss_iter = kmeans.inertia_
   wcss.append(wcss_iter)
```

#view the WCSS values for the data points

```
wcss
```

#plot the WCSS values

```
number_clusters = range(1,11)
plt.plot(number_clusters,wcss)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('Within-cluster Sum of Squares')
```

 # get the optimal number of cluster by elbow method

 # 2 or 3 clusters

----------LAB 04 (Classification) ----------

#import libries

```
import pandas as pd
```

 # use classifer and split libry fro, sklearn library

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
```

#import the data set

```
data_zoo = pd.read_csv("zoo.csv")
```

#see first five raws

```
data_zoo.head()
```

#see raws and columns

```
data_zoo.shape
```

 # assign 1 - 17 data set into x variable

```
x = data_zoo.iloc[:,1:17]
x.shape
```

 # assign last column to the y variable

```
y = data_zoo.iloc[:,17]
```

#when only one column not show the column number

```
y.shape
```

#first five y data

```
y.head()
```

#first five x data

```
x.head()
```

#devide data set to train and test

#test data part size is 0.25

```
x_train,x_test,y_train,y_test = train_test_split(x,y, test_size=0.25)
```

#view the x_train size

```
x_train.shape
```

#view the x_test size

```
x_test.shape
```

#75 raws for train

#only one column

```
y_train.shape
```

#25 raws for test

#only one column

```
y_test.shape
```

#build the classfier

```
zoo_classifier = DecisionTreeClassifier(random_state=0)
zoo_classifier
```

#train the classifer using train data set x

```
zoo_classifier.fit(x_train,y_train)
```

#test the accuracy using x_test and y_test

```
zoo_classifier.score(x_test,y_test)
```

#make prediction

#predict command is use

```
zoo_classifier.predict(x_test[10:15])
```

#see the result

```
y_test[10:15]
```

----------LAB 05 (Regression) ----------

#import pandas, numpy, matpolot

```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
```

#install statsemodels

 # import statmodel lib for regression

```
import statsmodels.api as sm
from sklearn.linear_model import LinearRegression
```

#import data set

```
salary_data=pd.read_csv("salary_data.csv")
```

#read the data first five

```
salary_data.head()
```

#display entire data set

```
salary_data
```

#to see the raws and columns

```
salary_data.shape
```

#get a statical description

```
salary_data['Salary'].describe()
```

 # assign independent values to x

 # frist para is raws

#second para is columns, all the columns -1

```
x = salary_data.iloc[:,:-1].values
```

#assign depenedet values to y

 # 2 cloumns mean 2 is eqal to index 1

```
y = salary_data.iloc[:,1].values
```

 # plot the data points

#check whther apply linear RM

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

#define the model

```
model = LinearRegrssion()
```

#apply fitted model

```
model.fit(x,y)
```

#show regression intercept value BETA 0 value

```
print(model.intercept_)
```

 # show the coefficent value, BETA 1 value

```
print(model.coef_)
```

#predict using 5 as x

#B0 + B1 * x

```
model.predict(np.array([[5]]))
```

#add the constatnt too the equation

```
x1 = sm.add_constant(x)
```

#ESTIMATE regression coefficient, summaries the model

#OLS regression result

```
model = sm.OLS(y,x1).fit()
model.summary()
```

#r Squred is vor the the accuracy

 # p value for do a hypothesis test