**ASSIGNMENT1 -Uber**

**import** pandas **as** pd

**import** numpy **as** np

**import** seaborn **as** sns

**import** matplotlib.pyplot **as** plt

df =read\_csv(“uber.csv”)

df**.**head() df**.**info()

df**.**columns

df **=** df**.**drop(['Unnamed: 0', 'key'], axis**=** 1)

df**.**head()   
df**.**shape

df**.**dtypes

df**.**info()

df**.**describe()

df**.**isnull()**.**sum()

df['dropoff\_latitude']**.**fillna(value**=**df['dropoff\_latitude']**.**mean(),inplace **=** **True**) df['dropoff\_longitude']**.**fillna(value**=**df['dropoff\_longitude’]**.**median(),inplace **=** **True** df**.**isnull()**.**sum() df**.**dtypes

df**=** df**.**assign(hour **=** df**.**pickup\_datetime**.**dt**.**hour, day**=** df**.**pickup\_datetime**.**dt**.**day, month **=** df**.**pickup\_datetime**.**dt**.**month, year **=** df**.**pickup\_datetime**.**dt**.**year, dayofweek **=** df**.**pickup\_datetime**.**dt**.**dayofweek) df**.**head()

df **=** df**.**drop('pickup\_datetime',axis**=**1) df**.**head() df**.**dtypes

df.plot(kinf=”box”, subplots = True , layout=(7,2), figsize=(15,20))

**def** treat\_outliers\_all(df1 , col\_list): **for** c **in** col\_list:

df1 **=** remove\_outlier(df , c) **return** df1

df **=** treat\_outliers\_all(df , df**.**iloc[: , 0::])

df**.**plot(kind **=** "box",subplots **=** **True**,layout **=** (7,2),figsize**=**(15,20))

|  |  |
| --- | --- |
| x **=** y **=** | df[['pickup\_longitude','pickup\_latitude','dropoff\_longitude','dropoff\_latitude', df['fare\_amount'] |

**from** sklearn.model\_selection **import** train\_test\_split

X\_train,X\_test,y\_train,y\_test **=** train\_test\_split(x,y,test\_size **=** 0.33)

|  |  |  |
| --- | --- | --- |
|  | **from** sklearn.linear\_model **import** LinearRegression regression **=** LinearRegression() regression**.**fit(X\_train,y\_train)  regression**.**coef\_ *#To find the linear coeeficient* regression**.**intercept\_ *#To find the linear intercept*  prediction **=** regression**.**predict(X\_test) *#To predict the target values* |  |

In [31]:

|  |  |  |
| --- | --- | --- |
|  | print(prediction) y\_test  **from** sklearn.metrics **import** r2\_score r2\_score(y\_test,prediction)  **from** sklearn.metrics **import** mean\_squared\_error  MSE **=** mean\_squared\_error(y\_test,prediction)  MSE  RMSE **=** np**.**sqrt(MSE)  RMSE |  |

|  |
| --- |
| **ASSIGNMENT 2-**Email Spam detection  **import** pandas **as** pd **import** numpy **as** np **import** seaborn **as** sns  **import** matplotlib.pyplot **as** plt  **%matplotlib** inline **import** warnings warnings**.**filterwarnings('ignore')  **from** sklearn.model\_selection **import** train\_test\_split **from** sklearn.svm **import** SVC **from** sklearn **import** metrics df**=**pd**.**read\_csv('emails.csv') df**.**head() df**.**columns df**.**isnull()**.**sum() df**.**dropna(inplace **=** **True**)  df**.**drop(['Email No.'],axis**=**1,inplace**=True**) X **=** df**.**drop(['Prediction'],axis **=** 1) y **=** df['Prediction'] **from** sklearn.preprocessing **import** scale  X **=** scale(X)  *# split into train and test*  X\_train, X\_test, y\_train, y\_test **=** train\_test\_split(X, y, test\_size **=** 0.3, random\_st  **from** sklearn.neighbors **import** KNeighborsClassifier knn **=** KNeighborsClassifier(n\_neighbors**=**7) knn**.**fit(X\_train, y\_train) y\_pred **=** knn**.**predict(X\_test) print("Prediction",y\_pred)  print("KNN accuracy = ",metrics**.**accuracy\_score(y\_test,y\_pred)) print("Confusion matrix",metrics**.**confusion\_matrix(y\_test,y\_pred))  model **=** SVC(C **=** 1)  *# fit*  model**.**fit(X\_train, y\_train)  *# predict*  y\_pred **=** model**.**predict(X\_test)  metrics**.**confusion\_matrix(y\_true**=**y\_test, y\_pred**=**y\_pred) print("SVM accuracy = ",metrics**.**accuracy\_score(y\_test,y\_pred)) |

**ASSIGNMENT 4 - Gradient Descent Algorithm**

cur\_x = 3 # The algorithm starts at x=3

rate = 0.01 # Learning rate

precision = 0.000001 #This tells us when to stop the algorithm

previous\_step\_size = 1 #

max\_iters = 10000 # maximum number of iterations

iters = 0 #iteration counter

df = lambda x: 2\*(x+5) #Gradient of our function

while previous\_step\_size > precision and iters < max\_iters:

prev\_x = cur\_x #Store current x value in prev\_x

cur\_x = cur\_x - rate \* df(prev\_x) #Grad descent

previous\_step\_size = abs(cur\_x - prev\_x) #Change in x

iters = iters+1 #iteration count

print("Iteration",iters,"\nX value is",cur\_x) #Print iterations

print("The local minimum occurs at", cur\_x)

**ASSIGNMENT 3 -Neural Network**

**import** pandas **as** pd

**import** numpy **as** np

**import** seaborn **as** sns

**import** matplotlib.pyplot **as** plt *#Importing the libraries*

df **=** pd**.**read\_csv("Churn\_Modelling.csv")

df**.**head() df**.**shape df**.**describe() df**.**isnull() df**.**isnull()**.**sum() df**.**info() df**.**dtypes df**.**columns

df **=** df**.**drop(['RowNumber', 'Surname', 'CustomerId'], axis**=** 1) *#Dropping the unnecess* df**.**head()

**def** visualization(x, y, xlabel):

plt**.**figure(figsize**=**(10,5))

plt**.**hist([x, y], color**=**['red', 'green'], label **=** ['exit', 'not\_exit'])

plt**.**xlabel(xlabel,fontsize**=**20)

plt**.**ylabel("No. of customers", fontsize**=**20)

plt**.**legend()

df\_churn\_exited **=** df[df['Exited']**==**1]['Tenure']

df\_churn\_not\_exited **=** df[df['Exited']**==**0]['Tenure']

visualization(df\_churn\_exited, df\_churn\_not\_exited, "Tenure") df\_churn\_exited2 **=** df[df['Exited']**==**1]['Age'] df\_churn\_not\_exited2 **=** df[df['Exited']**==**0]['Age'] visualization(df\_churn\_exited2, df\_churn\_not\_exited2, "Age")

X **=** df[['CreditScore','Gender','Age','Tenure','Balance','NumOfProducts','HasCrCard', states **=** pd**.**get\_dummies(df['Geography'],drop\_first **=** **True**) gender **=** pd**.**get\_dummies(df['Gender'],drop\_first **=** **True**) df **=** pd**.**concat([df,gender,states], axis **=** 1)

df**.**head()

X **=** df[['CreditScore','Age','Tenure','Balance','NumOfProducts','HasCrCard','IsActive y **=** df['Exited'] **from** sklearn.model\_selection **import** train\_test\_split

X\_train,X\_test,y\_train,y\_test **=** train\_test\_split(X,y,test\_size **=** 0.30)

**from** sklearn.preprocessing **import** StandardScaler sc **=** StandardScaler()

X\_train **=** sc**.**fit\_transform(X\_train)

X\_test **=** sc**.**transform(X\_test)

X\_train

X\_test

**import** keras *#Can use Tenserflow as well but won't be able to understand the errors i*

**from** keras.models **import** Sequential *#To create sequential neural network*

**from** keras.layers **import** Dense *#To create hidden layers*

classifier **=** Sequential()

classifier**.**add(Dense(activation **=** "relu",input\_dim **=** 11,units **=** 6,kernel\_initializer classifier**.**add(Dense(activation **=** "relu",units **=** 6,kernel\_initializer **=** "uniform")) classifier**.**add(Dense(activation **=** "sigmoid",units **=** 1,kernel\_initializer **=** "uniform" classifier**.**compile(optimizer**=**"adam",loss **=** 'binary\_crossentropy',metrics **=** ['accurac classifier**.**summary()

classifier**.**fit(X\_train,y\_train,batch\_size**=**10,epochs**=**50) *#Fitting the ANN to training* y\_pred **=**classifier**.**predict(X\_test)

y\_pred **=** (y\_pred **>** 0.5) *#Predicting the result*

**from** sklearn.metrics **import** confusion\_matrix,accuracy\_score,classification\_report cm **=** confusion\_matrix(y\_test,y\_pred) cm

accuracy **=** accuracy\_score(y\_test,y\_pred) accuracy

plt**.**figure(figsize **=** (10,7)) sns**.**heatmap(cm,annot **=** **True**) plt**.**xlabel('Predicted') plt**.**ylabel('Truth')

print(classification\_report(y\_test,y\_pred))

**DAA- Knapsack**

class Item:

def \_\_init\_\_(self,value,weight):

self.value = value

self.weight = weight

def knapsack(W,arr):

arr.sort(key = lambda x:(x.value/x.weight), reverse = True)

finalvalue = 0.0

for item in arr:

if item.weight <= W:

W -= item.weight

finalvalue+=item.value

else:

finalvalue+=item.value\*W/item.weight

return finalvalue

W = 50

arr = [Item(60,10), Item(100,20),Item(120,30)]

maaxval = knapsack(W,arr)

print(maaxval)

**FIBONACCI**

nterms = int(input("How many terms? ")) # first two terms

n1, n2 = 0, 1

count = 0 # check if the number of terms is valid   
if nterms <= 0:

print("Please enter a positive integer") # if there is only one term,

return n1

elif nterms == 1:

print("Fibonacci sequence upto",nterms,":")

print(n1) # generate fibonacci sequence

else:

print("Fibonacci sequence:")

while count < nterms:

print(n1)

nth = n1 + n2 # update values

n1 = n2

n2 = nth

count += 1

**KNAPSACK ) 0-1**

def knapSack(W, wt, val, n):

dp = [0 for i in range(W+1)] # Making the dp array

for i in range(1, n+1): # taking first i elements

for w in range(W, 0, -1): # starting from back,so that we also have data of

# previous computation when taking i-1 items

if wt[i-1] <= w: # finding the maximum value

dp[w] = max(dp[w], dp[w-wt[i-1]]+val[i-1])

return dp[W] # returning the maximum value of knapsack

# Driver code

val = [60, 100, 120]

wt = [10, 20, 30]

W = 50

n = len(val)

print(knapSack(W, wt, val, n))