Bct

1 bank

// SPDX-License-Identifier: UNLICENSED

pragma solidity >= 0.7.0;

// Write a smart contract on a test network, for Bank account of a customer for

// following operations: Deposit money | Withdraw Money | Show balance

contract Bank{

mapping(address => uint) public user\_account;

mapping(address => bool) public user\_exist;

function create\_account() public payable returns(string memory){

require(user\_exist[msg.sender] == false, "Account Already created!");

user\_account[msg.sender] = msg.value;

user\_exist[msg.sender] = true;

return "Account created";

}

function deposit(uint amount) public payable returns(string memory){

require(user\_exist[msg.sender] == true, "Account not created!");

require(amount > 0, "Amount should be greater than 0");

user\_account[msg.sender] += amount;

return "Amount deposisted sucessfully";

}

function withdraw(uint amount) public payable returns(string memory){

require(user\_exist[msg.sender] == true, "Account not created!");

require(amount > 0, "Amount should be greater than 0");

require(user\_account[msg.sender] >= amount, "Amount is greater than money deposisted");

user\_account[msg.sender] -= amount;

return "Amount withdrawn sucessfully";

}

function account\_balance() public view returns(uint){

return user\_account[msg.sender];

}

function account\_exists() public view returns(bool){

return user\_exist[msg.sender];

}

}

2)student

// SPDX-License-Identifier: MIT

pragma solidity >= 0.7.0;

contract StudentRegister{

address public owner;

mapping(address=>student)students;

constructor() public{

owner = msg.sender;

}

modifier onlyOwner{

require(msg.sender==owner);

\_;

}

struct student{

address studentid;

string name;

string course;

uint256 mark1;

uint256 mark2;

uint256 mark3;

uint256 totalMarks;

uint256 percentage;

bool isExist;

}

function register(address studentid, string memory name, string memory course,uint256 mark1, uint256 mark2, uint256 mark3) public onlyOwner{

require(students[studentid].isExist==false,"ha..ha..Fraud Not Possible.");

uint256 totalMarks;

uint256 percentage;

totalMarks = (mark1+mark2+mark3);

percentage=(totalMarks/3);

students[studentid] = student(studentid,name, course,mark1,mark2,mark3,totalMarks,percentage,true);

}

function getStudentDetails(address studentid) public view returns(address, string memory, string memory, uint256, uint256){

return(students[studentid].studentid,students[studentid].name,students[studentid].course,students[studentid].totalMarks,students[studentid].percentage);

Machine Learning

1)uber

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

df = pd.read\_csv("uber.csv")

df.head()

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

df.info()

df.shape

df.dtypes

df.describe()

df.isnull().sum()

df['fare\_amount'].fillna(value=df['fare\_amount'].mean(), inplace=True)

df['pickup\_longitude'].fillna(value=df['pickup\_longitude'].mean(), inplace=True)

df['pickup\_latitude'].fillna(value=df['pickup\_latitude'].mean(), inplace=True)

df['dropoff\_longitude'].fillna(value=df['dropoff\_longitude'].mean(), inplace=True)

df['dropoff\_latitude'].fillna(value=df['dropoff\_latitude'].mean(), inplace=True)

df['passenger\_count'].fillna(value=df['passenger\_count'].mean(), inplace=True)

df.isnull().sum()

df.pickup\_datetime = pd.to\_datetime(df.pickup\_datetime, errors='coerce')

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(kind="box", subplots=True, layout=(7,2), figsize=(15,20))

def remove\_outlier(df1, col):

Q1 = df1[col].quantile(0.25)

Q3 = df1[col].quantile(0.75)

IQR = Q3 - Q1

lower\_whisker = Q1 - 1.5\*IQR

upper\_whisker = Q3 + 1.5\*IQR

df1[col] = np.clip(df1[col], lower\_whisker, upper\_whisker)

return df1

**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))

!pip install haversine

**import** haversine **as** hs  
travel\_dist = []  
**for** pos **in** range(len(df['pickup\_longitude'])):  
  long1, lati1, long2, lati2 = df['pickup\_longitude'][pos], df['pickup\_latitude'][pos], df['dropoff\_longitude'][pos], df['dropoff\_latitude'][pos]  
  loc1 = (lati1, long1)  
  loc2 = (lati2, long2)  
  c = hs.haversine(loc1, loc2)  
  travel\_dist.append(c)  
  
  
df['dist\_travel\_km'] = travel\_dist  
df.head()

df = df.loc[(df.dist\_travel\_km >= 1) | (df.dist\_travel\_km <= 130)]  
print("Remaining Observations in the dataset: ", df.shape)

df.isnull().sum()

sns.heatmap(df.isnull())

corr = df.corr()

corr

fig, axis = plt.subplots(figsize = (10,6))  
sns.heatmap(df.corr(), annot = True)

x = df[['pickup\_longitude', 'pickup\_longitude', 'dropoff\_longitude', 'dropoff\_latitude']]  
y = 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)

*#Linear Regression*

**from** sklearn.linear\_model **import** LinearRegression  
regression = LinearRegression()

regression.fit(x\_train, y\_train)

regression.intercept\_

regression.coef\_

prediction = regression.predict(x\_test)

print(prediction)

y\_test

*#Metrics Evaluation*

**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

*#Random Forest Regression*

**from** sklearn.ensemble **import** RandomForestRegressor

rf = RandomForestRegressor(n\_estimators=100)

rf.fit(x\_train, y\_train)

y\_pred = rf.predict(x\_test)

y\_pred

*#Metrics Evalution for Random Forest*

R2\_Random = r2\_score(y\_test, y\_pred)

R2\_Random

MSE\_Random = mean\_squared\_error(y\_test, y\_pred)

MSE\_Random

RMSE\_Random = np.sqrt(MSE\_Random)

RMSE\_Random

2) email

**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.isnull().sum()

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)  
*#Splitting dataset*  
x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size = 0.3, random\_state = 0)

*#KNN Classifier*

**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))

*#SVM Classifier*

model = SVC(C = 1)

model.fit(x\_train, y\_train)

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))

3) churn modelling

**import** pandas **as** pd  
**import** numpy **as** np  
**import** matplotlib.pyplot **as** plt  
**import** seaborn **as** sns  
**from** sklearn.preprocessing **import** StandardScaler  
**import** io

**from** google.colab **import** files  
uploaded=files.upload()

df=pd.read\_csv(io.StringIO(uploaded['Churn\_Modelling.csv'].decode('utf-8')))  
df.head()

df=df.drop(['RowNumber','CustomerId','Surname'],axis=1)  
df.head()

df.isna().any()  
df.isna().sum()

print(df.shape)  
df.info()

df.describe()

*## Scale the data*  
scaler=StandardScaler()  
*## Extract only the Numerical Columns to perform Bivariate Analysis*  
subset=df.drop(['Geography','Gender','HasCrCard','IsActiveMember'],axis=1)  
scaled=scaler.fit\_transform(subset)  
scaled\_df=pd.DataFrame(scaled,columns=subset.columns)  
sns.pairplot(scaled\_df,diag\_kind='kde')

sns.heatmap(scaled\_df.corr(),annot=True,cmap='rainbow')

*## Categorical Features vs Target Variable*  
sns.countplot(x='Geography',data=df,hue='Exited')  
plt.show()  
sns.countplot(x='Gender',data=df,hue='Exited')  
plt.show()  
sns.countplot(x='HasCrCard',data=df,hue='Exited')  
plt.show()  
sns.countplot(x='IsActiveMember',data=df,hue='Exited')  
plt.show()

subset=subset.drop('Exited',axis=1)  
**for** i **in** subset.columns:    
  sns.boxplot(df['Exited'],df[i],hue=df['Gender'])  
  plt.show()

X=df.drop('Exited',axis=1)  
y=df.pop('Exited')

**from** sklearn.model\_selection **import** train\_test\_split  
X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,y,test\_size=0.10,random\_state=5)  
X\_train,X\_val,y\_train,y\_val=train\_test\_split(X\_train,y\_train,test\_size=0.10,random\_state=5)  
print("X\_train size is {}".format(X\_train.shape[0]))  
print("X\_val size is {}".format(X\_val.shape[0]))  
print("X\_test size is {}".format(X\_test.shape[0]))

*## Standardising the train, Val and Test data*  
**from** sklearn.preprocessing **import** StandardScaler  
scaler=StandardScaler()  
num\_cols=['CreditScore','Age','Tenure','Balance','NumOfProducts','EstimatedSalary']  
num\_subset=scaler.fit\_transform(X\_train[num\_cols])  
X\_train\_num\_df=pd.DataFrame(num\_subset,columns=num\_cols)  
X\_train\_num\_df['Geography']=list(X\_train['Geography'])  
X\_train\_num\_df['Gender']=list(X\_train['Gender'])  
X\_train\_num\_df['HasCrCard']=list(X\_train['HasCrCard'])  
X\_train\_num\_df['IsActiveMember']=list(X\_train['IsActiveMember'])  
X\_train\_num\_df.head()  
*## Standardise the Validation data*  
num\_subset=scaler.fit\_transform(X\_val[num\_cols])  
X\_val\_num\_df=pd.DataFrame(num\_subset,columns=num\_cols)  
X\_val\_num\_df['Geography']=list(X\_val['Geography'])  
X\_val\_num\_df['Gender']=list(X\_val['Gender'])  
X\_val\_num\_df['HasCrCard']=list(X\_val['HasCrCard'])  
X\_val\_num\_df['IsActiveMember']=list(X\_val['IsActiveMember'])  
*## Standardise the Test data*  
num\_subset=scaler.fit\_transform(X\_test[num\_cols])  
X\_test\_num\_df=pd.DataFrame(num\_subset,columns=num\_cols)  
X\_test\_num\_df['Geography']=list(X\_test['Geography'])  
X\_test\_num\_df['Gender']=list(X\_test['Gender'])  
X\_test\_num\_df['HasCrCard']=list(X\_test['HasCrCard'])  
X\_test\_num\_df['IsActiveMember']=list(X\_test['IsActiveMember'])

*## Convert the categorical features to numerical*  
X\_train\_num\_df=pd.get\_dummies(X\_train\_num\_df,columns=['Geography','Gender'])  
X\_test\_num\_df=pd.get\_dummies(X\_test\_num\_df,columns=['Geography','Gender'])  
X\_val\_num\_df=pd.get\_dummies(X\_val\_num\_df,columns=['Geography','Gender'])  
X\_train\_num\_df.head()

**from** tensorflow.keras **import** Sequential  
**from** tensorflow.keras.layers **import** Dense  
  
model=Sequential()  
model.add(Dense(7,activation='relu'))  
model.add(Dense(10,activation='relu'))  
model.add(Dense(1,activation='sigmoid'))

**import** tensorflow **as** tf  
optimizer=tf.keras.optimizers.Adam(0.01)  
model.compile(loss='binary\_crossentropy',optimizer=optimizer,metrics=['accuracy'])

model.fit(X\_train\_num\_df,y\_train,epochs=100,batch\_size=10,verbose=1)

y\_pred\_val=model.predict(X\_val\_num\_df)  
y\_pred\_val[y\_pred\_val>0.5]=1  
y\_pred\_val[y\_pred\_val <0.5]=0

y\_pred\_val=y\_pred\_val.tolist()  
X\_compare\_val=X\_val.copy()  
X\_compare\_val['y\_actual']=y\_val  
X\_compare\_val['y\_pred']=y\_pred\_val  
X\_compare\_val.head(10)

*#Confusion Matrix*

**from** sklearn.metrics **import** confusion\_matrix  
cm\_val=confusion\_matrix(y\_val,y\_pred\_val)  
cm\_val

array([[687, 29],  
 [ 97, 87]])

Accuracy=782/900  
print("Accuracy of the Model on the Validation Data set is 86.89%")

loss1,accuracy1=model.evaluate(X\_train\_num\_df,y\_train,verbose=False)  
loss2,accuracy2=model.evaluate(X\_val\_num\_df,y\_val,verbose=False)  
print("Train Loss {}".format(loss1))  
print("Train Accuracy {}".format(accuracy1))  
print("Val Loss {}".format(loss2))  
print("Val Accuracy {}".format(accuracy2))

**from** sklearn **import** metrics  
y\_pred\_test=model.predict(X\_test\_num\_df)  
y\_pred\_test[y\_pred\_test>0.5]=1  
y\_pred\_test[y\_pred\_test <0.5]=0  
cm\_test=metrics.confusion\_matrix(y\_test,y\_pred\_test)  
cm\_test  
print("Test Confusion Matrix")

cm\_test

loss3,accuracy3=model.evaluate(X\_test\_num\_df,y\_test,verbose=False)  
print("Test Accuracy is {}".format(accuracy3))  
print("Test loss is {}".format(loss3))

4)diabetes

**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('diabetes.csv')

df.columns

df.isnull().sum()

x = df.drop('Outcome', axis = 1)  
y = df['Outcome']

**from** sklearn.preprocessing **import** scale  
x = scale(x)  
x\_train, x\_test, y\_train, y\_test = train\_test\_split(x,y,test\_size=0.3,random\_state=None)

**from** sklearn.neighbors **import** KNeighborsClassifier  
knn = KNeighborsClassifier(n\_neighbors = 7)  
knn.fit(x\_train, y\_train)  
y\_pred = knn.predict(x\_test)

print("Confusion Matrix: ")  
cs = metrics.confusion\_matrix(y\_test, y\_pred)  
print(cs)

print("Accuracy ", metrics.accuracy\_score(y\_test, y\_pred))

total\_misclassified = cs[0,1] + cs[1,0]  
print(total\_misclassified)  
total\_examples = cs[0,0] + cs[0,1] + cs[1,0] + cs[1,1]  
print(total\_examples)  
print("Error rate", total\_misclassified/total\_examples)  
print("Error rate", 1-metrics.accuracy\_score(y\_test, y\_pred))

print("Precision score", metrics.precision\_score(y\_test, y\_pred))

print("Precision score", metrics.recall\_score(y\_test, y\_pred))

print("Classification report", metrics.classification\_report(y\_test, y\_pred))

DAA

1)0-1 knapsack

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("Max value is: ", knapSack(W, wt, val, n))

2)fractional knapsack

class Item:

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

self.value = value

self.weight = weight

def fractionalKnapsack(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

break

return finalValue

W = 50

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

max\_val = fractionalKnapsack(W, arr)

print("Max Value of the Capacity is: ", max\_val)

3) 8-queen problem

global N  
N = 8  
  
def printSolution(board):  
 for i in range(N):  
 for j in range(N):  
 print(board[i][j], end=" ")  
 print()  
  
def isSafe(board, row, col):  
 for i in range(col):  
 if board[row][i] == 1:  
 return False

for i, j in zip(range(row, -1, -1),  
 range(col, -1, -1)):  
 if board[i][j] == 1:  
 return False

for i, j in zip(range(row, N, 1),  
 range(col, -1, -1)):  
 if board[i][j] == 1:  
 return False  
  
 return True  
  
  
def solveNQUtil(board, col):  
 if col >= N:  
 return True  
 for i in range(N):  
  
 if isSafe(board, i, col):  
   
 board[i][col] = 1

if solveNQUtil(board, col + 1) == True:  
 return True  
 board[i][col] = 0  
  
 return False  
  
def solveNQ():  
 board = [[0, 0, 0, 0, 0, 0, 0, 0],  
 [0, 0, 0, 0, 0 ,0, 0, 0],  
 [0, 0, 0, 0, 0, 0, 0, 0],  
 [0, 0, 0, 0, 0, 0, 0, 0],  
 [0, 0, 0, 0, 0, 0, 0, 0],  
 [0, 0, 0, 0, 0, 0, 0, 0],  
 [0, 0, 0, 0, 0, 0, 0, 0],  
 [0, 0, 0, 0, 0, 0, 0, 0]]  
  
 if solveNQUtil(board, 0) == False:  
 print("Solution does not exist")  
 return False  
  
 printSolution(board)  
 return True  
  
  
# Driver Code  
solveNQ()

4)Fibonacci(Recirsive)

def fibonacci(n):  
 if n <= 1:  
 return n  
 else:  
 return fibonacci(n-1) + fibonacci(n-2)  
  
  
nterms = int(input("Enter number of terms: "))  
if(nterms <= 0):  
 print("Enter a positive integer: ")  
else:  
 print("Fibonnaci Series: ")  
 for i in range(nterms):  
 print(fibonacci(i))

5)Fibonacci Non-recursive)

n = int(input("Enter a number: "))  
f1 = 0  
f2 = 1  
  
if n <= 1:  
 print(n)  
else:  
 print(f1)  
 print(f2)  
 for i in range(2, n):  
 f3 = f1 + f2  
 print(f3)  
 f1 = f2  
 f2 = f3