|  |  |
| --- | --- |
| ML.Uber1  import pandas as pd  df = pd.read\_csv('uber.csv')  df.head();;  *\*pre-process the dataset*  df.info();;  df.describe();;  df.isnull().sum();;  print(df.isnull().values.sum());;  df = df.drop(['Unnamed: 0','key'], axis = 1);;  df.head();;  df = df[df.fare\_amount > 0];;  df.shape;;  df.describe();;  df = df[(df.passenger\_count <= 6) & (df.passenger\_count > 0)];;  df.head();;  df = df[(df.pickup\_longitude.between(-180,180,inclusive = "both")) & (df.pickup\_latitude.between(-90,90,inclusive = "both")) & (df.dropoff\_longitude.between(-180,180,inclusive = "both")) & (df.dropoff\_latitude.between(-90,90,inclusive = "both"))];;  df.head(10);;  df.info();;  df['pickup\_datetime'] = pd.to\_datetime(df['pickup\_datetime']);;  df.info();;  df.head(10);;  import calendar;;  df['year'] = df.pickup\_datetime.dt.year  df['month'] = df.pickup\_datetime.dt.month  df['weekday'] = df.pickup\_datetime.dt.weekday  df['hour'] = df.pickup\_datetime.dt.hour;;  df = df.drop(['pickup\_datetime'], axis=1);;  df.head(10);;  df.describe();;  \*Remove outliers  df = df.reset\_index();;  df.head();;  df = df.drop(['index'], axis = 1);;  df.head(10);;  import numpy as np;;  Q1 = np.percentile(df['fare\_amount'], 25 )  Q3 = np.percentile(df['fare\_amount'], 75 )  IQR = Q3 - Q1  print(f'IQR = {IQR}');;  upper = np.where(df['fare\_amount'] >= (Q3 + 1.5\*IQR))  lower = np.where(df['fare\_amount'] <= (Q1 - 1.5\*IQR))  print(f'upper = {upper}')  print(f'lower = {lower}');;  df = df.drop(upper[0])  df = df.drop(lower[0]);;  df.describe();;  \*Find Correlation  import seaborn as sns  import matplotlib.pyplot as plt  uber\_corr = df.corr() #use heatmap  plt.figure(figsize=(10,7))  sns.heatmap(uber\_corr,annot=True)  plt.show();;  *\*implement linear reg. & random fore reg.*  from sklearn.model\_selection import train\_test\_split;;  X = df.drop('fare\_amount', axis = 1)  y = df["fare\_amount"];;  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.2);;  from sklearn.linear\_model import LinearRegression  lrmodel = LinearRegression()  lrmodel.fit(X\_train, y\_train)  lr\_pred = lrmodel.predict(X\_test)  lr\_pred;;  from sklearn.ensemble import RandomForestRegressor  rfmodel = RandomForestRegressor(n\_estimators=100, random\_state=101)  rfmodel.fit(X\_train, y\_train)  rf\_pred = rfmodel.predict(X\_test)  rf\_pred;;  *\*evaluate model*  from sklearn.metrics import mean\_squared\_error;;  lrmodel\_rmse = np.sqrt(mean\_squared\_error(y\_test, lr\_pred))  rfmodel\_rmse = np.sqrt(mean\_squared\_error(y\_test, rf\_pred))  print(f'Linear Regression RMSE = {lrmodel\_rmse}')  print(f'Random Forest RMSE = {rfmodel\_rmse}');;  from sklearn.metrics import r2\_score  lrmodel\_r2 = r2\_score(y\_test, lr\_pred)  rfmodel\_r2 = r2\_score(y\_test, rf\_pred)  print(f'Linear Regression R2 = {lrmodel\_r2}')  print(f'Random Forest R2 = {rfmodel\_r2}');;  ML5KMeansClustering  import pandas as pd  import numpy as np  import seaborn as sns  import matplotlib.pyplot as plt;;  from sklearn.cluster import KMeans, k\_means  from sklearn.decomposition import PCA;;  df = pd.read\_csv('/content/sales\_data\_sample.csv');;  df.head();;  df.shape;;  df.describe();;  df.info();;  df.isnull().sum();;  df.dtypes;;  df\_drop = ['ADDRESSLINE1', 'ADDRESSLINE2', 'STATUS','POSTALCODE', 'CITY', 'TERRITORY', 'PHONE', 'STATE', 'CONTACTFIRSTNAME', 'CONTACTLASTNAME', 'CUSTOMERNAME', 'ORDERNUMBER']  df = df.drop(df\_drop, axis=1);;  df.isnull().sum();;  df.dtypes;;  \**checking the categorical columns*  df['COUNTRY'].unique();;  df['PRODUCTLINE'].unique();;  df['DEALSIZE'].unique();;  productline = pd.get\_dummies(df['PRODUCTLINE'])  Dealsize = pd.get\_dummies(df['DEALSIZE']);;  df = pd.concat([df,productline,Dealsize], axis = 1);;  df\_drop = ['COUNTRY','PRODUCTLINE','DEALSIZE'] #Dropping Country too as there are alot of countries.  df = df.drop(df\_drop, axis=1);;  df['PRODUCTCODE'] = pd.Categorical(df['PRODUCTCODE']).codes;;  df.drop('ORDERDATE', axis=1, inplace=True);;  df.dtypes;;  distortions = []  K = range(1,10)  for k in K:  kmeanModel = KMeans(n\_clusters=k)  kmeanModel.fit(df)  distortions.append(kmeanModel.inertia\_);;  plt.figure(figsize=(16,8))  plt.plot(K, distortions, 'bx-')  plt.xlabel('k')  plt.ylabel('Distortion')  plt.title('The Elbow Method showing the optimal k')  plt.show();;  \**numb. Of k increases inertia decreases*  X\_train = df.values;;  X\_train.shape;;  model = KMeans(n\_clusters=3,random\_state=2)  model = model.fit(X\_train)  predictions = model.predict(X\_train);;  unique,counts = np.unique(predictions,return\_counts=True);;  counts = counts.reshape(1,3);;  counts\_df = pd.DataFrame(counts,columns=['Cluster1','Cluster2','Cluster3']);;  counts\_df.head();;  pca = PCA(n\_components=2);;  reduced\_X = pd.DataFrame(pca.fit\_transform(X\_train),columns=['PCA1','PCA2']);;  reduced\_X.head();;  plt.figure(figsize=(14,10))  plt.scatter(reduced\_X['PCA1'],reduced\_X['PCA2']);;  model.cluster\_centers\_;;  reduced\_centers = pca.transform(model.cluster\_centers\_);;  reduced\_centers;;  plt.figure(figsize=(14,10))  plt.scatter(reduced\_X['PCA1'],reduced\_X['PCA2'])  plt.scatter(reduced\_centers[:,0],reduced\_centers[:,1],color='black',marker='x',s=300);;  reduced\_X['Clusters'] = predictions;;  reduced\_X.head();;  plt.figure(figsize=(14,10))  plt.scatter(reduced\_X[reduced\_X['Clusters'] == 0].loc[:,'PCA1'],reduced\_X[reduced\_X['Clusters'] == 0].loc[:,'PCA2'],color='slateblue') .  plt.scatter(reduced\_X[reduced\_X['Clusters'] == 1].loc[:,'PCA1'],reduced\_X[reduced\_X['Clusters'] == 1].loc[:,'PCA2'],color='springgreen') .  plt.scatter(reduced\_X[reduced\_X['Clusters'] == 2].loc[:,'PCA1'],reduced\_X[reduced\_X['Clusters'] == 2].loc[:,'PCA2'],color='indigo') .  plt.scatter(reduced\_centers[:,0],reduced\_centers[:,1],color='black',marker='x',s=300);; | ML3GradientDescentAlgorithm  import pandas as pd  cur\_x = 2  rate = 0.01  precision = 0.0000001  previous\_step\_size = 1  max\_iters = 10000  iters = 0  df = lambda x: (2 \* (x + 3));;  while previous\_step\_size > precision and iters < max\_iters:  prev\_x = cur\_x  cur\_x = cur\_x - rate \* df(prev\_x)  previous\_step\_size = abs(cur\_x - prev\_x)  iters = iters + 1  print(f'Iteration {iters} \n value is {cur\_x} ')  print(f'The local minima occurs at {cur\_x}')  ML2bankcustomer,build a neuralnetwork  import pandas as pd  import numpy as np;;  ds = pd.read\_csv('Churn\_Modelling.csv')  ds.head(10);;  ds.columns;;  ds.shape;;  ds['Geography'].value\_counts(normalize=True);;  ds = ds.drop(['RowNumber', 'CustomerId', 'Surname'], axis=1);;  ds.info();;  ds.describe();;  X = ds.iloc[:,0:10].values  y = ds.iloc[:,10].values;;  X;;  from sklearn.preprocessing import LabelEncoder  print(X[:8,1], '... will now become: ')  label\_X\_country\_encoder = LabelEncoder()  X[:,1] = label\_X\_country\_encoder.fit\_transform(X[:,1])  print(X[:8,1]);;  print(X[:6,2], '... will now become: ')  label\_X\_gender\_encoder = LabelEncoder()  X[:,2] = label\_X\_gender\_encoder.fit\_transform(X[:,2])  print(X[:6,2]);;  X.shape;;  from sklearn.compose import ColumnTransformer  from sklearn.preprocessing import OneHotEncoder  transform = ColumnTransformer([("countries", OneHotEncoder(), [1])], remainder="passthrough")  X = transform.fit\_transform(X);;  X;;  X = X[:,1:];;  X.shape;;  from sklearn.model\_selection import train\_test\_split  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y, test\_size = 0.2, random\_state = 0)  numeric\_cols = ['CreditScore','Age','Tenure','Balance','NumOfProducts','EstimatedSalary'];;  X\_train[:,np.array([2,4,5,6,7,10])];;  from sklearn.preprocessing import StandardScaler  sc=StandardScaler()  X\_train[:,np.array([2,4,5,6,7,10])] = sc.fit\_transform(X\_train[:,np.array([2,4,5,6,7,10])])  X\_test[:,np.array([2,4,5,6,7,10])] = sc.transform(X\_test[:,np.array([2,4,5,6,7,10])]);;  X\_train[0];;  from sklearn.preprocessing import StandardScaler  sc=StandardScaler()  X\_train = sc.fit\_transform(X\_train)  X\_test = sc.transform(X\_test);;  X\_train;;  X\_train.shape;;  from tensorflow.keras.models import Sequential  # Initializing the ANN  classifier = Sequential();;  from tensorflow.keras.layers import Dense  classifier.add(Dense(activation = 'relu', input\_dim = 11, units=16, kernel\_initializer='uniform'));;  classifier.add(Dense(8, activation='relu', kernel\_initializer='uniform'));;  classifier.add(Dense(1, activation = 'sigmoid', kernel\_initializer='uniform'));;  192-11\*16;;  classifier.summary();;  classifier.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy']);;  classifier.summary();;  classifier.fit(X\_train, y\_train,  validation\_data=(X\_test,y\_test),  epochs=20);;  y\_pred = classifier.predict(X\_test)  print(y\_pred);;  y\_pred = (y\_pred > 0.5)  print(y\_pred);;  from sklearn.metrics import confusion\_matrix,classification\_report  cm1 = confusion\_matrix(y\_test, y\_pred)  print(cm1);;  print(classification\_report(y\_test, y\_pred));;  accuracy\_model1 = ((cm1[0][0]+cm1[1][1])\*100)/(cm1[0][0]+cm1[1][1]+cm1[0][1]+cm1[1][0])  print (accuracy\_model1, '% of testing data was classified correctly');;  classifier.summary();;  classifier.compile(optimizer='adam', loss = 'binary\_crossentropy', metrics=['accuracy']);;  classifier.fit(X\_train, y\_train,  validation\_data=(X\_test,y\_test),  epochs=20,  batch\_size=32);;  y\_pred = classifier.predict(X\_test)  print(y\_pred);;  y\_pred = (y\_pred > 0.5)  print(y\_pred);;  cm2 = confusion\_matrix(y\_test, y\_pred)  print(cm2);;  cm2 = classification\_report(y\_test, y\_pred)  print(cm2);;  ML4KNN  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('/content/diabetes.csv');;  df.columns;;  df.isnull().sum();;  X = df.drop('Outcome',axis = 1)  y = df['Outcome'];;  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\_state = 42);;  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("Acccuracy ",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("Recall score ",metrics.recall\_score(y\_test,y\_pred));;  print("Classification report ",metrics.classification\_report(y\_test,y\_pred));; |