```
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}')
```

```
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();;
```

```
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);;
```

```
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));;
```