ML LAB EXERCISES

1. **A) Find the line with the least error among these lines and store it as the line of best fit.**

B) Simple linear regression from scratch without using sklearn libraries and print the RMSE and mean absolute error values.

a)

```
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
np.random.seed(42)
data = np.random.rand(1500, 2)
X = data[:, 0]
y = 3 * X + 2 + 0.1 * np.random.randn(1500)
num_samples = 1000
i = np.random.choice(len(X), num_samples, replace=False)
x_s = X[i]
y_s = y[i]
for i in range(1000):
i = np.random.choice(num_samples, 2, replace=False)
X_fit = x_s[i]
y_fit = y_s[i]
mdl = LinearRegression()
mdl.fit(X_fit.reshape(-1, 1), y_fit)
y_pred = mdl.predict(x_s.reshape(-1, 1))
bf_i = np.random.choice(num_samples, 2, replace=False)
X_bf_ = x_s[bf_i]
y_bf_ = y_s[bf_i]
bf_mdl = LinearRegression()
bf_mdl.fit(X_bf_.reshape(-1, 1), y_bf_)
X_test = X[~np.isin(np.arange(len(X)), i)]
y_test_pred = bf_mdl.predict(X_test.reshape(-1, 1))
plt.scatter(x_s, y_s, label="Sample Data")
plt.plot(X_test, y_test_pred, color='red', label="Best Fit Line")
plt.legend()
plt.show()
```

b)

```
import numpy as np
```

```
import pandas as pd
from math import sqrt
file = "data1.csv"
data = pd.read_csv(file)
X = data['x'].values
y = data['y'].values
n = len(X)
num = np.sum(X * y - np.mean(y) * X)
den = np.sum(X**2 - np.mean(X) * X)
m = num / den
c = (np.sum(y) - m * np.sum(X)) / n
y_pred = m * X + c
rmse = sqrt(np.mean((y - y_pred)**2))
mae = np.mean(np.abs(y - y_pred))
print("Equation from formula:")
print(f"intercept c: {c:.4f}")
print(f"slope m: {m:.4f}")
print(f"RMSE: {rmse:.4f}")
print(f"MAE: {mae:.4f}")
print("\n")from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error
model = LinearRegression()
model.fit(X.reshape(-1, 1), y)
y_pred_sklearn = model.predict(X.reshape(-1, 1))
rmse_sklearn = sqrt(mean_squared_error(y, y_pred_sklearn))
mae_sklearn = mean_absolute_error(y, y_pred_sklearn)
print("Equation from scikit-learn:")
print(f"intercept c: {model.intercept_:.4f}")
print(f"model m: {model.coef_[0]:.4f}")
print(f"RMSE: {rmse_sklearn:.4f}")
print(f"MAE: {mae_sklearn:.4f}")
```

2. **Use the house_pred.csv file to build a multiple linear regression model. sklearn shall be used to fit the model.Perform necessary preprocessing and check for outliers and multi-collinearity. Apply the same set of preprocessing to the test.csv and use the data to predict the house price. The evaluation criteria will be Root Mean Squared Error**

```
# import statements

import pandas as pd
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_squared_error
import math
train = pd.read_csv('house_pred.csv')
test = pd.read_csv("test.csv")
```

```
train.head()
test.head()
score = train.isnull().sum()
for i in train.columns:
    if (score[i] > 1000):
        train = train.drop(i, axis=1)
        test = test.drop(i, axis=1)

for i in train.columns:
    count = (train[i] == 0).sum()
    if (count > 500):
        train = train.drop(i, axis=1)
        test = test.drop(i, axis=1)
string_columns = train.select_dtypes(include=['object']).columns
string_columns
for i in string_columns:
    state = pd.get_dummies(train[i], drop_first=True)
    state = state.astype(int)
    train = pd.concat([train, state], axis=1)
    state = pd.get_dummies(test[i], drop_first=True)
    state = state.astype(int)
    test = pd.concat([test, state], axis=1)
train = train.drop(string_columns, axis=1)
test = test.drop(string_columns, axis=1)
train = train.loc[:, ~train.columns.duplicated()]
test = test.loc[:, ~test.columns.duplicated()]

train = train.fillna(0)
test = test.fillna(0)
a = train['SalePrice']
train = train.drop('SalePrice',axis=1)
train = pd.concat([train,a], axis=1)
x = train.iloc[:,:-1].values
y = train.iloc[:,-1].values
x = pd.DataFrame(x)
col = list(train.columns)
col.remove('SalePrice')
x.columns = col
x['intercept']=1
while True:
    vif = pd.DataFrame()
    vif['variable'] = x.columns
    vif['vif'] = [variance_inflation_factor(x.values,i) for i in range(x.shape[1])]
    max_row = vif.loc[vif['vif'].idxmax()]
    if max_row[1] <= 3:
        break
    x = x.drop(max_row[0], axis=1)
vif
x_train,x_test,y_train,y_test = train_test_split(x,y, test_size=0.3, random_state=0)
reg = LinearRegression()
reg.fit(x_train,y_train)
```

```
reg.intercept_, reg.coef_
y_pred = reg.predict(x_test)
r2_score(y_test,y_pred)
math.sqrt(mean_squared_error(y_test,y_pred))
```

3. **Use the dataset, perform necessary pre-processing and build a logistic regression model. divide the train data itself into 70-30 ratio and print the performance metrics**

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import statsmodels.api as smd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
data = pd.read_csv('~/Documents/Sem 6/ML/Exercise 3/telecom_customer_churn.csv')

data.head()

data.shape

data.info()

for column in data.columns:
    print(data[column].value_counts().head())

data[data.columns[data.isnull().any()]].isnull().sum()* 100 / data.shape[0]

data = data.drop(['Churn Category','Churn Reason', 'Offer'],axis=1)

object_columns_data = data.select_dtypes(include=['object'])
numerical_columns_data =data.select_dtypes(exclude=['object'])

null_counts = object_columns_data.isnull().sum()
print("Number of null values in each column:\n{}".format(null_counts))
columns_None = ['Multiple Lines']

object_columns_data[columns_None]= object_columns_data[columns_None].fillna('None')
columns_with_lowNA = ['Internet Type', 'Online Security', 'Online Backup',
                      'Device Protection Plan', 'Premium Tech Support', 'Streaming TV',
                      'Streaming Movies', 'Streaming Music', 'Unlimited Data']

#fill missing values for each column (using its own most frequent value)
object_columns_data[columns_with_lowNA] =
object_columns_data[columns_with_lowNA].fillna(object_columns_data.mode().iloc[0])
```

```
for column in object_columns_data.columns:
    print(object_columns_data[column].value_counts().head())

bin_map  = {'Yes':1, 'No':0}
object_columns_data['Married'] = object_columns_data['Married'].map(bin_map)
object_columns_data['Phone Service'] = object_columns_data['Phone Service'].map(bin_map)
object_columns_data['Multiple Lines'] = object_columns_data['Multiple Lines'].map(bin_map)
object_columns_data['Internet Service'] = object_columns_data['Internet Service'].map(bin_map)
object_columns_data['Online Security'] = object_columns_data['Online Security'].map(bin_map)
object_columns_data['Online Backup'] = object_columns_data['Online Backup'].map(bin_map)
object_columns_data['Device Protection Plan'] = object_columns_data['Device Protection Plan'].map(bin_map)
object_columns_data['Premium Tech Support'] = object_columns_data['Premium Tech Support'].map(bin_map)
object_columns_data['Streaming TV'] = object_columns_data['Streaming TV'].map(bin_map)
object_columns_data['Streaming Movies'] = object_columns_data['Streaming Movies'].map(bin_map)
object_columns_data['Streaming Music'] = object_columns_data['Streaming Music'].map(bin_map)
object_columns_data['Unlimited Data'] = object_columns_data['Unlimited Data'].map(bin_map)
object_columns_data['Paperless Billing'] = object_columns_data['Paperless Billing'].map(bin_map)

#Select categorical features
rest_object_columns = object_columns_data.select_dtypes(include=['object'])

#Using One hot encoder
rest_object_columns
rc = pd.get_dummies(object_columns_data[['Gender', 'Internet Type', 'Contract', 'Payment Method']])

object_columns_data = pd.concat([object_columns_data, rc], axis=1)

object_columns_data.head()

object_columns_data = object_columns_data.drop(['Gender', 'Internet Type', 'Contract', 'Payment Method'],
axis=1)
object_columns_data.head()
#Number of null values in each feature
null_counts = numerical_columns_data.isnull().sum()
print("Number of null values in each column:\n{}".format(null_counts))

print(numerical_columns_data['Avg Monthly Long Distance Charges'].median())
print(numerical_columns_data['Avg Monthly GB Download'].median())

numerical_columns_data['Avg Monthly Long Distance Charges'] = numerical_columns_data['Avg Monthly Long
Distance Charges'].fillna(25.72)

numerical_columns_data['Avg Monthly GB Download'] = numerical_columns_data['Avg Monthly GB
Download'].fillna(21.0)

numerical_columns_data= numerical_columns_data.fillna(0)

df = pd.concat([object_columns_data, numerical_columns_data], axis=1,sort=False)
cols = df.columns.tolist()
df.head()
```

```
df = df.fillna(0)

df.info()

df = df.drop(['Customer ID', 'City'], axis=1)
df.head()

from sklearn.model_selection import train_test_split
X = df.drop('Customer Status', axis=1)
y = df['Customer Status']

X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7, test_size=0.3, random_state=100)

X_train.head()

scaler = StandardScaler()
X_train[['Monthly Charge','Total Charges']] = scaler.fit_transform(X_train[['Monthly Charge','Total Charges']])
X_train.head()


df.head()

cor = df.corr()
cor

correlated_features = set()
for i in range(len(cor.columns)):
    for j in range(i):
        if abs(cor.iloc[i, j]) > 0.7:
            colname1 = cor.columns[i]
            colname2 = cor.columns[j]
            print(abs(cor.iloc[i, j]), "--", i, '--', j, '--', colname1, '--', colname2)
            correlated_features.add(colname1)
            correlated_features.add(colname2)

print(cor.columns)
print('------')
print(correlated_features)

X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size=0.3, random_state=42)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

model = LogisticRegression()
model.fit(X_train_scaled, Y_train)

Y_pred = model.predict(X_test_scaled)
```

```
accuracy = accuracy_score(Y_test, Y_pred)
conf_matrix = confusion_matrix(Y_test, Y_pred)
classification_rep = classification_report(Y_test, Y_pred)

print(f"Accuracy: {accuracy:.2f}")
print("\nConfusion Matrix:")
print(conf_matrix)
print("\nClassification Report:")
print(classification_rep)
```

5. **Apply all the classification algorithms (KNN, Logisitc Regression, Naive Bayes, Decision Trees, SVM) on this dataset and print the accuracies.**
Find which algorithm gave the best accuracy.

```
#import statements

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

df = pd.read_csv("Telco-Customer-Churn.csv")
df

df = df.drop("customerID", axis=1)
df

categorical_columns = df.select_dtypes(include=['object']).columns
for col in categorical_columns:
    df[col] = LabelEncoder().fit_transform(df[col])
df

df = df.dropna()
df

x = df.iloc[:,:-1].values
y = df.iloc[:,-1].values
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)

scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)

classifiers = {
```

```
    'KNN': KNeighborsClassifier(),
    'Logistic Regression': LogisticRegression(),
    'Naive Bayes': GaussianNB(),
    'Decision Tree': DecisionTreeClassifier(),
    'SVM': SVC()
}

for name, classifier in classifiers.items():
    classifier.fit(x_train, y_train)
    y_pred = classifier.predict(x_test)
    accuracy = accuracy_score(y_test, y_pred)
    print(f'{name} Accuracy: {accuracy}')

best_algorithm = max(classifiers, key=lambda k: accuracy_score(y_test, classifiers[k].predict(x_test)))
print(f'The best algorithm is: {best_algorithm}')


## Justification

Logistic Regression providing the best accuracy could be attributed to several factors:

- **Linear Relationship:**
  Logistic Regression assumes a linear relationship between the independent variables and the log-odds of the dependent variable. In scenarios where the relationship is approximately linear, Logistic Regression tends to perform well.

- **Simplicity:**
  Logistic Regression is a relatively simple algorithm compared to some other classification algorithms. It is less prone to overfitting, especially when the dataset is not extremely complex.

- **Interpretability:**
  Logistic Regression provides interpretable coefficients that can be useful in understanding the impact of each feature on the predicted outcome.

- **No Assumption of Normality:**
  Logistic Regression does not assume that the independent variables are normally distributed, making it robust to deviations from normality.

- **Binary Classification Task:**
  Since the problem seems to be a binary classification task (Churn: Yes/No), Logistic Regression is well-suited for such problems.
```

6. **Use the attached file and run SVM, Decision tree, Random Forest and any one boosting algorithm. Find out the different tunable parameters for each algorithms mentioned above. Find out the different tunable parameters for each algorithms mentioned above. Apply gridsearchCV and randomizedsearchCV for all the above classification algorithms and get the best parameters.**

```
import pandas as pd

data=pd.read_csv(r'./Telco-Customer-Churn.csv')
data
```

```
data.drop(labels=['customerID'], axis=1, inplace=True)
data
```

```
from sklearn.preprocessing import LabelEncoder

le=LabelEncoder()
object_columns=data.select_dtypes(include=['object']).columns
for col in object_columns:
    data[col]=le.fit_transform(data[col])
data
```

```
data.dropna(inplace=True)
```

```
from sklearn.model_selection import train_test_split

x=data.iloc[:,:-1]
y=data.iloc[:,-1]

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=56)
```

```
from sklearn.preprocessing import StandardScaler

scaler=StandardScaler()
x_train=scaler.fit_transform(x_train)
x_test=scaler.transform(x_test)
```

```
from sklearn import svm
from sklearn import metrics
from sklearn.metrics import classification_report

"""
PARAMS
C: Regularization parameter.
kernel: Specifies the kernel type to be used in the algorithm.
coef0 : Independent term in kernel function.
"""

svc=svm.SVC(C=10, kernel='linear')
svc.fit(x_train, y_train)
y_pred=svc.predict(x_test)
print(classification_report(y_test, y_pred))
```

```
from sklearn.model_selection import GridSearchCV
```

```
param_grid = {'C': [0.1, 1, 10, 100],'kernel': ['linear'], 'coef0': [10, 20, 40]}
grid = GridSearchCV(svm.SVC(), param_grid, refit = True, verbose = 3,n_jobs=-1)
grid.fit(x_train, y_train)
y_pred=grid.predict(x_test)
print(grid.best_params_)
print(classification_report(y_test, y_pred))
```

```
from sklearn import tree

"""
PARAMS
criterion :  The function to measure the quality of a split.
max_depth :  The maximum depth of the tree.
random_state:Controls the randomness of the estimator.
"""

dt=tree.DecisionTreeClassifier(criterion="gini", max_depth=15, random_state=45)
dt.fit(x_train, y_train)
y_pred=dt.predict(x_test)
print(classification_report(y_test, y_pred))
```

```
param_grid = {'criterion': ['gini','entropy'], 'max_depth': [10, 15, 20, 30], 'random_state':[20, 45, 112]}
grid = GridSearchCV(tree.DecisionTreeClassifier(), param_grid, refit = True, verbose = 3,n_jobs=-1)
grid.fit(x_train, y_train)
y_pred=grid.predict(x_test)
print(grid.best_params_)
print(classification_report(y_test, y_pred))
```

```
from sklearn import ensemble

"""
PARAMS
n_estimators: Number of trees in the forest.
criterion: The function to measure the quality of a split.
max_features: The number of features to consider when looking for the best split.
"""

rfc=ensemble.RandomForestClassifier(n_estimators=50, criterion="gini", max_features="sqrt")
rfc.fit(x_train, y_train)
y_pred=rfc.predict(x_test)
print(classification_report(y_test, y_pred))
```

```
param_grid = {'n_estimators': [50, 100,200],'criterion': ['gini','entropy'], 'max_depth': [10, 15, 20, 30],
'max_features':["log2", "sqrt"]}
grid = GridSearchCV(ensemble.RandomForestClassifier(), param_grid, refit = True, verbose = 3,n_jobs=-1)
grid.fit(x_train, y_train)
y_pred=grid.predict(x_test)
print(grid.best_params_)
print(classification_report(y_test, y_pred))
```

```
ada=ensemble.AdaBoostClassifier(n_estimators=75, learning_rate=1.5, random_state=45)
ada.fit(x_train, y_train)
y_pred=ada.predict(x_test)
print(classification_report(y_test, y_pred))
```

```
param_grid = {'n_estimators': [50, 75, 100], 'learning_rate': [0.75, 1.0, 1.5], 'random_state':[20, 45, 70]}
grid = GridSearchCV(ensemble.AdaBoostClassifier(), param_grid, refit = True, verbose = 3,n_jobs=-1)
grid.fit(x_train, y_train)
y_pred=grid.predict(x_test)
print(grid.best_params_)
print(classification_report(y_test, y_pred))
```

7. Use the given dataset and implement K-Means from scratch and use the sklearn K-Means implementation. Compare the results of both the implementations and write your inferences in the ipynb file itself.

```
import pandas as pd
import numpy as np
import sklearn
import matplotlib.pyplot as plt
```

```
df = pd.read_csv('data.csv')
```

```
df.head()
```

```
df.info()
```

```
gender = pd.get_dummies(df['Gender'])
df = df.drop(['Gender'], axis=1)
df.head()
```

```
from sklearn.model_selection import train_test_split

x=df.iloc[:,:-1]
y=df.iloc[:,-1]

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=56)
```

```
from sklearn import preprocessing

X_train_norm = preprocessing.normalize(x_train)
X_test_norm = preprocessing.normalize(x_test)
```

```
from sklearn.cluster import KMeans
kmeans_sklearn = KMeans(n_clusters=3)
kmeans_sklearn.fit(X_train_norm)
```

```
sklearn_centroids = kmeans_sklearn.cluster_centers_
sklearn_labels = kmeans_sklearn.labels_
```

```
import numpy as np

class KMeansCustom:
    def __init__(self, n_clusters=8, max_iter=300):
        self.n_clusters = n_clusters
        self.max_iter = max_iter

    def fit(self, X):
        # Randomly initialize centroids
        centroids_idx = np.random.choice(len(X), size=self.n_clusters, replace=False)
        centroids = X[centroids_idx]

        for _ in range(self.max_iter):
            # Assign each data point to the nearest centroid
            labels = self._assign_clusters(X, centroids)

            # Update centroids
            new_centroids = self._update_centroids(X, labels)

            # Check for convergence
            if np.allclose(centroids, new_centroids):
                break

            centroids = new_centroids

        self.labels_ = labels
        self.cluster_centers_ = centroids

    def _assign_clusters(self, X, centroids):
        distances = np.sqrt(((X - centroids[:, np.newaxis])**2).sum(axis=2))
        return np.argmin(distances, axis=0)

    def _update_centroids(self, X, labels):
        centroids = np.zeros((self.n_clusters, X.shape[1]))
        for i in range(self.n_clusters):
            centroids[i] = np.mean(X[labels == i], axis=0)
        return centroids
```

```
# Initialize and fit the KMeansCustom model
kmeans_custom = KMeansCustom(n_clusters=3)
kmeans_custom.fit(X_train_norm)
custom_centroids = kmeans_custom.cluster_centers_
custom_labels = kmeans_custom.labels_
```

```
# Compare centroids
print("Custom Implementation Centroids:")
```

```
print(custom_centroids)
print("\nSklearn Implementation Centroids:")
print(sklearn_centroids)

# Compare cluster assignments
print("\nCustom Implementation Labels:")
print(custom_labels)
print("\nSklearn Implementation Labels:")
print(sklearn_labels)

# Compute metrics for comparing cluster assignments (optional)
from sklearn.metrics import adjusted_rand_score
ari = adjusted_rand_score(custom_labels, sklearn_labels)
print("\nAdjusted Rand Index:", ari)

# Visualize clusters (optional)
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.scatter(X_train_norm[:, 0], X_train_norm[:, 1], c=custom_labels, cmap='viridis')
plt.title('Custom K-Means Clusters')
plt.scatter(custom_centroids[:, 0], custom_centroids[:, 1], marker='x', color='red', s=100)

plt.subplot(1, 2, 2)
plt.scatter(X_train_norm[:, 0], X_train_norm[:, 1], c=sklearn_labels, cmap='viridis')
plt.title('Sklearn K-Means Clusters')
plt.scatter(sklearn_centroids[:, 0], sklearn_centroids[:, 1], marker='x', color='red', s=100)

plt.show()
```

Inference

The adjusted rand index is 0.975, which means that the custom algorithm and sklearn have classified the points into 3 clusters in a very similar way.

Based on the plots, it can be said that most of the points are classified in the same cluster. The points that are in cluster 1 in the custom implementation are in cluster 0 in the sklearn implementation. A few point that are in the bottommost cluster in the custom implementation are in the middle cluster in the sklearn implementation.

8. **Download MNIST dataset, apply PCA from scratch.**

```
# MNIST dataset downloaded from Kaggle : https://www.kaggle.com/c/digit-recognizer/data

# importing necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# load data from csv into pandas dataframe
```

```
mnist_df = pd.read_csv('train.csv')

# display first few rows
mnist_df.head()
```

```
# store the labels into a variable y
y = mnist_df['label']

# store the pixel data in X
X = mnist_df.drop("label", axis = 1)
```

```
# shape of data
print(X.shape)
print(y.shape)
```

```
# display some random number from datset
for i in [2, 99, 10001, 20422]:
    index = i
    plt.figure(figsize = (3,3))
    grid_data = X.iloc[index].values.reshape(28,28)  # reshape from 1d to 2d pixel array
    plt.imshow(grid_data, interpolation = "none", cmap = 'gray')
    plt.show()
    # print the corresponding class label
    print("- "*50)
    print("class label:", y[index])
```

Implementing PCA from scratch

```
# Data-preprocessing: Standardizing the data with standard scaler
from sklearn.preprocessing import StandardScaler

X_std = StandardScaler().fit_transform(X)
print(X_std.shape)
```

```
# find the co-variance matrix:  (A^T * A)/n
# matrix multiplication using numpy
covar_matrix = np.matmul(X_std.T , X_std)/X_std.shape[1]
print ( "The shape of covariance matrix = ", covar_matrix.shape)
```

```
# find top two eigen-values and corresponding eigen-vectors for projection on a 2-D space

from scipy.linalg import eigh

# the parameter 'eigvals' is defined (low value to high value)
# eigh function will return the eigen values in asending order
# this code generates only the top 2 (782 and 783) eigenvalues

values, vectors = eigh(covar_matrix, eigvals = (782,783))

print("Shape of eigen vectors: ",vectors.shape)
```

```
# projecting the original data sample on the plane formed by two principal eigen vectors by vector-vector multiplication

new_coordinates = np.matmul(vectors.T, X_std.T)
print ("resultant new data points' shape ", vectors.T.shape, "X", X_std.T.shape," = ", new_coordinates.shape)
```

```
# appending class label to the new coordinated in 2D projected space
new_coordinates = np.vstack((new_coordinates, y)).T

# creating a new data frame for plotting the labeled points
new_df = pd.DataFrame(data = new_coordinates, columns = ("2nd_principal", "1st_principal", "label"))
print(new_df.head())
```

```
# ploting the 2D data points with seaborn

sns.lmplot(x = '1st_principal', y = '2nd_principal', data = new_df,
           hue = 'label', legend = True, legend_out = True, height = 10,
           fit_reg = False, scatter_kws = {'alpha': 0.5})
plt.show()
```

9. Implement a neural network from scratch. Take any dataset. If you take a regression problem, use the equations derived in the class

For the same dataset, build a neural network using keras library. Run the same number of epochs and compare the results obtained with your model vs the built-in keras model.

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

```
np.random.seed(0)
X = np.random.randn(100, 2)
y = np.logical_xor(X[:, 0] > 0, X[:, 1] > 0)
y = np.where(y, 1, 0)


plt.scatter(X[:,0], X[:,1], c=y, cmap=plt.cm.Spectral)
plt.show()
```

```
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def sigmoid_derivative(x):
    return sigmoid(x) * (1 - sigmoid(x))
```

```
class NeuralNetwork:
    def __init__(self, input_size, hidden_size, output_size):
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.output_size = output_size

        self.weights_input_hidden = np.random.randn(self.input_size, self.hidden_size)
        self.weights_hidden_output = np.random.randn(self.hidden_size, self.output_size)

    def forward(self, X):
        self.hidden_input = np.dot(X, self.weights_input_hidden)
        self.hidden_output = sigmoid(self.hidden_input)
        self.output = sigmoid(np.dot(self.hidden_output, self.weights_hidden_output))
        return self.output

    def backward(self, X, y, learning_rate):

        output_error = y - self.output
        output_delta = output_error * sigmoid_derivative(self.output)

        hidden_error = output_delta.dot(self.weights_hidden_output.T)
        hidden_delta = hidden_error * sigmoid_derivative(self.hidden_output)


        self.weights_hidden_output += self.hidden_output.T.dot(output_delta) * learning_rate
        self.weights_input_hidden += X.T.dot(hidden_delta) * learning_rate

    def train(self, X, y, epochs, learning_rate):
        self.loss = []
        for i in range(epochs):
            output = self.forward(X)
            self.backward(X, y, learning_rate)
            loss = np.mean(np.square(y - output))
            self.loss.append(loss)
            if i % 10 == 0:
                print(f'Epoch {i}, Loss: {loss:.4f}')

    def predict(self, X):
        return np.round(self.forward(X))
```

```
input_size = 2
hidden_size = 4
output_size = 1
nn = NeuralNetwork(input_size, hidden_size, output_size)


epochs = 400
learning_rate = 1.0
nn.train(X, y.reshape(-1, 1), epochs, learning_rate)

# Plot loss curve
```

```
plt.plot(range(epochs), nn.loss)
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training Loss Curve')
plt.show()
```

keras model

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import SGD
from keras.utils import to_categorical


X1 = np.random.randn(100, 2) + np.array([2, 2])   # Class 1
X2 = np.random.randn(100, 2) + np.array([-2, -2]) # Class 2
X = np.vstack([X1, X2])
y = np.array([0] * 100 + [1] * 100)


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)


y_train_encoded = to_categorical(y_train)
y_test_encoded = to_categorical(y_test)


model = Sequential([
    Dense(10, activation='relu', input_shape=(X_train.shape[1],)),
    Dense(2, activation='softmax')
])


optimizer = SGD(learning_rate=1.0)


model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])


history = model.fit(X_train, y_train_encoded, epochs=200, validation_data=(X_test, y_test_encoded), verbose=0)
```

```
loss, accuracy = model.evaluate(X_test, y_test_encoded)
print("Test Loss:", loss)
print("Test Accuracy:", accuracy)


plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

Inference

- By building a neural network from scratch, one can customize the functions used and change them according to the needs. They will have a better understanding of how the neural network works as well.
- On the other hand, a neural network can be built quickly and efficiently by using functions from the Keras library, which is more user-friendly.