```
import tensorflow as tf
from tensorflow import keras
fashion_mnist = keras.datasets.fashion_mnist
(X_train_full, y_train_full), (X_test, y_test) = fashion_mnist.load_data()


from sklearn.model_selection import train_test_split
X_train, X_valid, y_train, y_valid = train_test_split( X_train_full, y_train_full)

X_valid, X_train = X_valid[::] / 255.0, X_train[::] / 255.0

class_names = ["T-shirt/top", "Trouser", "Pullover", "Dress", "Coat","Sandal", "Shirt", "Sneaker", "Bag", "Ankle boot"]
class_names[y_train[0]]

model = keras.models.Sequential()

model.add(keras.layers.Flatten(input_shape=[28, 28]))
model.add(keras.layers.Dense(300, activation="relu"))
model.add(keras.layers.Dense(100, activation="relu"))
model.add(keras.layers.Dense(10, activation="softmax"))

model.summary()
model.compile(loss="sparse_categorical_crossentropy",optimizer="sgd",metrics=["accuracy"])
history = model.fit(X_train, y_train, epochs=30,validation_data=(X_valid, y_valid))

loss,accuracy=model.evaluate(X_test, y_test)
print(accuracy*100)

X_test = X_test[::] / 255.0
model.evaluate(X_test, y_test)
X_new = X_test[:3]/255.0
y_pred = model.predict(X_new)
y_pred
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/reshaping/flatten.py:37: UserWarning: Do not pass an `input_shape`/`input_d
  super().__init__(**kwargs)
Model: "sequential_3"
```

Layer (type)	Output Shape	Param #
flatten_1 (Flatten)	(None, 784)	0
dense_6 (Dense)	(None, 300)	235,500
dense_7 (Dense)	(None, 100)	30,100
dense_8 (Dense)	(None, 10)	1,010

```
 Total params: 266,610 (1.02 MB)
 Trainable params: 266,610 (1.02 MB)
 Non-trainable params: 0 (0.00 B)
Epoch 1/30
1407/1407 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.6660 - loss: 1.0632 - val_accuracy: 0.7676 - val_loss: 0.6444
Epoch 2/30
1407/1407 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.8162 - loss: 0.5373 - val_accuracy: 0.8324 - val_loss: 0.4783
Epoch 3/30
1407/1407 ━━━━━━━━━━━━━━━━━━━━ 9s 4ms/step - accuracy: 0.8355 - loss: 0.4729 - val_accuracy: 0.8095 - val_loss: 0.5371
Epoch 4/30
1407/1407 ━━━━━━━━━━━━━━━━━━━━ 11s 5ms/step - accuracy: 0.8447 - loss: 0.4451 - val_accuracy: 0.8473 - val_loss: 0.4388
Epoch 5/30
1407/1407 ━━━━━━━━━━━━━━━━━━━━ 10s 5ms/step - accuracy: 0.8514 - loss: 0.4255 - val_accuracy: 0.8541 - val_loss: 0.4102
Epoch 6/30
1407/1407 ━━━━━━━━━━━━━━━━━━━━ 11s 6ms/step - accuracy: 0.8581 - loss: 0.4020 - val_accuracy: 0.8393 - val_loss: 0.4642
Epoch 7/30
1407/1407 ━━━━━━━━━━━━━━━━━━━━ 10s 6ms/step - accuracy: 0.8627 - loss: 0.3933 - val_accuracy: 0.8649 - val_loss: 0.3829
Epoch 8/30
1407/1407 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.8667 - loss: 0.3779 - val_accuracy: 0.8656 - val_loss: 0.3827
Epoch 9/30
1407/1407 ━━━━━━━━━━━━━━━━━━━━ 8s 6ms/step - accuracy: 0.8704 - loss: 0.3646 - val_accuracy: 0.8652 - val_loss: 0.3822
Epoch 10/30
1407/1407 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.8749 - loss: 0.3508 - val_accuracy: 0.7683 - val_loss: 0.6536
Epoch 11/30
1407/1407 ━━━━━━━━━━━━━━━━━━━━ 10s 5ms/step - accuracy: 0.8788 - loss: 0.3448 - val_accuracy: 0.8619 - val_loss: 0.3934
Epoch 12/30
1407/1407 ━━━━━━━━━━━━━━━━━━━━ 8s 6ms/step - accuracy: 0.8787 - loss: 0.3403 - val_accuracy: 0.8709 - val_loss: 0.3627
Epoch 13/30
1407/1407 ━━━━━━━━━━━━━━━━━━━━ 10s 5ms/step - accuracy: 0.8816 - loss: 0.3293 - val_accuracy: 0.8662 - val_loss: 0.3770
Epoch 14/30
1407/1407 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.8839 - loss: 0.3259 - val_accuracy: 0.8731 - val_loss: 0.3572
Epoch 15/30
1407/1407 ━━━━━━━━━━━━━━━━━━━━ 10s 4ms/step - accuracy: 0.8890 - loss: 0.3140 - val_accuracy: 0.8523 - val_loss: 0.4052
Epoch 16/30
1407/1407 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.8909 - loss: 0.3038 - val_accuracy: 0.8719 - val_loss: 0.3543
Epoch 17/30
1407/1407 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.8912 - loss: 0.3038 - val_accuracy: 0.8753 - val_loss: 0.3499
Epoch 18/30
1407/1407 ━━━━━━━━━━━━━━━━━━━━ 8s 6ms/step - accuracy: 0.8955 - loss: 0.2932 - val_accuracy: 0.8770 - val_loss: 0.3473
Epoch 19/30
1407/1407 ━━━━━━━━━━━━━━━━━━━━ 12s 7ms/step - accuracy: 0.8953 - loss: 0.2911 - val_accuracy: 0.8637 - val_loss: 0.3753
Epoch 20/30
1407/1407 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.8994 - loss: 0.2848 - val_accuracy: 0.8800 - val_loss: 0.3349
Epoch 21/30
1407/1407 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.9004 - loss: 0.2770 - val_accuracy: 0.8805 - val_loss: 0.3359
Epoch 22/30
1407/1407 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.9011 - loss: 0.2730 - val_accuracy: 0.8833 - val_loss: 0.3273
Epoch 23/30
1407/1407 ━━━━━━━━━━━━━━━━━━━━ 11s 6ms/step - accuracy: 0.9021 - loss: 0.2708 - val_accuracy: 0.8792 - val_loss: 0.3451
Epoch 24/30
1407/1407 ━━━━━━━━━━━━━━━━━━━━ 8s 6ms/step - accuracy: 0.9068 - loss: 0.2617 - val_accuracy: 0.8863 - val_loss: 0.3217
Epoch 25/30
1407/1407 ━━━━━━━━━━━━━━━━━━━━ 6s 5ms/step - accuracy: 0.9037 - loss: 0.2613 - val_accuracy: 0.8834 - val_loss: 0.3272
Epoch 26/30
1407/1407 ━━━━━━━━━━━━━━━━━━━━ 8s 6ms/step - accuracy: 0.9094 - loss: 0.2489 - val_accuracy: 0.8837 - val_loss: 0.3208
Epoch 27/30
1407/1407 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.9083 - loss: 0.2529 - val_accuracy: 0.8880 - val_loss: 0.3201
Epoch 28/30
1407/1407 ━━━━━━━━━━━━━━━━━━━━ 10s 5ms/step - accuracy: 0.9096 - loss: 0.2489 - val_accuracy: 0.8761 - val_loss: 0.3477
Epoch 29/30
1407/1407 ━━━━━━━━━━━━━━━━━━━━ 8s 6ms/step - accuracy: 0.9120 - loss: 0.2409 - val_accuracy: 0.8866 - val_loss: 0.3199
Epoch 30/30
1407/1407 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.9169 - loss: 0.2337 - val_accuracy: 0.8786 - val_loss: 0.3518
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8667 - loss: 0.3784
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 63ms/step
array([[0.06981207, 0.02124535, 0.04405966, 0.09455435, 0.01718243,
        0.51279676, 0.08281755, 0.12551674, 0.02397619, 0.00803886],
       [0.07738118, 0.02220649, 0.04938439, 0.09969143, 0.01920634,
        0.48865128, 0.09026166, 0.12002552, 0.02509572, 0.00809605],
       [0.07863333, 0.02349445, 0.04726755, 0.10378694, 0.01832011,
        0.48674482, 0.08676431, 0.12267151, 0.02428198, 0.00803502]],
      dtype=float32)
```

Tensor OR gate

```
import numpy as np
X=np.array([[0, 0],[0, 1],[1, 0],[1, 1]])
y=np.array([0, 1, 1, 1])
import tensorflow as tf
from tensorflow.keras.layers import Dense

model = keras.models.Sequential()
model.add( Dense(units=1,input_dim=2,activation='sigmoid') )
model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])
model.fit(X,y,epochs=1000,verbose=0)


loss,accuracy=model.evaluate(X,y)
print(f"Model accuracy:{accuracy*100:.2f}%")

predictions=model.predict(X)
predictions=(predictions>0.5).astype(int)
print("Predictions:")
print(predictions)
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` arg
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 150ms/step - accuracy: 0.7500 - loss: 0.3431
WARNING:tensorflow:5 out of the last 5 calls to <function TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distribu
Model accuracy:75.00%
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step
Predictions:
[[1]
 [1]
 [1]
 [1]]
```

```
import numpy as np
X=np.array([[0, 0],[1, 0],[0, 1],[1, 1]])
y=np.array([0, 1, 1, 0])
import tensorflow as tf
from tensorflow.keras.layers import Dense

model = keras.models.Sequential()

model.add( Dense(units=2,input_dim=2,activation='relu') )
model.add( Dense(units=1,activation='sigmoid'))

model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])
model.fit(X,y,epochs=100,verbose=0)

loss,accuracy=model.evaluate(X,y)
print(f"Model accuracy:{accuracy*100:.2f}%")

predictions=model.predict(X)
predictions=(predictions>0.5).astype(int)
print("Predictions:")
print(predictions)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 149ms/step - accuracy: 0.5000 - loss: 0.7165
Model accuracy:50.00%
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 49ms/step
Predictions:
[[1]
 [1]
 [0]
 [0]]
```

coeffient correlation on qinspection

```
import sklearn
import pandas as pd
import numpy
from sklearn.preprocessing import StandardScaler

df=pd.read_csv("/content/Qinspection.csv")
df.head()
x=df.iloc[:,0:6]
y=df.iloc[:,-1]
scaler=StandardScaler()
std_predictor_features=scaler.fit_transform(x)

new_df=pd.DataFrame(std_predictor_features,columns=x.columns)
```

```
print(new_df.head())

abs_cor_matrix=new_df.corr().abs()
upper_cor_matrix=abs_cor_matrix.where(numpy.triu(numpy.ones(abs_cor_matrix.shape),k=1).astype(bool))

features=[column for column in upper_cor_matrix.columns if any(upper_cor_matrix[column]>0.95)]
print()
print(features) # remove redundant features highly correlated
print()
df1=new_df.drop(new_df[features],axis=1)
print(df1.head())
# merged_df=pd.concat([df1,y],axis=1)
# merged_df.head()
```

```
   feature1  feature2  feature3  feature4  feature5  feature6
0 -1.379335 -1.379335  0.329517 -1.397616 -1.316654 -1.316654
1 -1.499061 -1.499061  0.102481 -1.284200 -1.316654 -1.316654
2 -1.020160 -1.020160  1.237661 -1.340908 -1.316654 -1.316654
3 -0.541258 -0.541258  1.918768 -1.170784 -1.053140 -1.053140
4 -1.499061 -1.499061  0.783589 -1.340908 -1.184897 -1.184897

['feature2', 'feature5', 'feature6']

   feature1  feature3  feature4
0 -1.379335  0.329517 -1.397616
1 -1.499061  0.102481 -1.284200
2 -1.020160  1.237661 -1.340908
3 -0.541258  1.918768 -1.170784
4 -1.499061  0.783589 -1.340908
```

Demonstrate PCA using iris dataset

```
import numpy as np
import pandas as pd
from sklearn import datasets
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
# iris = datasets.load_iris()
# type(iris)
# X = iris.data[:, :4]
# y = iris.target
df=pd.read_csv('/content/Iris (1).csv')
print(df.head())
X=df.iloc[:,1:5]
y=df.iloc[:,-1]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

pca4 = PCA()
X_train = pca4.fit_transform(X_train)
X_test = pca4.transform(X_test)

explained_variance = pca4.explained_variance_ratio_
print('explained variance=',explained_variance)

# from sklearn.ensemble import RandomForestClassifier
# classifier = RandomForestClassifier(max_depth=2, random_state=0)
# classifier.fit(X_train, y_train)
# y_pred = classifier.predict(X_test)
# cm = confusion_matrix(y_test, y_pred)
# print(cm)
# print('Accuracy=' + str(accuracy_score(y_test, y_pred)))
```

```
   Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm      Species
0   1            5.1           3.5            1.4           0.2  Iris-setosa
1   2            4.9           3.0            1.4           0.2  Iris-setosa
2   3            4.7           3.2            1.3           0.2  Iris-setosa
3   4            4.6           3.1            1.5           0.2  Iris-setosa
4   5            5.0           3.6            1.4           0.2  Iris-setosa
explained variance= [0.72226528 0.23974795 0.03338117 0.0046056 ]
```

CHI2

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
iris_dataset=pd.read_csv("/content/Iris (1).csv")
df=pd.DataFrame(iris_dataset)

X = iris_dataset.iloc[:,1:5]
y = iris_dataset.iloc[:,-1]
print(X.head())
chi2_features = SelectKBest(chi2, k = 2) # chi2 relationships between categorical features and the target.
kbest_features = chi2_features.fit_transform(X, y)

print('Original feature number:', X.shape[1])
print('Reduced feature number:', kbest_features.shape[1])
print()
print("scores for the features ")

for i in range(len(chi2_features.scores_)):
  print('Feature %d: %f' % (i, chi2_features.scores_[i]))
            # or
# for i, score in enumerate(chi2_features.scores_):
#     print(f'Feature {i}: {score:.2f}')

print()
print("selected features ")
selectedfetures=chi2_features.get_feature_names_out()
print(selectedfetures)

# newdf=iris_dataset[selectedfetures]
# newdf=pd.concat([newdf,y],axis=1)
# print("final data frame \n")
# display(newdf.head())
```

```
   SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
0            5.1           3.5            1.4           0.2
1            4.9           3.0            1.4           0.2
2            4.7           3.2            1.3           0.2
3            4.6           3.1            1.5           0.2
4            5.0           3.6            1.4           0.2
Original feature number: 4
Reduced feature number: 2

scores for the features
Feature 0: 10.817821
Feature 1: 3.594499
Feature 2: 116.169847
Feature 3: 67.244828

selected features
['PetalLengthCm' 'PetalWidthCm']
```

LDA IRIS

```
import numpy as np
import pandas as pd
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.ensemble import RandomForestClassifier

iris = datasets.load_iris()
X = iris.data[:, :4]
y = iris.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

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

from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
lda = LDA(n_components=1)
```

```
X_train = lda.fit_transform(X_train,y_train)
X_test = lda.transform(X_test)

classifier = RandomForestClassifier(max_depth=2, random_state=0)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
print("Confusion Matrix \n")
cm = confusion_matrix(y_test, y_pred)
print(cm,"\n")
print('Accuracy = ' + str(accuracy_score(y_test, y_pred)))
```

```
Confusion Matrix

[[11  0  0]
 [ 0 13  0]
 [ 0  0  6]]

Accuracy = 1.0
```

Varaince Threshold qin

```
from typing import final
import sklearn
import pandas as pd
import numpy
from sklearn.feature_selection import VarianceThreshold
df=pd.read_csv("/content/Qinspection.csv")

X=df.iloc[:,0:6]
y=df.iloc[:,-1]

selector = VarianceThreshold(threshold=0.6)
selector.fit(X)
selector_predictor=selector.fit_transform(X)

print("features along with their variance ")
for s in zip(X.columns,selector.variances_):
  display(s)

remaining_features =X.columns[selector.get_support()]
print("\nFeatures with Threshold 0.6 \n",remaining_features)

finaldf=df[remaining_features]
finaldf=pd.concat([finaldf,y],axis=1)
print("Final Data Frame \n",finaldf.head())
```

```
features along with their variance
('feature1', 0.6976345486111111)
('feature2', 0.6976345486111113)
('feature3', 0.19400414737654317)
('feature4', 3.1096484375)
('feature5', 0.5760411844135802)
('feature6', 0.5760411844135803)

Features with Threshold 0.6
 Index(['feature1', 'feature2', 'feature4'], dtype='object')
Final Data Frame
    feature1  feature2  feature4 class
0       4.9       4.7       1.3     A
1       4.8       4.6       1.5     A
2       5.2       5.0       1.4     A
3       5.6       5.4       1.7     A
```

Pima logistic

```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, precision_score, recall_score, f1_score

df=pd.read_csv('/content/pima-indians-diabetes.csv')

X=df.iloc[:,0:8]
y=df.iloc[:,-1]
```

```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3, random_state=42)

model = LogisticRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

from sklearn.metrics import confusion_matrix, accuracy_score
import seaborn as sns
import matplotlib.pyplot as plt

cm = confusion_matrix(y_test, y_pred)
print(" The confusion matrix is=")
print(cm)

print("The accuracy score is=")
print(accuracy_score(y_test, y_pred))
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
print(classification_report(y_test,y_pred))
# print(" The confusion matrix is in the graphical form")
# sns.heatmap(cm, annot=True, fmt='d')
# plt.xlabel('Predicted')
# plt.ylabel('Actual')
# plt.title('Confusion Matrix')
# plt.show()
```

```
 The confusion matrix is=
[[137  14]
 [ 34  46]]
The accuracy score is=
0.7922077922077922
Precision: 0.7666666666666667
Recall: 0.575
F1 Score: 0.6571428571428571
              precision    recall  f1-score   support

           0       0.80      0.91      0.85       151
           1       0.77      0.57      0.66        80

    accuracy                           0.79       231
   macro avg       0.78      0.74      0.75       231
weighted avg       0.79      0.79      0.78       231
```

logistic defaulter

```
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Load the dataset
df = pd.read_csv('/content/defaulter (1).csv')

X = df[['student', 'balance', 'income']]
y = df['defaulter']


X.loc[:, 'student'] = X['student'].map({'Yes': 1, 'No': 0})

print(X.head())
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.25, random_state=42)

model = LogisticRegression()
model.fit(X_train, y_train)
```

```
y_pred = model.predict(X_test)


accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", cm)


# sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
# plt.xlabel('Predicted')
# plt.ylabel('Actual')
# plt.title('Confusion Matrix')
# plt.show()
```

```
   student      balance       income
0        0   729.526495  44361.62507
1        1   817.180407  12106.13470
2        0  1073.549164  31767.13895
3        0   529.250605  35704.49394
4        0   785.655883  38463.49588
Accuracy: 0.97
Confusion Matrix:
 [[2409   10]
 [  59   22]]
```

```
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import matplotlib.pyplot as plt
# data= import data
# df = pd.DataFrame(data)
X = df.iloc[:, :-1]
y = df.iloc[:, -1]
model = LinearRegression()
model.fit(X, y)
y_pred = model.predict(X)

mae = mean_absolute_error(y, y_pred)
mse = mean_squared_error(y, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y, y_pred)

# Calculate adjusted R-squared
n = len(y)
k = X.shape[1]
adjusted_r2 = 1 - ((1 - r2) * (n - 1) / (n - k - 1))

# Display results
print(f"Mean Absolute Error (MAE): {mae}")
print(f"Mean Squared Error (MSE): {mse}")
print(f"Root Mean Squared Error (RMSE): {rmse}")
print(f"R-squared (R2): {r2}")
print(f"Adjusted R-squared: {adjusted_r2}")



# # Visualize the Results
# # Plot actual vs predicted delivery costs
# plt.scatter(y, y_pred, color='blue')
# plt.plot([min(y), max(y)], [min(y_pred), max(y_pred)], color='red', linewidth=2)
# plt.xlabel('Actual Delivery Costs ($)')
# plt.ylabel('Predicted Delivery Costs ($)')
# plt.title('Actual vs Predicted Delivery Costs')
# plt.show()
```

bank knn

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import seaborn as sns
```

```
df=pd.read_csv('/content/bank-additional.csv', delimiter=';')

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

X_encoded = pd.get_dummies(X, drop_first=True)

X_train, X_test, y_train, y_test = train_test_split(X_encoded, y, test_size=0.3, random_state=42)

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

knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)

y_pred = knn.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
class_report = classification_report(y_test, y_pred)
print("Accuracy:", accuracy)
print("\nClassification Report:\n", class_report)
```

```
Accuracy: 0.8940129449838188

Classification Report:
               precision    recall  f1-score   support

          no       0.91      0.97      0.94      1105
         yes       0.50      0.24      0.32       131

    accuracy                           0.89      1236
   macro avg       0.71      0.60      0.63      1236
weighted avg       0.87      0.89      0.88      1236
```

Navis Bayes spambase

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

data=pd.read_csv('/content/spambase (2).csv')
df = pd.DataFrame(data)

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=42)

model = GaussianNB()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

# y_prob = model.predict_proba(X_test)

accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

print("Accuracy:", accuracy)
print("\nConfusion Matrix:\n", conf_matrix)
print("\nClassification Report:\n", class_report)

# # Print the predicted probabilities for test data
# print("\nPredicted Probabilities for Test Data:")
# for i, probs in enumerate(y_prob):
#     print(f"Instance {i+1}: No: {probs[0]:.4f}, Yes: {probs[1]:.4f}")
```

```
Accuracy: 0.8247646632874729

Confusion Matrix:
 [[592 212]
 [ 30 547]]

Classification Report:
               precision    recall  f1-score   support

           0       0.95      0.74      0.83       804
           1       0.72      0.95      0.82       577
```

```
    accuracy                           0.82      1381
   macro avg       0.84      0.84      0.82      1381
weighted avg       0.86      0.82      0.83      1381
```

Compare random adaboost with descion

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, classification_report
df = pd.read_csv('/content/pima-indians-diabetes.csv')

X=df.iloc[:,0:8]
y=df.iloc[:,-1]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

dt_model = DecisionTreeClassifier(random_state=42)
dt_model.fit(X_train, y_train)
y_pred_dt = dt_model.predict(X_test)


# 1. Decision Tree Classifier
dt_model2 = DecisionTreeClassifier(random_state=42,min_samples_split=4, min_impurity_decrease=0.01)
dt_model2.fit(X_train, y_train)
y_pred_dt2 = dt_model2.predict(X_test)


# 2. Random Forest Classifier
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
y_pred_rf = rf_model.predict(X_test)

# 3. AdaBoost Classifier
adaboost_model = AdaBoostClassifier(estimator=DecisionTreeClassifier(max_depth=1), n_estimators=100, random_state=42,algorithm='SAMME')
adaboost_model.fit(X_train, y_train)
y_pred_boost = adaboost_model.predict(X_test)


ad = AdaBoostClassifier( n_estimators=100, random_state=42,algorithm='SAMME')
ad.fit(X_train, y_train)
y_pred_ad= ad.predict(X_test)


# Evaluating the models
def evaluate_model(y_test, y_pred):
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    print(f"Accuracy: {accuracy:.4f}, Precision: {precision:.4f}, Recall: {recall:.4f}, F1-Score: {f1:.4f}")



print("\nDecision Tree Classifier Results:")
evaluate_model(y_test, y_pred_dt)

print("\nRandom Forest Classifier Results:")
evaluate_model(y_test, y_pred_rf)

print("\nAdaBoost Classifier Results:")
evaluate_model(y_test, y_pred_boost)

evaluate_model(y_test, y_pred_ad)
print("\nDecision Tree Classifier Results:")
evaluate_model(y_test, y_pred_dt2)
```

```
Decision Tree Classifier Results:
Accuracy: 0.7100, Precision: 0.5823, Recall: 0.5750, F1-Score: 0.5786

Random Forest Classifier Results:
Accuracy: 0.7706, Precision: 0.6800, Recall: 0.6375, F1-Score: 0.6581

AdaBoost Classifier Results:
Accuracy: 0.7532, Precision: 0.6620, Recall: 0.5875, F1-Score: 0.6225
Accuracy: 0.7532, Precision: 0.6620, Recall: 0.5875, F1-Score: 0.6225
```

```
Decision Tree Classifier Results:
Accuracy: 0.7273, Precision: 0.6024, Recall: 0.6250, F1-Score: 0.6135
```

Bank-ruptacy descion

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, classification_report
df = pd.read_csv('/content/Qualitative_Bankruptcy.csv')

X=df.iloc[:,0:6]
y=df.iloc[:,-1]

label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)
X_encoded = pd.get_dummies(X, drop_first=True)
X_train, X_test, y_train, y_test = train_test_split(X_encoded, y_encoded, test_size=0.3, random_state=42)
dt_model = DecisionTreeClassifier(random_state=42,min_samples_split=2, min_impurity_decrease=0.01)
dt_model.fit(X_train, y_train)
y_pred= dt_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)

precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}, Precision: {precision:.4f}, Recall: {recall:.4f}, F1-Score: {f1:.4f}")
```

```
Accuracy: 1.0000, Precision: 1.0000, Recall: 1.0000, F1-Score: 1.0000
```

Dbscan mall

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler

df=pd.read_csv('Mall_customers.csv')
df1=df.drop('CustomerID',axis=1)

df_encoded = pd.get_dummies(df1, columns=['Gender'], drop_first=True)
features = df_encoded[['Annual Income (k$)', 'Spending Score (1-100)']].values

scaler = StandardScaler()
features_scaled = scaler.fit_transform(features)


db = DBSCAN(eps=0.4, min_samples=5).fit(features_scaled)
labels = db.labels_

plt.figure(figsize=(8, 5))
scatter = plt.scatter(features[:, 0], features[:, 1],
                      c=labels, cmap='viridis', s=100, marker='o')
plt.scatter(features[labels == -1][:, 0], features[labels == -1][:, 1],
            color='red', s=100, label='Outliers')
plt.colorbar(scatter, label='Cluster Label')

plt.title(f'DBSCAN Clustering (eps=0.4, min_samples=5)')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.grid(True)
plt.show()
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-14-31740c6f4573> in <cell line: 7>()
      5 from sklearn.preprocessing import StandardScaler
      6
----> 7 df=pd.read_csv('Mall_customers.csv')
      8 df1=df.drop('CustomerID',axis=1)
      9

                              4 frames
/usr/local/lib/python3.10/dist-packages/pandas/io/common.py in get_handle(path_or_buf, mode, encoding, compression, memory_map,
is_text, errors, storage_options)
    871         if ioargs.encoding and "b" not in ioargs.mode:
    872             # Encoding
--> 873             handle = open(
    874                 handle,
    875                 ioargs.mode,

FileNotFoundError: [Errno 2] No such file or directory: 'Mall_customers.csv'
```

Next steps: Explain error

Agglo

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import AgglomerativeClustering
from scipy.cluster.hierarchy import dendrogram, linkage

df=pd.read_csv('Mall_customers.csv')
df1=df.drop('CustomerID',axis=1)

df_encoded = pd.get_dummies(df1, columns=['Gender'], drop_first=True)
features = df_encoded[['Annual Income (k$)', 'Spending Score (1-100)']].values
hac = AgglomerativeClustering(n_clusters=4)
labels = hac.fit_predict(features_scaled)

df_encoded['Cluster'] = labels

plt.figure(figsize=(10, 7))
linked = linkage(features_scaled, method='ward')
dendrogram(linked, orientation='top', distance_sort='descending', show_leaf_counts=True)
plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('Sample Index')
plt.ylabel('Distance')
plt.show()

# Step 8: Plot the clusters
plt.figure(figsize=(8, 5))
plt.scatter(features_scaled[:, 0], features_scaled[:, 1], c=labels, cmap='viridis', s=100)
plt.title(f'Hierarchical Agglomerative Clustering (n_clusters=4)')
plt.xlabel('Scaled Annual Income (k$)')
plt.ylabel('Scaled Spending Score (1-100)')
plt.grid(True)
plt.show()
```

Kmeasn mall

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

df=pd.read_csv('Mall_customers.csv')
df1=df.drop('CustomerID',axis=1)

df_encoded = pd.get_dummies(df1, columns=['Gender'], drop_first=True)
df_encoded

features = df_encoded[['Annual Income (k$)', 'Spending Score (1-100)']]
scaler = StandardScaler()
```

```
features_scaled = scaler.fit_transform(features)

kmeans = KMeans(n_clusters=4, n_init=10, random_state=42)
kmeans.fit(features_scaled)


plt.figure(figsize=(10, 6))
plt.scatter(features_scaled[:, 0], features_scaled[:, 1], c=kmeans.labels_, cmap='viridis', marker='o', s=50)
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], c='red', s=200, alpha=0.75, marker='X', label='Centroids')
plt.title('K-Means Clustering of Mall Customers')
plt.xlabel('Scaled Annual Income (k$)')
plt.ylabel('Scaled Spending Score (1-100)')
plt.legend()
plt.grid(True)
plt.show()

k_values = range(1, 11)
inertia = []
for k in k_values:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(features_scaled)
    inertia.append(kmeans.inertia_)


plt.figure(figsize=(10, 6))
plt.plot(k_values, inertia, marker='o')
plt.title('Elbow Method for Optimal K')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('Inertia (Sum of Squared Distances)')
plt.xticks(k_values)
plt.grid(True)
plt.show()
```

svm iris variance

```
import pandas as pd
from sklearn.svm import SVC
from sklearn.preprocessing import LabelEncoder
from sklearn.feature_selection import VarianceThreshold
from matplotlib import pyplot as plt


data = pd.read_csv('Iris (1).csv')

X = data.iloc[:100, 1:3]
y = data.iloc[:100, -1]


encoder = LabelEncoder()
y_encoded = encoder.fit_transform(y)


selector = VarianceThreshold(threshold=0.4)
X_selected = selector.fit_transform(X)

print("Features along with their variance:")
for feature, variance in zip(X.columns, selector.variances_):
    print(f"{feature}: {variance:.4f}")


remaining_features = X.columns[selector.get_support()]
print("\nFeatures with Variance Threshold > 0.6:", remaining_features)


svm = SVC(kernel='linear')
svm.fit(X_selected, y_encoded)


plt.figure(figsize=(10, 6))

colors = {'Iris-setosa': 'red', 'Iris-versicolor': 'blue'}
plt.scatter(X.iloc[:, 0], X.iloc[:, 1], c=y.map(colors), s=50)

plt.title('Iris Dataset Scatter Plot')
plt.xlabel('Sepal Length')
plt.ylabel('Sepal Width')
plt.legend(colors.keys())
plt.grid()

plt.show()
```

```
plt.show()
```

variance descion

```
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.feature_selection import VarianceThreshold, SelectKBest, f_classif
from sklearn.metrics import accuracy_score, classification_report
from sklearn.preprocessing import LabelEncoder

# Load the Product Quality Dataset (adjust the dataset path as needed)
data = pd.read_csv('Product_quality-classification.csv')

# Let's assume the last column is the target variable 'quality'
X = data.iloc[:, :-1]  # Features
y = data.iloc[:, -1]   # Target variable (Product Quality)

# Encode the target variable if it is categorical
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)

# Feature Selection using Variance Threshold
selector_variance = VarianceThreshold(threshold=0.01)  # Keep features with variance > 0.01
X_selected_variance = selector_variance.fit_transform(X)
```