MODELS

LINEAR REGRESSION:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# Load the dataset using pandas
data = pd.read_csv('your_dataset.csv')  # Replace 'your_dataset.csv' with the path to your dataset

# Assuming your dataset has columns 'X1', 'X2', ..., 'Xn' for features and 'y' for target variable
X = data.drop('y', axis=1)  # Input features (excluding the target variable)
y = data['y']             # Target variable

# Split the dataset into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize the features using StandardScaler (although not necessary for Linear Regression)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Create a Linear Regression model
model = LinearRegression()

# Fit the model to the training data
model.fit(X_train_scaled, y_train)
```

```
# Predict the target variable for the test set
y_pred = model.predict(X_test_scaled)


# Calculate the mean squared error
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)
```

LOGISTIC REGRESSION:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score


# Load the dataset using pandas
data = pd.read_csv('your_dataset.csv')  # Replace 'your_dataset.csv' with the path to your dataset


# Assuming your dataset has columns 'X1', 'X2', ..., 'Xn' for features and 'y' for target variable
X = data.drop('y', axis=1)  # Input features (excluding the target variable)
y = data['y']               # Target variable


# Split the dataset into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


# Standardize the features using StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
# Create a Logistic Regression model
model = LogisticRegression()

# Fit the model to the training data
model.fit(X_train_scaled, y_train)

# Predict the labels for the test set
y_pred = model.predict(X_test_scaled)

# Calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

NAÏVE BAYES:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score

# Load the dataset using pandas
data = pd.read_csv('your_dataset.csv')  # Replace 'your_dataset.csv' with the path to your dataset

# Assuming your dataset has columns 'X1', 'X2', ..., 'Xn' for features and 'y' for target variable
X = data.drop('y', axis=1)  # Input features (excluding the target variable)
y = data['y']               # Target variable

# Split the dataset into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# Standardize the features using StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Create a Gaussian Naive Bayes model
model = GaussianNB()

# Fit the model to the training data
model.fit(X_train_scaled, y_train)

# Predict the labels for the test set
y_pred = model.predict(X_test_scaled)

# Calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

KNN:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Load the dataset using pandas
data = pd.read_csv('your_dataset.csv')  # Replace 'your_dataset.csv' with the path to your dataset

# Assuming your dataset has columns 'X1', 'X2', ..., 'Xn' for features and 'y' for target variable
X = data.drop('y', axis=1)  # Input features (excluding the target variable)
```

```
y = data['y']             # Target variable

# Split the dataset into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize the features using StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Create a K-Nearest Neighbors model with k=5
model = KNeighborsClassifier(n_neighbors=5)

# Fit the model to the training data
model.fit(X_train_scaled, y_train)

# Predict the labels for the test set
y_pred = model.predict(X_test_scaled)

# Calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

HIREARCHIAL CLUSTERING:

```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import AgglomerativeClustering

# Load the dataset using pandas
data = pd.read_csv('your_dataset.csv')  # Replace 'your_dataset.csv' with the path to your dataset
```

```
# Assuming your dataset has columns 'X1', 'X2', ..., 'Xn' for features
X = data[['X1', 'X2', ..., 'Xn']]  # Input features

# Standardize the features using StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Create a Hierarchical clustering model
model = AgglomerativeClustering(n_clusters=3)  # Example: 3 clusters

# Fit the model to the data
model.fit(X_scaled)

# Get cluster labels
labels = model.labels_
```

BOOSTED TREE:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score

# Load the dataset using pandas
data = pd.read_csv('your_dataset.csv')  # Replace 'your_dataset.csv' with the path to your dataset

# Assuming your dataset has columns 'X1', 'X2', ..., 'Xn' for features and 'y' for target variable
X = data[['X1', 'X2', ..., 'Xn']]  # Input features
y = data['y']             # Target variable
```

```
# Split the dataset into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize the features using StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Create a Gradient Boosting Classifier model
model = GradientBoostingClassifier()

# Fit the model to the training data
model.fit(X_train_scaled, y_train)

# Predict the labels for the test set
y_pred = model.predict(X_test_scaled)

# Calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

XGBOOSTER TREE:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score

# Load the dataset using pandas
```

```
data = pd.read_csv('your_dataset.csv')  # Replace 'your_dataset.csv' with the path to your dataset

# Assuming your dataset has columns 'X1', 'X2', ..., 'Xn' for features and 'y' for target variable
X = data[['X1', 'X2', ..., 'Xn']]  # Input features
y = data['y']              # Target variable

# Split the dataset into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize the features using StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Create an XGBoost Classifier model
model = XGBClassifier()

# Fit the model to the training data
model.fit(X_train_scaled, y_train)

# Predict the labels for the test set
y_pred = model.predict(X_test_scaled)

# Calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

BAG OF WORDS DOCUMENT CLSSIFIER[EMAIL SPAM]:

```
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, confusion_matrix

# Load the dataset
data = pd.read_csv('spam_dataset.csv')  # Replace 'spam_dataset.csv' with your dataset file

# Assuming your dataset has columns 'text' for email content and 'label' for spam/ham
X_text = data['text']  # Email content
y = data['label'] # Spam/ham label

# Convert text data into numerical features using bag-of-words approach
vectorizer = CountVectorizer()
X_bow = vectorizer.fit_transform(X_text)

# Split the dataset into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X_bow, y, test_size=0.2, random_state=42)

# Convert to dense format for StandardScaler
X_train_dense = X_train.toarray()
X_test_dense = X_test.toarray()

# Standardize the features using StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_dense)
X_test_scaled = scaler.transform(X_test_dense)

# Train a Naive Bayes classifier
```

```
classifier = MultinomialNB()
classifier.fit(X_train_scaled, y_train)

# Predict on the test set
y_pred = classifier.predict(X_test_scaled)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)
```

ENSEMBLE MODELS[RANDOM FOREST]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score

# Load dataset
df = pd.read_csv('your_dataset.csv')

# Assuming the target variable is in a column named 'target'
X = df.drop(columns=['target'])  # Features
y = df['target']  # Target variable

# Split dataset into training set and test set
```

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

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

# Create Random Forest classifier object
clf = RandomForestClassifier()

# Train Random Forest Classifier
clf = clf.fit(X_train_scaled, y_train)

# Predict the response for test dataset
y_pred = clf.predict(X_test_scaled)

# Model Accuracy
print("Accuracy:", accuracy_score(y_test, y_pred))
```

PCA:

```
# Importing necessary libraries
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

# Load your dataset
df = pd.read_csv('your_dataset.csv')

# Assuming your dataset has features and possibly a target variable
# Separate features from the target variable (if applicable)
X = df.drop(columns=['target_column_name'])  # Adjust 'target_column_name' with your target variable name
y = df['target_column_name']  # Adjust 'target_column_name' with your target variable name if applicable

# Standardize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Perform PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Visualize the PCA results
plt.figure(figsize=(8, 6))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y, cmap='viridis', edgecolor='k')  # Assuming y is available, else remove c=y
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('PCA of Your Dataset with StandardScaler')
plt.colorbar(label='Target')  # Adjust if you have a target variable
plt.show()
```

SVC:

```
# Importing necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
```

```
# Load your dataset
df = pd.read_csv('your_dataset.csv')

# Assuming your dataset has features and a target variable
# Separate features from the target variable
X = df.drop(columns=['target_column_name'])  # Adjust 'target_column_name' with your target variable name
y = df['target_column_name']  # Adjust 'target_column_name' with your target variable name

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Initialize the SVC classifier
svc = SVC()

# Train the SVC classifier
svc.fit(X_train_scaled, y_train)

# Predict on the test set
y_pred = svc.predict(X_test_scaled)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
```

```
print(f"Accuracy: {accuracy}")
```

SVR:

```
# Importing necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error

# Load your dataset
df = pd.read_csv('your_dataset.csv')

# Assuming your dataset has features and a target variable
# Separate features from the target variable
X = df.drop(columns=['target_column_name'])  # Adjust 'target_column_name' with your target variable name
y = df['target_column_name']  # Adjust 'target_column_name' with your target variable name

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Initialize the SVR model
svr = SVR()
```

```
# Train the SVR model
svr.fit(X_train_scaled, y_train)

# Predict on the test set
y_pred = svr.predict(X_test_scaled)

# Calculate Mean Squared Error
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")
```