SLIP 01

1_1.PY

```
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules
from mlxtend.preprocessing import TransactionEncoder

transactions = []
with open(r'/home/pc/smita/MCS-II-ML-2024/Practical_Slip_Solustions/slip_1/groceries.csv', 'r') as f:
    for line in f:
        # Split each line by comma and strip extra spaces
        transaction = [item.strip() for item in line.strip().split(',')]
        transactions.append(transaction)

print("Sample Transactions:")
for transaction in transactions[:5]:
    print(transaction)

TransactionEncoder
te = TransactionEncoder()
te_ary = te.fit(transactions).transform(transactions)
df_onehot = pd.DataFrame(te_ary, columns=te.columns_)

frequent_itemsets = apriori(df_onehot, min_support=0.25, use_colnames=True)
```

```
print(line)
print("Frequent Itemsets with support >= 0.25:")
print(frequent_itemsets)

rules = association_rules(frequent_itemsets, metric="confidence",
min_threshold=0.5)

print("\nAssociation Rules:")
print(rules[['antecedents', 'consequents', 'support', 'confidence', 'lift']])
```

1_2.PY

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.preprocessing import LabelEncoder

iris_data = load_iris()

df = pd.DataFrame(iris_data.data, columns=iris_data.feature_names)

df['species'] = iris_data.target

label_encoder = LabelEncoder()
df['species'] = label_encoder.fit_transform(df['species'])
```

```
print(df.head())


plt.figure(figsize=(8, 6))


sns.scatterplot(x='sepal length (cm)', y='sepal width (cm)', hue='species',
data=df, palette='Set2')


plt.title('Scatter Plot of Sepal Length vs Sepal Width')

plt.xlabel('Sepal Length (cm)')

plt.ylabel('Sepal Width (cm)')


plt.show()
```

SLIP 02

2_1.PY

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt

df = pd.read_csv('house_data.csv')

print("Columns in the dataset:", df.columns)

print("\nFirst few rows of the dataset:")
print(df.head())

print("\nChecking for null values in the dataset:")
print(df.isnull().sum())

df_cleaned = df.dropna()



print("\nNull values after cleaning:")
print(df_cleaned.isnull().sum())

print("\nColumns in the cleaned DataFrame:")
```

```
print(df_cleaned.columns)


X = df_cleaned[['sqft_above']]

y = df_cleaned['price']

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


model = LinearRegression()

model.fit(X_train, y_train)


y_pred = model.predict(X_test)


mse = mean_squared_error(y_test, y_pred)

print("\nMean Squared Error (MSE):", mse)


plt.figure(figsize=(8, 6))

plt.scatter(X_test, y_test, color='blue', label='Actual Prices')

plt.plot(X_test, y_pred, color='red', label='Predicted Prices')

plt.title('Simple Linear Regression - House Price Prediction')

plt.xlabel('sqft_above')

plt.ylabel('price')

plt.legend()

plt.show()
```

2_2.PY

```
import pandas as pd

import numpy as np
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import AgglomerativeClustering
from sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt
import seaborn as sns

Step 1: Load the dataset
df = pd.read_csv('Wholesale_customers_data.csv')

print("\nFirst few rows of the dataset:")
print(df.head())

print("\nChecking for null values:")
print(df.isnull().sum())

df = df.dropna()

scaler = StandardScaler()
df_scaled = scaler.fit_transform(df)

clustering = AgglomerativeClustering(linkage='ward')
labels = clustering.fit_predict(df_scaled)

df['Cluster'] = labels

print("\nClustered Data (with labels):")
```

```
print(df.head())


sil_score = silhouette_score(df_scaled, labels)
print("\nSilhouette Score:", sil_score)



from sklearn.decomposition import PCA
pca = PCA(n_components=2)
pca_components = pca.fit_transform(df_scaled)


plt.figure(figsize=(8, 6))
sns.scatterplot(x=pca_components[:, 0], y=pca_components[:, 1], hue=labels,
palette="viridis", s=100, alpha=0.7)
plt.title('Agglomerative Clustering of Wholesale Customers')
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')
plt.legend(title='Cluster')
plt.show()
```

SLIP 03

3_1.PY

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler

df = pd.read_csv('house_price_data.csv')

print("\nFirst few rows of the dataset:")
print(df.head())

print("\nChecking for null values in the dataset:")
print(df.isnull().sum())

df = df.dropna()  # Alternatively: df.fillna(df.mean(), inplace=True)

print("\nColumns in the dataset:", df.columns)

X = df[['Size', 'Num_Rooms', 'Age']]
y = df['Price']
```

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

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

model = LinearRegression()
model.fit(X_train_scaled, y_train)

y_pred = model.predict(X_test_scaled)

mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("\nMean Squared Error (MSE):", mse)
print("R² Score:", r2)

plt.figure(figsize=(8, 6))
sns.scatterplot(x=y_test, y=y_pred, color='blue')
plt.title('Actual vs Predicted House Prices')
plt.xlabel('Actual Prices')
plt.ylabel('Predicted Prices')
plt.show()
```

```
print("\nModel Coefficients:")
print(f"Intercept: {model.intercept_}")
for feature, coef in zip(X.columns, model.coef_):
    print(f"{feature}: {coef}")
```

3_2.PY

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report
import matplotlib.pyplot as plt

df = pd.read_csv('crash.csv')

print("\nFirst few rows of the dataset:")
print(df.head())

print("\nChecking for missing values:")
print(df.isnull().sum())

df.dropna(inplace=True)

X = df[['Age', 'Speed']]
```

```
y = df['Survived']


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


model = LogisticRegression()


model.fit(X_train, y_train)


y_pred = model.predict(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("\nModel Accuracy:", accuracy)
print("\nConfusion Matrix:")
print(conf_matrix)
print("\nClassification Report:")
print(class_report)


fig, ax = plt.subplots(figsize=(6, 6))
ax.matshow(conf_matrix, cmap='Blues', alpha=0.7)
for i in range(conf_matrix.shape[0]):
    for j in range(conf_matrix.shape[1]):
        ax.text(j, i, str(conf_matrix[i, j]), ha='center', va='center', color='black')
```

```
plt.title('Confusion Matrix')

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.show()


age = 45

speed = 60

new_data = pd.DataFrame([[age, speed]], columns=['Age', 'Speed'])

survival_prob = model.predict_proba(new_data)

print(f"\nSurvival probability for a passenger with Age={age} and
Speed={speed} mph: {survival_prob[0][1]:.2f}")
```

SLIP 04

4_1.PY

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

df = pd.read_csv('mall_customers.csv')

print("\nFirst few rows of the dataset:")
print(df.head())

X = df[['Annual Income (k$)', 'Spending Score (1-100)']]

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

kmeans = KMeans(n_clusters=4, random_state=42, n_init=10)
df['Cluster'] = kmeans.fit_predict(X_scaled)

print("\nClusters assigned to each customer:")
print(df[['CustomerID', 'Cluster']].head())

plt.figure(figsize=(8, 6))
plt.scatter(df['Annual Income (k$)'], df['Spending Score (1-100)'],
c=df['Cluster'], cmap='viridis')
```

```
plt.title('K-Means Clustering of Mall Customers')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.colorbar(label='Cluster')
plt.show()
```

4_2.PY

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt

df = pd.read_csv('house_price.csv')


print("\nFirst few rows of the dataset:")
print(df.head())

df = df.dropna()

X = df[['SquareFootage']]
y = df['Price']
```

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


model = LinearRegression()

model.fit(X_train, y_train)


y_pred = model.predict(X_test)


mse = mean_squared_error(y_test, y_pred)

print("\nMean Squared Error (MSE):", mse)


plt.figure(figsize=(8,6))

plt.scatter(X_test, y_test, color='blue', label='Actual Prices')

plt.plot(X_test, y_pred, color='red', label='Predicted Prices')

plt.title('Simple Linear Regression: House Price Prediction')

plt.xlabel('Square Footage')

plt.ylabel('Price')

plt.legend()

plt.show()
```

SLIP 05

5_1.PY

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt

df = pd.read_csv('fuel_consumption.csv')

print("\nMissing values in the dataset:")
print(df.isnull().sum())

X = df[['EngineSize', 'Cylinders', 'Horsepower', 'Weight', 'Acceleration']]
y = df['FuelConsumption']

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

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

y_pred = model.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
```

```
print("\nMean Squared Error (MSE):", mse)


plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred, color='blue')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', linewidth=2)  # Line of equality
plt.title('Actual vs Predicted Fuel Consumption')
plt.xlabel('Actual Fuel Consumption')
plt.ylabel('Predicted Fuel Consumption')
plt.show()
```

5_2.PY

```
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
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris

iris = load_iris()

df = pd.DataFrame(data=iris.data, columns=iris.feature_names)

df['species'] = iris.target
```

```
X = df.drop('species', axis=1)
y = df['species']

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

knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)

y_pred = knn.predict(X_test)

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

conf_matrix = confusion_matrix(y_test, y_pred)
print("\nConfusion Matrix:")
print(conf_matrix)

print("\nClassification Report:")
print(classification_report(y_test, y_pred))

plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues",
xticklabels=iris.target_names, yticklabels=iris.target_names)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix for KNN Classifier')
```

```
plt.show()
```

SLIP 06

6_1.PY

```
import numpy as np
import pandas as pd
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt

housing = fetch_california_housing()

df = pd.DataFrame(housing.data, columns=housing.feature_names)

df['PRICE'] = housing.target

print(df.head())

X = df.drop('PRICE', axis=1)
y = df['PRICE']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

```
poly = PolynomialFeatures(degree=2)

X_train_poly = poly.fit_transform(X_train)
X_test_poly = poly.transform(X_test)

model = LinearRegression()
model.fit(X_train_poly, y_train)

y_pred = model.predict(X_test_poly)

mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse:.2f}")

plt.scatter(y_test, y_pred)
plt.xlabel("Actual Prices")
plt.ylabel("Predicted Prices")
plt.title("Actual vs Predicted House Prices (Polynomial Regression)")
plt.show()
```

6_2.PY

```
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
import warnings
```

```
warnings.filterwarnings("ignore", category=FutureWarning)

df = pd.read_csv('employees.csv')

print("Checking for missing values in the dataset:")
print(df.isnull().sum())

df_cleaned = df.dropna()
print("\nData after removing rows with missing values:")
print(df_cleaned.isnull().sum())

X = df_cleaned[['Income', 'Age']]

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

kmeans = KMeans(n_clusters=3, n_init=10, random_state=42)
kmeans.fit(X_scaled)

df_cleaned['Cluster'] = kmeans.labels_

print("\nClustered data with labels:")
print(df_cleaned.head())

df_cleaned.to_csv('employees_with_clusters.csv', index=False)
```

```
import matplotlib.pyplot as plt


plt.figure(figsize=(8, 6))
plt.scatter(df_cleaned['Income'], df_cleaned['Age'], c=df_cleaned['Cluster'],
cmap='viridis')
plt.title('K-Means Clustering of Employees')
plt.xlabel('Income')
plt.ylabel('Age')
plt.colorbar(label='Cluster')
plt.show()
```

SLIP 07

7_1.PY

```
import pandas as pd
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt

df = pd.read_csv('Salary_positions.csv')

print(df.isnull().sum())

df = df.dropna()

X = df[['Position_Level']]
y = df['Salary']

model = LinearRegression()

model.fit(X, y)

levels_to_predict = pd.DataFrame([[11], [12]], columns=['Position_Level'])
predicted_salaries = model.predict(levels_to_predict)

for level, salary in zip([11, 12], predicted_salaries):
    print(f"Predicted salary for Level {level}: ${salary:.2f}")

plt.scatter(X, y, color='blue')
```

```
plt.plot(X, model.predict(X), color='red')
plt.title('Salary vs Position Level')
plt.xlabel('Position Level')
plt.ylabel('Salary')
plt.show()
```

7_2.PY

```
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
from sklearn.preprocessing import LabelEncoder


df = pd.read_csv('weather.csv')

label_encoder = LabelEncoder()

df['Outlook'] = label_encoder.fit_transform(df['Outlook'])
df['Temperature'] = label_encoder.fit_transform(df['Temperature'])
df['Humidity'] = label_encoder.fit_transform(df['Humidity'])
df['Wind'] = label_encoder.fit_transform(df['Wind'])
df['PlayTennis'] = label_encoder.fit_transform(df['PlayTennis'])

X = df.drop('PlayTennis', axis=1)
y = df['PlayTennis']
```

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


naive_bayes_model = GaussianNB()


naive_bayes_model.fit(X_train, y_train)


y_pred = naive_bayes_model.predict(X_test)


accuracy = accuracy_score(y_test, y_pred)

conf_matrix = confusion_matrix(y_test, y_pred)


print(f"Accuracy of Naive Bayes Model: {accuracy * 100:.2f}%")

print("Confusion Matrix:")

print(conf_matrix)
```

SLIP 08

8_1.PY

```
from sklearn.datasets import fetch_20newsgroups

from sklearn.model_selection import train_test_split

from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.naive_bayes import MultinomialNB

from sklearn.metrics import classification_report, accuracy_score


newsgroups = fetch_20newsgroups(subset='all')

X, y = newsgroups.data, newsgroups.target


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


vectorizer = TfidfVectorizer(stop_words='english', max_df=0.5)

X_train_tfidf = vectorizer.fit_transform(X_train)

X_test_tfidf = vectorizer.transform(X_test)


model = MultinomialNB()


model.fit(X_train_tfidf, y_train)


y_pred = model.predict(X_test_tfidf)

print("Classification Report:\n", classification_report(y_test, y_pred,
target_names=newsgroups.target_names))
```

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

def categorize_news(text):
    text_tfidf = vectorizer.transform([text])
    category_index = model.predict(text_tfidf)[0]
    return newsgroups.target_names[category_index]

sample_text = "The government has announced new tax reforms for the next fiscal year."
category = categorize_news(sample_text)
print(f"The category of the given text is: {category}")
```

8_2.PY

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier, export_text
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
from sklearn.preprocessing import LabelEncoder

data = {
    'Outlook': ['Sunny', 'Sunny', 'Overcast', 'Rainy', 'Rainy', 'Rainy', 'Overcast', 'Sunny', 'Sunny', 'Rainy', 'Sunny', 'Overcast', 'Overcast', 'Rainy'],
    'Temperature': ['Hot', 'Hot', 'Hot', 'Mild', 'Cool', 'Cool', 'Mild', 'Mild', 'Cool', 'Mild', 'Mild', 'Mild', 'Hot', 'Mild'],
    'Humidity': ['High', 'High', 'High', 'High', 'High', 'Normal', 'Normal', 'High', 'Normal', 'Normal', 'Normal', 'Normal', 'High', 'High'],
```

```
    'Wind': ['Weak', 'Strong', 'Weak', 'Weak', 'Weak', 'Weak', 'Strong', 'Weak', 'Weak', 'Strong', 'Weak', 'Strong', 'Strong', 'Weak'],
    'PlayTennis': ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No']
}

df = pd.DataFrame(data)

label_encoder = LabelEncoder()

df['Outlook'] = label_encoder.fit_transform(df['Outlook'])
df['Temperature'] = label_encoder.fit_transform(df['Temperature'])
df['Humidity'] = label_encoder.fit_transform(df['Humidity'])
df['Wind'] = label_encoder.fit_transform(df['Wind'])
df['PlayTennis'] = label_encoder.fit_transform(df['PlayTennis'])

X = df.drop('PlayTennis', axis=1)
y = df['PlayTennis']

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

dt_classifier = DecisionTreeClassifier(random_state=42)
dt_classifier.fit(X_train, y_train)

y_pred = dt_classifier.predict(X_test)
```

```
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")

print("\nClassification Report:")
print(classification_report(y_test, y_pred))

tree_rules = export_text(dt_classifier, feature_names=list(X.columns))
print("\nDecision Tree Rules:")
print(tree_rules)

new_data = pd.DataFrame({
    'Outlook': label_encoder.transform(['Sunny']),
    'Temperature': label_encoder.transform(['Mild']),
    'Humidity': label_encoder.transform(['High']),
    'Wind': label_encoder.transform(['Weak'])
})

prediction = dt_classifier.predict(new_data)
predicted_class = label_encoder.inverse_transform(prediction)
print(f"\nPredicted class for new sample: {predicted_class[0]}")
```

SLIP 09

9_1.PY

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge, Lasso
from sklearn.metrics import mean_squared_error

df = pd.read_csv('boston_houses.csv')

X = df[['RM']]
y = df['Price']

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

ridge = Ridge(alpha=1.0)
lasso = Lasso(alpha=0.1)

ridge.fit(X_train, y_train)
lasso.fit(X_train, y_train)

ridge_predictions = ridge.predict(X_test)
lasso_predictions = lasso.predict(X_test)

ridge_mse = mean_squared_error(y_test, ridge_predictions)
```

```
lasso_mse = mean_squared_error(y_test, lasso_predictions)

print(f"Ridge Regression Mean Squared Error: {ridge_mse}")
print(f"Lasso Regression Mean Squared Error: {lasso_mse}")

room_count = pd.DataFrame([[5]], columns=['RM'])

ridge_price = ridge.predict(room_count)
lasso_price = lasso.predict(room_count)

print(f"Predicted Price of a house with 5 rooms (Ridge): {ridge_price[0]}")
print(f"Predicted Price of a house with 5 rooms (Lasso): {lasso_price[0]}")
```

9_2.PY

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix

df = pd.read_csv('UniversalBank.csv')

df = df.drop(['ID', 'ZIP Code'], axis=1
```

```
label_encoder = LabelEncoder()
df['Education'] = label_encoder.fit_transform(df['Education'])

X = df.drop('PersonalLoan', axis=1)
y = df['PersonalLoan']

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

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

svm = SVC(kernel='linear', random_state=42)
svm.fit(X_train_scaled, y_train)

y_pred = svm.predict(X_test_scaled)

accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")

print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

SLIP 10

10_1.PY

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import load_iris

iris = load_iris()
X = iris.data  # Features
y = iris.target  # Target (species)
feature_names = iris.feature_names

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

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

pca_df = pd.DataFrame(X_pca, columns=['Principal Component 1', 'Principal Component 2'])
pca_df['Target'] = y

plt.figure(figsize=(8, 6))
```

```
plt.scatter(pca_df['Principal Component 1'], pca_df['Principal Component 2'],
c=pca_df['Target'], cmap='viridis', edgecolor='k', s=50)

plt.title('PCA of Iris Dataset')

plt.xlabel('Principal Component 1')

plt.ylabel('Principal Component 2')

plt.colorbar(label='Target (Species)')

plt.show()


print("Explained variance ratio of the components:")

print(pca.explained_variance_ratio_)


cumulative_variance = np.cumsum(pca.explained_variance_ratio_)

print("\nCumulative explained variance:")

print(cumulative_variance)
```

10_2.PY

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import load_iris

from sklearn.preprocessing import LabelEncoder


iris = load_iris()

X = iris.data

y = iris.target
```

```
label_encoder = LabelEncoder()
y_numeric = label_encoder.fit_transform(y)
df = pd.DataFrame(X, columns=iris.feature_names)
df['Species'] = y_numeric

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

plt.subplot(1, 2, 1)
plt.scatter(df['sepal length (cm)'], df['sepal width (cm)'], c=df['Species'], cmap='viridis')
plt.title('Sepal Length vs Sepal Width')
plt.xlabel('Sepal Length (cm)')
plt.ylabel('Sepal Width (cm)')

plt.subplot(1, 2, 2)
plt.scatter(df['petal length (cm)'], df['petal width (cm)'], c=df['Species'], cmap='viridis')
plt.title('Petal Length vs Petal Width')
plt.xlabel('Petal Length (cm)')
plt.ylabel('Petal Width (cm)')

plt.tight_layout()
plt.show()
```

SLIP 11

11_1.PY

```
import numpy as np
import pandas as pd
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt

housing = fetch_california_housing()

df = pd.DataFrame(housing.data, columns=housing.feature_names)

df['PRICE'] = housing.target

print(df.head())

X = df.drop('PRICE', axis=1)
y = df['PRICE']

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

poly = PolynomialFeatures(degree=2)
```

```
X_train_poly = poly.fit_transform(X_train)
X_test_poly = poly.transform(X_test)

model = LinearRegression()
model.fit(X_train_poly, y_train)

y_pred = model.predict(X_test_poly)

mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse:.2f}")

plt.scatter(y_test, y_pred)
plt.xlabel("Actual Prices")
plt.ylabel("Predicted Prices")
plt.title("Actual vs Predicted House Prices (Polynomial Regression)")
plt.show()
```

11_2.PY

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
```

```
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/00267/data_banknote_authentication.txt"

column_names = ["variance", "skewness", "curtosis", "entropy", "class"]

df = pd.read_csv(url, header=None, names=column_names)


X = df.drop("class", axis=1)

y = df["class"]


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


dt_classifier = DecisionTreeClassifier(random_state=42)

dt_classifier.fit(X_train, y_train)


y_pred = dt_classifier.predict(X_test)


accuracy = accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy * 100:.2f}%")


print("\nClassification Report:")

print(classification_report(y_test, y_pred))


print("\nConfusion Matrix:")

print(confusion_matrix(y_test, y_pred))
```

SLIP 12

12_1.PY

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report

iris = load_iris()
X = iris.data  # Feature matrix
y = iris.target  # Target vector

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

knn = KNeighborsClassifier(n_neighbors=3)

knn.fit(X_train, y_train)

y_pred = knn.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")

print("\nClassification Report:")
print(classification_report(y_test, y_pred, target_names=iris.target_names))
```

12_2

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error

data = pd.read_csv("Salary_positions.csv")

X = data[['Position_Level']].values
y = data['Salary'].values

linear_model = LinearRegression()
linear_model.fit(X, y)

poly_features = PolynomialFeatures(degree=4)
X_poly = poly_features.fit_transform(X)
poly_model = LinearRegression()
poly_model.fit(X_poly, y)

y_pred_linear = linear_model.predict(X)
mse_linear = mean_squared_error(y, y_pred_linear)

y_pred_poly = poly_model.predict(X_poly)
mse_poly = mean_squared_error(y, y_pred_poly)
```

```
print(f"Linear Regression MSE: {mse_linear:.2f}")
print(f"Polynomial Regression (Degree 4) MSE: {mse_poly:.2f}")

plt.scatter(X, y, color="blue", label="Actual Salary Data")
plt.plot(X, y_pred_linear, color="red", label="Simple Linear Regression")
plt.plot(X, y_pred_poly, color="green", label="Polynomial Regression (Degree 4)")
plt.xlabel("Level")
plt.ylabel("Salary")
plt.legend()
plt.show()

level_11 = np.array([[11]])
level_12 = np.array([[12]])

salary_11_linear = linear_model.predict(level_11)[0]
salary_12_linear = linear_model.predict(level_12)[0]

salary_11_poly = poly_model.predict(poly_features.transform(level_11))[0]
salary_12_poly = poly_model.predict(poly_features.transform(level_12))[0]

print(f"\nPredicted Salary for Level 11 (Linear): {salary_11_linear:.2f}")
print(f"Predicted Salary for Level 11 (Polynomial): {salary_11_poly:.2f}")

print(f"Predicted Salary for Level 12 (Linear): {salary_12_linear:.2f}")
print(f"Predicted Salary for Level 12 (Polynomial): {salary_12_poly:.2f}")
```

SLIP 13

13_1

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import yfinance as yf
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense, LSTM
from sklearn.metrics import accuracy_score

ticker = 'GOOGL'
data = yf.download(ticker, start='2015-01-01', end='2023-01-01')

data = data[['Close']]
data = data.dropna()

scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(data)

def create_dataset(dataset, time_step=60):
    X, Y = [], []
    for i in range(len(dataset) - time_step - 1):
        X.append(dataset[i:(i + time_step), 0])
        Y.append(dataset[i + time_step, 0])
```

```
    return np.array(X), np.array(Y)

time_step = 60
X, Y = create_dataset(scaled_data, time_step)

X = X.reshape(X.shape[0], X.shape[1], 1)

train_size = int(len(X) * 0.8)
X_train, X_test = X[:train_size], X[train_size:]
Y_train, Y_test = Y[:train_size], Y[train_size:]

model = Sequential()
model.add(LSTM(units=50, return_sequences=True,
input_shape=(X_train.shape[1], 1)))
model.add(LSTM(units=50, return_sequences=False))
model.add(Dense(units=25))
model.add(Dense(units=1))

model.compile(optimizer='adam', loss='mean_squared_error')
model.fit(X_train, Y_train, batch_size=64, epochs=20)

predictions = model.predict(X_test)
predictions = scaler.inverse_transform(predictions.reshape(-1, 1))

Y_test_actual = scaler.inverse_transform(Y_test.reshape(-1, 1))

pred_trend = np.where(predictions[1:] > predictions[:-1], 1, 0)
```

```
actual_trend = np.where(Y_test_actual[1:] > Y_test_actual[:-1], 1, 0)

accuracy = accuracy_score(actual_trend, pred_trend) * 100
print(f'Trend Prediction Accuracy: {accuracy:.2f}%')

plt.figure(figsize=(14, 7))
plt.plot(Y_test_actual, color='blue', label='Actual Google Stock Price')
plt.plot(predictions, color='red', label='Predicted Google Stock Price')
plt.title('Google Stock Price Prediction')
plt.xlabel('Time')
plt.ylabel('Stock Price')
plt.legend()
plt.show()
```

13_2.PY

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt

df = pd.read_csv('house_price.csv')

print("\nFirst few rows of the dataset:")
```

```
print(df.head())


df = df.dropna()


X = df[['SquareFootage']]
y = df['Price']


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


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


y_pred = model.predict(X_test)


mse = mean_squared_error(y_test, y_pred)
print("\nMean Squared Error (MSE):", mse)
plt.figure(figsize=(8,6))
plt.scatter(X_test, y_test, color='blue', label='Actual Prices')  # Actual data
points
plt.plot(X_test, y_pred, color='red', label='Predicted Prices')  # Model
predictions
plt.title('Simple Linear Regression: House Price Prediction')
plt.xlabel('Square Footage')
plt.ylabel('Price')
plt.legend()
plt.show()
```

SLIP 14

14_1.PY

```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.preprocessing import image

(x_train, y_train), (x_test, y_test) = mnist.load_data()

x_train = x_train.reshape(-1, 28, 28, 1).astype('float32') / 255
x_test = x_test.reshape(-1, 28, 28, 1).astype('float32') / 255

y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    Flatten(),
    Dense(128, activation='relu'),
```

```
    Dense(10, activation='softmax')
])

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

history = model.fit(x_train, y_train, epochs=5, batch_size=64,
validation_split=0.2)

test_loss, test_acc = model.evaluate(x_test, y_test)
print(f'Test accuracy: {test_acc:.4f}')

plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()

def predict_digit(image_path):
    # Load and preprocess the image
    img = image.load_img(image_path, target_size=(28, 28),
color_mode='grayscale')
    img_array = image.img_to_array(img) / 255.0
    img_array = np.expand_dims(img_array, axis=0)  # Add batch dimension
```

```
    predictions = model.predict(img_array)
    predicted_digit = np.argmax(predictions[0])
    return predicted_digit

image_path = 'path_to_your_digit_image.png'  # Update this path
predicted_digit = predict_digit(image_path)
print(f'The digit in the image is: {predicted_digit}')
```

14_2.PY

```
import pandas as pd
import numpy as np

data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eve', np.nan],
    'Age': [24, np.nan, 22, 32, 29, 27],
    'City': ['New York', 'Los Angeles', 'Chicago', np.nan, 'Houston', 'Phoenix'],
    'Salary': [70000, 80000, np.nan, 54000, 62000, 67000]
}

df = pd.DataFrame(data)
print("Original Dataset with Null Values:")
print(df)

print("\nNull Values in Each Column:")
print(df.isnull().sum())
df_cleaned = df.dropna()
```

```
print("\nDataset after Removing Rows with Null Values:")
print(df_cleaned)
```

SLIP 15

15_1.PY

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import fetch_california_housing
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.metrics import accuracy_score

data = fetch_california_housing()

X = pd.DataFrame(data.data, columns=data.feature_names)
y = data.target

average_price = y.mean()

y_binary = (y > average_price).astype(int)

X_train, X_test, y_train, y_test = train_test_split(X, y_binary,test_size=0.2,
random_state=42)
scaler = StandardScaler()
```

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

model = Sequential()

model.add(Dense(16, input_dim=X_train_scaled.shape[1], activation='relu'))

model.add(Dense(8, activation='relu'))

model.add(Dense(1, activation='sigmoid'))

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

model.fit(X_train_scaled, y_train, epochs=50, batch_size=32, verbose=1,
        validation_split=0.2)

loss, accuracy = model.evaluate(X_test_scaled, y_test)
print(f"Test Accuracy: {accuracy:.4f}")

y_pred = (model.predict(X_test_scaled) > 0.5).astype("int32")

accuracy_test = accuracy_score(y_test, y_pred)
print(f"Accuracy on Test Set: {accuracy_test:.4f}")
```

15_2.PY

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

data = {
    'Size': [1500, 2000, 2500, 1800, 2200, 1700],
    'Bedrooms': [3, 4, 3, 2, 4, 3],
    'Age': [20, 15, 10, 30, 8, 12],
    'Price': [300000, 400000, 450000, 350000, 500000, 320000]
}

df = pd.DataFrame(data)

X = df[['Size', 'Bedrooms', 'Age']]
y = df['Price']

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

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

y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
```

```
r2 = r2_score(y_test, y_pred)

print("\nModel Evaluation:")
print(f"Mean Squared Error: {mse:.2f}")
print(f"R-squared: {r2:.2f}")

print("\nCoefficients:", model.coef_)
print("Intercept:", model.intercept_)

new_house = pd.DataFrame({'Size': [2000], 'Bedrooms': [3], 'Age': [10]})
predicted_price = model.predict(new_house)
print(f"\nPredicted Price for new house (Size=2000, Bedrooms=3, Age=10): ${predicted_price[0]:,.2f}")
```

SLIP 16

16_1.PY

```
import numpy as np
import pandas as pd
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score

data = pd.read_csv("https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv",
                   names=["Pregnancies", "Glucose", "BloodPressure", "SkinThickness",
                          "Insulin", "BMI", "DiabetesPedigreeFunction", "Age",
"Outcome"])

X = data.drop('Outcome', axis=1)
y = data['Outcome']

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

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,
random_state=42)
model = Sequential()
```

```
model.add(Dense(16, input_dim=X_train.shape[1], activation='relu'))

model.add(Dense(1, activation='sigmoid'

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

history = model.fit(X_train, y_train, epochs=50, batch_size=10, verbose=1,
validation_data=(X_test, y_test))

loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {accuracy:.4f}")

y_pred = (model.predict(X_test) > 0.5).astype("int32")

print(f"Accuracy Score: {accuracy_score(y_test, y_pred):.4f}")
```

16_2.PY

```
import numpy as np
import pandas as pd
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt
housing = fetch_california_housing()
```

```
df = pd.DataFrame(housing.data, columns=housing.feature_names)
df['MedHouseVal'] = housing.target  # Target column (similar to 'PRICE')

X = df[['AveRooms']]
y = df['MedHouseVal']

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

poly = PolynomialFeatures(degree=2)
X_train_poly = poly.fit_transform(X_train)
X_test_poly = poly.transform(X_test)

model = LinearRegression()
model.fit(X_train_poly, y_train)

y_pred = model.predict(X_test_poly)

mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Polynomial Regression Model Evaluation:")
print(f"Mean Squared Error: {mse:.2f}")
print(f"R-squared: {r2:.2f}")

plt.scatter(X, y, color='blue', label='Data Points')
```

```
X_fit = pd.DataFrame(np.linspace(X.min(), X.max(), 100),
columns=['AveRooms'])

X_fit_poly = poly.transform(X_fit)

y_fit = model.predict(X_fit_poly)

plt.plot(X_fit, y_fit, color='red', label='Polynomial Fit (Degree 2)')

plt.xlabel("Average number of rooms per household (AveRooms)")

plt.ylabel("Median House Value (MedHouseVal)")

plt.title("Polynomial Regression Fit for California Housing Data")

plt.legend()

plt.show()
```

SLIP 17

17_1.PY

```
import pandas as pd

import numpy as np

from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier,
GradientBoostingClassifier, VotingClassifier, StackingClassifier

from sklearn.linear_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn.svm import SVC

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report

from sklearn.datasets import load_diabetes


data =
pd.read_csv("https://raw.githubusercontent.com/jbrownlee/Datasets/master/
pima-indians-diabetes.data.csv",
                names=["Pregnancies", "Glucose", "BloodPressure", "SkinThickness",
                       "Insulin", "BMI", "DiabetesPedigreeFunction", "Age",
"Outcome"])


X = data.drop('Outcome', axis=1)

y = data['Outcome']


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


random_forest = RandomForestClassifier(n_estimators=100, random_state=42)
```

```
random_forest.fit(X_train, y_train)
y_pred_rf = random_forest.predict(X_test)
accuracy_rf = accuracy_score(y_test, y_pred_rf)

ada_boost = AdaBoostClassifier(n_estimators=100, random_state=42)
ada_boost.fit(X_train, y_train)
y_pred_ada = ada_boost.predict(X_test)
accuracy_ada = accuracy_score(y_test, y_pred_ada)

gradient_boost = GradientBoostingClassifier(n_estimators=100,
random_state=42)
gradient_boost.fit(X_train, y_train)
y_pred_gb = gradient_boost.predict(X_test)
accuracy_gb = accuracy_score(y_test, y_pred_gb)

log_clf = LogisticRegression(solver='lbfgs', max_iter=1000)
svc_clf = SVC(probability=True, random_state=42)

voting_clf = VotingClassifier(estimators=[
    ('lr', log_clf),
    ('rf', random_forest),
    ('svc', svc_clf)], voting='soft')

voting_clf.fit(X_train, y_train)
y_pred_voting = voting_clf.predict(X_test)
accuracy_voting = accuracy_score(y_test, y_pred_voting)
```

```
stacking_clf = StackingClassifier(estimators=[
    ('lr', log_clf),
    ('dt', DecisionTreeClassifier(random_state=42)),
    ('svc', svc_clf)],
    final_estimator=RandomForestClassifier(random_state=42)
)

stacking_clf.fit(X_train, y_train)
y_pred_stacking = stacking_clf.predict(X_test)
accuracy_stacking = accuracy_score(y_test, y_pred_stacking)

print("Accuracy Results:")
print(f"Random Forest (Bagging): {accuracy_rf:.4f}")
print(f"AdaBoost (Boosting): {accuracy_ada:.4f}")
print(f"GradientBoosting (Boosting): {accuracy_gb:.4f}")
print(f"Voting Classifier: {accuracy_voting:.4f}")
print(f"Stacking Classifier: {accuracy_stacking:.4f}")

print("\nConfusion Matrix for Stacking Classifier:")
print(confusion_matrix(y_test, y_pred_stacking))

print("\nClassification Report for Stacking Classifier:")
print(classification_report(y_test, y_pred_stacking))
```

17_2.PY

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler

df = pd.read_csv('house_price_data.csv')

print("\nFirst few rows of the dataset:")
print(df.head())

print("\nChecking for null values in the dataset:")
print(df.isnull().sum())

df = df.dropna()

print("\nColumns in the dataset:", df.columns)

X = df[['Size', 'Num_Rooms', 'Age']]
y = df['Price']
```

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

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

model = LinearRegression()
model.fit(X_train_scaled, y_train)  # Using scaled features

y_pred = model.predict(X_test_scaled)

mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("\nMean Squared Error (MSE):", mse)
print("R² Score:", r2)

plt.figure(figsize=(8, 6))
sns.scatterplot(x=y_test, y=y_pred, color='blue')
plt.title('Actual vs Predicted House Prices')
plt.xlabel('Actual Prices')
plt.ylabel('Predicted Prices')
plt.show()

print("\nModel Coefficients:")
print(f"Intercept: {model.intercept_}")
```

```
for feature, coef in zip(X.columns, model.coef_):
    print(f"{feature}: {coef}")
```

SLIP 18

18_1.PY

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

url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"
columns = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI',
           'DiabetesPedigreeFunction', 'Age', 'Outcome']

df = pd.read_csv(url, names=columns)

df.fillna(df.mean(), inplace=True)

scaler = StandardScaler()
X = df.drop('Outcome', axis=1)

X_scaled = scaler.fit_transform(X)
```

```
kmeans = KMeans(n_clusters=2, n_init=10, random_state=42)
df['Cluster'] = kmeans.fit_predict(X_scaled)

print("\nCluster vs Outcome Distribution:")
print(df[['Cluster', 'Outcome']].value_counts())

plt.figure(figsize=(8, 6))
plt.scatter(df['Glucose'], df['BMI'], c=df['Cluster'], cmap='viridis', marker='o')
plt.title("K-Means Clustering on Diabetes Dataset")
plt.xlabel("Glucose")
plt.ylabel("BMI")
plt.colorbar(label='Cluster')
plt.show()
print("\nCluster Centers (Centroids):")
print(kmeans.cluster_centers_)
```

18_2.PY

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures

df = pd.read_csv("Salary_positions.csv")
X = df[['Position_Level']].values
```

```
y = df['Salary'].values

poly = PolynomialFeatures(degree=4)

X_poly = poly.fit_transform(X)

poly_regressor = LinearRegression()
poly_regressor.fit(X_poly, y)

y_pred = poly_regressor.predict(X_poly)



plt.scatter(X, y, color='red')
plt.plot(X, y_pred, color='blue')
plt.title("Polynomial Linear Regression for Salary Prediction")
plt.xlabel("Position Level")
plt.ylabel("Salary")
plt.show()

level_11 = poly.transform([[11]])
level_12 = poly.transform([[12]])

predicted_salary_11 = poly_regressor.predict(level_11)
predicted_salary_12 = poly_regressor.predict(level_12)

print(f"Predicted salary for Level 11: ${predicted_salary_11[0]:,.2f}")
```

```
print(f"Predicted salary for Level 12: ${predicted_salary_12[0]:,.2f}")
```

SLIP 19

19_1.PY

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report

iris = load_iris()
X = iris.data  # Feature matrix
y = iris.target  # Target vector

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

knn = KNeighborsClassifier(n_neighbors=3)

knn.fit(X_train, y_train)

y_pred = knn.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")
```

```
print("\nClassification Report:")
print(classification_report(y_test, y_pred, target_names=iris.target_names))
```

19_2.PY

```
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
from sklearn.preprocessing import LabelEncoder

df = pd.read_csv('weather.csv')

label_encoder = LabelEncoder()

df['Outlook'] = label_encoder.fit_transform(df['Outlook'])
df['Temperature'] = label_encoder.fit_transform(df['Temperature'])
df['Humidity'] = label_encoder.fit_transform(df['Humidity'])
df['Wind'] = label_encoder.fit_transform(df['Wind'])
df['PlayTennis'] = label_encoder.fit_transform(df['PlayTennis'])

X = df.drop('PlayTennis', axis=1)
y = df['PlayTennis']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)
```

```
naive_bayes_model = GaussianNB()

naive_bayes_model.fit(X_train, y_train)

y_pred = naive_bayes_model.predict(X_test)



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

print(f"Accuracy of Naive Bayes Model: {accuracy * 100:.2f}%")
print("Confusion Matrix:")
print(conf_matrix)
```

SLIP 20

20_1.PY

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge, Lasso
from sklearn.metrics import mean_squared_error

df = pd.read_csv('boston_houses.csv')
X = df[['RM']]
y = df['Price']
```

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

ridge = Ridge(alpha=1.0)
lasso = Lasso(alpha=0.1)

ridge.fit(X_train, y_train)
lasso.fit(X_train, y_train)



ridge_predictions = ridge.predict(X_test)
lasso_predictions = lasso.predict(X_test)

ridge_mse = mean_squared_error(y_test, ridge_predictions)
lasso_mse = mean_squared_error(y_test, lasso_predictions)

print(f"Ridge Regression Mean Squared Error: {ridge_mse}")
print(f"Lasso Regression Mean Squared Error: {lasso_mse}")

room_count = pd.DataFrame([[5]], columns=['RM'])

ridge_price = ridge.predict(room_count)
lasso_price = lasso.predict(room_count)

print(f"Predicted Price of a house with 5 rooms (Ridge): {ridge_price[0]}")
print(f"Predicted Price of a house with 5 rooms (Lasso): {lasso_price[0]}")
```

20_2.PY

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier, export_text
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
from sklearn.preprocessing import LabelEncoder

data = {
    'Outlook': ['Sunny', 'Sunny', 'Overcast', 'Rainy', 'Rainy', 'Rainy', 'Overcast', 'Sunny', 'Sunny', 'Rainy', 'Sunny', 'Overcast', 'Overcast', 'Rainy'],
    'Temperature': ['Hot', 'Hot', 'Hot', 'Mild', 'Cool', 'Cool', 'Mild', 'Mild', 'Cool', 'Mild', 'Mild', 'Mild', 'Hot', 'Mild'],
    'Humidity': ['High', 'High', 'High', 'High', 'High', 'Normal', 'Normal', 'High', 'Normal', 'Normal', 'Normal', 'Normal', 'High', 'High'],
    'Wind': ['Weak', 'Strong', 'Weak', 'Weak', 'Weak', 'Weak', 'Strong', 'Weak', 'Weak', 'Strong', 'Weak', 'Strong', 'Strong', 'Weak'],
    'PlayTennis': ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No']
}

df = pd.DataFrame(data)

label_encoder = LabelEncoder()


df['Outlook'] = label_encoder.fit_transform(df['Outlook'])
df['Temperature'] = label_encoder.fit_transform(df['Temperature'])
```

```
df['Humidity'] = label_encoder.fit_transform(df['Humidity'])
df['Wind'] = label_encoder.fit_transform(df['Wind'])
df['PlayTennis'] = label_encoder.fit_transform(df['PlayTennis'])

X = df.drop('PlayTennis', axis=1)  # Features (all columns except 'PlayTennis')
y = df['PlayTennis']  # Target (the 'PlayTennis' column)

# Step 4: Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

dt_classifier = DecisionTreeClassifier(random_state=42)
dt_classifier.fit(X_train, y_train)

y_pred = dt_classifier.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")

print("\nClassification Report:")
print(classification_report(y_test, y_pred))

tree_rules = export_text(dt_classifier, feature_names=list(X.columns))
print("\nDecision Tree Rules:")
print(tree_rules)

new_data = pd.DataFrame({
```

```
    'Outlook': label_encoder.transform(['Sunny']),
    'Temperature': label_encoder.transform(['Mild']),
    'Humidity': label_encoder.transform(['High']),
    'Wind': label_encoder.transform(['Weak'])
})
prediction = dt_classifier.predict(new_data)
predicted_class = label_encoder.inverse_transform(prediction)
print(f"\nPredicted class for new sample: {predicted_class[0]}")
```

SLIP 21

21_1.PY

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
df = pd.read_csv('HousePrice.csv')

print("First few rows of the dataset:")
print(df.head())

X = df[['bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors',
       'waterfront', 'view', 'condition', 'sqft_above', 'sqft_basement',
       'yr_built', 'yr_renovated']]
y = df['price']
```

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


model = LinearRegression()


model.fit(X_train, y_train)


y_pred = model.predict(X_test)

mse = mean_squared_error(y_test, y_pred)

r2 = r2_score(y_test, y_pred)


print("\nModel Evaluation:")

print(f"Mean Squared Error (MSE): {mse}")

print(f"R-squared (R2): {r2}")


comparison_df = pd.DataFrame({'Actual Price': y_test, 'Predicted Price':
y_pred})

print("\nActual vs Predicted Prices:")

print(comparison_df.head())
```

21_2.PY

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error, r2_score

data = pd.read_csv("1Salary_positions.csv")

X = data[['Level']].values
y = data['Salary'].values

lin_reg = LinearRegression()
lin_reg.fit(X, y)

poly_features = PolynomialFeatures(degree=4)
X_poly = poly_features.fit_transform(X)
poly_reg = LinearRegression()
poly_reg.fit(X_poly, y)

y_pred_linear = lin_reg.predict(X)
y_pred_poly = poly_reg.predict(X_poly)

r2_linear = r2_score(y, y_pred_linear)
mse_linear = mean_squared_error(y, y_pred_linear)
```

```
r2_poly = r2_score(y, y_pred_poly)
mse_poly = mean_squared_error(y, y_pred_poly)

print("Simple Linear Regression R²:", r2_linear)
print("Simple Linear Regression MSE:", mse_linear)
print("Polynomial Regression R²:", r2_poly)
print("Polynomial Regression MSE:", mse_poly)

level_11 = np.array([[11]])
level_12 = np.array([[12]])

salary_11_linear = lin_reg.predict(level_11)
salary_12_linear = lin_reg.predict(level_12)

salary_11_poly = poly_reg.predict(poly_features.transform(level_11))
salary_12_poly = poly_reg.predict(poly_features.transform(level_12))

print(f"Predicted salary for level 11 (Simple Linear Regression):
{salary_11_linear[0]}")
print(f"Predicted salary for level 12 (Simple Linear Regression):
{salary_12_linear[0]}")
print(f"Predicted salary for level 11 (Polynomial Regression):
{salary_11_poly[0]}")
print(f"Predicted salary for level 12 (Polynomial Regression):
{salary_12_poly[0]}")

plt.scatter(X, y, color='red', label='Actual Salaries')
```

```
plt.plot(X, y_pred_linear, color='blue', label='Simple Linear Regression')
plt.plot(X, y_pred_poly, color='green', label='Polynomial Regression')
plt.xlabel("Employee Level")
plt.ylabel("Salary")
plt.legend()
plt.title("Simple Linear vs Polynomial Regression")
plt.show()
```

SLIP 22

22_1.PY

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt

df = pd.read_csv('house_price.csv')

print("\nFirst few rows of the dataset:")
print(df.head())

df = df.dropna()

X = df[['SquareFootage']]
y = df['Price']
```

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


model = LinearRegression()

model.fit(X_train, y_train)


y_pred = model.predict(X_test)


mse = mean_squared_error(y_test, y_pred)

print("\nMean Squared Error (MSE):", mse)


plt.figure(figsize=(8,6))

plt.scatter(X_test, y_test, color='blue', label='Actual Prices')

plt.plot(X_test, y_pred, color='red', label='Predicted Prices')

plt.title('Simple Linear Regression: House Price Prediction')

plt.xlabel('Square Footage')

plt.ylabel('Price')

plt.legend()

plt.show()
```

22_2.PY

```
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules
from mlxtend.preprocessing import TransactionEncoder

transactions = []
with open('groceries.csv') as f:
    for line in f:

        transaction = [item.strip() for item in line.strip().split(',')]
        transactions.append(transaction)

print("Sample Transactions:")
for transaction in transactions[:5]:
    print(transaction)

te = TransactionEncoder()
te_ary = te.fit(transactions).transform(transactions)
df_onehot = pd.DataFrame(te_ary, columns=te.columns_)

frequent_itemsets = apriori(df_onehot, min_support=0.25,
use_colnames=True)

print(line)
print("Frequent Itemsets with support >= 0.25:")
print(frequent_itemsets)
```

```
rules = association_rules(frequent_itemsets, metric="confidence",
min_threshold=0.5)


print("\nAssociation Rules:")
print(rules[['antecedents', 'consequents', 'support', 'confidence', 'lift']])
```

SLIP 23

23_1.PY

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report

iris = load_iris()
X = iris.data
y = iris.target

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

knn = KNeighborsClassifier(n_neighbors=3)

knn.fit(X_train, y_train)
```

```
y_pred = knn.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")

print("\nClassification Report:")
print(classification_report(y_test, y_pred, target_names=iris.target_names))
```

23_2_1.PY

```
import pandas as pd
import numpy as np

data = {
    'EmployeeID': [1, 2, 3, 4, 5],
    'Name': ['Alice', 'Bob', np.nan, 'David', 'Eva'],
    'Age': [25, np.nan, 29, 40, 35],
    'Department': ['HR', 'Finance', 'IT', np.nan, 'Marketing'],
    'Salary': [50000, 60000, 55000, 65000, np.nan]
}

df = pd.DataFrame(data)

df.to_csv("your_dataset.csv", index=False)

print("Sample CSV file 'your_dataset.csv' created with dummy data and null values.")
```

23_2.PY

```
import pandas as pd

data = pd.read_csv("1your_dataset.csv")

print("Null values in each column before removing:")
print(data.isnull().sum())

data_cleaned = data.dropna()

print("\nNull values in each column after removing:")
print(data_cleaned.isnull().sum())

print(f"\nNumber of rows before removing nulls: {len(data)}")
print(f"Number of rows after removing nulls: {len(data_cleaned)}")

data_cleaned.to_csv("cleaned_dataset.csv", index=True)
```

SLIP 24

24_1.PY

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix

from sklearn.tree import plot_tree

import matplotlib.pyplot as plt


url = "https://archive.ics.uci.edu/ml/machine-learning-
databases/00267/data_banknote_authentication.txt"

column_names = ['Variance', 'Skewness', 'Curtosis', 'Entropy', 'Class']

data = pd.read_csv(url, header=None, names=column_names)


print("First few rows of the dataset:")

print(data.head())



X = data.drop('Class', axis=1)

y = data['Class']


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


clf = DecisionTreeClassifier(random_state=42)

clf.fit(X_train, y_train)
```

```
y_pred = clf.predict(X_test)


accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy of the Decision Tree classifier: {accuracy:.2f}")
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))


plt.figure(figsize=(15, 10))
plot_tree(clf, feature_names=X.columns, class_names=['Forged', 'Genuine'], filled=True)
plt.title("Decision Tree for Banknote Authentication")
plt.show()
```

24_2.PY

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix


df = pd.read_csv('UniversalBank.csv')


df = df.drop(['ID', 'ZIP Code'], axis=1)
```

```
label_encoder = LabelEncoder()
df['Education'] = label_encoder.fit_transform(df['Education'])

X = df.drop('PersonalLoan', axis=1)
y = df['PersonalLoan']

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

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

svm = SVC(kernel='linear', random_state=42)
svm.fit(X_train_scaled, y_train)

y_pred = svm.predict(X_test_scaled)



accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")

print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("\nClassification Report:")
```

```
print(classification_report(y_test, y_pred))
```

SLIP 25

25_1.PY

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt

np.random.seed(0)
house_size = 2.5 * np.random.rand(100, 1) + 0.5
house_price = 50 + 20 * house_size + 10 * (house_size ** 2) +
np.random.randn(100, 1) * 5

data = pd.DataFrame({'Size': house_size.flatten(), 'Price': house_price.flatten()})

X = data[['Size']]
y = data['Price']

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

poly_features = PolynomialFeatures(degree=2)
```

```
X_train_poly = poly_features.fit_transform(X_train)
X_test_poly = poly_features.transform(X_test)

poly_reg_model = LinearRegression()
poly_reg_model.fit(X_train_poly, y_train)

y_pred = poly_reg_model.predict(X_test_poly)

mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f"Mean Squared Error: {mse:.2f}")
print(f"R-squared Score: {r2:.2f}")

X_sorted = np.sort(X.values, axis=0)
X_poly_sorted = poly_features.transform(X_sorted)
y_poly_pred = poly_reg_model.predict(X_poly_sorted)

plt.scatter(X, y, color='blue', label='Actual Data')
plt.plot(X_sorted, y_poly_pred, color='red', label='Polynomial Regression Fit')
plt.xlabel("House Size (units)")
plt.ylabel("House Price")
plt.title("Polynomial Regression for House Price Prediction")
plt.legend()
plt.show()
```

25_2.PY

```
import numpy as np
import pandas as pd
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score

data =
pd.read_csv("https://raw.githubusercontent.com/jbrownlee/Datasets/master/
pima-indians-diabetes.data.csv",
            names=["Pregnancies", "Glucose", "BloodPressure", "SkinThickness",
                "Insulin", "BMI", "DiabetesPedigreeFunction", "Age",
"Outcome"])

X = data.drop('Outcome', axis=1)
y = data['Outcome']

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

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,
random_state=42)
model = Sequential()
```

```
model.add(Dense(16, input_dim=X_train.shape[1], activation='relu'))

model.add(Dense(1, activation='sigmoid'))

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

history = model.fit(X_train, y_train, epochs=50, batch_size=10, verbose=1,
validation_data=(X_test, y_test))

loss, accuracy = model.evaluate(X_test, y_test)

print(f"Test Accuracy: {accuracy:.4f}")

y_pred = (model.predict(X_test) > 0.5).astype("int32")

print(f"Accuracy Score: {accuracy_score(y_test, y_pred):.4f}")
```

SLIP 26

26_1.PY

```
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules
from mlxtend.preprocessing import TransactionEncoder

transactions = []
with open('groceries.csv') as f:
    for line in f:
        # Split each line by comma and strip extra spaces
        transaction = [item.strip() for item in line.strip().split(',')]
        transactions.append(transaction)

print("Sample Transactions:")
for transaction in transactions[:5
    print(transaction)
te = TransactionEncoder()
te_ary = te.fit(transactions).transform(transactions)
df_onehot = pd.DataFrame(te_ary, columns=te.columns_)

frequent_itemsets = apriori(df_onehot, min_support=0.25,
use_colnames=True)

print(line)
print("Frequent Itemsets with support >= 0.25:")
print(frequent_itemsets)
```

```
rules = association_rules(frequent_itemsets, metric="confidence",
min_threshold=0.5)


print("\nAssociation Rules:")
print(rules[['antecedents', 'consequents', 'support', 'confidence', 'lift']])
```

26_2.PY

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


sns.set()


data = pd.read_csv('diabetes.csv')


print(data.head())


data.info()
```

```
columns_with_zeros = ['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
'BMI']
data[columns_with_zeros] = data[columns_with_zeros].replace(0, np.NaN)

data['Glucose'].fillna(data['Glucose'].mean(), inplace=True)
data['BloodPressure'].fillna(data['BloodPressure'].mean(), inplace=True)
data['SkinThickness'].fillna(data['SkinThickness'].median(), inplace=True)
data['Insulin'].fillna(data['Insulin'].median(), inplace=True)
data['BMI'].fillna(data['BMI'].median(), inplace=True)

X = data.drop("Outcome", axis=1)
y = data["Outcome"]

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

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y,
                                    test_size=0.33, random_state=42,
                                    stratify=y)

test_scores = []
train_scores = []

for k in range(1, 21):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
```

```
    train_scores.append(knn.score(X_train, y_train))
    test_scores.append(knn.score(X_test, y_test))

plt.figure(figsize=(12, 5))
plt.plot(range(1, 21), train_scores, marker='*', label='Train Score')
plt.plot(range(1, 21), test_scores, marker='o', label='Test Score')
plt.xlabel('Number of Neighbors (k)')
plt.ylabel('Accuracy')
plt.title('Train and Test Scores for different k values')
plt.legend()
plt.show()

best_k = test_scores.index(max(test_scores)) + 1
print(f'Optimal value of K: {best_k}')

knn_best = KNeighborsClassifier(n_neighbors=best_k)
knn_best.fit(X_train, y_train)

y_pred = knn_best.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f'KNN model accuracy on test data: {accuracy*100:.2f}%')

cnf_matrix = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(8, 6))
```

```
sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, cmap="YlGnBu", fmt='g',
          xticklabels=['No Diabetes', 'Diabetes'],
          yticklabels=['No Diabetes', 'Diabetes'])
plt.title('Confusion Matrix')
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
plt.show()


def predict_diabetes(patient_data):
   if len(patient_data) != X.shape[1]:
      raise ValueError(f'Expected {X.shape[1]} features, but got
{len(patient_data)} features.')


   patient_data_df = pd.DataFrame([patient_data], columns=X.columns)


   patient_data_scaled = scaler.transform(patient_data_df)


   prediction = knn_best.predict(patient_data_scaled)


   return 'Diabetic' if prediction[0] == 1 else 'Not Diabetic'
new_patient_data = [120, 70, 30, 200, 30.0, 25, 1, 0]
print(f'Prediction for the new patient: {predict_diabetes(new_patient_data)}')
```

SLIP 27

27_1.PY

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
df = pd.read_csv('HousePrice.csv')

print("First few rows of the dataset:")
print(df.head())

X = df[['bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors',
       'waterfront', 'view', 'condition', 'sqft_above', 'sqft_basement',
       'yr_built', 'yr_renovated']]
y = df['price']

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

model = LinearRegression()

model.fit(X_train, y_train)

y_pred = model.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
```

```
r2 = r2_score(y_test, y_pred)


print("\nModel Evaluation:")
print(f"Mean Squared Error (MSE): {mse}")
print(f"R-squared (R2): {r2}")


comparison_df = pd.DataFrame({'Actual Price': y_test, 'Predicted Price':
y_pred})
print("\nActual vs Predicted Prices:")
print(comparison_df.head())
```

27_2.PY

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report


iris = load_iris()
X = iris.data
y = iris.target


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


knn = KNeighborsClassifier(n_neighbors=3)
```

```
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")
print("\nClassification Report:")
print(classification_report(y_test, y_pred, target_names=iris.target_names))
```

SLIP 28

28_1.PY

```
from sklearn.datasets import fetch_20newsgroups
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report, accuracy_score

newsgroups = fetch_20newsgroups(subset='all')
X, y = newsgroups.data, newsgroups.target

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

vectorizer = TfidfVectorizer(stop_words='english', max_df=0.5)
X_train_tfidf = vectorizer.fit_transform(X_train)
X_test_tfidf = vectorizer.transform(X_test)
```

```
model = MultinomialNB()

model.fit(X_train_tfidf, y_train)

y_pred = model.predict(X_test_tfidf)



print("Classification Report:\n", classification_report(y_test, y_pred,
target_names=newsgroups.target_names))

accuracy = accuracy_score(y_test, y_pred)

print(f"Model Accuracy: {accuracy:.2f}")

def categorize_news(text):

    text_tfidf = vectorizer.transform([text])

    category_index = model.predict(text_tfidf)[0]

    return newsgroups.target_names[category_index]

sample_text = "The government has announced new tax reforms for the next
fiscal year."

category = categorize_news(sample_text)

print(f"The category of the given text is: {category}")
```

28_2.PY

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
iris = load_iris()
X = iris.data
y = iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)
svm_kernels = {
        'linear': SVC(kernel='linear'),
        'poly': SVC(kernel='poly', degree=3),
        'rbf': SVC(kernel='rbf')
        }
accuracy_results = {}
for kernel, model in svm_kernels.items():
  model.fit(X_train, y_train)
  y_pred = model.predict(X_test)
  accuracy = accuracy_score(y_test, y_pred)
  accuracy_results[kernel] = accuracy

for kernel, accuracy in accuracy_results.items():
  print(f'Accuracy of SVM with {kernel} kernel: {accuracy:.4f}')
```

SLIP 29

29_1.PY

```
from sklearn import datasets
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

iris = datasets.load_iris()
X = iris.data
y = iris.target

pca = PCA(n_components=2)
X_reduced = pca.fit_transform(X)



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

model = SVC(kernel='linear')
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
```

```
print(f"Model accuracy after PCA and SVM classification: {accuracy:.2f}")

new_sample = [[5.1, 3.5, 1.4, 0.2]]


new_sample_reduced = pca.transform(new_sample)

predicted_class = model.predict(new_sample_reduced)
predicted_flower = iris.target_names[predicted_class[0]]

print(f"The predicted flower type for the new measurements is:
{predicted_flower}")
```

29_2.PY

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

np.random.seed(42)
income_data = np.random.normal(50000, 15000, 100)
df = pd.DataFrame({'Income': income_data})
```

```
df.dropna(inplace=True)

scaler = StandardScaler()
df['Income_scaled'] = scaler.fit_transform(df[['Income']])

inertia = []
silhouette_scores = []
K_range = range(2, 11)

for k in K_range:
    kmeans = KMeans(n_clusters=k, n_init=10, random_state=42)
    kmeans.fit(df[['Income_scaled']])
    inertia.append(kmeans.inertia_)
    silhouette_scores.append(silhouette_score(df[['Income_scaled']],
kmeans.labels_))

plt.figure(figsize=(10, 5))
plt.plot(K_range, inertia, marker='o')
plt.xlabel("Number of clusters (k)")
plt.ylabel("Inertia")
plt.title("Elbow Method for Optimal k")
plt.show()

plt.figure(figsize=(10, 5))
plt.plot(K_range, silhouette_scores, marker='o', color='orange')
plt.xlabel("Number of clusters (k)")
```

```
plt.ylabel("Silhouette Score")
plt.title("Silhouette Score for Optimal k")
plt.show()

optimal_k = K_range[silhouette_scores.index(max(silhouette_scores))]
print(f"Optimal number of clusters based on silhouette score: {optimal_k}")

kmeans_optimal = KMeans(n_clusters=optimal_k, n_init=10,
random_state=42)
df['Income_Cluster'] = kmeans_optimal.fit_predict(df[['Income_scaled']])

print("Cluster centers (Income groups):")
print(scaler.inverse_transform(kmeans_optimal.cluster_centers_))

sns.scatterplot(data=df, x='Income', y='Income_scaled', hue='Income_Cluster',
palette='viridis')
plt.title("Income Clusters")
plt.show()
```