```
"""3. Write a python program the Categorical values in numeric format for a given
dataset."""
# Import necessary libraries
from sklearn.preprocessing import LabelEncoder
import pandas as pd

# Read a CSV file into a DataFrame
df = pd.read_csv("forestfires.csv")

# Display a random sample of 5 rows from the DataFrame
print("Random sample of 5 rows from the original DataFrame:")
print(df.sample(5))

# Initialize a LabelEncoder
label_encoder = LabelEncoder()

# Encode the 'month' column into a new column 'Numeric_month'
df["Numeric_month"] = label_encoder.fit_transform(df["month"])

# Encode the 'day' column into a new column 'Numeric_day'
df["Numeric_day"] = label_encoder.fit_transform(df["day"])

# Display a random sample of 5 rows from the DataFrame with the new numeric
columns
print("\nRandom sample of 5 rows from the DataFrame with Numeric_month and
Numeric_day columns:")
print(df.sample(5))

# Save the DataFrame with encoded columns to a new CSV file 'encoded_dataset.csv'
(index column not included)
df.to_csv('encoded_dataset.csv', index=False)

# Print a message to confirm the save
print("\nDataFrame with encoded columns saved to 'encoded_dataset.csv'")


""" 8. Write a python program to Implement Decision Tree whether or not to play
tennis."""

# Import necessary libraries
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import LabelEncoder

# Read the dataset from a CSV file
data = pd.read_csv("play_tennis.csv")

# Display a random sample of 5 rows from the dataset
print("Random sample of 5 rows from the dataset:")
print(data.sample(5))
```

```
# Initialize LabelEncoder for encoding categorical columns
label_encoder = LabelEncoder()

# Encode the categorical columns in the dataset
data['outlook'] = label_encoder.fit_transform(data['outlook'])
data['temp'] = label_encoder.fit_transform(data['temp'])
data['humidity'] = label_encoder.fit_transform(data['humidity'])
data['wind'] = label_encoder.fit_transform(data['wind'])
data['play'] = label_encoder.fit_transform(data['play'])

# Display the dataset with categorical columns encoded
print("\nDataset with categorical columns encoded:")
print(data)

# Define the features (independent variables)
x = data[['outlook', 'temp', 'humidity', 'wind']]

# Define the target variable
y = data['play']

# Create a DecisionTreeClassifier
clf = DecisionTreeClassifier()

# Fit the classifier on the features and target variable
clf.fit(x, y)

# Define a new day's features to make a prediction
new_day = [1, 2, 0, 1]

# Use the trained model to predict whether to play tennis on the new day
prediction = clf.predict([new_day])

# Inverse transform the prediction to get the original label
predicted_play = label_encoder.inverse_transform(prediction)

# Print the prediction
print("\nPrediction (0: No, 1: Yes):", prediction[0])
print("Predicted Play:", predicted_play[0])


""" 9. Write a python program to implement linear SVM. """
import pandas as pd
from matplotlib import pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.preprocessing import LabelEncoder

# Read the dataset from a CSV file
df = pd.read_csv('data.csv')

# Display a random sample of 5 rows from the dataset
```

```
print("Random sample of 5 rows from the dataset:")
print(df.sample(5))

# Encode the 'Species' column to create a 'Species_num' column
label_encoder = LabelEncoder()
df['Species_num'] = label_encoder.fit_transform(df['Species'])

# Separate data for each species
df0 = df[df.Species_num == 0]
df1 = df[df.Species_num == 1]
df2 = df[df.Species_num == 2]

# Create scatter plots for each species
plt.scatter(df0['SepalLengthCm'], df0['SepalWidthCm'], color='green', marker='+', 
label='Species 0')
plt.scatter(df1['SepalLengthCm'], df1['SepalWidthCm'], color='red', marker='o', 
label='Species 1')
plt.scatter(df2['SepalLengthCm'], df2['SepalWidthCm'], color='blue', marker='x', 
label='Species 2')
plt.xlabel('Sepal Length (cm)')
plt.ylabel('Sepal Width (cm)')
plt.legend()
plt.show()

# Create the input data by dropping 'Species' and 'Species_num' columns
input = df.drop(['Species', 'Species_num'], axis='columns')

# Display a random sample of 5 rows from the input data
print("\nInput data (features):")
print(input.sample(5))

# Create the target variable
target = df['Species_num']

# Display a random sample of 5 target values
print("\nTarget data (Species_num):")
print(target.sample(5))

# Split the data into training and testing sets (80% train, 20% test)
x_train, x_test, y_train, y_test = train_test_split(input, target, test_size=0.2)

# Print the number of samples in the test set
print("\nNumber of samples in the test set:", len(y_test))

# Create a Support Vector Classifier with a linear kernel
model = SVC(kernel='linear')

# Fit the model on the training data
model.fit(x_train, y_train)

# Calculate the accuracy score of the model on the test data
```

```python
accuracy = model.score(x_test, y_test)

# Print the accuracy score
print("\nAccuracy score on the test data:", accuracy)

# Make predictions on the first 10 samples from the test set
print("\nPredictions on the first 10 samples from the test set:")
predictions = model.predict(x_test[:10])
print(predictions)

# Compare the predictions with the actual values
print("\nActual values for the first 10 samples from the test set:")
actual_values = y_test[:10]
print(actual_values)


""" 1.  Write a python program to Prepare Scatter Plot (Use Forge Dataset / Iris Dataset)"""
import matplotlib.pyplot as plt
import pandas as pd
iris_df = pd.read_csv('data.csv')
iris_df
data =iris_df[['SepalLengthCm','SepalWidthCm']].values
target = iris_df['Species'].values
target_name = iris_df['Species'].unique()
target_name

plt.figure(figsize=(8,6))
for target_value in target_name:
    plt.scatter(data[target == target_value,0],
                data[target == target_value,1],
                label=target_value)
plt.xlabel("sepal Length(cm)")
plt.ylabel("sepal width (cm)")
plt.title("scatter plot of sepal legth VS sepal width")
plt.legend(loc="best",title="iris species")
plt.show()


import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

# Read the dataset
data = pd.read_csv("driver_data.csv", index_col="id")

# Display the first few rows of the dataset
print("First few rows of the dataset:")
print(data.head())
```

```
# Initialize K-Means with 4 clusters
kmeans = KMeans(n_clusters=4)

# Fit K-Means to the data
kmeans.fit(data)

# Display cluster centers
print("Cluster centers:")
print(kmeans.cluster_centers_)

# Get cluster labels for each data point
print("Cluster labels for each data point:")
print(kmeans.labels_)

# Count the number of data points in each cluster
unique, counts = np.unique(kmeans.labels_, return_counts=True)
dict_data = dict(zip(unique, counts))
print("Number of data points in each cluster:")
print(dict_data)

# Assign cluster labels to the original dataset
data["cluster"] = kmeans.labels_

# Create a scatterplot to visualize the clustering results
plt.figure(figsize=(8, 6))
sns.scatterplot(x='mean_dist_day', y='mean_over_speed_perc', data=data,
hue='cluster', palette='coolwarm')
plt.title("K-Means Clustering Result")
plt.xlabel('Mean Distance Driven per Day')
plt.ylabel('Mean Percentage of Time Over Speed Limit')
plt.show()

# Inertia (within-cluster sum of squares)
print("Inertia (within-cluster sum of squares):")
print(kmeans.inertia_)

# Score of the K-Means model (negative inertia)
print("Score of the K-Means model (negative inertia):")
#print(kmeans.score(data))

# Display the modified dataset with cluster labels
print("Modified dataset with cluster labels:")
print(data)

"""12. Write a python program to implement k-nearest Neighbors ML algorithm to
build prediction model (Use Forge Dataset) """

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

```
# importing or loading the dataset
dataset = pd.read_csv('wine.csv')

# distributing the dataset into two components X and Y
X = dataset.iloc[:, 0:13].values
y = dataset.iloc[:, 13].values
# Splitting the X and Y into the
# Training set and Testing set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,
random_state = 0)
# performing preprocessing part
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
# Applying PCA function on training
# and testing set of X component
from sklearn.decomposition import PCA
pca = PCA(n_components = 2)
X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)
explained_variance = pca.explained_variance_ratio_
# Fitting Logistic Regression To the training set
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train, y_train)
# Predicting the test set result using
# predict function under LogisticRegression
y_pred = classifier.predict(X_test)
# making confusion matrix between
# test set of Y and predicted value.
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
# Predicting the training set
# result through scatter plot
from matplotlib.colors import ListedColormap

X_set, y_set = X_train, y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1,
                    stop = X_set[:, 0].max() + 1, step = 0.01),
                    np.arange(start = X_set[:, 1].min() - 1,
                    stop = X_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(),
            X2.ravel()]).T).reshape(X1.shape), alpha = 0.75,
            cmap = ListedColormap(('yellow', 'white', 'aquamarine')))

plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
```

```
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green', 'blue'))(i), label = j)
plt.title('Logistic Regression (Training set)')
plt.xlabel('PC1') # for Xlabel
plt.ylabel('PC2') # for Ylabel
plt.legend() # to show legend
# show scatter plot
plt.show()

# Visualising the Test set results through scatter plot
from matplotlib.colors import ListedColormap

X_set, y_set = X_test, y_test

X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1,
                    stop = X_set[:, 0].max() + 1, step = 0.01),
                    np.arange(start = X_set[:, 1].min() - 1,
                    stop = X_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(),
            X2.ravel()]).T).reshape(X1.shape), alpha = 0.75,
            cmap = ListedColormap(('yellow', 'white', 'aquamarine')))

plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green', 'blue'))(i), label = j)

# title for scatter plot
plt.title('Logistic Regression (Test set)')
plt.xlabel('PC1') # for Xlabel
plt.ylabel('PC2') # for Ylabel
plt.legend()

# show scatter plot
plt.show()


"""4.Write a python program to implement simple Linear Regression for predicting
house price."""
# Import necessary libraries
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import linear_model

# Read the CSV file into a DataFrame
ps = pd.read_csv("Housing.csv")

# Select the 'area' and 'price' columns from the DataFrame
```

```
df = ps[['area', 'price']]

# Display a random sample of 5 rows from the DataFrame
print("Random sample of 5 rows from the original DataFrame:")
print(df.sample(5))

# Create a scatter plot of 'area' vs. 'price'
plt.scatter(df['area'], df['price'], color='red', marker='+')
plt.xlabel('Area')
plt.ylabel('Price')
plt.title('Scatter Plot of Area vs. Price')
plt.show()

# Define the independent variable (features) by dropping the 'price' column
independent_variables = df.drop('price', axis='columns')

# Display the DataFrame with independent variables
print("\nDataFrame with independent variables (without 'price' column):")
print(independent_variables)

# Create a LinearRegression model
model = linear_model.LinearRegression()

# Fit the model using the independent variable and the 'price' column as the
target
model.fit(independent_variables, df['price'])

# Make a prediction for a new area value (e.g., 6720)
new_area = [[6720]]
predicted_price = model.predict(new_area)

# Print the predicted price
print("\nPredicted price for a new area (6720 sq. ft):")
print(predicted_price)


"""5. Write a python program to implement multiple Linear Regression for a given
dataset."""
# Import necessary libraries
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import linear_model

# Read the CSV file into a DataFrame
ps = pd.read_csv("Housing.csv")

# Display a random sample of 10 rows from the DataFrame
print("Random sample of 10 rows from the original DataFrame:")
print(ps.sample(10))

# Select the columns 'area', 'bedrooms', 'bathrooms', 'stories', and 'price'
```

```
df = ps[['area', 'bedrooms', 'bathrooms', 'stories', 'price']]

# Display a random sample of 5 rows from the selected DataFrame
print("\nRandom sample of 5 rows from the selected DataFrame:")
print(df.sample(5))

# Create a LinearRegression model
model = linear_model.LinearRegression()

# Fit the model using the independent variables (all columns except 'price') and
the 'price' column as the target
model.fit(df.drop('price', axis='columns'), df['price'])

# Make a prediction for a new set of features (e.g., area: 3000, bedrooms: 1,
bathrooms: 2, stories: 2)
new_features = [[3000, 2, 2, 2]]
predicted_price = model.predict(new_features)

# Print the predicted price
print("\nPredicted price for new features (area: 3000, bedrooms: 2, bathrooms: 2,
stories: 2):")
print(predicted_price)

"""7. Write a python program to Implement Naïve Bayes."""
# Import necessary libraries
import pandas as pd

# Read the Titanic dataset from a CSV file
df = pd.read_csv("Titanic-Dataset.csv")

# Display the first row of the DataFrame
print("First row of the DataFrame:")
print(df.head(1))

# Remove unnecessary columns
df = df.drop(['PassengerId', 'Name', 'SibSp', 'Parch', 'Embarked', 'Cabin',
'Ticket'], axis='columns')

# Display the updated DataFrame
print("\nDataFrame after removing unnecessary columns:")
print(df.head())

# Import LabelEncoder from scikit-learn
from sklearn.preprocessing import LabelEncoder

# Encode the 'Gender' column
label = LabelEncoder()
df['Gender'] = label.fit_transform(df['Gender'])

# Display the DataFrame with the encoded 'Gender' column
print("\nDataFrame with 'Gender' column encoded:")
```

```
print(df.head())

# Create the input data by dropping the 'Survived' column
input = df.drop(['Survived'], axis='columns')

# Display the input data
print("\nInput data (features):")
print(input.head())

# Create the target data
target = df['Survived']

# Display the target data
print("\nTarget data (Survived):")
print(target.head())

# Check for columns with missing values
print("\nColumns with missing values:")
print(input.columns[input.isna().any()])

# Fill missing values in the 'Age' column with the mean
input.Age = input.Age.fillna(input.Age.mean())

# Display the input data after handling missing values
print("\nInput data after filling missing values:")
print(input.head())

# Import train_test_split from scikit-learn
from sklearn.model_selection import train_test_split

# Split the data into training and testing sets (80% train, 20% test)
x_train, x_test, y_train, y_test = train_test_split(input, target, test_size=0.2)

# Print the number of samples in the test set
print("\nNumber of samples in the test set:", len(y_test))

# Import GaussianNB from scikit-learn
from sklearn.naive_bayes import GaussianNB

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

# Fit the model on the training data
model.fit(x_train, y_train)

# Calculate the accuracy score of the model on the test data
accuracy = model.score(x_test, y_test)

# Print the accuracy score
print("\nAccuracy score on the test data:", accuracy)
```

```
# Make predictions on the first 10 samples from the test set
print("\nPredictions on the first 10 samples from the test set:")
predictions = model.predict(x_test[:10])
print(predictions)

# Compare the predictions with the actual values
print("\nActual values for the first 10 samples from the test set:")
actual_values = y_test[:10]
print(actual_values)


"""10. Write a python program to find Decision boundary by using a neural network
with 10 hidden units on two moons dataset"""

import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_moons
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam

# Generate a synthetic dataset with two moons
X, y = make_moons(n_samples=1000, noise=0.2, random_state=42)

# Display the generated data (X and y)
print("Generated Data:")
print("X (features):")
print(X[:5])
print("\ny (labels):")
print(y[:5])

# 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)

# Create a Sequential model using Keras
model = Sequential()

# Add layers to the model
model.add(Dense(10, input_dim=2, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

# Compile the model
model.compile(loss='binary_crossentropy', optimizer=Adam(learning_rate=0.01),
metrics=['accuracy'])

# Train the model
history = model.fit(X_train, y_train, epochs=50, verbose=1, validation_data=
(X_test, y_test))
```

```
# Generate a grid of points for decision boundary visualization
xx, yy = np.meshgrid(np.linspace(-3, 3, 100), np.linspace(-3, 3, 100))
X_grid = np.c_[xx.ravel(), yy.ravel()]

# Predict the labels for the grid points
Z = model.predict(X_grid).reshape(xx.shape)

# Create a contour plot to visualize the decision boundary
plt.contourf(xx, yy, Z, cmap=plt.cm.RdBu, alpha=0.8)

# Scatter plot the original data points
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.RdBu)
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('Decision Boundary')

# Show the plot
plt.show()

# Display the model training history (loss and accuracy over epochs)
print("Model Training History:")
print("Loss and Accuracy over Epochs:")
print(history.history)

# Evaluate the model on the test data and print the test accuracy
test_loss, test_accuracy = model.evaluate(X_test, y_test, verbose=0)
print("\nTest Accuracy:", test_accuracy)


"""2.   Write a python program to find all null values in a given data set and
remove them.
import pandas as pd
"""
# Import necessary libraries
import pandas as pd
import numpy as np

# Create a dictionary with sample data, including missing values (NaN)
data = {'first score': [100, 90, np.nan, 95],
        'second score': [30, 45, 56, np.nan],
        'third score': [np.nan, 40, 80, 98]}

# Create a DataFrame from the dictionary
df = pd.DataFrame(data)
#df = pd.read_csv("forestfires.csv")  """ Use this only for To read Data from
CSV"""
# Display a DataFrame that shows True for NaN values and False for non-NaN values
print("DataFrame with True for NaN values and False for non-NaN values:")
print(df.isnull())

# Create a new DataFrame that shows True for non-NaN values and False for NaN
```

```
values
not_null_df = df.notnull()
print("\nDataFrame with True for non-NaN values and False for NaN values:")
print(not_null_df)

# Fill NaN values with 0 in the DataFrame (this does not modify the original
DataFrame)
filled_with_zero_df = df.fillna(0)
print("\nDataFrame with NaN values filled with 0:")
print(filled_with_zero_df)

# Fill NaN values with the previous non-NaN value (forward fill)
forward_filled_df = df.fillna(method='pad')
print("\nDataFrame with NaN values forward-filled:")
print(forward_filled_df)

# Fill NaN values with the next non-NaN value (backward fill)
backward_filled_df = df.fillna(method='bfill')
print("\nDataFrame with NaN values backward-filled:")
print(backward_filled_df)

# Replace all NaN values with -99 in the DataFrame (this does not modify the
original DataFrame)
replaced_df = df.replace(to_replace=np.nan, value=-99)
print("\nDataFrame with NaN values replaced by -99:")
print(replaced_df)

# Drop rows with any NaN values in the original DataFrame
dropped_rows_df = df.dropna()
print("\nDataFrame with rows containing NaN values dropped:")
print(dropped_rows_df)

# Drop columns with any NaN values in the original DataFrame
dropped_columns_df = df.dropna(axis=1)
print("\nDataFrame with columns containing NaN values dropped:")
print(dropped_columns_df)

# Create a new DataFrame by dropping rows with any NaN values from the original
DataFrame
new_data = df.dropna()
print("\nNew DataFrame after dropping rows with NaN values:")
print(new_data)


"""6. Write a python program to implement Polynomial Regression for given
dataset."""

# Import necessary libraries
import pandas as pd
import matplotlib.pyplot as plt  # Corrected import statement
```

```
# Read the CSV file into a DataFrame
df = pd.read_csv('Salary.csv')

# Display the DataFrame
print(df)

# Create a scatter plot of 'YearsExperience' vs. 'Salary'
plt.scatter(df['YearsExperience'], df['Salary'])
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.title('Scatter Plot of Years of Experience vs. Salary')
plt.show()

# Extract independent and dependent variables
x = df.iloc[:, 1:-1].values  # Independent variable
y = df.iloc[:, -1].values  # Dependent variable

# Display the independent variable (x) and dependent variable (y)
print("\nIndependent variable (x):")
print(x)
print("\nDependent variable (y):")
print(y)

# Import PolynomialFeatures from scikit-learn
from sklearn.preprocessing import PolynomialFeatures

# Create a PolynomialFeatures object with a degree of 3
poly = PolynomialFeatures(degree=3)

# Transform the independent variable (x) into polynomial features
x_poly = poly.fit_transform(x)

# Display the transformed polynomial features
print("\nPolynomial features (x_poly):")
print(x_poly)

# Import LinearRegression from scikit-learn
from sklearn.linear_model import LinearRegression

# Create a LinearRegression model
model = LinearRegression()

# Fit the model using the polynomial features and the dependent variable (y)
model.fit(x_poly, y)

# Create a scatter plot of the original data points
plt.scatter(x, y)

# Plot the regression curve
plt.plot(x, model.predict(x_poly), color='red')
```

```
# Display the plot
plt.show()

# Make a prediction for a new value (e.g., YearsExperience = 6)
new_value = [[6]]
predicted_salary = model.predict(poly.transform(new_value))

# Print the predicted salary
print("\nPredicted salary for YearsExperience = 6:")
print(predicted_salary)


"""11. Write a python program to transform data with Principal Component Analysis
(PCA)"""
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

# Load the dataset
data = pd.read_csv('mpg.csv')
data.head()

# Remove 'car name' and 'origin' columns
data = data.drop(['name', 'origin'], axis=1)
data.head()

# Identify non-digit entries in 'horsepower' column
hpisdigit = pd.DataFrame(data.horsepower.str.isdigit())
print("Rows with non-digit entries in 'horsepower' column:")
print(data[hpisdigit['horsepower'] == False])

# Replace '?' with NaN and convert 'horsepower' to float
data = data.replace('?', np.nan)
data['horsepower'] = data['horsepower'].astype('float64')

# Handle missing values by filling with median
medianfiller = lambda x: x.fillna(x.median())
data = data.apply(medianfiller, axis=0)
data['horsepower'] = data['horsepower'].astype('float64')

# Separate features and target variable
x = data.drop(['mpg'], axis=1)
y = data[['mpg']]

# Visualize pairplots of the features
sns.pairplot(x)
plt.show()
```

```
# Standardize the features using z-score
scaler = StandardScaler()
Xscaled = scaler.fit_transform(x)
Xscaled = pd.DataFrame(Xscaled, columns=x.columns)
Xscaled.head()

# Calculate the covariance matrix
covmatrix = Xscaled.cov()
print("Covariance matrix:")
print(covmatrix)

# Apply Principal Component Analysis (PCA)
pca = PCA(n_components=5)
pca.fit(Xscaled)

print("Explained variance by each principal component:")
print(pca.explained_variance_)

print("Principal components (eigenvectors):")
print(pca.components_)

print("Explained variance ratio:")
print(pca.explained_variance_ratio_)

# Transform data using PCA
xpca = pca.transform(Xscaled)

# Train a linear regression model on the original data
regression_model = LinearRegression()
regression_model.fit(Xscaled, y)

print("R-squared score (Original Data):", regression_model.score(Xscaled, y))

# Train a linear regression model on PCA-transformed data
regression_model_pca = LinearRegression()
regression_model_pca.fit(xpca, y)

print("R-squared score (PCA-Transformed Data):", regression_model_pca.score(xpca,
y))

# Redo PCA with 3 components
pca = PCA(n_components=3)
pca.fit(Xscaled)

print("Explained variance by each principal component (3 components):")
print(pca.explained_variance_)

print("Principal components (eigenvectors):")
print(pca.components_)
```

```python
print("Explained variance ratio (3 components):")
print(pca.explained_variance_ratio_)

# Transform data using PCA with 3 components
xpca = pca.transform(Xscaled)

# Train a linear regression model on the original data
regression_model = LinearRegression()
regression_model.fit(Xscaled, y)

print("R-squared score (Original Data):", regression_model.score(Xscaled, y))

# Train a linear regression model on PCA-transformed data (3 components)
regression_model_pca = LinearRegression()
regression_model_pca.fit(xpca, y)

print("R-squared score (PCA-Transformed Data - 3 components):",
regression_model_pca.score(xpca, y))
```