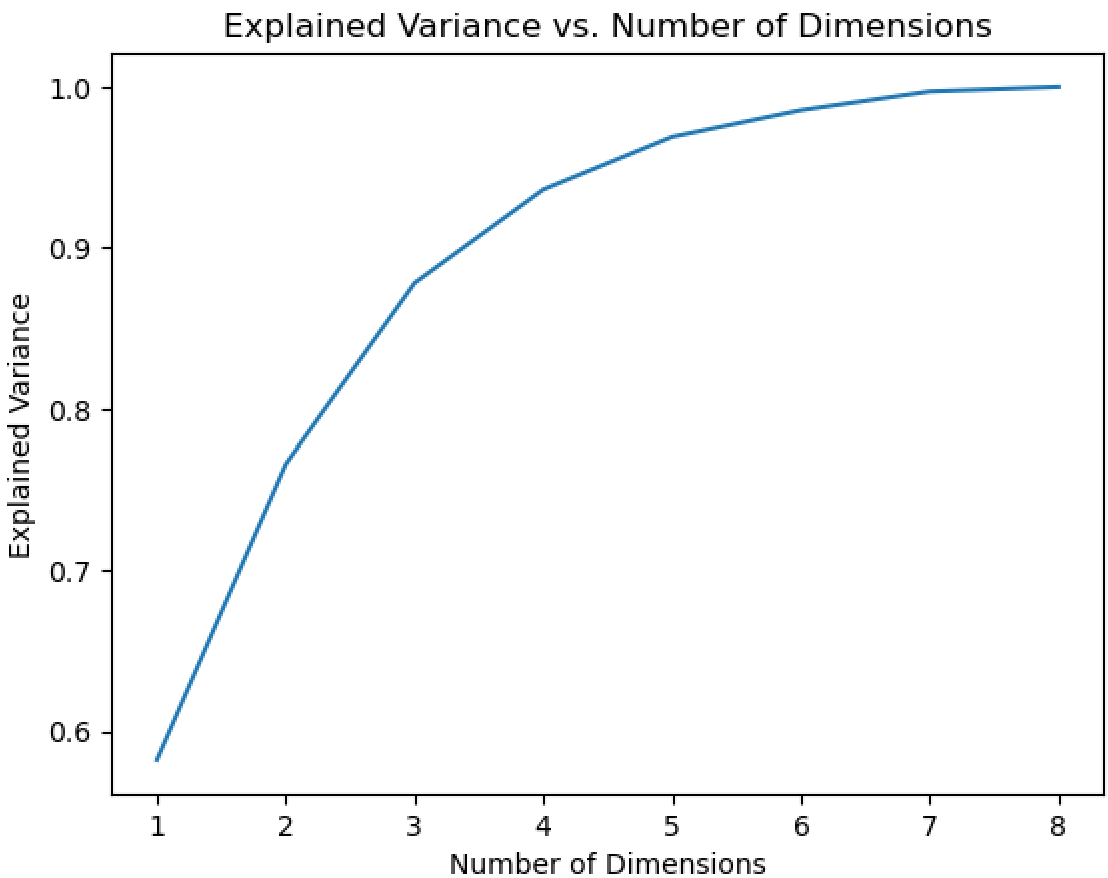
CSCI 6223 Practical Data Science
Homework-2

Sivadeep kotha, sivadeep.kotha@my.okcu.edu
Venkat sai sheelam, vsheelam@my.okcu.edu
Vineela pedapudi, vpedapudi@my.okcu.edu

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

# Load the dataset
data = pd.read_csv('C:\Users\sivad\OneDrive\Desktop\auto_data.csv')

# Select relevant numeric features
numeric_columns = ['wheel-base', 'length', 'width', 'height', 'engine-
size', 'peak-rpm', 'city-mpg', 'highway-mpg']

# Replace missing values (in this case, '?') with 0 and convert to float
numeric_features = data[numeric_columns].replace('?', '0').astype(float)

# Standardize the data
scaler = StandardScaler()
scaled_data = scaler.fit_transform(numeric_features)

# Perform PCA
pca = PCA()
pca.fit(scaled_data)

# Calculate cumulative explained variance
explained_variance_ratio = pca.explained_variance_ratio_
cumulative_variance = explained_variance_ratio.cumsum()

# Determine the number of components to explain 90% of variance
n_components = np.argmax(cumulative_variance >= 0.9) + 1

print("Number of components to explain 90% variance:", n_components)
```

```python
# Plot explained variance versus the number of dimensions
plt.plot(range(1, len(explained_variance_ratio) + 1), cumulative_variance)
plt.xlabel('Number of Dimensions')
plt.ylabel('Explained Variance')
plt.title('Explained Variance vs. Number of Dimensions')
plt.show()
```



REASONING

Dimensionality Reduction: Reducing the number of features can lead to simpler models , better computational efficiency, and improved model generalization.

Variance: PCA aims to get as much information as possible. By setting the goal to 90 % explained variance, we are balancing between dimensionality reduction and fetching relevant information:

Practicality : While retaining all the principle components would capture 100% of the Variance, it may lead to little to no reduction in dimensionality. So, retaining too few components might result in information loss.

The specific choice to explain 90% of the variance is pragmatic and seeks to simplify the dataset while preserving most of the relevant information. It allows for a balance between reducing dimensionality and maintaining model interpretability and predictive power.

The code includes the plot visualizes with this we can say that the variance changes as the number of principal components increases. And also useful for gaining insight into how much variance each principal component contributes.It also explains how the variance is distributed across the dimensions and helps in understanding the trade off between dimensionality and information reduction.

```
# One-Hot Encoding
X = pd.get_dummies(data, columns=['num-of-doors', 'number-of-cylinders'],
drop_first=True)

from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
X['num-of-doors'] = le.fit_transform(data['num-of-doors'])
X['number-of-cylinders'] = le.fit_transform(data['number-of-cylinders'])
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

# Load the dataset and replace '?' with NaN
data = pd.read_csv('C:\Users\sivad\OneDrive\Desktop\auto_data.csv',
na_values=['?'])

# Handle missing values
data = data.dropna()  # Remove rows with missing values

# Select the features
selected_features = ['wheel-base', 'length', 'width', 'height', 'engine-
size', 'peak-rpm', 'city-mpg', 'highway-mpg']
X = data[selected_features]
```

```
y = data['price']

# Split the data 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 linear regression model
model = LinearRegression()

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

# Make predictions on the test data
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Mean Squared Error:", mse)
print("R-squared:", r2)

Mean Squared Error: 17728206.55574337
R-squared: 0.8422405281870393

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


# Select the relevant numeric features for PCA
numeric_features = data[['wheel-base', 'length', 'width', 'height',
'engine-size', 'peak-rpm', 'city-mpg', 'highway-mpg']]

# Standardize the data
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaled_data = scaler.fit_transform(numeric_features)
```

```
# Perform PCA with the number of components determined in Part 1
pca = PCA(n_components=4)  # Use the number of components that explain 90%
of the variance
pca_data = pca.fit_transform(scaled_data)

# Create a DataFrame with the principal components
pca_df = pd.DataFrame(data=C:\Users\sivad\OneDrive\Desktop\pca_data.csv,
columns=['PC1', 'PC2', 'PC3', 'PC4'])

# Select the principal components as input features
X = pca_df
y = data['price']

# Split the data 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 linear regression model
model = LinearRegression()

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

# Make predictions on the test data
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Mean Squared Error:", mse)
print("R-squared:", r2)

Mean Squared Error: 30251309.647427015
R-squared: 0.7308001451459682
```