In [264…
```
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
```

CS 548 Homework #2: Data Preprocessing and Exploration

Problem 1 - KDD Definition

Based on the following article: Fayyad, U., Piatetsky-Shapiro, G., and Smith, P. “From Data Mining to Knowledge Discovery in Databases”. AAAI Magazine. pp. 37-54. Fall, 1996, as well as on the Tan text and lecture content, answer the following questions:

1. Define the term, **knowledge discovery in databases.**

KDD is the multi-step process of extracting insights (knowledge) from data. Data mining is a step within this process.

2. Briefly describe the steps of the knowledge discovery in databases process.

1. Become familiar with the domain the data is from. Understand what the customer hopes to get out of the KDD process.
2. Identify the data to be used.
3. Clean and preprocess data
4. Perform dimensionality reduction/feature transformation
5. Identify a data mining method that is suited to the KDD goal.
6. Perform EDA and select a model and hypothesis. It sounds from the paper as though the distinction between steps 5 and 6 is that 5 is identifying a general approach where as 6 is identifying the specific algorithms/techniques to be used.
7. Perform data mining.
8. Interpret and visualize patterns discovered during data mining. If needed, repeat steps 1-7.
9. Act on the knowledge discovered.

3. Define the term, **data mining**.

Data mining is the step of the KDD process that searches for patterns in the data. It is one of the two discovery steps of the KDD process (data mining and pattern interpretation).

Problem 2 - Data Preprocessing

Consider the following dataset:

COUNTRY	LIFE-EXP	GDPPC	AC-S-ED	SWL
Switzerland	80.5	32.3	99.9	[250-275)
Canada	80	34	102.6	[250-275)
USA	77.4	?	94.6	[225-250)
Germany	78.7	30.4	99	[225-250)
Mexico	75.1	10	73.4	[225-250)
France	79.5	29.9	108.7	[200-225)
Thailand	70	8.3	79	[200-225)
Brazil	70.5	8.4	103.2	[200-225)
Japan	82	31.5	102.1	[200-225)
India	63.3	3.3	49.9	[175-200)
Ethiopia	47.6	0.9	5.2	[150-175)
Russia	65.3	11.1	81.9	[125-150)

Table 1: International Demographic/Economic/Life Satisfaction Data

where each of the columns is defined below:

- LIFE-EXP: Life Expectancy from UN Human Development Report (2003)
- GDPPC: GDP per capita from figure published by the CIA (2006) in US$.
- AC-S-ED: Access to secondary education rating from UNESCO (2002)
- SWL (satisfaction with life) index calculated from data published by New Economics Foundation (2006).

1. Assuming that the missing value (marked with "?") in GDPPC cannot be ignored, discuss 3 different alternatives to fill in that missing value. In each case, state what the selected value would be and the advantages and disadvantages of the approach. You may assume that the SWL attribute is the target attribute.

Option A: Impute with the column mean or median.

If using the mean the filled value would be 18.2. If using median the filled value would be 11.1. Both mean and median can be good approaches in situations like this where the amount of missing data is limited. Mean can be susceptible to outliers, so median may be a

better approach than mean, but I don't think either is ideal here since the results of either imputation would be inconsistent with the GDPPC for other countries with similar SWL to the US.

In [9]:
```
GDPPC_vals = [32.3, 34, 30.4, 10, 29.9, 8.3, 8.4, 31.5, 3.3, 0.9, 11.1]
print('The mean GDPPC value is: ', round(np.mean(GDPPC_vals), 1))
print('The median GDPPC value is: ', np.median(GDPPC_vals))
```

```
The mean GDPPC value is:  18.2
The median GDPPC value is:  11.1
```

Option B: Impute with **group** mean or median.

To try and fill the value most accurately we might choose to focus only on the samples that are most similar to the sample we are trying to fill data for. Since we are interested in SWL, this could be a good feature to group the samples by. SWL is, in effect, a categorical variable, and we could group the USA with other countries with a SWL of 225-250. Since only two other countries are in this group (Germany and Mexico), the mean and median will be the same, 20.2.

In [21]:
```
print('The mean GDPPC value for the SWL 225-250 group is: ', round(np.mean(GDPPC_va
print('The median GDPPC value for the SWL 225-250 group is: ', np.median(GDPPC_vals
```

```
The mean GDPPC value for the SWL 225-250 group is:  20.2
The median GDPPC value for the SWL 225-250 group is:  20.2
```

Option C: Impute with value from predictive model.

We could also train a model to predict the GDPPC value for the USA. A linear regression fit to this task predicted 23.3. This approach is useful because it allows us to use the correlations between the missing feature with a value and other features in the dataset. The only downside I can think of for this method is that it, with a larger dataset, this approach could be meaningfully more computationally expensive than the other methods.

In [7]:
```
data = {
    'COUNTRY': ['Switzerland', 'Canada', 'USA', 'Germany', 'Mexico',
                'France', 'Thailand', 'Brazil', 'Japan', 'India',
                'Ethiopia', 'Russia'],
    'LIFE-EXP': [80.5, 80, 77.4, 78.7, 75.1,
                 79.5, 70, 70.5, 82, 63.3,
                 47.6, 65.3],
    'GDPPC': [32.3, 34, None, 30.4, 10,
              29.9, 8.3, 8.4, 31.5, 3.3,
              0.9, 11.1],
    'AC-S-ED': [99.9, 102.6, 94.6, 99, 73.4,
                108.7, 79, 103.2, 102.1, 49.9,
                5.2, 81.9],
    'SWL': ['[250-275]', '[250-275]', '[225-250]', '[225-250]', '[225-250]',
            '[200-225]', '[200-225]', '[200-225]', '[200-225]', '[175-200]',
            '[150-175]', '[125-150]']
}
```

```
data = pd.DataFrame(data)

# df = pd.get_dummies(df, columns=['COUNTRY','SWL'], drop_first=True)
df = pd.get_dummies(data, columns=['SWL'], drop_first=True)
df = df.drop(columns=['COUNTRY'])
df.head()
```

Out[7]:

	LIFE-EXP	GDPPC	AC-S-ED	SWL_[150-175]	SWL_[175-200]	SWL_[200-225]	SWL_[225-250]	SWL_[250-275]
0	80.5	32.3	99.9	False	False	False	False	True
1	80.0	34.0	102.6	False	False	False	False	True
2	77.4	NaN	94.6	False	False	False	True	False
3	78.7	30.4	99.0	False	False	False	True	False
4	75.1	10.0	73.4	False	False	False	True	False

In [5]:
```
test = df[df['GDPPC'].isnull()==True]
train = df[df['GDPPC'].isnull()==False]

x_train = train.drop(columns=['GDPPC'])
y_train = train['GDPPC']
x_test = test.drop(columns=['GDPPC'])

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

y_pred = model.predict(x_test)

print('The predicted GDPPC value for the USA is : ', round(y_pred[0],1))
```

```
The predicted GDPPC value for the USA is :  23.3
```

2. Would you keep the attribute COUNTRY in your dataset when mining for patterns that predict the values for the SWL attribute? Explain your answer.

I would not keep COUNTRY since this feature really just acts as an individual identifier and doesn't give the model any information it could use to learn patterns in the data. If there were more than one entry per country I would keep the feature.

3. Describe a reasonable transformation of the attribute COUNTRY so that the number of different values for that attribute is reduced to just 4.

We could transform COUNTRY into CONTINENT with 4 values:

1. North America (US, Canada)
2. Europe (Germany, France, Switzerland, Russia)

3. Asia (Thailand, Japan, India)
4. Other (Ethopia, Mexico)

The majority of Russia's landmass is in Asia but the majority of its population is in the European portion. Since our features are more closely tied to population than land I elected to put Russia in the Europe group.

4. Discretize the AC-S-ED attribute by binning it into 4 equi-width intervals using unsupervised discretization. Perform this discretization by hand (i.e., do not use any software tools). Explain your answer.

The distance between the minimum (5.2) and maximum (108.7) values for AC-S-ED is 103.5. Dividing 103.5 into 4 groups of equal size results in groups at 25.875 intervals. We end up with the following bins:

- Bin 1: 5.2 - 31.075
- Bin 2: 31.075 - 56.95
- Bin 3: 56.95 - 82.825
- Bin 4: 82.825 - 108.7

5. Discretize the AC-S-ED attribute by binning it into 4 equi-depth (= equal- frequency) intervals using unsupervised discretization. Perform this dis- cretization by hand (i.e., do not use any software). Explain your answer.

We can bin the 12 samples in our dataset into 4 bins of 3. If we sort the samples by AC-S-ED value we end up with the following bins:

- Bin 1: 5.2-73.4
- Bin 2: 79-94
- Bin 3: 99-102.1
- Bin 4: 102.6-108.7

6. Consider the following new approach to discretizing a numeric attribute: Given the mean and the standard deviation (sd) of the attribute values, bin the attribute values into the following intervals:

$$[mean - (k+1) \times sd, mean - k \times sd], \forall k = \ldots, -4, -3, -2, -1, 0, 1, 2, \ldots (1)$$

> Assume that the mean of the attribute AC-S-ED above is 83 and that the standard deviation sd of this attribute is 30. Discretize AC-S-ED by hand using this new approach. Show your work.

The approach described above will create bins that each cover 30 (the standard deviation) AC-S-ED values. To find the appropriate range for k, we need to find the first value of k where $mean - k \times sd$ will be larger than the largest value in our dataset and the next value of k where $mean - (k+1) \times sd$ will be smaller than the smallest value. A bin created with $k = -1$ would have an upper bound of 113:

$$mean - k \times sd = 83 - (-1) \times 30 = 113,$$

which is larger than our largest value of 108.7, while having a lower bound ($113 - 30 = 83$) that is within the range of our data. As such, $k = -1$ is an appropriate first value for k.
$k = 2$ is the next value of k where the lower bound (-7) is smaller than our column minimum.

For k∈{−1,0,1,2} we have the following bins:

$k = -1$:

$$[83 - (-1+1) * 30, 83 - (-1) * 30] = [83, 113]$$

$k = 0$:

$$[83 - (0+1) * 30, 83 - (0) * 30] = [53, 83]$$

$k = 1$:

$$[83 - (1+1) * 30, 83 - (1) * 30] = [23, 53]$$

$k = 2$:

$$[83 - (2+1) * 30, 83 - (2) * 30] = [83, 113]$$

Problem 3 - Exploratory Data Analysis

Consider Auto Dataset available on the Kaggle website. Load the dataset into a Jupyter notebook.

- Make the attributes mpg, displacement, horsepower, weight, and acceleration continuous.
- Make the attributes cylinders, model-year, and origin discrete;
- Make the attribute car-name string.

In [265…

```
file_path = 'data/auto-mpg.csv'
auto_dataset = pd.read_csv(file_path)
print(auto_dataset.info())
print('-----------------')
auto_dataset.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 398 entries, 0 to 397
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   mpg           398 non-null    float64
 1   cylinders     398 non-null    int64
 2   displacement  398 non-null    float64
 3   horsepower    398 non-null    object
 4   weight        398 non-null    int64
 5   acceleration  398 non-null    float64
 6   model year    398 non-null    int64
 7   origin        398 non-null    int64
 8   car name      398 non-null    object
dtypes: float64(3), int64(4), object(2)
memory usage: 28.1+ KB
None
-----------------
```

Out[265…

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	nar
0	18.0	8	307.0	130	3504	12.0	70	1	chevro cheve mali
1	15.0	8	350.0	165	3693	11.5	70	1	bu skyl 3
2	18.0	8	318.0	150	3436	11.0	70	1	plymou satell
3	16.0	8	304.0	150	3433	12.0	70	1	a rebel
4	17.0	8	302.0	140	3449	10.5	70	1	fc tori

In [267…

```
# mpg, displacement, and acceleration are already continuous, just need to convert
auto_dataset['horsepower'] = pd.to_numeric(auto_dataset['horsepower'], errors='coer
auto_dataset['weight'] = pd.to_numeric(auto_dataset['weight'], errors='coerce').ast
# cylinders, model-year, and origin are already discreet, no need to adjust those
auto_dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 398 entries, 0 to 397
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   mpg           398 non-null    float64
 1   cylinders     398 non-null    int64  
 2   displacement  398 non-null    float64
 3   horsepower    392 non-null    float64
 4   weight        398 non-null    float64
 5   acceleration  398 non-null    float64
 6   model year    398 non-null    int64  
 7   origin        398 non-null    int64  
 8   car name      398 non-null    object 
dtypes: float64(5), int64(3), object(1)
memory usage: 28.1+ KB
```

In [ ]:
```
# car-name is an object so the values might already be strings but I'll check to be
is_string_column = auto_dataset['car name'].apply(lambda x: isinstance(x, str))
# Check if all values are strings
print(is_string_column.all())
```

```
True
```

1. Start by familiarizing yourself with the dataset. Carefully look at the data directly (for this, use Python functionality to explore and visualize the data). Describe in your report your observations about what is good about this data (mention at least 2 different good things), and what is problematic about this data (mention at least 2 different bad things). If appropriate, include visualizations of those good/bad things.

Good observations:

- The data does not contain many null values. Of the 3582 values in the dataset (398 samples, 9 columns) only 6 were null, .17% of the data.
- None of the continuous features have extremely long tails, outliers appear minimal.

Bad observations:

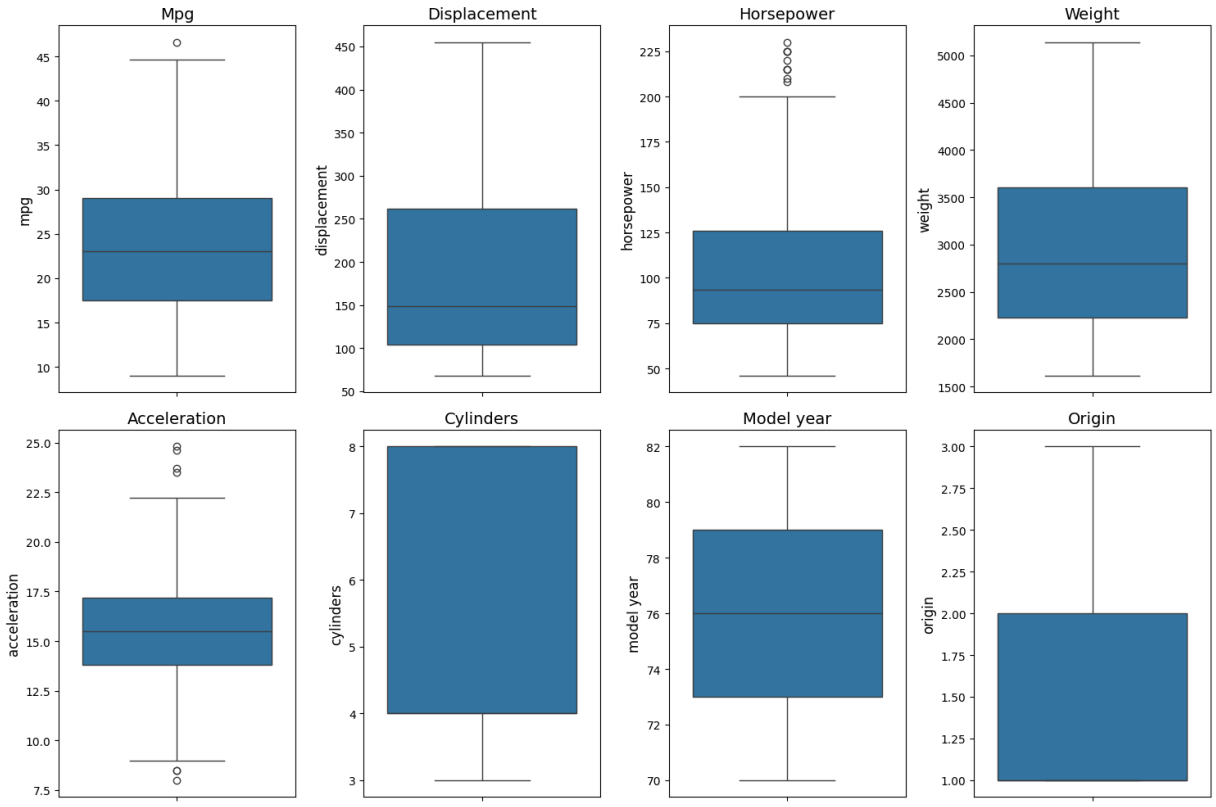
- Box plots show skewness especially for origin, cylinders and displacement.
- Plotting the correlation matrix shows high collinearity between the continuous features.
- Many of the car names (63%) only appear once in the dataset.

In [ ]:
```
print(auto_dataset.isnull().sum())
```

```
mpg             0
cylinders       0
displacement    0
horsepower      6
weight          0
acceleration    0
model year      0
origin          0
car name        0
dtype: int64
```

In [117…

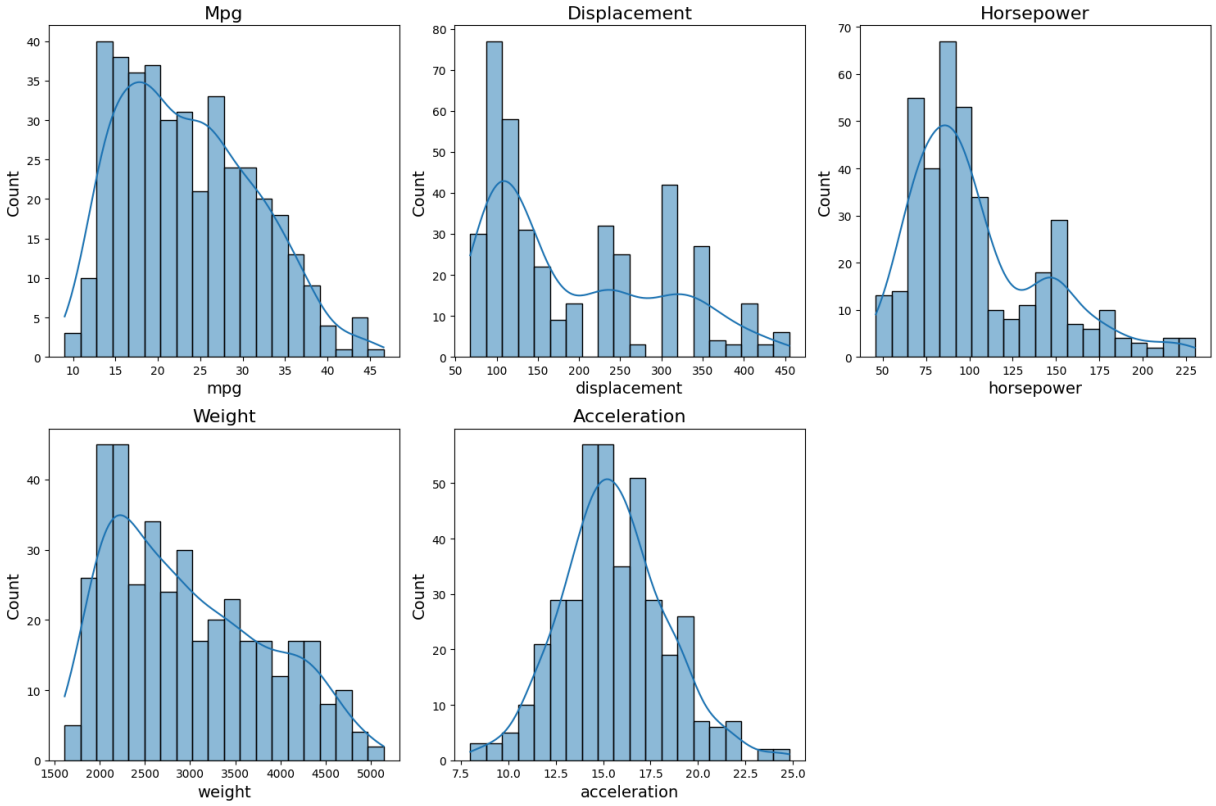
```
# Create box plots for each continuous feature
continuous_features = [
    'mpg', 'displacement', 'horsepower', 'weight', 'acceleration',
    'cylinders', 'model year', 'origin'
    ]

plt.figure(figsize=(15, 10))
for i, feature in enumerate(continuous_features, 1):
    plt.subplot(2, 4, i)
    sns.boxplot(y=auto_dataset[feature])
    plt.title(feature.capitalize(), fontsize=14)
    plt.ylabel(feature, fontsize=12)

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



In [147…

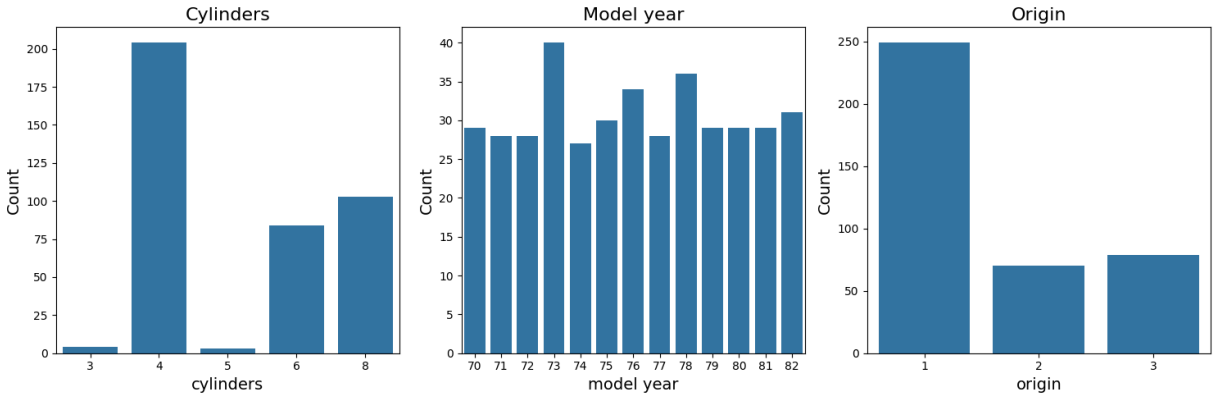
```
continuous_features = ['mpg', 'displacement', 'horsepower', 'weight', 'acceleration
# Create bar plots for each continuous feature
plt.figure(figsize=(15, 10))
for i, feature in enumerate(continuous_features, 1):
```

```
    plt.subplot(2, 3, i)
    sns.histplot(auto_dataset[feature], bins=20, kde=True)
    plt.title(feature.capitalize(), fontsize=16)
    plt.xlabel(feature, fontsize=14)
    plt.ylabel('Count', fontsize=14)

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

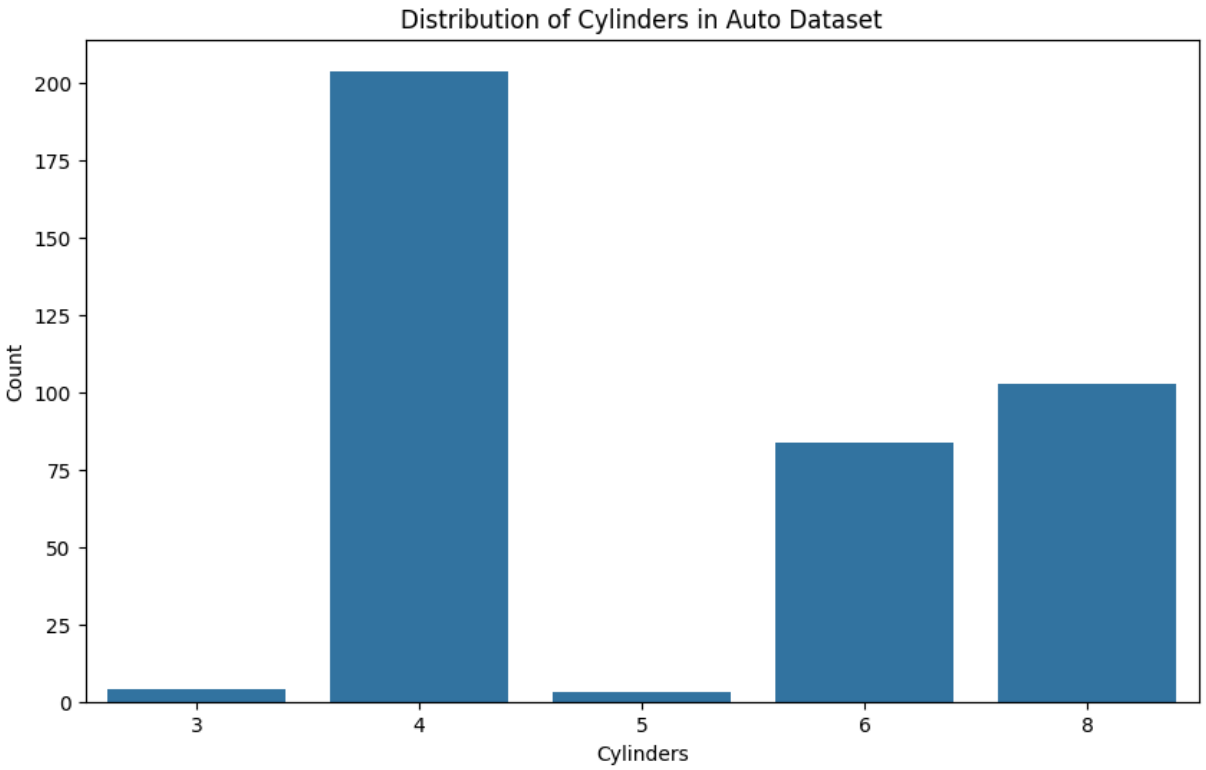


In [148…

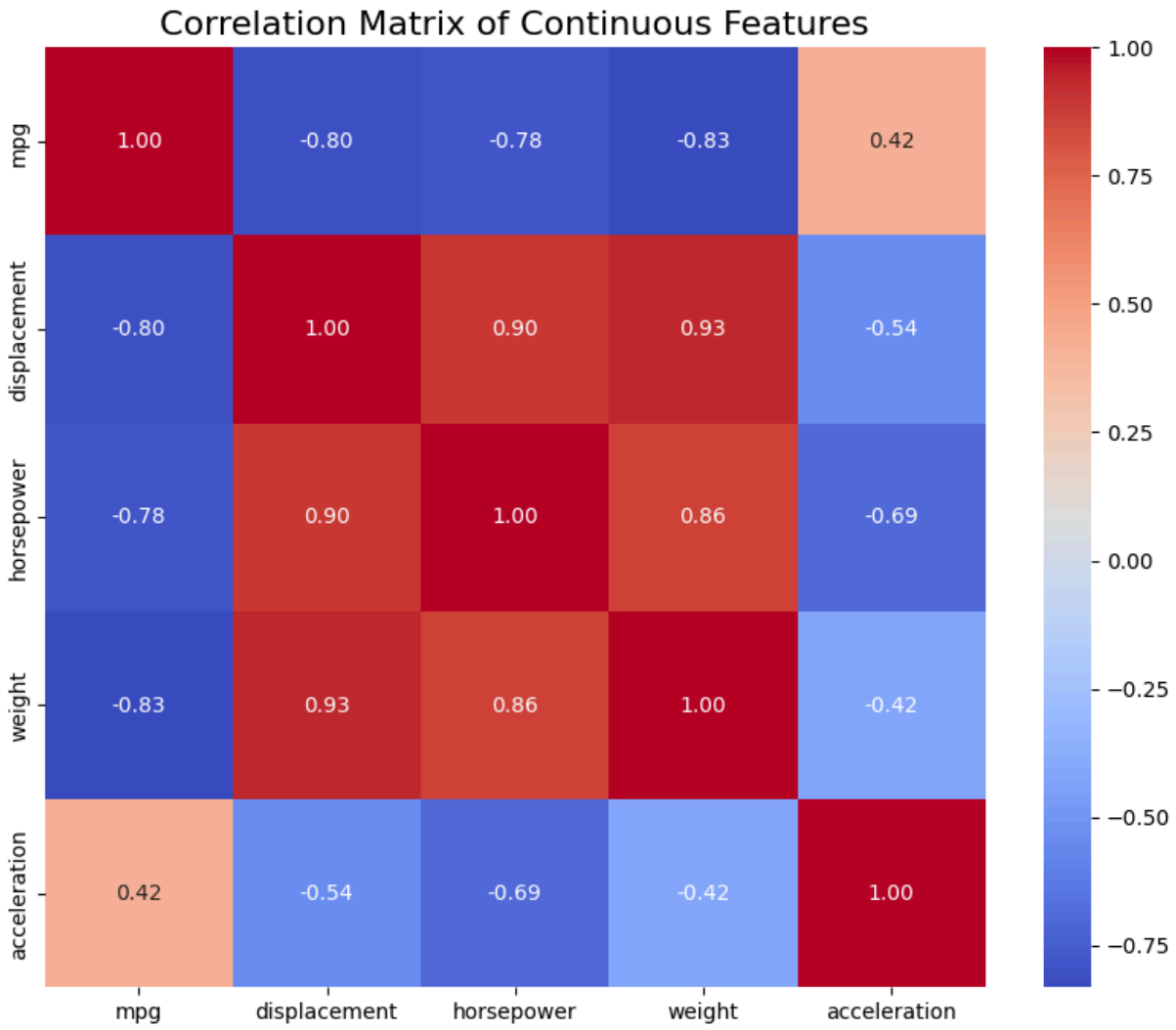
```
discreet_features = ['cylinders', 'model year', 'origin']
# Create bar plots for each discrete feature
plt.figure(figsize=(15, 5))
for i, feature in enumerate(discreet_features, 1):
    plt.subplot(1, 3, i)
    sns.countplot(x=auto_dataset[feature])
    plt.title(feature.capitalize(), fontsize=16)
    plt.xlabel(feature, fontsize=14)
    plt.ylabel('Count', fontsize=14)

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



In [144…
```
# Count the occurrences of each unique value in the 'cylinders' column
cylinders_counts = auto_dataset['cylinders'].value_counts()

# Create a bar plot
plt.figure(figsize=(10, 6))
sns.barplot(x=cylinders_counts.index, y=cylinders_counts.values)#, palette='viridis
plt.xlabel('Cylinders')
plt.ylabel('Count')
plt.title('Distribution of Cylinders in Auto Dataset')
plt.show()
```



In [ ]:
```
# Create a correlation matrix for continuous features
correlation_matrix_continuous = auto_dataset[continuous_features].corr()

plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix_continuous, annot=True, fmt=".2f", cmap='coolwarm',
plt.title('Correlation Matrix of Continuous Features', fontsize=16)
plt.show()
```



In [142…
```
print(f'There are {len(auto_dataset["car name"])} unique car names in the dataset.'
cars_per_name_counts = auto_dataset['car name'].value_counts().value_counts()
print('There are:')
for count, number_of_cars in cars_per_name_counts.items():
    print(f'  {number_of_cars} car names that appear {count} times.')
print(f"{round(cars_per_name_counts[1]/len(auto_dataset['car name'])*100)}% of car
```

```
There are 398 unique car names in the dataset.
There are:
  249 car names that appear 1 times.
  34 car names that appear 2 times.
  12 car names that appear 3 times.
  6 car names that appear 4 times.
  3 car names that appear 5 times.
  1 car names that appear 6 times.
63% of car names appear only once in the dataset.
```

2. For the horsepower attribute:

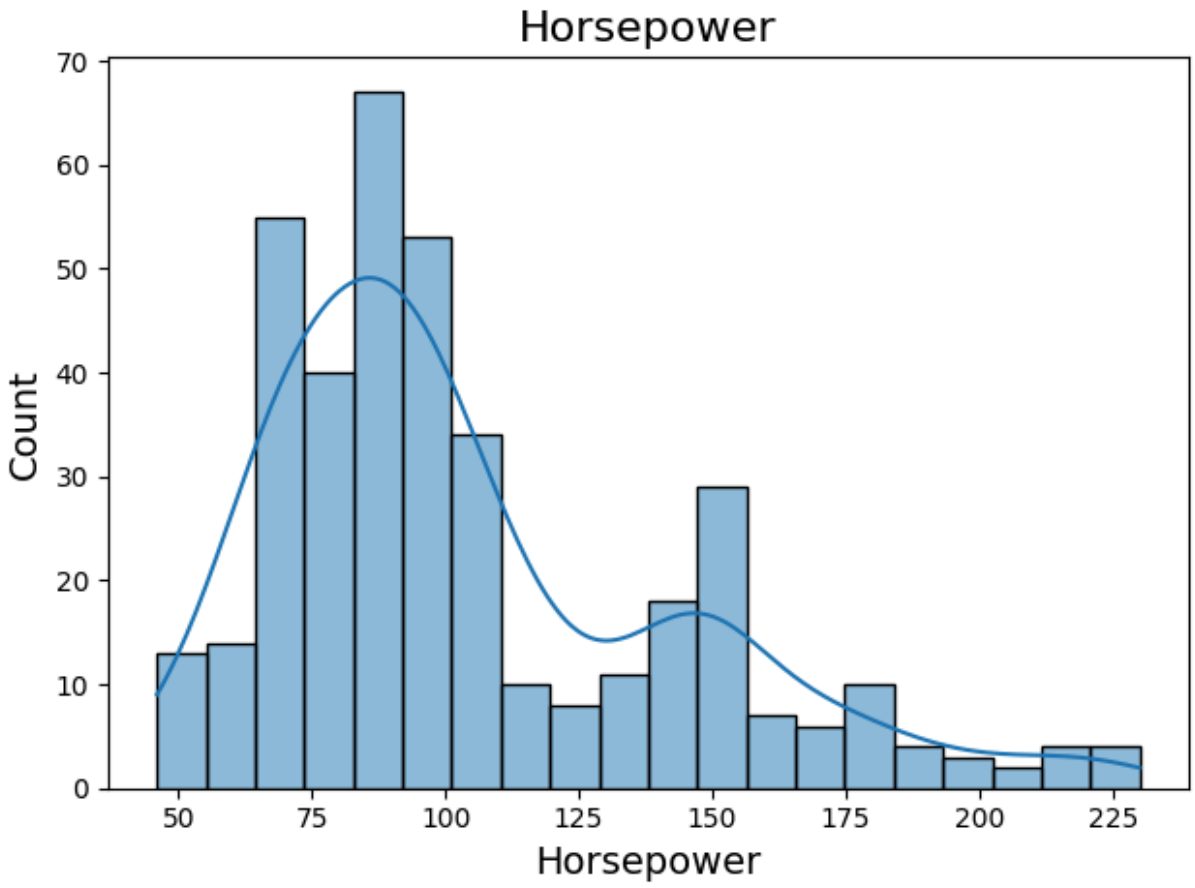
(a) Calculate the percentiles in increments of 10, the mean, median, range, and variance.

In [149…
```
# Calculate percentiles in increments of 10
percentiles = np.percentile(auto_dataset['horsepower'].dropna(), np.arange(0, 101,
print("Percentiles in increments of 10:", percentiles)

# Calculate mean
mean_hp = np.mean(auto_dataset['horsepower'])
print("Mean:", mean_hp)

# Calculate median
median_hp = np.median(auto_dataset['horsepower'].dropna())
print("Median:", median_hp)

# Calculate range
range_hp = np.ptp(auto_dataset['horsepower'].dropna())
print("Range:", range_hp)

# Calculate variance
variance_hp = np.var(auto_dataset['horsepower'].dropna())
print("Variance:", variance_hp)
```

```
Percentiles in increments of 10: [ 46.   67.   72.   80.   88.   93.5 100.  110.  14
0.  157.7 230. ]
Mean: 104.46938775510205
Median: 93.5
Range: 184.0
Variance: 1477.789879216993
```

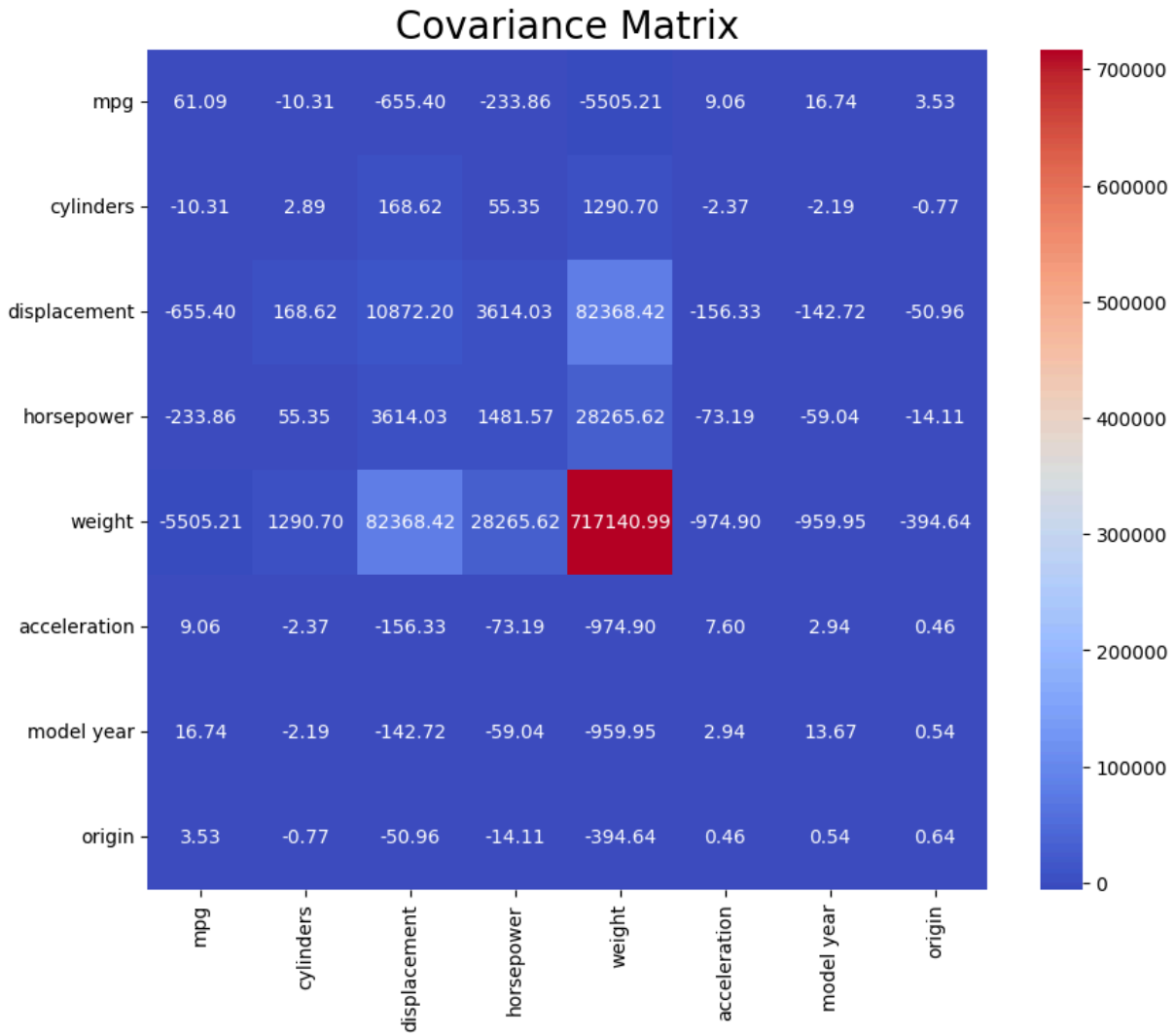
(b) Plot a histogram of the attribute using 10 or 20 bins.

In [150…
```
sns.histplot(auto_dataset['horsepower'], bins=20, kde=True)
plt.title('Horsepower', fontsize=16)
plt.xlabel('Horsepower', fontsize=14)
plt.ylabel('Count', fontsize=14)

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

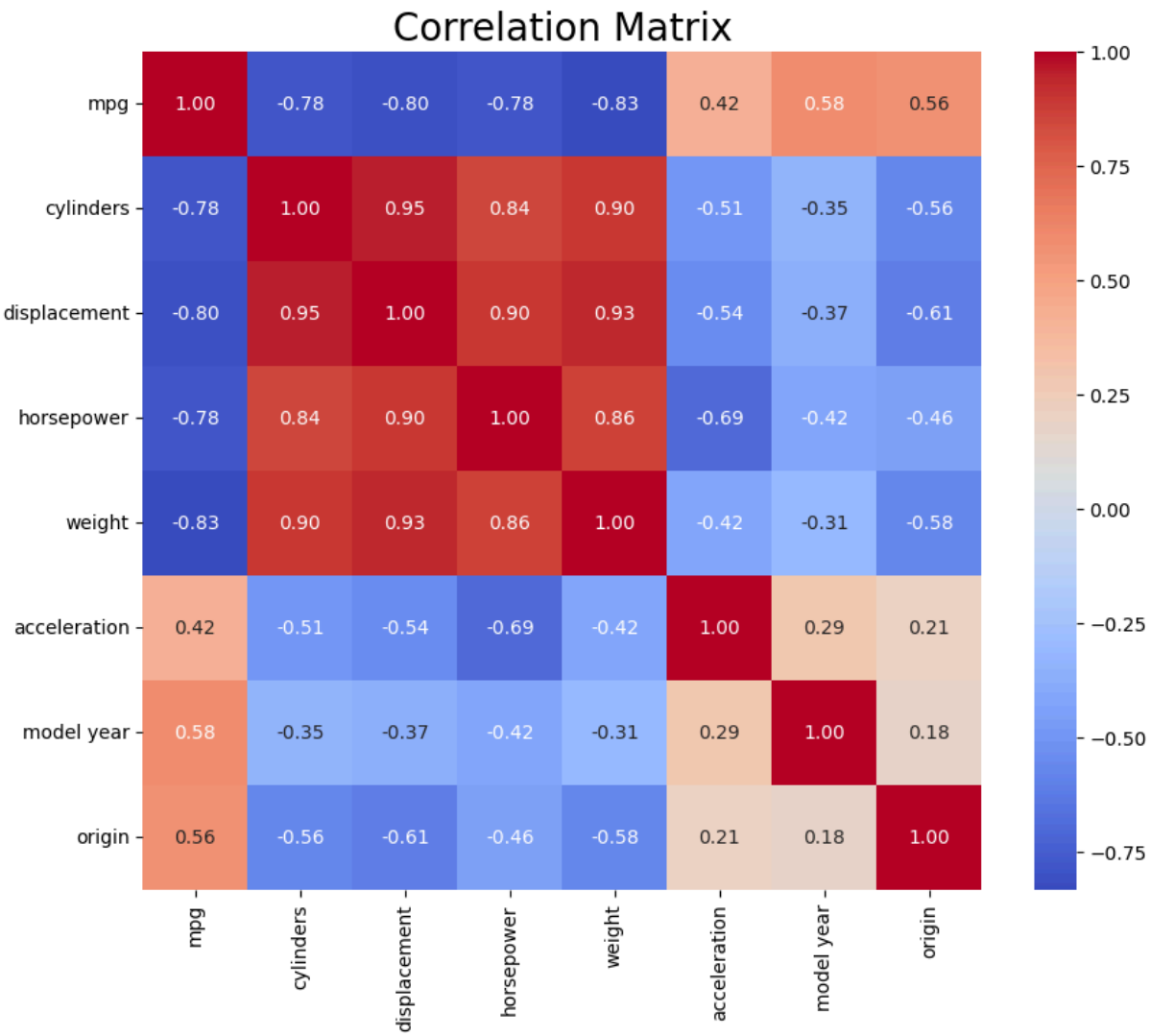


3. Use the discrete attributes as if they were continuous. For the set of all attributes in the dataset except for car-name, calculate 1) the covariance matrix and 2) the correlation matrix of these attributes. Construct a visualization of each of these matrices to more easily understand them.

In [ ]:
```
# create and plot the covariance matrix
covariance_matrix = auto_dataset.iloc[:, :-1].cov()

plt.figure(figsize=(10, 8))
sns.heatmap(covariance_matrix, annot=True, fmt=".2f", cmap='coolwarm', square=True)
plt.title('Covariance Matrix', fontsize=20)
plt.show()
```



In [ ]:
```
# create and plot a correlation matrix for numerical features
correlation_matrix = auto_dataset.iloc[:,:-1].corr()

plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap='coolwarm', square=True
plt.title('Correlation Matrix', fontsize=20)
plt.show()
```



4. If you had to remove 2 of the attributes above from the dataset based on these two matrices, which attributes would you remove and why? Explain your answer.

Based on the two plots above I would have selected two of cylinders, displacement, horsepower and weight. Looking at the pairs with the highest correlations and covariances below, I would select weight and displacement. Each of those features have the highest correlations and covariance with other features and, by my count, removing those two features would remove the maximum number of high correlation/covariance relationships.

In [163…

```
# Extract the upper triangle of the correlation matrix
corr_matrix_upper = correlation_matrix.where(np.triu(np.ones(correlation_matrix.sha

# Find the pairs with the highest correlations
highest_corr_pairs = corr_matrix_upper.unstack().sort_values(ascending=False).dropn
print("Pairs with the highest correlations:")
print(highest_corr_pairs.head())

# Extract the upper triangle of the covariance matrix
cov_matrix_upper = covariance_matrix.where(np.triu(np.ones(covariance_matrix.shape)
```

```
# Find the pairs with the highest covariances
highest_cov_pairs = cov_matrix_upper.unstack().sort_values(ascending=False).dropna(
print("\nPairs with the highest covariances:")
print(highest_cov_pairs.head())
```

```
Pairs with the highest correlations:
displacement  cylinders       0.950721
weight        displacement    0.932824
horsepower    displacement    0.897257
weight        cylinders       0.896017
              horsepower      0.864538
dtype: float64

Pairs with the highest covariances:
weight        displacement    82368.423240
              horsepower      28265.620231
horsepower    displacement     3614.033744
weight        cylinders        1290.695575
displacement  cylinders         168.623214
dtype: float64
```

5. Dimensionality Reduction

Apply Principal Components Analysis to reduce the dimensionality of the full dataset. How many dimensions does the original dataset contain? How many dimensions are obtained after PCA? How much of the variance do they explain? Include in your report the linear combinations that define the first new attribute (= component) obtained. Look at the results and elaborate on any interesting observations you can make about the results. Here are a couple tutorials plus a discussion of the mathematics of PCA that you may find helpful:

- DataCamp PCA Tutorial
- GeeksforGeeks PCA Tutorial
- Mathematics behind PCA

The original dataset contains 8 numeric features. By default, PCA will create as many components as there are features, so the PCA also has 8 components. Between all components PCA will always explain 100% of the variation. Nearly 90% of the variance was explained by the first three components. Displacement, cylinders, horsepower and mpg were the most import features for the first component.

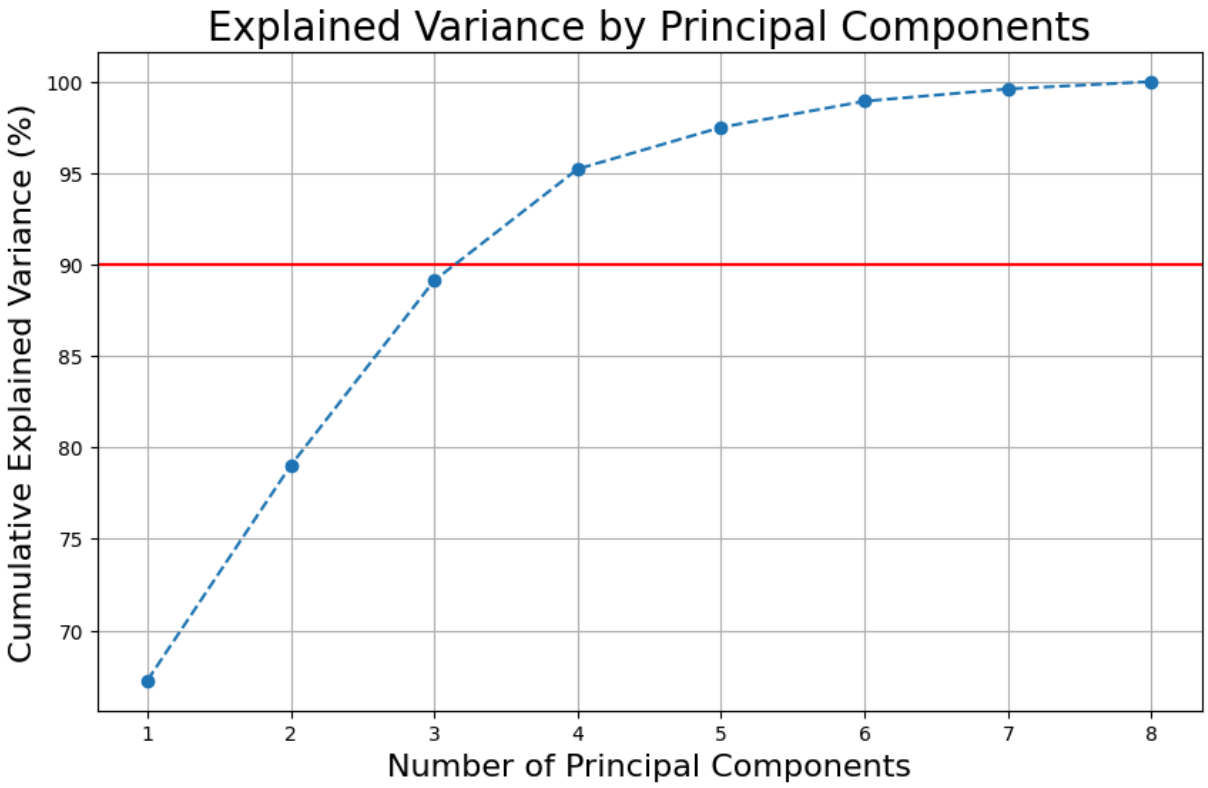
In [275…

```
# scaling the data before applying PCA
scaler=StandardScaler()
scaled_data=pd.DataFrame(scaler.fit_transform(auto_dataset.iloc[:,:-1]), columns=au
```

In [253…

```
pca = PCA()
principal_components = pca.fit_transform(scaled_data.dropna())
explained_variance = np.cumsum(pca.explained_variance_ratio_) * 100

plt.figure(figsize=(10, 6))
plt.axhline(y=90, color='r', linestyle='-')
```

```
plt.plot(range(1, len(explained_variance) + 1), explained_variance, marker='o', lin
plt.title('Explained Variance by Principal Components', fontsize=20)
plt.xlabel('Number of Principal Components', fontsize=16)
plt.ylabel('Cumulative Explained Variance (%)', fontsize=16)
plt.xticks(range(1, len(explained_variance) + 1))
plt.grid(True)
plt.show()
```



In [254…

```
# Get the feature names
feature_names = scaled_data.columns[:-1]

# Get the first principal component
first_component = pca.components_[0]

# Combine the feature names with their corresponding coefficients
first_component_features = list(zip(feature_names, first_component))

# Print the feature names and their coefficients
print("Feature names and coefficients for the first principal component:")
first_component_features.sort(key=lambda x: abs(x[1]), reverse=True)
for feature, coefficient in first_component_features:
    print(f"{feature}: {coefficient}")
```

```
Feature names and coefficients for the first principal component:
displacement: 0.4174213446321361
cylinders: 0.40303724683167397
weight: 0.4023145713065013
horsepower: 0.40116246935523425
mpg: -0.38455419069891195
acceleration: -0.26424533085494006
model year: -0.21213928312652672
```

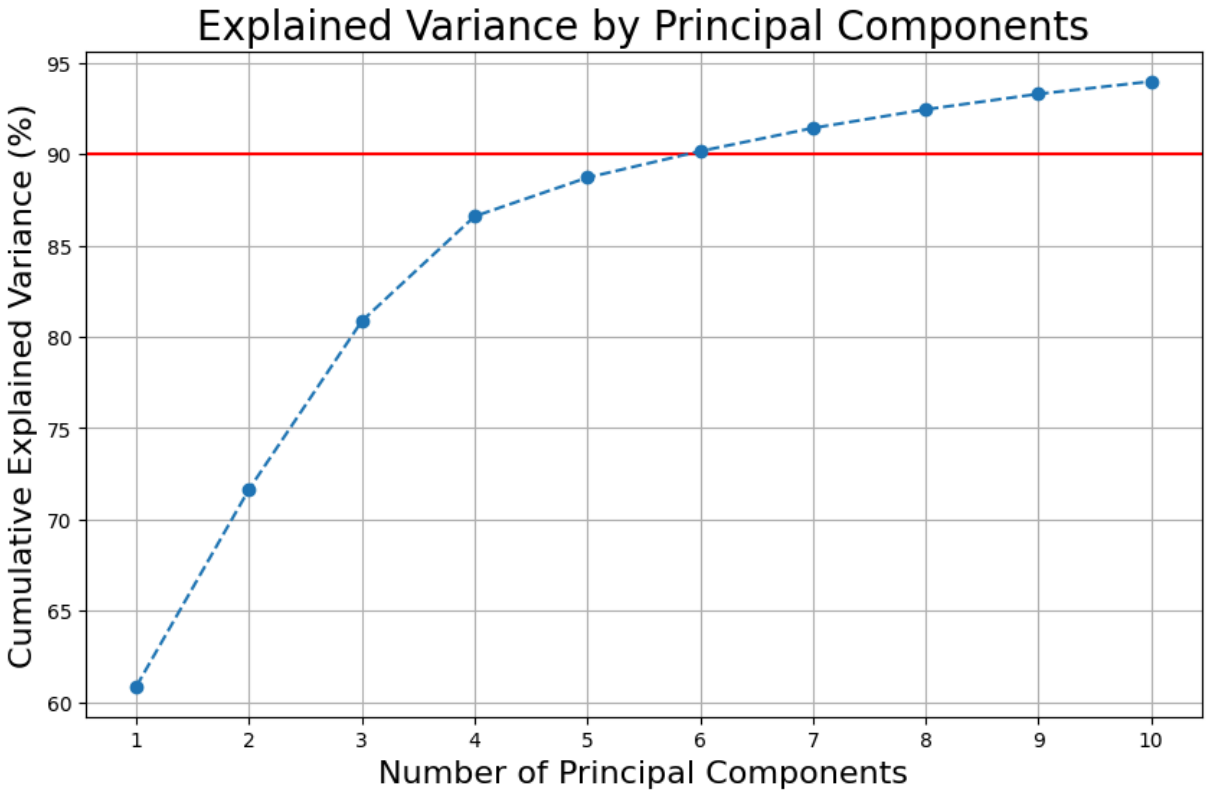
6. Attribute Transformation

Convert the car-name attribute into a discrete attribute by changing each car-name into just the car brand (e.g., toyota, ford, audi, ...). Using this modified dataset, run PCA in Python again as you did above and report any changes you observe in the results

With car brand added in it now takes 6 principal principal components to explain 90% of the data. Displacement, cylinders, horsepower and mpg were still the most import features for the first component.

In [276…
```
scaled_data['car_brand'] = auto_dataset['car name'].str.split().str[0]
scaled_data = pd.get_dummies(scaled_data, columns=['car_brand'], drop_first=True)
```

In [261…
```
pca = PCA(n_components=10)
principal_components = pca.fit_transform(scaled_data.dropna())
explained_variance = np.cumsum(pca.explained_variance_ratio_) * 100

plt.figure(figsize=(10, 6))
plt.axhline(y=90, color='r', linestyle='-')
plt.plot(range(1, len(explained_variance) + 1), explained_variance, marker='o', lin
plt.title('Explained Variance by Principal Components', fontsize=20)
plt.xlabel('Number of Principal Components', fontsize=16)
plt.ylabel('Cumulative Explained Variance (%)', fontsize=16)
plt.xticks(range(1, len(explained_variance) + 1))
plt.grid(True)
plt.show()
```



In [ ]:
```python
# Get the feature names
feature_names = scaled_data.columns[:-1]

# Get the first principal component
first_component = pca.components_[0]

# Combine the feature names with their corresponding coefficients
first_component_features = list(zip(feature_names, first_component))

# Print the 10 most important feature names (for the first component) and their coe
print("Feature names and coefficients for the first principal component:")
first_component_features.sort(key=lambda x: abs(x[1]), reverse=True)
for feature, coefficient in first_component_features[:10]:
    print(f"{feature}: {coefficient}")
```

```
Feature names and coefficients for the first principal component:
displacement: 0.39565739092274077
weight: 0.38217778790852897
cylinders: 0.3800219050972382
horsepower: 0.37369189217486903
mpg: -0.36366984950659254
origin: -0.29457529988648723
acceleration: -0.23892365433817833
model year: -0.1885157480650004
car_brand_datsun: -0.11071466275352966
car_brand_toyota: -0.10390207650563323
```