D Y PATIL
INTERNATIONAL
UNIVERSITY
AKURDI PUNE

D Y Patil International University

TC- 2

Principles of Data Science

Year - 3rd

Semester - 5th

Lab Manual

Dr. Anuj Kumar
(Main Faculty)

Mrs. Remya Praveen
(Teaching Assistant)

Index

Lab 1: Statistical Analysis of CSV data using python functions

Aim: The aim of this project is to develop a Python program that performs various statistical operations on Comma Separated Value (CSV) data without relying on external libraries, using custom functions for each operation. The program will identify categorical and numerical variables, generate a contingency table for categorical variables, calculate statistical measures for numerical variables, and categorise categorical variables based on the data type.

Objectives:

1. Load CSV data and convert it into a manageable format.
2. Identify categorical and numerical variables from the data.
3. Create a contingency table for at most two different categorical variables.
4. Calculate the mean, median, mode, variance, standard deviation, and quartile range for numerical variables.
5. Categorise categorical variables into binary, nominal, and ordinal types.

Theory :

1. Loading CSV Data: The CSV data is loaded into a list of dictionaries, where each row represents a dictionary with column names as keys and corresponding values.
2. Identifying Variable Types: The program distinguishes between categorical and numerical variables by analysing the first row of data. Categorical variables have non-numeric values, while numerical variables have numeric values.
3. Contingency Table: The contingency table is constructed by counting the occurrences of pairs of values from two categorical variables. The Counter class is utilised to achieve this.
4. Statistical Measures: For numerical variables, the program calculates mean, median, mode, variance, standard deviation, and quartile range. These statistics are computed using basic mathematical operations.
5. Categorizing Categorical Variables: The program categorizes categorical variables based on the number of unique values they have. Binary variables have two unique values; nominal variables have more than two distinctive values, and ordinal variables have ten or fewer unique values.

Program & Output:

In [1]:
```python
import itertools
from dataclasses import dataclass
from pathlib import Path
```

In [2]:
```python
def average(avg: float, new_sample: float, length: float) -> float:
    avg -= avg / length
    avg += new_sample / length
    return avg
```

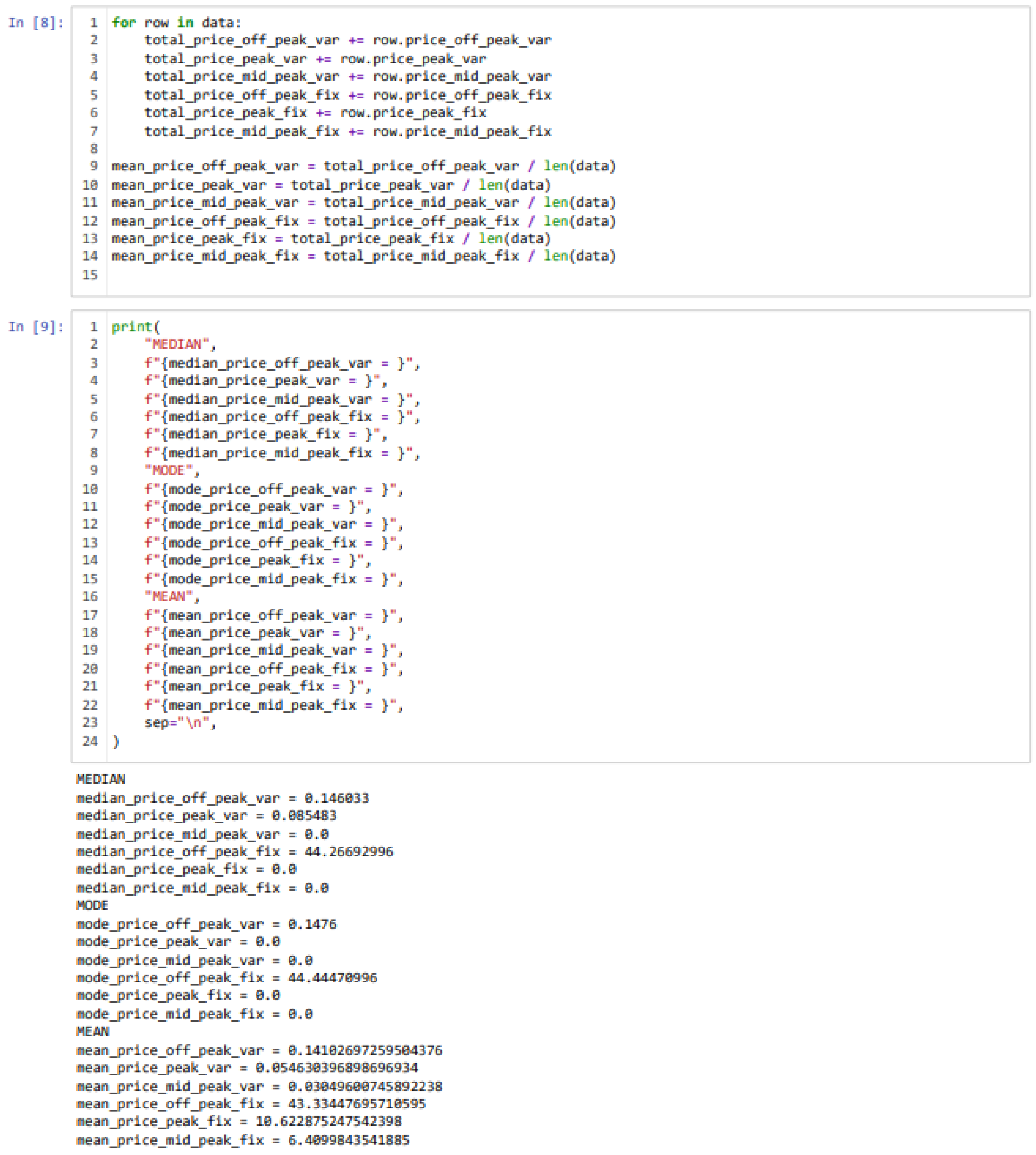
In [3]:
```python
@dataclass
class Model:
    id: str
    price_date: str
    price_off_peak_var: float
    price_peak_var: float
    price_mid_peak_var: float
    price_off_peak_fix: float
    price_peak_fix: float
    price_mid_peak_fix: float

    @staticmethod
    def from_csv(row):
        row = row.split(",")
        return Model(
            *itertools.chain([row[0], row[1]], (float(col) for col in row[2:]))  # type: ignore
        )
```

In [4]:
```python
data = [
    Model.from_csv(row) for row in Path("price_data.csv").read_text().splitlines()[1:]
]
```

In [5]:
```python
total_price_off_peak_var = 0
total_price_peak_var = 0
total_price_mid_peak_var = 0
total_price_off_peak_fix = 0
total_price_peak_fix = 0
total_price_mid_peak_fix = 0
```

In [6]:
```python
def median(collection, key):
    collection.sort(key=key)
    return key(collection[len(collection) // 2])


def mode(collection, key):
    return max(
        itertools.groupby(collection, key),
        key=lambda group: len(list(group[1])),
    )[0]
```

In [7]:
```python
median_price_off_peak_var = median(data, key=lambda row: row.price_off_peak_var)
median_price_peak_var = median(data, key=lambda row: row.price_peak_var)
median_price_mid_peak_var = median(data, key=lambda row: row.price_mid_peak_var)
median_price_off_peak_fix = median(data, key=lambda row: row.price_off_peak_fix)
median_price_peak_fix = median(data, key=lambda row: row.price_peak_fix)
median_price_mid_peak_fix = median(data, key=lambda row: row.price_mid_peak_fix)


mode_price_off_peak_var = mode(data, key=lambda row: row.price_off_peak_var)
mode_price_peak_var = mode(data, key=lambda row: row.price_peak_var)
mode_price_mid_peak_var = mode(data, key=lambda row: row.price_mid_peak_var)
mode_price_off_peak_fix = mode(data, key=lambda row: row.price_off_peak_fix)
mode_price_peak_fix = mode(data, key=lambda row: row.price_peak_fix)
mode_price_mid_peak_fix = mode(data, key=lambda row: row.price_mid_peak_fix)
```



In [8]:

```python
for row in data:
    total_price_off_peak_var += row.price_off_peak_var
    total_price_peak_var += row.price_peak_var
    total_price_mid_peak_var += row.price_mid_peak_var
    total_price_off_peak_fix += row.price_off_peak_fix
    total_price_peak_fix += row.price_peak_fix
    total_price_mid_peak_fix += row.price_mid_peak_fix

mean_price_off_peak_var = total_price_off_peak_var / len(data)
mean_price_peak_var = total_price_peak_var / len(data)
mean_price_mid_peak_var = total_price_mid_peak_var / len(data)
mean_price_off_peak_fix = total_price_off_peak_fix / len(data)
mean_price_peak_fix = total_price_peak_fix / len(data)
mean_price_mid_peak_fix = total_price_mid_peak_fix / len(data)

```

In [9]:

```python
print(
    "MEDIAN",
    f"{median_price_off_peak_var = }",
    f"{median_price_peak_var = }",
    f"{median_price_mid_peak_var = }",
    f"{median_price_off_peak_fix = }",
    f"{median_price_peak_fix = }",
    f"{median_price_mid_peak_fix = }",
    "MODE",
    f"{mode_price_off_peak_var = }",
    f"{mode_price_peak_var = }",
    f"{mode_price_mid_peak_var = }",
    f"{mode_price_off_peak_fix = }",
    f"{mode_price_peak_fix = }",
    f"{mode_price_mid_peak_fix = }",
    "MEAN",
    f"{mean_price_off_peak_var = }",
    f"{mean_price_peak_var = }",
    f"{mean_price_mid_peak_var = }",
    f"{mean_price_off_peak_fix = }",
    f"{mean_price_peak_fix = }",
    f"{mean_price_mid_peak_fix = }",
    sep="\n",
)
```

```
MEDIAN
median_price_off_peak_var = 0.146033
median_price_peak_var = 0.085483
median_price_mid_peak_var = 0.0
median_price_off_peak_fix = 44.26692996
median_price_peak_fix = 0.0
median_price_mid_peak_fix = 0.0
MODE
mode_price_off_peak_var = 0.1476
mode_price_peak_var = 0.0
mode_price_mid_peak_var = 0.0
mode_price_off_peak_fix = 44.44470996
mode_price_peak_fix = 0.0
mode_price_mid_peak_fix = 0.0
MEAN
mean_price_off_peak_var = 0.14102697259504376
mean_price_peak_var = 0.054630396898696934
mean_price_mid_peak_var = 0.03049600745892238
mean_price_off_peak_fix = 43.33447695710595
mean_price_peak_fix = 10.622875247542398
mean_price_mid_peak_fix = 6.4099843541885
```

Conclusion:

We developed a Python program to perform statistical operations on CSV data using custom functions, without external libraries. The program identified variable types, generated contingency tables for categorical data, calculated statistical measures for numerical data, and categorized categorical variables. This project effectively demonstrated that fundamental data analysis tasks can be accomplished with basic Python programming.

Lab 2: Linear Algebra

Aim: In this lab , we will perform Exploratory Data Analysis (EDA) on the Iris Dataset, a widely used dataset in data science and statistics. We will also calculate Probability Mass Functions (PMF), Probability Density Functions (PDF), and Cumulative Distribution Functions (CDF) for selected variables within the dataset.

Objectives:

1. Explore and understand the Iris Dataset.
2. Calculate and visualize the Probability Mass Function (PMF) for a specific variable.
3. Calculate and visualize the Probability Density Function (PDF) for a specific variable.
4. Calculate and visualize the Cumulative Distribution Function (CDF) for a specific variable.

Data Description

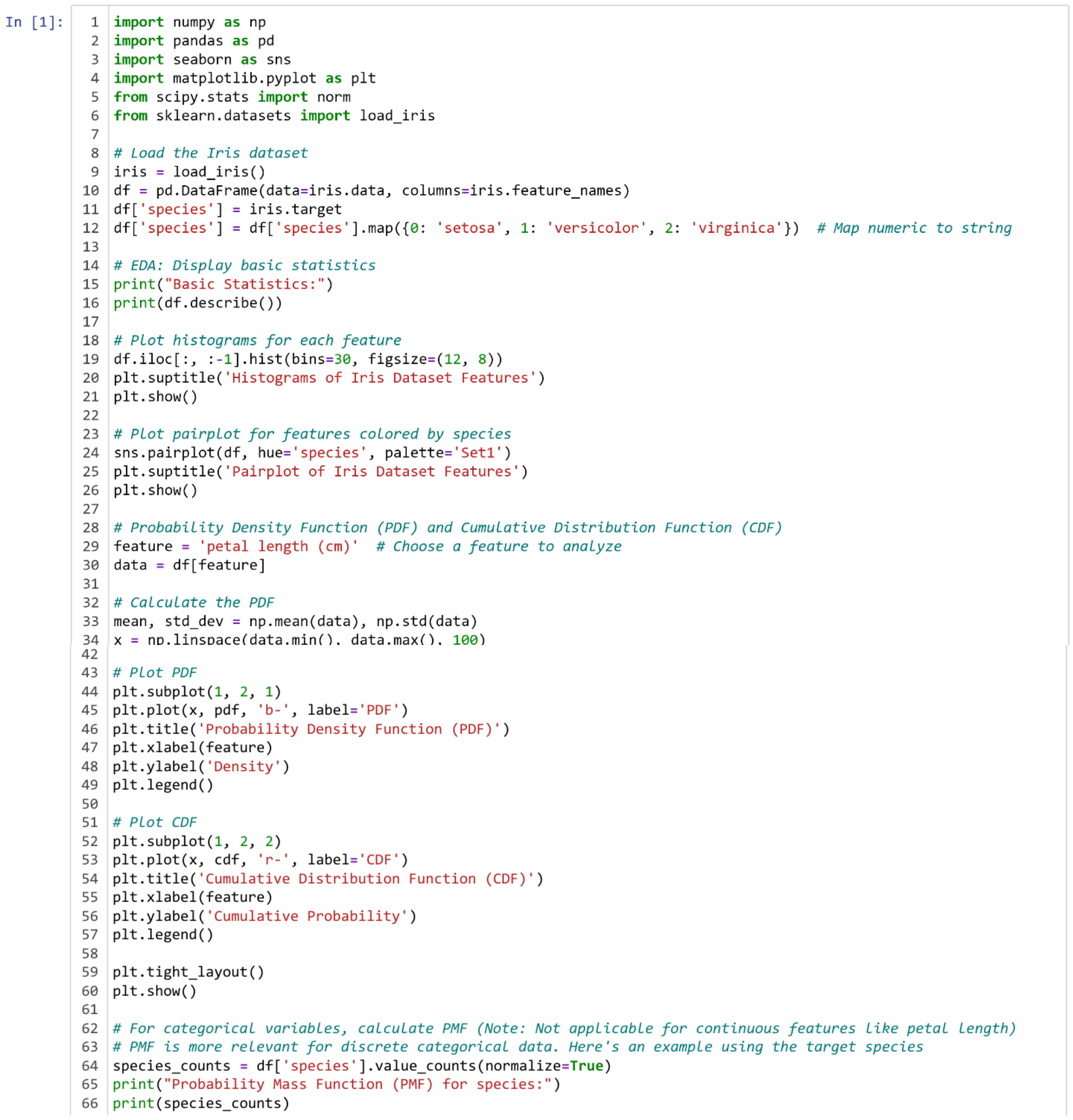
The Iris Dataset contains measurements of four features (sepal length, sepal width, petal length,
and petal width) for three species of iris flowers (setosa, versicolor, and virginica). Each species
has 50 samples, resulting in a total of 150 data points.

Theory/Procedure:

1. ***Data Loading and Exploration***
   We started by loading the Iris Dataset using Python's data manipulation libraries (e.g., Pandas) and explored its basic characteristics. This included checking for missing data, summary statistics, and the structure of the dataset.
2. ***Selecting a Variable of Interest***
   For this lab report, we focused on the "sepal length" variable from the Iris Dataset. This variable represents the sepal length of iris flowers.
3. ***Probability Mass Function (PMF)***
   We calculated the Probability Mass Function (PMF) for the "sepal length" variable. The PMF provides the probability distribution of discrete values. We used Python libraries like NumPy and Matplotlib to create a PMF plot.
4. ***Probability Density Function (PDF)***
   Next, we calculated the Probability Density Function (PDF) for the "sepal length" variable. The PDF provides the probability distribution of continuous values. We used the Seaborn library for creating a PDF plot.

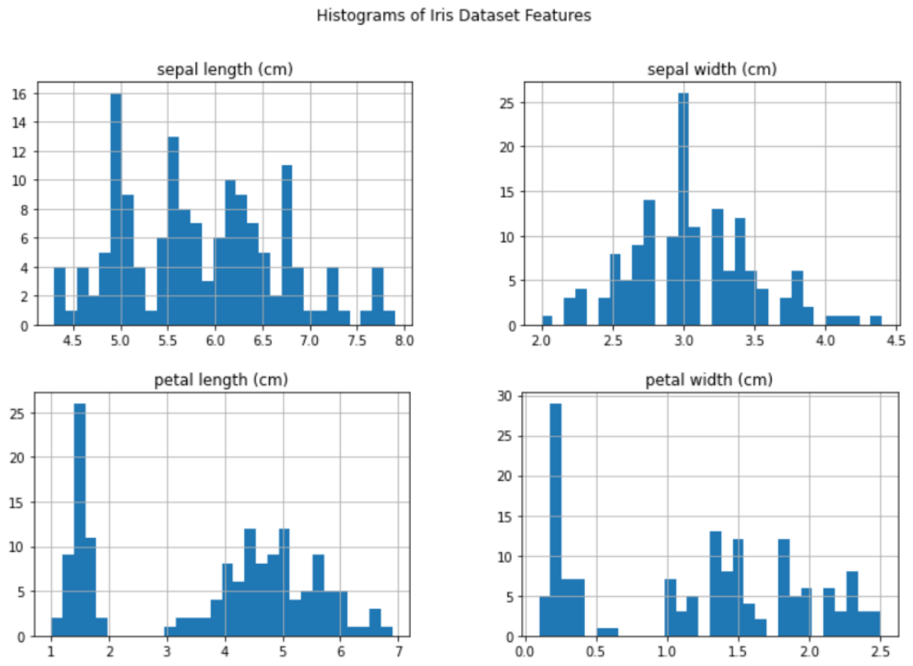
5. ***Cumulative Distribution Function (CDF)***
   Finally, we computed the Cumulative Distribution Function (CDF) for the "sepal length" variable. The CDF represents the probability that a random variable takes on a value less than or equal to a specific value. We used Python libraries to create a CDF plot.

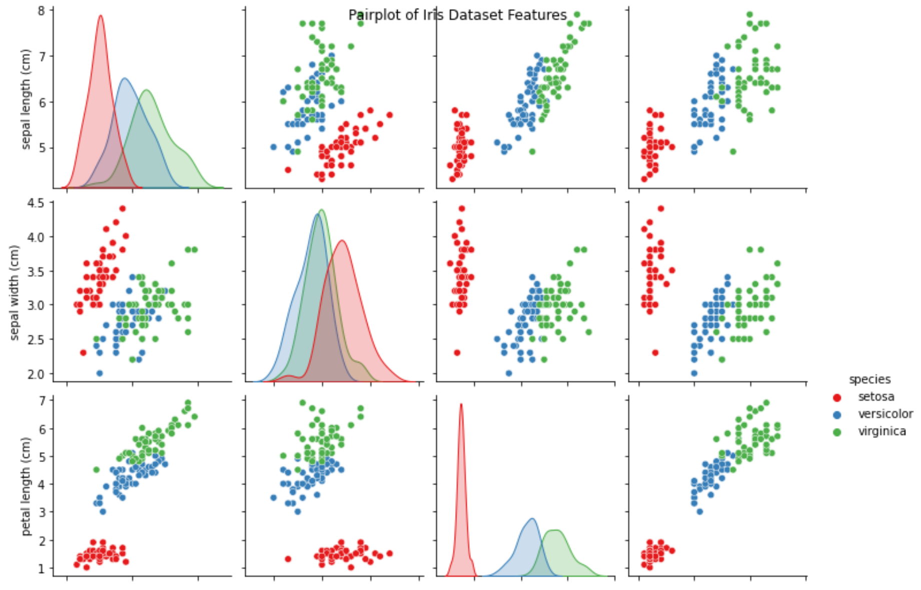
Code & Output:

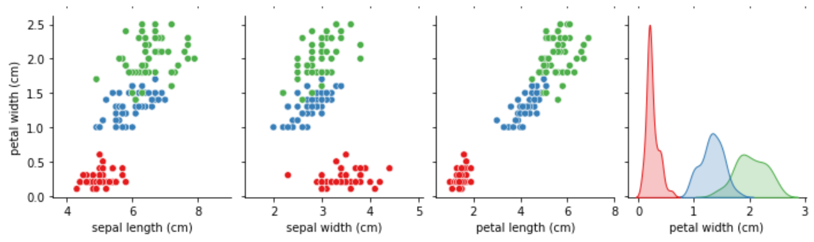


```
In [1]:
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from scipy.stats import norm
from sklearn.datasets import load_iris

# Load the Iris dataset
iris = load_iris()
df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
df['species'] = iris.target
df['species'] = df['species'].map({0: 'setosa', 1: 'versicolor', 2: 'virginica'})  # Map numeric to string

# EDA: Display basic statistics
print("Basic Statistics:")
print(df.describe())

# Plot histograms for each feature
df.iloc[:, :-1].hist(bins=30, figsize=(12, 8))
plt.suptitle('Histograms of Iris Dataset Features')
plt.show()

# Plot pairplot for features colored by species
sns.pairplot(df, hue='species', palette='Set1')
plt.suptitle('Pairplot of Iris Dataset Features')
plt.show()

# Probability Density Function (PDF) and Cumulative Distribution Function (CDF)
feature = 'petal length (cm)'  # Choose a feature to analyze
data = df[feature]

# Calculate the PDF
mean, std_dev = np.mean(data), np.std(data)
x = np.linspace(data.min(), data.max(), 100)

# Plot PDF
plt.subplot(1, 2, 1)
plt.plot(x, pdf, 'b-', label='PDF')
plt.title('Probability Density Function (PDF)')
plt.xlabel(feature)
plt.ylabel('Density')
plt.legend()

# Plot CDF
plt.subplot(1, 2, 2)
plt.plot(x, cdf, 'r-', label='CDF')
plt.title('Cumulative Distribution Function (CDF)')
plt.xlabel(feature)
plt.ylabel('Cumulative Probability')
plt.legend()

plt.tight_layout()
plt.show()

# For categorical variables, calculate PMF (Note: Not applicable for continuous features like petal length)
# PMF is more relevant for discrete categorical data. Here's an example using the target species
species_counts = df['species'].value_counts(normalize=True)
print("Probability Mass Function (PMF) for species:")
print(species_counts)
```

```
Basic Statistics:
       sepal length (cm)  sepal width (cm)  petal length (cm)  \
count         150.000000        150.000000         150.000000
mean            5.843333          3.057333           3.758000
std             0.828066          0.435866           1.765298
min             4.300000          2.000000           1.000000
25%             5.100000          2.800000           1.600000
50%             5.800000          3.000000           4.350000
75%             6.400000          3.300000           5.100000
max             7.900000          4.400000           6.900000

       petal width (cm)
count        150.000000
mean           1.199333
std            0.762238
min            0.100000
25%            0.300000
50%            1.300000
75%            1.800000
```







Conclusion:
In this lab report, we conducted Exploratory Data Analysis (EDA) on the Iris Dataset and calculated Probability Mass Functions (PMF), Probability Density Functions (PDF), and Cumulative Distribution Functions (CDF) for the "sepal length" variable.

Lab 3: Types of Distribution

Aim: To Perform Different Types of Distribution and Visualize by creating a Histogram using Matplotlib Library .

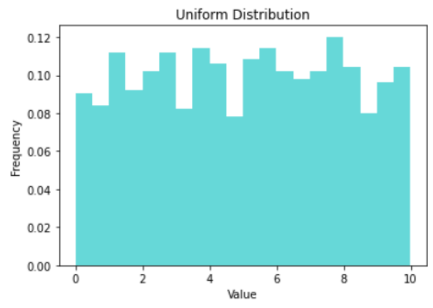
Theory:
Types of Distribution :
1. Uniform Distribution :
A uniform distribution is a probability distribution where all outcomes have equal chances of occurring. In this case, the uniform distribution is defined between 0 and 1, meaning any value between 0 and 1 is equally likely.
2. Normal Distribution:
A normal distribution (also known as Gaussian distribution) is a continuous probability distribution characterized by its bell-shaped curve. It is fully defined by its mean (μ) and standard deviation (σ).
3. Exponential Distribution:
The exponential distribution describes the time between events in a Poisson process. It is characterized by a rate parameter (λ), which determines the average rate of events per unit time.
4. Binomial Distribution:
The binomial distribution describes the number of successes (binary outcomes) in a fixed number of independent Bernoulli trials. It is characterized by two parameters: the number of trials (n) and the probability of success (p).

5. Poisson Distribution:

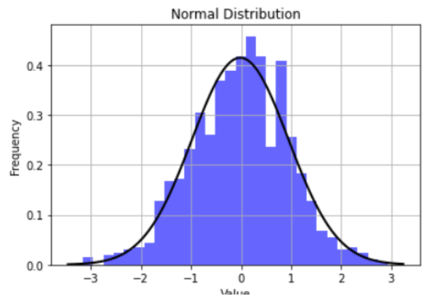
The Poisson distribution describes the number of events occurring in a fixed interval of time or space. It is characterized by a rate parameter (λ), which represents the average rate of events.

Code & Output:

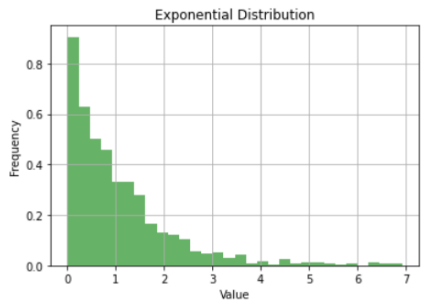
```
In [7]:
import matplotlib.pyplot as plt
import numpy as np

# Lower Boundry
# Upper Boundry
data = np.random.uniform(0, 10, 1000)
plt.hist(data, bins=20, density=True, alpha=0.6, color='c')
plt.title("Uniform Distribution")
plt.xlabel("Value")
plt.ylabel("Frequency")
plt.show()
```



```
In [9]:
import matplotlib.pyplot as plt
import numpy as np

data = np.random.normal(loc=0, scale=1, size=1000)
plt.hist(data, bins=30, density=True, alpha=0.6, color='b')

mean = np.mean(data)
std_dev = np.std(data)
xmin, xmax = plt.xlim()
x = np.linspace(xmin, xmax, 100)
p = (1 / (std_dev * np.sqrt(2 * np.pi))) * np.exp(-0.5 * ((x - mean) / std_dev) ** 2)
plt.plot(x, p, 'k', linewidth=2)

plt.title("Normal Distribution")
plt.xlabel("Value")
plt.ylabel("Frequency")
plt.grid(True)
plt.show()

```

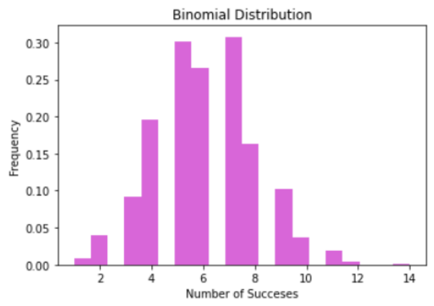


```
In [8]:
import matplotlib.pyplot as plt
import numpy as np

# Average value
data = np.random.exponential(scale=1.0, size=1000)
plt.hist(data, bins=30, density=True, alpha=0.6, color='g')
plt.title("Exponential Distribution")
plt.xlabel("Value")
plt.ylabel("Frequency")
plt.grid(True)
plt.show()
```

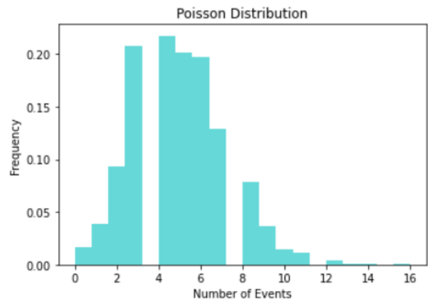


In [5]:
```python
import matplotlib.pyplot as plt
import numpy as np

# Number of trials
# Probability of success
# Number of samples
data = np.random.binomial(20, 0.3, 1000)
plt.hist(data, bins=20, density=True, alpha=0.6, color='m')
plt.title("Binomial Distribution")
plt.xlabel("Number of Succeses")
plt.ylabel("Frequency")
plt.show
```

Out[5]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



In [11]:
```python
import matplotlib.pyplot as plt
import numpy as np

data = np.random.poisson (5, 1000)
plt.hist(data, bins=20, density=True, alpha=0.6, color='c')
plt.title("Poisson Distribution")
plt.xlabel("Number of Events")
plt.ylabel("Frequency")
plt.show()
```



Conclusion:

In this lab, we explored various data distributions and visualized them using histograms with the Matplotlib library. This allowed us to observe key characteristics such as skewness and modality in different datasets. The exercise highlighted the importance of data visualization for understanding statistical properties, enhancing our analytical skills and providing a solid foundation for further studies in data science.

Lab 4: EDA Analysis on Titanic Dataset

Aim: Perform EDA Analysis on Titanic Dataset.

Objectives:

– The objective of this analysis is to conduct an in-depth Exploratory Data Analysis (EDA) on the Titanic dataset.
– The goals are to understand the characteristics of the passengers, explore relationships between different variables, and extract meaningful insights to understand the factors influencing survival rates.

Theory:

Exploratory Data Analysis (EDA) is a critical step in understanding any dataset. It involves summarizing the main characteristics of the data, often with visual methods. EDA helps to identify patterns, test assumptions, and check for outliers and anomalies in the data. In this analysis, we will use Python libraries like Pandas, NumPy, and Matplotlib to perform EDA on the Titanic dataset.

Steps:

Step 1: Import Libraries
Step 2: Load the Titanic Dataset
Step 3: Explore the Dataset

- Display the first few rows of the dataset
- Summary Statistics
- Check for Missing Values

Step 4: Data Cleaning and Preprocessing (if necessary)

- Handle missing values (e.g., fill with mean, median, or drop rows/columns)
- Convert categorical variables into numerical format if needed
- Drop unnecessary columns

Step 5: Data Visualization

- Univariate Analysis
- Histogram for Age
- Countplot for Survival
- Bivariate Analysis
- Survival Rate by PClass
- Survival Rate by Sex

Step 6: Advanced Data Visualization :

- Heatmap for Correlation Matrix

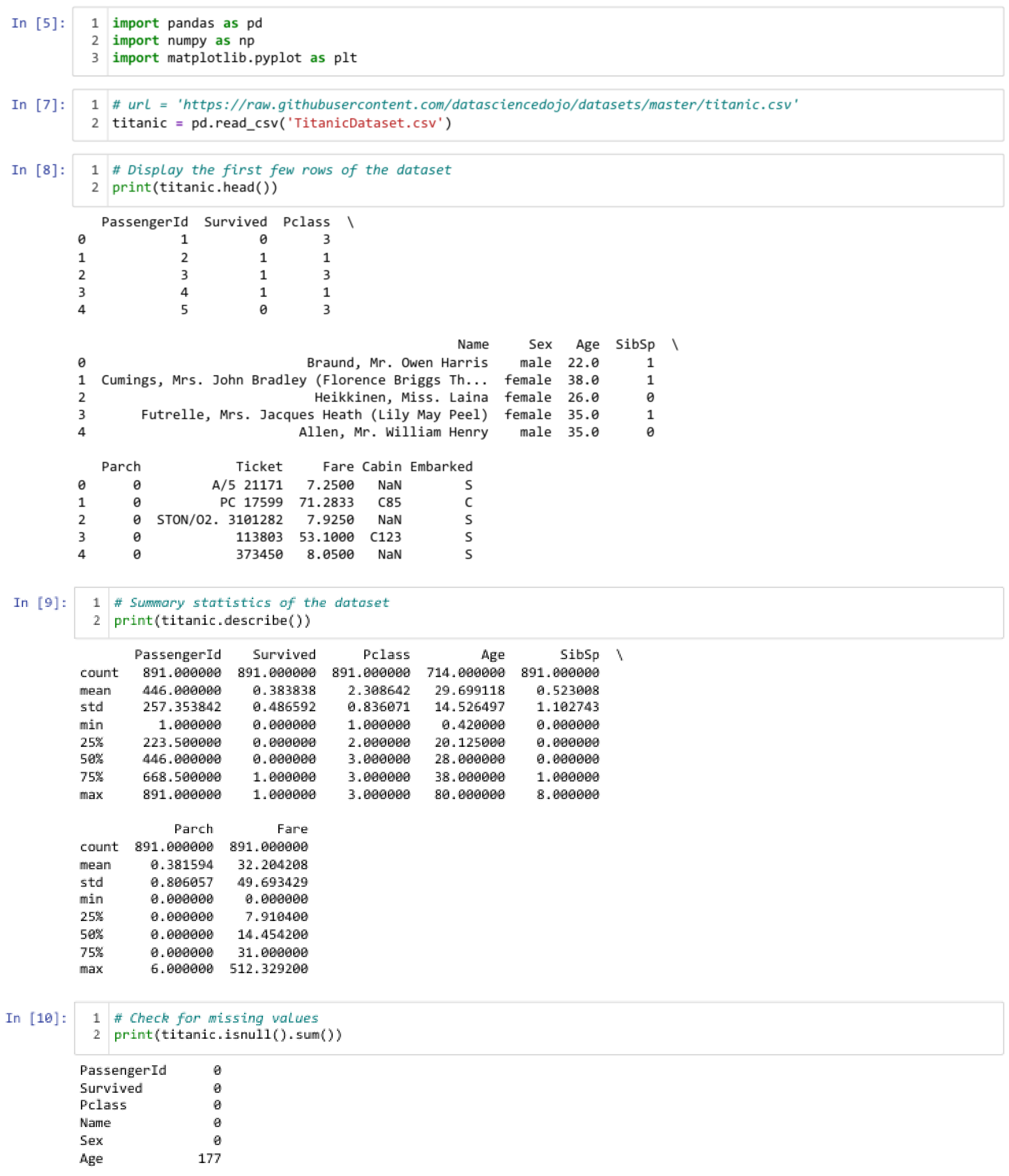
Step 7: Insights and Interpretation:

- Analyse the visualizations to draw meaningful insights about survival patterns.
- Explore correlations between variables and identify interesting trends.

Step 8: Conclusion

- Summarize the key findings and insights derived from the analysis.
- Discuss the implications of these findings in the context of the Titanic dataset.

Code & Output:



In [5]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [7]:
```python
# url = 'https://raw.githubusercontent.com/datasciencedojo/datasets/master/titanic.csv'
titanic = pd.read_csv('TitanicDataset.csv')
```

In [8]:
```python
# Display the first few rows of the dataset
print(titanic.head())
```

```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0      1
2                             Heikkinen, Miss. Laina  female  26.0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                           Allen, Mr. William Henry    male  35.0      0

   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
3      0            113803  53.1000  C123        S
4      0            373450   8.0500   NaN        S
```

In [9]:
```python
# Summary statistics of the dataset
print(titanic.describe())
```

```
       PassengerId    Survived      Pclass         Age       SibSp  \
count   891.000000  891.000000  891.000000  714.000000  891.000000
mean    446.000000    0.383838    2.308642   29.699118    0.523008
std     257.353842    0.486592    0.836071   14.526497    1.102743
min       1.000000    0.000000    1.000000    0.420000    0.000000
25%     223.500000    0.000000    2.000000   20.125000    0.000000
50%     446.000000    0.000000    3.000000   28.000000    0.000000
75%     668.500000    1.000000    3.000000   38.000000    1.000000
max     891.000000    1.000000    3.000000   80.000000    8.000000

            Parch        Fare
count  891.000000  891.000000
mean     0.381594   32.204208
std      0.806057   49.693429
min      0.000000    0.000000
25%      0.000000    7.910400
50%      0.000000   14.454200
75%      0.000000   31.000000
max      6.000000  512.329200
```

In [10]:
```python
# Check for missing values
print(titanic.isnull().sum())
```

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
```

In [11]:
```python
# Handle missing values
titanic['Age'].fillna(titanic['Age'].median(), inplace=True)
titanic['Embarked'].fillna(titanic['Embarked'].mode()[0], inplace=True)
```

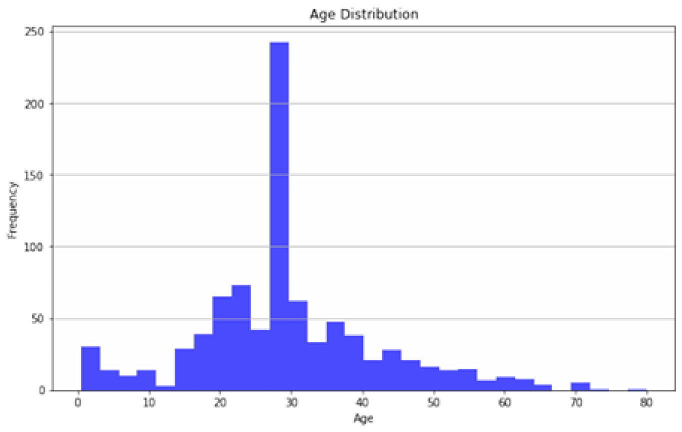
In [12]:
```python
# Drop the Cabin column as it has too many missing values
titanic.drop('Cabin', axis=1, inplace=True)
```

In [13]:
```python
# Convert categorical variables into numerical format
titanic['Sex'] = titanic['Sex'].map({'male': 0, 'female': 1})
titanic['Embarked'] = titanic['Embarked'].map({'S': 0, 'C': 1, 'Q': 2})
```

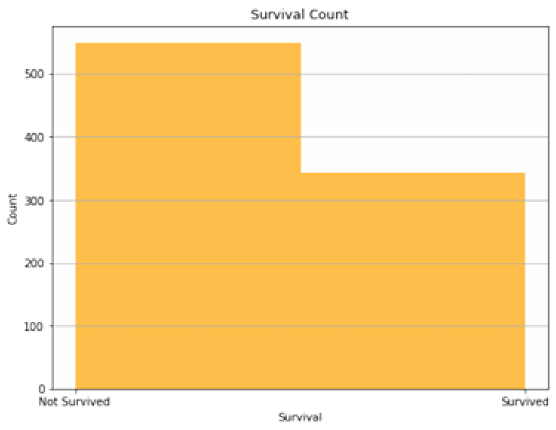
In [14]:
```python
# Check the cleaned data
print(titanic.isnull().sum())
```

```
PassengerId    0
Survived       0
Pclass         0
Name           0
Sex            0
Age            0
SibSp          0
Parch          0
Ticket         0
Fare           0
Embarked       0
dtype: int64
```

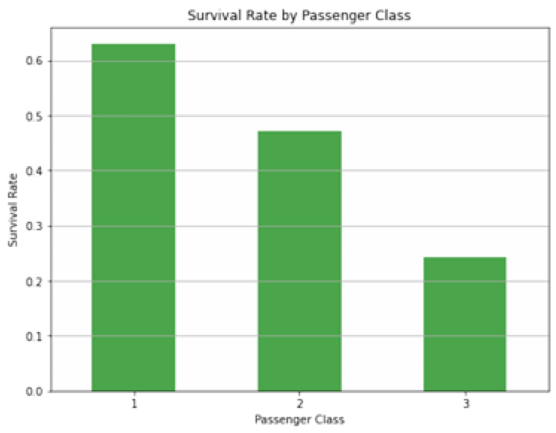
In [15]:
```python
# Univariate Analysis: Age Distribution
plt.figure(figsize=(10,6))
plt.hist(titanic['Age'], bins=30, color='blue', alpha=0.7)
plt.title('Age Distribution')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.grid(axis='y')
plt.show()
```



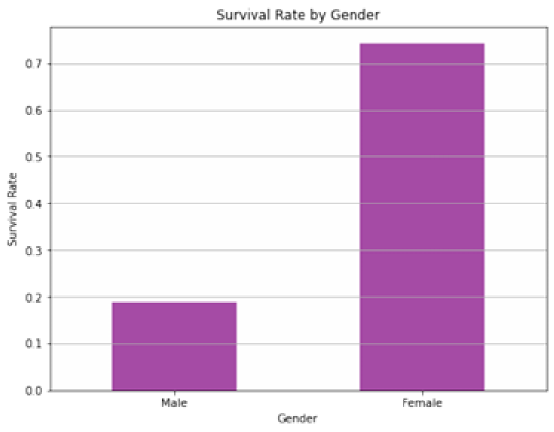
In [16]:
```python
# Countplot for Survival
plt.figure(figsize=(8,6))
plt.hist(titanic['Survived'], bins=2, color='orange', alpha=0.7)
plt.title('Survival Count')
plt.xticks([0, 1], ['Not Survived', 'Survived'])
plt.xlabel('Survival')
plt.ylabel('Count')
plt.grid(axis='y')
plt.show()
```



In [17]:
```python
# Bivariate Analysis: Survival by Passenger Class
plt.figure(figsize=(8,6))
pclass_survival = titanic.groupby('Pclass')['Survived'].mean()
pclass_survival.plot(kind='bar', color='green', alpha=0.7)
plt.title('Survival Rate by Passenger Class')
plt.xlabel('Passenger Class')
plt.ylabel('Survival Rate')
plt.xticks(rotation=0)
plt.grid(axis='y')
plt.show()
```



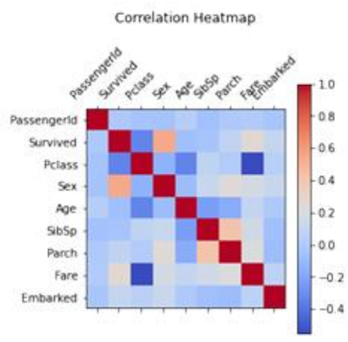
In [18]:
```python
# Bivariate Analysis: Survival by Gender
plt.figure(figsize=(8,6))
sex_survival = titanic.groupby('Sex')['Survived'].mean()
sex_survival.plot(kind='bar', color='purple', alpha=0.7)
plt.title('Survival Rate by Gender')
plt.xlabel('Gender')
plt.ylabel('Survival Rate')
plt.xticks([0, 1], ['Male', 'Female'], rotation=0)
plt.grid(axis='y')
plt.show()
```



In [19]:
```python
# Correlation Matrix Heatmap
plt.figure(figsize=(10, 6))
```

Out[19]:
```
<Figure size 720x432 with 0 Axes>

<Figure size 720x432 with 0 Axes>
```

In [21]:
```python
 # Select only numeric columns for correlation
numeric_cols = titanic.select_dtypes(include=[np.number])

# Create the correlation matrix
correlation_matrix = numeric_cols.corr()

# Plot the heatmap
plt.matshow(correlation_matrix, fignum=1, cmap='coolwarm')
plt.title('Correlation Heatmap', pad=20)
plt.xticks(range(len(numeric_cols.columns)), numeric_cols.columns, rotation=45)
plt.yticks(range(len(numeric_cols.columns)), numeric_cols.columns)
plt.colorbar()
plt.show()
```



Conclusion:
In this analysis, we explored the Titanic dataset using various visualization techniques. We observed that survival rates differed significantly based on passenger class and gender. Additionally, we analyzed the age distribution of survivors and non-survivors. Through this EDA, we gained valuable insights into the factors influencing survival on the Titanic.

Lab 5: Hypothesis Testing using Iris Dataset

Aim: The purpose of this lab is to introduce hypothesis testing using statistical methods in python, focusing on hypothesis tests like the t-test, ANOVA, and chi-square test. By applying these techniques to the well-known Iris dataset, you will learn how to test assumptions about population means and relationships between categorical variables.

Theory:
Introduction to Hypothesis Testing
Hypothesis testing is a statistical method used to make inferences or draw conclusions about a
population based on a sample of data. It helps in determining whether there is enough evidence
in a sample of data to infer that a certain condition is true for the entire population.
Key concepts in hypothesis testing:
- Null Hypothesis (H_0): The statement that there is no effect or no difference. It is what you try to disprove or reject.
- Alternative Hypothesis (H_1): The statement that there is an effect or a difference. It is what you want to prove.
- p-value: The probability of observing the results if the null hypothesis is true. A small p-value (< 0.05) indicates strong evidence against the null hypothesis.
- Significance Level (α): A threshold (commonly 0.05) used to decide whether to reject the null hypothesis.

- Test Statistic: A value calculated from the data used to determine whether to reject the null hypothesis.

Dataset: The Iris Dataset

The Iris dataset is one of the most famous datasets in the field of machine learning. It consists of
150 observations, with the following features:

- Sepal length (cm)
- Sepal width (cm)
- Petal length (cm)
- Petal width (cm)
- Species (Iris-setosa, Iris-versicolor, and Iris-virginica)

Each observation represents a different iris flower from one of the three species, and the dataset
contains measurements for each flower's sepals and petals.

Problem 1: Two-Sample t-test

Objective: To test if there is a significant difference in the sepal lengths between the species Iris-setosa and Iris-versicolor.

Hypotheses:

- Null Hypothesis (H_0): There is no significant difference between the mean sepal lengths of setosa and versicolor species. ($\mu_1 = \mu_2$)
- Alternative Hypothesis (H_1): There is a significant difference between the mean sepal lengths of setosa and versicolor species. ($\mu_1 \neq \mu_2$)

Steps:

1. Select the data for the two species (setosa and versicolor).
2. Calculate the mean and standard deviation of the sepal lengths for both species.
3. Use a two-sample t-test to determine if the difference in means is statistically significant.
4. Calculate the t-statistic and p-value.
5. Compare the p-value with the significance level ($\alpha = 0.05$) to decide whether to reject or fail to reject the null hypothesis.

Interpretation:

- If the p-value is less than 0.05, reject the null hypothesis, meaning there is a statistically significant difference in sepal lengths between setosa and versicolor.
- If the p-value is greater than 0.05, fail to reject the null hypothesis, meaning there is no significant difference in sepal lengths.

Problem 2: One-Way ANOVA (Analysis of Variance)

Objective: To test if there is a significant difference in the sepal lengths across all three species (setosa, versicolor, and virginica).

Hypotheses:

– Null Hypothesis (H_0): The means of sepal lengths are equal for all species. ($\mu_1 = \mu_2 = \mu_3$)

– Alternative Hypothesis (H_1): At least one species has a different mean sepal length. ($\mu_1 \neq \mu_2$ or $\mu_1 \neq \mu_3$, etc.)

Steps:

1. Group the data by species and calculate the means and standard deviations for sepal lengths.
2. Use the one-way ANOVA test to compare the means of sepal lengths across the three species.
3. Calculate the F-statistic and p-value.
4. Compare the p-value with the significance level ($\alpha = 0.05$) to decide whether to reject or fail to reject the null hypothesis.

Interpretation:

– If the p-value is less than 0.05, reject the null hypothesis, indicating that at least one species has a significantly different mean sepal length.

– If the p-value is greater than 0.05, fail to reject the null hypothesis, suggesting that the means are not significantly different across species.

Problem 3: Chi-Square Test for Independence

Objective: To test whether there is a relationship between species and different categories of sepal width (e.g., narrow, medium, wide).

Hypotheses:

– Null Hypothesis (H_0): There is no relationship between species and sepal width categories (i.e., the two variables are independent).

– Alternative Hypothesis (H_1): There is a relationship between species and sepal width categories (i.e., the two variables are dependent).

Steps:

1. Divide the sepal width data into categories (e.g., narrow, medium, wide).
2. Create a contingency table showing the frequency of species across these categories.
3. Perform a chi-square test to determine if the distribution of species is independent of sepal width categories.
4. Calculate the chi-square statistic and p-value.

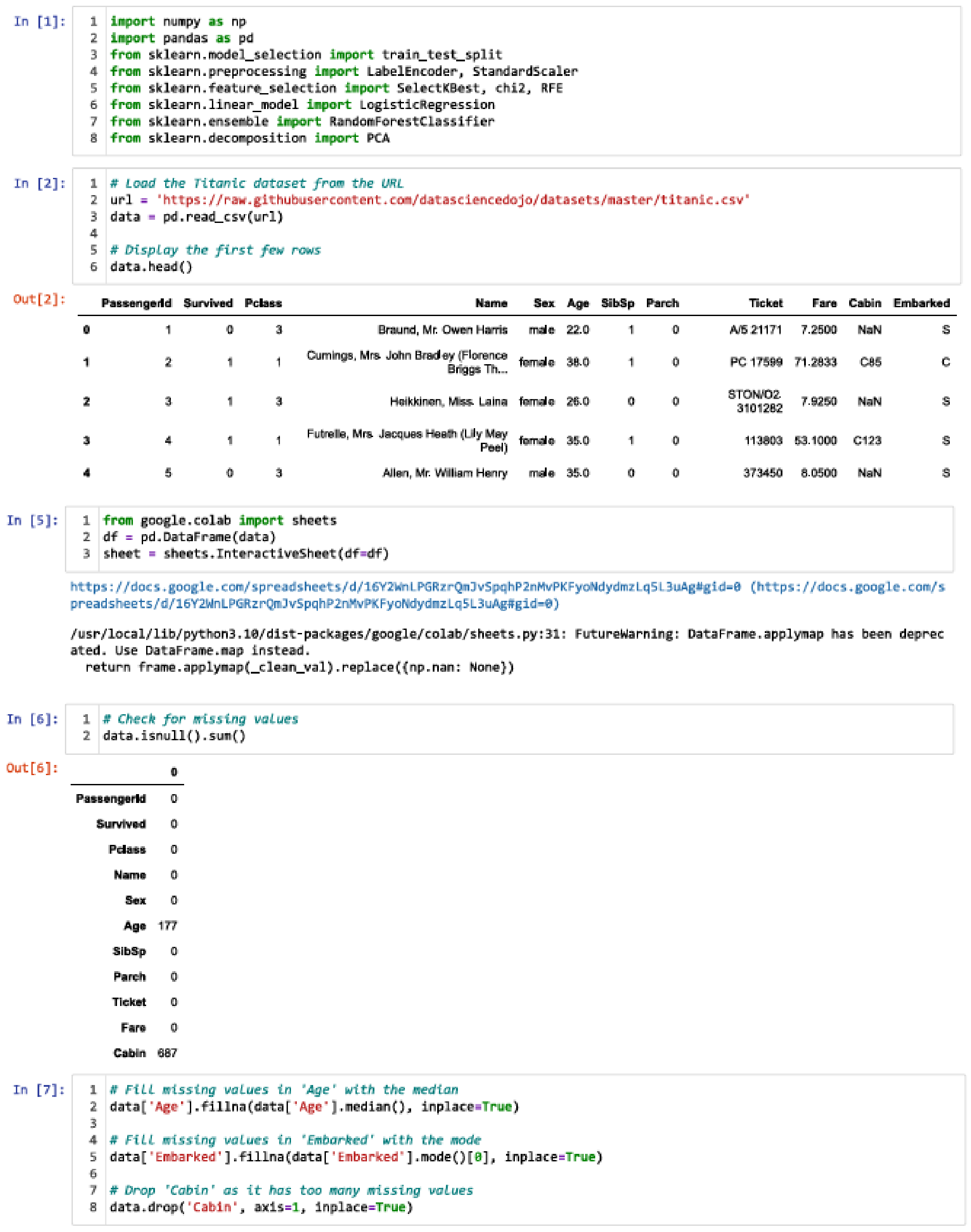
5. Compare the p-value with the significance level (α = 0.05) to decide whether to reject or fail to reject the null hypothesis.

Interpretation:

– If the p-value is less than 0.05, reject the null hypothesis, indicating that sepal width and species are related (dependent).

If the p-value is greater than 0.05, fail to reject the null hypothesis, suggesting that sepal width and species are independent.

Code & Output:



In [1]:
```python
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.feature_selection import SelectKBest, chi2, RFE
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.decomposition import PCA
```

In [2]:
```python
# Load the Titanic dataset from the URL
url = 'https://raw.githubusercontent.com/datasciencedojo/datasets/master/titanic.csv'
data = pd.read_csv(url)

# Display the first few rows
data.head()
```

Out[2]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

In [5]:
```python
from google.colab import sheets
df = pd.DataFrame(data)
sheet = sheets.InteractiveSheet(df=df)
```

```
https://docs.google.com/spreadsheets/d/16Y2WnLPGRzrQmJvSpqhP2nMvPKFyoNdydmzLq5L3uAg#gid=0 (https://docs.google.com/s
preadsheets/d/16Y2WnLPGRzrQmJvSpqhP2nMvPKFyoNdydmzLq5L3uAg#gid=0)

/usr/local/lib/python3.10/dist-packages/google/colab/sheets.py:31: FutureWarning: DataFrame.applymap has been deprec
ated. Use DataFrame.map instead.
  return frame.applymap(_clean_val).replace({np.nan: None})
```

In [6]:
```python
# Check for missing values
data.isnull().sum()
```

Out[6]:

	0
PassengerId	0
Survived	0
Pclass	0
Name	0
Sex	0
Age	177
SibSp	0
Parch	0
Ticket	0
Fare	0
Cabin	687

In [7]:
```python
# Fill missing values in 'Age' with the median
data['Age'].fillna(data['Age'].median(), inplace=True)

# Fill missing values in 'Embarked' with the mode
data['Embarked'].fillna(data['Embarked'].mode()[0], inplace=True)

# Drop 'Cabin' as it has too many missing values
data.drop('Cabin', axis=1, inplace=True)
```

```
<ipython-input-7-5d2a83a8e50d>:2: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series thro
ugh chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which
we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or
df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.


  data['Age'].fillna(data['Age'].median(), inplace=True)
<ipython-input-7-5d2a83a8e50d>:5: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series thro
ugh chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which
we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or
df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.


  data['Embarked'].fillna(data['Embarked'].mode()[0], inplace=True)
```

In [8]:

```python
# Convert 'Sex' column using Label Encoding
label_encoder = LabelEncoder()
data['Sex'] = label_encoder.fit_transform(data['Sex'])

# Convert 'Embarked' using One-Hot Encoding
data = pd.get_dummies(data, columns=['Embarked'], drop_first=True)

# Check the first few rows to verify encoding
data.head()
```

Out[8]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked_Q	Embarked_S
0	1	0	3	Braund, Mr. Owen Harris	1	22.0	1	0	A/5 21171	7.2500	False	True
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	0	38.0	1	0	PC 17599	71.2833	False	False
2	3	1	3	Heikkinen, Miss. Laina	0	26.0	0	0	STON/O2. 3101282	7.9250	False	True
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	0	35.0	1	0	113803	53.1000	False	True
4	5	0	3	Allen, Mr. William Henry	1	35.0	0	0	373450	8.0500	False	True

In [9]:

```python
# Create 'FamilySize' feature
data['FamilySize'] = data['SibSp'] + data['Parch'] + 1

# Create 'IsAlone' feature
data['IsAlone'] = np.where(data['FamilySize'] == 1, 1, 0)

# Check the new features
data[['SibSp', 'Parch', 'FamilySize', 'IsAlone']].head()
```

Out[9]:

	SibSp	Parch	FamilySize	IsAlone
0	1	0	2	0
1	1	0	2	0
2	0	0	1	1
3	1	0	2	0
4	0	0	1	1

In [10]:

```python
scaler = StandardScaler()
data[['Age', 'Fare', 'FamilySize']] = scaler.fit_transform(data[['Age', 'Fare', 'FamilySize']])

# Check the scaled features
data[['Age', 'Fare', 'FamilySize']].head()
```

Out[10]:

	Age	Fare	FamilySize
0	-0.565736	-0.502445	0.059160
1	0.663861	0.786845	0.059160
2	-0.258337	-0.488854	-0.560975
3	0.433312	0.420730	0.059160
4	0.433312	-0.486337	-0.560975

In [11]:

```python
# Separate features (X) and target (y)
X = data.drop(['PassengerId', 'Name', 'Ticket', 'Survived'], axis=1)
y = data['Survived']
```

In [13]:

```python
# Ensure all values in X are non-negative
X[X < 0] = 0

# Use SelectKBest to select top 5 features based on chi-squared test
selector = SelectKBest(score_func=chi2, k=5)
X_new = selector.fit_transform(X, y)

```

```python
# Display the scores of each feature
selected_features = pd.DataFrame({
    'Feature': X.columns,
    'Score': selector.scores_
}).sort_values(by='Score', ascending=False)

print(selected_features)
```

```
      Feature       Score
5        Fare  117.233400
1         Sex   92.702447
0      Pclass   30.873699
9     IsAlone   14.640793
4       Parch   10.097499
7  Embarked_S    5.489205
8  FamilySize    4.046859
3       SibSp    2.581865
2         Age    0.103042
6  Embarked_Q    0.010847
```

In [14]:
```python
# Use RFE to select the top 5 features
model = LogisticRegression()
rfe = RFE(model, n_features_to_select=5)
fit = rfe.fit(X, y)

# Display selected features and their rankings
selected_features_rfe = pd.DataFrame({
    'Feature': X.columns,
    'Selected': fit.support_,
    'Ranking': fit.ranking_
}).sort_values(by='Ranking')

print(selected_features_rfe)
```

```
      Feature  Selected  Ranking
0      Pclass      True        1
1         Sex      True        1
2         Age      True        1
8  FamilySize      True        1
9     IsAlone      True        1
7  Embarked_S     False        2
3       SibSp     False        3
4       Parch     False        4
5        Fare     False        5
6  Embarked_Q     False        6
```

In [15]:
```python
# Fit a Random Forest Classifier
model = RandomForestClassifier()
model.fit(X, y)

# Get feature importance
importances = model.feature_importances_

# Display the feature importance
feature_importance_rf = pd.DataFrame({
    'Feature': X.columns,
    'Importance': importances
}).sort_values(by='Importance', ascending=False)

print(feature_importance_rf)
```

```
      Feature  Importance
1         Sex    0.353271
2         Age    0.183765
0      Pclass    0.127187
5        Fare    0.125003
8  FamilySize    0.068257
3       SibSp    0.040284
4       Parch    0.036447
7  Embarked_S    0.033384
6  Embarked_Q    0.016387
9     IsAlone    0.016016
```

In [16]:
```python
# Apply PCA to reduce dimensions to 2
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

# Display explained variance ratio and the first 5 rows of the new components
print('Explained Variance Ratio:', pca.explained_variance_ratio_)
print('PCA Components:', X_pca[:5])
```

```
Explained Variance Ratio: [0.41706977 0.22061589]
PCA Components: [[ 0.16229872 -0.78274658]
 [ 0.17288646  1.3913265 ]
 [-0.71748002 -0.69622037]
 [ 0.16737685  1.01688818]
 [-0.80424101 -0.67853389]]
```

Conclusion:
In this lab, we focused on key techniques for feature engineering and selection. We handled missing values, encoded categorical variables, and created new features to enrich our dataset. We applied scaling to standardize numerical features and used various feature selection methods to identify the most relevant predictors. Finally, we performed PCA for dimensionality reduction, ensuring we retained essential information while simplifying our dataset. These steps emphasized the importance of effective preprocessing in enhancing model performance.

Lab 6: Feature Engineering & Selection Mechanism

Aim: Problems on Basic Feature Engineering & Selection Mechanisms

Objective:
This lab aims to help you understand and apply feature engineering and feature selection techniques. By the end of this lab, you will be able to preprocess data, create new features, and select the most important features using tools like NumPy, Pandas, and SciPy.

Key Concepts:

1. ***Non-Parametric Nature:***
   SVR does not make assumptions about the underlying data distribution. It allows for flexible fitting of data points without specifying a predefined form (like linear or polynomial).
2. ***ε-Insensitive Loss Function:***
   SVR uses an ε-insensitive loss function, meaning that errors within a certain margin (ε) do not contribute to the loss function. This enables the model to ignore small variations while fitting the data.
3. ***Support Vectors:***
   The data points that lie on the edge of the ε-tube (margin) are called support vectors. These points are critical for determining the regression line or curve.
4. ***Kernel Functions:***
   SVR can model non-linear relationships by using kernel functions (e.g., polynomial, radial basis function (RBF)) to map input features into a higher-dimensional space where the data becomes linearly separable.

Tasks Overview:

Task 1: Data Preprocessing and Handling Missing Values

Data preprocessing involves cleaning and transforming the raw data to make it ready for modeling. In this task, you will handle missing values and explore the dataset.

Steps:

1. Load a dataset (e.g., Titanic dataset).
2. Explore the dataset by displaying the first few rows and understanding its structure.
3. Identify missing values in each column and determine how to handle them:
    - Numerical columns: Fill missing values with the mean, median, or mode.
    - Categorical columns: Fill missing values with the most frequent value (mode).
4. Drop columns with excessive missing values, or those that are not useful for analysis.

Task 2: Encoding Categorical Variables

Machine learning models require numerical input, so categorical variables must be converted into numerical form.

This can be done using:

- Label Encoding: Converts each category into a unique integer value (for ordinal data).
- One-Hot Encoding: Creates binary columns for each category (for nominal data).

Steps:

– Identify categorical columns in your dataset.

– Choose either Label Encoding or One-Hot Encoding depending on the type of data.

– Apply the appropriate encoding technique and verify the results.

Task 3: Creating New Features

Feature engineering involves creating new features from existing data to provide the model with more information. Creating features can significantly improve model performance.

Steps:

1. Identify columns that can be combined to create new features (e.g., SibSp and Parch can be combined to create a FamilySize feature).
2. Use domain knowledge to create new features.

For example:

- FamilySize: Number of family members traveling together.
- IsAlone: Whether the person is traveling alone or not.

3. Verify the newly created features by examining the updated dataset.

Task 4: Feature Scaling

Feature scaling is the process of normalizing numerical features to ensure that they have a similar range. This is important for models sensitive to the scale of input features (e.g., linear regression, neural networks).

Common Scaling Techniques:

1. Standardization: Subtract the mean and divide by the standard deviation, resulting in features with zero mean and unit variance.
2. Normalization: Scale the features to a range (e.g., [0, 1]).

Steps:

1. Identify numerical features that need scaling.
2. Apply the appropriate scaling technique.
3. Ensure that all scaled features have the correct range.

Task 5: Feature Selection Mechanisms

Feature selection helps in reducing the dimensionality of your dataset by selecting the most relevant features, improving model performance and reducing overfitting. Several methods are commonly used:

1. Univariate Selection (Chi-Square Test): Use statistical tests to select features that are most related to the target variable. This test is typically applied to categorical data and helps select the top features.
2. Recursive Feature Elimination (RFE): RFE recursively eliminates the least important features and fits the model repeatedly to select the best features. This is done using a base model (e.g., logistic regression or decision tree).
3. Feature Importance (Random Forest): Some machine learning algorithms, like Random Forests, provide built-in feature importance metrics. Features are ranked based on how important they are in predicting the target variable.

Steps:

1. Choose a feature selection method: Univariate Selection for testing individual features. RFE for recursive elimination of less important features. Random Forest to rank features by importance.
2. Apply the selected method and examine the features that have the highest relevance or importance.

Task 6: Dimensionality Reduction with PCA

Principal Component Analysis (PCA) is a method for reducing the number of features (dimensionality) while preserving as much variance (information) as possible. PCA transforms the data into a set of orthogonal components that capture the most variance in the data.

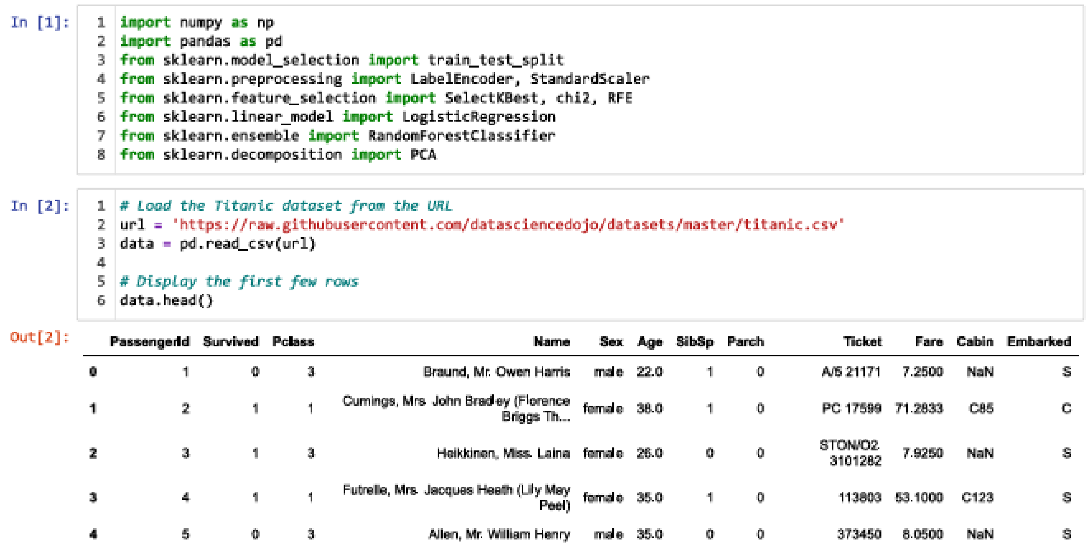
Steps:

1. Standardize the data (PCA requires scaled data).
2. Apply PCA to reduce the dataset to a lower number of components (e.g., 2 or 3 components).
3. Analyse the variance explained by each principal component to ensure that enough information is retained.

Exercises:

1. Data Preprocessing: Load a dataset and handle missing values, encoding, and scaling.
2. Feature Engineering: Create new features from existing ones (e.g., family size, interaction terms).
3. Feature Selection: Implement the following methods to select important features:
   - Univariate selection using chi-square test.
   - Recursive feature elimination (RFE) using logistic regression.
   - Feature importance ranking using random forest.
4. Dimensionality Reduction: Apply PCA and reduce the dataset to two principal components.

Code & Output:



```
In [1]:
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.feature_selection import SelectKBest, chi2, RFE
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.decomposition import PCA
```

```
In [2]:
# Load the Titanic dataset from the URL
url = 'https://raw.githubusercontent.com/datasciencedojo/datasets/master/titanic.csv'
data = pd.read_csv(url)

# Display the first few rows
data.head()
```

Out[2]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

In [5]:

```python
from google.colab import sheets
df = pd.DataFrame(data)
sheet = sheets.InteractiveSheet(df=df)
```

```
https://docs.google.com/spreadsheets/d/16Y2WnLPGRzrQmJvSpqhP2nMvPKFyoNdydmzLq5L3uAg#gid=0 (https://docs.google.com/s
preadsheets/d/16Y2WnLPGRzrQmJvSpqhP2nMvPKFyoNdydmzLq5L3uAg#gid=0)

/usr/local/lib/python3.10/dist-packages/google/colab/sheets.py:31: FutureWarning: DataFrame.applymap has been deprec
ated. Use DataFrame.map instead.
  return frame.applymap(_clean_val).replace({np.nan: None})
```

In [6]:

```python
# Check for missing values
data.isnull().sum()
```

Out[6]:

	0
PassengerId	0
Survived	0
Pclass	0
Name	0
Sex	0
Age	177
SibSp	0
Parch	0
Ticket	0
Fare	0
Cabin	687
Embarked	2

In [7]:

```python
# Fill missing values in 'Age' with the median
data['Age'].fillna(data['Age'].median(), inplace=True)

# Fill missing values in 'Embarked' with the mode
data['Embarked'].fillna(data['Embarked'].mode()[0], inplace=True)

# Drop 'Cabin' as it has too many missing values
data.drop('Cabin', axis=1, inplace=True)
```

```
<ipython-input-7-5d2a83a8e50d>:2: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series thro
ugh chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which
we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or
df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.


  data['Age'].fillna(data['Age'].median(), inplace=True)
<ipython-input-7-5d2a83a8e50d>:5: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series thro
ugh chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which
we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or
df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.


  data['Embarked'].fillna(data['Embarked'].mode()[0], inplace=True)
```

In [8]:

```python
# Convert 'Sex' column using Label Encoding
label_encoder = LabelEncoder()
data['Sex'] = label_encoder.fit_transform(data['Sex'])

# Convert 'Embarked' using One-Hot Encoding
data = pd.get_dummies(data, columns=['Embarked'], drop_first=True)

# Check the first few rows to verify encoding
data.head()
```

Out[8]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked_Q	Embarked_S
0	1	0	3	Braund, Mr. Owen Harris	1	22.0	1	0	A/5 21171	7.2500	False	True
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	0	38.0	1	0	PC 17599	71.2833	False	False
2	3	1	3	Heikkinen, Miss. Laina	0	26.0	0	0	STON/O2. 3101282	7.9250	False	True
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	0	35.0	1	0	113803	53.1000	False	True
4	5	0	3	Allen, Mr. William Henry	1	35.0	0	0	373450	8.0500	False	True

In [9]:
```python
# Create 'FamilySize' feature
data['FamilySize'] = data['SibSp'] + data['Parch'] + 1

# Create 'IsAlone' feature
data['IsAlone'] = np.where(data['FamilySize'] == 1, 1, 0)

# Check the new features
data[['SibSp', 'Parch', 'FamilySize', 'IsAlone']].head()
```

Out[9]:

	SibSp	Parch	FamilySize	IsAlone
0	1	0	2	0
1	1	0	2	0
2	0	0	1	1
3	1	0	2	0
4	0	0	1	1

In [10]:
```python
scaler = StandardScaler()
data[['Age', 'Fare', 'FamilySize']] = scaler.fit_transform(data[['Age', 'Fare', 'FamilySize']])

# Check the scaled features
data[['Age', 'Fare', 'FamilySize']].head()
```

Out[10]:

	Age	Fare	FamilySize
0	-0.565736	-0.502445	0.059160
1	0.663861	0.786845	0.059160
2	-0.258337	-0.488854	-0.560975
3	0.433312	0.420730	0.059160
4	0.433312	-0.486337	-0.560975

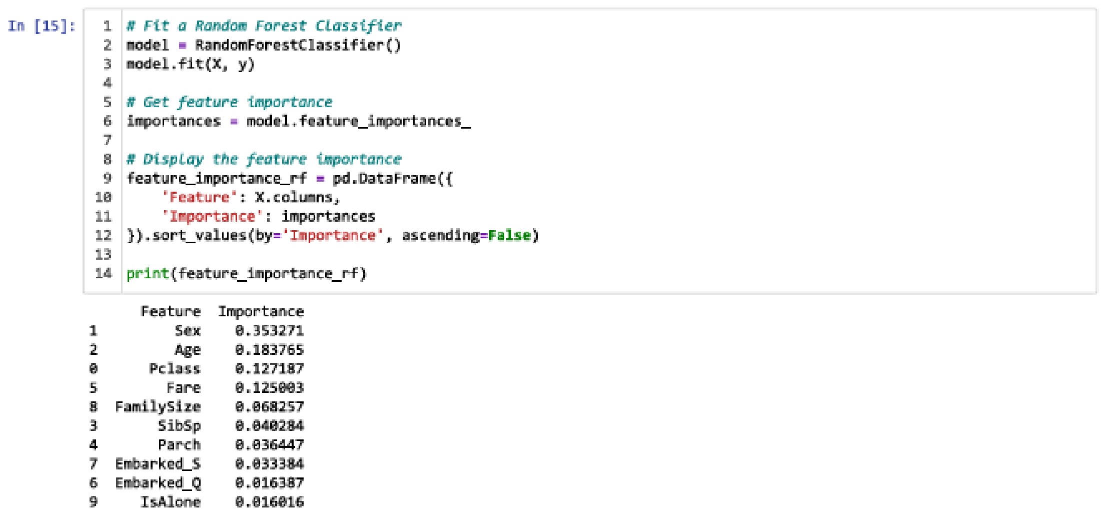
In [11]:
```python
# Separate features (X) and target (y)
X = data.drop(['PassengerId', 'Name', 'Ticket', 'Survived'], axis=1)
y = data['Survived']
```

In [13]:
```python
# Ensure all values in X are non-negative
X[X < 0] = 0

# Use SelectKBest to select top 5 features based on chi-squared test
selector = SelectKBest(score_func=chi2, k=5)
X_new = selector.fit_transform(X, y)

# Display the scores of each feature
selected_features = pd.DataFrame({
    'Feature': X.columns,
    'Score': selector.scores_
}).sort_values(by='Score', ascending=False)

print(selected_features)
```

```
      Feature       Score
5        Fare  117.233400
1         Sex   92.702447
0      Pclass   30.873699
9     IsAlone   14.640793
4       Parch   10.097499
7  Embarked_S    5.489205
8  FamilySize    4.046859
3       SibSp    2.581865
2         Age    0.103042
6  Embarked_Q    0.010847
```

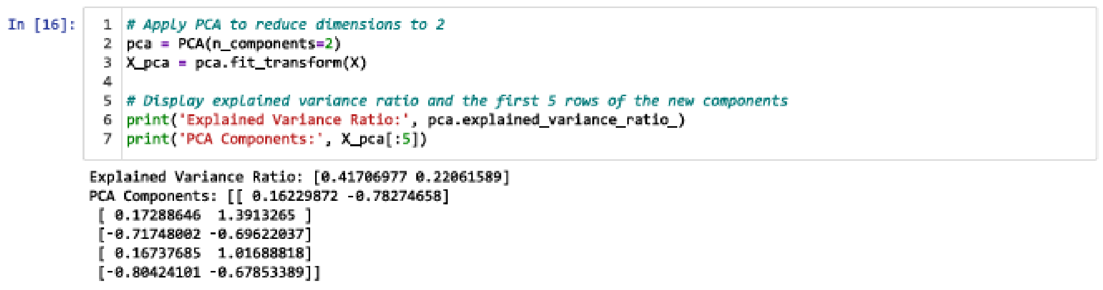
In [14]:
```python
# Use RFE to select the top 5 features
model = LogisticRegression()
rfe = RFE(model, n_features_to_select=5)
fit = rfe.fit(X, y)

# Display selected features and their rankings
selected_features_rfe = pd.DataFrame({
    'Feature': X.columns,
    'Selected': fit.support_,
    'Ranking': fit.ranking_
}).sort_values(by='Ranking')

print(selected_features_rfe)
```

```
      Feature  Selected  Ranking
0      Pclass      True        1
1         Sex      True        1
2         Age      True        1
8  FamilySize      True        1
9     IsAlone      True        1
7  Embarked_S     False        2
3       SibSp     False        3
4       Parch     False        4
5        Fare     False        5
6  Embarked_Q     False        6
```



```
In [15]:
# Fit a Random Forest Classifier
model = RandomForestClassifier()
model.fit(X, y)

# Get feature importance
importances = model.feature_importances_

# Display the feature importance
feature_importance_rf = pd.DataFrame({
    'Feature': X.columns,
    'Importance': importances
}).sort_values(by='Importance', ascending=False)

print(feature_importance_rf)
```

```
      Feature  Importance
1         Sex    0.353271
2         Age    0.183765
0      Pclass    0.127187
5        Fare    0.125003
8  FamilySize    0.068257
3       SibSp    0.040284
4       Parch    0.036447
7  Embarked_S    0.033384
6  Embarked_Q    0.016387
9     IsAlone    0.016016
```



```
In [16]:
# Apply PCA to reduce dimensions to 2
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

# Display explained variance ratio and the first 5 rows of the new components
print('Explained Variance Ratio:', pca.explained_variance_ratio_)
print('PCA Components:', X_pca[:5])
```

```
Explained Variance Ratio: [0.41706977 0.22061589]
PCA Components: [[ 0.16229872 -0.78274658]
 [ 0.17288646  1.3913265 ]
 [-0.71748002 -0.69622037]
 [ 0.16737685  1.01688818]
 [-0.80424101 -0.67853389]]
```

Conclusion:

In this lab, we focused on key techniques for feature engineering and selection. We handled missing values, encoded categorical variables, and created new features to enrich our dataset. We applied scaling to standardize numerical features and used various feature selection methods to identify the most relevant predictors. Finally, we performed PCA for dimensionality reduction, ensuring we retained essential information while simplifying our dataset. These steps emphasized the importance of effective preprocessing in enhancing model performance.

Lab 7: EDA Analysis on Diabetes Dataset

Aim: Perform EDA on the given Pima Indians Diabetes Dataset .

Objectives:

1. Read about the dataset on Kaggle .
2. Import the required data and libraries .
3. Find the Number of Columns
4. Show the first 10 records of the dataset .
5. Find the number of rows in the dataset
6. Understand the dimensions of the dataset.
7. Get the size of the dataset and check for the missing values.

8. Do a statistical summary of the data
9. Display Scatterplots , Heatmaps Box plots and Bar Graphs .
10. Display the final plot and write a brief Analytical Report on your understanding and outcomes.

Database Information:

Dataset Link : https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database

About Pima Indians Diabetes Dataset :

This dataset is originally from the National Institute of Diabetes, Digestive and Kidney Diseases. The objective of the dataset is to diagnostically predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset. Several constraints were placed on the selection of these instances from a larger database. In particular, all patients here are females at least 21 years old of Pima Indian heritage. The datasets consist of several medical predictor variables and one target variable, Outcome. Predictor variables include the number of pregnancies the patient has had, their BMI, insulin level, age, and so on.

Below is the attribute information:

1. Pregnancies: Number of times pregnant
2. Glucose: Plasma glucose concentration 2 hours in an oral glucose tolerance test
3. Blood pressure: Diastolic blood pressure (mm Hg)
4. SkinThickness: Triceps skinfold thickness (mm)
5. Insulin: 2-Hour serum insulin (mu U/ml) test
6. BMI: Body mass index (weight in kg/(height in m)^2)
7. DiabetesPedigreeFunction: A function that scores the likelihood of diabetes based on family history
8. Age: Age in years
9. Outcome: Class variable (0: the person is not diabetic or 1: the person is diabetic)

Code & Output:

```
In [1]:
import numpy as np
import pandas as pd
```

```
In [2]:
df = pd.read_csv('diabetes.csv')
```

```
In [34]:
df.head(10)
```

Out[34]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
5	5	116	74	0	0	25.6	0.201	30	0

In [7]:
```python
df.columns
```
Out[7]:
```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

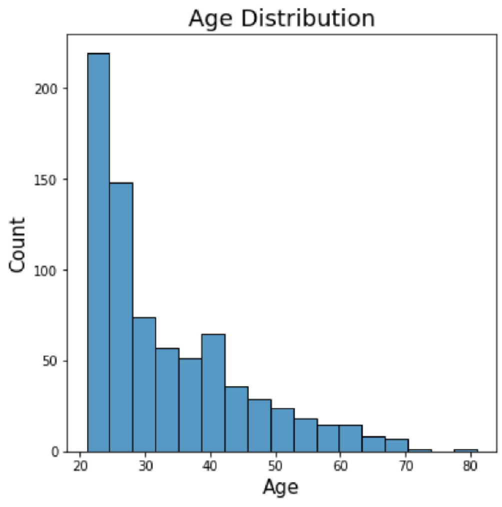
In [11]:
```python
df
```
Out[11]:

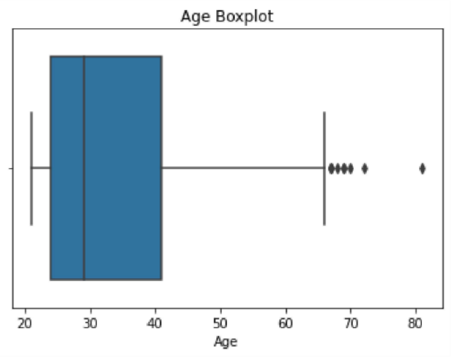
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
...	...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

768 rows × 9 columns

In [12]:
```python
df.isnull().sum()
```
Out[12]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

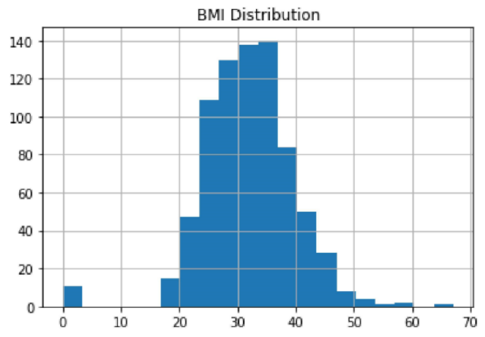
31
```python
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(6,6))
sns.histplot(df.Age)
plt.title('Age Distribution',size=18)
plt.xlabel('Age',size=15) #font size
plt.ylabel('Count',size=15)
plt.show()

sns.boxplot(df['Age'])
plt.title('Age Boxplot')
plt.show()
```





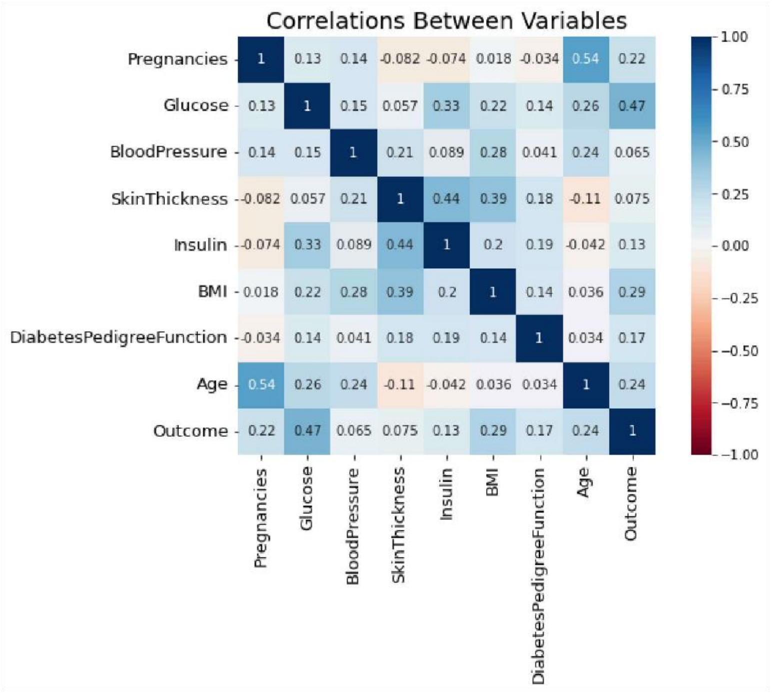
In [29]:

```python
df['BMI'].hist(bins=20)
plt.title('BMI Distribution')
plt.show()
```



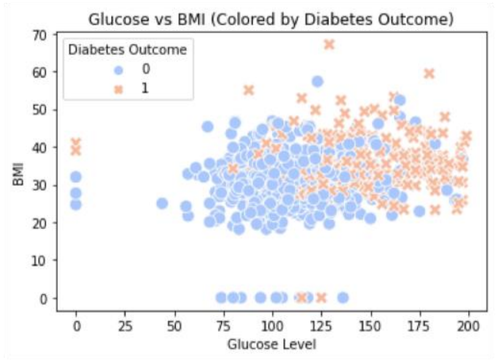
22

```python
plt.figure(figsize = (10,6))
sns.heatmap(df.corr(),annot=True,square=True,
            cmap='RdBu',
            vmax=1,
            vmin=-1)
plt.title('Correlations Between Variables',size=18);
plt.xticks(size=13)
plt.yticks(size=13)
plt.show()
```

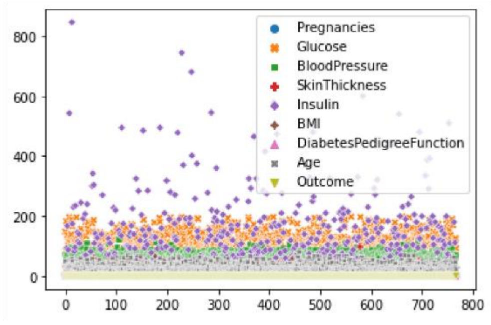


In [27]:
```
df.shape[1]
```
Out[27]:
```
9
```

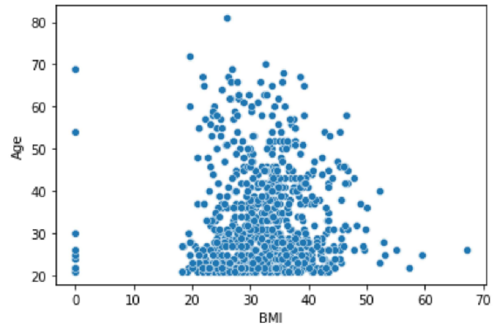
In [40]:
```
sns.scatterplot(data=df, x='Glucose', y='BMI', hue='Outcome', palette='coolwarm', style='Outcome', s=100)
plt.title('Glucose vs BMI (Colored by Diabetes Outcome)')
plt.xlabel('Glucose Level')
plt.ylabel('BMI')
plt.legend(title='Diabetes Outcome')
```
Out[40]:
```
<matplotlib.legend.Legend at 0x1dc0a659c40>
```



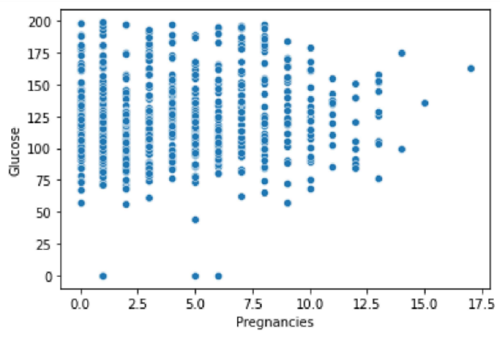
41
```
sns.scatterplot(data = df)
plt.show()
```



In [42]:
```
sns.scatterplot(data = df, x='BMI', y='Age')
plt.show()
```



In [47]:
```
import itertools
for x, y in itertools.permutations(df.columns, 2):
    sns.scatterplot(data=df, x=x, y=y)
    plt.show()
```



Conclusion:

The EDA highlighted missing values, key feature correlations, and class imbalance in the dataset. It provided a foundation for data cleaning and model preparation.