ABC Trainings

Data Science Lab Manual

1. Exploratory Data Analysis (EDA)

Exploratory Data Analysis (EDA) is a crucial step in the data analysis process that involves examining and summarizing datasets to understand their underlying patterns, spot anomalies, test hypotheses, and check assumptions with the help of statistical graphs and other data visualization techniques. It helps to gain insights and guide further analysis, model building, or feature engineering.

Lab Requirements

- Anaconda application

Lab Objective

Exploratory Data Analysis (EDA)

1. Steps in Exploratory Data Analysis (EDA)

1.1 Understanding the Data Structure

Before diving into analysis, it's essential to load and inspect the dataset. You can check for the structure, data types, number of records, missing values, and general descriptive statistics.

1.2 Cleaning the Data

Often, raw data needs cleaning before any meaningful analysis. This includes:

- Handling missing values.
- Removing duplicates.
- Correcting incorrect or inconsistent data entries.
- Transforming or encoding categorical variables if necessary.

1.3 Analyzing Patterns

You can begin by using various plots to identify relationships between variables, distributions, and outliers in the data. It often involves statistical summaries and visualizations such as histograms, scatter plots, box plots, etc.

To perform Exploratory Data Analysis (EDA) on the IPL dataset using only Matplotlib, here's how you can achieve the same visualizations and analyses.

Step 1: Load the Dataset

First, load the dataset and inspect the first few rows.

Import necessary libraries

import pandas as pd

import matplotlib.pyplot as plt

Load the IPL dataset

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

Display the first few rows of the dataset

df.head()

In [1]:

```
# Import essential libraries
import pandas as pd
import matplotlib.pyplot as plt

# Load the IPL dataset
df = pd.read_csv(r'file:///C:\Users\PRAVIN\Downloads\data.csv')

# Display the first few rows of the dataset
df.head()
```

Out[1]:

	id	season	city	date	team1	team2	toss_winner	toss_decision	result	dl_applied	winner	win_by_runs	win_by_wickets	player_of_m
0	1	2008	Bangalore	2008-04-18	Kolkata Knight Riders	Royal Challengers Bangalore	Royal Challengers Bangalore	field	normal	0	Kolkata Knight Riders	140	0	BB McC
1	2	2008	Chandigarh	2008-04-19	Chennai Super Kings	Kings XI Punjab	Chennai Super Kings	bat	normal	0	Chennai Super Kings	33	0	MEK Hu
2	3	2008	Delhi	2008-04-19	Rajasthan Royals	Delhi Daredevils	Rajasthan Royals	bat	normal	0	Delhi Daredevils	0	9	MF Mah
3	4	2008	Mumbai	2008-04-20	Mumbai Indians	Royal Challengers Bangalore	Mumbai Indians	bat	normal	0	Royal Challengers Bangalore	0	5	MV Bou
4	5	2008	Kolkata	2008-04-20	Deccan Chargers	Kolkata Knight Riders	Deccan Chargers	bat	normal	0	Kolkata Knight Riders	0	5	DJ Hu

Step 1.1 Understanding the Data Structure

Get basic info about the dataset

df.info()

Check for missing values

df.isnull().sum()

Get summary statistics of numerical columns

df.describe()

```
In [2]: # Get basic info about the dataset
        df.info()

        # Check for missing values
        df.isnull().sum()

        # Get summary statistics of numerical columns
        df.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 577 entries, 0 to 576
Data columns (total 18 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   id               577 non-null    int64
 1   season           577 non-null    int64
 2   city             570 non-null    object
 3   date             577 non-null    object
 4   team1            577 non-null    object
 5   team2            577 non-null    object
 6   toss_winner      577 non-null    object
 7   toss_decision    577 non-null    object
 8   result           577 non-null    object
 9   dl_applied       577 non-null    int64
 10  winner           574 non-null    object
 11  win_by_runs      577 non-null    int64
 12  win_by_wickets   577 non-null    int64
 13  player_of_match  574 non-null    object
 14  venue            577 non-null    object
 15  umpire1          577 non-null    object
 16  umpire2          577 non-null    object
 17  umpire3          0 non-null      float64
dtypes: float64(1), int64(5), object(12)
memory usage: 81.3+ KB
```

Out[2]:

	id	season	dl_applied	win_by_runs	win_by_wickets	umpire3
count	577.000000	577.000000	577.000000	577.000000	577.000000	0.0
mean	289.000000	2012.029463	0.025997	13.715771	3.363951	NaN
std	166.709828	2.486247	0.159263	23.619282	3.416049	NaN
min	1.000000	2008.000000	0.000000	0.000000	0.000000	NaN
25%	145.000000	2010.000000	0.000000	0.000000	0.000000	NaN

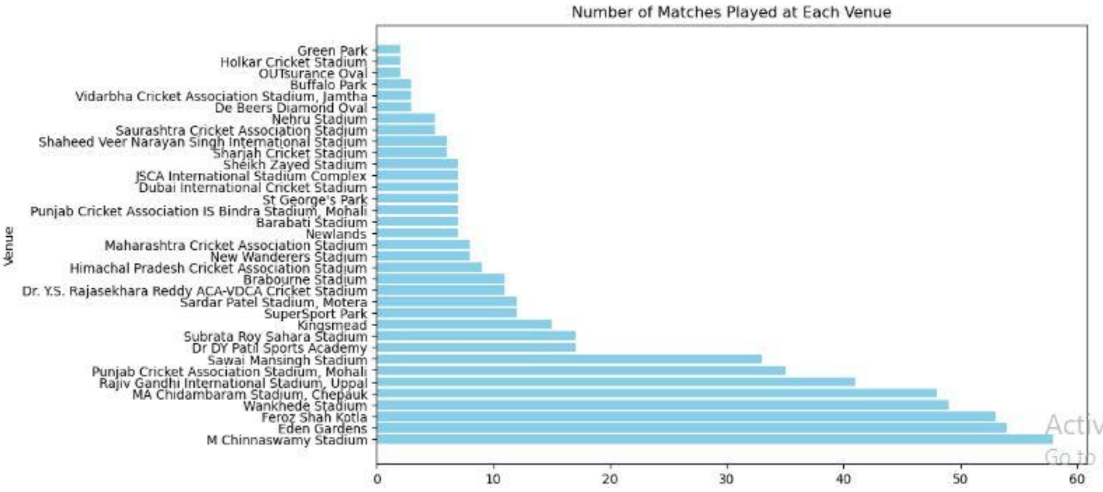
Step 2: Univariate Analysis

2.1 Frequency of Matches Played at Different Venues

To plot the number of matches played at each venue, we will use Matplotlib's `barh` for horizontal bar plots.

```
# Frequency of matches at each venue

venue_counts = df['Venue'].value_counts()

# Plot using Matplotlib

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

plt.barh(venue_counts.index, venue_counts.values, color='skyblue')

plt.title('Number of Matches Played at Each Venue')

plt.xlabel('Number of Matches')

plt.ylabel('Venue')

plt.show()
```

```
In [3]: # Frequency of matches at each venue
venue_counts = df['venue'].value_counts()

# Plot using Matplotlib
plt.figure(figsize=(10, 6))
plt.barh(venue_counts.index, venue_counts.values, color='skyblue')
plt.title('Number of Matches Played at Each Venue')
plt.xlabel('Number of Matches')
plt.ylabel('Venue')
plt.show()
```



2.2 Top Players with Most Player of the Match Awards

Top 10 players who won the most Player of the Match awards

top_players = df['Player_of_match'].value_counts().head(10)

Bar plot for top 10 players

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

plt.bar(top_players.index, top_players.values, color='lightgreen')

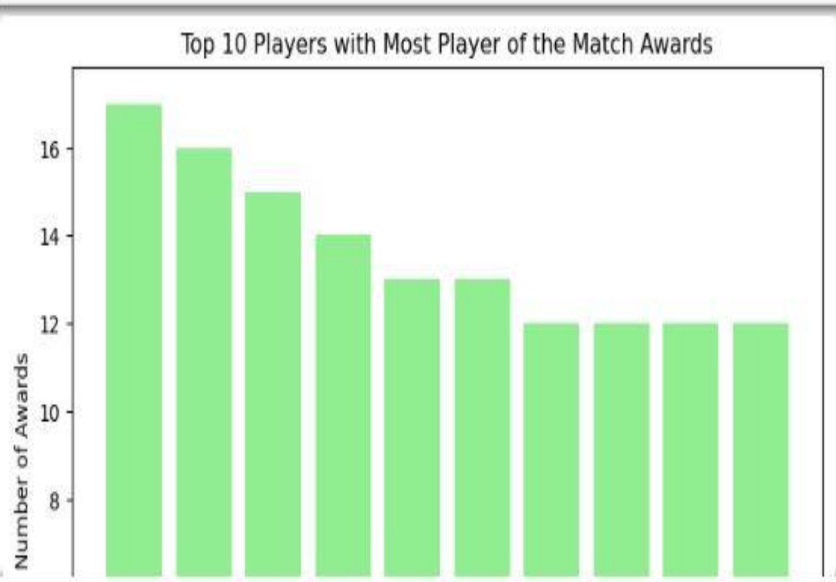
plt.xticks(rotation=45)

plt.title('Top 10 Players with Most Player of the Match Awards')

plt.ylabel('Number of Awards')

plt.show()

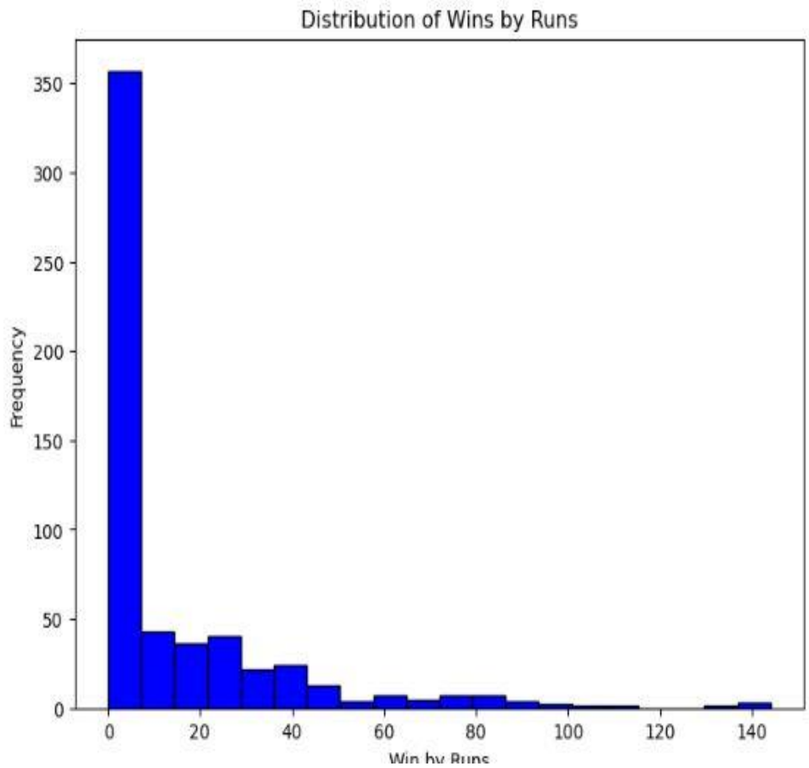
```
In [5]: # Top 10 players who won the most Player of the Match awards
        top_players = df['player_of_match'].value_counts().head(10)

        # Bar plot for top 10 players
        plt.figure(figsize=(8, 6))
        plt.bar(top_players.index, top_players.values, color='lightgreen')
        plt.xticks(rotation=45)
        plt.title('Top 10 Players with Most Player of the Match Awards')
        plt.ylabel('Number of Awards')
        plt.show()
```



2.3 Distribution of Wins by Runs

```
# Plot histogram for distribution of wins by runs

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

plt.hist(df['Win_by_runs'], bins=20, color='blue', edgecolor='black')

plt.title('Distribution of Wins by Runs')

plt.xlabel('Win by Runs')

plt.ylabel('Frequency')

plt.show()
```

In [6]:
```
# Plot histogram for distribution of wins by runs
plt.figure(figsize=(8, 6))
plt.hist(df['win_by_runs'], bins=20, color='blue', edgecolor='black')
plt.title('Distribution of Wins by Runs')
plt.xlabel('Win by Runs')
plt.ylabel('Frequency')
plt.show()
```



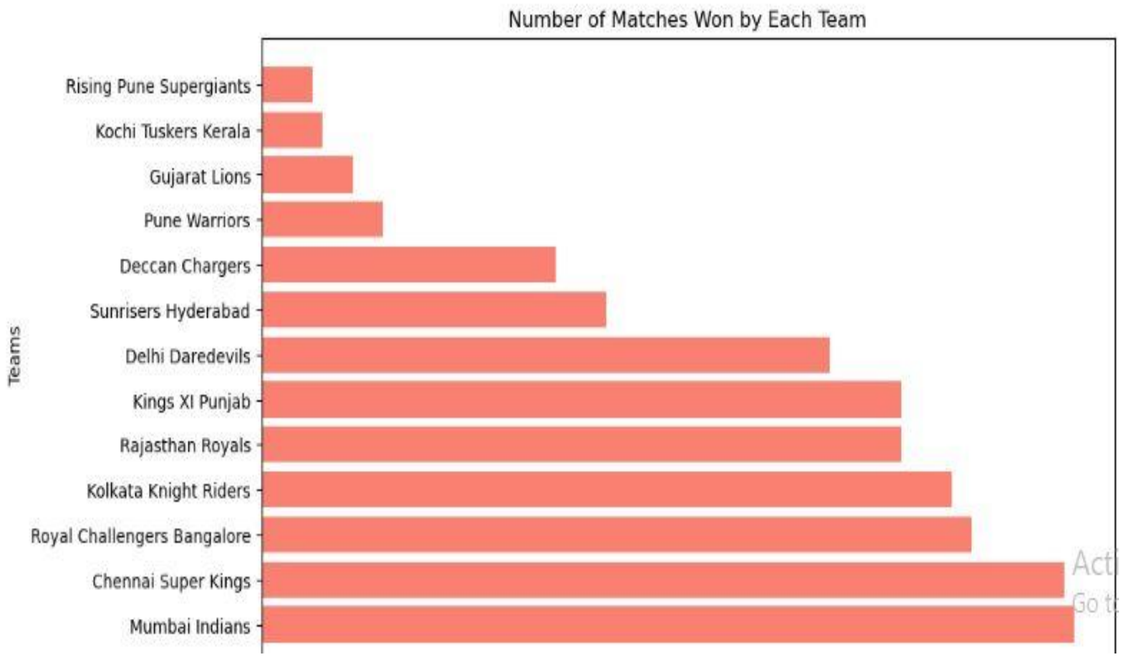
Step 3: Bivariate Analysis

3.1 Winning Teams Performance

We can analyze which teams have won the most matches using a bar chart.

```
# Count of matches won by each team

winner_counts = df['Winner'].value_counts()

# Plot using Matplotlib

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

plt.barh(winner_counts.index, winner_counts.values, color='salmon')

plt.title('Number of Matches Won by Each Team')

plt.xlabel('Number of Wins')

plt.ylabel('Teams')

plt.show()
```

```
In [7]: # Count of matches won by each team
winner_counts = df['winner'].value_counts()

# Plot using Matplotlib
plt.figure(figsize=(10, 6))
plt.barh(winner_counts.index, winner_counts.values, color='salmon')
plt.title('Number of Matches Won by Each Team')
plt.xlabel('Number of Wins')
plt.ylabel('Teams')
plt.show()
```

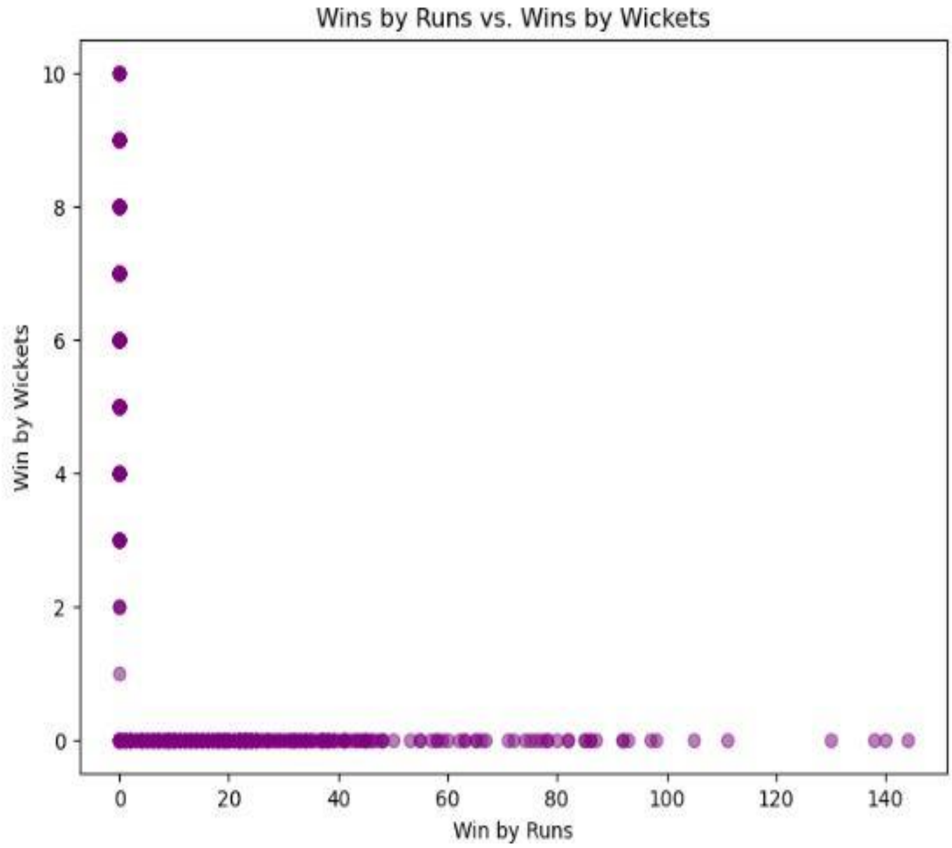


3.2 Team Wins by Runs vs. Wickets

For this scatter plot, we'll use `plt.scatter` to visualize the relationship between wins by runs and wins by wickets.

```
# Scatter plot of win by runs vs. win by wickets

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

plt.scatter(df['Win_by_runs'], df['Win_by_wickets'], c='purple', alpha=0.5)

plt.title('Wins by Runs vs. Wins by Wickets')

plt.xlabel('Win by Runs')

plt.ylabel('Win by Wickets')

plt.show()
```

```
In [10]: # Scatter plot of win by runs vs. win by wickets
         plt.figure(figsize=(8, 6))
         plt.scatter(df['win_by_runs'], df['win_by_wickets'], c='purple', alpha=0.5)
         plt.title('Wins by Runs vs. Wins by Wickets')
         plt.xlabel('Win by Runs')
         plt.ylabel('Win by Wickets')
         plt.show()
```



Step 4: Time-Series Analysis

4.1 Number of Matches Played Over the Years

We can plot the number of matches played each year by first extracting the year from the `Date` column.

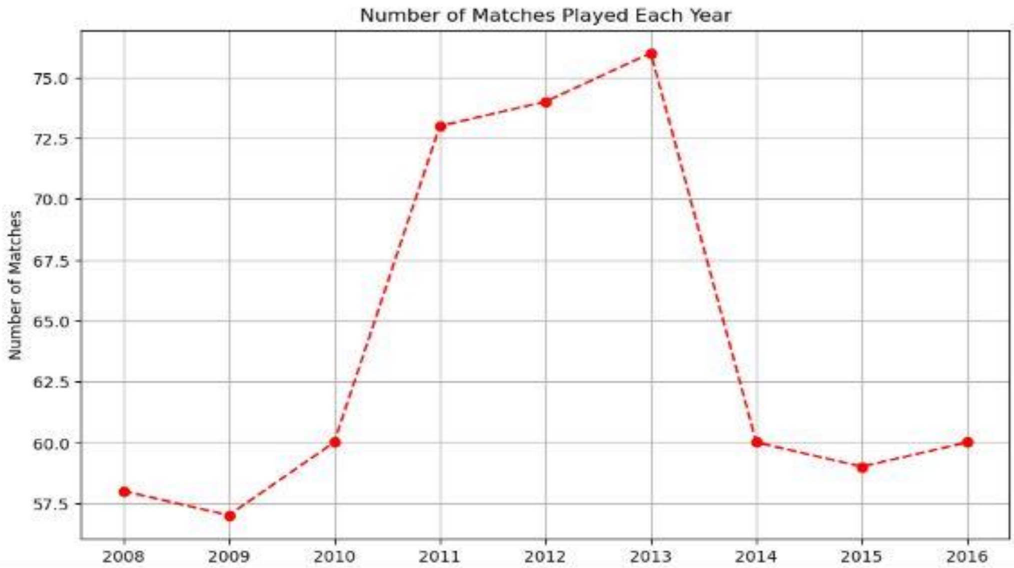
Convert the Date column to datetime format

df['Date'] = pd.to_datetime(df['Date'])

Extract the year from the Date column

```
df['Year'] = df['Date'].dt.year

# Count of matches played each year

yearly_counts = df['Year'].value_counts().sort_index()

# Plot using Matplotlib

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

plt.plot(yearly_counts.index, yearly_counts.values, marker='o', color='red',
linestyle='--')

plt.title('Number of Matches Played Each Year')

plt.xlabel('Year')

plt.ylabel('Number of Matches')

plt.grid(True)

plt.show()
```

```python
In [13]: # Convert the Date column to datetime format
df['date'] = pd.to_datetime(df['date'])

# Extract the year from the Date column
df['year'] = df['date'].dt.year

# Count of matches played each year
yearly_counts = df['year'].value_counts().sort_index()

# Plot using Matplotlib
plt.figure(figsize=(10, 6))
plt.plot(yearly_counts.index, yearly_counts.values, marker='o', color='red', linestyle='--')
plt.title('Number of Matches Played Each Year')
plt.xlabel('Year')
plt.ylabel('Number of Matches')
plt.grid(True)
plt.show()
```



Tasks:

1. Perform Exploratory Data Analysis (EDA) on iris dataset.
2. Perform Exploratory Data Analysis (EDA) on corona virus dataset.

Questions:

1. What is Exploratory Data Analysis (EDA) and why is it important?
2. What are some common techniques used in EDA?

2. PROBABILITY DISTRIBUTION

In this lab manual, we will explore basic probability concepts through Python programming. The experiments are designed to help you understand probability through various real-world examples, simulations, and experiments

Lab Requirements

- Anaconda application

Lab Objective

The experiments are designed to help you understand probability through various real-world example

Step 1 : create dataset trainer how many classes handle by trainer

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

In [2]:
df=pd.DataFrame({
    "Event":["trainer_1","trainer_2","trainer_3","trainer_4","trainer_5"],
    "No_of_Classes":[20,7,26,11,16]
})

In [3]: df
Out[3]:
```

	Event	No_of_Classes
0	trainer_1	20
1	trainer_2	7
2	trainer_3	26
3	trainer_4	11
4	trainer_5	16

Step 2 : find the probability

```
In [4]: #total of class
        totalsum=df["No_of_Classes"].sum()
        print(totalsum)

        80

In [5]: #find the probilty
        df["probability"]=df.No_of_Classes/totalsum

In [6]: df
```

Out[6]:

	Event	No_of_Classes	probability
0	trainer_1	20	0.2500
1	trainer_2	7	0.0875
2	trainer_3	26	0.3250
3	trainer_4	11	0.1375
4	trainer_5	16	0.2000

```
In [9]: #sum of probability is always 1

        df["probability"].sum()

Out[9]: 1.0
```

TRAININGS

Step 3 : create visualize the probability

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

```
In [31]: plt.xlabel("trainer taken by classes")
         plt.title("trainer taken by classes")
         plt.bar(df.Event,df.probability,width=0.4)
         plt.grid(axis='y')

         plt.show()
```

Tasks:

1. Perform probability distribution table based on a different dataset. This one focuses on the number of pets owned by a group of students.
2. Perform probability distribution table based on the provided dataset of favorite fruit.

Questions:

1. What is the difference between discrete and continuous probability distributions?
2. Can you explain what a probability is?

3. Confidence Interval

In this lab manual, This experiment demonstrates how to calculate the 90% confidence interval using the **z-score** and visualize it using a **normal distribution curve**.

Lab Requirements

- Anaconda application

Lab Objective

Calculate and visualize the 90% confidence interval using the **z-score.**

Step 1: Define the Data:

Use a dataset (n > 30). For example

```
In [8]: data = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
        print(data)

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
```

Step 2: Calculate the Mean and Standard Deviation:

Use numpy to calculate the **mean** and **standard deviation**

```
In [5]: mean_data = np.mean(data)
        std_data = np.std(data, ddof=1)  # Use ddof=1 for sample standard deviation
        n = len(data)
```

Step 3: Calculate the Confidence Interval Using Z-Score:

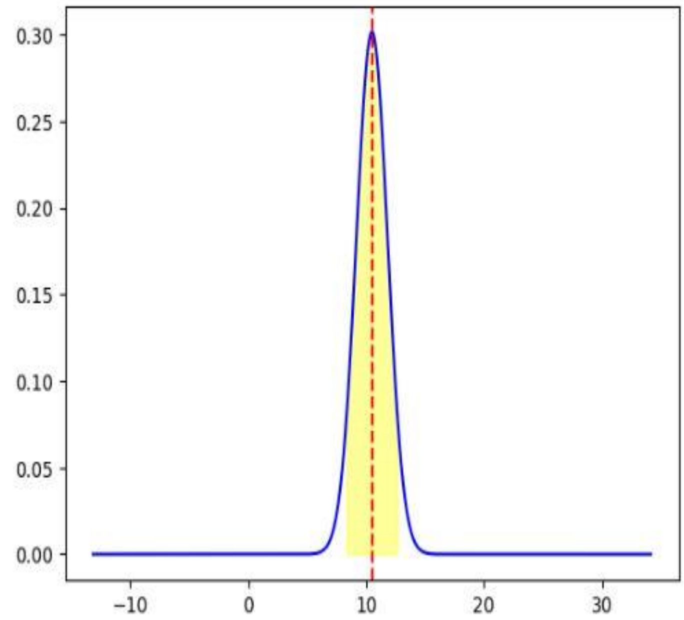
For a 90% confidence level, the **z-score** is 1.645.

Use the formula for confidence interval:

```
In [6]: z_score = 1.645
        margin_of_error = z_score * (std_data / np.sqrt(n))
        confidence_interval = (mean_data - margin_of_error, mean_data + margin_of_error)
```

Step 4: Plot the Curve and Confidence Interval

```
In [7]: x = np.linspace(mean_data - 4*std_data, mean_data + 4*std_data, 1000)
        y = st.norm.pdf(x, mean_data, std_data / np.sqrt(n))

        plt.plot(x, y, color='blue')
        plt.fill_between(x, 0, y, where=(x >= confidence_interval[0]) & (x <= confidence_interval[1]), color='yellow', alpha=0.4)
        plt.axvline(mean_data, color='red', linestyle='--')
        plt.show()
```



Output:

1. A **bell-shaped curve** showing the **normal distribution** of your data.
2. The **90% confidence interval** shaded on the curve.
3. A **vertical line** marking the sample mean.

Tasks:

1. Calculate a 95% confidence interval for the mean age from the dataset: 18, 21, 19, 22, 20, 23, 18, 19, 20, 22, 21, 19, 20, 18, 23.
2. Compare the proportions of two groups (Group A: 60 out of 100 prefer online learning; Group B: 75 out of 120) and calculate their 95% confidence intervals.

Questions:

1. What is the significance of the confidence level in a confidence interval?
2. How does sample size affect the width of a confidence interval?

4. Data preprocessing

Data preprocessing is a crucial step in the data science workflow. It involves preparing and cleaning data to make it suitable for analysis. This process ensures that the data is accurate, consistent, and usable, which can significantly enhance the quality of your models and insights. Below is a simplified guide to data preprocessing, including common techniques and steps.

Lab Requirements

- Anaconda application

Lab Objective

To learn and apply the essential steps for data preprocessing in Python, ensuring the dataset is ready for analysis.

Step 1: Load the Dataset

Load your dataset into a pandas DataFrame for easier manipulation.

In [26]:
```
import pandas as pd

# Create a sample DataFrame similar to the steps above
data = {
    'ID': [1,1, 2, 3, 4, 5, 6],
    'Age': [23,23, 45, 31, 35, 29, None],  # Contains a missing value
    'Salary': [50000,50000, 62000, 58000, None, 60000, 52000],  # Contains a missing value
    'Department': ['HR','HR' ,'Finance', 'IT', 'HR', 'Finance', 'IT'],
    'Experience': [2, 2,10, 5, 8, None, 3],  # Contains a missing value
    'Performance': ['Good','Good' ,'Excellent', 'Good', 'Fair', 'Good', 'Excellent']
}

df = pd.DataFrame(data)
df
```

Out[26]:

	ID	Age	Salary	Department	Experience	Performance
0	1	23.0	50000.0	HR	2.0	Good
1	1	23.0	50000.0	HR	2.0	Good
2	2	45.0	62000.0	Finance	10.0	Excellent
3	3	31.0	58000.0	IT	5.0	Good
4	4	35.0	NaN	HR	8.0	Fair
5	5	29.0	60000.0	Finance	NaN	Good
6	6	NaN	52000.0	IT	3.0	Excellent

Step 2: Inspect the Data

Understanding the structure and contents of the data is crucial.

1. **View Data Types:** Check the data types to understand which columns are numerical and which are categorical.

```
In [4]: print(df.dtypes)
```

```
ID               int64
Age            float64
Salary         float64
Department      object
Experience     float64
Performance     object
dtype: object
```

2. **Check for Missing Values:** Missing values can cause errors in analysis. Let's check how many missing values are present in each column

```
In [5]: print(df.isnull().sum())
```

```
ID             0
Age            1
Salary         1
Department     0
Experience     1
Performance    0
dtype: int64
```

Step 4: Handle Missing Data

There are several ways to deal with missing data, depending on the scenario:

1. **Drop Missing Values**: This removes rows with missing values.

```
In [7]: data = df.dropna()
        data
```

Out[7]:

	ID	Age	Salary	Department	Experience	Performance
0	1	23.0	50000.0	HR	2.0	Good
1	2	45.0	62000.0	Finance	10.0	Excellent
2	3	31.0	58000.0	IT	5.0	Good

2. **Impute Missing Values**: Replace missing values with the mean, median, or mode of the column.

```
In [29]: # Fill missing values with the mean
         df=df.fillna(0)
         df
```

Out[29]:

	ID	Age	Salary	Department	Experience	Performance
0	1	23.0	50000.0	HR	2.0	Good
1	1	23.0	50000.0	HR	2.0	Good
2	2	45.0	62000.0	Finance	10.0	Excellent
3	3	31.0	58000.0	IT	5.0	Good
4	4	35.0	0.0	HR	8.0	Fair
5	5	29.0	60000.0	Finance	0.0	Good
6	6	0.0	52000.0	IT	3.0	Excellent

Step 5: Handle Duplicate Data

Duplicate rows can distort your analysis. Remove duplicates as follows:

```
In [30]: # Remove duplicate rows
         data = df.drop_duplicates()
         data
```

Out[30]:

	ID	Age	Salary	Department	Experience	Performance
0	1	23.0	50000.0	HR	2.0	Good
2	2	45.0	62000.0	Finance	10.0	Excellent
3	3	31.0	58000.0	IT	5.0	Good
4	4	35.0	0.0	HR	8.0	Fair
5	5	29.0	60000.0	Finance	0.0	Good
6	6	0.0	52000.0	IT	3.0	Excellent

Step 9: Save the Preprocessed Data

After preprocessing, save the cleaned data for further analysis or modeling.

```
In [31]: #to saved processed data
         data.to_csv('preprocessed_data.csv', index=False)
```

Tasks:

1. Clean the dataset by handling missing values from the following data: Age, Salary, Gender (25, 50000, Male; 30, 60000, Female; NaN, 70000, Male; 45, NaN, Female; 22, 30000, Male; 50, 120000, NaN; 35, 55000, Female; 40, 75000, Male).
2. Clean the dataset by remove missing values from above dataset.

Questions:

1. What are the common techniques used for handling missing values in a dataset?
2. What are the common techniques used for inspect a dataset?

5. Supervised Learning Algorithm

5.1. LinearRegression

Objective:
To understand and apply linear regression to predict a continuous target variable using Python.

Step 1: Import Required Libraries

First, import the necessary libraries for data manipulation, linear regression, and visualization

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

        from sklearn.linear_model import LinearRegression
```

Step 2: Load and Inspect the Dataset

1. **Load Dataset:** Load a dataset with continuous variables. We will use a simple example, but you can load your own dataset in CSV format.

```
In [23]: #linear regression
         data=pd.read_csv(r"file:///C:\Users\PRAVIN\Documents\data3\abc\ML\6_train_test_split\carprices.csv")

In [24]: data
```

Out[24]:

	Mileage	Age(yrs)	Sell Price($)
0	69000	6	18000
1	35000	3	34000
2	57000	5	26100
3	22500	2	40000
4	46000	4	31500
5	59000	5	26750
6	52000	5	32000
7	72000	6	19300
8	91000	8	12000
9	67000	6	22000

Step 3: Split the Data

```
In [29]: from sklearn.model_selection import train_test_split
         x_train,y_train,x_test,y_test=train_test_split(data[['Mileage','Age(yrs)']],data[['Sell Price($)']],test_size=0.3)
```

Step 4: Train the Linear Regression Model

1. Initialize and Train the Model:

```
In [33]: model=LinearRegression()
         model.fit(x_train,x_test)

Out[33]: LinearRegression()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

Step 5:predict the data

```
In [36]: model.predict(y_train)

Out[36]: array([[16214.39930299],
                [26381.08300629],
                [17520.59306828],
                [14949.9023872 ],
                [23066.57061004],
                [37998.21905417]])
```

Step 6: check the accuracy

```
In [41]: model.score(x_train,x_test)

Out[41]: 0.8917664210649244

In [42]: model.score(y_train,y_test)

Out[42]: 0.9497033670318675
```

5.2. Logistic regression

Objective :
To understand and apply logistic regression for binary classification problems using Python.

Step 1: Import Required Libraries

First, import the necessary libraries for data manipulation, logistic regression, and visualization

```
In [2]: import pandas as pd
        from sklearn.linear_model import LogisticRegression
        import matplotlib.pyplot as plt
        from sklearn.model_selection import train_test_split
```

Step 2: Load and Inspect the Dataset

1. **Load Dataset:** Load a dataset with continuous variables. We will use a simple example, but you can load your own dataset in CSV format.

```
In [3]: df=pd.read_csv(r"file:///F:\abc\ML\7_logistic_reg\insurance_data.csv")
```

```
In [4]: df
```

Out[4]:

	age	bought_insurance
0	22	0
1	25	0
2	47	1
3	52	0
4	46	1
5	56	1
6	55	0
7	60	1
8	62	1
9	61	1
10	18	0

Step 3: Split the Data

```
In [20]: x_train,x_test,y_train,y_test=train_test_split(df[['age']],df.bought_insurance,test_size=0.2)
```

Step 4: Train the Logistic Regression Model

```
In [14]: #create model
         model=LogisticRegression()

In [23]: model.fit(x_train,y_train)

Out[23]: LogisticRegression()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

Step 5: predict the data

```
In [24]: model.predict([[46]])

         C:\Users\PRAVIN\anaconda3\Lib\site-packages\sklearn\base.py:464: U
         Regression was fitted with feature names
           warnings.warn(

Out[24]: array([1], dtype=int64)
```

Step 6: check the accuracy

```
In [30]: model.score(x_test,y_test)

Out[30]: 1.0
```

5.3 naive bayes

Objective :

To understand and apply the Naive Bayes algorithm for classification problems using Python.

Step 1: Import Required Libraries

First, import the necessary libraries for data manipulation, naive bayes , and visualization

```
In [1]: import pandas as pd
        from sklearn.model_selection import train_test_split
        from sklearn.naive_bayes import MultinomialNB
```

Step 2: Load and Inspect the Dataset

1. **Load Dataset:** Load a dataset with continuous variables. We will use a simple example, but you can load your own dataset in CSV format.

```
In [2]: df = pd.read_csv("spam.csv")
        df.head()
```

Out[2]:

	Category	Message
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...

```
In [4]: df['spam']=df['Category'].apply(lambda x: 1 if x=='spam' else 0)
        df.head()
```

Out[4]:

	Category	Message	spam
0	ham	Go until jurong point, crazy.. Available only ...	0
1	ham	Ok lar... Joking wif u oni...	0
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	1
3	ham	U dun say so early hor... U c already then say...	0
4	ham	Nah I don't think he goes to usf, he lives aro...	0

Step 3: Split the Data

```
In [7]: from sklearn.model_selection import train_test_split
        X_train, X_test, y_train, y_test = train_test_split(df.Message,df.spam)
```

```
In [31]: from sklearn.feature_extraction.text import CountVectorizer
         v = CountVectorizer()
         X_train_count = v.fit_transform(X_train.values)
         X_train_count.toarray()[:2]

Out[31]: array([[0, 0, 0, ..., 0, 0, 0],
                [0, 0, 0, ..., 0, 0, 0]], dtype=int64)
```

Step 4 : Train the Model

```
In [23]: from sklearn.naive_bayes import MultinomialNB
         model = MultinomialNB()
         model.fit(X_train_count,y_train)

Out[23]: MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
```

Step 5 : predict the data

```
In [37]: emails = [
             'Hey mohan, can we get together to watch footbal game tomorrow?',
             'Upto 20% discount on parking, exclusive offer just for you. Dont miss this reward!'
         ]
         emails_count = v.transform(emails)
         model.predict(emails_count)

Out[37]: array([0, 1], dtype=int64)
```

5.4 Random forest

Objective :

To understand and apply the Random Forest algorithm for classification problems using Python.

Step 1: Load and Inspect the Dataset

1. **Load Dataset:** Load a dataset with continuous variables. We will use a simple example, but you can load your own dataset in CSV format.

```
In [1]: from sklearn.datasets import load_iris
        iris = load_iris()
        dir(iris)

Out[1]: ['DESCR', 'data', 'feature_names', 'target', 'target_names']
```

```
In [2]: import pandas as pd
        df = pd.DataFrame(iris.data, columns=iris.feature_names)
        df.head()
```

Out[2]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

```
In [4]: df['target'] = iris.target
        df.head()
```

Out[4]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

Step 3: Split the Data

```
In [5]: from sklearn.model_selection import train_test_split
        X_train, X_test, y_train, y_test = train_test_split(df.drop(['target'],axis='columns'),iris.target,test_size=0.2)
```

Step 4: Train the Model

```
In [6]: from sklearn.ensemble import RandomForestClassifier
        model = RandomForestClassifier()
        model.fit(X_train, y_train)

Out[6]: RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                    max_depth=None, max_features='auto', max_leaf_nodes=None,
                    min_impurity_split=1e-07, min_samples_leaf=1,
                    min_samples_split=2, min_weight_fraction_leaf=0.0,
                    n_estimators=10, n_jobs=1, oob_score=False, random_state=None,
                    verbose=0, warm_start=False)
```

Step 5: predict the data

```
In [10]: model.predict([[5.5,3,1,0.2]])

         C:\Users\PRAVIN\anaconda3\Lib\site-packages\sklearn\base.py:464: UserWarning: X does not have valid
         restClassifier was fitted with feature names
           warnings.warn(

Out[10]: array([0])
```

Step 6: check the accuracy

```
In [7]: model.score(X_test,y_test)

Out[7]: 0.9333333333333335

In [14]: model = RandomForestClassifier(n_estimators=40)
         model.fit(X_train, y_train)
         model.score(X_test,y_test)

Out[14]: 0.9666666666666667
```

Tasks:

1. Implement regression model to predict car prices using the dataset: (Year, Mileage, Condition, Price) - (2015, 30000, Good, 20000; 2018, 15000, Excellent, 25000; 2012, 60000, Fair, 15000; 2019, 10000, Excellent, 28000; 2016, 40000, Good, 22000).
2. Train a Random Forest classifier to predict customer churn based on features in the dataset: (Age, MonthlyCharges, Tenure, Churn) - (25, 70, 12, Yes; 40, 50, 24, No; 30, 80, 6, Yes; 50, 60, 36, No; 35, 90, 18, Yes).

Questions:

1. What is a Linear Regression?
2. What is a Logistic Regression?

6. Unsupervised Learning Algorithm

6.1 Kmeans

Objective:

To understand and apply logistic regression for binary classification problems using Python.

Step 1: import module

```
In [196]: from sklearn.cluster import KMeans
          import pandas as pd
          from sklearn.preprocessing import MinMaxScaler
          from matplotlib import pyplot as plt
          %matplotlib inline
```

Step 2 : Load the dataset

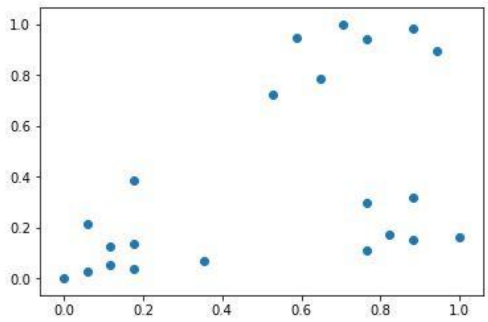
```
In [197]: df = pd.read_csv("income.csv")
          df.head()
```

Out[197]:

	Name	Age	Income($)
0	Rob	27	70000
1	Michael	29	90000
2	Mohan	29	61000
3	Ismail	28	60000
4	Kory	42	150000

Step 3: data plot using scatterplot

```
In [205]: plt.scatter(df.Age,df['Income($)'])
Out[205]: <matplotlib.collections.PathCollection at 0x159c78f2358>
```



Step 4: Create model and predict data

```
In [206]: km = KMeans(n_clusters=3)
          y_predicted = km.fit_predict(df[['Age','Income($)']])
          y_predicted
Out[206]: array([0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 2, 2, 2, 2, 2, 2])
```

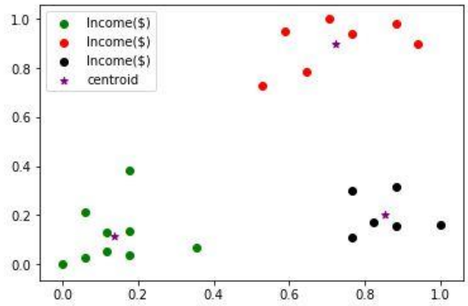
```
In [207]: df['cluster']=y_predicted
          df.head()
```

Out[207]:

	Name	Age	Income($)	cluster
0	Rob	0.058824	0.213675	0
1	Michael	0.176471	0.384615	0
2	Mohan	0.176471	0.136752	0
3	Ismail	0.117647	0.128205	0
4	Kory	0.941176	0.897436	1

Step 6: result plot using scatterplot

```
In [209]: df1 = df[df.cluster==0]
          df2 = df[df.cluster==1]
          df3 = df[df.cluster==2]
          plt.scatter(df1.Age,df1['Income($)'],color='green')
          plt.scatter(df2.Age,df2['Income($)'],color='red')
          plt.scatter(df3.Age,df3['Income($)'],color='black')
          plt.scatter(km.cluster_centers_[:,0],km.cluster_centers_[:,1],color='purple',marker='*',label='centroid')
          plt.legend()

Out[209]: <matplotlib.legend.Legend at 0x159c7982f60>
```



Tasks:

1. Perform k-means on iris dataset?
2. Apply K-means clustering to the dataset (X, Y) - (2, 3; 4, 5; 1, 2; 7, 8; 9, 10; 6, 6) and assign each point to a cluster.

Questions:

1. What is a k-means?
2. How does the K-means clustering algorithm work, and what steps are involved in forming the clusters?

7. NLP

Natural Language Processing (NLP) is a subfield of artificial intelligence (AI) that focuses on the interaction between computers and human languages. NLP enables computers to understand, interpret, and generate human language in a meaningful way. It bridges the gap between human communication (natural languages like English, Spanish, or Mandarin) and computer understanding (which is based on binary and structured data).

Lab Requirements

- Anaconda application

Lab Objective

To understand and apply fundamental Natural Language Processing (NLP) techniques using Python.

Step 1: Tokenization: Split text into individual words (tokens).

```
In [1]: from nltk.tokenize import word_tokenize

In [4]: word_tokenize("this list contains 43535 mobiles phones in india")

Out[4]: ['this', 'list', 'contains', '43535', 'mobiles', 'phones', 'in', 'india']
```

Step 2: Stemming: Convert words to their base forms using stemming .

```
In [5]: from nltk.stem import  PorterStemmer

In [19]: l=["wait","waits","waited","waiting"]

In [18]: ps=PorterStemmer()

In [20]: for i in l:
             root=ps.stem(i)
             print(root)

         wait
         wait
         wait
         wait

In [21]: ps.stem("playing")

Out[21]: 'play'
```

Step 3: Lemmatization: Convert words to their base forms using lemmatization.

```
In [24]: from nltk.stem import WordNetLemmatizer
```

st:8888/notebooks/nlp.ipynb

, 11:18 AM nlp - Jupyter Notebook

```
In [27]: lemma=WordNetLemmatizer()

In [33]: lemma.lemmatize('played')

Out[33]: 'played'

In [34]: lemma.lemmatize('plays')

Out[34]: 'play'
```

Step 4: part of speech : find the each words pos

```
In [36]: from nltk import pos_tag

In [42]: token=word_tokenize("i am going to school")

In [55]: postags=pos_tag(token)
         postags

Out[55]: [('i', 'NN'), ('am', 'VBP'), ('going', 'VBG'), ('to', 'TO'), ('school', 'N
         N')]
```

Tasks:

1. Tokenize the sentence: "I love learning about NLP!" into individual words.
2. Apply POS tagging to the sentence: "The cat is sitting on the mat." and identify the parts of speech for each word.

Questions:

1. What is tokenization in NLP?
2. What is POS tagging in NLP and why is it important?

8. Word Cloud

A **Word Cloud** is a visual representation of text data where the size of each word indicates its frequency or importance in the dataset. It is commonly used to quickly visualize the most frequent terms in a large body of text, making it easier to spot trends, keywords, or important concepts.

Lab Requirements

- Anaconda application

Objective :

To understand and create a word cloud visualization from a text dataset using Python.

Step 1: Import Required Libraries

```
In [1]: #Importing libraries

        import pandas as pd
        import matplotlib.pyplot as plt
        %matplotlib inline
        from wordcloud import WordCloud
```

Step 2: Load dataset

```
In [2]: # importing dataset into a dataframe
        df = pd.read_csv("../input/game-of-thronesgot/game of thrones.csv")
        # printing first five rows
        df.head()
```

Out[2]:

	No. overall	No. in season	Season	Title	Directed by	Written by	Novel(s) adapted
0	1	1	1	"Winter Is Coming"	Tim Van Patten	David Benioff & D. B. Weiss	A Game of Thrones
1	2	2	1	"The Kingsroad"	Tim Van Patten	David Benioff & D. B. Weiss	A Game of Thrones
2	3	3	1	"Lord Snow"	Brian Kirk	David Benioff & D. B. Weiss	A Game of Thrones
3	4	4	1	"Cripples, Bastards, and Broken Things"	Brian Kirk	Bryan Cogman	A Game of Thrones
4	5	5	1	"The Wolf and the Lion"	Brian Kirk	David Benioff & D. B. Weiss	A Game of Thrones

Step 3: Create text variable

```
In [3]: # creating the text variable
        text1 = " ".join(title for title in df.Title)
```

So, the text variable contains a string made by joining all the titles

Step 4: Create word cloud and save file

```
In [4]: # Creating word_cloud with text as argument in .generate() method

        word_cloud1 = WordCloud(collocations = False, background_color = 'white',
                                width = 2048, height = 1080).generate(text1)
        # saving the image
        word_cloud1.to_file('got.png')

Out[4]: <wordcloud.wordcloud.WordCloud at 0x7fbdd94b0210>
```

Step 5: display word cloud

```
In [5]: # Display the generated Word Cloud

        plt.imshow(word_cloud1, interpolation='bilinear')
        plt.axis("off")
        plt.show()
```

Tasks:

1. Create a WordCloud from the sentence: "Learning NLP is fun and exciting."
2. Make a WordCloud using the words: "Data, Science, AI, Machine Learning, Analytics."

Questions:

1. What is a WordCloud?
2. How does a WordCloud show the importance of words?

9. Forecasting

What is Time Series Data?

Time series data is a sequence of data points collected or recorded at specific time intervals. Examples include daily stock prices, monthly sales data, and yearly weather patterns.

Components of Time Series:

- **Trend**: The overall upward or downward movement in the data.
- **Seasonality**: Repeated patterns or cycles at regular intervals.
- **Residual/Noise**: The random variation that cannot be explained by the trend or seasonality.

Importance of Forecasting:

Forecasting is crucial in various fields such as finance, economics, and operations, enabling businesses to make data-driven decisions for planning and resource allocation.

Objective:

To understand the core concepts of time series forecasting and implement various forecasting techniques using Python.

Step 1: import required library

```
In [15]: import numpy as np
         import pandas as pd
         import matplotlib.pyplot as plt
         from statsmodels.tsa.seasonal import seasonal_decompose
         from statsmodels.tsa.statespace.sarimax import SARIMAX
```

Step 2: Load dataset

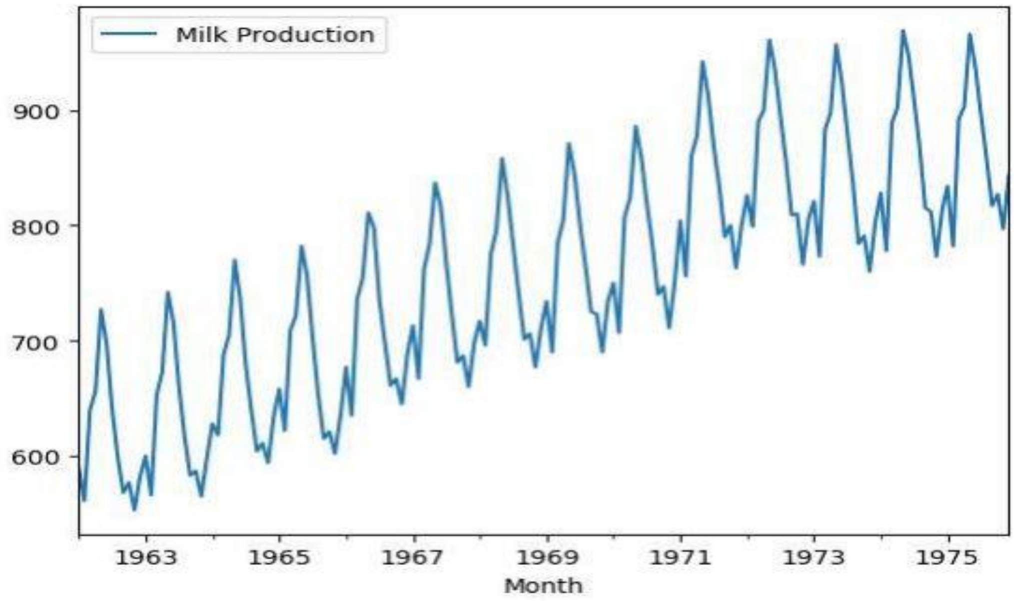
```
In [3]: df=pd.read_csv(r'file:///C:\Users\CADD_Cidco\Downloads\monthly-milk-product:
        df
```

Out[3]:

	Milk Production
Month	
1962-01-01	589.0
1962-02-01	561.0
1962-03-01	640.0
1962-04-01	656.0
1962-05-01	727.0
...	...
1975-08-01	858.0
1975-09-01	817.0

Step 3: plot graph

```
In [4]: df.plot()
Out[4]: <Axes: xlabel='Month'>
```

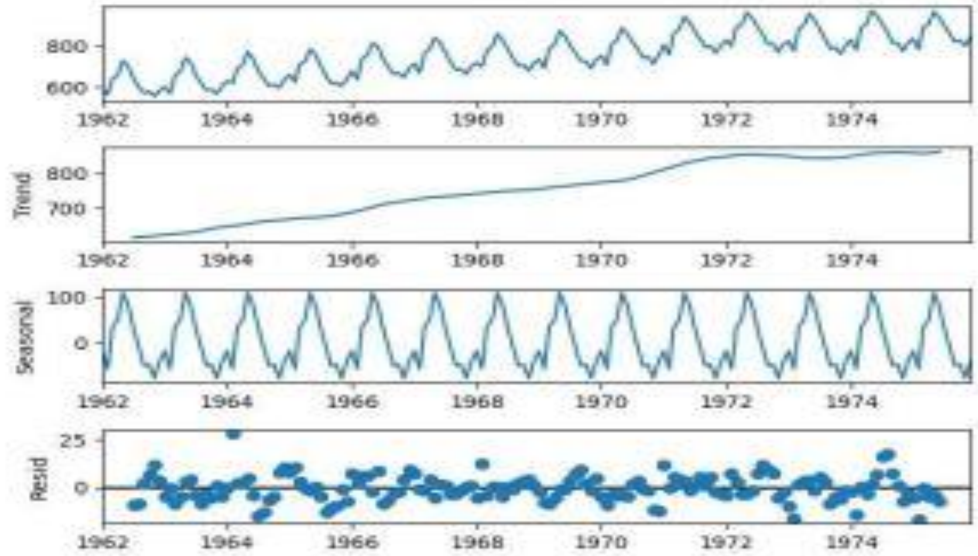


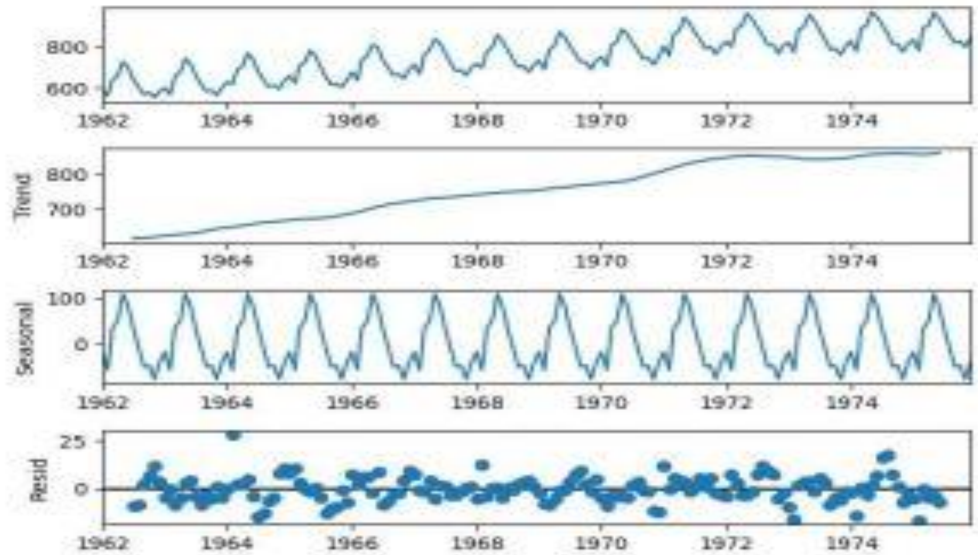
Step 4: create season decompose

In [8]:

```
decomposition = seasonal_decompose(df)
fig=decomposition.plot()
fig
```

Out[8]:





Step 5: create model

In [17]:
```
model = SARIMAX(df['Milk Production'], order = (1,1,0), seasonal_order = (0
result = model.fit()
result.summary()
```

```
C:\Users\CADD_Cidco\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_m
odel.py:471: ValueWarning: No frequency information was provided, so infer
red frequency MS will be used.
  self._init_dates(dates, freq)
C:\Users\CADD_Cidco\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_m
odel.py:471: ValueWarning: No frequency information was provided, so infer
red frequency MS will be used.
  self._init_dates(dates, freq)
```

Out[17]:

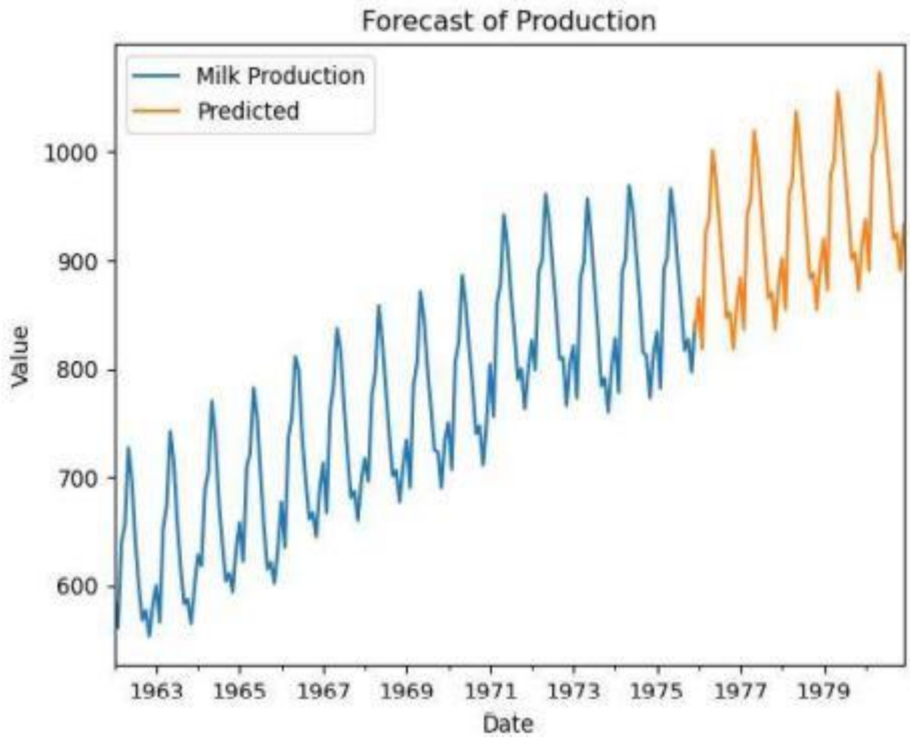
SARIMAX Results

Dep. Variable:	Milk Production	**No. Observations:**	168
Model:	SARIMAX(1, 1, 0)x(0, 1, [1], 12)	**Log Likelihood**	-530.104
Date:	Tue, 30 Jan 2024	**AIC**	1066.207
Time:	12:21:20	**BIC**	1075.337
Sample:	01-01-1962	**HQIC**	1069.916
	- 12-01-1975		
Covariance Type:	opg		

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
ar.L1	-0.2253	0.077	-2.925	0.003	-0.376	-0.074

Step 6: Predict data next 5 year

In [20]:
```
forecast = result.get_prediction(start = '1975-12-01', end = '1980-12-01')


forecast_values = forecast.predicted_mean


fig, ax = plt.subplots()
df.plot(ax = ax, label='observed')
forecast_values.plot(ax = ax, label = 'Predicted')
ax.set_xlabel('Date')
ax.set_ylabel('Value')
plt.legend()
ax.set_title('Forecast of Production')
plt.show()
```



Tasks:

1. Forecast future sales using the data: (Jan, 100; Feb, 120; Mar, 150).
2. Calculate the moving average for the following demand data: (Week 1, 20; Week 2, 30; Week 3, 40).

Questions:

1. What is forecasting?
2. Why is forecasting important for businesses?

10. Dimensionality Reduction

Objective:

The objective of this lab manual is to introduce dimensionality reduction techniques, particularly **Principal Component Analysis (PCA)**. We will apply these methods using Python on a dataset to reduce the number of features while retaining maximum information.

Step 1: import required library

```
In [1]: # Import necessary libraries
        from sklearn import datasets # to retrieve the iris Dataset
        import pandas as pd # to load the dataframe
        from sklearn.preprocessing import StandardScaler # to standardize the featu
        from sklearn.decomposition import PCA # to apply PCA
        import seaborn as sns # to plot the heat maps
```

Step 2: Load and Explore the Dataset

```
In [3]: #Load the Dataset
        iris = datasets.load_iris()
        #convert the dataset into a pandas data frame
        df = pd.DataFrame(iris['data'], columns = iris['feature_names'])
        #display the head (first 5 rows) of the dataset
        df.head()
```

Out[3]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

Step 3: Standardize the Data

```
In [6]: #Standardize the features
        #Create an object of StandardScaler which is present in sklearn.preprocessi
        scalar = StandardScaler()
        scaled_data = pd.DataFrame(scalar.fit_transform(df)) #scaling the data
        scaled_data
```

Out[6]:

	0	1	2	3
0	-0.900681	1.019004	-1.340227	-1.315444
1	-1.143017	-0.131979	-1.340227	-1.315444
2	-1.385353	0.328414	-1.397064	-1.315444
3	-1.506521	0.098217	-1.283389	-1.315444
4	-1.021849	1.249201	-1.340227	-1.315444
...	...	...	...	...
145	1.038005	-0.131979	0.819596	1.448832
146	0.553333	-1.282963	0.705921	0.922303
147	0.795669	-0.131979	0.819596	1.053935
148	0.432165	0.788808	0.933271	1.448832
149	0.068662	-0.131979	0.762758	0.790671

Step 4: Apply Principal Component Analysis (PCA)

```
In [7]: #Applying PCA
        #Taking no. of Principal Components as 3
        pca = PCA(n_components = 3)
        pca.fit(scaled_data)
        data_pca = pca.transform(scaled_data)
        data_pca = pd.DataFrame(data_pca,columns=['PC1','PC2','PC3'])
        data_pca.head()
```

Out[7]:

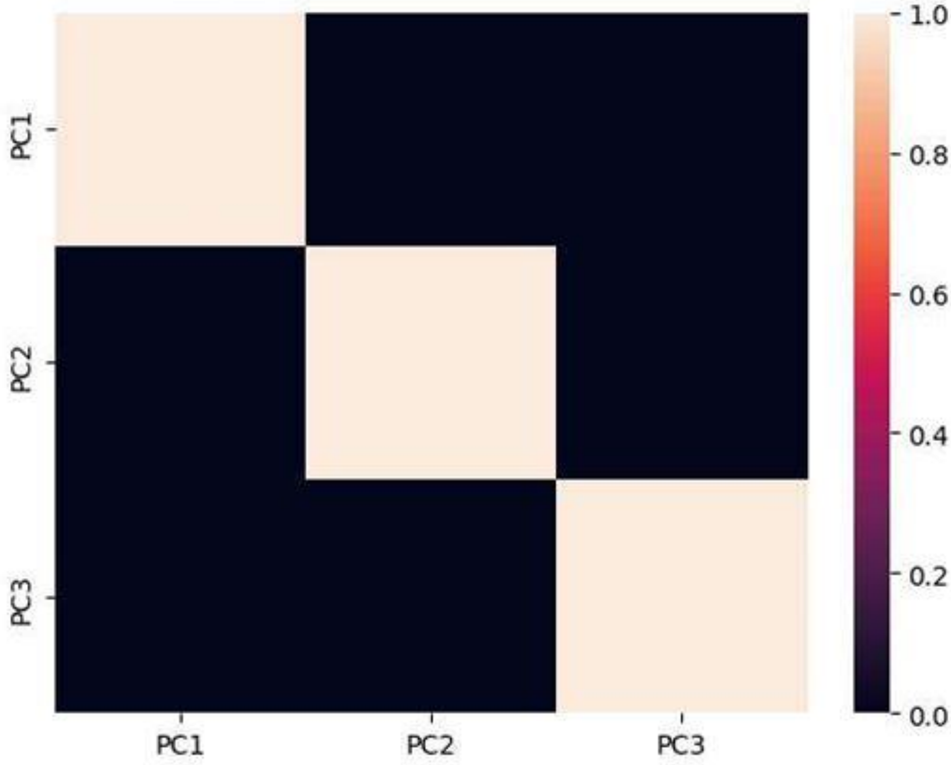
	PC1	PC2	PC3
0	-2.264703	0.480027	-0.127706
1	-2.080961	-0.674134	-0.234609
2	-2.364229	-0.341908	0.044201
3	-2.299384	-0.597395	0.091290
4	-2.389842	0.646835	0.015738

Step 5: Visualize the PCA Results:

In [8]:

```
#Checking Co-relation between features after PCA
sns.heatmap(data_pca.corr())
```

Out[8]: <Axes: >



Tasks:

1. Perform PCA on the dataset (Feature1, Feature2, Feature3) - (2, 4, 6; 3, 6, 9; 5, 7, 11; 8, 10, 14) to reduce its dimensions from 3 to 2.
2. Apply PCA to the dataset (FeatureA, FeatureB, FeatureC) - (5, 10, 15; 10, 15, 20; 15, 20, 25; 20, 25, 30) and visualize the explained variance ratio for each principal component.

Questions:

1. What is dimensionality reduction, and why is it useful?
2. How does PCA help in reducing the number of features in a dataset?

11. Association Rule Mining

Association rule mining is a technique in data mining for discovering interesting relations between variables in large datasets. It's used in market basket analysis, recommendation systems, and pattern recognition.

- **Support**: Frequency of an itemset in the data.
- **Confidence**: How often a rule has been found to be true.
- **Lift**: Measures how much more likely the rule is to occur than if the itemsets were independent.

Lab Requirements

- Anaconda application

Objectives

- Learn to apply association rule mining algorithms.
- Understand the concept of support, confidence, and lift.
- Generate association rules from a dataset.

Step 1: Define Transactions

In [20]:
```
import pandas as pd
#sample transaction data
data = {
    'Transaction ID': [1, 2, 3, 4, 5],
    'Items':[['bread', 'milk', 'eggs'],
             ['bread', 'butter', 'eggs'],
             ['butter', 'milk'],
             ['bread', 'butter', 'milk', 'eggs'],
             ['bread', 'milk', 'butter']]
}
df=pd.DataFrame(data)
df
```

Out[20]:

	Transaction ID	Items
0	1	[bread, milk, eggs]
1	2	[bread, butter, eggs]
2	3	[butter, milk]
3	4	[bread, butter, milk, eggs]
4	5	[bread, milk, butter]

Step 2: Convert Transactions to One-Hot Encoding

In [22]:
```
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori, association_rules

#convert a transaction data to a one-hot enoded format
te= TransactionEncoder()
te_ary = te.fit(df['Items']).transform(df['Items'])
df_encoded =pd.DataFrame(te_ary, columns=te.columns_)

df_encoded
```

Out[22]:

	bread	butter	eggs	milk
0	True	False	True	True
1	True	True	True	False
2	False	True	False	True
3	True	True	True	True
4	True	True	False	True

Step 3: Apply the Apriori Algorithm

In [39]:
```
#use the apriori algorithm tofind frequent itemset
frequent_itemset=apriori(df_encoded,min_support=0.2,use_colnames=True)

#generate asociation rules
rules= association_rules(frequent_itemset,metric='confidence',
                          min_threshold=0.2)

#print the rules
print("Association Rules:")
print(rules)
```

```
Association Rules:
          antecedents          consequents  antecedent support  \
0            (butter)              (bread)                 0.8
1             (bread)             (butter)                 0.8
2              (eggs)              (bread)                 0.6
3             (bread)               (eggs)                 0.8
4              (milk)              (bread)                 0.8
5             (bread)               (milk)                 0.8
6            (butter)               (eggs)                 0.8
7              (eggs)             (butter)                 0.6
8              (milk)             (butter)                 0.8
9            (butter)               (milk)                 0.8
10             (milk)               (eggs)                 0.8
11             (eggs)               (milk)                 0.6
12     (butter, eggs)              (bread)                 0.4
13      (eggs, bread)             (butter)                 0.6
14    (butter, bread)               (eggs)                 0.6
15             (eggs)      (butter, bread)                 0.6
```

Step 4: Generate Association Rules

```
In [42]: rules[
                (rules['confidence'] > 0.75) &
                (rules['lift'] > 1) ]
```

Out[42]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage
2	(eggs)	(bread)	0.6	0.8	0.6	1.0	1.25	0.12
12	(butter, eggs)	(bread)	0.4	0.8	0.4	1.0	1.25	0.08
24	(milk, eggs)	(bread)	0.4	0.8	0.4	1.0	1.25	0.08
36	(milk, eggs, butter)	(bread)	0.2	0.8	0.2	1.0	1.25	0.04

Tasks:

1. Apply the Apriori algorithm to find frequent itemsets from the transaction dataset: (1: Bread, Milk; 2: Bread, Diaper, Beer, Eggs; 3: Milk, Diaper, Beer, Cola; 4: Bread, Milk, Diaper, Beer; 5: Bread, Milk, Cola).
2. Generate association rules from the frequent itemsets {Bread}, {Milk}, {Diaper}, {Beer}, {Cola} with a minimum support of 60% and confidence of 80%.

Questions:

1. What is the purpose of Association Rule Mining in data analysis?
2. How do support, confidence, and lift metrics help evaluate the quality of association rules?