

CHAPTER 1

Introduction to Basic Python

Python Introduction :

Python is a high-level, versatile programming language created by Guido van Rossum in 1991. Known for its simple and readable syntax, Python is an ideal language for beginners and experts alike. It's widely used in various domains like web development, data analysis, artificial intelligence, and scientific computing. Python's extensive libraries and frameworks (like Django and NumPy) make it easy to build and deploy projects efficiently. Its cross-platform compatibility allows Python code to run on multiple operating systems, including Windows, macOS, and Linux. Python's large community ensures there's plenty of resources and support available. Whether you're automating tasks, building apps, or exploring data science, Python's ease of use and flexibility make it a top choice for developers and non-developers worldwide.

Basic Concepts Of Python :

- **Operators :**

- **Arithmetic Operators:** Used for math operations
(`+`, `-`, `*`, `/`, `%`, `**`, `//`)
- **Comparison Operators:** Used to compare values
(`==`, `!=`, `>`, `<`, `>=`, `<=`)
- **Logical Operators:** Used for logical operations
(`and`, `or`, `not`)
- **Assignment Operators:** Used to assign values
(`=`, `+=`, `-=`, `=`, `/=`, etc.)
- **Bitwise Operators:** Used for bit-level operations
(`&`, `|`, `^`, `~`, `<<`, `>>`)
- **Membership Operators:** Used to check if a value is part of a sequence.
(`in`, `not in`)
- **Identity Operators:** Used to compare the memory locations of two objects.
(`is`, `is not`)

- **Control statements :**

- **if Statement** :It is used for Conditional execution. If the condition is true, then the code written in if block is executed. Otherwise, if the condition is false then, the if block is skipped over.

Syntax:

if condition:

 #code to execute

Code:

```
x = 5
```

```
if x > 3:
```

```
    print("x is greater than 3")
```

Output:

```
x is greater than 3
```

- **elif Statement** : It is used to check additional conditions if the initial “if” is False

Syntax:

if condition1:

 # code if condition1 is True

elif condition2:

 # code if condition2 is True

elif condition3:

 # code if condition3 is True

Code:

```
x = 5
```

```
if x > 7:
```

```
    print("x is greater than 7")
```

```
elif x == 5:
```

```
    print("x is equal to 5")
```

Output:

x is equal to 5

- **else Statement** :

Use: Execute code when all preceding conditions are False

Syntax:

if condition:

 # code if condition is True

else:

 # code if condition is False

Code:

```
x = 3
```

```
if x > 7:
```

```
    print("x is greater than 7")
```

```
elif x == 5:
```

```
    print("x is equal to 5")
```

```
else:
```

```
    print("x is some other value")
```

Output:

x is some other value

- **Loops :**

- **for Loop :** It is a control structure that allows for the iteration over an iterable object, such as a list, tuple, or string, enabling the execution of a block of code for each element within that iterable. This facilitates repetitive operations without the need for explicit counter variables.

Syntax :

for variable in iterable:

 # code to execute for each item

Example :

```
for i in range(5):
```

```
    print(i)
```

Output :

0, 1, 2, 3, 4

- **while Loop :** It repeatedly executes a block of code as long as a given condition remains True. This facilitates repetitive operations as long as the condition is met.

Syntax :

while condition:

 # code to execute while condition is True

Example :

```
i = 0
```

```
while i < 3:
```

```
    print(i)
```

```
    i += 1
```

Output :

0

1

2

- **continue Statement**: It is used when we have to skip a code based on the condition. If the condition is TRUE, the block of code is skipped.

Syntax:

```
for i in range(value):  
    if condition:  
        continue
```

Code :

```
for i in range(5):  
    if i == 3:  
        continue  
    print(i)
```

Output :

```
0  
1  
2  
4
```

- **break Statement** : It is used when we have to skip a code based on the condition. If the condition is TRUE, the block of code is skipped.

Syntax :

```
for i in range(value):  
    if condition:  
        break
```

Code:

```
for i in range(5):  
    if i == 3:  
        break  
    print(i)
```

Output :

```
0  
1  
2
```

- **Functions** :

- **Built in Functions** : These are Pre-defined functions that are always available for use, providing a wide range of functionalities such as type conversions (`int()`, `str()`), mathematical operations (`abs()`, `pow()`), sequence operations (`len()`, `max()`), and more. These functions are part of the Python standard library and can be used directly in code to perform common tasks efficiently.

Example :

```
lst= [1, 2, 3, 4, 5]
```

```
length = len(lst)
```

```
print(length)
```

Output :

5

- **User defined Functions** : The user-defined functions are custom functions created by the programmer to encapsulate reusable code, perform specific tasks, and improve code modularity and readability. These functions can take arguments, return values, and be called multiple times from different parts of the program, allowing for efficient code organization and reuse.

Example :

```
lst= [1, 2, 3, 4, 5]
```

```
length = len(lst)
```

```
print(length)
```

Output :

5

- **Data Structures** :

- **String** : Strings are ordered, immutable sequences of characters.

- **Syntax** :

- Stringname= 'Content'**

- **Example** :

- `str= 'Trendwave'`

- Various operations that can be performed on strings :

- **upper()** : Converts all characters to uppercase.

- Example:** ``str.upper()``

- Output:** `'TRENDWAVE'`

- **lower()** : Converts all characters to lowercase.

- Example:** ``str.lower()``

- Output:** `'trendwave'`

- **capitalize()** : Converts the first character to uppercase and the rest to lowercase.

- Example:** ``str.capitalize()``

- Output:** `'Trendwave'`

- **split()** : Splits the string into a list of substrings based on a delimiter.

- Example:** ``str.split('d')``

- Output:** ``['Tren', 'wave']``

- **find('char')** : Returns the index of the first occurrence of the specified character.

- Example:** ``str.find('w')``

- Output:** ``4``

- **replace('old', 'new')** : Replaces occurrences of the old substring with the new substring.

- Example:** ``str.replace('Trend', 'Wave')``

- Output:** ``'Wavewave'``

- **len()** : Returns the length of the string.

- Example:** ``len(str)``

- Output:** ``9``

- **List :** It is ordered, mutable collections of items that can be of any data type.
 - **Syntax:**
`listname=[value1, value2, value3....]`
 - **Example :**
`lst=[1,2,3,4]`
 - Various operations that can be performed on list :
 - **append():** Adds an element to the end.
 Example: ``lst.append(5)``
 Output: ``[1, 2, 3, 4, 5]``
 - **insert():** Inserts an element at a specified index.
 Example: ``lst.insert(2, 10)``
 Output: ``[1, 2, 10, 3, 4]``
 - **remove():** Removes the first occurrence of an element.
 Example: ``lst.remove(3)``
 Output: ``[1, 2, 4]``
 - **pop():** Removes and returns an element at a specified index.
 Example: ``lst.pop(0)``
 Returns: ``1``
 Output: ``[2, 3, 4]``
 - **len():** Returns the length of the list.
 Example: ``len(lst)``
 Output: ``4``
 - **extend():** Adds multiple elements to the end.
 Example: ``lst.extend([5, 6])``
 Output: ``[1, 2, 3, 4, 5, 6]``
 - **index():** Returns the index of the first occurrence of an element.
 Example: ``lst.index(2)``
 Output: ``1``
 - **reverse():** Reverses the list in-place.
 Example: ``lst.reverse()``
 Output: ``[4, 3, 2, 1]``
 - **clear():** Removes all elements from the list.
 Example: ``lst.clear()``
 Output: ``[]``

- **Tuples:** It is an ordered, immutable collection of items that can be of any data type.

- **Syntax:**

``tuplename = (value1, value2, value3, ...)``

- **Example:**

``tpl = (1, 2, "A", "B", 3)``

- Various operations that can be performed on tuple:

- **index():** Returns the index of the first occurrence of an element.

Example: ``tpl.index(2)``

Output: ``1``

- **len():** Returns the length of the tuple.

Example: ``len(tpl)``

Output: ``5``

- **count():** Returns the number of occurrences of a specified element.

Example: ``tpl.count("A")``

Output: ``1``

- **Dictionary:** It is an unordered, mutable collection of key-value pairs.

- **Syntax:**

`dictname = {key1: value1, key2: value2, ...}`

- **Example:**

`dct = {1: "Apple", 2: "Mango", 3: "Grapes"}`

- Various operations that can be performed on dictionary:

- **get():** Returns the value associated with a specified key.

Example: `dct.get(1)`

Output: `"Apple"`

- **update():** Updates the dictionary with new key-value pairs.

Example: `dct.update({4: "Banana"})`

Output: `{1: "Apple", 2: "Mango", 3: "Grapes", 4: "Banana"}`

- **pop():** Removes and returns the value associated with a specified key.

Example: `dct.pop(1)`

Returns: `"Apple"`

Output: `{2: "Mango", 3: "Grapes"}`

- **keys()**: Returns a view object displaying a list of all keys.

Example: `list(dct.keys())`

Output: `[1, 2, 3]`

- **values()**: Returns a view object displaying a list of all values.

Example: `list(dct.values())`

Output: `["Apple", "Mango", "Grapes"]`

- **len()**: Returns the number of key-value pairs in the dictionary.

Example: `len(dct)`

Output: `3`

CHAPTER 2

SQLite

Introduction:

SQLite is a lightweight, self-contained, serverless database engine that can be used within Python applications. It's an optimal option for small to medium-sized projects due to its simplicity and ease of use.

- **Syntax to import:**

```
import sqlite3 as sq
```

- **Operations:**

- Connecting to database
- Creating a cursor
- Executing queries
- Fetching results

- **Key Details:**

- **sqlite3 Module:** Python has a built-in module called `sqlite3` that allows you to interact with SQLite databases.
- **Connecting to a Database:** You can connect to an SQLite database using `sqlite3.connect('database_name.db')`. If the database doesn't exist, it will be created.
- **Cursor Object:** A cursor object is used to execute SQL queries and fetch results. You can create a cursor using `conn.cursor()`.
- **Executing Queries:** You can execute SQL queries using `cursor.execute()` or `cursor.executemany()` for multiple queries.
- **Committing Changes:** To save changes, you need to commit them using `conn.commit()`.
- **Closing the Connection:** Finally, close the connection using `conn.close()` to free up resources.

Code:

```
import sqlite3 as sq

# Connect to the database
conn = sq.connect('example.db')

# Create a cursor
cursor = conn.cursor()

# Create a table
cursor.execute("""
    CREATE TABLE IF NOT EXISTS users (
        id INTEGER PRIMARY KEY,
        name TEXT,
        age INTEGER
    )
""")

# Insert a record
cursor.execute("INSERT INTO users (name, age) VALUES ('John Doe', 30)")

# Commit changes
conn.commit()

# Close the connection
conn.close()
```

Explanation:

This code will create a new SQLite database named 'example.db' if it doesn't already exist, and then create a table named 'users' with columns for 'id', 'name', and 'age' if the table doesn't already exist. It will then insert a new record into the 'users' table with the name 'John Doe' and age 30. Finally, it will save the changes and close the connection to the database. Process completed successfully. Operation finished.

CHAPTER 3

Advanced Python

Introduction:

Python libraries are superpower toolboxes that make coding easier and faster . They're pre-written code collections that save time and effort, letting you focus on your program's main logic , and they come in types like data science (Pandas, NumPy).

- **NumPy** : It is a powerful library for working with arrays and matrices in Python. It's a foundation for most scientific computing and data analysis tasks in Python.

- **Key features:**

- **Efficient arrays** : NumPy arrays are faster and more memory-efficient than Python lists.
- **Vectorized operations** : NumPy allows you to perform operations on entire arrays at once, making code more concise and faster.
- **Mathematical functions** : NumPy has a wide range of built-in mathematical functions for trigonometry, statistics, and more.

- **Import:**

import numpy as np

- **Different Functions:**

- **numpy.array()**: Creates a NumPy array from a given object.

Syntax: np.array(object)

- **numpy.linspace()**: Returns evenly spaced numbers over a specified interval.

Syntax: np.linspace(start, stop, num=50)

- **numpy.arange()**: Creates an array with a range of values.

Syntax: np.arange(start, stop, step)

- **numpy.zeros()**: Creates an array filled with zeros.

Syntax: np.zeros(shape)

- **numpy.ones():** Creates an array filled with ones.

Syntax: `np.ones(shape)`

- **numpy.log():** Computes the natural logarithm of all elements in the input array.

Syntax: `np.log(x)`

- **numpy.mean():** Computes the mean of an array.

Syntax: `np.mean(array)`

- **numpy.max():** Computes the maximum value of an array.

Syntax: `np.max(array)`

- **numpy.min():** Computes the minimum value of an array.

Syntax: `np.min(array)`

- **numpy.reshape():** Reshapes an array to a new shape.

Syntax: `np.reshape(array, newshape)`

- **numpy.transpose():** Transposes an array.

Syntax: `np.transpose(array)`

- **Pandas:** It is a super powerful Python library used for data manipulation and analysis.

- **Key Features:**

- DataFrames: Pandas' core data structure, allowing you to store and manipulate tabular data like spreadsheets or SQL tables.
- Data Cleaning: Easily handle missing data, data filtering, and data transformation.
- Data Merging: Combine DataFrames based on common columns or indices.
- Data Analysis: Perform operations like grouping, sorting, and aggregating data.
- Makes working with structured data a breeze.
- Integrates well with other popular libraries like NumPy, Matplotlib, and Scikit-learn.
- Highly versatile and widely used in data science, science, and engineering.

- **Import:**

Creating Dataframe :

```
data = {'Fruits': ['Mango', 'Apple'], 'Vegetables': ['Potato', 'Carrot']}
```

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

- **Different functions:**

- **read_csv():** Reads a CSV file into a DataFrame.

Syntax: `pd.read_csv('file_path')`

- **head():** Displays the first few rows of a DataFrame.

Syntax: `df.head(n)`

- **tail():** Displays the last few rows of a DataFrame.

Syntax: `df.tail(n)`

- **info():** Provides a concise summary of a DataFrame.

Syntax: `df.info()`

- **describe():** Generates descriptive statistics for a DataFrame.

Syntax: `df.describe()`

- **dropna():** Removes rows with missing values from a DataFrame.

Syntax: `df.dropna(axis=0, how='any')`

- **fillna():** Replaces missing values in a DataFrame.

Syntax: `df.fillna(value)`

- **groupby():** Groups a DataFrame by one or more columns.

Syntax: `df.groupby('column_name')`

- **merge():** Merges two DataFrames based on a common column.

Syntax: `pd.merge(df1, df2, on='common_column')`

- **Matplotlib :**

- **Introduction:**

It is a popular Python library used for creating static, animated, and interactive visualizations. It's widely used for data visualization and is a key component of the Python data science ecosystem. Matplotlib offers a wide range of tools for creating high-quality 2D and 3D plots, charts, and graphs. Some key features of Matplotlib include:

- Creating line plots, scatter plots, histograms, and more
- Customizing plot appearance with colors, fonts, and labels
- Saving plots to various file formats, such as PNG and PDF

- **Import:**

`import matplotlib.pyplot as plt`

- **Types of graphs that can be created using matplotlib :**

- **Line Plot:** `plt.plot(x, y)`
- **Scatter Plot:** `plt.scatter(x, y)`
- **Bar Chart:** `plt.bar(x, y)`
- **Histogram:** `plt.hist(x, bins=n)`
- **Pie Chart:** `plt.pie(x)`
- **Box Plot:** `plt.boxplot(x)`
- **Violin Plot:** `plt.violinplot(x)`

- **Examples :**

- **Plot Graph :**

```
import pandas as pd

import matplotlib.pyplot as plt

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

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

df['year'] = df['ts'].dt.year

artist_counts_per_year = df.groupby('year')['artist_name'].nunique()

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

plt.plot(artist_counts_per_year.index, artist_counts_per_year.values,
marker='o', linestyle='-')

plt.title('Count of Distinct Artists per Year')

plt.xlabel('Year')

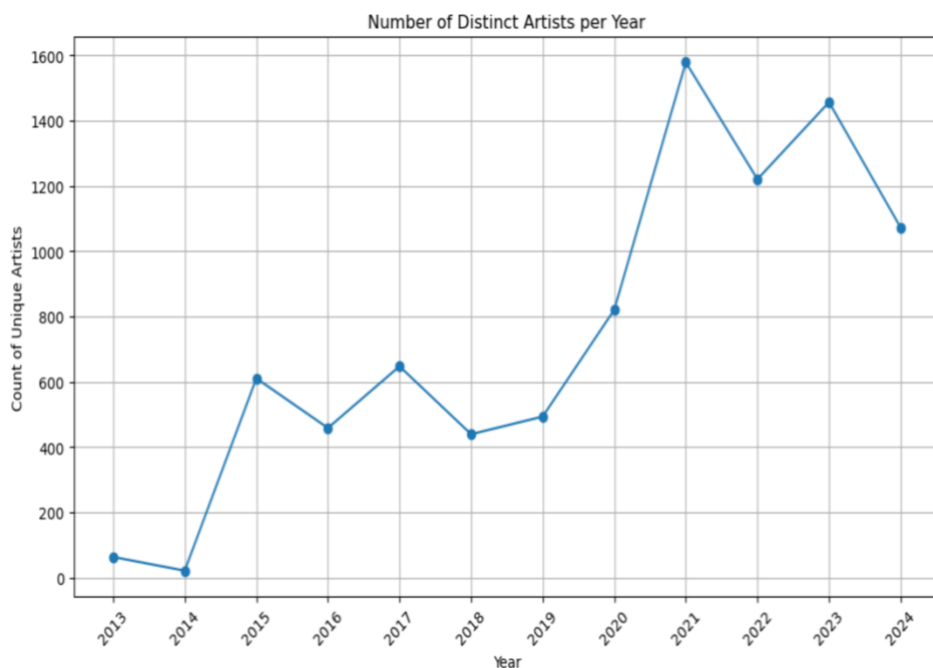
plt.ylabel('Count of Unique Artists')

plt.grid(True)

plt.xticks(artist_counts_per_year.index, rotation=45) # rotate years if needed

plt.tight_layout()

Plt.show()
```



- **Horizontal Bar Graph :**

```
import pandas as pd

import matplotlib.pyplot as plt

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

track_counts = df['track_name'].value_counts()

top_5_tracks = track_counts.head(5)

plt.figure(figsize=(8,5))

plt.barh(top_5_tracks.index, top_5_tracks.values, color='skyblue')

plt.xlabel('Count')

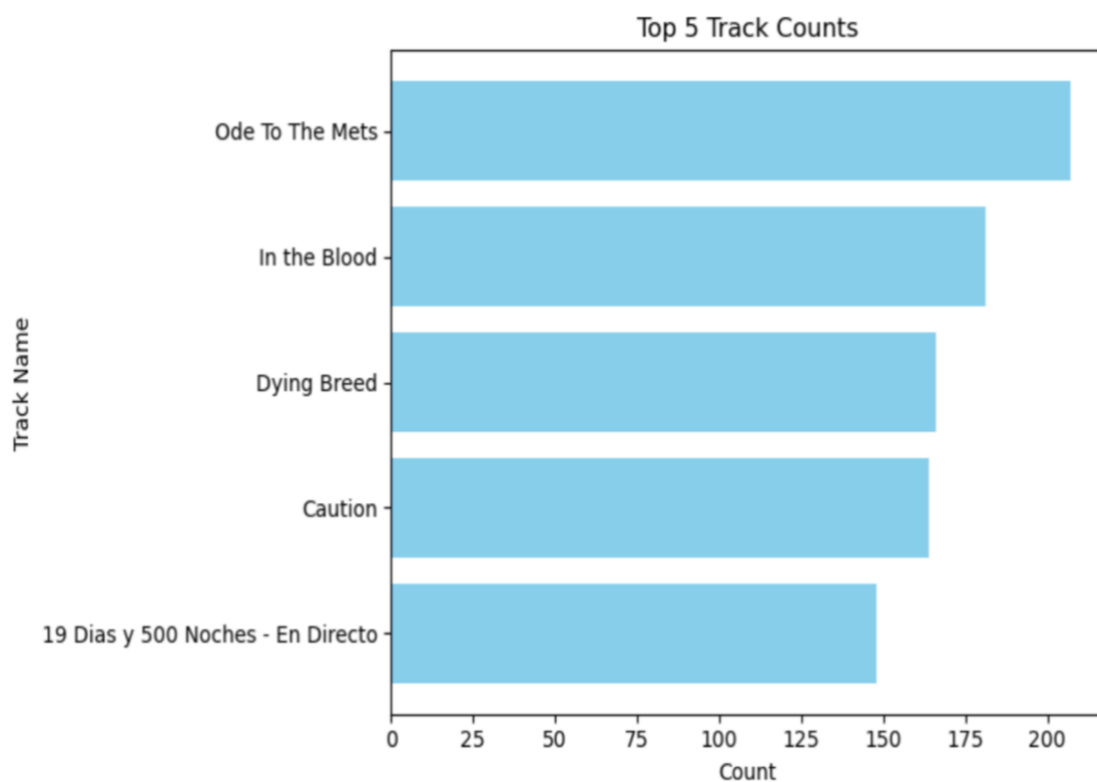
plt.ylabel('Track Name')

plt.title('Top 5 Track Counts')

plt.gca().invert_yaxis() # Highest count on top

plt.tight_layout()

plt.show()
```



CHAPTER 4

Advanced Excel

Introduction to Excel:

Microsoft Excel is a powerful spreadsheet software that allows users to store, organize, and analyze data. It is widely used in various industries for financial analysis, data management, and reporting. Excel provides a grid-like interface where data can be entered, calculated, and visualized through charts and graphs. Its versatility and functionality make it an essential tool for businesses and individuals alike. Excel also supports various formulas and functions, enabling users to perform complex calculations and data analysis tasks efficiently. This makes it a go-to solution for tasks like budgeting, forecasting, and reporting.

- **Basic Excel Functions :**

- **SUM Function:** The SUM function is used to calculate the total of a range of cells.

Syntax: =SUM(range)

- **AVERAGE Function:** The AVERAGE function calculates the average of a range of cells.

Syntax: =AVERAGE(range)

- **COUNT Function:** The COUNT function counts the number of cells in a range that contain numbers.

Syntax: =COUNT(range)

- **MAX Function:** The MAX function return the maximum values in a range of cells.

Syntax: =MAX(range)

- **MIN Function:** The MIN function return the minimum values in a range of cells.

Syntax: =MIN(range)

- **TODAY():** Returns the current date.

Syntax: =TODAY()

- **NOW():** Returns the current date and time.

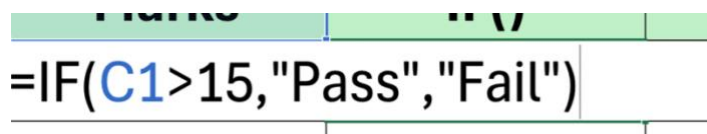
Syntax: =NOW()

- **SUM()**: Calculates the total of a range of cells.
Syntax: =SUM(range)
- **COUNTIF()**: Counts cells in a range that meet a specific condition.
Syntax: =COUNTIF(range, criteria)
- **AVERAGEIF()**: Calculates the average of cells in a range that meet a specific condition.
Syntax: =AVERAGEIF(range, criteria, [average_range])
- **ROUND()**: Rounds a number to a specified number of digits.
Syntax: =ROUND(number, num_digits)
- **INT()**: Returns the integer part of a number.
Syntax: =INT(number)

- **Different Logical Functions In Excel :**

- **IF()**: Returns a value based on condition.

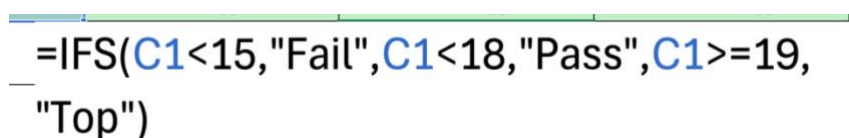
Syntax: =IF(condition, "If condition True", "If condition FALSE")



=IF(C1>15,"Pass","Fail")

- **IFS()** : Apply multiple If statements.

Syntax: =IFS(condition1, "If condition1 True", condition2, "If condition2 True", condition3, "If condition3 True")



=IFS(C1<15,"Fail",C1<18,"Pass",C1>=19,"Top")

- **AND()** : Returns a value if both given conditions are satisfied.

Syntax: =IF(AND(condition1, condition2) "If condition True", "If condition FALSE")

```
=IF(AND(C2>15,B2="da"),"Valid",
"Invalid")
```

- **OR()** : Returns a value if either of the conditions are satisfied.

Syntax: =IF(OR(condition1, condition2) "If condition True", "If condition FALSE")

```
=IF(OR(C2>15,B2="da"),"Valid","Invalid")
```

- **NOT()** : Reverses the logic

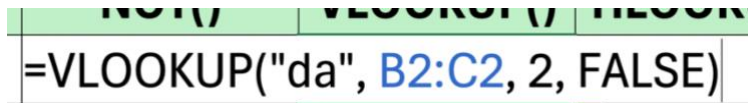
Syntax: =IF(NOT(condition), "If condition True", "If condition FALSE")

```
=NOT(C12>15)
```

- **Different Lookup Functions In Excel :**

- **VLOOKUP()**: It is performed vertically.

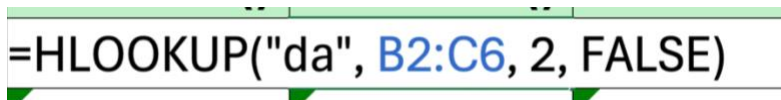
Syntax: =VLOOKUP(find_data, where_to_find_data :
value_returned_from, no_of_columns_used , FALSE/TRUE)



=VLOOKUP("da", B2:C2, 2, FALSE)

- **HLOOKUP()** : It is performed horizontally.

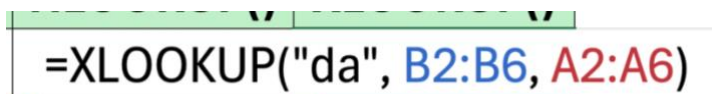
Syntax: = HLOOKUP(find_data, where_to_find_data :
value_returned_from, no_of_columns_used ,FALSE/TRUE)



=HLOOKUP("da", B2:C6, 2, FALSE)

- **XLOOKUP()** : It can search in any direction

Syntax: =XLOOKUP(data_to_search, where_to_search_range ,
where_to_return_from)



=XLOOKUP("da", B2:B6, A2:A6)

CHAPTER 5

POWER BI

Introduction :

It is a powerful business analytics service by Microsoft that enables organizations to visualize and analyze data, making better-informed decisions. It connects to various data sources, creates interactive dashboards, and provides business intelligence insights. Power BI helps users transform raw data into meaningful reports and visualizations, facilitating data-driven decision-making. Its key features include data modeling, data visualization, and business analytics tools. Power BI is widely used across industries for its ability to simplify complex data, identify trends, and drive business outcomes. With its user-friendly interface and robust capabilities, Power BI is an essential tool for businesses seeking to leverage data analytics and stay competitive. It integrates well with other Microsoft products, making it a popular choice for organizations already using Microsoft solutions. By using Power BI, businesses can gain deeper insights, improve operational efficiency, and enhance overall performance.

• Features :

- **Data Visualization:** Power BI offers a wide range of visualizations to help users create interactive and meaningful reports.
- **Data Modeling:** Allows users to create data models and relationships between different data sources.
- **Business Analytics:** Provides built-in analytics tools to help users analyze data and gain insights.
- **Data Connectivity :** Connects to various data sources, including Excel, SQL, and cloud-based services.
- **Dashboards :** Enables users to create custom dashboards to monitor key performance indicators (KPIs).
- **Reports :** Allows users to create interactive reports that can be shared across the organization.
- **Mobile Access :** Power BI has mobile apps for iOS and Android, enabling users to access reports on-the-go.
- **Collaboration :** Facilitates collaboration through features like sharing and commenting on reports.
- **Security :** Offers enterprise-grade security features to protect sensitive data.
- **Integration :** Integrates well with other Microsoft products, such as Excel and Azure.

CHAPTER 6

DATASET

Introduction :

This project involves a comprehensive analysis of the `spotify_history.csv` dataset, spanning 11 years from 2013 to 2024, to examine Spotify's various aspects including album release trends, track popularity patterns, and artist popularity evolution. The analysis covers albums, tracks, and artists to address key questions such as the most popular albums and tracks over the years, changes in artist popularity trends since 2013, and dominant genres on Spotify during this period, providing valuable insights into historical trends and patterns on Spotify within the scope of this report.

- **Details :**

- **spotify_track_uri :**

- **Description:** A unique identifier assigned to each track in Spotify's database.
- **Format:** "spotify:track:<base-62 string>" (e.g.,
spotify:track:3n3Ppam7vgaValiaRUc9Lp).
- **Purpose:** Helps link tracks to their metadata and allows for cross-referencing with Spotify's catalog.

- **ts (Timestamp) :**

- **Description:** The exact time (in UTC) when the track stopped playing.
- **Format:** ISO 8601 format (e.g., 2024-02-07T14:30:45Z).
- **Purpose:** Used for analyzing listening patterns, session durations, and track end times.

- **platform :**

- **Description:** The device or platform used to stream the track.
- **Possible Values:**

"desktop" (Windows/Mac app)

"mobile" (iOS/Android app)

"web" (Spotify Web Player)

"smart_speaker" (Amazon Echo, Google Home, etc.)

- **Purpose:** Helps understand where users are consuming music.

- **ms_played :**
 - **Description:** The total number of milliseconds the track was played before stopping or skipping.
 - **Format:** Integer value (e.g., 215000 for 3 minutes 35 seconds).
 - **Purpose:** Useful for engagement analysis, identifying completed plays, and calculating revenue (as Spotify pays artists based on streaming duration).

- **track_name:**
 - **Description:** The title of the song being played.
 - **Example:** "Shape of You"
 - **Purpose:** Helps in analyzing the most played tracks.

- **artist_name:**
 - **Description:** The name of the artist performing the song.
 - **Example:** "Ed Sheeran"
 - **Purpose:** Useful for ranking popular artists and identifying user preferences.

- **album_name :**
 - **Description:** The name of the album the track belongs to.
 - **Example:** "÷ (Divide) "
 - **Purpose:** Helps analyze album popularity and user listening trends.

- **reason_start :**
 - **Description:** The reason why the track started playing.
 - **Possible Values:**
 - "trackdone" (Previous track ended)
 - "clickrow" (User manually selected the song)
 - "backbtn" (User pressed back)
 - "fwdbtn" (User pressed next)
 - "playbtn" (User pressed play)
 - "autoplay" (Spotify automatically selected the next track)
 - **Purpose:** Helps understand user behavior and track engagement patterns.

- **reason_end :**
 - **Description:** The reason why the track stopped playing.
 - **Possible Values:**
 - "trackdone" (Track finished playing)
 - "endplay" (User paused or stopped playback)
 - "fwdbtn" (User skipped to the next track)
 - "backbtn" (User went back to the previous track)
 - "logout" (User logged out or session ended)
 - **Purpose:** Helps identify why users stop listening to tracks, which is crucial for retention analysis.

- **shuffle**
 - **Description:** Indicates whether shuffle mode was enabled during playback.
 - **Possible Values:**
 - TRUE (Shuffle mode was ON)
 - FALSE (Shuffle mode was OFF)
 - **Purpose:** Helps analyze how often users use shuffle mode in their listening sessions.

- **skipped**
 - **Description:** Indicates whether the user skipped the song before it finished.
 - **Possible Values:**
 - TRUE (User skipped to the next track)
 - FALSE (User did not skip)
 - **Purpose:** Important for understanding user engagement, drop-off rates, and song popularity.

CHAPTER 7

DATA HANDLING

Introduction :

Each step is performed to ensure data quality and reliability. This includes importing necessary libraries, loading and cleaning the data, handling missing values, correcting errors, and transforming data types. These steps prepare the data for analysis, ensuring insights are accurate and meaningful. This systematic approach guarantees the dataset is reliable for extracting valuable insights about Spotify's history.

DATA CLEANING :

Step 1: Import Libraries

Code: `import pandas as pd`

Explanation: This step imports the pandas library, which is a crucial tool for data manipulation and analysis. Pandas enables efficient handling of the dataset, providing data structures like dataframes that simplify complex data operations, making it a foundational step for the entire data processing workflow.

Step 2 : Import file as a dataframe

Code: `file = pd.read_csv('spotify_history.csv')`
`df = pd.DataFrame(file)`

Explanation: This step loads the `spotify_history.csv` dataset into a dataframe. By doing so, it makes the dataset accessible for manipulation and analysis within the Python environment. This is essential for initiating any data analysis project, as it allows for the application of pandas' powerful data manipulation capabilities.

Step 3: Retrieve basic information and summary

Code: `df.info()`

`df.describe()`

Explanation: Retrieving basic information (like data types and counts) via `df.info()` and summary statistics (like mean and standard deviation) via `df.describe()` provides an initial understanding of the dataset's structure and content. This step is vital for gaining insights into the dataset's characteristics, identifying potential issues, and guiding subsequent data cleaning and preprocessing steps.

Step 4: Handling missing data values.

Code: `df['reason_end'] = df['reason_end'].fillna('Unknown')`

`df['reason_start'] = df['reason_start'].fillna('Unknown')`

Explanation: Handling missing values in the `reason_end` and `reason_start` columns by replacing them with 'Unknown' ensures these fields are complete for analysis. This step is crucial for preventing issues that could arise from missing data, such as biased analyses or errors in data processing pipelines.

Step 5: Handling wrong values.

Code: `df['ts'] = df['ts'].replace('Not Available', pd.NaT)`

`df['ts'] = pd.to_datetime(df['ts'], errors='coerce')`

Explanation: Correcting 'Not Available' values in the `ts` column by replacing them with `pd.NaT` (Not a Time) and then converting the `ts` column to datetime format facilitates time-based analyses. This conversion is essential for enabling temporal analyses, such as trend identification over time, which are key aspects of understanding Spotify's historical data.

Step 6: Clean text columns.

Code: `df['track_name'] = df['track_name'].str.strip()`

Explanation: Cleaning the `track_name` column by removing leading and trailing whitespace ensures accurate matching and grouping of track names. This step is important for maintaining data consistency, which is critical for accurate analyses, such as identifying the most popular tracks or analyzing track naming trends.

Step 7: Reindex and sort

Code: `df.reset_index(drop=True, inplace=True)`

Explanation: Reindexing the dataframe to ensure a continuous index is crucial for sorting and further analysis. This step ensures that the data is organized and accessible, facilitating subsequent operations like data filtering or the application of analytical models.

Step 8: Save cleaned data.

Code: `df.to_csv("cleaned_data.csv", index=False)`

Explanation: Saving the cleaned and processed data to a new CSV file named "cleaned_data.csv", without including the index, makes the dataset ready for subsequent analysis or reporting. This final step ensures that the cleaned data is preserved and readily available for further use, completing the data preparation phase of the project.

CHAPTER 8

DATA VISUALIZATION

Introduction :

Data visualization is crucial because it transforms complex data into clear, actionable insights. By presenting data in visual formats like charts, graphs, and maps, it helps identify patterns, trends, and outliers that might be missed in raw data. Visualization enhances understanding and facilitates communication, making it easier for stakeholders to grasp key findings and make informed decisions. It also aids in storytelling, allowing analysts to convey the narrative behind the data more effectively. Moreover, data visualization supports exploratory data analysis, enabling users to interactively explore data, discover hidden insights, and formulate hypotheses. Overall, data visualization is an essential tool in data analysis, bridging the gap between data and decision making, and driving more intuitive and evidence-based outcomes.

Data Visualization ways :

We can utilize both Power BI and Matplotlib to create visualizations. Both tools offer robust capabilities for transforming data into insightful visual representations, catering to different user preferences and needs.

If we have to choose one then power bi will be an better option than Matplotlib for several reasons. Power BI provides a more user-friendly, drag-and-drop interface that requires minimal coding knowledge, making it accessible to a broader range of users. Additionally, Power BI offers seamless integration with various data sources and business tools, facilitating the creation of comprehensive, interactive dashboards that support real-time decision-making. Its advanced features and enterprise-focused functionalities make it particularly suited for business intelligence applications, whereas Matplotlib, being a Python library, is more geared towards data scientists and programmers who prefer coding for customization and analysis.

- **Plotting graph using Matplotlib :**

```
import matplotlib.pyplot as plt

import numpy as np

# Data for the line graphs

years = np.arange(2014, 2025)

albums_played = np.array([22, 100, 500, 1200, 1800, 2200, 2500, 2668, 2300, 2000, 1824])

artists_played = np.array([21, 100, 400, 900, 1400, 1600, 1500, 1578, 1400, 1200, 1071])

tracks_played = np.array([23, 100, 500, 1500, 3000, 4000, 5106, 4800, 4500, 3800, 3568])

# Plot the line graphs

axs[0].plot(years, albums_played)

axs[0].set_title('Albums Played Over Time')

axs[0].set_xlabel('Year')

axs[0].set_ylabel('Albums Played')

axs[0].set_ylim([0, 3000])

axs[1].plot(years, artists_played)

axs[1].set_title('Artists Played Over Time')

axs[1].set_xlabel('Year')

axs[1].set_ylabel('Artists Played')

axs[1].set_ylim([0, 2000])

axs[2].plot(years, tracks_played)

axs[2].set_title('Tracks Played Over Time')

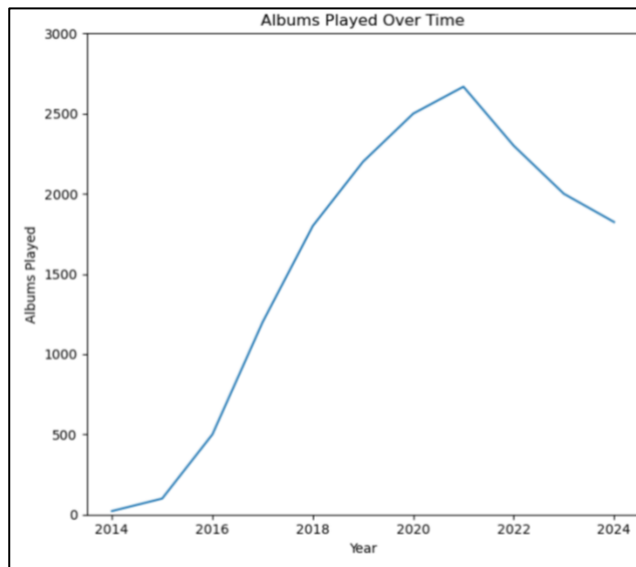
axs[2].set_xlabel('Year')

axs[2].set_ylabel('Tracks Played')

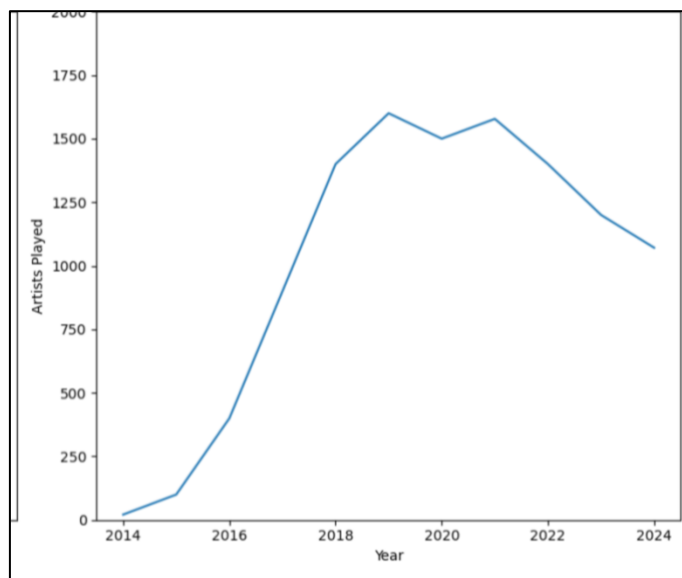
axs[2].set_ylim([0, 6000])

plt.tight_layout()
```

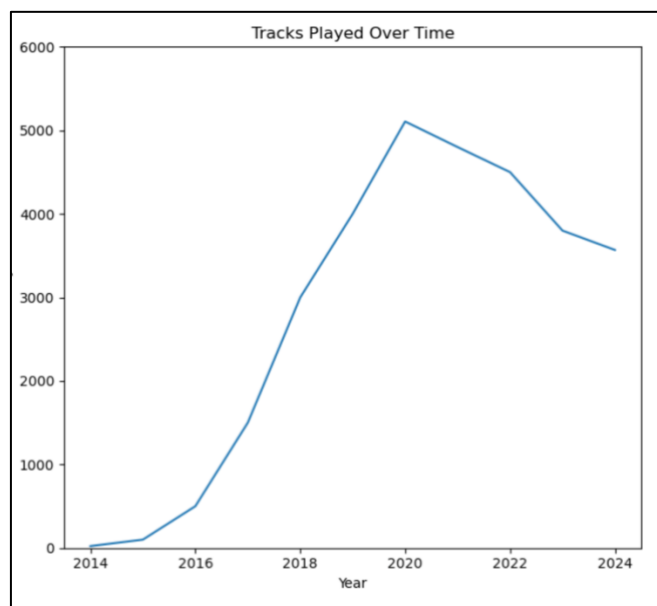

ALBUMS PLAYED OVER YEAR



ARTISTS PLAYED OVER YEAR



TRACKS PLAYED PER YEAR

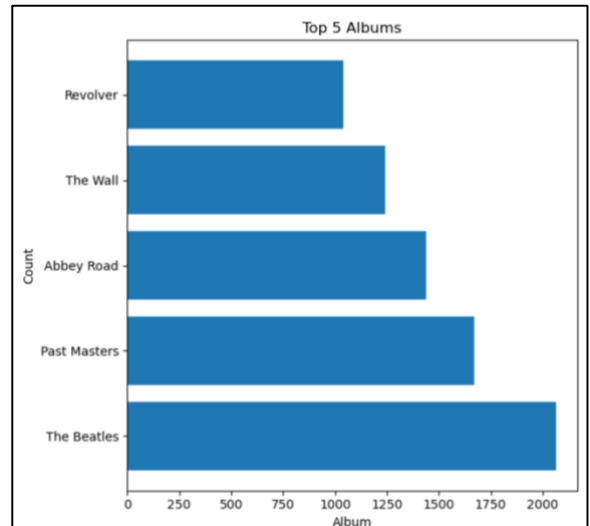


```

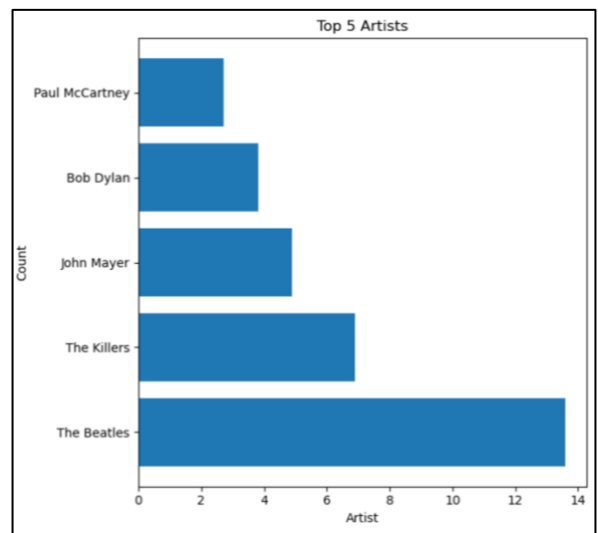
# Create bar charts for the 'Top 5 Albums', 'Top 5 Artists', and 'Top 5 Tracks' sections
top_albums = ['The Beatles', 'Past Masters', 'Abbey Road', 'The Wall', 'Revolver']
top_albums_count = [2063, 1672, 1439, 1241, 1038]
top_artists = ['The Beatles', 'The Killers', 'John Mayer', 'Bob Dylan', 'Paul McCartney']
top_artists_count = [13.6, 6.9, 4.9, 3.8, 2.7]
top_tracks = ['Ode To The Mets', 'In the Blood', 'Dying Breed', 'Caution', '19 Dias y 500 N...']
top_tracks_count = [207, 181, 166, 164, 148]
fig3, axs3 = plt.subplots(1, 3, figsize=(20, 6))
axs3[0].barh(top_albums, top_albums_count)
axs3[0].set_title('Top 5 Albums')
axs3[0].set_xlabel('Album')
axs3[0].set_ylabel('Count')
axs3[1].barh(top_artists, top_artists_count)
axs3[1].set_title('Top 5 Artists')
axs3[1].set_xlabel('Artist')
axs3[1].set_ylabel('Count')
axs3[2].barh(top_tracks, top_tracks_count)
axs3[2].set_title('Top 5 Tracks')
axs3[2].set_xlabel('Track')
axs3[2].set_ylabel('Count')
plt.tight_layout()
plt.show()

```

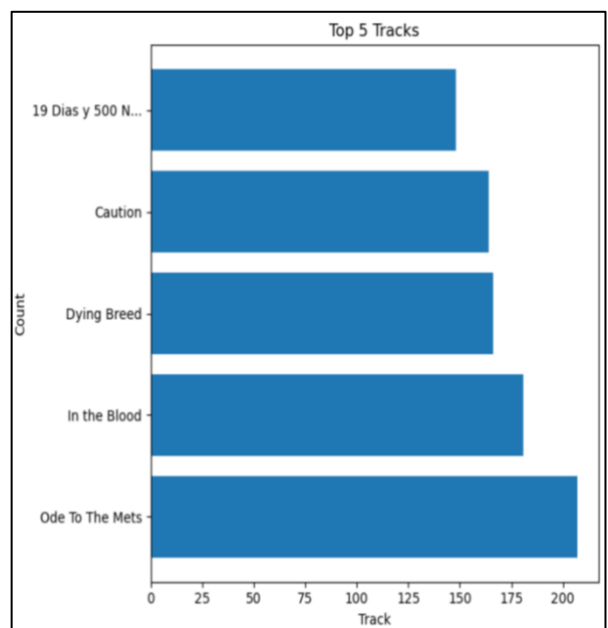
TOP 5 ALBUMS



TOP 5 ARTISTS



TOP 5 TRACKS



CHAPTER 9

DASHBOARD AND INSIGHTS

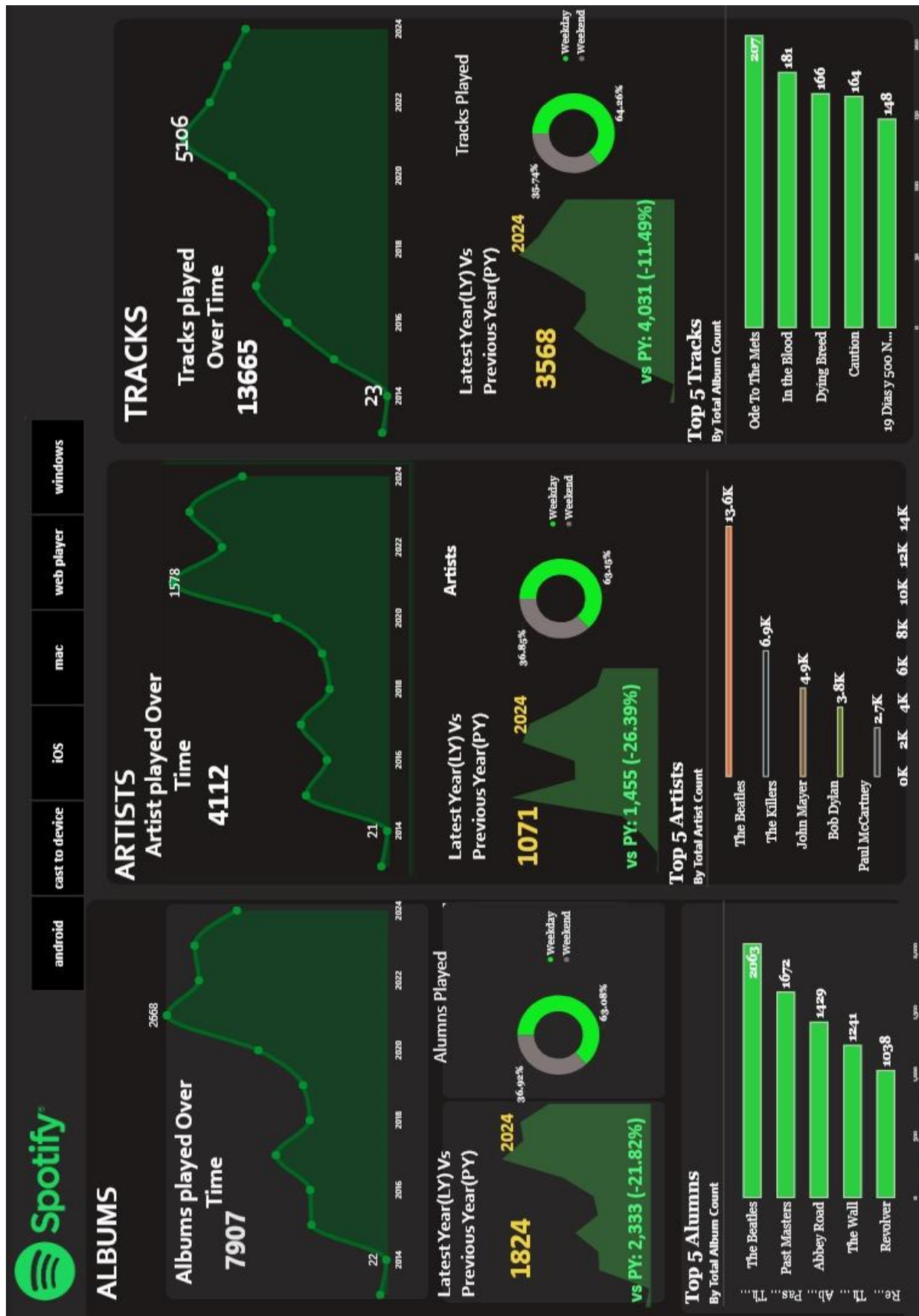
Introduction :

A dashboard is a visual interface that consolidates and displays key performance indicators (KPIs), metrics, and data points from various sources into a single, centralized location. It provides a clear and concise overview of performance, allowing users to monitor, analyze, and make informed decisions. Dashboards can be customized to suit specific needs and can include charts, graphs, tables, maps, and other visual elements. They are commonly used in business intelligence, data analysis, and management to track progress, identify trends, and pinpoint areas for improvement. By presenting complex data in an intuitive and accessible way, dashboards facilitate data-driven decision-making and enhance organizational visibility. They can be designed for various purposes, such as sales, marketing, finance, or operations, catering to different user roles and requirements.

Importance :

The importance of a dashboard lies in its ability to streamline data access and enhance decision-making. By consolidating key metrics and KPIs into a single, visual interface, dashboards provide users with a clear and instant overview of performance. This enables organizations to monitor progress in real-time, identify trends and patterns, and pinpoint areas for improvement. Dashboards facilitate data-driven decision-making by presenting complex data in an intuitive and accessible way, reducing the need for manual data analysis. They also improve communication and collaboration among teams by providing a shared understanding of goals and performance. Overall, dashboards are a vital tool for organizations seeking to boost efficiency, productivity, and informed decision-making, ultimately driving business success.

DASHBOARD :



INSIGHTS :

The provided Spotify dashboard offers a comprehensive overview of a user's listening habits, providing insights into their music preferences and behavior. The dashboard is divided into three main sections: Albums, Artists, and Tracks.

Albums :

The Albums section reveals that the user has played a total of 7,907 albums over time, with a peak of 2,668 albums played in 2021. The graph shows a steady increase in album plays from 2014 to 2021, followed by a decline in 2022 and 2023. The latest year (2024) saw 1,824 albums played, representing a 21.82% decrease compared to the previous year (2,333 albums). The top 5 albums by total album count are:

1. The Beatles - 2,063
2. Past Masters - 1,672
3. Abbey Road - 1,439
4. The Wall - 1,241
5. Revolver - 1,038

These results suggest that the user has a strong affinity for The Beatles, with multiple albums featuring in the top 5.

Artists :

The Artists section indicates that the user has played a total of 4,112 artists over time, with a peak of 1,578 artists played in 2021. The graph shows a similar trend to the Albums section, with a steady increase in artist plays from 2014 to 2021, followed by a decline in 2022 and 2023. The latest year (2024) saw 1,071 artists played, representing a 26.39% decrease compared to the previous year (1,455 artists). The top 5 artists by total artist count are:

1. The Beatles - 13.6K
2. The Killers - 6.9K
3. John Mayer - 4.9K
4. Bob Dylan - 3.8K
5. Paul McCartney - 2.7K

The dominance of The Beatles in the top 5 artists list is consistent with their presence in the top 5 albums list, reinforcing the user's fondness for the band.

Tracks :

The Tracks section reveals that the user has played a total of 13,665 tracks over time, with a peak of 5,106 tracks played in 2020. The graph shows a steady increase in track plays from 2014 to 2020, followed by a decline in 2021 and 2022. The latest year (2024) saw 3,568 tracks played, representing an 11.49% decrease compared to the previous year (4,031 tracks). The top 5 tracks by total album count are:

1. Ode To The Mets - 207
2. In the Blood - 181
3. Dying Breed - 166
4. Caution - 164
5. 19 Dias y 500 N... - 148

These results suggest that the user enjoys a diverse range of tracks, with no single artist or genre dominating the top 5.

Additional Insights :

The dashboard also provides additional insights into the user's listening habits, including:

- The user's listening habits are more active on weekdays than weekends, as indicated by the pie charts in each section.
- The user's listening habits have decreased in recent years, as shown by the decline in album, artist, and track plays from 2021 to 2024.

Conclusion :

In conclusion, the Spotify dashboard provides a detailed analysis of the user's listening habits, revealing a strong affinity for The Beatles and a diverse range of tracks. The data suggests that the user's listening habits are more active on weekdays and have decreased in recent years. These insights can be useful for understanding the user's music preferences and behavior, and can inform recommendations for future music discovery. Overall, the dashboard offers a valuable snapshot of the user's listening habits, providing a rich source of data for analysis and interpretation.

CHAPTER 10

FUTURE SCOPE

The future scope for this Spotify dashboard analysis is vast and exciting. Here are some potential areas to explore:

- **Personalized Recommendations**
 - **Enhanced Discovery:** Use the insights from the dashboard to create hyper-personalized playlists and recommendations, catering to the user's affinity for The Beatles and diverse track preferences.
 - **Mood-Based Playlists:** Develop playlists that adapt to the user's mood, using data from their listening habits, such as weekday vs. weekend preferences.
- **Artist and Label Insights**
 - **Targeted Marketing:** Artists and labels can leverage the dashboard insights to target their marketing efforts, focusing on fans of similar artists and genres.
 - **Localized Content:** Curate localized playlists and content to cater to regional preferences, such as promoting K-Pop in Asia.
- **Spotify Platform Development**
 - **AI-Driven Audio Experiences:** Integrate AI-powered features that provide dynamic, real-time audio experiences based on the user's location, activity, and mood.
 - **New Monetization Options:** Explore innovative revenue models, such as micro-payments and fan-funded exclusives, to empower creators and artists.

- **Data Science and Analytics**

- **Advanced Data Visualization:** Develop more sophisticated data visualization tools to help users better understand their listening habits and preferences.
- **Predictive Modeling:** Use machine learning algorithms to predict user behavior and preferences, enabling proactive recommendations and playlist curation.

- **Business Growth and Expansion**

- **Strategic Partnerships:** Foster partnerships with telecommunication companies and other stakeholders to expand Spotify's reach and affordability in emerging markets.
- **Exclusive Content:** Offer exclusive content and experiences to premium subscribers, driving conversion and retention.