



LOVELY
PROFESSIONAL
UNIVERSITY

MOOC COURSE CERTIFICATION & PROJECT

**Classic Cars Data Visualization
& Analysis (Python)**

Developed By

VYSHNAV.P | 12205220 | D2213 |MCA

ACKNOWLEDGMENT

The completion of the MOOC course "Google's Get Started with Python" on Coursera, with an accomplishment date of August 28th, marks a significant milestone in my journey of learning and professional development. I consider myself fortunate to have had the opportunity to participate in this course, which has enriched my knowledge and skills in Python programming.

I extend my heartfelt gratitude to the Training and Placement Coordinator at the School of Computer Application, Lovely Professional University, for enabling me to access this valuable learning opportunity. Their guidance and advice played a pivotal role in making my experience with this MOOC course a rewarding and educational one.

VYSHNAV P
12205220

MOOC Certificate

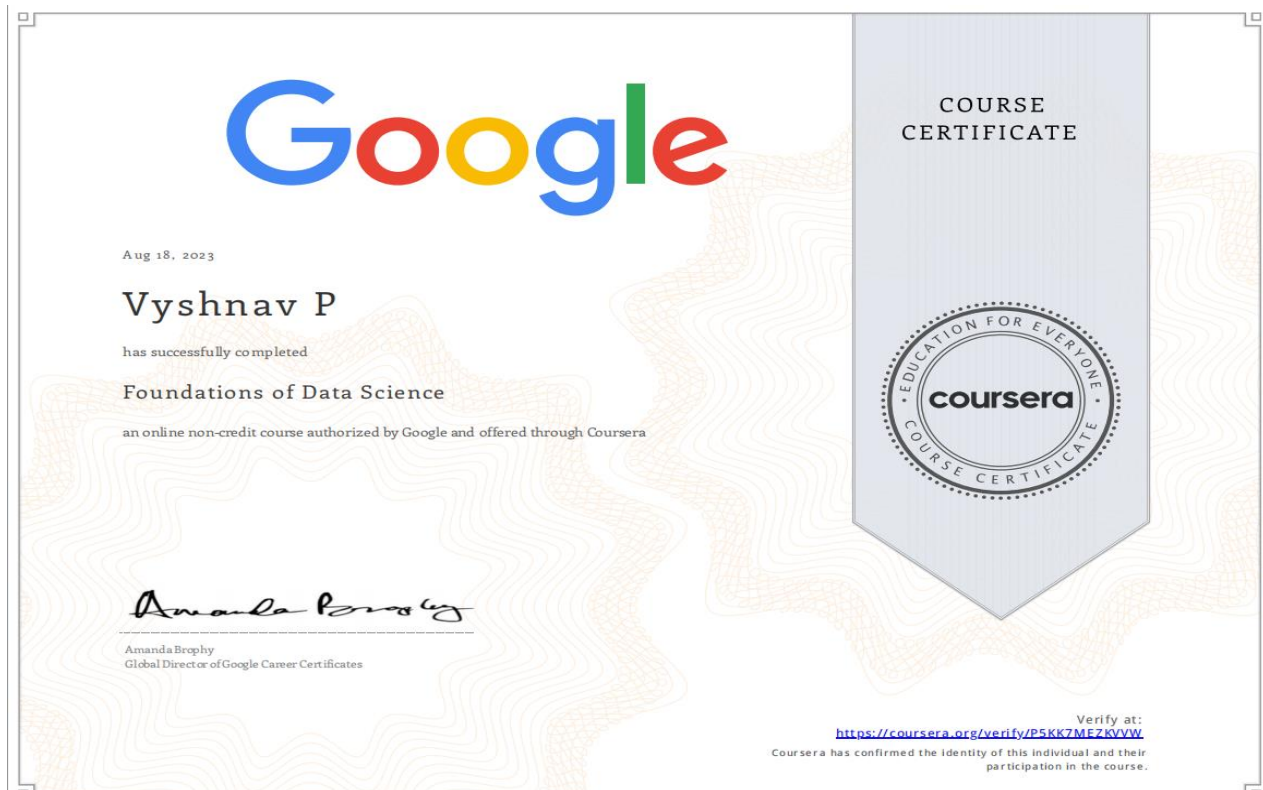


Table of contents

SI.NO	DESCRIPTION	PAGE NO
1	INTRODUCTION OF THE COURSE	5
2	OVERVIEW	6
3	TECHNICAL LEARNINGS FROM THE COURSE	9
3	INTRODUCTION OF MINI PROJECT	13
3.1	USE CASE	14
3.2	DELETED OR DEFERRED REQUIREMENTS	15
4	SNAPSHOTS	16
5	CODE SNIPPETS WITH DETAILS	16
5.1	HISTOGRAM	17
5.2	SCATTER PLOT	18
5.3	CUMILATIVE WEIGHT PLOT	20
5.4	VIOLIN PLOT	22
5.5	HEXBIN PLOT	23
5	GRADE SHEET OF ASSIGNMENTS/ MARKS CARD FROM THE MOOC	24
6	BIBLIOGRAPHY OR REFERENCES	25

INTRODUCTION TO COURSE

The "Hello, Python!" course is the second module in the Google Advanced Data Analytics Certificate program, designed to equip learners with essential Python programming skills for data analysis and data science. This course is part of a comprehensive program aimed at preparing individuals for careers in data science and advanced data analytics. Before diving into the technical learnings from this course, let's provide an overview of its structure and objectives.

Course Structure:

1. Module 1: Hello, Python!

- Duration: 4 hours
- Introduction to Python programming basics
- Exploration of Jupiter Notebooks
- Understanding variables, data types, and coding fundamentals
- Practical coding exercises

2. Module 2: Functions and Conditional Statements

- Duration: 3 hours
- Learning to call functions for data manipulation
- Writing conditional statements for decision-making
- Emphasizing clean and reusable code

3. Module 3: Loops and Strings

- Duration: 5 hours
- Mastery of iterative statements (loops) for automation
- Manipulating strings using slicing, indexing, and formatting

4. Module 4: Data Structures in Python

- Duration: 7 hours
- Exploration of fundamental data structures: lists, tuples, dictionaries, sets, and arrays
- Introduction to critical Python libraries for data analysis: NumPy and pandas

5. Module 5: Course 2 End-of-Course Project

- Duration: 6 hours
- Application of Python skills to solve a real-world business problem
- Selection of a project from a list of options
- Opportunity to build a professional portfolio for showcasing to potential employers

Python Programming Language: An Overview

Introduction

Python is a high-level programming language known for its simplicity, readability, and versatility. It was created by Guido van Rossum in December 1989 and has since evolved into a powerful tool with a wide range of applications across multiple industries. This report provides an overview of Python's history, versions, features, and common use cases.

History of Python

Python's journey began in 1989 when Guido van Rossum initiated its development at CWI in the Netherlands. It was designed as a successor to the ABC language, with a focus on exceptional handling and compatibility with the Amoeba operating system. Guido van Rossum's influential role in shaping Python earned him the title "Benevolent Dictator for Life" (BDFL) within the Python community, although he stepped down as the leader in 2018. The name "Python" was inspired by the British TV show "Monty Python's Flying Circus."

Python Versions

Python has undergone several major releases:

- 1. Python 1.0 (January 26, 1994):** The initial release of Python.
- 2. Python 2.0 (October 16, 2000):** Introduced list comprehensions, garbage collection, and Unicode support.
- 3. Python 3.0 (December 3, 2008):** A significant overhaul to address design flaws and inconsistencies, not backward compatible with Python 2.x.
- 4. Subsequent Python 3.x Releases:** Following Python 3.0, Python 3 continued to evolve with new features and improvements.
- 5. Python 2.7 (July 3, 2010):** The last release in the Python 2 series, no longer maintained after January 1, 2020.
- 6. Python 3.8, 3.9, 3.10, etc.:** Subsequent releases of Python 3, each bringing new features, optimizations, and bug fixes.

Features of Python

Python is acclaimed for its key features:

1. **Readability:** Python's syntax emphasizes clear code expression, making it accessible to both beginners and experts.
2. **Simplicity:** Python's minimalistic design and straightforward syntax promote concise and clean code.
3. **Large Standard Library:** Python includes a comprehensive standard library, reducing the need for external dependencies.
4. **Dynamic Typing:** Python allows dynamic changes to variable data types.
5. **Interpreted:** Python is an interpreted language, eliminating the need for compilation before execution.
6. **Cross-platform:** Python is compatible with Windows, macOS, Linux, and more.
7. **Object-Oriented:** Python supports object-oriented programming, facilitating code structuring with classes and objects.
8. **Extensive Third-party Libraries:** Python boasts a vast ecosystem of third-party libraries and frameworks for various domains.

Common Uses of Python

Python's versatility leads to its widespread use in various domains:

1. **Web Development:** Frameworks like Django and Flask are popular for web applications and server-side scripting.
2. **Data Analysis and Visualization:** Libraries like Pandas, NumPy, and Matplotlib aid data manipulation and visualization.
3. **Machine Learning and AI:** Python is essential for tasks like image recognition, natural language processing, and recommendation systems.
4. **Scientific Computing:** Python supports simulations and data analysis in scientific fields.
5. **Automation and Scripting:** Ideal for automating tasks, data extraction, and system administration.
6. **Game Development:** Python, with libraries like Pygame, is suitable for game prototyping.
7. **Desktop GUI Applications:** Libraries like Tkinter, PyQt, or Kivy enable desktop application development.
8. **Web Scraping:** Python libraries like BeautifulSoup and Scrapy are widely used for web data extraction.
9. **IoT (Internet of Things):** Python is used in IoT projects for device control and monitoring.
10. **Networking:** Python supports network programming, socket programming, and network tool development.
11. **Financial and Economic Modeling:** Python is utilized in finance for risk analysis and algorithmic trading.
12. **Databases:** Python interfaces with various databases using libraries like SQL Alchemy and Django ORM.
13. **Education:** Python's simplicity makes it a popular choice for teaching programming.
14. **Natural Language Processing (NLP):** Python is extensively used in NLP tasks such as sentiment analysis and chatbot development.
15. **Security:** Used in penetration testing, ethical hacking, and building security tools.
16. **Image and Video Processing:** Libraries like OpenCV make Python a choice for image and video analysis.

TECHNICAL LEARNINGS FROM THE COURSE:

I have gain valuable technical skills and knowledge essential for data analysis and data science:

Understanding Python: You will define what a programming language is and why Python is a preferred choice for data scientists.

Python Scripting: You'll learn how to create Python scripts to display data and perform various data operations.

Control Flow: The course covers how to control the flow of programs using conditions and functions, allowing you to make decisions and automate tasks.

Loops and Iteration: You will be equipped with the ability to utilize different types of loops when performing repeated operations, streamlining your coding process.

Data Types: You'll identify various data types, including integers, floats, strings, and Booleans, which are fundamental for data manipulation.

Data Structures: The course explores essential data structures such as lists, tuples, dictionaries, and sets, which are crucial for organizing and managing data efficiently.

Python Libraries: You'll learn how to import and use Python libraries like NumPy and pandas, which are essential tools for advanced data analysis and manipulation.

Exploratory Data Analysis (EDA) using Python.

Introduction to EDA

Exploratory Data Analysis (EDA) is a crucial preliminary step in data analysis. It involves examining and summarizing a dataset to gain insights, identify patterns, and prepare for more advanced tasks like machine learning or statistical modeling. Python, with its rich ecosystem of libraries, is an excellent choice for performing EDA efficiently.

Python Libraries for EDA

1. Pandas

Pandas is a powerful Python library for data manipulation and analysis. It offers data structures like Data Frames and Series, which are essential for storing and manipulating data.

```
import pandas as pd
data = pd.read_csv('your_dataset.csv')
```

Use `data.head()` to view the first few rows of your dataset and `data.info()` to get an overview of column data types and missing values. `data.describe()` provides summary statistics for numeric columns.

```
print(data.head())
print(data.info())
print(data.describe())
```

2. Matplotlib and Seaborn

Matplotlib is a versatile library for creating various types of visualizations in Python. Seaborn, built on top of Matplotlib, simplifies the creation of informative and attractive statistical graphics.

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

Histogram

```
sns.histplot(data['column_name'], kde=True)
plt.title('Distribution of column_name')
plt.show()
```

3. NumPy

NumPy is a fundamental library for numerical computations in Python. It provides support for arrays and matrices, which are essential for many data analysis tasks.

Steps in EDA using Python.

1. **Load Data:** Start by importing your dataset into a Pandas Data Frame from various sources like CSV files, Excel spreadsheets, databases, or APIs.
2. **Handling Missing Data:** Identify and handle missing data using methods like `data.isnull()`, `data.fillna()`, or `data.dropna()`.
3. **Data Visualization:** Create various plots and visualizations to understand the data better. Matplotlib and Seaborn are helpful for this.
4. **Feature Engineering:** Create new features or modify existing ones to extract more information from the data.
5. **Exploring Relationships:** Analyze relationships between variables using scatter plots, pair plots, and correlation matrices.
6. **Outlier Detection:** Identify and handle outliers using various statistical and visualization techniques.
7. **Summary and Insights:** Summarize your findings, draw conclusions, and communicate insights gained from the EDA process.
8. **Report Generation:** Create visual reports or presentations summarizing the EDA results using tools like Jupyter Notebooks or Tableau.

Data Types in Python

- Tuple: An ordered collection of items enclosed within parentheses (). It is immutable.
- List: An ordered collection of items enclosed within square brackets []. It allows duplicates.
- String (str): A sequence of characters enclosed within single (') or double (") quotes. Used to represent text data.
- Set: An unordered collection of unique elements enclosed within curly braces { }.
- Dictionary: An unordered collection of key-value pairs enclosed within curly braces { }.
- Integer (int): A whole number without a decimal point.
- Float (float): A number with a decimal point or in scientific notation, representing both whole and fractional numbers.

INTRODUCTION TO THE PROJECT

As part of my Coursera MOOC course, I'm excited to embark on a fascinating project involving Data Visualization and Analysis of a Classic Cars Dataset sourced from Kaggle. Classic cars are more than just vehicles; they are iconic symbols of automotive history and craftsmanship, each with a unique story to tell. In this project, I'll immerse myself in the world of classic cars to explore their characteristics, trends, and valuable insights.

OBJECTIVE:

My primary objective in this project is to conduct a thorough data visualization and analysis of the Classic Cars Dataset obtained from Kaggle. Using visualizations and statistical techniques, I aim to extract valuable information, patterns, and features related to classic cars, including their models, makes, production years, and more.

DATASET:

The core of this project is a dataset containing comprehensive information about classic cars. It encompasses key attributes such as the car's model, make, year of production, price, and condition. This dataset is a goldmine of data that will enable me to dive deep into the world of classic automobiles and gain a profound understanding of their characteristics.

DATA CLEANING AND PREPROCESSING:

Before I can dive into analysis, it's crucial to clean and preprocess the dataset. This step involves addressing missing values, removing duplicates, and ensuring data consistency and integrity.

DESCRIPTIVE STATISTICS:

I will compute basic statistics to gain insights into the central tendencies, spreads, and distributions of key variables. This includes summary statistics of car prices, production years, and other relevant attributes.

DATA VISUALIZATION:

Visualizations are potent tools for unveiling trends and patterns that may not be immediately evident in raw data. I'll create a variety of visualizations, such as histograms, scatter plots, and pie charts, to visually represent the data and identify interesting relationships between different attributes of classic cars.

DATASET USED FROM KAGGELE

HISTORICAL ANALYSIS:

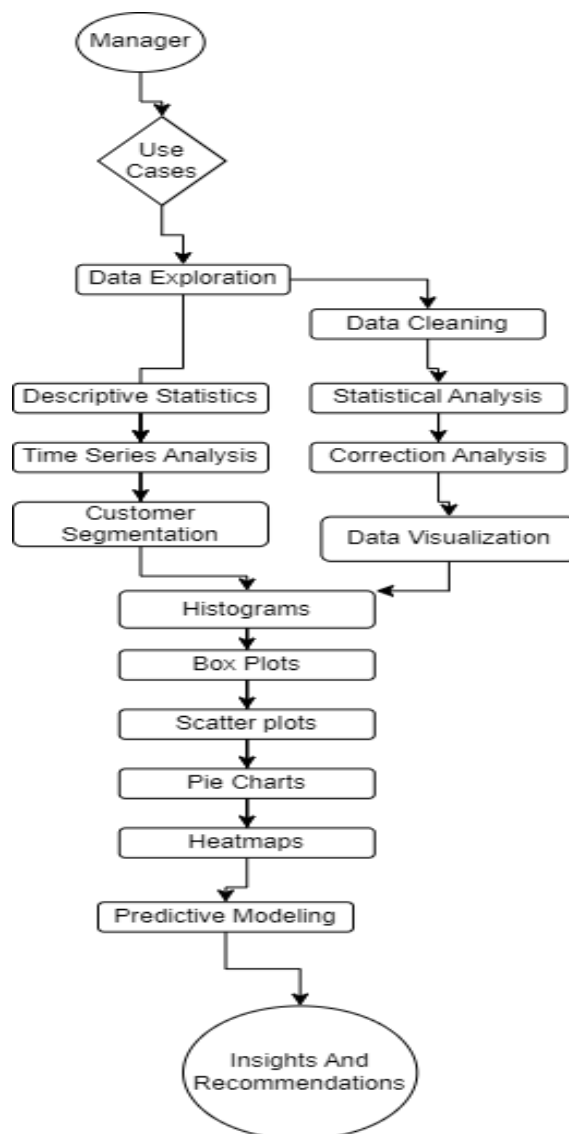
Classic cars often have historical significance. I will delve into the historical context by examining how the production and popularity of classic cars have evolved over time and identify any noteworthy trends in their desirability.

MARKET TRENDS:

I'll explore market trends related to classic cars, looking at factors that influence their value and demand. This will include analyzing the impact of make, model, and condition on classic car prices.

USER SCENARIOS|USE CASES

User scenarios or use cases are descriptions of how users interact with a system or application to achieve specific goals or tasks. In the context of the Classic Cars Data Visualization and Analysis project, here are some user scenarios or use cases:



Deleted or Deferred Requirements

Sometimes, during the development of a project, certain requirements may need to be deleted or deferred due to various reasons, such as changes in project scope, resource constraints, or shifting priorities. Here are examples of deleted or deferred requirements for the Classic Cars Data Visualization and Analysis project:

DELETED REQUIREMENTS:

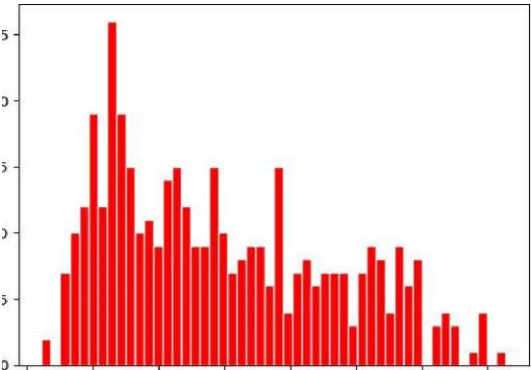
1. **VIRTUAL REALITY INTEGRATION:** Initially, there was a requirement to integrate virtual reality (VR) technology for an immersive classic car exploration experience. However, due to technical complexities and resource limitations, this requirement has been deleted.
2. **ADVANCED NATURAL LANGUAGE PROCESSING (NLP):** Originally, there was a plan to implement advanced NLP for sentiment analysis of user comments and reviews related to classic cars. This requirement has been deleted to streamline the project and prioritize core functionality.
3. **MOBILE APPLICATION:** Initially, there was a requirement to develop a dedicated mobile application for iOS and Android platforms. However, due to budget constraints, the mobile app requirement has been deleted, and the focus is on a responsive web interface.

DEFERRED REQUIREMENTS:

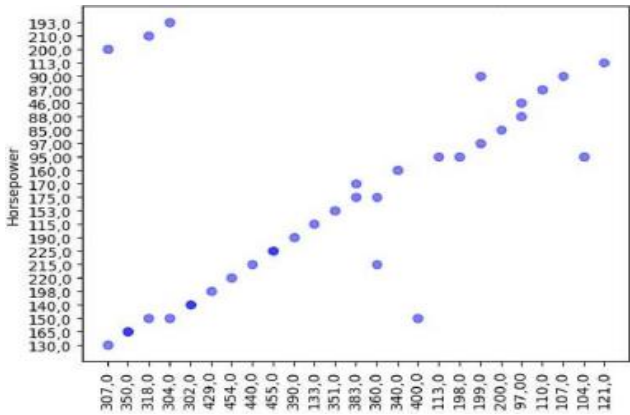
1. **MACHINE LEARNING RECOMMENDATIONS:** The requirement to implement machine learning-driven personalized recommendations for classic car enthusiasts has been deferred to a later phase of the project. The initial release will focus on basic data analysis and visualization.
2. **INTEGRATION WITH SOCIAL MEDIA:** The integration with social media platforms to allow users to share their favorite classic cars and insights will be deferred to a future update, as it requires additional development effort.
3. **MULTI-LANGUAGE SUPPORT:** While the project initially aimed for multi-language support, it has been deferred to a later stage to prioritize core functionalities. The primary language will be English in the initial release.
4. **ADVANCED USER ANALYTICS:** The implementation of advanced user analytics, including user behavior tracking and advanced reporting, has been deferred to a post-launch phase to ensure timely project delivery.

SNAPSHOTS

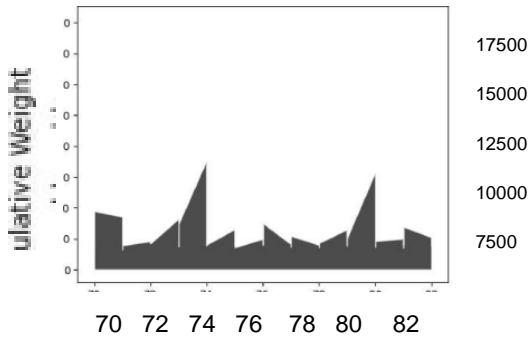
DISTRIBUTION OF CAR WEIGHTS



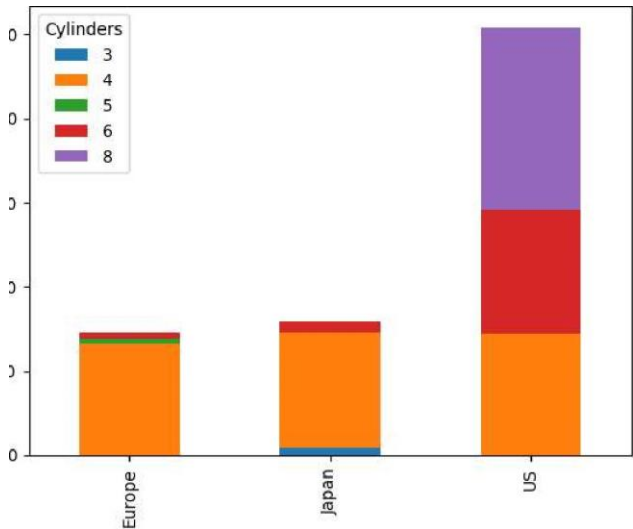
SCATTER PLOT: DISPLACEMENT VS HORSEPOWER



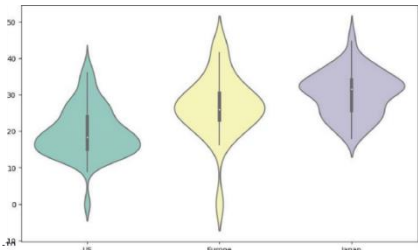
CUMULATIVE WEIGHT OF CARS OVER THE YEARS



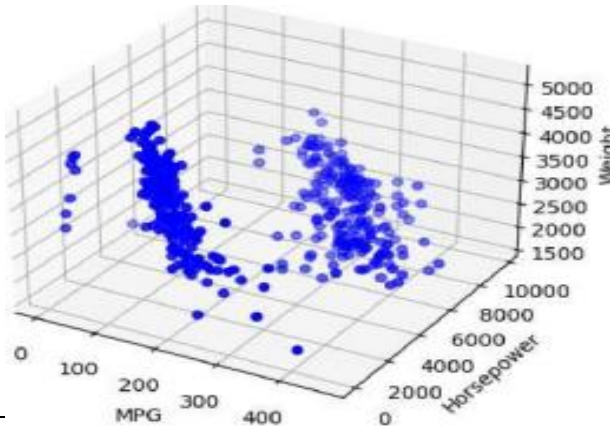
STACKED BAR CHART: CYLINDERS BY CAR ORIGIN



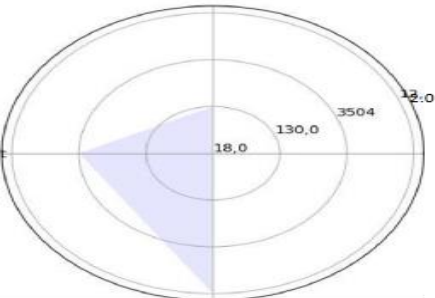
VIOLIN PLOT: MPG BY CAR ORIGIN



3D SCATTER PLOT: MPG VS. HORSEPOWER VS. WEIGHT



RADAR CHART: COMPARISON OF CAR ATTRIBUTES



HORSEPOWER & ACCELERATION

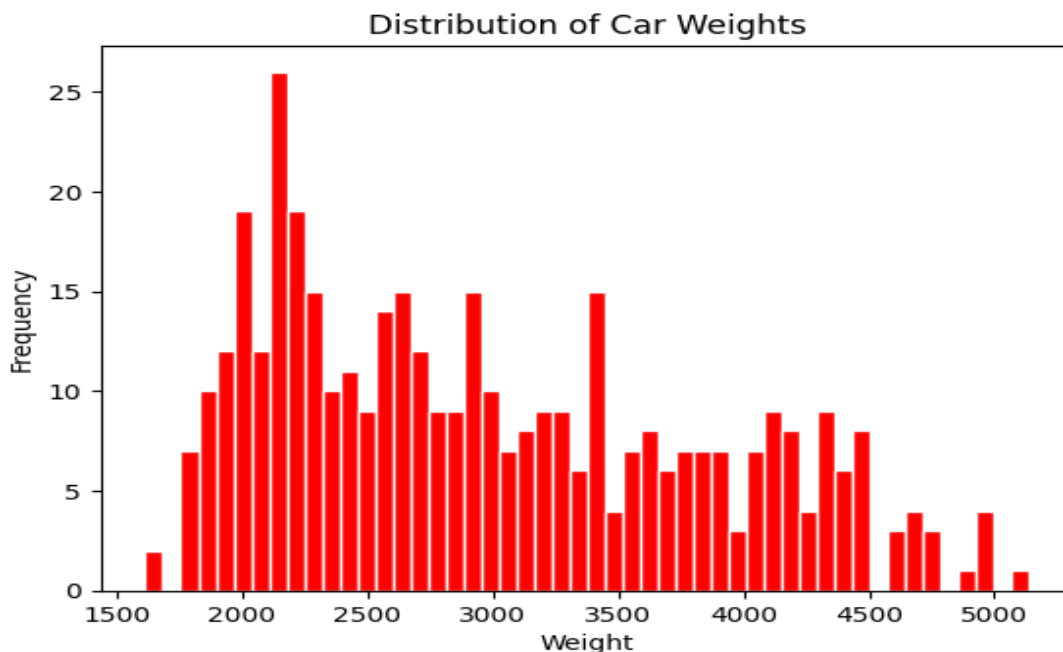
CODE SNIPPETS WITH DETAILS

HISTOGRAM

A histogram is a graphical representation of the distribution of data, typically used to display the frequency or density of values in a dataset. When discussing the distribution of car weights in the form of a histogram, you are essentially visualizing the different weight categories or ranges of cars and the frequency (or count) of cars falling into each category.

Histogram for car weights:

1. **Data Collection:** You start by collecting data on car weights. This data can be in the form of a list of car weights in pounds or kilograms, depending on your preference.
2. **Data Binning:** To create a histogram, you need to group the data into "bins" or intervals. For car weights, you might choose bins like 2000-2500 pounds, 2501-3000 pounds, 3001-3500 pounds, and so on. The choice of bin sizes can vary based on the level of detail you want to represent.
3. **Count Data in Each Bin:** You count how many cars fall into each weight bin. For example, you may find that 50 cars weigh between 2000 and 2500 pounds, 75 cars weigh between 2501 and 3000 pounds, and so on.
4. **Create the Histogram:** You create a bar chart where each bar represents a weight bin, and the height of the bar corresponds to the count of cars in that bin. The x-axis represents the weight bins, and the y-axis represents the frequency or count. You can also use density instead of count on the y-axis to show the relative proportion of cars in each bin.
5. **Visualize:** With these steps, you'll have a histogram that visually represents the distribution of car weights. If the data is normally distributed, the histogram might resemble a bell-shaped curve.



Classic Cars Data Visualization & Analysis

The x-axis represents car weight bins, the y-axis represents the count of cars in each weight range, and the bars show how many cars fall into each weight category.

CODE

```
import pandas as pd
import matplotlib.pyplot as plt

# Create a DataFrame
df = pd.read_csv("C:/Users/007vy/Desktop/Cars.csv", sep=";")

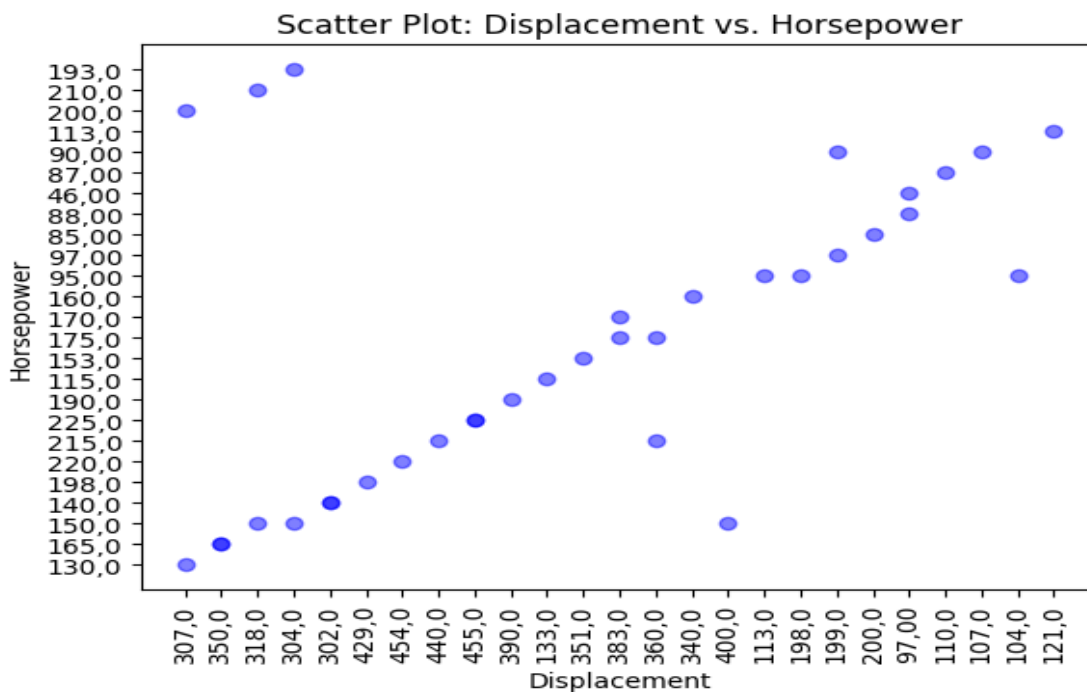
# Create a histogram of car weights
plt.hist(df['Weight'], bins=50, edgecolor='white', color="red")
plt.xlabel('Weight')
plt.ylabel('Frequency')
plt.title('Distribution of Car Weights')
plt.show()
```

SCATTER PLOT

A scatter plot is a graphical representation that displays individual data points on a two-dimensional graph, with one variable on the x-axis and another variable on the y-axis. In this case, you want to create a scatter plot to explore the relationship between "displacement" and "horsepower" for a set of cars.

scatter plot for "Displacement vs. Horsepower":

1. Data Collection: Gather a dataset that contains information on car models, including their engine displacement (usually measured in cubic inches or liters) and the horsepower of their engines.
2. Data Preparation: Ensure that the data is clean and properly formatted. Check for missing or outlier values and handle them if necessary.
3. Plot Creation: Use software or tools like Excel, Python (with libraries like Matplotlib or Seaborn), or other data visualization software to create the scatter plot.
4. X and Y Variables: Place "Displacement" on the x-axis and "Horsepower" on the y-axis. Displacement values will go on the horizontal axis, while Horsepower values will go on the vertical axis.
5. Plotting Data Points: For each car in your data set, plot a point on the graph. The x-coordinate represents the car's engine displacement, and the y-coordinate represents its horsepower. Each point represents a single car.
6. Labeling: Label the axes, provide a title for the plot, and include a legend or other explanatory information if needed.



Each point on the scatter plot represents a car, and the position of the point corresponds to its engine displacement and horsepower. You can visually inspect the plot to identify trends or patterns, such as whether there is a positive or negative correlation between engine displacement and horsepower.

A positive correlation would imply that as engine displacement increases, so does horsepower, while a negative correlation would indicate the opposite. A lack of correlation would mean that there is no clear relationship between the two variables.

Scatter plots are useful for visualizing relationships between two continuous variables and can help you understand the data distribution, identify outliers, and make informed decisions, such as in the context of automotive design or performance analysis.

CODE

```
import pandas as pd
import matplotlib.pyplot as plt

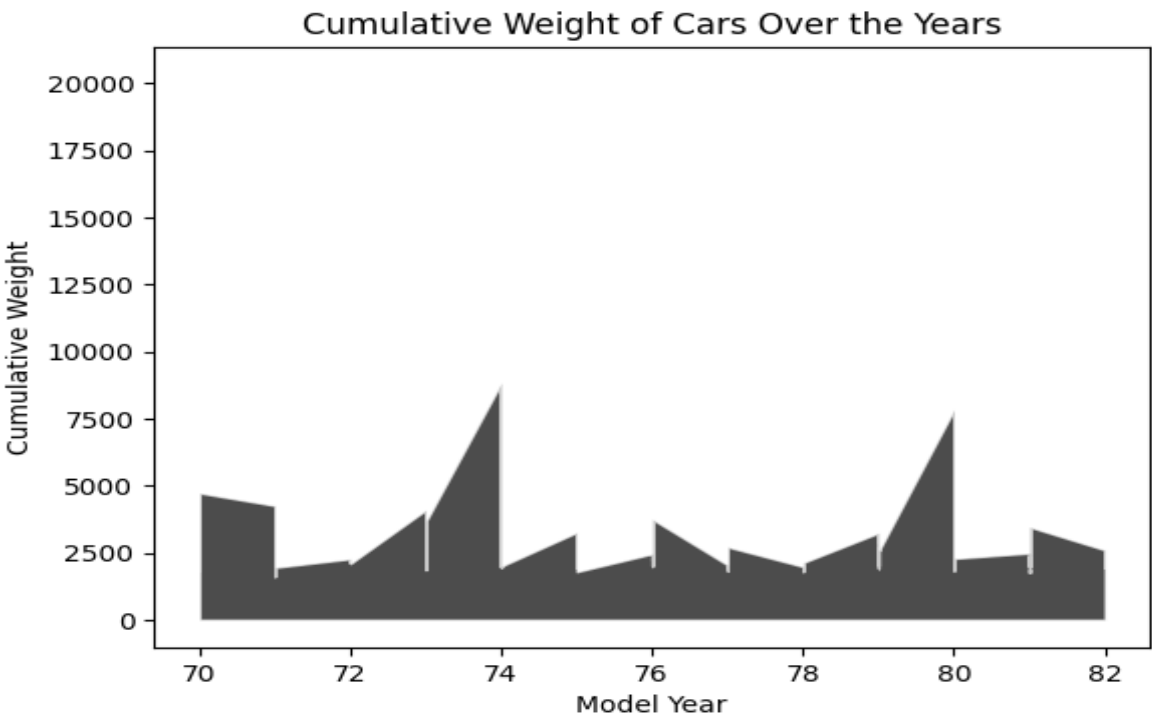
# Create a DataFrame
df = pd.read_csv("C:/Users/007vy/Desktop/Cars.csv", sep=";")
df = df.head(35)

# Create a scatter plot of Displacement vs. Horsepower
plt.scatter(df['Displacement'], df['Horsepower'], alpha=0.5, color="blue")
plt.xlabel('Displacement')
plt.xticks(rotation=90)
plt.ylabel('Horsepower')
plt.title('Scatter Plot: Displacement vs. Horsepower')
plt.grid(False)
plt.show()
```

CUMULATIVE WEIGHT PLOT

A cumulative weight plot shows the total or cumulative weight of a particular variable (in this case, the weight of cars) over a series of years or time intervals. Creating a cumulative weight plot allows you to visualize how the total weight of cars has changed over time.

1. **Data Collection:** Collect data on the weights of cars for each year or time interval you want to analyze. Make sure the data is organized with a year or time period associated with each weight measurement.
2. **Data Preparation:** Ensure the data is clean, consistent, and properly formatted. If there are missing or outlier values, address them as needed.
3. **Cumulative Calculation:** Calculate the cumulative weight for each year. To do this, start with the weight for the earliest year and add the weight of each subsequent year to the running total.
4. **Plot Creation:** Use data visualization software or tools like Excel, Python (with libraries like Matplotlib or Seaborn), or other software to create the cumulative weight plot.
5. **X and Y Variables:** Place the years (or time intervals) on the x-axis and the cumulative weight on the y-axis. The x-axis represents time, while the y-axis represents the cumulative weight of cars.
6. **Plotting Data Points:** For each year or time interval, plot a point on the graph. The x-coordinate represents the year or time, and the y-coordinate represents the cumulative weight up to that point in time.
7. **Labeling:** Label the axes, provide a title for the plot, and include any relevant information, such as units of measurement for weight and time.



Classic Cars Data Visualization & Analysis

Each point on the plot represents a year, and the height of the point on the y-axis shows the cumulative weight of cars up to that year. The plot illustrates how the cumulative weight of cars has changed over a specific time span.

A cumulative weight plot can be useful for tracking trends and changes in car weights over the years, which can be valuable for various purposes, including understanding the evolution of automotive engineering and the impact of weight-related regulations or trends in the automotive industry.

CODE

```
import pandas as pd
import matplotlib.pyplot as plt

# Create a DataFrame
df = pd.read_csv("C:/Users/007vy/Desktop/Cars.csv", sep=";")

# Calculate the cumulative sum of weights over the years
cumulative_weights = df.groupby('Car')['Weight'].cumsum()

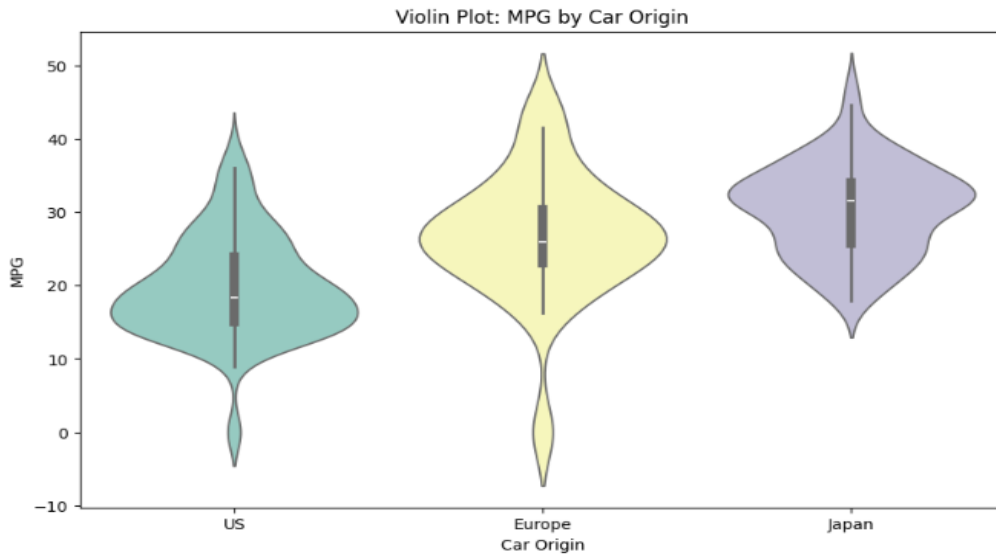
# Create an area plot
plt.fill_between(df['Model'], cumulative_weights, alpha=0.7,color="black",edgecolor="white")
plt.xlabel('Model Year')
plt.ylabel('Cumulative Weight')
plt.title('Cumulative Weight of Cars Over the Years')
plt.show()
```

VIOLIN PLOT

A violin plot is a data visualization that combines aspects of a box plot and a kernel density plot to show the distribution of a numerical variable (in this case, miles per gallon, or MPG) across different categories or groups (in this case, car origins). Creating a violin plot for MPG by car origin allows you to visualize the distribution of MPG values for cars from different countries of origin.

1. **Data Collection:** Gather a dataset that includes information on MPG values for cars, as well as the country of origin for each car (e.g., USA, Japan, Europe).
2. **Data Preparation:** Ensure that the data is clean and properly formatted. Check for missing values and handle them if necessary.
3. **Plot Creation:** Use data visualization software or libraries like Seaborn in Python to create the violin plot.
4. **X and Y Variables:** Place the car origin on the x-axis and MPG values on the y-axis. Car origin categories will be on the horizontal axis, while MPG values will be on the vertical axis.
5. **Plotting Data:** For each car origin category (e.g., USA, Japan, Europe), a violin plot will be drawn that represents the distribution of MPG values within that category. The width of the violin plot at any given point indicates the density of MPG values. You'll also typically see a box plot inside the violin, which shows the median, quartiles, and potential outliers within the data.

6. Labeling and Styling: Label the axes, provide a title for the plot, and format the plot to make it clear and visually appealing. You can use different colors or styles for the violins to distinguish them.



Each has three violin plots, each representing the distribution of MPG values for cars from the USA, Japan, and Europe, respectively. The width of the violin at different points shows the density of MPG values, and the box plot inside each violin provides additional statistical information about the data.

A violin plot is a valuable tool for comparing the distribution of a numerical variable across different categories or groups and can help you understand how MPG values vary by the car's country of origin.

CODE

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Create a DataFrame
df = pd.read_csv("C:/Users/007vy/Desktop/Cars.csv", sep=";")

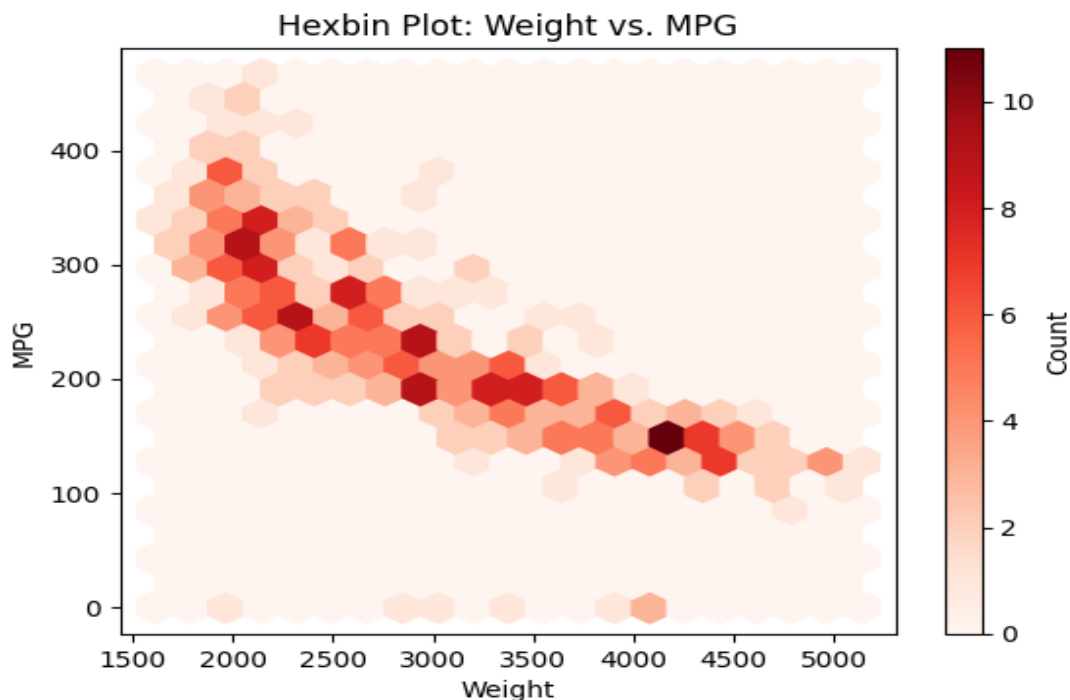
# Convert 'MPG' column to numeric (replacing commas with periods as well)
df['MPG'] = df['MPG'].str.replace(',', '.').astype(float)

# Create a violin plot of MPG by Origin
plt.figure(figsize=(10, 6))
sns.violinplot(x='Origin', y='MPG', data=df, palette='Set3')
plt.xlabel('Car Origin')
plt.ylabel('MPG')
plt.title('Violin Plot: MPG by Car Origin')
plt.show()
```

HEXBIN PLOT

A hexbin plot is a two-dimensional data visualization that's particularly useful for visualizing the distribution and relationship between two numerical variables, especially when dealing with a large dataset. In this case, you want to create a hexbin plot to explore the relationship between "Weight" and "MPG" (miles per gallon) for a set of cars. Hexbin plots divide the data space into hexagonal bins and color-code them based on the density of data points within each bin.

1. **Data Collection:** Collect a dataset that includes information on car weights (independent variable) and MPG values (dependent variable).
2. **Data Preparation:** Ensure that the data is clean and properly formatted, handling any missing or outlier values if necessary.
3. **Plot Creation:** Use data visualization software or libraries like Matplotlib or Seaborn in Python to create the hexbin plot.
4. **X and Y Variables:** Place "Weight" on the x-axis and "MPG" on the y-axis. Weight values will be on the horizontal axis, while MPG values will be on the vertical axis.
5. **Plotting Data Points:** Rather than plotting individual data points, a hexbin plot divides the data space into hexagonal bins. Each bin represents a portion of the data space. The color of each hexagon represents the density of data points within it. Denser regions are represented with darker colors, while sparser regions are lighter.
6. **Labeling and Styling:** Label the axes, provide a title for the plot, and choose appropriate color schemes to make the hexbin plot informative and visually appealing. You can also include a color scale that indicates the density of data points.



Classic Cars Data Visualization & Analysis

In this example, the hexagon plot shows the relationship between car weight and MPG. Denser regions, where there are more data points, are represented by darker hexagons, while sparser regions are lighter.

A hexbin plot is particularly useful when you have a large dataset with many data points, making it difficult to visualize individual points effectively. It allows you to see trends and patterns in the data, such as the concentration of cars with certain weights and MPG values, and how they are distributed across the two variables.




CODE







```
import pandas as pd
import matplotlib.pyplot as plt

# Create a DataFrame, specifying thousands to handle commas as thousands separators
df = pd.read_csv("C:/Users/007vy/Desktop/Cars.csv", sep=";", thousands=',')


# Create a hexbin plot for Weight vs. MPG
plt.hexbin(df['Weight'], df['MPG'], gridsize=20, cmap='Reds')
plt.xlabel('Weight')
plt.ylabel('MPG')
plt.title('Hexbin Plot: Weight vs. MPG')
plt.colorbar(label='Count')
plt.show()
```

GRADE SHEET OF ASSIGNMENTS | MARKS CARD FROM THE MOOC


coursera  **Explore**  


Online Degrees  Find your New Career  English    Vyshnav 

My Courses





Get Started with Figma
Coursera Project Network
Grade Achieved: 100%

Add to LinkedIn 





Get Started with Python
Google
Grade Achieved: 100%

Add to LinkedIn 





Foundations of Data Science
Google
Grade Achieved: 95.80%





Add to LinkedIn 




Classic Cars Data Visualization & Analysis



Search in course

English    Vyshnav 








Get Started with Python
Google


Course Material

- Week 1
- Week 2
- Week 3
- Week 4
- Week 5

Grades

Notes

Item	Status	Due	Weight	Grade
 Module 1 challenge Quiz	Passed	Aug 27 11:59 PM IST	19.92%	100%
 Module 2 challenge Quiz	Passed	Sep 3 11:59 PM IST	19.92%	100%
 Module 3 challenge Quiz	Passed	Sep 10 11:59 PM IST	20.04%	100%
 Module 4 challenge Quiz	Passed	Sep 17 11:59 PM IST	20.04%	100%
 Assess your Course 2 end-of-course project Quiz	Passed	Sep 24 11:59 PM IST	20.04%	100%



References

Wikipedia: Histogram

Wikipedia: Scatter plot

Wikipedia: Cumulative weight plot

Wikipedia: Violin plot

Wikipedia: Hexbin plot

FOR DATASET

<https://www.kaggle.com/code/antoniosabatini/classic-cars-data-visualization-and-analysis>