IBM – COURSERA DATASCIENCE CAPSTONE

SRIDHARAN CHANDRAN

18.08.2023

TOOLS FOR DATA SCIENCE

Data Science Tools and Ecosystem

- Welcome to the presentation on Data Science Tools and Ecosystem.
- In today's rapidly evolving technological landscape, data has become a cornerstone of decision-making and innovation.
- Data Scientists play a pivotal role in extracting insights from vast datasets, enabling businesses to make informed choices.
- This presentation offers a concise overview of the essential tools and languages that empower Data Scientists in their work.
- As we delve into the intricacies of data science tools, we'll explore popular programming languages, essential libraries, and practical examples.
- Let's embark on a journey through the dynamic world of Data Science and its indispensable tools.

POPULAR PROGRAMMING LANGUAGES

List the popular programming languages used by Data Scientists:

- Python
- SQL
- R programming language
- Julia
- JavaScript
- Scala
- Java

Data Science Tools and Ecosystem

- **Popular Languages for Data Scientists:**
  - **Python:**
    - General-purpose programming language for software development.
  - **SQL (Structured Query Language):**
    - Widely used for database manipulation and querying.
  - **R Programming Language:**
    - Statistical computing and graphics.
  - **Julia, JavaScript, Scala, Java:**
    - Diverse languages with specific applications in data science.

COMMONLY USED LIBRARIES

List the commonly used libraries by Data Scientists

- TensorFlow
- Numpy
- SciPy
- Pandas
- Matplotlib
- Keras
- SciKit-Learn
- PyTorch

Overview of Specific Tools

Sl. No.	Tools Name
1	Numpy
2	Pandas
3	Scikit-Learn
4	SciPy
5	Matplotlib

Examples of Arithmetic Evaluation in Python

- **Evaluate arithmetic expressions using eval() function**

```
# Initial expression
text = "4 * 6 - 3"

# Print original string
print("Original string is : " + text)

# Expression evaluation using eval()
result = eval(text)
print(result)
```

Output:

```
Original string is : 4 * 6 - 3
21
```

Converting Minutes to Hours

```
Total_minutes = 200
Minutes_per_Hour = 60
Total_Hours = Total_minutes / Minutes_per_Hour
print(Total_Hours)
```

Output:

```
3.3333333333333335
```

Objectives

- Understand the prominent languages utilized by Data Scientists.
- Recognize the significance of various programming languages in data science.
- Explore essential libraries for data analysis and manipulation.
- Gain insights into evaluating arithmetic expressions and unit conversions in Python.

DATA SCIENCE METHODOLOGY

Credit Card Analysis

- Welcome to today's presentation on the application of Data Science Methodology to Credit Card Analysis.
- In this session, we will explore how data science techniques can be leveraged to address a critical challenge in the banking industry: determining client suitability for credit card issuance.
- As the financial landscape continues to evolve, banks face the complex task of assessing the creditworthiness of potential clients.

Credit Card Analysis

- Accurate credit card suitability assessment is essential to mitigate risks, ensure responsible lending practices, and maintain healthy financial portfolios.
- By applying data science methodology, we can create predictive models that analyze a range of variables to make informed decisions about credit card approvals.
- Let's delve into the details of this process and understand how data-driven insights are transforming the credit card issuance process for both clients and financial institutions.

BUSINESS UNDERSTANDING

- **Role Play**: In our exploration of credit card analysis, let's step into the shoes of both the client and the data scientist. We'll envision the challenges faced by banks and how data science can provide solutions.
- **The Problem**: The core issue we address is how banks can effectively determine if a client is suitable for obtaining a credit card based on their economic strength.
- **Question Framing**: Can we leverage data-driven insights to automatically assess whether a client is suitable to obtain a credit card?

ANALYTIC APPROACH

- **Classification Model**: To tackle the problem at hand, we will adopt a classification model, which aims to provide a binary "Yes" or "No" answer.

DATA REQUIREMENTS

- **Personal Data**: The success of our classification model hinges on the availability of comprehensive personal data of bank clients. This data should encompass both clients who defaulted on payments and those who did not.

DATA COLLECTION

- **Descriptive Statistics:** We will employ descriptive statistics to summarize and analyze the gathered data, ensuring it meets the prerequisites for our model.

- **Data Evaluation:** Rigorous data evaluation techniques will be applied to ensure that the data collected is relevant and provides valuable insights.

DATA UNDERSTANDING AND PREPARATION

- **Data Understanding**: In this phase, we will dive deep into our data, evaluating variables and calculating essential univariate statistics. We'll also investigate correlations between variables.

- **Data Preparation**: To prepare our data for the classification model, we'll transform and preprocess it as required. This step is crucial to ensure that the model receives the necessary input.

MODELING AND EVALUATION

- **Modeling Phase**: Here, we'll construct a classification model using the meticulously prepared data. This model will be trained to make accurate predictions regarding credit card suitability.

- **Evaluation Process**: The model's performance will be rigorously evaluated, and iterative improvements will be made to enhance accuracy and predictive power.

CONCLUSION

- **Methodology Recap**: To sum up, we have explored the various stages of the Data Science Methodology applied to credit card analysis, from problem framing to model evaluation.

- **Critical Importance**: Accurate credit card suitability assessment is a critical aspect of responsible lending, risk management, and ensuring positive financial outcomes for both clients and institutions.

- **Thank You**: Thank you for your attention. By embracing data science, we can drive transformation in the banking sector and ensure that credit card issuance aligns with clients' economic capabilities.

PYTHON PROJECT FOR DATA SCIENCE

INTRODUCTION

Introduction to stock data extraction & visualization

- Extracting and visualizing stock data is essential for informed decision-making in data science projects.
- In this presentation, we'll explore a project that involves extracting stock and revenue data and creating visualizations.
- By the end, you'll have a clear understanding of the process and its significance.

PROJECT OVERVIEW

- Objective: Extract stock and revenue data, create informative visualizations.
- Tools Used: yfinance, pandas, requests, BeautifulSoup, plotly.
- Tasks:
  - Stock data extraction for Tesla and GameStop.
  - Web scraping for revenue data.
  - Visualizing stock and revenue data.

Overview of Tasks

- Define a Function that Makes a Graph
- Use yfinance to Extract Stock Data
- Use Webscraping to Extract Tesla Revenue Data
- Use yfinance to Extract Stock Data
- Use Webscraping to Extract GME Revenue Data
- Plot Tesla Stock Graph
- Plot GameStop Stock Graph

Purpose and Goals:

- In this assignment, our main objective is to extract essential stock data and leverage visualization techniques to empower informed decision-making. By harnessing the power of data analysis and visualization, we aim to uncover meaningful insights that drive effective choices in the realm of stock investments.

Importance of Data Analysis and Visualization:

- Data analysis and visualization are pivotal pillars of data science. They allow us to transform raw data into actionable insights, providing a clear understanding of complex trends, patterns, and relationships. Through insightful visualizations, we not only communicate information effectively but also unlock the potential for deeper comprehension, enabling us to navigate the intricate landscape of stock data with confidence.

Key Takeaways:

- Understanding the significance of extracting and visualizing stock data.Realizing the power of data analysis and visualization in driving informed decisions.Exploring the intersection of data science and financial insights for strategic stock management.

TABLE OF CONTENTS

- Define Graphing Function
- Question 1: Tesla Stock Data Extraction
- Question 2: Tesla Revenue Data Extraction
- Question 3: GameStop Stock Data Extraction
- Question 4: GameStop Revenue Data Extraction
- Question 5: Plot Tesla Stock Graph
- Question 6: Plot GameStop Stock Graph
- Conclusion

Extracting and Visualizing Stock Data

Description:

- Extracting essential data from a dataset and displaying it is a fundamental aspect of data science.
- Enables informed decision-making based on data insights.

Subtasks:

- **Use yfinance to Extract Stock Data:**
  - Utilize the yfinance library to extract historical stock data.
  - Retrieve and process stock data for Tesla (TSLA) and GameStop (GME) stocks.
- **Use Webscraping to Extract Revenue Data:**
  - Apply web scraping techniques to gather revenue data for Tesla and GameStop.
  - Enhance data analysis by incorporating revenue insights.
- **Plot Historical Stock Graphs:**
  - Create interactive visualizations of historical share prices and revenues.
  - Utilize Plotly and subplots to present both data sets in a single graph.

Data Visualization and Interpretation

- **Description:**
  - Visualize extracted data to identify patterns, trends, and relationships.
  - Interpretation of data aids in drawing meaningful conclusions.
- **Subtasks:**
  - **Plot Tesla Stock Graph:**
    - Generate a graph depicting Tesla's historical stock performance.
    - Compare share prices and revenues over time.
  - **Plot GameStop Stock Graph:**
    - Visualize GameStop's historical stock trends and revenue growth.
    - Analyze patterns in share prices and revenue data.

- **Estimated Time Needed: 30 min**
  - Presentation of task details and concepts.
  - Code execution and graph visualization.
  - Q&A session for clarifications and discussion.

Defining the Graphing Function

```
def make_graph(stock_data, revenue_data, stock):
    fig = make_subplots(rows=2, cols=1, shared_xaxes=True, subplot_titles=("Historical Share Price", "Historical Revenue"), vertical_spacing = .3)
    fig.add_trace(go.Scatter(x=pd.to_datetime(stock_data.Date, infer_datetime_format=True), y=stock_data.Close.astype("float"), name="Share Price"), row=1, col=1)
    fig.add_trace(go.Scatter(x=pd.to_datetime(revenue_data.Date, infer_datetime_format=True), y=revenue_data.Revenue.astype("float"), name="Revenue"), row=2, col=1)
    fig.update_xaxes(title_text="Date", row=1, col=1)
    fig.update_xaxes(title_text="Date", row=2, col=1)
    fig.update_yaxes(title_text="Price ($US)", row=1, col=1)
    fig.update_yaxes(title_text="Revenue ($US Millions)", row=2, col=1)
    fig.update_layout(showlegend=False,
    height=900,
    title=stock,
    xaxis_rangeslider_visible=True)
    fig.show()
```

- Introduction to the make_graph function used for creating stock and revenue graphs.
- The function takes dataframes with stock and revenue data along with the stock name.

Tesla Stock Data Extraction

Using the `Ticker` function enter the ticker symbol of the stock we want to extract data on to create a ticker object. The stock is Tesla and its ticker symbol is `TSLA`.

n [4]:
```
tesla = yf.Ticker("TSLA")
```

Using the ticker object and the function `history` extract stock information and save it in a dataframe named `tesla_data`. Set the `period` parameter to `max` so we get information for the maximum amount of time.

n [5]:
```
tesla_data = tesla.history(period="max")
```

Reset the index using the `reset_index(inplace=True)` function on the tesla_data DataFrame and display the first five rows of the `tesla_data` dataframe using the `head` function. Take a screenshot of the results and code from the beginning of Question 1 to the results below.

n [6]:
```
tesla_data.reset_index(inplace=True)
tesla_data.head()
```

ut[6]:

	Date	Open	High	Low	Close	Volume	Dividends	Stock Splits
0	2010-06-29 00:00:00-04:00	1.266667	1.666667	1.169333	1.592667	281494500	0.0	0.0
1	2010-06-30 00:00:00-04:00	1.719333	2.028000	1.553333	1.588667	257806500	0.0	0.0
2	2010-07-01 00:00:00-04:00	1.666667	1.728000	1.351333	1.464000	123282000	0.0	0.0
3	2010-07-02 00:00:00-04:00	1.533333	1.540000	1.247333	1.280000	77097000	0.0	0.0
4	2010-07-06 00:00:00-04:00	1.333333	1.333333	1.055333	1.074000	103003500	0.0	0.0

Tesla Revenue Data Extraction

Question 2: Use Webscraping to Extract Tesla Revenue Data

Use the `requests` library to download the webpage https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-PY0220EN-SkillsNetwork/labs/project/revenue.htm Save the text of the response as a variable named `html_data`.

Parse the html data using `beautiful_soup`.

In [7]:
```
url = "https://www.macrotrends.net/stocks/charts/TSLA/tesla/revenue?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=6

headers = {
    "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110 Safari/5

html_data = requests.get(url, headers=headers).text
```

In [8]:
```
soup = BeautifulSoup(html_data, 'html5lib')
```

In [9]:
```
tesla_revenue = pd.read_html(html_data, match="Tesla Quarterly Revenue")[0]

# Rename columns
tesla_revenue.columns = ["Date", "Revenue"]
```

Using `BeautifulSoup` or the `read_html` function extract the table with `Tesla Revenue` and store it into a dataframe named `tesla_revenue`. The dataframe should have columns `Date` and `Revenue`.

Click here if you need help locating the table

Execute the following line to remove the comma and dollar sign from the `Revenue` column.

In [10]:
```
tesla_revenue
```

Out[10]:

	Date	Revenue
0	2023-06-30	$24,927
1	2023-03-31	$23,329
2	2022-12-31	$24,318
3	2022-09-30	$21,454

GameStop Stock Data Extraction

Question 3: Use yfinance to Extract Stock Data

Using the `Ticker` function enter the ticker symbol of the stock we want to extract data on to create a ticker object. The stock is GameStop and its ticker symbol is `GME`.

In [14]:
```
gme = yf.Ticker("GME")
```

Using the ticker object and the function `history` extract stock information and save it in a dataframe named `gme_data`. Set the `period` parameter to `max` so we get information for the maximum amount of time.

In [15]:
```
gme_data = gme.history(period="max")
```

Reset the index using the `reset_index(inplace=True)` function on the gme_data DataFrame and display the first five rows of the `gme_data` dataframe using the `head` function. Take a screenshot of the results and code from the beginning of Question 3 to the results below.

In [16]:
```
gme_data.reset_index(inplace=True)
gme_data.head()
```

Out[16]:

	Date	Open	High	Low	Close	Volume	Dividends	Stock Splits
0	2002-02-13 00:00:00-05:00	1.620129	1.693350	1.603296	1.691667	76216000	0.0	0.0
1	2002-02-14 00:00:00-05:00	1.712707	1.716074	1.670626	1.683250	11021600	0.0	0.0
2	2002-02-15 00:00:00-05:00	1.683251	1.687459	1.658002	1.674834	8389600	0.0	0.0
3	2002-02-19 00:00:00-05:00	1.666417	1.666417	1.578047	1.607504	7410400	0.0	0.0
4	2002-02-20 00:00:00-05:00	1.615921	1.662210	1.603296	1.662210	6892800	0.0	0.0

GameStop Revenue Data Extraction

Question 4: Use Webscraping to Extract GME Revenue Data

Use the `requests` library to download the webpage https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-PY0220EN-SkillsNetwork/labs/project/stock.html. Save the text of the response as a variable named `html_data`.

In [17]:
```
url = "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-PY0220EN-SkillsNetwork/labs/p
html_data  = requests.get(url).text
```

Parse the html data using `beautiful_soup`.

In [18]:
```
soup = BeautifulSoup(html_data, 'html5lib')
```

Using `BeautifulSoup` or the `read_html` function extract the table with `GameStop Revenue` and store it into a dataframe named `gme_revenue`. The dataframe should have columns `Date` and `Revenue`. Make sure the comma and dollar sign is removed from the `Revenue` column using a method similar to what you did in Question 2.

Click here if you need help locating the table

In [19]:
```
gme_revenue = pd.read_html(html_data, match="GameStop Quarterly Revenue")[0]
gme_revenue.columns=["Date", "Revenue"]
gme_revenue["Revenue"] = gme_revenue['Revenue'].str.replace(',|\$',"",regex=True)
gme_revenue.dropna(inplace=True)
gme_revenue = gme_revenue[gme_revenue['Revenue'] != ""]
```

Display the last five rows of the `gme_revenue` dataframe using the `tail` function. Take a screenshot of the results.

In [20]:
```
gme_revenue.tail()
```

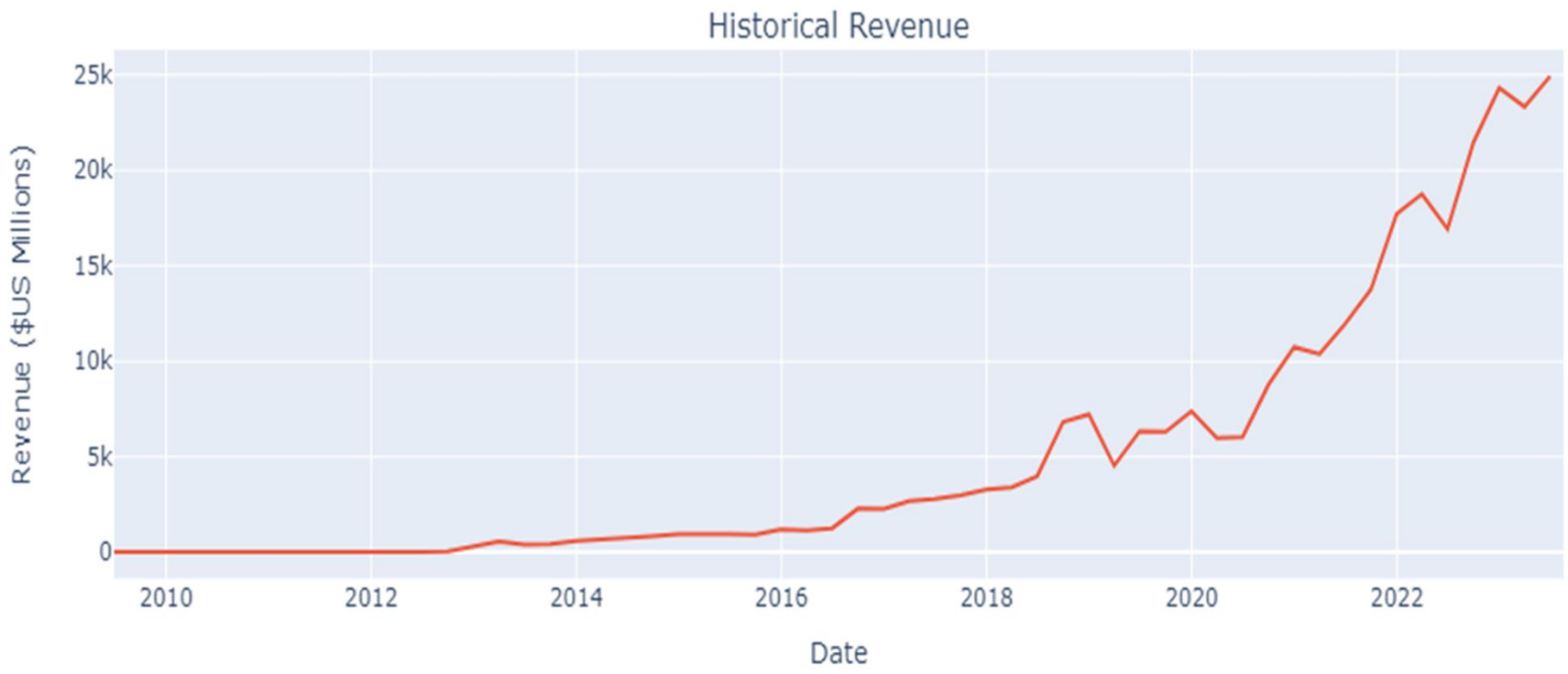
Out[20]:

	Date	Revenue
57	2006-01-31	1667
58	2005-10-31	534

Plot Tesla Stock Graph

```
In [21]: make_graph(tesla_data, tesla_revenue, "Tesla")
```

Plot Tesla Stock Graph

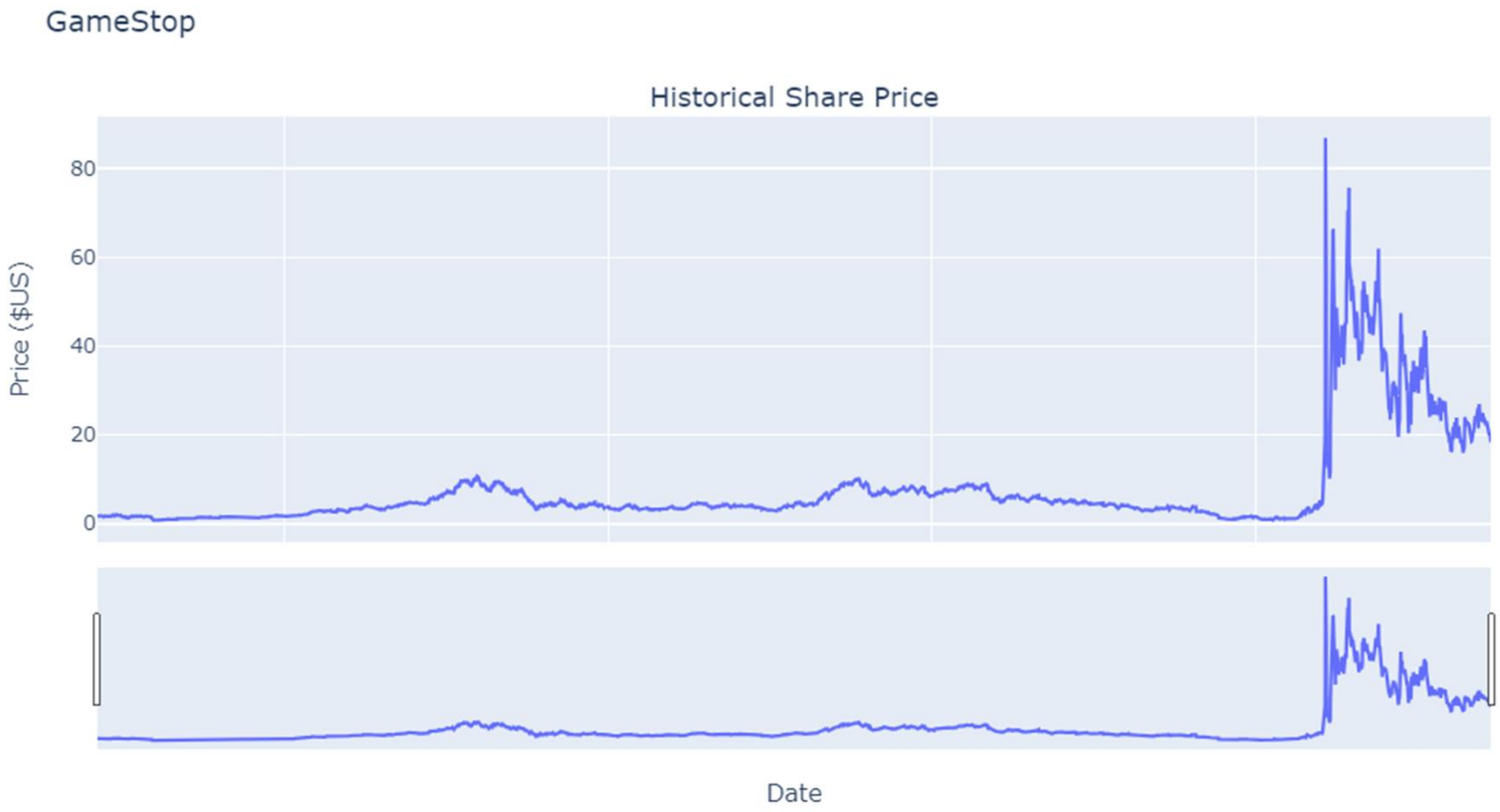


Plot GameStop Stock Graph

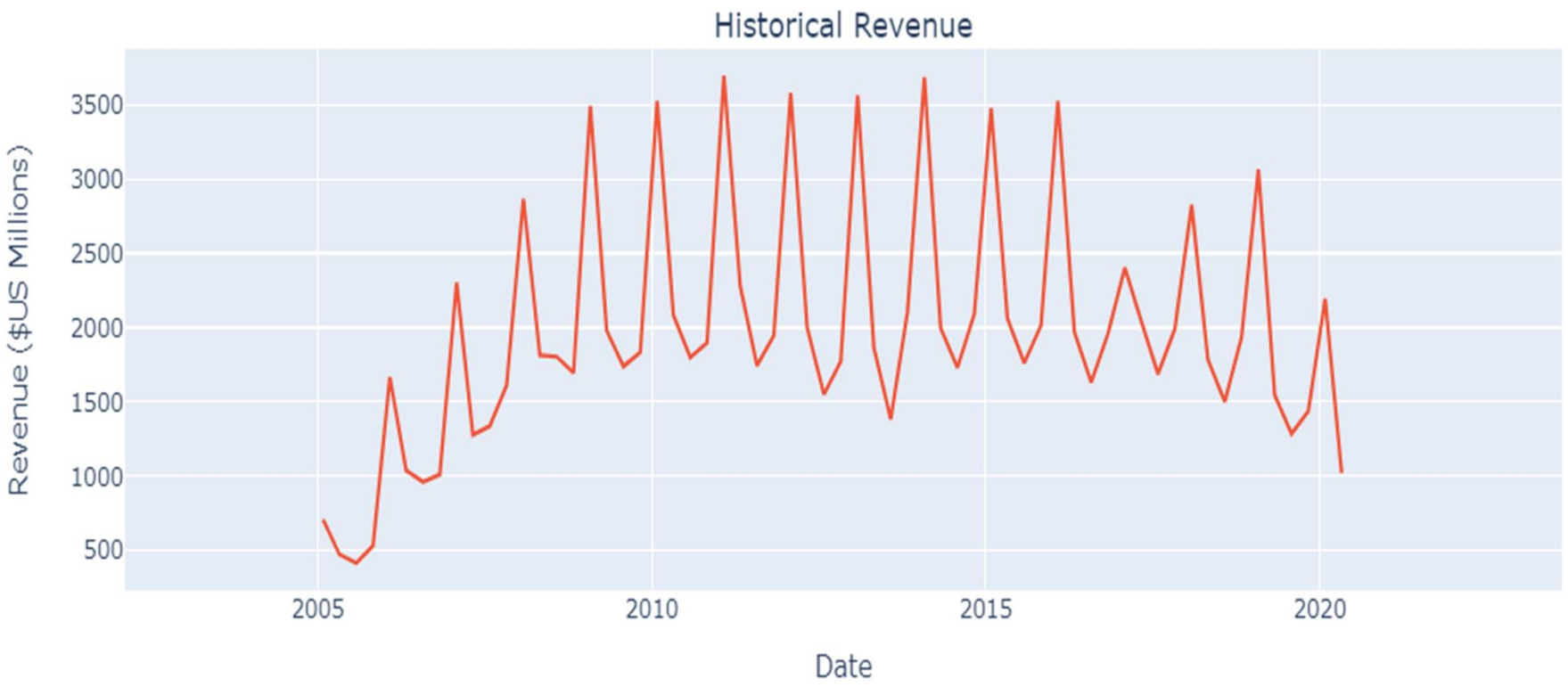
Question 6: Plot GameStop Stock Graph

Use the `make_graph` function to graph the GameStop Stock Data, also provide a title for the graph. The structure to call the `make_graph` function is `make_graph(gme_data, gme_revenue, 'GameStop')`. Note the graph will only show data upto June 2021.

In [22]:
```
make_graph(gme_data, gme_revenue, "GameStop")
```



Plot GameStop Stock Graph



Conclusion

- Extracting and visualizing stock data is crucial for data-driven decision-making.
- The project showcased the process of data extraction, web scraping, and visualization.
- Data visualization empowers us to gain insights and make informed choices.

DATABASE AND SQL FOR DATA SCIENCE WITH PYTHON

Exploring Chicago Datasets

Analyzing Socioeconomic Indicators, Schools, and Crime Data

Introduction

The primary objective of this presentation is to showcase how data analysis, specifically through SQL queries, can provide valuable insights into various facets of a city's dynamics. By utilizing real-world datasets, we aim to demonstrate the power of data-driven decision-making and its relevance in understanding complex urban environments.

Datasets Overview

- **Socioeconomic Indicators in Chicago:** This dataset provides a comprehensive set of socioeconomic indicators that shed light on the public health and well-being of different community areas within Chicago. It encompasses data from the years 2008 to 2012.

- **Chicago Public Schools:** Our second dataset presents performance data from Chicago Public Schools, specifically focusing on the 2011-2012 academic year. This dataset offers valuable insights into the educational landscape of the city.

- **Chicago Crime Data:** Lastly, we will delve into crime data from 2001 to the present, providing a deeper understanding of the city's safety and security trends over time.

Use of SQL Queries

- To derive meaningful insights from these datasets, we will employ SQL queries. SQL, or Structured Query Language, is a powerful tool that enables us to interact with databases and retrieve specific information. Through carefully crafted SQL queries, we can unveil patterns, trends, and relationships within the data, thereby enhancing our understanding of Chicago's socioeconomic, educational, and safety dynamics.

- As we progress through the presentation, you will witness firsthand how SQL queries can transform raw data into actionable insights. Let's embark on this journey of exploration and discovery, unraveling the multifaceted nature of the city of Chicago.

- So, without further ado, let's dive into our analysis and see what the data has to reveal.

SQL Queries

- Problem 1 - Total Number of Crimes

```
SELECT COUNT(*) AS TotalCrimes FROM
CHICAGO_CRIME_DATA;
```

- Problem 2 - Community Areas with Low Per Capita Income

```
SELECT COMMUNITY_AREA_NAME
FROM CENSUS_DATA
WHERE PER_CAPITA_INCOME < 11000;
```

- Problem 3 - Crimes Involving Minors

```
SELECT CASE_NUMBER, PRIMARY_TYPE, DESCRIPTION
FROM CHICAGO_CRIME_DATA
WHERE DESCRIPTION LIKE '%MINOR%' OR PRIMARY_TYPE
LIKE '%MINOR%';
```

- Problem 4 - Kidnapping Crimes Involving Children

```
SELECT DISTINCT CASE_NUMBER, PRIMARY_TYPE, DATE,
DESCRIPTION
FROM CHICAGO_CRIME_DATA
WHERE PRIMARY_TYPE = 'KIDNAPPING';
```

Problem 8 - Most Crime-Prone Community Area

```
SELECT COMMUNITY_AREA_NUMBER, COUNT(COMMUNITY_AREA_NUMBER) AS Frequency
FROM CHICAGO_CRIME_DATA
GROUP BY COMMUNITY_AREA_NUMBER
ORDER BY Frequency DESC
LIMIT 1;
```

Problem 9 - Community Area with Highest Hardship Index

```
SELECT COMMUNITY_AREA_NAME
FROM CENSUS_DATA
WHERE HARDSHIP_INDEX = (SELECT MAX(HARDSHIP_INDEX) FROM CENSUS_DATA);
```

Problem 10 - Community Area with Most Crimes

```
SELECT COMMUNITY_AREA_NAME
FROM CENSUS_DATA
WHERE COMMUNITY_AREA_NUMBER = (SELECT COMMUNITY_AREA_NUMBER FROM
CHICAGO_CRIME_DATA
   GROUP BY COMMUNITY_AREA_NUMBER
   ORDER BY COUNT(COMMUNITY_AREA_NUMBER) DESC
   LIMIT 1)
LIMIT 1;
```

Conclusion

- **1. Data-Driven Insights:** Through the strategic use of SQL queries, we've been able to extract meaningful insights from three diverse datasets. Our analysis has showcased the power of data in uncovering hidden patterns, trends, and relationships within complex real-world scenarios.
- **2. Socioeconomic Landscape:** Our examination of socioeconomic indicators has illuminated the disparities and challenges faced by different community areas within Chicago. We've identified areas with lower per capita income and higher percentages of households below the poverty line.
- **3. Educational Perspective:** The analysis of Chicago Public Schools data has provided valuable insights into the educational landscape. We've explored safety scores across different school types, revealing variations in safety perceptions within the city's educational institutions.
- **4. Crime Trends and Hotspots:** By diving into crime data, we've identified crime-prone community areas and specific types of crimes recorded at schools. This information is crucial for policymakers and law enforcement agencies to prioritize resources and enhance public safety.

- **5. Community Empowerment:** Our findings empower communities with data-driven knowledge to advocate for positive change. By understanding hardship indices and poverty rates, communities can better address socio-economic challenges and work towards a brighter future.
- **6. Importance of Data Analysis:** This analysis underscores the importance of data analysis in informed decision-making. It showcases how data-driven insights can guide policy formulation, resource allocation, and targeted interventions.
- **7. Continuous Learning:** The world of data analysis is vast and ever-evolving. This presentation serves as a starting point, encouraging us to continue exploring, learning, and applying data-driven approaches to understand and transform our communities.
- In closing, we've witnessed how data has the power to inform, guide, and inspire positive change. Let this be a reminder that behind every data point lies a story waiting to be told, and as data enthusiasts, we have the privilege of unraveling these stories for the betterment of society.

Data Analysis with Python

Introduction

Introduction to House Sales Data Analysis

- Dataset: House sale prices in King County, USA (May 2014 - May 2015)
- Features: id, date, price, bedrooms, bathrooms, sqft_living, sqft_lot, floors, waterfront, view, condition, grade, sqft_above, sqft_basement, yr_built, yr_renovated, zipcode, lat, long, sqft_living15, sqft_lot15

Dataset Overview

- Display the first few rows of the dataset
- Display data types of each column

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

Out[ ]:

	Unnamed: 0	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	...	grade	sqft_above	sqft_basement	yr_built	yr_renovated	zipcode	lat	long	sqft_living15	sqft_lot15
0	0	7129300520	20141013T000000	221900.0	3.0	1.00	1180	5650	1.0	0	...	7	1180	0	1955	0	98178	47.5112	-122.257	1340	5650
1	1	6414100192	20141209T000000	538000.0	3.0	2.25	2570	7242	2.0	0	...	7	2170	400	1951	1991	98125	47.7210	-122.319	1690	7639
2	2	5631500400	20150225T000000	180000.0	2.0	1.00	770	10000	1.0	0	...	6	770	0	1933	0	98028	47.7379	-122.233	2720	8062
3	3	2487200875	20141209T000000	604000.0	4.0	3.00	1960	5000	1.0	0	...	7	1050	910	1965	0	98136	47.5208	-122.393	1360	5000
4	4	1954400510	20150218T000000	510000.0	3.0	2.00	1680	8080	1.0	0	...	8	1680	0	1987	0	98074	47.6168	-122.045	1800	7503

5 rows × 22 columns

Question 1

Display the data types of each column using the function dtypes, then take a screenshot and submit it, include your code in the image.

In [ ]:
```
df.dtypes
```

Out[ ]:
```
Unnamed: 0         int64
id                 int64
date              object
price            float64
bedrooms         float64
bathrooms        float64
sqft_living        int64
sqft_lot           int64
floors           float64
waterfront         int64
view               int64
condition          int64
grade              int64
sqft_above         int64
sqft_basement      int64
yr_built           int64
yr_renovated       int64
zipcode            int64
lat              float64
long             float64
sqft_living15      int64
sqft_lot15         int64
dtype: object
```

Data Summary

- Display statistical summary of the dataset
- Drop columns 'id' and 'Unnamed: 0'
- Display updated statistical summary

Module 2: Data Wrangling

Question 2

Drop the columns `"id"` and `"Unnamed: 0"` from axis 1 using the method `drop()`, then use the method `describe()` to obtain a statistical summary of the data. Take a screenshot and submit it, make sure the `inplace` parameter is set to `True`

```
[ ]: df.drop(['id','Unnamed: 0'],axis=1,inplace=True)
     df.describe()
```

[ ]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	sqft_above	sqft_basement	yr_built	yr_renovated	zipcode	lat	long	sqft_living15	sqft_lot15
count	2.161300e+04	21600.000000	21603.000000	21613.000000	2.161300e+04	21613.000000	21613.000000	21613.000000	21613.000000	21613.000000	21613.000000	21613.000000	21613.000000	21613.000000	21613.000000	21613.000000	21613.000000	21613.000000	21613.000000
mean	5.400881e+05	3.372870	2.115736	2079.899736	1.510697e+04	1.494309	0.007542	0.234303	3.409430	7.656873	1788.390691	291.509045	1971.005136	84.402258	98077.939805	47.560053	-122.213896	1986.552492	12768.455652
std	3.671272e+05	0.926657	0.768996	918.440897	4.142051e+04	0.539989	0.086517	0.766318	0.650743	1.175459	828.090978	442.575043	29.373411	401.679240	53.505026	0.138564	0.140828	685.391304	27304.179631
min	7.500000e+04	1.000000	0.500000	290.000000	5.200000e+02	1.000000	0.000000	0.000000	1.000000	1.000000	290.000000	0.000000	1900.000000	0.000000	98001.000000	47.155900	-122.519000	399.000000	651.000000
25%	3.219500e+05	3.000000	1.750000	1427.000000	5.040000e+03	1.000000	0.000000	0.000000	3.000000	7.000000	1190.000000	0.000000	1951.000000	0.000000	98033.000000	47.471000	-122.328000	1490.000000	5100.000000
50%	4.500000e+05	3.000000	2.250000	1910.000000	7.618000e+03	1.500000	0.000000	0.000000	3.000000	7.000000	1560.000000	0.000000	1975.000000	0.000000	98065.000000	47.571800	-122.230000	1840.000000	7620.000000
75%	6.450000e+05	4.000000	2.500000	2550.000000	1.068800e+04	2.000000	0.000000	0.000000	4.000000	8.000000	2210.000000	560.000000	1997.000000	0.000000	98118.000000	47.678000	-122.125000	2360.000000	10083.000000
max	7.700000e+06	33.000000	8.000000	13540.000000	1.651359e+06	3.500000	1.000000	4.000000	5.000000	13.000000	9410.000000	4820.000000	2015.000000	2015.000000	98199.000000	47.777600	-121.315000	6210.000000	871200.000000

Handling Missing Values

- **Replace missing values in 'bedrooms' and 'bathrooms'**
- **Show the process of replacing missing values**

```
# Replace missing values in 'bedrooms'
mean_bedrooms = df['bedrooms'].mean()
df['bedrooms'].replace(np.nan, mean_bedrooms, inplace=True)

# Replace missing values in 'bathrooms'
mean_bathrooms = df['bathrooms'].mean()
df['bathrooms'].replace(np.nan, mean_bathrooms, inplace=True)
```

Exploratory Data Analysis

- **Count of Houses by Floors**

  df['floors'].value_counts().to_frame()

```
In [ ]: df.floors.value_counts().to_frame()
```

Out[12]:

	floors
1.0	10680
2.0	8241
1.5	1910
3.0	613
2.5	161
3.5	8

Waterfront View vs. Price

- **Price Outliers with Waterfront View**

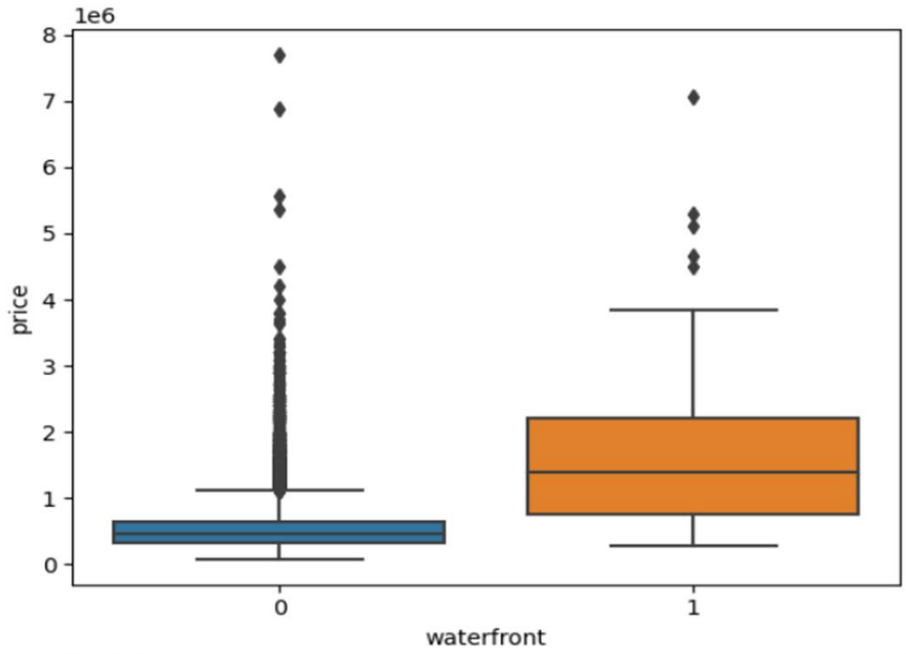
sns.boxplot(x='waterfront', y='price', data=df)

Question 4

Use the function `boxplot` in the seaborn library to determine whether houses with a waterfront view or without a waterfront view have more price outliers.

In [ ]:

```
sns.boxplot(x='waterfront',y='price',data=df);
```



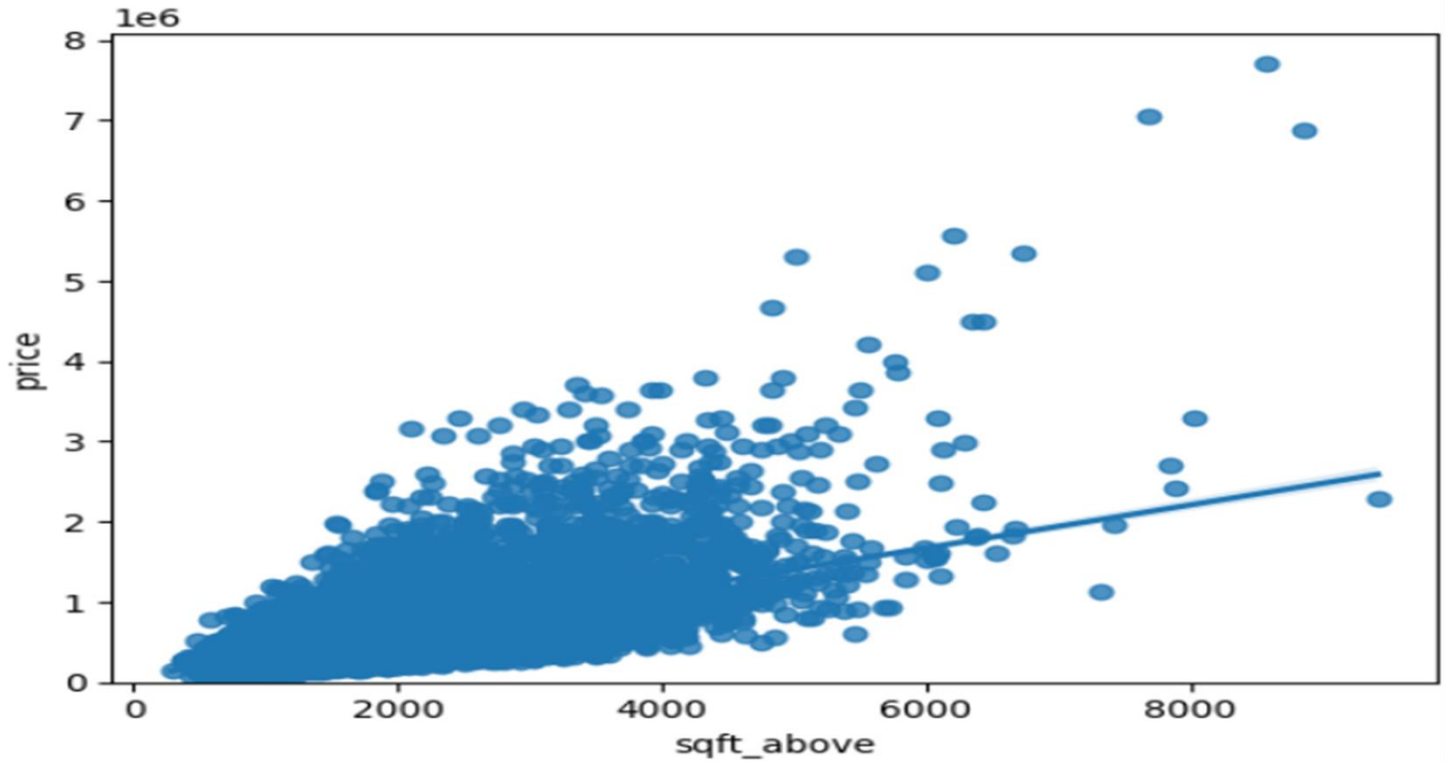
Sqft Above vs. Price

- **Sqft Above and Price**

sns.regplot(x=df['sqft_above'], y=df['price'], data=df)

```
[14]  sns.regplot(x=df["sqft_above"],y=df["price"],data=df)
      plt.ylim(0,)

      (0.0, 8081250.0)
```



Feature Correlations

- **Correlation of Features with Price**

df.corr()['price'].sort_values()

```
In [ ]: df.corr()['price'].sort_values()
```

```
<ipython-input-15-78b4f396fb2c>:1: FutureWarning: The default value of numeric_onl
e version, it will default to False. Select only valid columns or specify the valu
  df.corr()['price'].sort_values()
```

```
Out[15]: zipcode         -0.053203
long             0.021626
condition        0.036362
yr_built         0.054012
sqft_lot15       0.082447
sqft_lot         0.089661
yr_renovated     0.126434
floors           0.256794
waterfront       0.266369
lat              0.307003
bedrooms         0.308797
sqft_basement    0.323816
view             0.397293
bathrooms        0.525738
sqft_living15    0.585379
sqft_above       0.605567
grade            0.667434
sqft_living      0.702035
price            1.000000
Name: price, dtype: float64
```

Linear Regression - Longitude

Predicting Price Using Longitude

```
X = df[['long']]
Y = df['price']
lm = LinearRegression()
lm.fit(X, Y)
lm.score(X, Y)
```

Linear Regression - Multiple Features

- **Predicting Price Using Multiple Features**

```
features = ["floors", "waterfront", "lat",
"bedrooms", "sqft_basement", "view",
"bathrooms", "sqft_living15", "sqft_above",
"grade", "sqft_living"]
x = df[features]
y = df['price']
lr.fit(x, y)
lr.score(x, y)
```

Pipeline and Ridge Regression

- **Creating a Pipeline and Ridge Regression**

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.linear_model import LinearRegression, Ridge

features = ["floors", "waterfront", "lat", "bedrooms", "sqft_basement",
"view", "bathrooms", "sqft_living15", "sqft_above", "grade", "sqft_living"]
X = df[features]
Y = df['price']

steps = [('scale', StandardScaler()), ('polynomial',
PolynomialFeatures(include_bias=False)), ('model', LinearRegression())]
pipeline = Pipeline(steps)
pipeline.fit(X, Y)
pipeline.score(X, Y)
```

Model Evaluation and Refinement

```
from sklearn.model_selection import train_test_split

features = ["floors", "waterfront", "lat", "bedrooms",
"sqft_basement", "view", "bathrooms", "sqft_living15",
"sqft_above", "grade", "sqft_living"]
X = df[features]
Y = df['price']

x_train, x_test, y_train, y_test = train_test_split(X, Y,
test_size=0.15, random_state=1)

rr = Ridge(alpha=0.1)
rr.fit(x_train, y_train)
rr.score(x_test, y_test)
```

Data Visualization with python

Introduction

- The purpose of this presentation is to provide a comprehensive overview of our analysis and visualization of US domestic airline flight performance data.
- Our focus is on delving into the intricacies of airline operations, revealing insights that can shape strategic decisions and enhance customer experiences.
- To achieve this, we utilize a powerful tool: the Dash web application, which enables us to create interactive and dynamic visualizations of the data.
- Through our analysis and the intuitive Dash interface, we aim to uncover hidden patterns, trends, and key performance indicators within the realm of US domestic air travel.

Dash Web Application

- Dash is a robust Python web application framework designed for the creation of interactive and data-driven web applications.
- With Dash, we harness the power of Python to seamlessly develop dynamic web experiences that cater to data analysis and visualization needs.
- One of Dash's standout features is its ability to produce visually appealing and interactive data visualizations, making complex information more accessible and engaging.
- In the context of our presentation, Dash serves as the perfect tool to unlock insights from US domestic airline flight performance data.
- Our Dash application will be the gateway to exploring, understanding, and deriving valuable insights from the data, enabling us to make informed decisions and drive improvements in the airline industry.

Data Source

- Our analysis is based on a comprehensive dataset that captures the US domestic airline flight performance.
- This dataset provides a rich source of information, encompassing a wide range of attributes that shed light on various aspects of airline operations.

Key Features of the Dataset

- Year: The year of the recorded flight data.
- Month: The month of the recorded flight data.
- Reporting Airline: The airline responsible for reporting the flight data.
- Flights: Total number of flights for a given month and year.
- AirTime: Average duration of flights, offering insights into efficiency and operations.
- Cancellation Codes: Codes indicating reasons for flight cancellations.
- Delay Data: Different types of delays, such as carrier, weather, NAS, security, and late aircraft delays.

This dataset serves as the foundation for our analysis, enabling us to perform various visualizations and derive meaningful insights.

By leveraging this data, we can uncover trends, patterns, and performance metrics that drive our understanding of US domestic airline operations and contribute to informed decision-making within the industry.

Year and Report Type Selection

Our analysis is presented through an interactive Dash web application, allowing users to engage with the data and explore insights in a dynamic and user-friendly manner.

USER INTERACTION:

- **Year Selection Dropdown:**
  - Users can choose the year of interest within the range of 2005 to 2020.
  - This enables users to focus on specific timeframes and observe trends over the years.
- **Report Type Selection Dropdown:**
  - Users have the option to select between two report types:
    - Yearly Airline Performance Report: Provides an overview of airline performance metrics.
    - Yearly Airline Delay Report: Focuses on different delay factors affecting airline operations.

The user's selections will drive the content and visualization of the Dash application, allowing for tailored exploration and analysis of the US domestic airline flight performance data.

Yearly Airline Performance Report

- **Diverted Airport Landings:**
  - Presents the percentage of flights diverted to alternate airports per reporting airline.
  - Offers a perspective on airlines' operational flexibility and response to unforeseen circumstances.
- **Flight Counts by Origin State:**
  - Visualizes the number of flights departing from each US state.
  - Helps identify significant departure hubs and regional flight patterns.

Through these visualizations, users can gain valuable insights into the overall performance and operational aspects of domestic airlines throughout the selected year.

Monthly Flight Cancellation

- **Insights:**
  - Users can observe the variation in flight cancellations across different months.
  - Identification of peak cancellation months, possibly due to weather-related factors.
  - Analysis of the primary reasons for cancellations through cancellation codes.

Average Monthly Flight Time by Airline

Insights:

- Comparison of average flight times among different reporting airlines over the year.
- Identification of airlines with consistently shorter or longer flight durations.
- Understanding the potential impact of flight duration on customer satisfaction and operational efficiency.

Diverted Airport Landings

Insights:

- Assessment of the percentage of flights that were diverted to alternate airports by reporting airline.
- Evaluation of airlines' adaptability in handling unexpected situations.
- Identification of airlines with higher or lower rates of diverted flights.

Flight Counts by Origin State

Insights:

- Visualization of flight departure hubs across different US states.
- Recognition of major departure states and their contributions to overall flight volume.
- Understanding of regional flight distribution and its impact on airline network coverage.

Yearly Airline Delay Report

This section of the report focuses on analyzing different types of flight delays throughout the year.

Types of Delay:

- **Carrier Delay:** Delays caused by the airline, such as maintenance or crew issues.
- **Weather Delay:** Delays attributed to weather conditions.
- **NAS Delay (National Aviation System):** Delays due to factors like air traffic control, non-extreme weather, etc.
- **Security Delay:** Delays related to security concerns.
- **Late Aircraft Delay:** Delays caused by a previous flight being late.

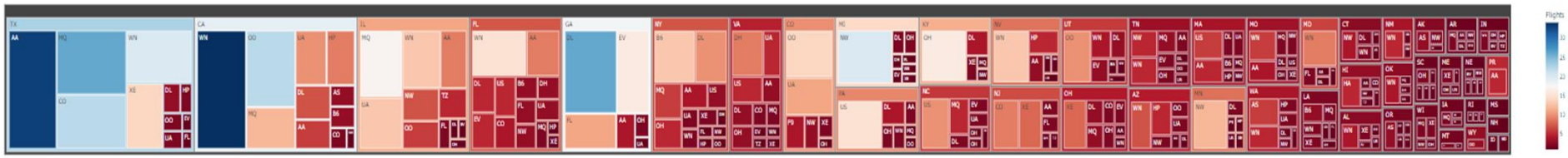
Yearly Airline Performance Report

US Domestic Airline Flights Performance

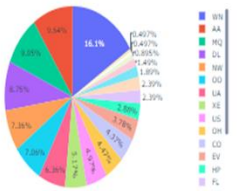
Report Type: Yearly Airline Performance Report

Choose Year: 2005

Flight count by airline to destination state

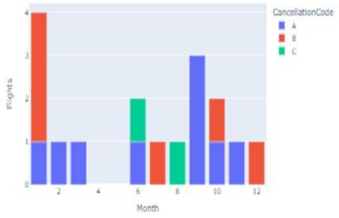


% of flights by reporting airline

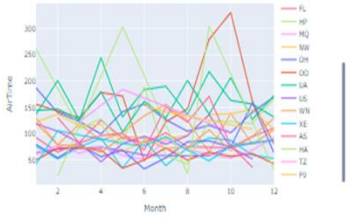


Number of flights from origin state

Monthly Flight Cancellation



Average monthly flight time (minutes) by airline



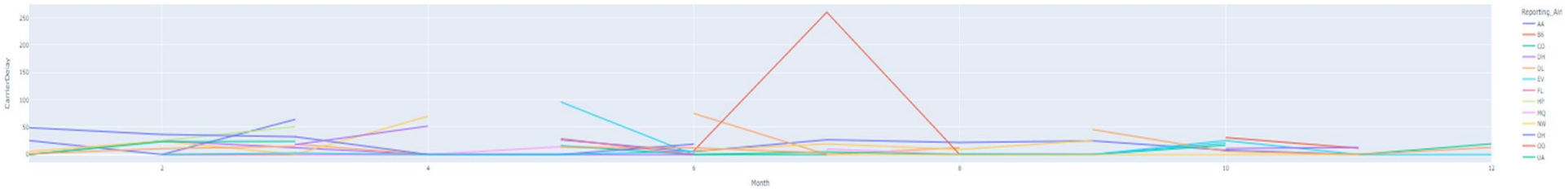
Yearly Airline Delay Report

US Domestic Airline Flights Performance

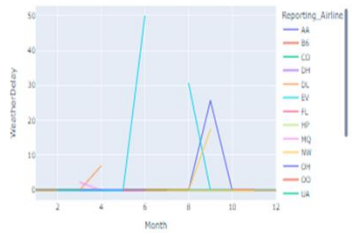
Report Type: Yearly Airline Delay Report

Choose Year: 2005

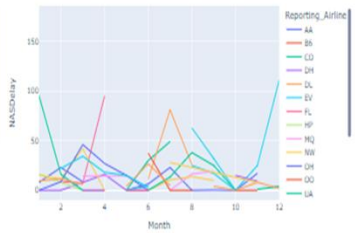
Average carrier delay time (minutes) by airline



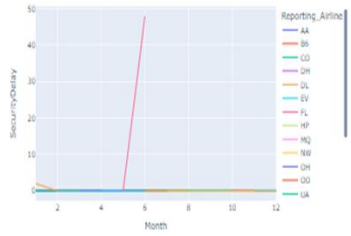
Average weather delay time (minutes) by airline



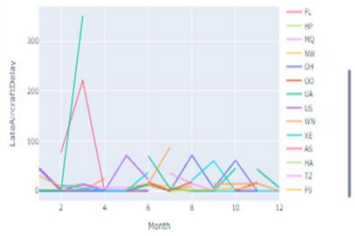
Average NAS delay time (minutes) by airline



Average security delay time (minutes) by airline



Average late aircraft delay time (minutes) by airline



Average Carrier Delay Time by Airline

Insights:

- Evaluation of average carrier delay times by reporting airline over the year.
- Identification of airlines contributing to higher average carrier delays.
- Understanding the impact of carrier-related issues on flight schedules.

Average Weather Delay Time by Airline

Insights:

- Analysis of average weather delay times experienced by different airlines.
- Identification of airlines more prone to weather-related disruptions.
- Understanding the extent to which weather affects airline operations.

Average NAS Delay Time by Airline

Insights:

- Assessment of average NAS delay times for each reporting airline.
- Identification of airlines affected by air traffic control or system-related delays.
- Understanding the role of the national aviation system in flight delays.

Average Security Delay Time by Airline

Insights:

- Analysis of average security delay times across reporting airlines.
- Identification of airlines with higher or lower security-related delays.
- Understanding the impact of security procedures on flight schedules

verage Late Aircraft Delay Time by Airline

Insights:

- Evaluation of average late aircraft delay times for each airline.
- Identification of airlines more prone to delays caused by previous late flights.
- Understanding the influence of interconnected flights on punctuality.

Conclusion

In conclusion, this presentation has provided a comprehensive overview of our analysis and visualization of US domestic airline flight performance using the Dash web application.

We started by introducing Dash as a powerful Python web application framework that enables the creation of interactive and visually appealing data visualizations. Our focus was on analyzing and visualizing US domestic airline flight performance data, aiming to offer valuable insights into various aspects of airline operations.

We explored two main report types: the "Yearly Airline Performance Report" and the "Yearly Airline Delay Report." Through these reports, users can interact with the Dash web application, selecting specific years and report types to gain a deeper understanding of airline performance and flight delays.

Conclusion

The "Yearly Airline Performance Report" provided insights into monthly flight cancellations, average monthly flight times by airline, diverted airport landings, and flight counts by origin state. These visualizations empower users to assess overall airline performance throughout the year, understand trends, and identify potential areas for improvement.

The "Yearly Airline Delay Report" focused on different types of flight delays, including carrier, weather, NAS, security, and late aircraft delays. By visualizing and analyzing these delay types, users can gain valuable insights into the factors affecting flight punctuality and make informed decisions to enhance airline operations and passenger experiences.

Through this presentation, we demonstrated the capabilities of the Dash web application in transforming complex airline performance data into meaningful and interactive visualizations, thereby facilitating data-driven decision-making and fostering a deeper understanding of the aviation industry.

THANK YOU