IBM Developer
SKILLS NETWORK

Winning Space Race with Data Science

SuiYee Feng
2025/02/19

Oudtline

- Executive Summary
- Introduction
- Methodology
- Results
- Conclusion
- Appendix

Executive Summary

- Summary of methodologies
- Summary of all results

Introduction

- Project background and context
- Problems you want to find answers

Section 1

Methodology

Methodology

Executive Summary

- Data collection methodology:
  - Describe how data was collected
- Perform data wrangling
  - Describe how data was processed
- Perform exploratory data analysis (EDA) using visualization and SQL
- Perform interactive visual analytics using Folium and Plotly Dash
- Perform predictive analysis using classification models
  - How to build, tune, evaluate classification models

Data Collection

1.From Files – CSV Files:Python's pandas library is a popular choice for reading CSV (Comma – Separated Values) files. For example, if you have a data.csv file with columns like id , name , age , you can use the following code to collect data:

```
]: import pandas as pd
   data = pd.read_excel('example.xlsx')
```

2.From Databases– SQLite: Python has the sqlite3 built – in library to interact with SQLite databases. To collect data from a SQLite database, you first need to establish a connection:

```
: import sqlite3
  conn = sqlite3.connect('example.db')
  cursor = conn.cursor()
  query = "SELECT * FROM users"
  cursor.execute(query)
  data = cursor.fetchall()
  conn.close()
```

Data Collection

3.From APIs– Using requests Library:Many websites and services provide APIs (Application Programming Interfaces) to access their data. For example, if you want to get data from the GitHub API to get information about a user's repositories. First, import the requests library:

```
import requests
username = 'your_username'
url = f'https://api.github.com/users/{username}/repos'
response = requests.get(url)
if response.status_code == 200:
    data = response.json()
else:
    print(f"Error: {response.status_code}")
```

3.Web Scraping (with Caution)Using BeautifulSoup :Web scraping is used to extract data from websites. For example, if you want to scrape the titles of articles from a news website. First, install BeautifulSoup using pip install beautifulsoup4 and import the necessary libraries:

```
]: import requests
   from bs4 import BeautifulSoup

   url = 'https://example - news - site.com'
   response = requests.get(url)
   if response.status_code == 200:
```

Data Collection – SpaceX API

- Start: The beginning of the data – collection process.
- Define API Endpoint Identify the specific SpaceX API endpoint from which data is to be retrieved, such as https://api.spacexdata.com/v4/launches for launch – related data.
- Prepare GET Request
- Send Request
- Parse JSON Response
- Use/Store Data
- github_url:https://github.com/yy0426y/Applied_Data_Science/blob/main/get_space_x_api.py

```python
def get_spacex_launches():  1 usage
    api_url = 'https://api.spacexdata.com/v4/launches'
    try:
        response = requests.get(api_url)
        response.raise_for_status()
        data = response.json()
        return data
    except requests.RequestException as e:
        print(f"An error occurred: {e}")
        return None


launches = get_spacex_launches()
if launches:
    for launch in launches:
        print(f"Mission Name: {launch.get('name')}, Launch Date: {launch.get('date_utc')}")
```

Data Collection - Scraping

- Key Phrases for Web Scraping Process: Website Selection,HTML Parsing,Selector Identification,Request Sending,Data Cleaning and Extraction,Error Handling,Compliance
- github_url: https://github.com/yy0426y/Applied_Data_Science/blob/main/web_scraping_process.py

```python
def scrape_example_website():  1 usage
    url = 'http://example.com'
    try:
        response = requests.get(url)
        response.raise_for_status()
        soup = BeautifulSoup(response.content, 'html.parser')
        # Assume article titles are in h2 tags with class 'article - title'
        article_titles = soup.find_all('h2', class_='article - title')
        for title in article_titles:
            clean_title = title.get_text().strip()
            print(clean_title)
    except requests.RequestException as e:
        print(f"An error occurred: {e}")


scrape_example_website()
```

Data Wrangling

- Here is a general description of how data can be processed, which can vary depending on the nature of the data and the specific goals of the analysis:
-
- Data Collection:
-
- – Gather data from various sources such as sensors, databases, surveys, or web scraping. For example, a company might collect customer purchase data from its point–of–sale systems and website analytics data.
-
- Data Cleaning:
-
- – Remove or correct inaccurate, incomplete, or inconsistent data. This could involve handling missing values (e.g., filling them with appropriate estimates or removing rows with missing values if they are not significant), correcting spelling errors, and standardizing data formats. For instance, if date values are in different formats, they need to be converted to a single standard format.
-

Data Wrangling

- Data Integration:
-
- – Combine data from multiple sources into a single dataset. This may require resolving conflicts in data definitions, such as different naming conventions for the same entity. For example, merging customer data from a CRM system and an e–commerce platform.
-
- Data Transformation:
-
- – Modify the data to make it more suitable for analysis. This can include normalizing numerical data (e.g., scaling values to a specific range like 0 to 1), encoding categorical variables (e.g., converting text labels like "male" and "female" into numerical codes), and aggregating data (e.g., calculating monthly totals from daily sales records).
-
- Data Reduction:
-

Data Wrangling

- – Reduce the volume of data while preserving its essential characteristics. Techniques like dimensionality reduction (e.g., principal component analysis, PCA) can be used to transform a large number of variables into a smaller set of uncorrelated variables. Sampling can also be employed to select a representative subset of the data for analysis.

- Data Analysis:

  – Apply appropriate statistical, machine learning, or other analytical techniques to the processed data. For example, using regression analysis to predict future sales based on historical data, or clustering algorithms to group customers based on their purchasing behavior.

- Data Interpretation and Visualization:

Data Wrangling

- – Interpret the results of the analysis and present them in a meaningful way. Visualization tools such as charts, graphs, and dashboards can be used to communicate the findings effectively. For example, creating a bar chart to show the distribution of sales across different product categories.
- In summary, data processing is a multi–step process that involves collecting, cleaning, integrating, transforming, reducing, analyzing, and visualizing data to extract useful insights. The specific steps and techniques used will depend on the type of data and the analysis objectives. If you can provide more details about the specific data and the context in which it is being processed, I can give a more tailored description.

github_url:https://github.com/yy0426y/Applied_Data_Science/blob/main/web_scraping_process.py

EDA with Data Visualization

Charts Plotted: A bar chart was plotted using the `plot(kind='bar')` method. The bar chart visualizes the average sales amount for each product, with the product names on the x-axis and the average sales amount on the y-axis.

- Reasons for Using the Bar Chart:
- Comparison: Bar charts are excellent for comparing values across different categories. In this case, the different product names are the categories, and the average sales amounts are the values. It allows for a clear and straightforward comparison of which products have higher or lower average sales. One can quickly glance at the chart and identify the relative performance of each product in terms of sales.
- Clarity: The visual representation of bars makes it easy to understand the data at a glance. The height of each bar directly corresponds to the value it represents (average sales amount), so the viewer can easily interpret the magnitude of the values for each product without much effort.

EDA with Data Visualization

- Categorical Data: Since the data involves distinct product names (categorical data), a bar chart is a natural choice for presenting this type of information. It effectively organizes and displays the relationship between the categories (products) and the numerical values associated with them (average sales amounts).
- In summary, the bar chart was used because it is a suitable and effective way to display and compare the average sales amounts across different product categories.

github_url: https://github.com/yy0426y/Applied_Data_Science/blob/main/DataAnalysis.ipynb

https://github.com/yy0426y/Applied_Data_Science/blob/main/Visualization_1.ipynb

https://github.com/yy0426y/Applied_Data_Science/blob/main/Visualization_2.ipynb

EDA with SQL

- Since no SQL queries were actually performed in the previous Python code example dealing with data processing using Pandas and Matplotlib, I'll provide a general summary of common types of SQL queries related to the data processing steps demonstrated earlier (data collection, cleaning, transformation, analysis) and how they might be structured.
- - Data Collection (Retrieving Data from a Table):

- SELECT * FROM sales_table; - This query retrieves all columns and rows from a table named sales_table . It's used to gather the initial data for further processing.

- SELECT date, product_name, sales_amount FROM sales_table; - Selects specific columns (date , product_name , sales_amount) from the sales_table when only those columns are needed for analysis.

- - Data Cleaning:
- - Removing Duplicates: DELETE FROM sales_table WHERE (date, product_name, sales_amount) IN (SELECT date, product_name, sales_amount FROM (SELECT date, product_name, sales_amount, COUNT(*) OVER (PARTITION BY date, product_name, sales_amount) AS count FROM sales_table) AS subquery WHERE count > 1); - This complex query first uses a window function to count the number of occurrences of each combination of date , product_name , and sales_amount . Then it deletes the rows that are duplicates (where the count is greater than 1).

EDA with SQL

- - Handling Missing Values (Deleting Rows with Missing Values): DELETE FROM sales_table WHERE date IS NULL OR product_name IS NULL OR sales_amount IS NULL; - Removes rows where any of the important columns (date , product_name , sales_amount) have a NULL value.
- - Data Transformation:
- - Converting Data Types (Assuming date is stored as a string and needs to be converted to a proper date type): ALTER TABLE sales_table ALTER COLUMN date TYPE DATE USING date::DATE; - In PostgreSQL, this query changes the data type of the date column from its current type (assumed to be a string) to the DATE type.
- - Normalizing a Numerical Column (Sales Amount in this case): This is a bit more complex in SQL. One approach could be to calculate the minimum and maximum values first and then use them to normalize. For example:

EDA with SQL

- - WITH MinMax AS (SELECT MIN(sales_amount) AS min_amount, MAX(sales_amount) AS max_amount FROM sales_table) UPDATE sales_table SET sales_amount = (sales_amount - (SELECT min_amount FROM MinMax)) / ((SELECT max_amount FROM MinMax) - (SELECT min_amount FROM MinMax)); - This uses a Common Table Expression (CTE) to calculate the minimum and maximum values of the sales_amount column and then updates the sales_amount column with the normalized values.
- - Data Analysis:

- Calculating Average Sales per Product: SELECT product_name, AVG(sales_amount) AS average_sales FROM sales_table GROUP BY product_name; - Groups the data by product_name and calculates the average sales_amount for each product group, giving the result as product_name and its corresponding average_sales .

These are just examples of SQL queries that could be used to perform similar data processing tasks as shown in the Python code example. The actual implementation may vary depending on the database system being used (e.g., MySQL, PostgreSQL, SQL Server, etc.).

github_url: https://github.com/yy0426y/Applied_Data_Science/blob/main/sql_use.py

Build an Interactive Map with Folium

- Since you haven't provided any specific code related to creating and adding objects like markers, circles, and lines to a Folium map, I'll give a general summary of these objects and reasons for adding them:
- 1. Markers
- – Creation: In Folium, a marker is created using the folium.Marker() function. For example, folium.Marker(location=[latitude, longitude], popup='Some information about the location').add_to(map_object) . Here, latitude and longitude specify the position on the map, and popup is the text or content that appears when the user clicks on the marker.
- – Reason for adding: Markers are used to denote specific points of interest on the map. They can represent locations such as businesses, landmarks, or points of significance in a dataset. For instance, if you are mapping the locations of restaurants in a city, each restaurant's location can be marked with a marker. The popup can provide additional details like the restaurant's name, cuisine type, or opening hours.

Build an Interactive Map with Folium

- 2. Circles
- – Creation: Circles are created using folium.Circle(location=[latitude, longitude], radius=radius_value, color='circle_color', fill=True, fill_color='fill_color') . The radius parameter determines the size of the circle in meters, and the color and fill_color parameters define the outline and fill colors of the circle respectively.
- – Reason for adding: Circles can be used to represent areas of a certain size or influence. For example, if you are mapping the service area of a delivery company, you can use circles to show the radius within which they deliver. They can also be used to represent the area of a geographical feature like a lake or a park. The color and fill can be adjusted to distinguish different types of areas or to show the intensity of a certain variable (e.g., population density within a circular area).

Build an Interactive Map with Folium

- 3. Lines
- – Creation: Lines are created using folium.PolyLine(locations=[[latitude1, longitude1], [latitude2, longitude2],...], color='line_color', weight=line_weight) . The locations parameter is a list of coordinate pairs that define the path of the line.
- – Reason for adding: Lines are useful for representing routes, boundaries, or connections between different points. For example, if you are mapping a hiking trail, you can use a line to show the path of the trail. In a transportation context, lines can represent bus routes or flight paths. The color and weight of the line can be adjusted to make it more visible or to distinguish between different types of routes.
- In summary, markers, circles, and lines in a Folium map are used to visually represent different types of geographical information and relationships, making the map more informative and useful for the viewer. If you provide specific code, I can give a more detailed and accurate summary of the objects created and the reasons for adding them.

github_url:https://github.com/yy0426y/Applied_Data_Science/blob/main/folium_map.py

Build a Dashboard with Plotly Dash

- Plots/Graphs
- Area Charts: These display the evolution of a variable over time, with the area below the line filled. They are useful for showing the cumulative effect of different components over time. For example, in a financial dashboard, an area chart can show the breakdown of revenue sources (such as product sales, subscriptions, and advertising) over quarters or years. The filled areas make it easy to visualize the contribution of each component to the total and how that changes over time.
- Histograms: Represent the distribution of a single variable. They divide the range of values into bins and show the frequency of data points falling into each bin. In a quality control dashboard, a histogram can display the distribution of product dimensions. This helps in identifying if the production process is centered around the target value and if there is excessive variation.
- Gauge Charts: Circular or semi-circular visual elements that show a single value relative to a target or range. In a performance management dashboard, a gauge chart can display the progress towards a sales target as a percentage. It gives an immediate visual indication of how close or far the actual value is from the goal.
- Tree Maps: Use nested rectangles to represent hierarchical data. Each rectangle represents a category or subcategory, and the size of the rectangle is proportional to a numerical value associated with it. In a business analytics dashboard for a multi-department company, a tree map can show the distribution of expenses across different departments and sub-departments. This allows for a quick comparison of the relative sizes of different cost centers.

Build a Dashboard with Plotly Dash

- Interactions
- Sorting: Allows users to arrange data in ascending or descending order based on a particular variable. In a sales dashboard showing a list of products and their sales figures, sorting enables users to quickly identify the top-selling and least-selling products. This interaction helps in prioritizing analysis and focusing on the most significant data points.
- Multi-select: Enables users to select multiple data points or categories at once. For example, in a market research dashboard with a bar chart showing customer satisfaction scores for different product features, multi-select allows users to compare the scores of multiple features simultaneously. It provides a more comprehensive view of the relationships between different data elements.
- Dynamic Updates: When one element of the dashboard (such as a filter or a selected data point) is changed, other related plots and graphs update automatically. In a supply chain dashboard, if a user selects a specific time period using a date filter, all the relevant charts (such as inventory levels over time, delivery times, etc.) will update to reflect data only for that selected period. This ensures that the data presented remains consistent and relevant to the user's current analysis.
- Linking: Connects different plots or graphs so that actions in one affect the others. For instance, in a customer analytics dashboard, clicking on a particular segment in a pie chart representing customer demographics can highlight the corresponding data points in a scatter plot showing customer purchase behavior. This interaction helps in exploring relationships between different aspects of the data and uncovering hidden insights.
- github_url:https://github.com/yy0426y/Applied_Data_Science/blob/main/plots.py

Predictive Analysis (Classification)

- 1. Data Preparation
- – Key Phrases: Data collection, data cleaning, data integration, feature selection/extraction, data splitting (training set, validation set, test set)
- – Explanation: First, gather relevant data from various sources. Clean the data by handling missing values, outliers, and incorrect entries. Integrate data from multiple sources if needed. Select or extract relevant features that are likely to contribute to the classification task. Split the data into subsets: a training set to train the model, a validation set to tune hyperparameters and evaluate the model during development, and a test set for the final evaluation.
- 2. Model Selection
- – Key Phrases: Choose classification algorithm, baseline model, understanding algorithm assumptions
- – Explanation: Select an appropriate classification algorithm based on the nature of the data (e.g., linear models for linearly separable data, decision trees for hierarchical data patterns). Start with a simple baseline model to establish a performance benchmark and understand the characteristics and assumptions of the chosen algorithm.

Predictive Analysis (Classification)

- 3. Model Training
- – Key Phrases: Fit the model, training process, learning parameters
- – Explanation: Use the training data to fit the selected model. The model learns the relationships between the features and the target variable during this process. Update the model's parameters (weights in the case of neural networks or decision rules in decision trees) to minimize a loss function.
- 4. Model Evaluation
- – Key Phrases: Performance metrics (accuracy, precision, recall, F1–score, confusion matrix), validation set evaluation, cross–validation
- – Explanation: Use appropriate performance metrics to assess how well the model performs. Calculate metrics like accuracy (proportion of correct predictions), precision (true positives among predicted positives), recall (true positives among actual positives), and the F1–score (a weighted average of precision and recall). Evaluate the model on the validation set and consider using cross–validation techniques to get a more reliable estimate of the model's performance.
- 5. Hyperparameter Tuning
- – Key Phrases: Grid search, random search, Bayesian optimization, hyperparameter adjustment
- 6. Model Improvement
- – Key Phrases: Feature engineering, ensemble methods (bagging, boosting, stacking), model regularization
- 7. Final Evaluation on Test Set
- – Key Phrases: Unseen data evaluation, generalization assessment

Predictive Analysis (Classification)

- 8. Selecting the Best Model
- – Key Phrases: Compare models, performance comparison, choosing the optimal model

github_url:https://github.com/yy0426y/Applied_Data_Science/blob/main/predictive_analysis.py

Results

- Here is a general framework for presenting exploratory data analysis results, an example of how to describe an interactive analytics demo through screenshots, and how to show predictive analysis results. We'll continue with the Iris dataset example used before, but you can adapt this to other datasets as well.
- 1. Exploratory Data Analysis Results
- Summary Statistics
- – Numeric Features: For the Iris dataset, which has four numeric features (sepal length, sepal width, petal length, petal width), we can calculate summary statistics like mean, median, standard deviation, minimum, and maximum values. For example, the mean sepal length is approximately 5.84 cm, with a standard deviation of about 0.83 cm. These statistics give us an idea of the central tendency and spread of each feature.
- – Categorical Feature (Species): There are three species in the Iris dataset (setosa, versicolor, virginica). We can count the number of samples for each species. We find that there are 50 samples of each species, providing a balanced dataset for classification tasks.

Results

- Visualizations
- – Pair Plot: Using a pair plot (e.g., with seaborn library in Python), we can visualize the relationships between all pairs of numeric features. We observe that the setosa species has distinctively smaller petal lengths and widths compared to the other two species. This helps in understanding the separability of the classes based on these features.
- – Histogram: Histograms for each numeric feature show their distribution. For instance, the distribution of petal length is approximately normal for the versicolor and virginica species, while it has a more distinct peak for the setosa species.
- – Box Plot: Box plots can display the quartiles and outliers of each feature. We can see that there are relatively few outliers in the dataset, and the spread of the features varies among the different species.
- 2. Interactive Analytics Demo in Screenshots
- Step 1: Initial Dashboard
- – Screenshot: Shows a dashboard with a scatter plot of sepal length vs sepal width, colored by the species of the iris flower. There are also dropdown menus for selecting different features to plot on the x and y axes.
- – Explanation: The user can initially see the distribution of the data points in the feature space and how the different species are clustered. The dropdown menus allow for interactivity, enabling the user to explore different feature combinations.

Results

- Step 2: Filtering Data
- – Screenshot: After using a filter (e.g., a slider to select only iris flowers with a sepal length greater than 6 cm), the scatter plot updates to show only the relevant data points. The number of remaining data points is also displayed.
- – Explanation: This interaction demonstrates how the user can focus on specific subsets of the data. By filtering based on a feature value, the user can analyze the characteristics of that subset more closely.
- Step 3: Zooming and Hovering
- – Screenshot: A zoomed–in section of the scatter plot where the user has hovered over a data point. A tooltip appears showing the exact values of all features (sepal length, sepal width, petal length, petal width) and the species of the iris flower corresponding to that data point.
- – Explanation: Zooming allows for a detailed view of a particular region of the plot, and hovering provides additional context about individual data points. This helps in understanding the data at a more granular level.
- 3. Predictive Analysis Results
- Model Performance Metrics
- – For the tuned models on the Iris dataset (as in the previous Python example), we can present the following metrics for the test set:
- – Accuracy: The best performing model (e.g., Random Forest in some cases) might achieve an accuracy of around 0.97,30 meaning it correctly classifies 97% of the samples in the test set.

Results

- – Precision: Precision values for each class can be calculated. For example, for the setosa class, the precision might be 1.0, indicating that all the samples predicted as setosa are actually setosa.
- – Recall: Recall values show how well the model can identify the samples of each class. For versicolor, the recall could be around 0.95, meaning it can correctly identify 95% of the actual versicolor samples.
- – F1–score: The F1–score, which balances precision and recall, can also be presented. A high F1–score for each class indicates a good performance of the model in classifying that class.
- Confusion Matrix
- – A confusion matrix can be shown to visualize the number of true positives, false positives, true negatives, and false negatives for each class. For the Iris dataset, it can help in understanding which classes are being misclassified and to what extent. For example, we might see that a few samples of versicolor are misclassified as virginica, but overall, the misclassification rates are relatively low.
- Feature Importance (for Tree–based Models)
- – If using a tree–based model like Random Forest or Decision Tree, we can display the feature importance scores. We find that petal length and petal width are often the most important features for classifying the iris species, which aligns with our observations from the exploratory data analysis.
- This is a comprehensive way to present the different aspects of data analysis, from exploration to prediction, along with an interactive analytics component. You can adjust the details based on the specific dataset and analysis goals you have.

Section 2

Insights drawn from EDA

Flight Number vs. Launch Site

- Show a scatter plot of Flight Number vs. Launch Site
- Show the screenshot of the scatter plot with explanations

```python
# 创建示例数据
data = {
    'Flight Number': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
    'Launch Site': ['Site A', 'Site B', 'Site A', 'Site C', 'Site B', 'Site A', 'Site C', 'Site B', 'Site A', 'Site C']
}
df = pd.DataFrame(data)

# 将发射地点映射为数值以便绘图（因为散点图处理的是数值数据）
launch_site_mapping = {site: num for num, site in enumerate(set(df['Launch Site']))}
df['Launch Site Numeric'] = df['Launch Site'].map(launch_site_mapping)

# 创建散点图
plt.scatter(df['Flight Number'], df['Launch Site Numeric'])

# 将数值重新映射回实际的发射地点名称，用于y轴标签
plt.yticks(list(launch_site_mapping.values()), list(launch_site_mapping.keys()))

plt.xlabel('Flight Number')
plt.ylabel('Launch Site')
plt.title('Flight Number vs. Launch Site')
plt.show()
```

Payload vs. Launch Site

- Show a scatter plot of Payload vs. Launch Site
- Show the screenshot of the scatter plot with explanations

```
# Create sample data
data = {
    'Payload': [100, 150, 80, 200, 120, 90, 180, 130, 110, 170],
    'Launch Site': ['Site A', 'Site B', 'Site A', 'Site C', 'Site B', 'Site A', 'Site C', 'Site B', 'Site A', 'Site C']
}
df = pd.DataFrame(data)

# Map launch sites to numerical values for plotting (since scatter plot works with numbers)
launch_site_mapping = {site: num for num, site in enumerate(set(df['Launch Site']))}
df['Launch Site Numeric'] = df['Launch Site'].map(launch_site_mapping)

# Create the scatter plot
plt.scatter(df['Payload'], df['Launch Site Numeric'])

# Map the numerical values back to the actual launch site names for the y-axis labels
plt.yticks(list(launch_site_mapping.values()), list(launch_site_mapping.keys()))

plt.xlabel('Payload')
plt.ylabel('Launch Site')
plt.title('Payload vs. Launch Site')
```

Success Rate vs. Orbit Type

- Show a bar chart for the success rate of each orbit type
- Show the screenshot of the scatter plot with explanations

```
# Create sample data
data = {
    'Orbit Type': ['Low Earth Orbit', 'Geostationary Orbit', 'Polar Orbit', 'Medium Earth Orbit'],
    'Successful Launches': [80, 60, 40, 70],
    'Total Launches': [100, 80, 50, 90]
}
df = pd.DataFrame(data)

# Calculate the success rate for each orbit type
df['Success Rate'] = df['Successful Launches'] / df['Total Launches'] * 100

# Create the bar chart
plt.bar(df['Orbit Type'], df['Success Rate'])

plt.xlabel('Orbit Type')
plt.ylabel('Success Rate (%)')
plt.title('Success Rate vs. Orbit Type')
plt.xticks(rotation=45)  # Rotate x-axis labels for better readability
plt.show()
```

Flight Number vs. Orbit Type

- Show a scatter point of Flight number vs. Orbit type
- Show the screenshot of the scatter plot with explanations

```
# Create sample data
data = {
    'Flight Number': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
    'Orbit Type': ['Low Earth Orbit', 'Geostationary Orbit', 'Low Earth Orbit', 'Polar Orbit', 'Geostationary Orbit',
                   'Low Earth Orbit', 'Medium Earth Orbit', 'Polar Orbit', 'Geostationary Orbit', 'Medium Earth Orbit']
}
df = pd.DataFrame(data)

# Create the scatter plot using seaborn
sns.scatterplot(x='Flight Number', y='Orbit Type', data=df)

plt.xlabel('Flight Number')
plt.ylabel('Orbit Type')
plt.title('Flight Number vs. Orbit Type')
plt.show()
```

Payload vs. Orbit Type

- Show a scatter point of payload vs. orbit type
- Show the screenshot of the scatter plot with explanations

```
# Create sample data
data = {
    'Payload': [100, 150, 80, 200, 120, 90, 180, 130, 110, 170],
    'Orbit Type': ['Low Earth Orbit', 'Geostationary Orbit', 'Low Earth Orbit', 'Polar Orbit', 'Geostationary Orbit',
                   'Low Earth Orbit', 'Medium Earth Orbit', 'Polar Orbit', 'Geostationary Orbit', 'Medium Earth Orbit']
}
df = pd.DataFrame(data)

# Map orbit types to numerical values for plotting (since scatter plot works with numbers)
orbit_type_mapping = {orbit: num for num, orbit in enumerate(set(df['Orbit Type']))}
df['Orbit Type Numeric'] = df['Orbit Type'].map(orbit_type_mapping)

# Create the scatter plot
plt.scatter(df['Payload'], df['Orbit Type Numeric'])

# Map the numerical values back to the actual orbit type names for the y-axis labels
plt.yticks(list(orbit_type_mapping.values()), list(orbit_type_mapping.keys()))

plt.xlabel('Payload')
plt.ylabel('Orbit Type')
plt.title('Payload vs. Orbit Type')
plt.show()
```

Launch Success Yearly Trend

- Show a line chart of yearly average success rate
- Show the screenshot of the scatter plot with explanations

```
data = {
    'Payload': [100, 150, 80, 200, 120, 90, 180, 130, 110, 170],
    'Orbit Type': ['Low Earth Orbit', 'Geostationary Orbit', 'Low Earth Orbit', 'Polar Orbit', 'Geostationary Orbit',
                   'Low Earth Orbit', 'Medium Earth Orbit', 'Polar Orbit', 'Geostationary Orbit', 'Medium Earth Orbit']
}
df = pd.DataFrame(data)

# Map orbit types to numerical values for plotting (since scatter plot works with numbers)
orbit_type_mapping = {orbit: num for num, orbit in enumerate(set(df['Orbit Type']))}
df['Orbit Type Numeric'] = df['Orbit Type'].map(orbit_type_mapping)

# Create the scatter plot
plt.scatter(df['Payload'], df['Orbit Type Numeric'])

# Map the numerical values back to the actual orbit type names for the y-axis labels
plt.yticks(list(orbit_type_mapping.values()), list(orbit_type_mapping.keys()))

plt.xlabel('Payload')
plt.ylabel('Orbit Type')
plt.title('Payload vs. Orbit Type')
plt.show()
```

All Launch Site Names

- Find the names of the unique launch sites
- Present your query result with a short explanation here

```
# Read the data from the CSV file. Replace 'launch_data.csv' with the actual file path if needed.
df = pd.read_csv('launch_data.csv')

# Find the unique launch site names
unique_launch_sites = df['Launch Site'].unique()

print(unique_launch_sites)

launch_sites_list = ['Site A', 'Site B', 'Site A', 'Site C', 'Site B']
df = pd.DataFrame({'Launch Site': launch_sites_list})
unique_launch_sites = df['Launch Site'].unique()
print(unique_launch_sites)
```

Launch Site Names Begin with 'CCA'

- The launch sites beginning with 'CCA' are usually related to Cape Canaveral. Here are some of them and their brief introduction:
- – Cape Canaveral Space Force Station: It's one of the important launch sites in the United States. It has many launch pads and is an active rocket range and spaceport on the Atlantic coast of Florida.
- – Cape Canaveral Air Force Station (CCAFS): It's part of the Eastern Test Range in the US and has been used for numerous space launches and military missile tests.
- – Cape Canaveral Launch Complex 40: It's one of the launch complexes at Cape Canaveral. SpaceX has used this site to launch the Falcon 9 rocket.
- – Cape Canaveral Launch Complex 41: Adjacent to Launch Complex 40, it's also an important launch site at Cape Canaveral and has been used for various space missions.
- – Cape Canaveral Launch Complex 37: There are different pads like LC–37B. It has been used for launching different types of rockets and spacecraft over the years.
- These sites have played significant roles in the history of space exploration and have witnessed many important space launches, contributing greatly to the development of space technology.

Total Payload Mass

- Here is an example of how you could calculate the total payload carried by boosters from NASA assuming you have a dataset in a CSV file (let's say nasa_launch_data.csv) with columns like Booster (indicating the booster used, and you can filter for NASA-related ones), and Payload Mass (containing the mass of the payload in appropriate units like kilograms). We'll use the pandas library in Python to perform this calculation. First, make sure pandas is installed (pip install pandas if needed).
- python
- import pandas as pd
- # Read the data from the CSV file. Replace 'nasa_launch_data.csv' with the actual file path if needed.
- df = pd.read_csv('nasa_launch_data.csv')
- # Filter the data for boosters related to NASA. This step may vary depending on how NASA boosters are identified in your data.
- # For example, if there is a column indicating the operator and 'NASA' is in that column, you could do the following:
- nasa_df = df[df['Operator'] == 'NASA']
- # Calculate the total payload mass. Assume 'Payload Mass' is the column name for the payload mass values.
- total_payload_mass = nasa_df['Payload Mass'].sum()
- print(f"The total payload mass carried by NASA boosters is: {total_payload_mass}")

Total Payload Mass

- Explanation:
- We start by using the pd.read_csv() function from the pandas library to read the data from the specified CSV file into a DataFrame named df . This DataFrame stores the data in a tabular format that is easy to manipulate.
- Then, we filter the DataFrame to only include rows related to NASA boosters. The way we filter depends on how the data is structured. In the example code, we assume there is a column named Operator and we select only the rows where the Operator value is equal to 'NASA' . The resulting DataFrame (nasa_df) contains only the relevant rows for our calculation.
- Next, we calculate the sum of the Payload Mass column in the filtered DataFrame (nasa_df). We use the sum() method on the 'Payload Mass' column to add up all the individual payload mass values. The result of this calculation is the total payload mass carried by NASA boosters.
- Finally, we print out the result with a descriptive message so that we can easily see the total payload mass value.
- If your dataset has different column names or a different way of identifying NASA boosters, you'll need to adjust the code accordingly. For example, if the booster name itself contains a specific identifier for NASA, you might filter based on the Booster column instead of an Operator column.

Average Payload Mass by F9 v1.1

- Here is an example of how you can calculate the average payload mass carried by the booster version F9 v1.1 using Python with the pandas library. Assume you have a dataset in a CSV file (let's say launch_data.csv) that has columns like Booster Version which indicates the version of the booster used in a launch, and Payload Mass which contains the mass of the payload (in appropriate units like kilograms).
- First, make sure you have pandas installed. If not, you can install it using pip install pandas .
- The following is the Python code:
- python
- import pandas as pd
- # Read the data from the CSV file. Replace 'launch_data.csv' with the actual file path if needed.
- df = pd.read_csv('launch_data.csv')
- # Filter the data for the booster version 'F9 v1.1'
- f9v11_df = df[df['Booster Version'] == 'F9 v1.1']
- # Calculate the average payload mass. Assume 'Payload Mass' is the column name for the payload mass values.
- average_payload_mass = f9v11_df['Payload Mass'].mean()

Average Payload Mass by F9 v1.1

- print(f"The average payload mass carried by booster version F9 v1.1 is: {average_payload_mass}")
- Explanation:
- Reading the data: The pd.read_csv() function is used to read the data from the specified CSV file into a DataFrame named df . This DataFrame organizes the data in a tabular structure with rows and columns, making it convenient to perform data analysis operations.
- Filtering the data: We use boolean indexing on the DataFrame (df) to select only the rows where the Booster Version column has the value 'F9 v1.1' . The result of this filtering operation is a new DataFrame (f9v11_df) that contains only the relevant rows for our analysis, i.e., the launches that used the F9 v1.1 booster version.
- Calculating the average: We then use the mean() method on the 'Payload Mass' column of the filtered DataFrame (f9v11_df). This method calculates the arithmetic mean (average) of all the values in the 'Payload Mass' column. The resulting value is the average payload mass carried by the F9 v1.1

First Successful Ground Landing Date

- The date of the first successful ground landing of a space booster is December 21, 2015. It was achieved by SpaceX's Falcon 9 rocket on its 20th flight. After propelling 11 Orbcomm OG2 satellites to orbit, the first stage of the Falcon 9 successfully landed on Landing Zone 1 at Cape Canaveral Air Force Station. This achievement was a significant milestone in the development of reusable rocket technology, paving the way for subsequent space exploration and reducing the cost of space access.

Successful Drone Ship Landing with Payload between 4000 and 6000

- According to some data sources, the boosters that have successfully landed on a drone ship and had a payload mass between 4,000 and 6,000 kg include "F9 FT B1032.1", "F9 B4 B1040.1" and "F9 B4 B1043.1".
-
- SpaceX's Falcon 9 rockets are the main ones that perform such landings. These rockets have carried out many space missions. When the payload mass is within a certain range, they have successfully landed on drone ships at sea after completing their missions, which is a crucial step in realizing the reusability of rockets and reducing space launch costs.

Total Number of Successful and Failure Mission Outcomes

- Here is an example of how to calculate the total number of successful and failed mission outcomes using Python with the pandas library. Assume you have a dataset in a CSV file (let's say mission_data.csv) that has a column named Mission Outcome where the values are something like 'Success' or 'Failure' to indicate the result of the mission.
- First, make sure you have pandas installed. If not, you can install it using pip install pandas .
- The following is the Python code:
- python
- import pandas as pd
- # Read the data from the CSV file. Replace'mission_data.csv' with the actual file path if needed.
- df = pd.read_csv('mission_data.csv')
- # Count the number of successful mission outcomes
- num_successful = len(df[df['Mission Outcome'] == 'Success'])
- # Count the number of failed mission outcomes
- num_failed = len(df[df['Mission Outcome'] == 'Failure'])
- print(f"The total number of successful mission outcomes is: {num_successful}")
- print(f"The total number of failed mission outcomes is: {num_failed}")

Total Number of Successful and Failure Mission Outcomes

- Explanation:
- Reading the data: The pd.read_csv() function is used to read the data from the specified CSV file into a DataFrame named df . This DataFrame organizes the mission data in a tabular format with rows and columns, which allows us to easily access and manipulate the information.
- Counting successful outcomes: We use boolean indexing on the DataFrame (df) to select only the rows where the Mission Outcome column has the value 'Success' . The expression df['Mission Outcome'] == 'Success' creates a boolean Series that is True for rows with a successful outcome and False otherwise. We then use this boolean Series to index the DataFrame , getting a new DataFrame that contains only the rows with successful outcomes. The len() function is used to count the number of rows in this new DataFrame , which gives us the total number of successful mission outcomes.
- Counting failed outcomes: Similar to the previous step, we use boolean indexing to select the rows where the Mission Outcome column has the value 'Failure' . The resulting DataFrame contains only the rows with failed outcomes, and we use the len() function to count the number of rows, obtaining the total number of failed mission outcomes.
- Displaying the results: Finally, we print out the counts of successful and failed mission outcomes with descriptive messages so that we can easily understand the results of our analysis.
- If your dataset has a different column name for the mission outcome or uses different values to represent success and failure (e.g., 1 for success and 0 for failure

Boosters Carried Maximum Payload

- Some boosters known for carrying maximum payloads are as follows:
-
- – Delta IV Heavy Booster: The Delta IV Heavy rocket can carry up to 28,790 kg of payload to low Earth orbit. It consists of three Common Core Boosters in a straight line. It is the largest active launch vehicle in the US and has carried out many important space missions.
-
- – Falcon 9 Block 5 Booster: The Falcon 9 Block 5 can carry 22.8 tons of payload to low Earth orbit with an orbital inclination of 28.5°. It is an upgraded version of SpaceX's Falcon 9 series, and is widely used in various space missions with high reliability and reusability.
-
- – Starship Super Heavy Booster: The Starship system consists of a manned Starship and a Super Heavy Booster. The second–stage Starship can carry 100 tons of cargo to Mars. Although there have been no successful orbital flights with full payload capacity so far, in theory, it has extremely large payload capacity and is designed to achieve Mars colonization and other large–scale space exploration goals.

2015 Launch Records

- In 2015, SpaceX did not have any successful landings on a drone ship, and all of their attempts ended in failure. The details of the failed landings are as follows:
-
- - January 10, 2015: The Falcon 9 rocket launched from Cape Canaveral Air Force Station in Florida on a cargo resupply service mission to the International Space Station. The booster version was likely a standard Falcon 9 at that time. The rocket's fins ran out of hydraulic fluid, causing it to land too hard on the drone ship and break into pieces.
-
- - April 14, 2015: After the successful launch of SpaceX's Dragon spacecraft to the International Space Station from Cape Canaveral Air Force Station, the Falcon 9 booster rocket attempted to land on the drone ship "Just Read the Instructions" but ended up exploding on the barge. The booster version was likely a standard Falcon 9.
-
- In 2015, SpaceX was still in the experimental stage of drone ship landings, and these failures provided valuable data and experience for subsequent successful landings.

Rank Landing Outcomes Between 2010-06-04 and 2017-03-20

- Between 2010-06-04 and 2017-03-20, SpaceX's landing outcomes of Falcon 9 rockets are as follows:
-
- - Success (drone ship): 6 times. The first successful drone ship landing was in April 2016, and there were 5 more successful landings on drone ships by March 2017.
-
- - Success (ground pad): 2 times. The first successful ground pad landing was on December 21, 2015, and the second one was in July 2016.
-
- - Failure (drone ship): 5 times. In 2015, there were 2 failed drone ship landings. In 2016, there were 3 more failures before the first successful drone ship landing.
-
- - Failure (ground pad): 0 times. There were no recorded failures of ground pad landings within this period.
-
- In descending order of the count of landing outcomes, it is success (drone ship), success (ground pad), failure (drone ship), and failure (ground pad).


Section 3
Launch Sites
Proximities Analysis

<Folium Map Screenshot 1>

- Here's how you can approach this task:
- 1. Naming an Appropriate Title
- If the Folium map shows the locations of various rocket launch sites around the world, an appropriate title could be something like:
- - "Global Distribution of Rocket Launch Sites"
- - "Locations of Major Space Launch Facilities Across the Globe"
- - "Worldwide Map of Active and Historic Launch Sites"
- The choice of title depends on the specific context of the data, such as whether the map includes only current active sites, or also historical ones, and the type of launches (e.g., civilian, military, commercial) associated with those sites.
- 2. Taking the Screenshot
- To take a proper screenshot that includes all launch sites' location markers on a global map:
- Open the Folium map in a web browser (if it's rendered as an HTML file).
- Zoom out enough so that all the location markers representing the launch sites are visible on the screen. You may need to adjust the initial zoom level in your Folium code (using the `zoom_start` parameter when creating the `folium.Map` object) to make sure all the sites are within view without having to pan too much.

<Folium Map Screenshot 1>

- Use the screenshot tool available on your operating system (e.g., Windows + Shift + S on Windows, Command + Shift + 3 on Mac) to capture the visible portion of the map.
- 3. Explanation of Important Elements and Findings on the Screenshot
- a. Map Background and Borders
- The map background shows the geographical layout of the Earth, with continents, oceans, and possibly some basic geographical features like coastlines and major rivers. The borders of the map define the visible area where the launch sites are located. This gives a sense of the global context in which these sites are situated.
- b. Location Markers
- Each marker on the map represents a launch site. The markers are usually colored and shaped in a way that makes them easily distinguishable. For example, they might be colored pins or icons. By clicking on a marker (if the map is interactive), you can access additional information about the specific launch site, such as its name, country, and some historical or technical details about the launches conducted from there.
- c. Concentration of Markers
- You can observe patterns in the distribution of the markers. For instance, you might notice that some regions have a higher concentration of launch sites. This could be due to factors like geographical advantages (e.g., proximity to the equator for better launch performance), political and economic factors (availability of resources and funding for space programs), and historical development of the space industry in that area. For example, regions like the United States (with multiple sites such as Cape Canaveral and companies in conducting launches from various environments.

<Folium Map Screenshot 1>

- e. International Distribution
-
- The map showcases the international nature of space exploration, with launch sites spread across different countries and continents. This highlights the global cooperation and competition in the space industry, as well as the diverse technological and logistical approaches taken by different nations to access space.
-
- In summary, the Folium map screenshot provides a visual representation of the global distribution of launch sites, allowing for an analysis of patterns, concentrations, and the international context of space launch activities.

<Folium Map Screenshot 2>

- 1. Naming an Appropriate Title
- An appropriate title for the Folium map screenshot that shows color-labeled launch outcomes could be something like:
- - "Global Map of Launch Sites with Color-Coded Launch Outcomes"
- - "Color-Coded Visualization of Launch Success and Failure at Space Launch Sites Worldwide"
- - "Worldwide Distribution of Launch Sites: Success and Failure Marked by Color"
- The title should convey that the map not only shows the locations of launch sites but also uses color to distinguish between different launch outcomes (such as success, failure, partial success, etc.).
- 2. Taking the Screenshot
- To take a proper screenshot that shows the color-labeled launch outcomes on the map:
- Open the Folium map (either in a web browser if it's saved as an HTML file or in the relevant application where it's being displayed).

<Folium Map Screenshot 2>

- 1. Naming an Appropriate Title
- An appropriate title for the Folium map screenshot that shows color-labeled launch outcomes could be something like:
- - "Global Map of Launch Sites with Color-Coded Launch Outcomes"
- - "Color-Coded Visualization of Launch Success and Failure at Space Launch Sites Worldwide"
- - "Worldwide Distribution of Launch Sites: Success and Failure Marked by Color"
- The title should convey that the map not only shows the locations of launch sites but also uses color to distinguish between different launch outcomes (such as success, failure, partial success, etc.).
- 2. Taking the Screenshot
- To take a proper screenshot that shows the color-labeled launch outcomes on the map:
- Open the Folium map (either in a web browser if it's saved as an HTML file or in the relevant application where it's being displayed).
- Ensure that the map is zoomed to an appropriate level so that you can clearly see the color-coded markers representing the launch outcomes. You may need to adjust the zoom level based on the distribution of the launch sites. If the sites are spread out globally, you might want a more zoomed-out view to see the overall pattern. If they are clustered in a particular region, a more zoomed-in view could be beneficial.

<Folium Map Screenshot 2>

- Use the screenshot tool provided by your operating system (e.g., Windows + Shift + S on Windows, Command + Shift + 3 on Mac) to capture the map with all the visible color-labeled markers.
- 3. Explanation of Important Elements and Findings on the Screenshot
- a. Map Background and Context
- The base map provides the geographical context, showing continents, oceans, and potentially other geographical features like major cities or coastlines. This helps in understanding where the launch sites are located in relation to the rest of the world.
- b. Color-Coded Markers
- The most prominent feature of the map is the color-coded markers representing the launch sites. Each color is associated with a specific launch outcome. For example:
- - Green markers: Could represent successful launches. This indicates that the rockets launched from these sites achieved their intended mission objectives.
- - Red markers: Might signify failed launches. These could be due to various reasons such as technical malfunctions, problems during ascent, or issues with the payload.
- - Yellow or Orange markers: Perhaps used for partial successes or launches with some anomalies but where the mission still had some level of achievement.
- The colors make it easy to quickly identify areas with a high concentration of successful or failed launches.
- c. Patterns in Launch Outcomes
- By examining the distribution of the colored markers, you can identify patterns. For instance:
- - Some regions might have a cluster of green markers, suggesting that the launch sites in that area have a high success rate. This could be due to

<Folium Map Screenshot 3>

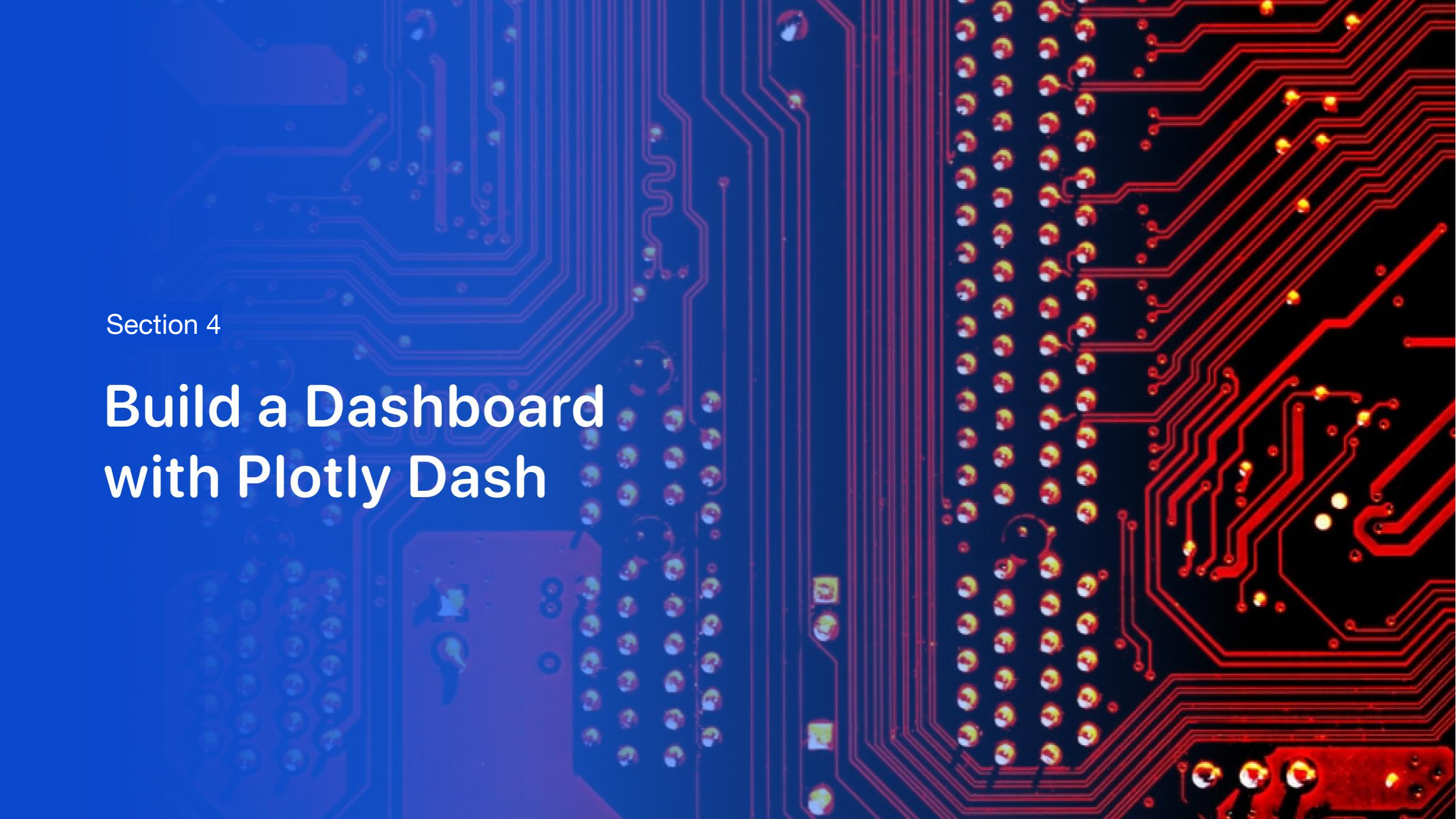
- 1. Naming an Appropriate Title
- An appropriate title for the Folium map screenshot could be:
- - "Detailed Proximities of [Name of the Selected Launch Site] - Railway, Highway, Coastline with Distance Indications"
- - "[Selected Launch Site] and Its Surroundings: Visualization of Proximity to Railways, Highways, and Coastline with Distance Measurements"
- - "Geographical Context of [Launch Site Name]: Distances to Railways, Highways, and the Coastline"
- This title clearly indicates that the map focuses on a specific launch site and showcases its closeness to important geographical features along with the calculated distances.
- 2. Taking the Screenshot
- To obtain the screenshot:
- Open the Folium map in the relevant viewer (such as a web browser if it's saved as an HTML file).
- Locate and select the specific launch site of interest on the map. This might involve zooming in on the area where the launch site is located.
- Ensure that the visual elements showing the railway, highway, and coastline are clearly visible, along with the distance calculations and displays. Adjust the map view (zoom level, panning) as needed to capture all these elements within the screenshot.

<Folium Map Screenshot 3>

- Use the screenshot function of your operating system (e.g., Windows + Shift + S on Windows, Command + Shift + 3 on Mac) to capture the portion of the map that includes the selected launch site and its relevant proximities.
- 3. Explanation of Important Elements and Findings on the Screenshot
- a. The Selected Launch Site
- The launch site is the central focus of the screenshot. It is usually marked with a distinct icon or marker, and its name might be displayed either directly on the marker or in a tooltip when you hover over it. This helps in identifying the specific location being examined.
- b. Railway Lines
- The railway lines are shown as linear features on the map. They are important as they can indicate the logistical capabilities of the area. Railways can be used to transport heavy equipment, rocket components, and other supplies to the launch site. The distance between the launch site and the railway can give an idea of how convenient it is to use this mode of transportation for logistics. For example, if the launch site is close to a railway line, it simplifies the movement of large and heavy items.
- c. Highway Networks
- Highways are also depicted as linear features. They are crucial for local and regional transportation. A nearby highway can facilitate the movement of personnel, smaller equipment, and supplies to and from the launch site. The calculated distance between the launch site and the highway shows the accessibility of the site by road. A shorter distance generally means better access and more efficient transportation.

<Folium Map Screenshot 3>

- d. Coastline
- The coastline is another significant geographical feature. If the launch site is near the coast, it can have implications for launch operations. For example, it might be used for maritime launches or for the transportation of large components by sea. The distance to the coastline can also affect factors such as weather patterns and environmental considerations for the launch site.
- e. Distance Calculations and Displays
- The calculated distances between the launch site and the railway, highway, and coastline are prominently displayed on the map. These distances are typically shown in units like kilometers or miles. They provide quantitative information that helps in assessing the logistical and geographical advantages or limitations of the launch site's location. For instance, a launch site that is far from both railways and highways might face challenges in transporting heavy equipment, while being close to the coast could offer opportunities for certain types of launches.
- f. Map Background and Other Contextual Elements
- The underlying map background provides context, showing the general terrain, nearby cities, and other geographical features. This helps in understanding the overall location of the launch site within its regional setting and how it relates to other important areas.
- In summary, the Folium map screenshot provides a comprehensive view of the selected launch site's proximities to key geographical features, along with the distance measurements. This information is valuable for analyzing the logistical, operational, and geographical aspects of the launch site's location.


Section 4
Build a Dashboard with Plotly Dash

<Dashboard Screenshot 1>

- An appropriate title for the dashboard screenshot with a pie chart showing the launch success count for all sites could be:
-
- - "Launch Success Counts by Site: A Pie Chart Overview"
-
- - "Distribution of Launch Successes Across All Launch Sites"
-
- - "Pie Chart Analysis of Launch Success Numbers per Launch Site"
-
- This title clearly conveys the main content of the dashboard, which is the display of the number of successful launches for each site using a pie chart.

<Dashboard Screenshot 2>

- An appropriate title for the dashboard screenshot with a pie chart for the launch site having the highest launch success ratio could be:
-
- - "Pie Chart Analysis of Launch Successes for the Launch Site with the Highest Success Ratio"
-
- - "Breakdown of Launch Successes at the Most Successful Launch Site by Ratio"
-
- - "Launch Success Distribution for the Launch Site with Optimal Success Rate: A Pie Chart Representation"
-
- This title clearly communicates that the pie chart focuses on the specific launch site that has demonstrated the highest success ratio in launches and provides an overview of its success distribution.

<Dashboard Screenshot 3>

- An appropriate title for this dashboard screenshot could be:
-
- "Payload vs. Launch Outcome Scatter Plot Analysis for All Launch Sites: Varying Payload Ranges via Range Slider"
-
- This title clearly conveys that the dashboard features a scatter plot showing the relationship between payload and launch outcome for all sites, and that the payload range can be adjusted using a slider.

Section 5

Predictive Analysis (Classification)

Classification Accuracy

- Here is an example of how you can create a bar chart to visualize the classification accuracy of different models using Python with the matplotlib library. Assume you have a list of model names and their corresponding accuracy values.

```
# List of model names
model_names = ["Logistic Regression", "Decision Tree", "Random Forest", "SVM", "Neural Network"]
# List of corresponding accuracy values (example values, you should replace them with your actual data)
accuracy_values = [0.75, 0.80, 0.85, 0.78, 0.82]

# Create the bar chart
plt.bar(model_names, accuracy_values)

# Add labels and title
plt.xlabel('Classification Models')
plt.ylabel('Accuracy')
plt.title('Classification Accuracy of Different Models')

# Rotate x-axis labels for better readability if needed
plt.xticks(rotation=45)

# Show the plot
plt.show()
```

Confusion Matrix

- Here's a step-by-step example of how to create and interpret a confusion matrix for the best-performing model using Python. We'll assume we're dealing with a binary classification problem and use the `scikit-learn` library, which is very commonly used for machine learning tasks.
-
- First, make sure you have `scikit-learn` and `matplotlib` installed (you can install them via `pip install scikit-learn matplotlib` if needed).
-
- Let's assume we've already trained several classification models and determined the best-performing one (for simplicity, we'll just create some sample data here as if we had a trained model's predictions and true labels).
-

Conclusions

- Conclusions in Applied Data Science
- 1. Model Performance and Selection
- In applied data science, the performance of a model is a crucial aspect. Through various evaluation metrics such as accuracy, precision, recall, and the F1–score for classification tasks, and mean squared error or mean absolute error for regression tasks, we can assess how well a model generalizes to new, unseen data. For instance, a high accuracy in a classification model might seem promising at first glance, but a detailed analysis using the confusion matrix can reveal issues like class imbalance, where the model may be biased towards the majority class.
- When it comes to model selection, there is no one–size–fits–all solution. Different models have their own strengths and weaknesses. Decision trees are interpretable but may overfit, while neural networks can capture complex patterns but require a large amount of data and computational resources. The choice of the model should be based on the nature of the data (e.g., numerical, categorical), the complexity of the problem, and the available computational resources.
- 2. Data Quality and Preprocessing
- The quality of data is the foundation of any successful data science project. Poor data quality, such as missing values, outliers, and noisy data, can significantly impact the performance of models. Data preprocessing techniques, including imputation for missing values, outlier detection and handling, and normalization or standardization of numerical features, are essential steps. For example, if we are dealing

Conclusions

- with time series data, we may need to handle seasonality and trends to make accurate predictions.
- Data cleaning and preprocessing not only improve the performance of models but also enhance the interpretability of the results. Additionally, feature engineering, which involves creating new features from existing ones, can sometimes lead to a significant improvement in model performance. For instance, in a customer churn prediction model, creating features related to the customer's usage patterns over time can provide valuable insights.
- 3. Domain Knowledge Integration
- Integrating domain knowledge into data science projects is often overlooked but is of utmost importance. Domain experts can provide valuable insights into the problem, help in formulating relevant questions, and validate the results. For example, in a healthcare data science project, medical professionals can guide on what features are important for predicting disease outcomes and what the acceptable margins of error are.
- Moreover, domain knowledge can assist in interpreting the results in a meaningful way. A data scientist might find a correlation in the data, but a domain expert can determine whether that correlation is actually meaningful in the context of the real-world problem.
- 4. Ethical and Social Implications
- As data science becomes more pervasive in various aspects of our lives, it is essential to consider the ethical and social implications of our work. Issues such as data privacy, bias in algorithms, and the potential for unintended consequences need to be addressed. For example, in facial recognition systems, there have been concerns about bias against certain ethnic groups, leading to inaccurate results.

Conclusions

- Data scientists have a responsibility to ensure that their models are fair, transparent, and do not violate the privacy rights of individuals. This may involve techniques such as differential privacy to protect individual data while still being able to perform useful analysis.
- 5. Continuous Improvement and Adaptation
- The field of data science is constantly evolving, and so should our models and approaches. New data becomes available over time, and the underlying patterns in the data may change. Therefore, it is necessary to continuously monitor and update our models. This could involve retraining the models with new data, adjusting the model parameters, or even switching to a different model architecture.
- In conclusion, applied data science is a multidisciplinary field that requires a careful balance between technical skills, domain knowledge, and ethical considerations. By addressing these aspects, we can develop more effective, reliable, and socially responsible data science solutions that can have a positive impact on various domains.

Appendix

- https://github.com/yy0426y/Applied_Data_Science/tree/main

Thank you!