**Project Title:** Exploratory Data Analysis (EDA) in Python

**1. Dataset Selection**

* You need to choose a dataset that interests you and is suitable for EDA.
* It should be from a public source like Kaggle or the UCI Machine Learning Repository.
* The dataset should have at least 500 rows and 5 columns.

**Example Dataset:**

For this example, let's use the "Titanic - Machine Learning from Disaster" dataset from Kaggle. It's a classic dataset for learning data analysis and has a good mix of categorical and numerical features.

* You can download it here: <https://www.kaggle.com/c/titanic/data>

**2. Project Setup**

* Create a new directory for your project.
* Inside the directory, create a Jupyter Notebook named EDA\_Project\_YourName.ipynb (replace "YourName" with your actual name).
* Also, create a README.md file in the project directory.

**README.md Content Example:**

# EDA Project: Titanic Dataset Analysis

This project performs Exploratory Data Analysis (EDA) on the Titanic dataset.

\*\*Steps:\*\*

1. Data Import and Cleaning

2. Exploratory Data Analysis

\* Descriptive Statistics

\* Data Visualization

\* Group Analysis

\* Feature Analysis

3. Advanced Python Techniques

4. Insights and Conclusions

**3. Data Import and Cleaning**

Jupyter Notebook (EDA\_Project\_YourName.ipynb)

**3.1 Import Libraries**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

* Import the necessary libraries: pandas, numpy, matplotlib, and seaborn.

**3.2 Load the Dataset**

# Load the training dataset

df = pd.read\_csv('train.csv') # Assuming your CSV file is named "train.csv"

* Load the dataset into a pandas DataFrame.

**3.3 Initial Data Inspection**

# Display the first 5 rows

print(df.head())

# Get information about the DataFrame (columns, data types, non-null counts)

print(df.info())

# Get summary statistics for numerical columns

print(df.describe())

# Get summary statistics for categorical columns

print(df.describe(include=['O'])) # 'O' stands for 'Object' type, which is often used for strings

* Check the shape, data types, and summary statistics of the data.

**3.4 Handle Missing Values**

# Check for missing values

print(df.isnull().sum())

# Handle missing values (example: for Age, fill with median; for Cabin, replace with 'Unknown'; for Embarked, fill with mode)

df['Age'].fillna(df['Age'].median(), inplace=True)

df['Cabin'].fillna('Unknown', inplace=True)

df['Embarked'].fillna(df['Embarked'].mode()[0], inplace=True)

# Verify that missing values are handled

print(df.isnull().sum())

* Identify missing values.
* Decide how to handle them (fill, drop, etc.).

**3.5 Detect and Remove Duplicates**

# Check for duplicates

print("Number of duplicate rows:", df.duplicated().sum())

# Remove duplicates (if any)

df.drop\_duplicates(inplace=True)

# Verify removal of duplicates

print("Number of duplicate rows after removal:", df.duplicated().sum())

* Detect and remove duplicate rows.

**3.6 Convert Data Types**

# Convert 'PassengerId' to string (it's an identifier, not a numeric value)

df['PassengerId'] = df['PassengerId'].astype(str)

# Convert 'Survived' to categorical

df['Survived'] = df['Survived'].astype('category')

* Convert data types if necessary.

**4. Exploratory Data Analysis (EDA)**

**4.1 Descriptive Statistics**

# Summary statistics for numerical columns

print("Numerical Data Description:")

print(df.describe())

# Summary statistics for categorical columns

print("\nCategorical Data Description:")

print(df.describe(include=['O']))

* Provide summary statistics for numerical and categorical columns.

**4.2 Data Visualization**

# Histograms for numerical features

df['Age'].hist(bins=20)

plt.title('Age Distribution')

plt.xlabel('Age')

plt.ylabel('Frequency')

plt.show()

df['Fare'].hist(bins=20)

plt.title('Fare Distribution')

plt.xlabel('Fare')

plt.ylabel('Frequency')

plt.show()

# Bar plots for categorical features

df['Survived'].value\_counts().plot(kind='bar')

plt.title('Survival Count')

plt.xlabel('Survived (0 = No, 1 = Yes)')

plt.ylabel('Count')

plt.show()

df['Pclass'].value\_counts().sort\_index().plot(kind='bar')

plt.title('Passenger Class Distribution')

plt.xlabel('Passenger Class')

plt.ylabel('Count')

plt.show()

# Box plots to identify outliers

sns.boxplot(x='Pclass', y='Age', data=df)

plt.title('Box Plot of Age by Passenger Class')

plt.show()

sns.boxplot(x='Survived', y='Fare', data=df)

plt.title('Box Plot of Fare by Survival')

plt.show()

# Scatter plots for relationships between numerical features

plt.scatter(df['Age'], df['Fare'])

plt.title('Scatter Plot of Age vs. Fare')

plt.xlabel('Age')

plt.ylabel('Fare')

plt.show()

# Heatmap to visualize correlations

correlation\_matrix = df.corr()

sns.heatmap(correlation\_matrix, annot=True, cmap='coolwarm')

plt.title('Correlation Heatmap')

plt.show()

* Create histograms or density plots for numerical features.
* Create bar plots for categorical features.
* Use box plots to identify outliers.
* Create scatter plots to explore relationships between numerical features.
* Use heatmaps to visualize correlations between numerical features.

**4.3 Group Analysis**

# Group by 'Pclass' and get survival rate

survival\_by\_pclass = df.groupby('Pclass')['Survived'].value\_counts(normalize=True) \* 100

print("Survival Rate by Passenger Class:")

print(survival\_by\_pclass)

# Group by 'Sex' and get survival rate

survival\_by\_sex = df.groupby('Sex')['Survived'].value\_counts(normalize=True) \* 100

print("\nSurvival Rate by Sex:")

print(survival\_by\_sex)

* Perform group-by operations to aggregate data based on categorical features.

**4.4 Feature Analysis**

# Analyze 'Age' and 'Survived'

age\_survival = pd.cut(df['Age'], bins=[0, 10, 20, 30, 40, 50, 60, 70, 80])

print("\nSurvival Rate by Age Group:")

print(df.groupby(age\_survival)['Survived'].value\_counts(normalize=True) \* 100)

# Analyze 'Fare' and 'Survived'

fare\_survival = pd.qcut(df['Fare'], q=4) # Divide fare into quartiles

print("\nSurvival Rate by Fare Quartile:")

print(df.groupby(fare\_survival)['Survived'].value\_counts(normalize=True) \* 100)

# Pair plots for feature relationships

sns.pairplot(df[['Age', 'Fare', 'Pclass', 'Survived', 'Sex']])

plt.show()

# Correlation matrix

correlation\_matrix = df[['Age', 'Fare', 'Pclass', 'Survived']].corr()

print("\nCorrelation Matrix:")

print(correlation\_matrix)

# Pivot table to analyze survival rate by Sex and Pclass

pivot\_table = df.pivot\_table(index='Sex', columns='Pclass', values='Survived', aggfunc='mean')

print("\nPivot Table (Survival Rate by Sex and Pclass):")

print(pivot\_table)

* Identify and analyze key features.
* Explore relationships between features using pair plots, correlation matrices, and pivot tables.

**5. Advanced Python Techniques**

**5.1 Lambda Functions**

# Create a new column 'Age\_Group' using a lambda function

df['Age\_Group'] = df['Age'].apply(lambda x: 'Child' if x <= 12 else ('Teenager' if x <= 19 else 'Adult'))

print("\nAge Group Counts:")

print(df['Age\_Group'].value\_counts())

* Use a lambda function for data transformation.

**5.2 User-Defined Functions**

# Define a function to categorize Fare into price ranges

def categorize\_fare(fare):

if fare <= 10:

return 'Low'

elif fare <= 50:

return 'Medium'

else:

return 'High'

df['Fare\_Category'] = df['Fare'].apply(categorize\_fare)

print("\nFare Category Counts:")

print(df['Fare\_Category'].value\_counts())

* Write a custom function for a complex calculation.

**5.3 List Comprehensions**

# Get a list of columns with missing values

columns\_with\_missing\_values = [col for col in df.columns if df[col].isnull().any()]

print("\nColumns with Missing Values:", columns\_with\_missing\_values) # Should be an empty list now if you've handled missing values

* Use list comprehensions for efficient data processing.

**6. Insights and Conclusions**

# Summarize key findings

print("\n--- Key Findings ---")

print("\nSurvival Rate Overview:")

print(df['Survived'].value\_counts(normalize=True) \* 100)

print("\nSurvival Rate by Sex:")

print(df.groupby('Sex')['Survived'].mean() \* 100)

print("\nSurvival Rate by Pclass:")

print(df.groupby('Pclass')['Survived'].mean() \* 100)

print("\nAge Distribution Insights:")

print(df['Age'].describe())

print("\nFare Distribution Insights:")

print(df['Fare'].describe())

print("\nCorrelation Analysis:")

print(df[['Age', 'Fare', 'Pclass', 'Survived']].corr())

# Discuss patterns, anomalies, and relationships

print("\n--- Insights ---")

print("- Women had a significantly higher survival rate than men.")

print("- Passengers in higher classes (Pclass 1) had a higher survival rate.")

print("- Age distribution was right-skewed, with most passengers being young adults.")

print("- Fare distribution was also right-skewed, with most passengers paying lower fares.")

print("- There was a weak negative correlation between Pclass and Survival, and a weak positive correlation between Fare and Survival.")

# Highlight potential areas for further analysis

print("\n--- Further Analysis ---")

print("- Investigate the interaction between Age, Sex, and Pclass in more detail.")

print("- Analyze the impact of family size (SibSp and Parch) on survival.")

print("- Explore other features like Cabin and Embarked in more depth.")

* Summarize your key findings.
* Discuss patterns, anomalies, and relationships.
* Highlight potential areas for further analysis.

**7. Documentation and Presentation**

* Ensure your Jupyter Notebook is well-documented with markdown cells.
* Visualizations should have clear titles, axis labels, and legends.
* Prepare a brief presentation (5-10 slides) summarizing your project.

**Presentation Outline (5-10 slides):**

1. **Title Slide:** Project Title, Your Name, Date
2. **Introduction:**
   * Dataset Description (Titanic dataset)
   * Project Objective (EDA to understand passenger survival)
3. **Data Cleaning:**
   * Summary of cleaning steps (missing value handling, duplicates)
4. **Exploratory Data Analysis:**
   * Key visualizations (histograms, bar plots, scatter plots, heatmaps)
   * Descriptive statistics summary
5. **Group Analysis:**
   * Survival rates by Sex and Pclass
6. **Feature Analysis:**
   * Important features and their impact on survival
7. **Advanced Python Techniques:**
   * Examples of lambda functions, user-defined functions, and list comprehensions
8. **Insights and Conclusions:**
   * Summary of key findings and insights
9. **Further Analysis:**
   * Potential areas for future exploration

**Submission**

* Submit your Jupyter Notebook (EDA\_Project\_YourName.ipynb)
* Submit the dataset file (e.g., train.csv)
* Submit your README file (README.md)
* Submit your presentation slides.

**1. Load the Dataset and Initial Inspection**

**Objective:** Load the dataset and get a first look at its structure and content.

import pandas as pd  
import numpy as np  
  
# Load the dataset  
df = pd.read\_csv('heart.csv')  # Replace 'heart.csv' with the actual path to your file  
  
# Display the first 5 rows  
print(df.head())  
  
# Get information about the DataFrame  
print(df.info())  
  
# Get summary statistics  
print(df.describe())

**Explanation:**

* **Import Libraries:** Import pandas for data manipulation and numpy for numerical operations.
* **Load Dataset:** Use pd.read\_csv() to load the data into a DataFrame. **Important:** Make sure the file 'heart.csv' is in the same directory as your Python script or Jupyter Notebook, or provide the correct path to the file.
* **df.head()**: Displays the first 5 rows of the DataFrame, giving you a quick look at the data.
* **df.info()**: Provides information about the DataFrame, including the column names, data types, and the number of non-null values in each column. This is crucial for identifying potential data issues.
* **df.describe()**: Generates descriptive statistics for numerical columns, such as mean, standard deviation, minimum, and maximum values. This helps in understanding the distribution and range of the data.

**2. Handling Missing Values**

**Objective:** Identify and handle any missing data in the dataset.

# Check for missing values  
print(df.isnull().sum())  
  
# Handle missing values (if any)  
# For this dataset, let's assume there are no missing values based on initial inspection.  If there were:  
# df['column\_name'].fillna(df['column\_name'].mean(), inplace=True)  # Example: Fill with mean  
# df.dropna(inplace=True)  # Example: Drop rows with any missing values

**Explanation:**

* **df.isnull().sum()**: Calculates the number of missing values in each column.
* **Handle Missing Values:**
* If there are missing values, you'll need to decide how to handle them. Common strategies include:
* Filling with the mean, median, or mode (for numerical columns).
* Filling with a constant value (e.g., 'Unknown' for categorical columns).
* Dropping rows with missing values.
* Interpolation (for time series data).
* The choice depends on the nature of the data and the specific column. *In this specific dataset, a preliminary check shows no missing values, so the handling part is not needed, but I've left the code as a general example.*

**3. Handling Duplicates**

**Objective:** Detect and remove any duplicate rows in the dataset.

# Check for duplicates  
print("Number of duplicate rows:", df.duplicated().sum())  
  
# Remove duplicates (if any)  
df.drop\_duplicates(inplace=True)  
  
# Verify removal of duplicates  
print("Number of duplicate rows after removal:", df.duplicated().sum())

**Explanation:**

* **df.duplicated().sum()**: Counts the number of duplicate rows.
* **df.drop\_duplicates(inplace=True)**: Removes duplicate rows from the DataFrame. The inplace=True argument modifies the DataFrame directly.

**4. Data Type Conversion**

**Objective:** Ensure that the columns have the correct data types.

# Convert data types if necessary  
# print(df.info()) # Use this to check the data types  
# Example (if needed):  
# df['date\_column'] = pd.to\_datetime(df['date\_column'])  
# df['categorical\_column'] = df['categorical\_column'].astype('category')

**Explanation:**

* Use df.info() to check the data types of the columns.
* If a column's data type is incorrect, you can convert it using:
* pd.to\_datetime(): For converting strings to datetime objects.
* .astype(): For converting to other data types (e.g., int, float, category).
* In this dataset, you'll likely find that the data types are already appropriate, but it's good practice to check.

**5. Descriptive Statistics**

**Objective:** Calculate and display summary statistics for the columns.

import numpy as np  
# Summary statistics for numerical columns  
print("Numerical Columns Summary:")  
print(df.describe())  
  
# Summary statistics for categorical columns  
print("\nCategorical Columns Summary:")  
print(df.describe(include=['O'])) # Object type is often used for strings, which represent categorical data.  
  
# Additional numerical statistics  
print("\nMedians for numerical columns:")  
print(df.median(numeric\_only=True))  
  
print("\nSkewness for numerical columns:")  
print(df.skew(numeric\_only=True))  
  
print("\nKurtosis for numerical columns:")  
print(df.kurtosis(numeric\_only=True))

**Explanation:**

* **df.describe()**: Provides summary statistics for numerical columns (count, mean, std, min, max, quartiles).
* **df.describe(include=\['O'\])**: Provides summary statistics for categorical columns (count, unique, top, frequency).
* **df.median()**: Calculates the median for numerical columns.
* **df.skew()**: Calculates the skewness of the numerical columns.
* **df.kurtosis()**: Calculates the kurtosis of the numerical columns.

**6. Data Visualization**

**Objective:** Visualize the data to understand distributions and relationships.

import matplotlib.pyplot as plt  
import seaborn as sns  
  
# Histograms for numerical features  
df.hist(figsize=(15, 10))  # Adjust figure size as needed  
plt.suptitle("Histograms of Numerical Features", fontsize=16)  
plt.show()  
  
# Bar plots for categorical features  
categorical\_cols = [col for col in df.columns if df[col].dtype == 'object']  # Get categorical column names  
if categorical\_cols:  
    plt.figure(figsize=(15, 10))  
    for i, col in enumerate(categorical\_cols, 1):  
        plt.subplot(len(categorical\_cols), 1, i)  
        df[col].value\_counts().plot(kind='bar')  
        plt.title(f"Bar Plot of {col}")  
    plt.tight\_layout()  
    plt.show()  
  
# Box plots to identify outliers  
plt.figure(figsize=(15, 10))  
for i, col in enumerate(df.select\_dtypes(include=np.number).columns, 1): # Boxplots for numerical columns  
    plt.subplot(3, 5, i)  # Assuming a maximum of 15 numerical features  
    sns.boxplot(y=df[col])  
    plt.title(f"Box Plot of {col}")  
plt.tight\_layout()  
plt.show()  
  
# Scatter plots to explore relationships between numerical features  
sns.pairplot(df)  
plt.suptitle("Pair Plots of Numerical Features", fontsize=16)  
plt.show()  
  
# Heatmap to visualize correlations  
correlation\_matrix = df.corr()  
plt.figure(figsize=(12, 10))  
sns.heatmap(correlation\_matrix, annot=True, cmap='coolwarm')  
plt.title("Correlation Heatmap", fontsize=16)  
plt.show()

**Explanation:**

* **Import Libraries:** Import matplotlib.pyplot for plotting and seaborn for enhanced visualizations.
* **Histograms:** Show the distribution of numerical features.
* **Bar Plots:** Show the frequency of each category for categorical features.
* **Box Plots:** Visualize the spread and identify potential outliers in numerical data.
* **Pair Plots:** Show scatter plots for all pairs of numerical features, revealing relationships.
* **Heatmap:** Visualizes the correlation matrix, showing the linear relationships between numerical features.

**7. Group Analysis**

**Objective:** Aggregate data based on categorical features.

# Group-by operations  
print("\nMean age by target:")  
print(df.groupby('target')['age'].mean())  
  
print("\nCholesterol statistics by sex:")  
print(df.groupby('sex')['chol'].describe())

**Explanation:**

* **df.groupby('target')\['age'\].mean()**: Calculates the mean age for each target group (e.g., patients with and without heart disease).
* **df.groupby('sex')\['chol'\].describe()**: Provides descriptive statistics for cholesterol levels, grouped by sex.

**8. Feature Analysis**

**Objective:** Analyze the features in the dataset and their relationships.

# Feature Analysis  
# Example: Analyze the relationship between age and target  
plt.figure(figsize=(8, 6))  
sns.boxplot(x='target', y='age', data=df)  
plt.title("Box Plot of Age vs. Target", fontsize=14)  
plt.show()  
  
# Example: Analyze the relationship between chest pain type (cp) and target  
plt.figure(figsize=(8, 6))  
sns.countplot(x='cp', hue='target', data=df)  
plt.title("Count Plot of Chest Pain Type vs. Target", fontsize=14)  
plt.show()  
  
# Correlation analysis of features  
correlation\_matrix = df.corr()  
print(correlation\_matrix['target'].sort\_values(ascending=False))

**Explanation:**

* This part of the analysis is crucial as it directly relates to understanding which factors contribute to the heart condition.
* I've added a boxplot to visualize the distribution of age for those with and without heart disease.
* A count plot shows how different types of chest pain relate to the presence or absence of the target variable.
* The code calculates the correlation between each feature and the 'target' variable, helping to identify which features are most strongly related to the outcome.

**9. Predictions We Can Make**

Based on this EDA, we can start to make some predictions and formulate hypotheses:

* **Risk Factors:** We can identify which factors are most strongly associated with the target variable (heart disease). For example, if we see a high correlation between 'age' and 'target', we can hypothesize that older patients are more likely to have heart disease.
* **Feature Importance:** The correlation analysis helps in understanding which features are more important than others in predicting the target variable.
* **Group Differences:** Group analysis (e.g., grouping by 'sex') can reveal differences in the prevalence or characteristics of heart disease between different groups.
* **Outlier Impact:** Box plots help identify outliers, which might represent errors in the data or unusual cases that warrant further investigation.
* **Data Distributions:** Histograms and density plots help us understand the distribution of the data, which can inform decisions about data transformations or modeling techniques.
* **Relationships:** Scatter plots and pair plots can reveal relationships between different features, such as whether there is a correlation between cholesterol levels and age.

**Potential Predictions/Hypotheses:**

* Older age is associated with a higher risk of heart disease.
* Certain types of chest pain are more indicative of heart disease than others.
* Cholesterol levels may be a significant factor in predicting heart disease.
* There might be differences in heart disease prevalence or characteristics between males and females.

**Remember:** EDA is an iterative process. You might need to go back and perform additional analysis or data cleaning based on your initial findings.