Data and Artificial Intelligence
Cyber Shujaa Program

Week 3 Assignment
Exploratory Data Analysis

Student Name: Violet Joy

Student ID: CS-dao1-25025

Introduction

Exploratory Data Analysis (EDA) serves as an essential initial phase of data science projects because it enables the discovery of patterns while detecting anomalies and generating hypotheses through the summarization of key dataset features. This project aims to conduct EDA with the Titanic dataset which contains historical information about passengers on RMS Titanic including age, sex, class, fare and survival status.

This analysis aims to discover which factors played a role in determining survival during the Titanic disaster. I plan to analyze factors like passenger class (Pclass), gender, age, and embarkation point to discover patterns which will help me understand survival outcomes. The dataset includes categorical data alongside missing values which require thorough preprocessing and transformation steps.

The project requires loading, cleaning data and visualizing results through Pandas, Seaborn and Matplotlib tools. The project employs univariate and bivariate analysis methods to examine variable distributions and their correlations. I plan to use visual tools including histograms, bar plots, and heatmaps to facilitate clear and meaningful data interpretation.

Link: https://www.kaggle.com/code/mariyamalshatta/masterclass-1-a-comprehensive-guide-for-eda

The purpose of the assignment is to practice Exploratory Data Analysis steps:

1. Initial Data Exploration
2. Handling Missing Values
3. Univariate Analysis
4. Bivariate Analysis

5. Multivariate Analysis
6. Handling Outliers
7. Target Variable Analysis

Tasks Completed

DATA SCIENCE PROJECT: EXPLORATORY DATA ANALYSIS

This project aims at analysing the titanic dataset to summarize their main characteristics and how it behaves.

```
# Import libraries
import pandas as pd  # Data manipulation
import numpy as np  # Numerical computations
import matplotlib. pyplot as plt # Static plots
import seaborn as sns  # Statistical plots
import missingno as msno  # Missing data visualization

# Configuring Seaborn plot aesthetics
sns.set_theme(style='darkgrid', context='notebook')

import warnings
warnings.filterwarnings("ignore")
```

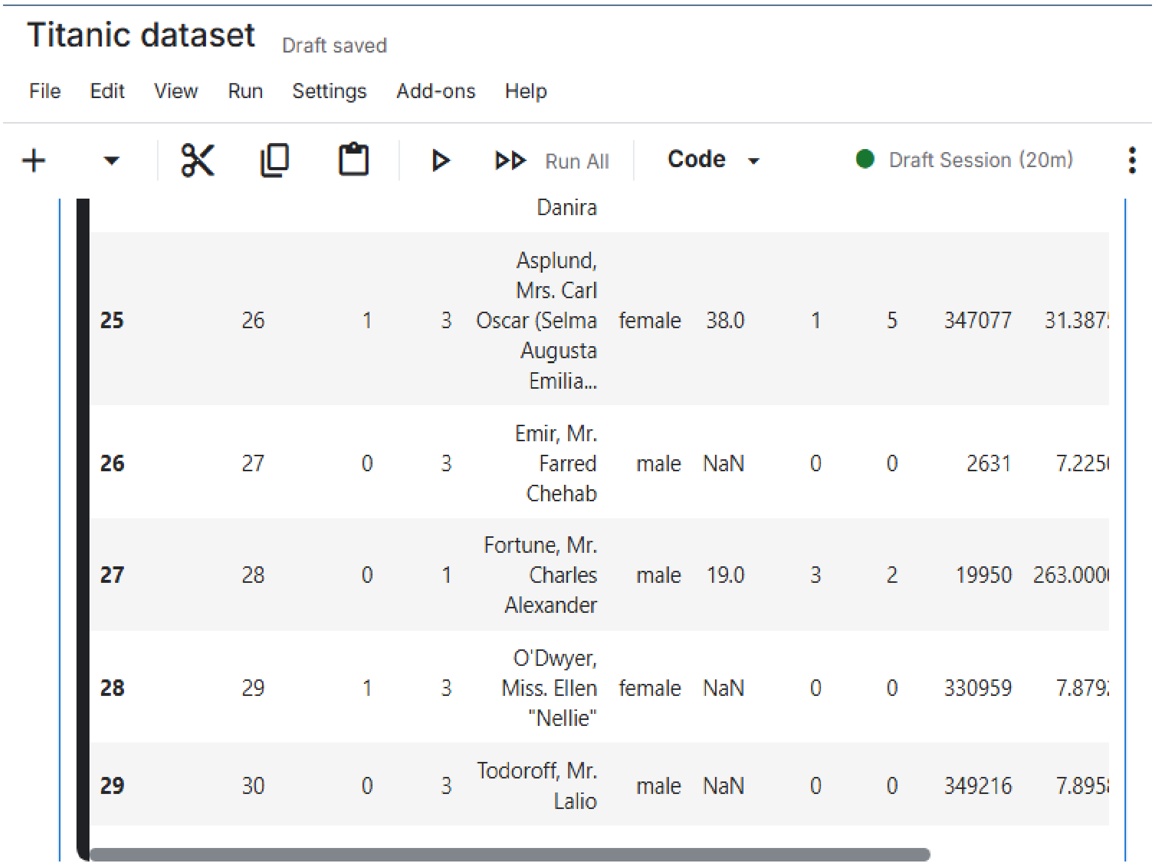
STEP 1: INITIAL DATA EXPLORATION

Here I am trying to understand my dataset and how it is structured.

```
# Load Data

train = pd.read_csv("/kaggle/input/titanic/train.csv")

test = pd.read_csv("/kaggle/input/titanic/test.csv")

gender = pd.read_csv("/kaggle/input/titanic/gender_submission.csv")
```

- **Displaying a few rows of the dataset**

```
# Preview the first 30 rows of the dataset

train.head(30)
```



Titanic dataset Draft saved

File Edit View Run Settings Add-ons Help

Run All Code Draft Session (20m)

				Danira						
25	26	1	3	Asplund, Mrs. Carl Oscar (Selma Augusta Emilia...	female	38.0	1	5	347077	31.387
26	27	0	3	Emir, Mr. Farred Chehab	male	NaN	0	0	2631	7.225
27	28	0	1	Fortune, Mr. Charles Alexander	male	19.0	3	2	19950	263.000
28	29	1	3	O'Dwyer, Miss. Ellen "Nellie"	female	NaN	0	0	330959	7.879
29	30	0	3	Todoroff, Mr. Lalio	male	NaN	0	0	349216	7.895

Fig 1.1; Output for displaying the first 30 rows of the titanic dataset

- **Checking the shape of the dataset**
  #number of rows and columns

  print(f'The dataset has {train.shape[0]} rows and {train.shape[1]} columns.')

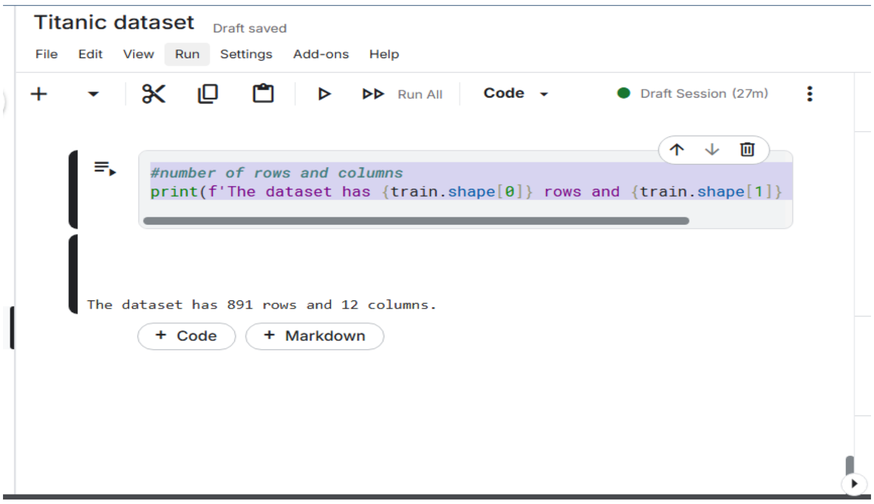
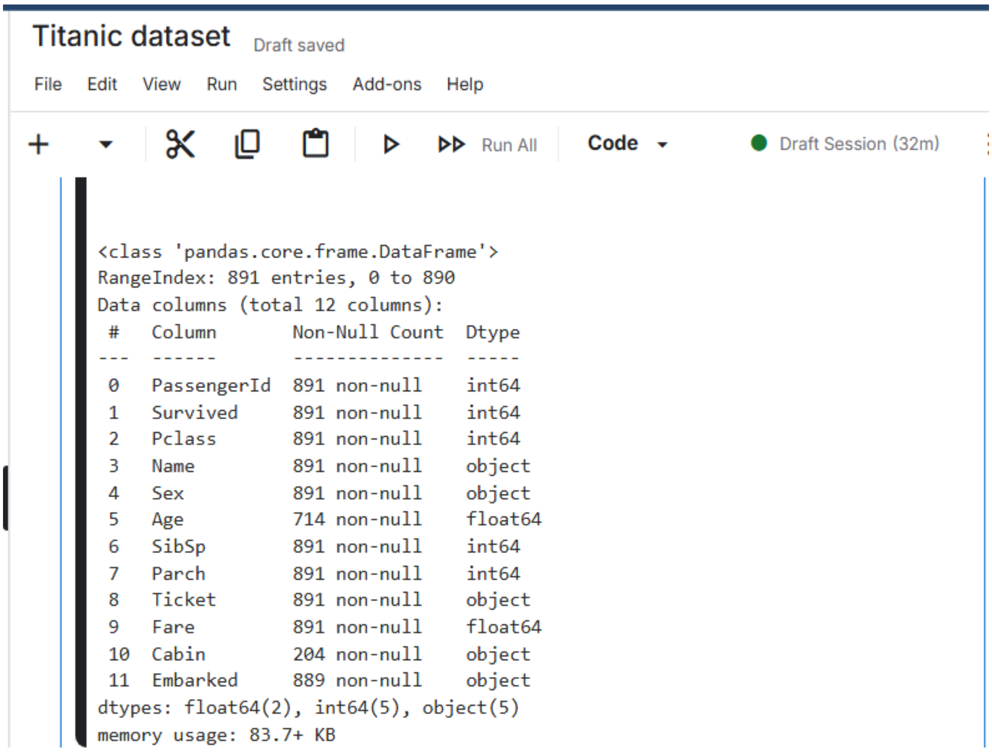


Fig 1.2; Output for the number of rows and columns

- **Details about the columns, their data types, and the number of non-null**
  #dataset's columns and their data types

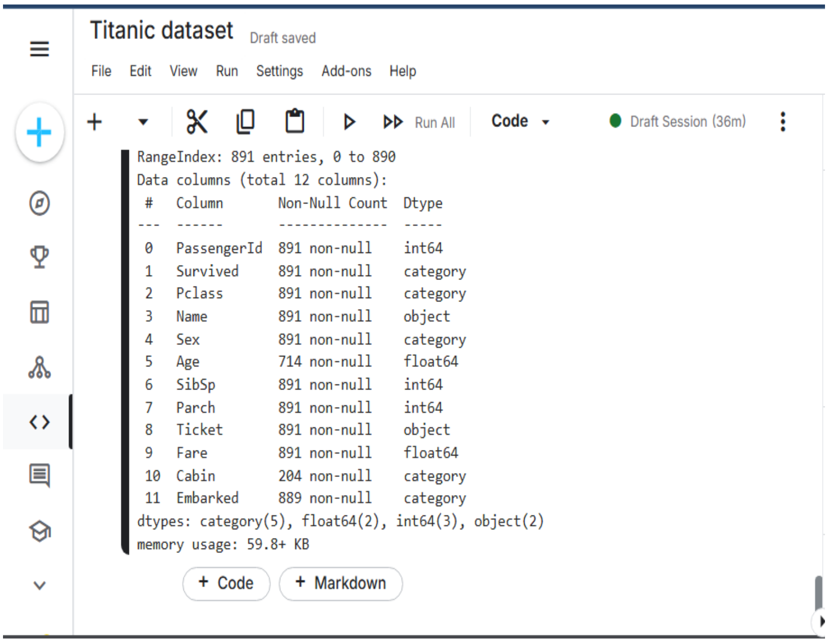
  train.info()




```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object
 11  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

Fig1.3; Output on the column and their data types

```
# Converting data types

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

train['Pclass'] = train['Pclass'].astype('category')

train['Sex'] = train['Sex'].astype('category')

train['Cabin'] = train['Cabin'].astype('category')

train['Embarked'] = train['Embarked'].astype('category')

train.info()
```

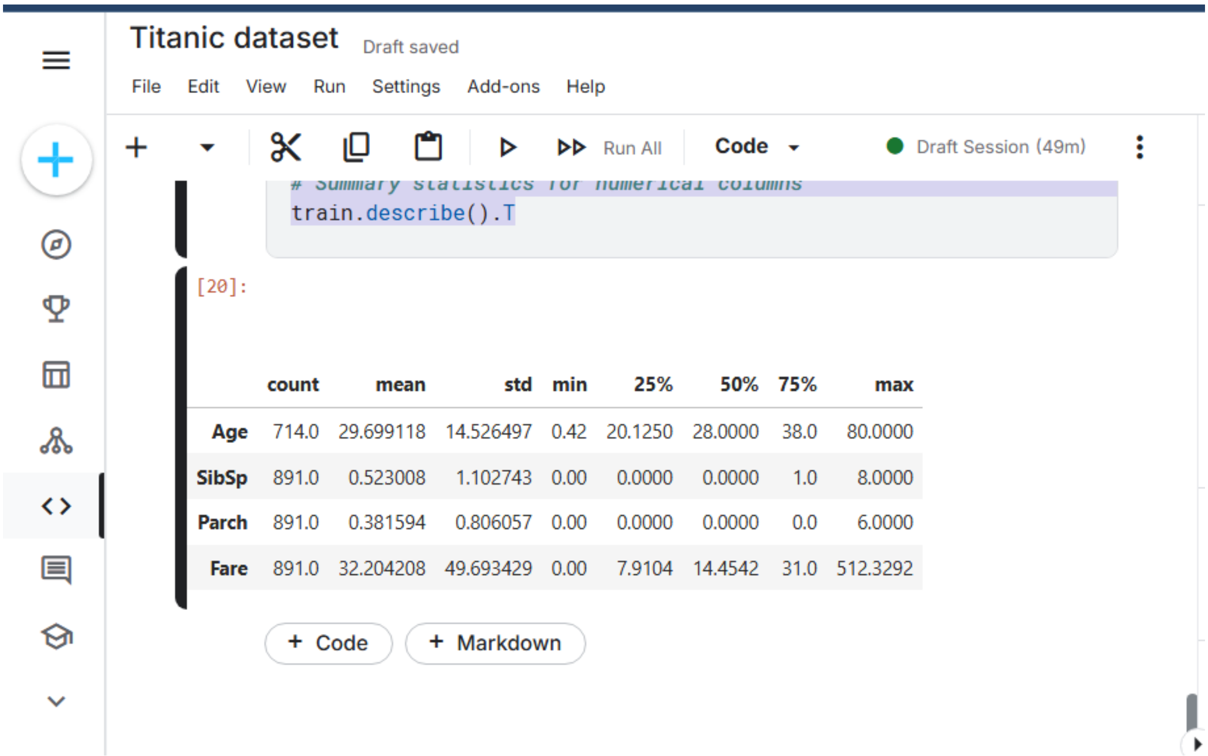



```
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    category
 2   Pclass       891 non-null    category
 3   Name         891 non-null    object
 4   Sex          891 non-null    category
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    category
 11  Embarked     889 non-null    category
dtypes: category(5), float64(2), int64(3), object(2)
memory usage: 59.8+ KB
```

Fig 1.4; Output on the corrected data types

- **Statistical summary of numerical columns**; This includes the count, mean, standard deviation, percentiles, minimum and maximum values
  # Summary statistics for numerical columns

  train. Describe().T



	count	mean	std	min	25%	50%	75%	max
Age	714.0	29.699118	14.526497	0.42	20.1250	28.0000	38.0	80.0000
SibSp	891.0	0.523008	1.102743	0.00	0.0000	0.0000	1.0	8.0000
Parch	891.0	0.381594	0.806057	0.00	0.0000	0.0000	0.0	6.0000
Fare	891.0	32.204208	49.693429	0.00	7.9104	14.4542	31.0	512.3292

Fig 1.5; Output on summary statistics

- **Listing column names**

List column names

train.columns

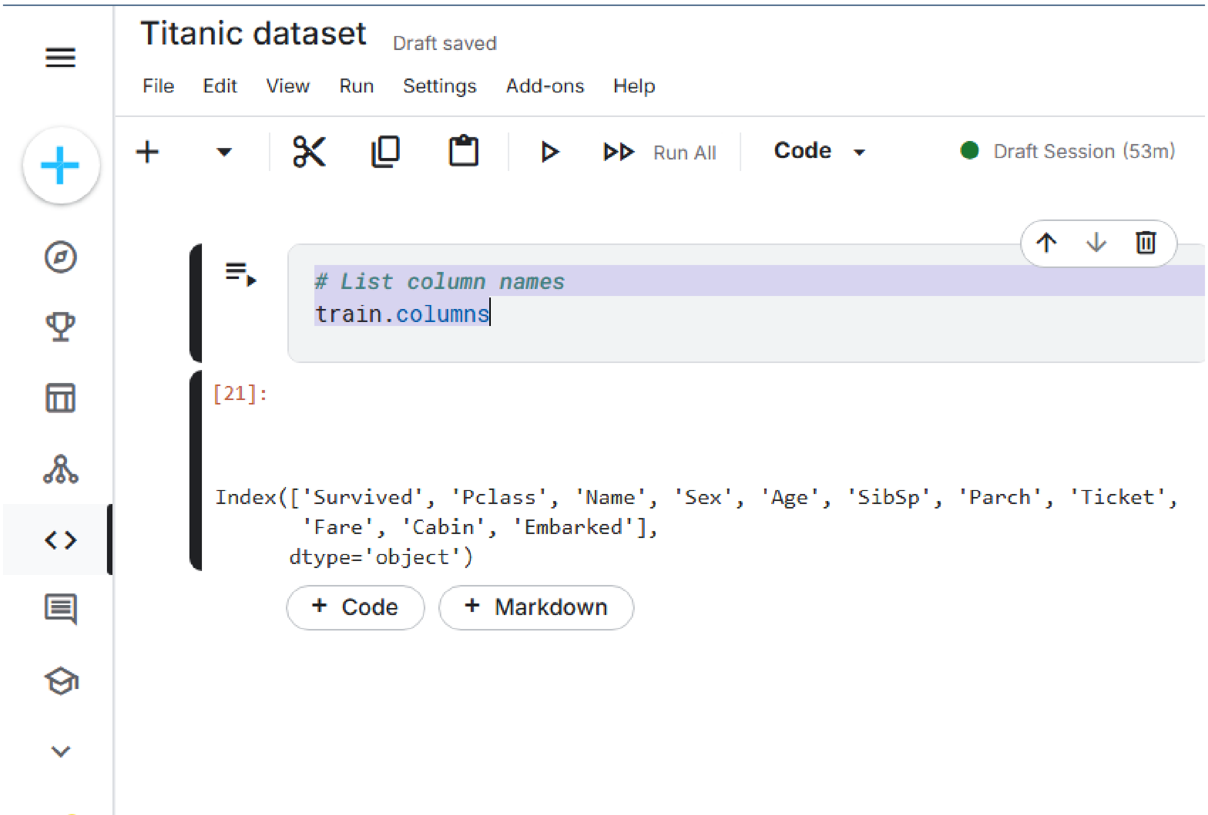


Fig 1.6; Output on column names

- **Checking for the number of unique values in each column**

 # Count the unique values in each column

train.nunique()

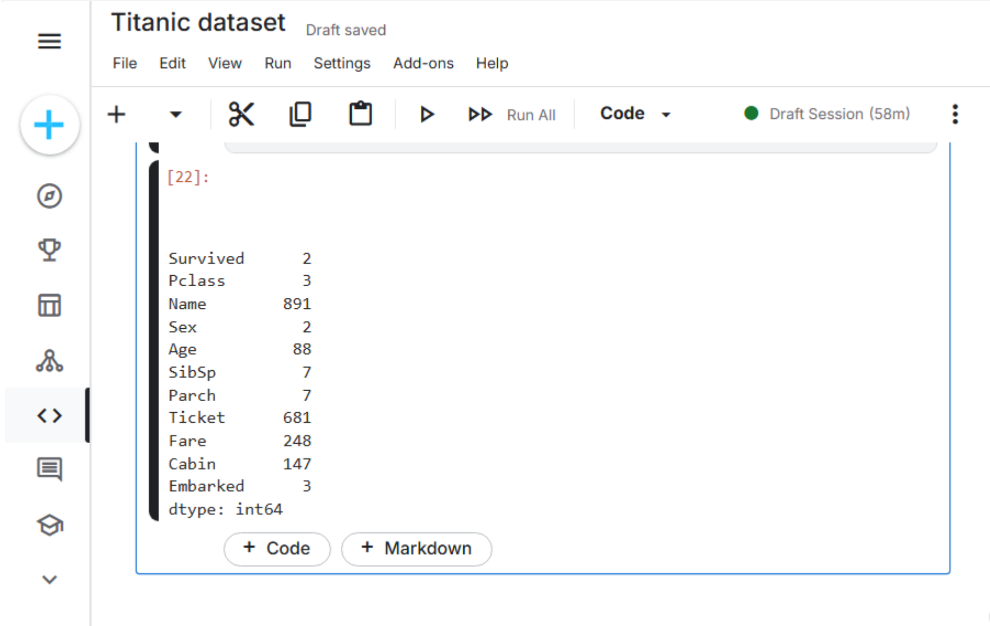


Fig 1.7; Output the number of unique values in each column

- **Checking for duplicates**

```
# check for duplicates
train.duplicated().sum()
```

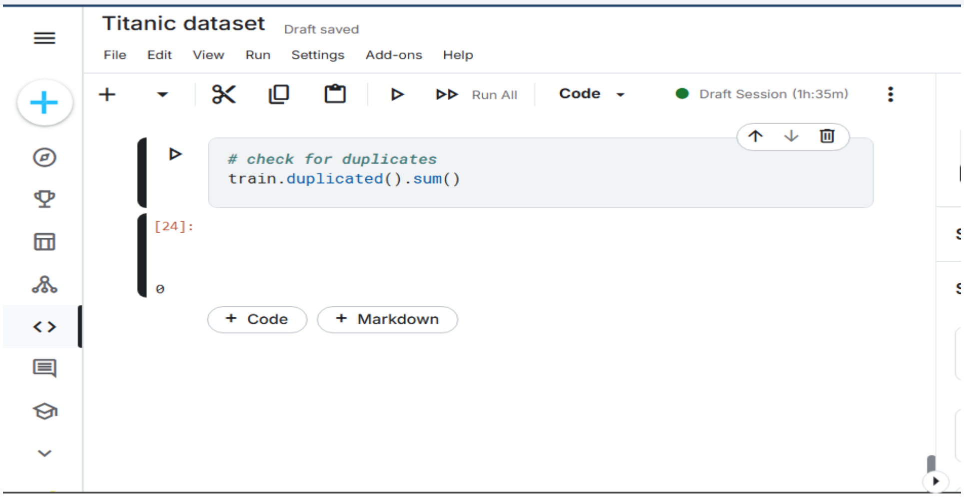


Fig 1.8; Output for checking duplicates

STEP 2: HANDLING MISSING VALUES

Visualize missing data using missingno library

import missingno as msno

msno.bar(train)

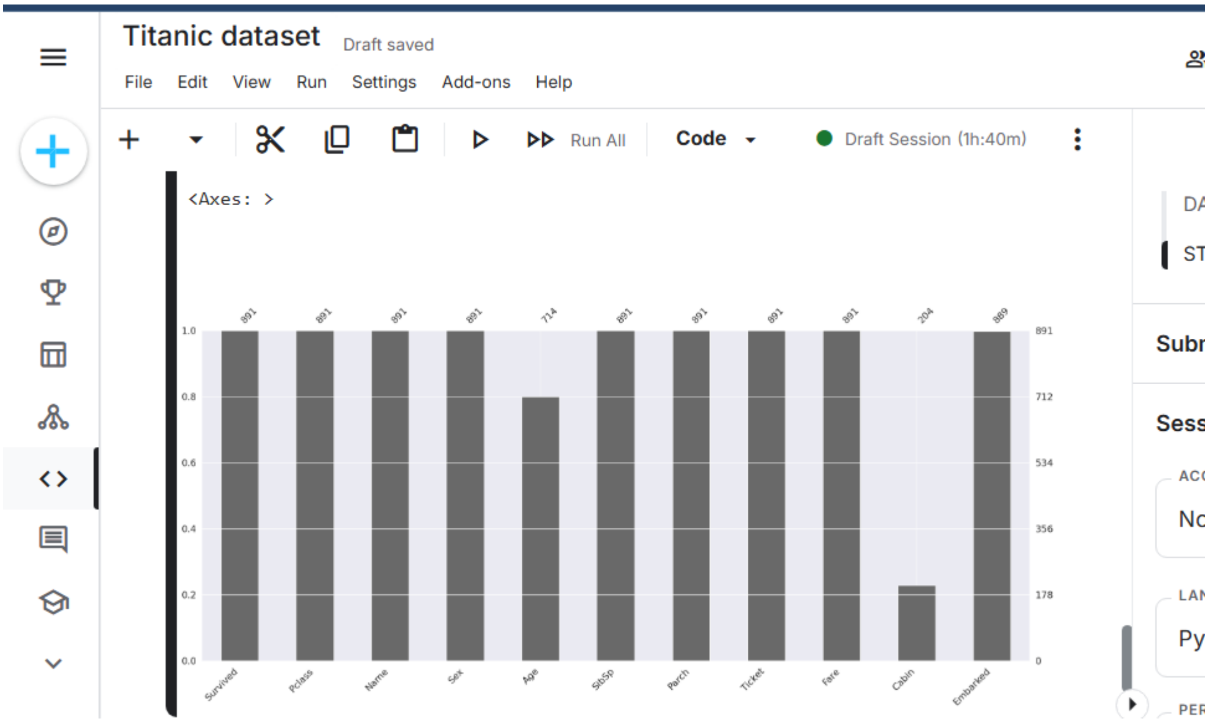


Fig 2.1; Output on missing values

missing values in each column

missing_values = train.isnull().sum().sort_values(ascending=False)

missing_percentage = (missing_values / len(train)) * 100

print(pd.DataFrame({'Missing Values': missing_values, 'Percentage': missing_percentage}))

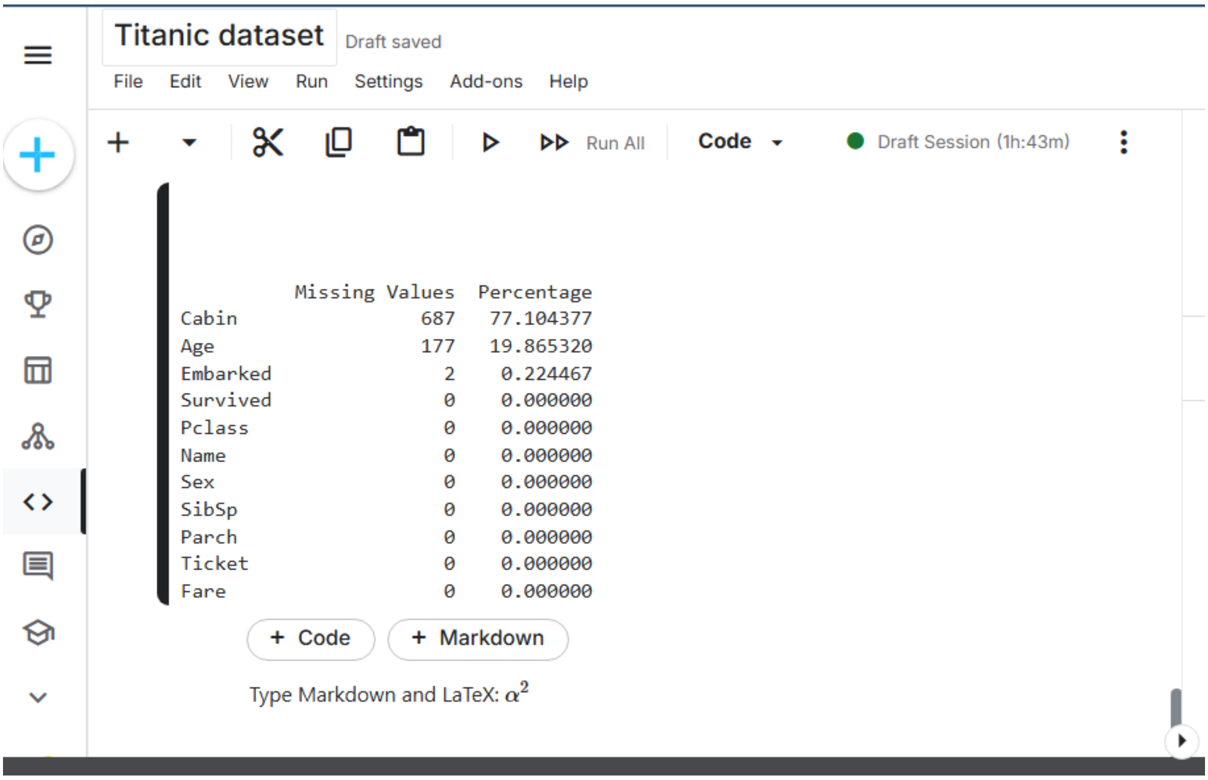


Fig 2.2; Output on missing values in each column

- **Dropping and inputting missing values**

train = train.drop(columns=['Cabin'], errors='ignore')

Fill missing values in the 'Age' column with the mean age

train['Age'].fillna(train['Age'].mean(), inplace=True)

Fill missing values in the 'Fare' column with the median

train['Fare'].fillna(train['Fare'].median(), inplace=True)

Fill missing values in the 'Embarked' column with the most common value (mode)

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

Create a missing value flag for 'Cabin' if the column exists

if 'Cabin' in train.columns:

    train['Cabin_missing_flag'] = train['Cabin'].isnull().astype(int)

else:

```
print("'Cabin' column not found in the DataFrame.")
```

STEP 3: UNIVARIATE ANALYSIS

- **Use a histogram to determine the age distribution of customers**

```
# Histogram for Age
plt.figure(figsize=(8, 5))
sns.histplot(train['Age'].dropna(), bins=30, kde=True)
plt.title('Distribution of Age')
plt.xlabel('Age')
plt.ylabel('Count')
plt.show()
```

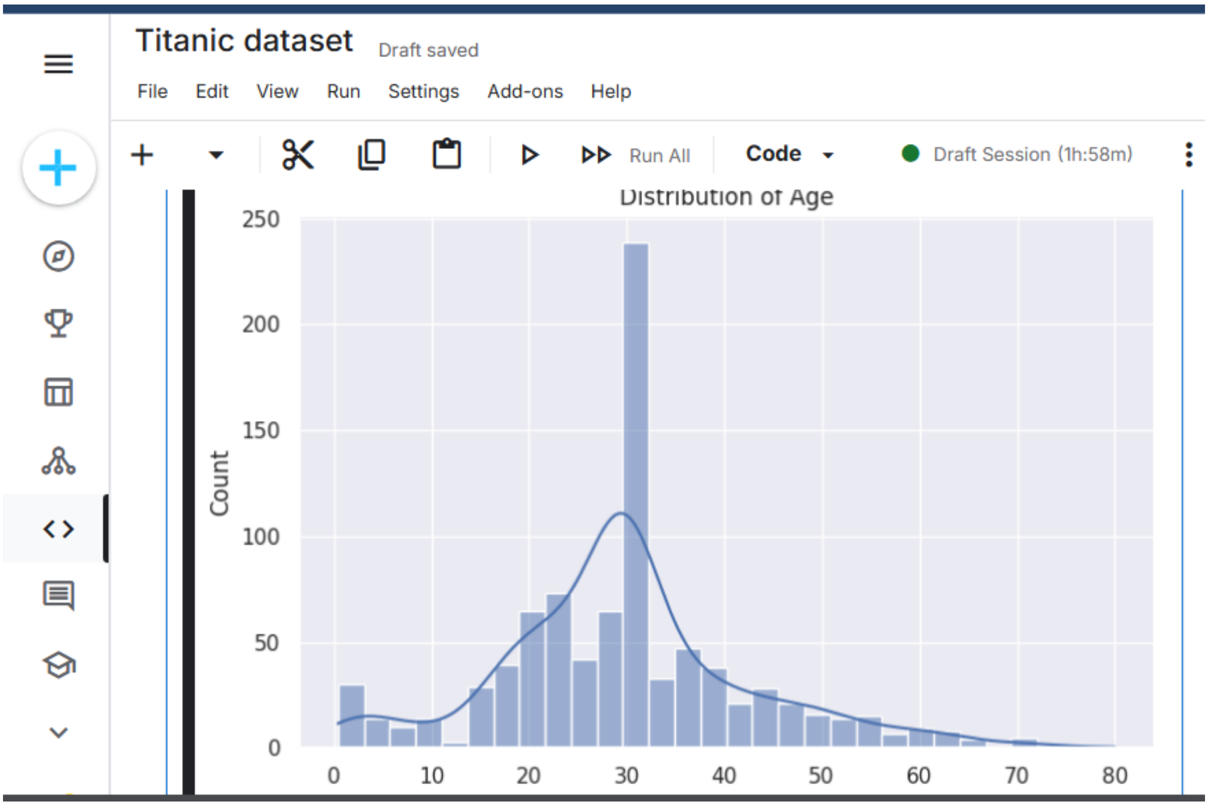


Fig 3.1;Output for age distribution of customers

```
# KDE Plot for Fare
plt.figure(figsize=(8, 5))
sns.kdeplot(train['Fare'], shade=True)
plt.title('KDE Plot of Fare')
plt.xlabel('Fare')
plt.show()
```

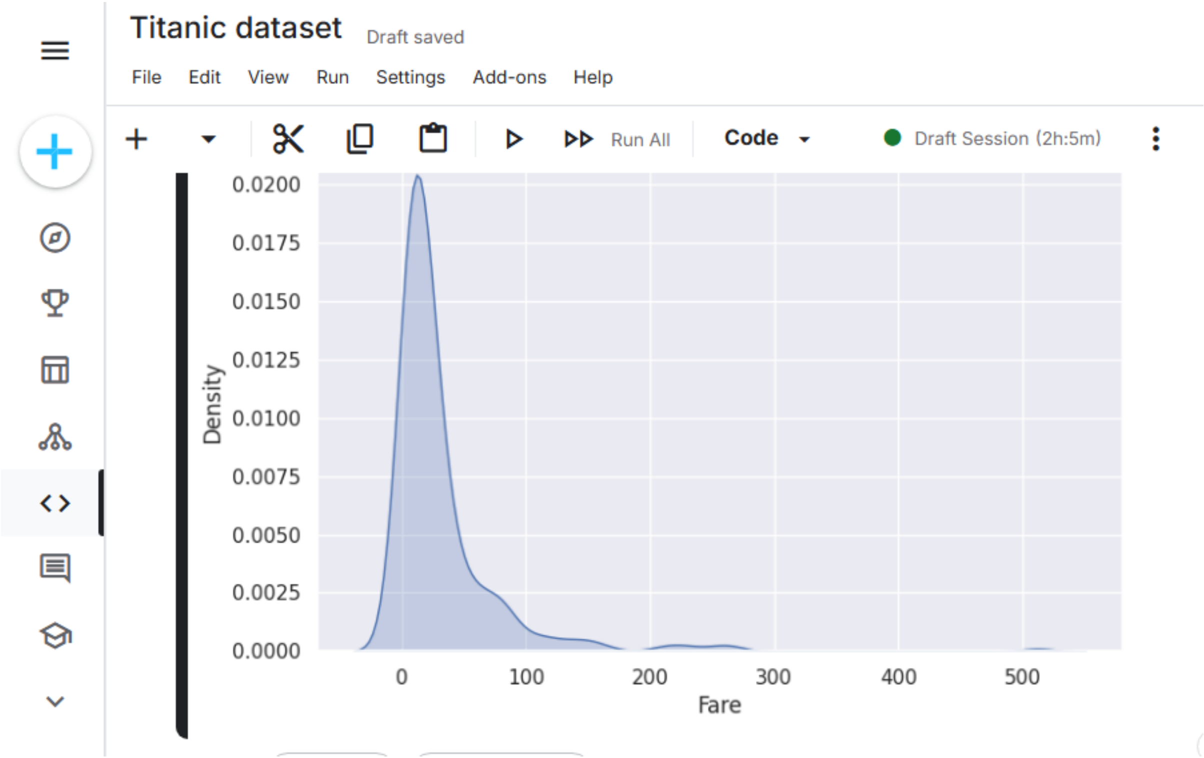


Fig 3.2; Output for KDE plot for fare

- **Use countplot to know how many passengers embarked from each location**

```
# Countplot for Embarked
plt.figure(figsize=(8, 5))
sns.countplot(x='Embarked', data=train, palette='pastel')
plt.title('Countplot of Embarked')
plt.xlabel('Embarkation Port')
plt.ylabel('Count')
plt.show()
```

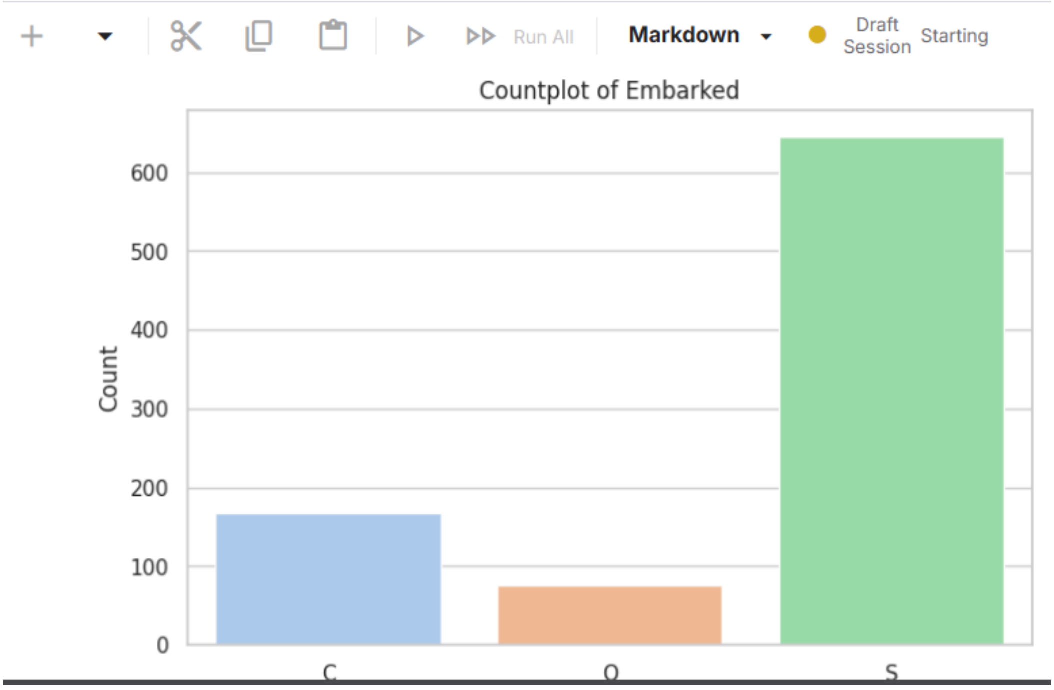


Fig 3.3; Output on countplot embarked

```
# Set a clean style
sns.set(style="whitegrid")
# Create the bar plot comparing survival by embarkation point
sns.countplot(data=train, x='Embarked', hue='Survived')
# Add labels and title
plt.title('Survival Counts by Embarkation Point')
plt.xlabel('Embarkation Point')
plt.ylabel('Passenger Count')
# Show the plot
plt.show()
```

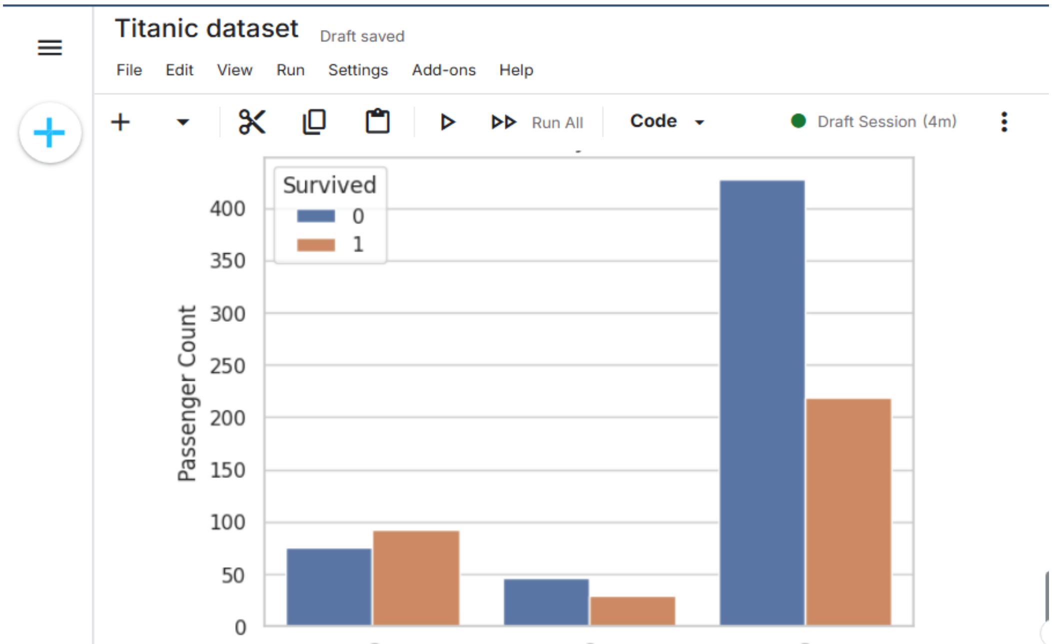


Fig 3.4; Output on survival count by embarked point

- **Summary statistics for categorical variables**

```
# Frequency count of unique values in the 'Pclass' column
print(train['Pclass'].value_counts())
```

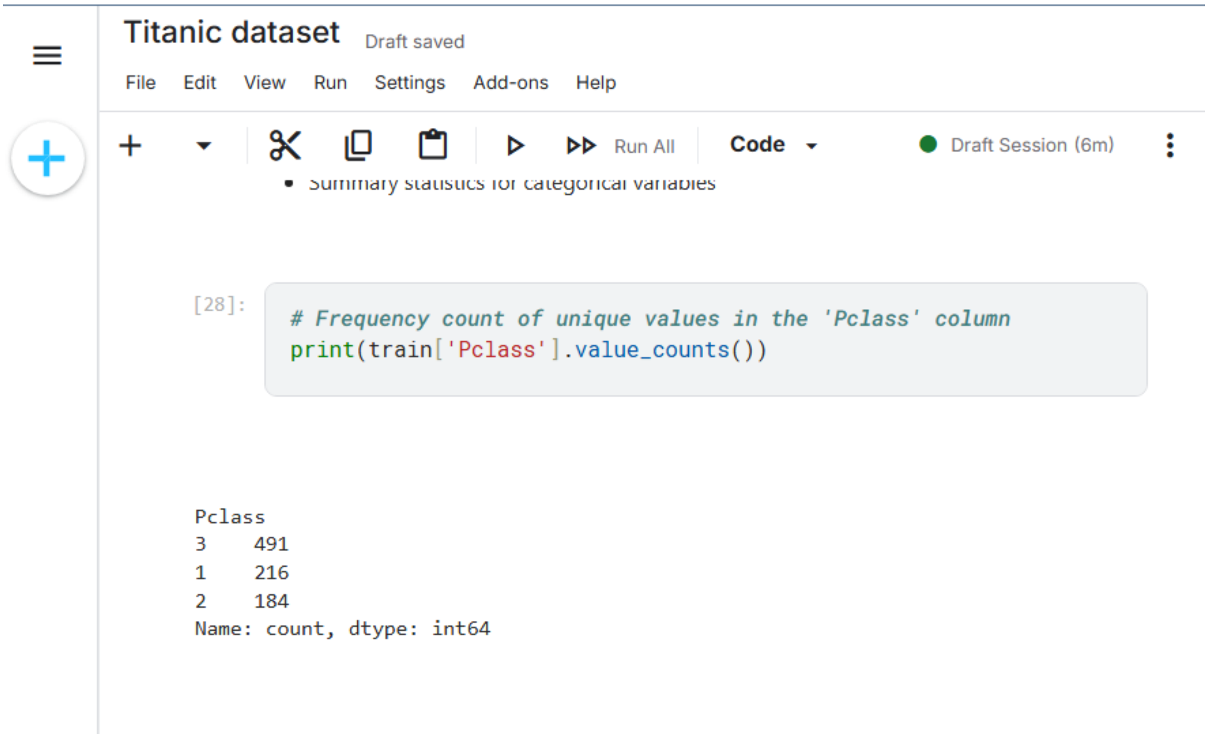


Fig 3.5;Output of summary statistics of categorical variables

STEP 4:BIVARIATE ANALYSIS

- **Numerical vs numerical analysis**

```
# Scatter plot for Age Vs Fare
plt.figure(figsize=(8, 5))
sns.scatterplot(x='Age', y='Fare', data=train, hue='Survived', palette='coolwarm')
plt.title('Scatter Plot of Age vs Fare (Colored by Survived)')
plt.show()
```

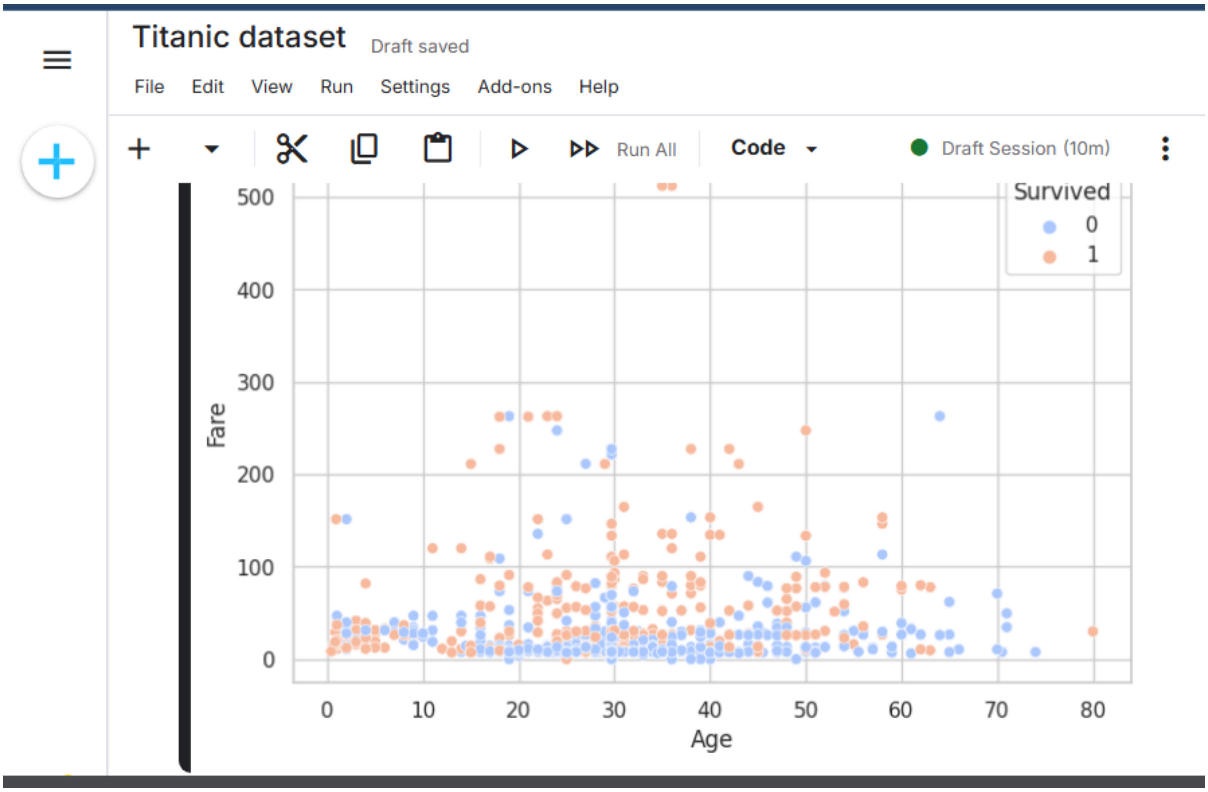


Fig 4.1; Output of fare vs age coloured by survival rate

- **Numerical vs categorical**

```
# Boxplot of Fare grouped by Pclass
plt.figure(figsize=(8, 5))
sns.boxplot(x='Pclass', y='Fare', data=train, palette='Set2')
plt.title('Boxplot of Fare by Pclass')
plt.xlabel('Passenger Class')
plt.ylabel('Fare')
plt.show()
```

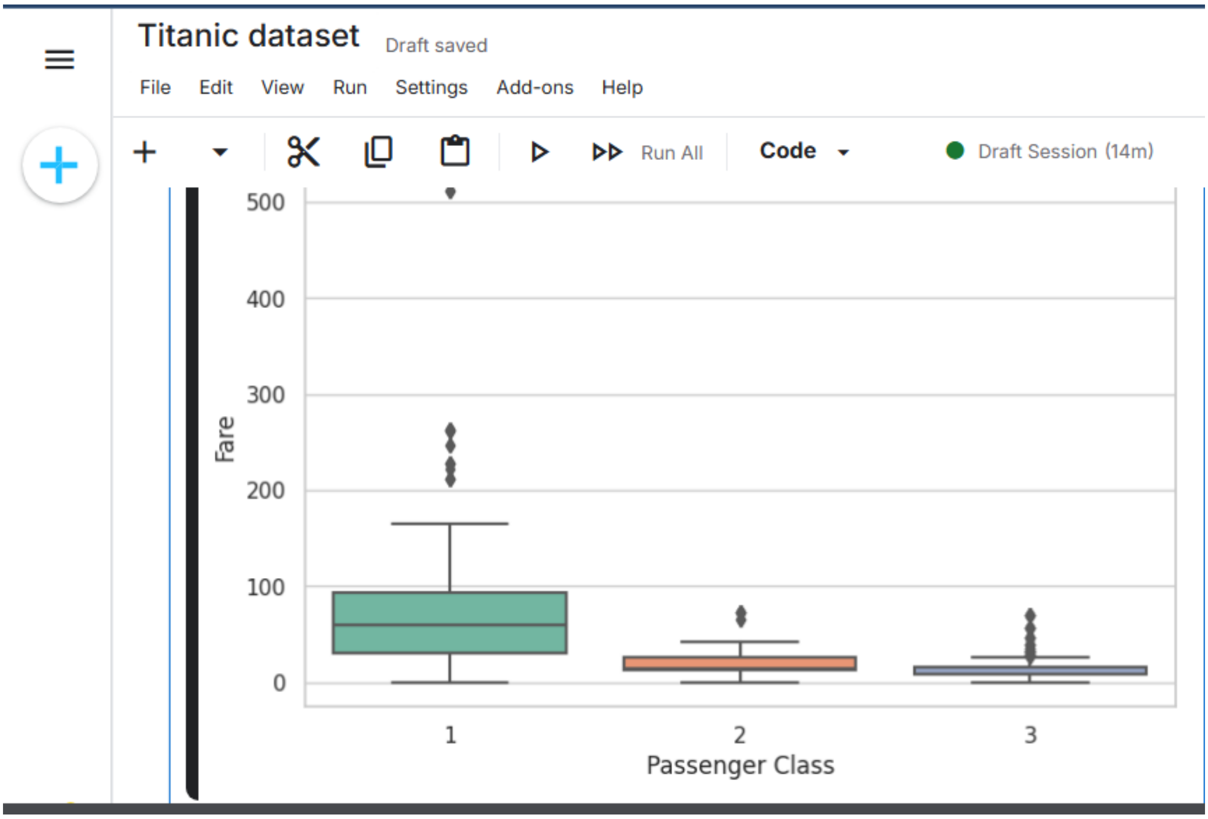


Fig 4.2; Output of boxplot of fare vs pclass

- **Categorical vs categorical**

```
# Grouped bar plot of Survived Vs Embarked
plt.figure(figsize=(8, 5))
sns.countplot(x='Embarked', hue='Survived', data=train, palette='pastel')
plt.title('Survival Counts by Embarked Port')
plt.xlabel('Embarked Port')
plt.ylabel('Count')
plt.show()
```

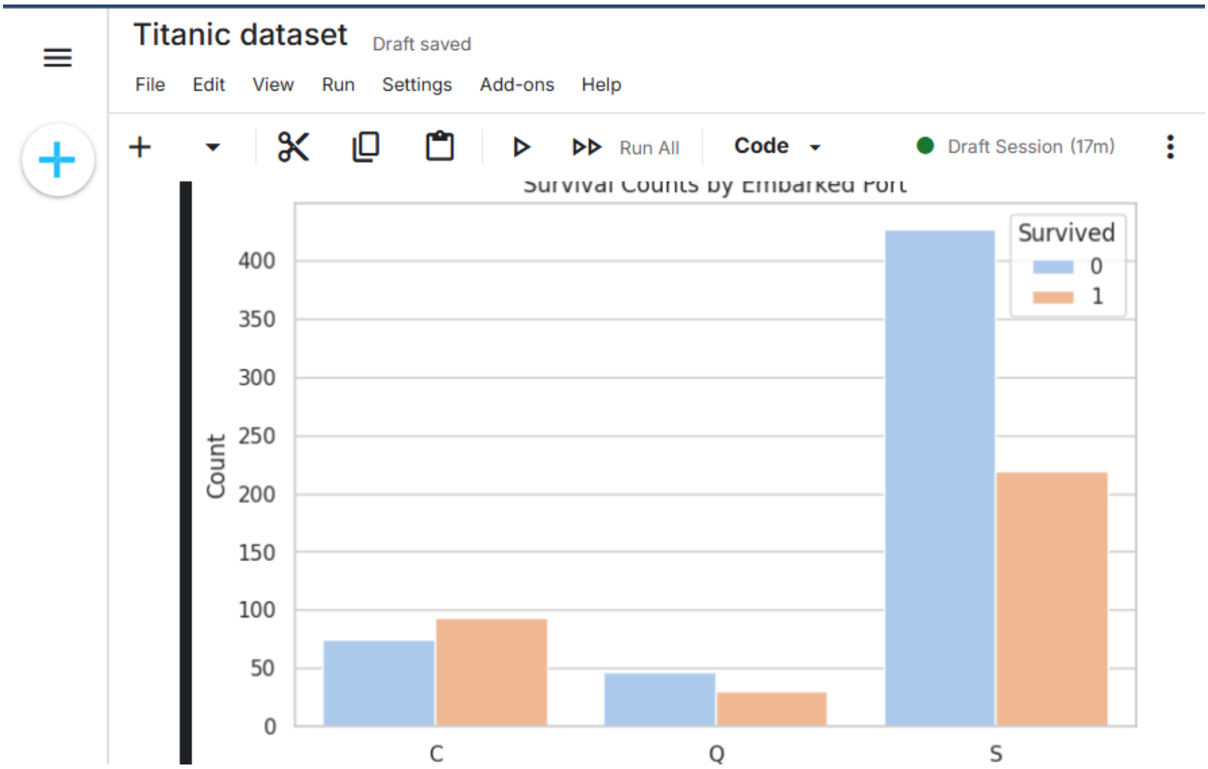


Fig 4.3; Output on survival counts by embarked point

STEP 5: MULTIVARIATE ANALYSIS

- **Pair plot**

```
# Pair plot for numerical columns
plt.figure(figsize=(5, 5))
sns.pairplot(train, hue='Survived', diag_kind='kde', palette='coolwarm')
plt.show()
```

Fig 5.1; Output for pairplot

- **Subplots for subgroups**

```
# FacetGrid for Age distribution by Survived and Pclass
g = sns.FacetGrid(train, col='Survived', row='Pclass', height=4, aspect=1.5)
g.map(sns.histplot, 'Age', kde=True)
plt.show()
```

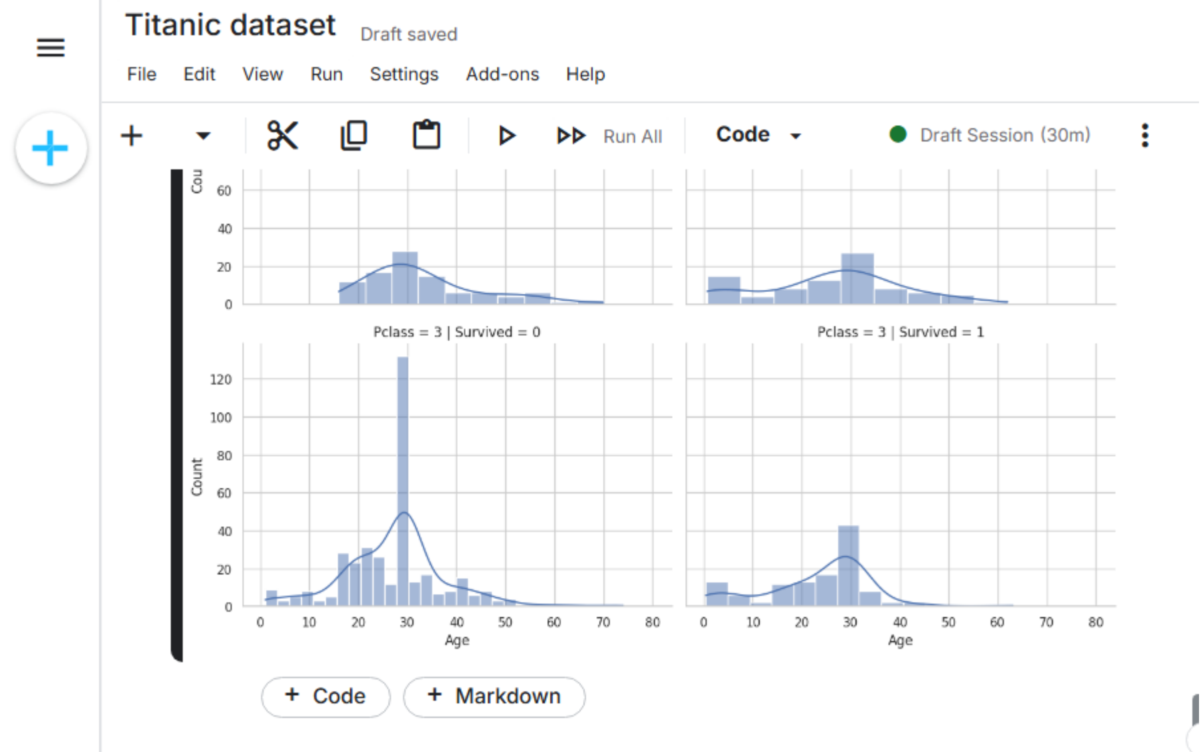


Fig 5.2; Output for facetgrid

- **Heatmaps for numerical columns**

```
# Heatmap of numerical features only
plt.figure(figsize=(8, 6))
numerical_columns = train.select_dtypes(include=['int64', 'float64']).columns  # Select only numerical columns
sns.heatmap(train[numerical_columns].corr(), annot=True, cmap='Blues', fmt='.2f')
plt.title('Correlation Heatmap of Numerical Features')
plt.show()
```

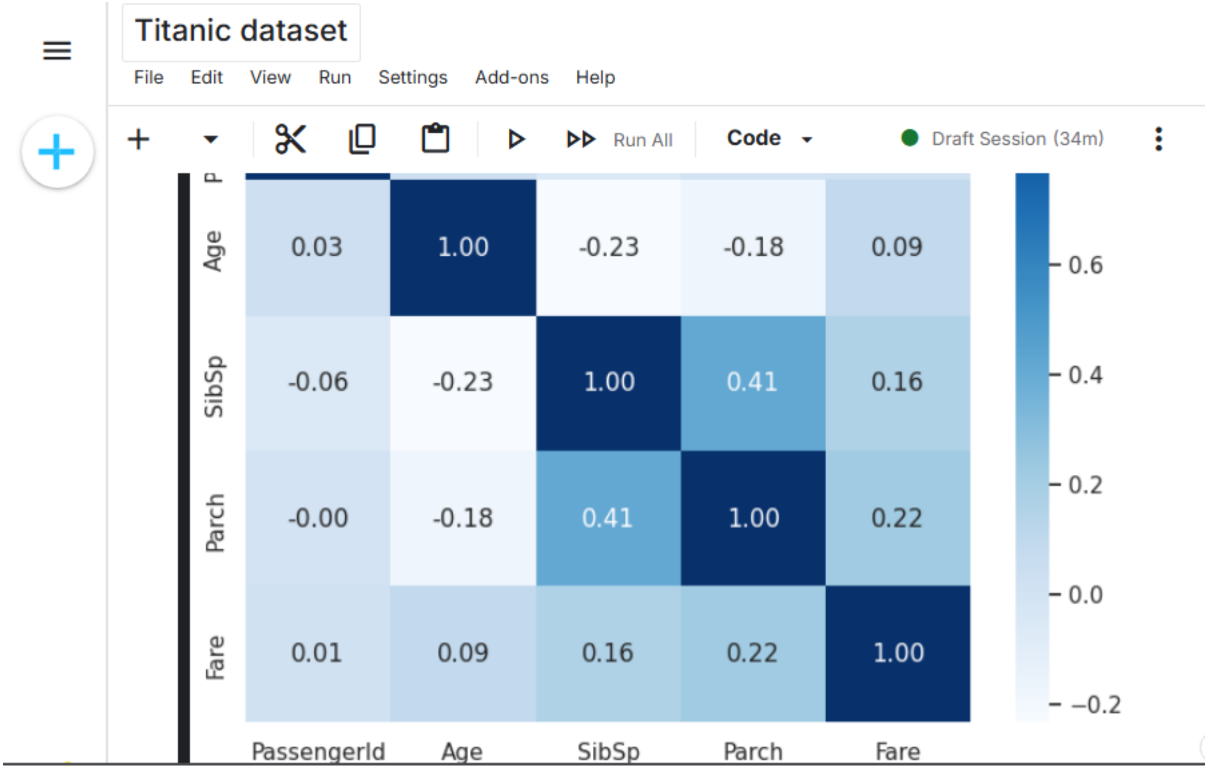


Fig 5.3; Output for correlation heatmap for numerical variables

- **3D scatter plot**

```python
# 3D scatter plot for Age, Fare, and Survived
import plotly.express as px
fig = px.scatter_3d(train, x='Age', y='Fare', z='Survived', color='Pclass', size='Fare', opacity=0.7)
fig.update_traces(marker=dict(line=dict(width=0)))
fig.update_layout(title='3D Scatter Plot: Age vs Fare Vs Survived')
fig.show()
```

Fig 5.4; Output on 3D scatter plot of age vs fare vs survived

STEP 6: OUTLIER DETECTION AND HANDLING

- **Detecting outliers using boxplots**

```
# Boxplot for Fare to detect outliers

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

sns.boxplot(x=train['Fare'], palette='pastel')

plt.title('Boxplot of Fare')

plt.show()
```

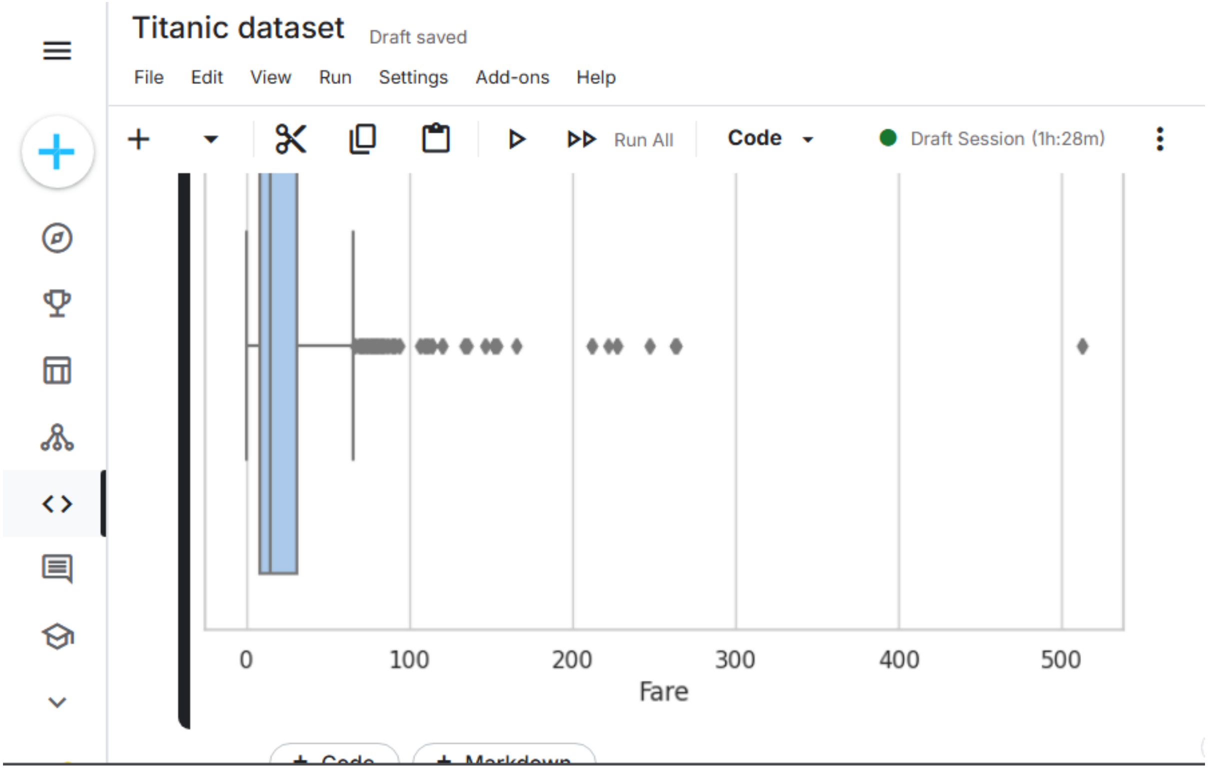


Fig 6.1; Output for boxplot for fare

- Dots outside the whiskers: Represent outliers.
- A long tail or many dots indicates that the column contains extreme values.

- **Detecting Outliers Using Z-Score**

```
# Function to detect outliers using Z-score
from scipy.stats import zscore
def detect_outliers_zscore(data, threshold=3):
    z_scores = zscore(data.dropna())  # Drop NaN to avoid errors
    outliers = data[(abs(z_scores) > threshold)]
    return outliers
# Detect outliers in the 'Age' column
outliers_age = detect_outliers_zscore(train['Age'])
print(f'Number of outliers in Age: {len(outliers_age)}')
```

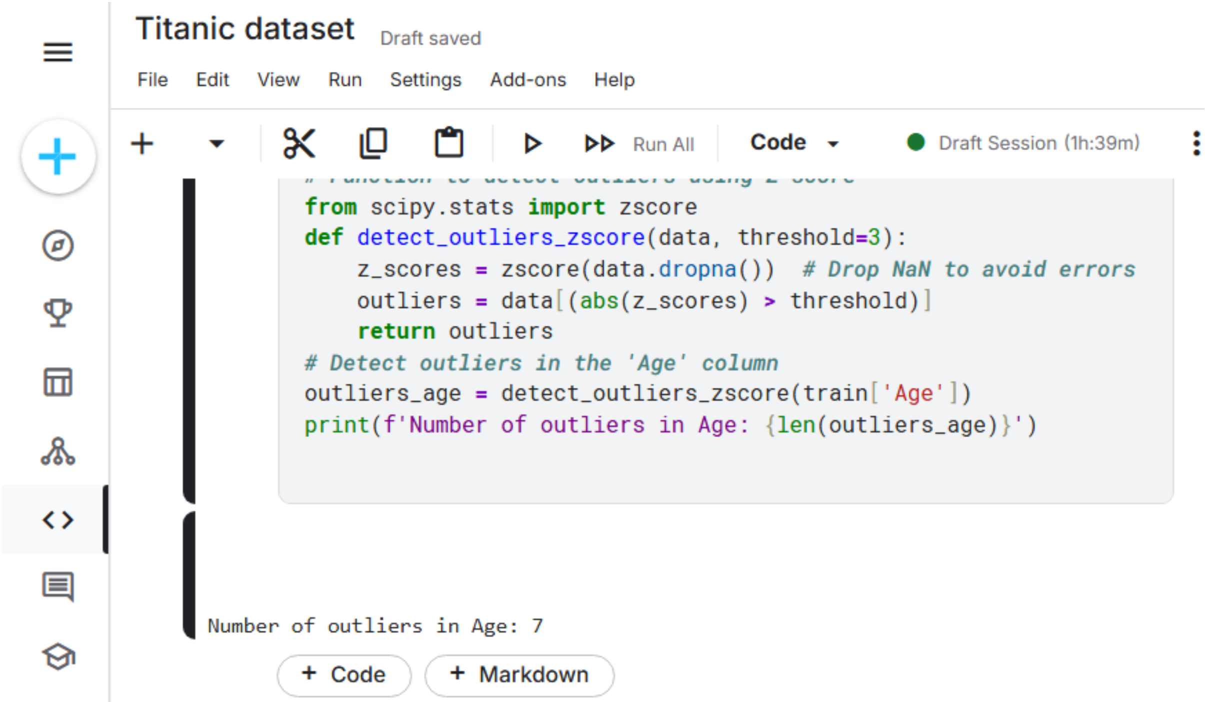


Fig 6.2; Output on number of outliers of age

- Data points with a Z-score greater than the threshold (typically 3) are considered outliers.

- **Detecting outliers using IQR**

```
# Function to detect outliers using IQR

def detect_outliers_iqr(data):

    Q1 = data.quantile(0.25)

    Q3 = data.quantile(0.75)

    IQR = Q3 - Q1

    lower_bound = Q1 - 1.5 * IQR

    upper_bound = Q3 + 1.5 * IQR

    outliers = data[(data < lower_bound) | (data > upper_bound)]

    return outliers

# Detect outliers in the 'Fare' column using IQR

outliers_fare = detect_outliers_iqr(train['Fare'])

print(f'Number of outliers in Fare: {len(outliers_fare)}')
```

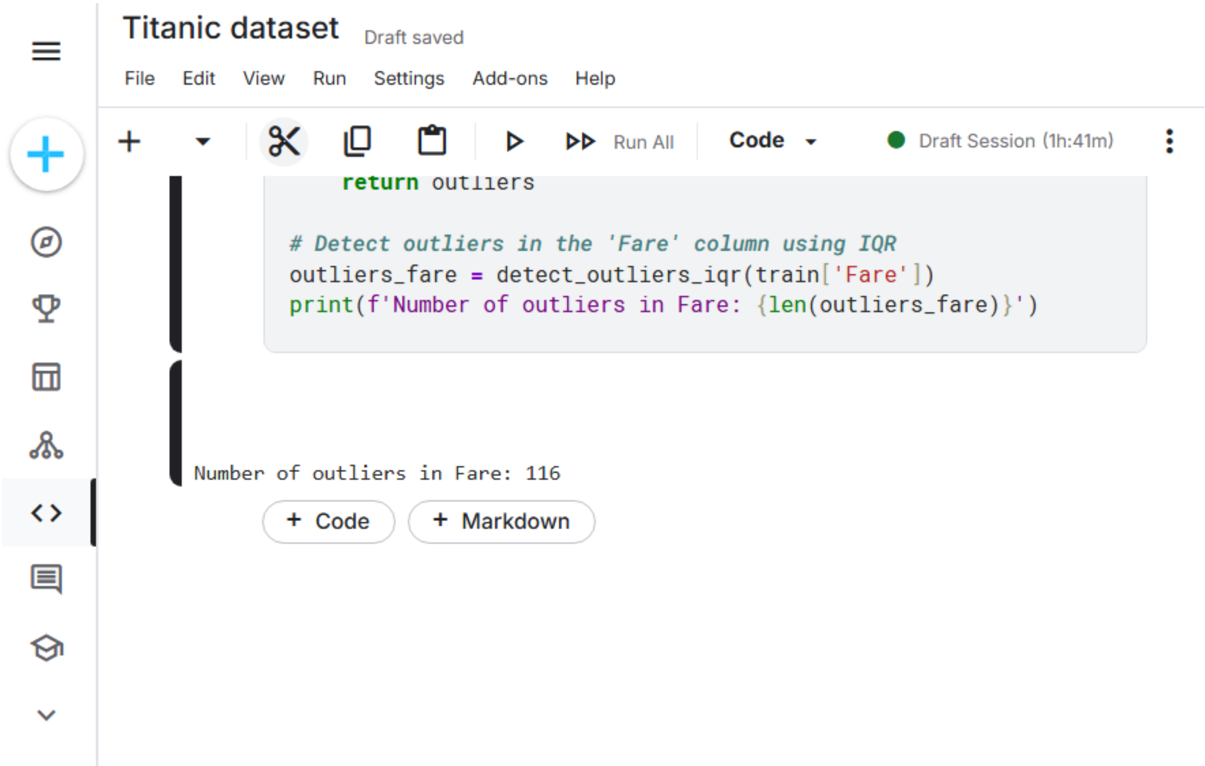


Fig 6.3; Output on number of outliers of variable fare

- Lower bound: Q1 - 1.5 * IQR (minimum expected value).
- Upper bound: Q3 + 1.5 * IQR (maximum expected value).
- Data points outside this range are considered outliers.

- **Handling outliers**

```
# Remove outliers in the 'Fare' column
#train = train[(df['Fare'] >= train['Fare'].quantile(0.25) - 1.5 * (train['Fare'].quantile(0.75) - train['Fare'].quantile(0.25))) &
    #(train['Fare'] <= train['Fare'].quantile(0.75) + 1.5 * (train['Fare'].quantile(0.75) - train['Fare'].quantile(0.25)))]
# Cap outliers in the 'Fare' column
#train['Fare'] = train['Fare'].clip(lower=train['Fare'].quantile(0.05), upper=train['Fare'].quantile(0.95))
# Impute outliers with the median
#train['Fare'] = train['Fare'].mask((train['Fare'] < train['Fare'].quantile(0.05)) | (train['Fare'] > train['Fare'].quantile(0.95)), train['Fare'].median())
```

STEP 7: TARGET VARIABLE EXPLORATION

- **Distribution of survived**

```
# Countplot for Survived
plt.figure(figsize=(8, 5))
sns.countplot(x='Survived', data=train, palette='Set2')
plt.title('Survival Count')
plt.xlabel('Survived (0 = No, 1 = Yes)')
plt.ylabel('Count')
plt.show()
```

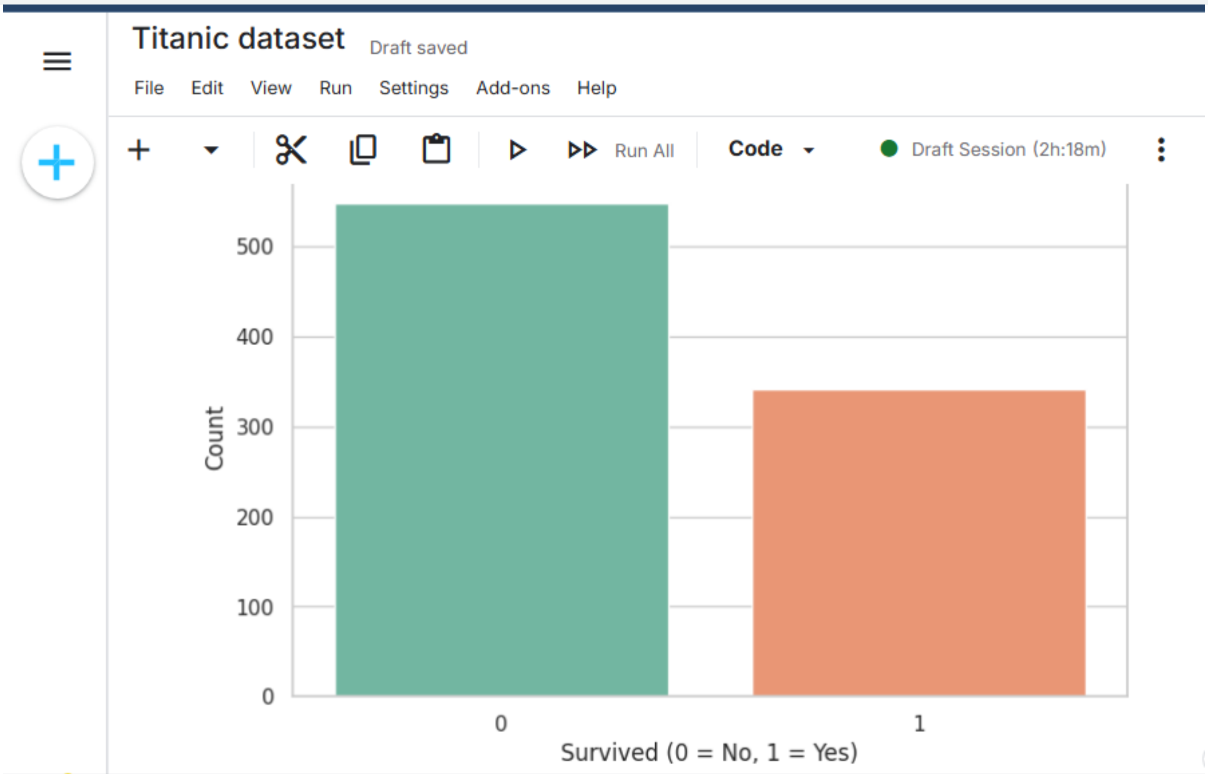


Fig 7.1; Output on survival count

- **Survival rate by numerical columns**

```
# KDE Plot for Age by Survival Status
plt.figure(figsize=(8, 5))
sns.kdeplot(train[train['Survived'] == 1]['Age'], shade=True, label='Survived', color='green')
sns.kdeplot(train[train['Survived'] == 0]['Age'], shade=True, label='Did Not Survive', color='re
d')
plt.title('Age Distribution by Survival Status')
plt.xlabel('Age')
plt.legend()
```

Fig 7.2; Output on age distribution by survival count

- **Survival rate by categorical columns**

```
# Countplot for Survived grouped by Gender
plt.figure(figsize=(8, 5))
sns.countplot(x='Survived', hue='Sex', data=train, palette='muted')
plt.title('Survival Rate by Gender')
plt.xlabel('Survived (0 = No, 1 = Yes)')
plt.ylabel('Count')
plt.show()
```

Fig 7.3; Output on survival rate by gender

- **Combined analysis (Gender, Class, and Survival)**

```
# Grouped bar plot for survival by Gender and Class
plt.figure(figsize=(10, 6))
sns.countplot(x='Pclass', hue='Sex', data=train[train['Survived'] == 1], palette='Set1')
plt.title('Survivors by Gender and Class')
plt.xlabel('Passenger Class')
plt.ylabel('Survivor Count')
plt.show()
```

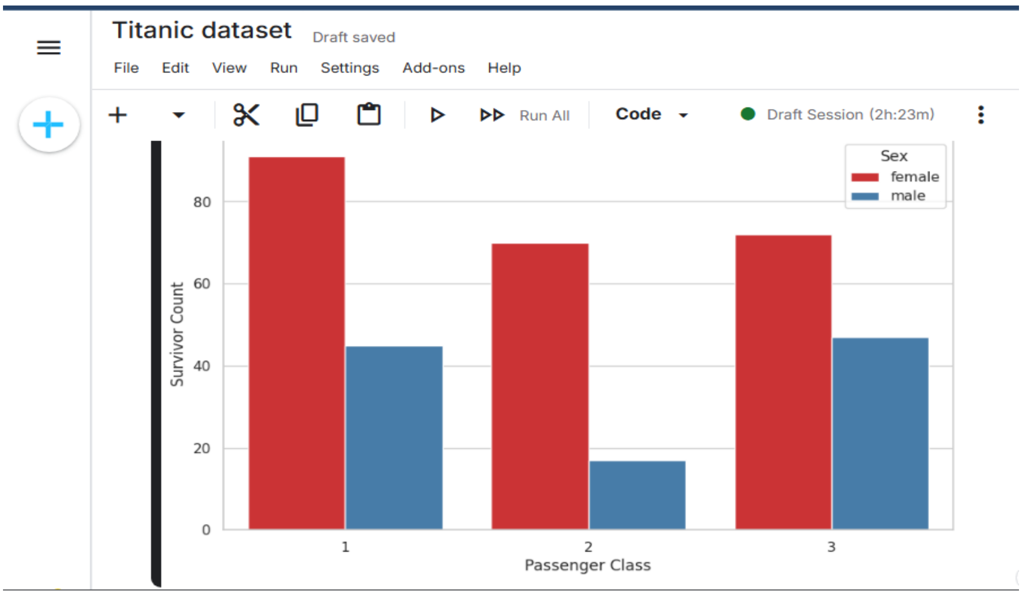


Fig 7.4; Output on survivors of gender and class

Link to Code:

https://www.kaggle.com/code/joyviolet/titanic-dataset

Conclusion

In this exploratory data analysis project, I examined the Titanic dataset to uncover patterns and relationships related to passenger survival. Through data cleaning, I handled missing values in key columns such as Age, Fare, and Embarked, and removed or flagged columns like Cabin which contained too many missing values.

The univariate and bivariate analyses revealed several important insights. Survival rates were significantly higher among female passengers, children, and those who traveled in first class. Embarkation point also showed some influence, with passengers boarding at Cherbourg (C) having a higher survival rate. Visualizations such as bar plots with hue='Survived' and histograms helped illustrate these patterns clearly.

Overall, EDA proved to be an essential step in understanding the dataset and identifying influential factors related to survival. The analysis highlighted social and economic inequalities that affected survival outcomes, which could inform predictive modeling in future stages. While the dataset had limitations such as missing values and potential biases, the insights gained are valuable and demonstrate the power of data exploration. This project reinforced the importance of EDA in any data science workflow.