CS603 INTRO TO DATA ANALYTICS

FINAL PROJECT – KAGGLE COMPETITION

CASE STUDY: Titanic Case Study

TEAM Members:

Safal Lamichhane

Zach Seitz

Kao Takahama

Tyler Selby

Tyler Blankenship

Niharika Rasthapuram

Table of Contents

Table of Figures

Introduction

The topic of our group is Spaceship Titanic. The data collection provides information about people aboard the Spaceship Titanic after it became stranded in space due to a spacetime anomaly. The most essential aspect of the data set is whether the passenger was moved to another dimension (Survival = False) or remained on board (Survival = True). Kaggle offers two data sets: a training set and a test set. The test set does not contain Transported data because it is used to test ML models. As a result, we will only use the data from the training set, train.csv.

Objective

The objective of our study is to do a general exploration of the features of the dataset and in the process discover which ones are most closely related to whether a passenger was transported by the anomaly.

Tools Used

We have used these tools as part of the case study as per the objective described.

- Microsoft Excel: For Data analysis quickly and for sanity checks and to glance the data
- Jupyter Notebook

Libraries Used

1. Pandas: Pandas is a machine learning tool and is used for data preprocessing and analysis. It has functionalities for data exploration, cleaning, transformation, and visualization
2. NumPy: NumPy could be used to carry out a variety of array-based mathematical operations. It extends Python with powerful data structures that ensure efficient estimations with arrays and matrices, as well as an immense library of elevated arithmetic operations that operate on such arrays and matrices
3. Matplotlib: Matplotlib is a Python library that allows you to create static, animated, and immersive visualizations. Matplotlib makes simple things simple and difficult things possible.
4. Seaborn: Seaborn is a popular data visualization library in Python that is built on top of the Matplotlib library. It provides a high-level interface for creating informative and attractive statistical graphics.

Data Loading

Data loading is the process of loading data from a source, such as a file or a database, into a software program or application for further processing. The data can be in various formats, including structured, semi-structured, or unstructured data. In the project, we read the data and undertake some basic exploring in this part. We'll figure out what data is missing and how to deal with it, as well as whether or not to execute certain data cleaning activities.

Importing the data

In this section, we'll read the data and do some basic exploration. We'll determine what data is missing and how to handle it, as well as decide whether or not to perform various other data cleaning operations. We have imported the data from the Kaggle dataset. There was train.csv and test.csv file in this dataset and we have worked on train.csv file. The provided train.csv has the personal records of around 8700 passengers. The fields are:

PassengerId - A unique Id for each passenger. Each Id takes the form gggg_pp where gggg indicates a group the passenger is travelling with and pp is their number within the group.

HomePlanet - The planet the passenger departed from, typically their planet of permanent residence.

CryoSleep - Indicates whether the passenger elected to be put into suspended animation for the duration of the voyage. Passengers in cryosleep are confined to their cabins.

Cabin - The cabin number where the passenger is staying. Takes the form deck/num/side, where side can be either P for Port or S for Starboard.

Destination - The planet the passenger will be debarking to.

Age - The age of the passenger.

VIP - Whether the passenger has paid for special VIP service during the voyage.

RoomService, FoodCourt, ShoppingMall, Spa, VRDeck - Amount the passenger has billed at each of the Spaceship Titanic's many luxury amenities.

Name - The first and last names of the passenger.

Transported - Whether the passenger was transported to another dimension.

Source: https://www.kaggle.com/competitions/spaceship-titanic/data

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```python
# Load the data set
df = pd.read_csv('train.csv')
df.head(1)
```

	PassengerId	HomePlanet	CryoSleep	Cabin	Destination	Age	VIP	RoomService	FoodCourt	ShoppingMall	Spa	VRDeck	Name	Transported
0	0001_01	Europa	False	B/0/P	TRAPPIST-1e	39.0	False	0.0	0.0	0.0	0.0	0.0	Maham Ofracculy	False

Figure 1:Loading the data

```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8693 entries, 0 to 8692
Data columns (total 14 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   PassengerId   8693 non-null   object 
 1   HomePlanet    8492 non-null   object 
 2   CryoSleep     8476 non-null   object 
 3   Cabin         8494 non-null   object 
 4   Destination   8511 non-null   object 
 5   Age           8514 non-null   float64
 6   VIP           8490 non-null   object 
 7   RoomService   8512 non-null   float64
 8   FoodCourt     8510 non-null   float64
 9   ShoppingMall  8485 non-null   float64
 10  Spa           8510 non-null   float64
 11  VRDeck        8505 non-null   float64
 12  Name          8493 non-null   object 
 13  Transported   8693 non-null   bool   
dtypes: bool(1), float64(6), object(7)
memory usage: 891.5+ KB
```

Figure 2 Attributes

It looks like data set contains 8693 total entries and 14 unique features. The table above also shows that the only columns without missing data are PassengerId and Transported. Ultimately, however, no single feature is missing so much data that it is worth dropping outright, so we'll keep them all in until we explore a bit further.

While most of the data is categorical, there are some numerical features we can look more closely at. Columns 7-11 above represent the amount of money passengers spent at the respective amenity aboard the Spaceship Titanic.

```
df.describe()
```

	Age	RoomService	FoodCourt	ShoppingMall	Spa	VRDeck
count	8514.000000	8512.000000	8510.000000	8485.000000	8510.000000	8505.000000
mean	28.827930	224.687617	458.077203	173.729169	311.138778	304.854791
std	14.489021	666.717663	1611.489240	604.696458	1136.705535	1145.717189
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	19.000000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	27.000000	0.000000	0.000000	0.000000	0.000000	0.000000
75%	38.000000	47.000000	76.000000	27.000000	59.000000	46.000000
max	79.000000	14327.000000	29813.000000	23492.000000	22408.000000	24133.000000

Figure 3 Descriptive status of data

From Figure 3, we can see the mean, std, max, count and other for out dataset. We could clearly see that it is 0 in the min field which meant there is possible that that is missing.

The Age Feature

The zeroes in the min row of the table above show possible missing data. So, we decided to explore more into the age feature here in respect to the passengers.

```
# Get count of ages amongst passengers
df_age_0_count = df.groupby('Age')[['PassengerId']].count()
print(df_age_0_count)
```

Figure 4 Exploring Age field

```
      PassengerId
Age
0.0           178
1.0            67
2.0            75
3.0            75
4.0            71
...           ...
75.0            4
76.0            2
77.0            2
78.0            3
79.0            3

[80 rows x 1 columns]
```

The table above show's that here are 178 passengers where `Age == 0`. Let's take a look at the first five of them.

Figure 5 Age Feature

Now, after seeing the age feature, we figured out that there are 174 passengers who have the age 0. Now, we did the further analysis to find out more about this field.

```
df_age_0 = df[df.Age == 0]
df_age_0.head()
```

	PassengerId	HomePlanet	CryoSleep	Cabin	Destination	Age	VIP	RoomService	FoodCourt	ShoppingMall	Spa	VRDeck	Name	Transported
19	0017_01	Earth	False	G/0/P	TRAPPIST-1e	0.0	False	0.0	0.0	0.0	0.0	0.0	Lyde Brighttt	True
61	0067_01	Earth	True	G/10/S	PSO J318.5-22	0.0	False	0.0	0.0	0.0	0.0	0.0	Ninaha Leeves	True
86	0092_02	Earth	True	G/9/P	TRAPPIST-1e	0.0	False	0.0	0.0	NaN	0.0	0.0	Stald Hewson	True
102	0108_03	Earth	False	G/19/S	TRAPPIST-1e	0.0	NaN	0.0	0.0	0.0	0.0	0.0	Oline Handertiz	True
157	0179_02	Earth	False	G/26/P	TRAPPIST-1e	0.0	False	0.0	0.0	0.0	0.0	0.0	Raque Webstephrey	False

Figure 6 Searching Age equal to 0 passengers

We saw that there are 5 people who had age equivalent to 0. Now, we will try to find out more about the passengers where age is NaN.

```
df_age_nan = df[pd.isna(df.Age)]
df_age_nan.head()
```

	PassengerId	HomePlanet	CryoSleep	Cabin	Destination	Age	VIP	RoomService	FoodCourt	ShoppingMall	Spa	VRDeck	Name	Transported
50	0052_01	Earth	False	G/6/S	TRAPPIST-1e	NaN	False	4.0	0.0	2.0	4683.0	0.0	Elaney Hubbarton	False
64	0068_01	Mars	False	E/4/S	TRAPPIST-1e	NaN	False	793.0	0.0	2.0	253.0	0.0	Cinst Binie	False
137	0149_01	Earth	True	G/27/S	55 Cancri e	NaN	False	0.0	0.0	0.0	0.0	0.0	Billya Hubbarrison	True
181	0202_02	Europa	False	A/2/P	55 Cancri e	NaN	False	0.0	2433.0	NaN	878.0	443.0	Vegas Embleng	True
184	0206_01	Europa	False	C/9/S	55 Cancri e	NaN	False	2.0	1720.0	12.0	1125.0	122.0	Nuson Brugashed	True

Figure 7 Passenger with age NaN

In the figure 7, we can see that the lists of passengers are different, we now know that the data set differentiates between passengers whose age is actually 0 (taken to mean less than 1 year of age) and those whose age is missing. As such, we can safely drop rows where Age == NaN.

Missing Data

At this point, it may be possible to drop all rows with missing data. Let's first get a count of the number of rows with missing data.

```python
# Check how many row contain missing data
original_len = df.shape[0]
new_len = df.dropna(inplace=False).shape[0]

print(f"Number with missing data: {original_len - new_len}")
print(f"Remaining rows: {new_len}")

print(f"Percent of data set retained: {new_len/original_len*100}%")
```

```
Number with missing data: 2087
Remaining rows: 6606
Percent of data set retained: 75.99217761417232%
```

Figure 8 Missing data rows

From figure 8, It looks like we'll keep about 76% of our original data set if we drop all NaN values. However, some columns will likely be dropped anyway, so if we hold off until we pare down the data set some more, we might be able to save some of our data. As such, we'll first take a look at which columns we can drop.

```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8693 entries, 0 to 8692
Data columns (total 14 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   PassengerId   8693 non-null   object 
 1   HomePlanet    8492 non-null   object 
 2   CryoSleep     8476 non-null   object 
 3   Cabin         8494 non-null   object 
 4   Destination   8511 non-null   object 
 5   Age           8514 non-null   float64
 6   VIP           8490 non-null   object 
 7   RoomService   8512 non-null   float64
 8   FoodCourt     8510 non-null   float64
 9   ShoppingMall  8485 non-null   float64
 10  Spa           8510 non-null   float64
 11  VRDeck        8505 non-null   float64
 12  Name          8493 non-null   object 
 13  Transported   8693 non-null   bool   
dtypes: bool(1), float64(6), object(7)
memory usage: 891.5+ KB
```

Figure 9 Dataframe Information

Name Feature

We don't expect any relationship between Name and any other feature, so we can safely drop that column. The CryoSleep feature also has important implications on the data set. CryoSleep is a Boolean representing whether or not the passenger was frozen for the duration of the trip. If CryoSleep == True for a given passenger, that passenger would not be able to spend money at the amenities listed in columns 7-11. As such, if any of those data are missing for passengers where CryoSleep == True, we can safely populate those values with zeros.

We can count the number of rows that meet this condition.

```
# Count the number of rows where CryoSleep == True
# and any value in columns 7-11 is NaN
condition = df['CryoSleep'] & df.iloc[:, 7:12].isna().any(axis=1)
num_records = len(df[condition])

print(f"The number of sleepers with NaN speding values: {num_records}")
```

```
The number of sleepers with NaN speding values: 347
```

Figure 10 Sleepers with Nan spending values

From figure 10, it looks like that would prevent us from losing 347 rows, so that's definitely a worthwhile operation. Finally, we'll check for duplicate entries, so we can remove any that are found.

```
# Check for duplicates
duplicates = df[df.duplicated()]

print("Number of duplicate rows:", len(duplicates))
```

```
Number of duplicate rows: 0
```

Figure 11 Checking Duplicate rows

Data Loading Summary

From Data loading section analysis, we further performed:

- Remove Name column
- Populate with zeros any NaN value in columns 7-11 where CryoSleep == True
- Drop all remaining rows containing NaN values
- Drop duplicate rows (we will skip this step since there weren't any)

These represent the more obvious cleaning operations that present themselves without looking into the dataset further. As such, it may be necessary to carry out other cleaning methods elsewhere. Importantly, we can also extract some features, which we'll talk more about in the Data Wrangling section

Data Cleaning

In Data Cleaning section, we implemented our decisions from above block to drop rows and columns that are irrelevant to our analysis (duplicate data, NaN values, values that are unique in every row, etc.)

Also on the further note, similar to the actual Titanic data set, we can create a Family column that is a count of the number of family members a passenger has on board by using the Name feature before it is dropped. We could potentially do analysis of spending habits/make questions out of this data (e.g., which types of passengers were more likely to spend at various locations, etc.)

Dropping Name column

```
# Drop Name column
df = df.drop(['Name'], axis=1)
```

```
# Verify Name column has been dropped
df.head(1)
```

	PassengerId	HomePlanet	CryoSleep	Cabin	Destination	Age	VIP	RoomService	FoodCourt	ShoppingMall	Spa	VRDeck	Transported
0	0001_01	Europa	False	B/0/P	TRAPPIST-1e	39.0	False	0.0	0.0	0.0	0.0	0.0	False

Figure 12 Dropping Name column

Here in the figure 12, we can see that we have dropped Name column.

Populating NaN Values

Now, we will populate the Nan values in columns 7-11 where CryoSleep is true.

```
# Populate with zeros any NaN value in columns 7-11 where CryoSleep == True
df.iloc[condition, 7:12] = 0
```

```
# Verify NaN proliferation
df.head(1)
```

	PassengerId	HomePlanet	CryoSleep	Cabin	Destination	Age	VIP	RoomService	FoodCourt	ShoppingMall	Spa	VRDeck	Transported
0	0001_01	Europa	False	B/0/P	TRAPPIST-1e	39.0	False	0.0	0.0	0.0	0.0	0.0	False

Figure 13 Populate NaN values

As we can see in the figure 13, we have populated the NaN values with zeroes.

Dropping NaN rows

```
# Get a new count of the number of rows containing NaN values
new_len = df.dropna(inplace=False).shape[0]

print(f"Number of rows with missing data: {original_len - new_len}")
print(f"Number of rows retained: {new_len}")
print(f"Percent of data set retained: {new_len/original_len*100}%")
```

```
Number of rows with missing data: 1621
Number of rows retained: 7072
Percent of data set retained: 81.3528126078454%
```

Figure 14 Getting new count of rows containing NaN values

From figure 14, we can see that:

Number of rows with missing data: 1621

Number of rows retained: 7072

Percent of data set retained: 81.3528126078454%

By performing selective cleaning operations, we were able to save an additional 5% of our data set compared to just dropping all NaN outright. Ultimately, this will be enough data to work with, so we will not need to populate other values and we can safely drop the remaining NaN values at this point.

```
# Drop all rows containing NaN values
df.dropna(inplace=True)
```

```
# Verify that NaN rows have been dropped (expected: 7072)
print(f"Remaining data set length: {df.shape[0]}")
```

```
Remaining data set length: 7072
```

Figure 15 Dropping rows containing NaN values

As we can see in the figure 15, we have dropped all the rows that contains NaN values and we can see that the remaining data set length is 7072

Extracting Features

We can extract a few features that may be helpful to our analysis. For example, the Cabin feature is in the following format: deck/num/side. It stands to reason maybe passengers on a particular deck or particular side were more impacted than others.

Deck and Side Features

The deck and side can be extracted into their own features. We won't need to keep num since that's just an identifier for the room, so we can drop the original Cabin column

```
# Extract the Deck and Side features from the Cabin feature
df['Deck'] = df['Cabin'].str[0]
df['Side'] = df['Cabin'].str[-1]
```

Figure 16 Extracting Deck and Side

```
# Drop the original Cabin
df = df.drop(['Cabin'], axis=1)
```

Figure 17 Removing original Cabin

```
# Show successful addition of the Deck and Side features
df.head(1)
```

	PassengerId	HomePlanet	CryoSleep	Destination	Age	VIP	RoomService	FoodCourt	ShoppingMall	Spa	VRDeck	Transported	Deck	Side
0	0001_01	Europa	False	TRAPPIST-1e	39.0	False	0.0	0.0	0.0	0.0	0.0	False	B	P

Figure 18 Deck and Side features

As we can see in the Figure 18, we have added the features.

Group Feature

PassengerId is in the form gggg_pp, where gggg represents the passenger's group and pp represents their id within the group. As such, we can similarly extract a Group feature from the PassengerId feature.

We then dropped the original PassengerId feature because the pp portion loses meaning without the group identifier and it would be unique to each passenger anyway, so it's not super useful to our analysis. That is to say, we don't expect a relationship between the passenger's unique ID and the other features, but their membership within a group may show some kind of relationship(s). We can just use the index as the *de facto* passenger ID instead.

```
# Extract Group feature from PassengerId
df['Group'] = df['PassengerId'].str.split('_').str[0]

# Drop PassengerId
df = df.drop(['PassengerId'], axis=1)
```

Figure 19 Extracting Group feature and dropping passengerID

```
# Verify creation of Group and dropping of PassengerId
df.head(1)
```

	HomePlanet	CryoSleep	Destination	Age	VIP	RoomService	FoodCourt	ShoppingMall	Spa	VRDeck	Transported	Deck	Side	Group
0	Europa	False	TRAPPIST-1e	39.0	False	0.0	0.0	0.0	0.0	0.0	False	B	P	0001

Figure 20 Group feature in our Dataframe

As we can see from the above figure, the Group feature has been extracted and the passengerId feature has been dropped.

Data Wrangling and Aggregation

Data wrangling is the process of organizing and integrating disorganized and complex amounts of data for easy processing and retrieval. Due to the continually expanding and evolving amount of data and data source materials, it is getting increasingly huge amounts of raw data must be organized for analysis. We have completed the process of data wrangling following data cleaning.

General Exploration

It's often a good idea to explore the data set in a more general way, as previously hidden trends may start to reveal themselves. As such, before making assumptions about the data, we're going to performs some basic exploration.

Check for balance

Before moving further, we made sure we have roughly equal numbers of passengers where Transported == True and passengers where Transported == False, to ensure the data set is roughly balanced after the cleaning process.

```
# Plot counts
plot = sns.countplot(x='Transported',data=df)

# Set title
plot.set(title="Count of Transported vs. Not Transported Passengers")

print(plot)
```

```
AxesSubplot(0.125,0.11;0.775x0.77)
```

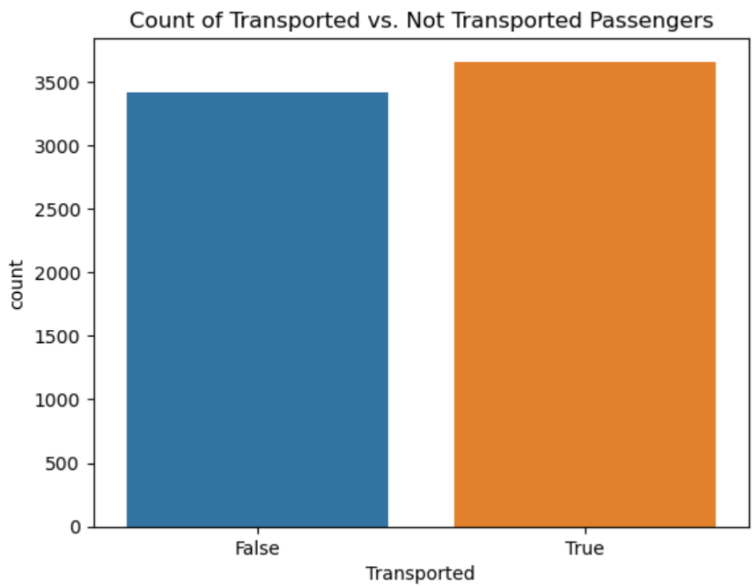


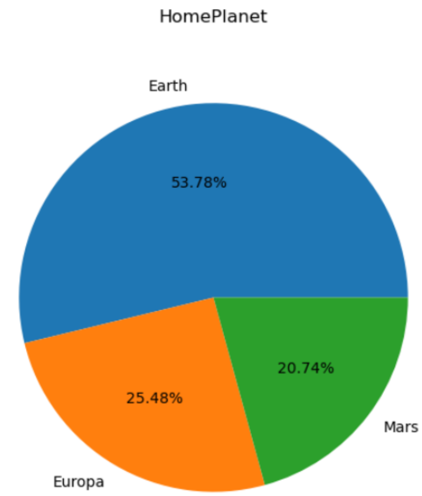
Figure 21 Checking Imbalances in Data (Transported vs non-Transported passengers)

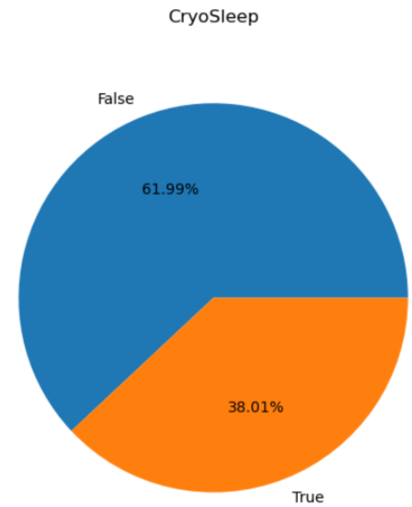
In the figure 21, we can see that while that data set does contain more passenger where Transported == True, the data set is mostly even, so we're not too concerned about data imbalance.

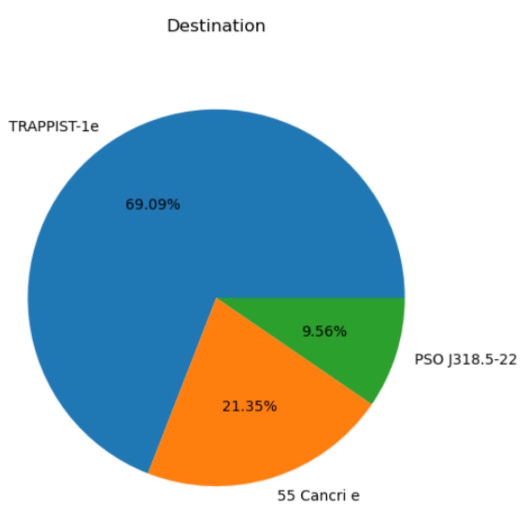
Categorical Features Summary

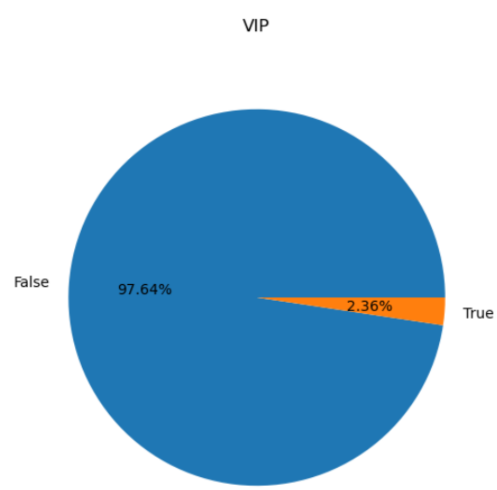
Pandas contains various built-in methods to provide statistics for numerical data, but we can get more valuable data if we plot those statistics ourselves.

```
# Define a list of column names for which we will be creating pie charts
columns = ['HomePlanet', 'CryoSleep', 'Destination',
           'VIP', 'Deck','Side']

# Determine the number of figures needed to display all columns
num_figs = len(columns)
num_cols_per_fig = 2
num_rows_per_fig = num_figs // num_cols_per_fig

# Create empty lists to store the figures and axes
figs = []
axes = []

# Iterate over all the columns and create pie charts for each
for i in range(num_figs):
    # Check if we need to create a new figure
    if i % (num_cols_per_fig * num_rows_per_fig) == 0:
        fig, axs = plt.subplots(num_rows_per_fig, num_cols_per_fig, figsize=(15, 20))
        figs.append(fig)
        axes.append(axs)

    # Calculate the indices of the current row and column in the grid of subplots
    fig_index = i // (num_cols_per_fig * num_rows_per_fig)
    row_index = (i - fig_index * num_cols_per_fig * num_rows_per_fig) // num_cols_per_fig
    col_index = i % num_cols_per_fig

    # Get the current axis object
    axs = axes[fig_index]
    ax = axs[row_index, col_index]

    # Create a pie chart for the current column
    data_counts = df[columns[i]].value_counts()
    ax.pie(data_counts.values, labels=data_counts.index, autopct='%.2f%%')
    ax.set_title(columns[i], y=1.05)
```


HomePlanet
Earth
53.78%
25.48%
20.74%
Mars
Europa


CryoSleep
False
61.99%
38.01%
True


Destination
TRAPPIST-1e
69.09%
9.56%
PSO J318.5-22
21.35%
55 Cancri e


VIP
False
97.64%
2.36%
True

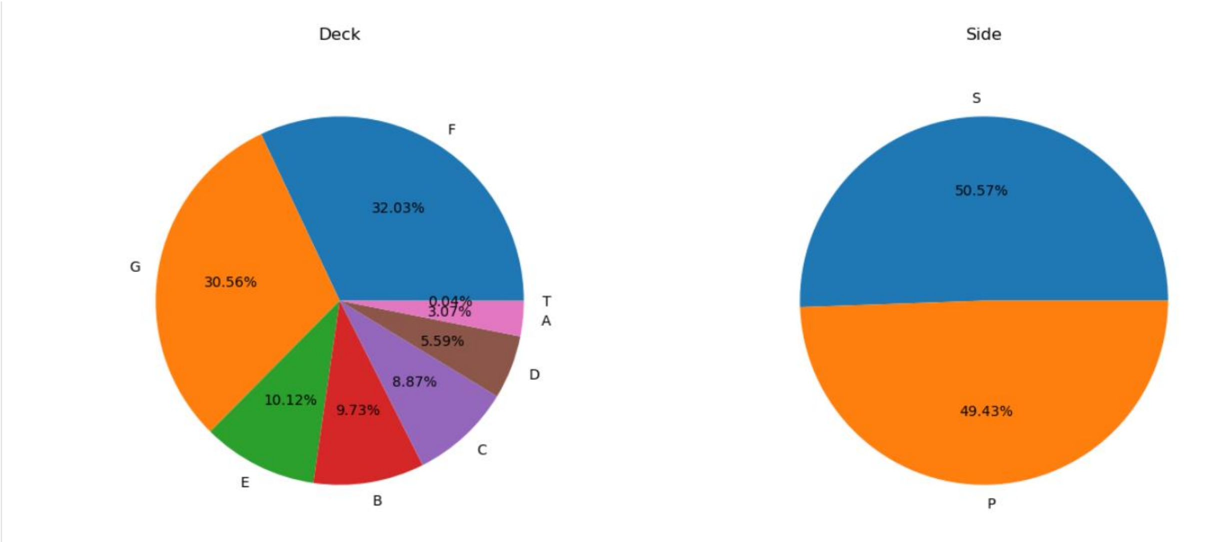


Figure 22 Exploration of Categorical Features

In Figure 22, we used Pandas which includes a number of built-in ways for providing statistics for numerical data shown by six pie charts.

```
# extract the age variable from the dataset
age = df['Age']

# determine the appropriate number of bins for the histogram
num_bins = 20

# create the histogram using matplotlib's hist function
plt.hist(age, bins=num_bins)

# add labels and a title to the histogram
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.title('Distribution of Age among Titanic Passengers')

# display the histogram
plt.show()
```

Figure 23 code for getting chart of Age dataset

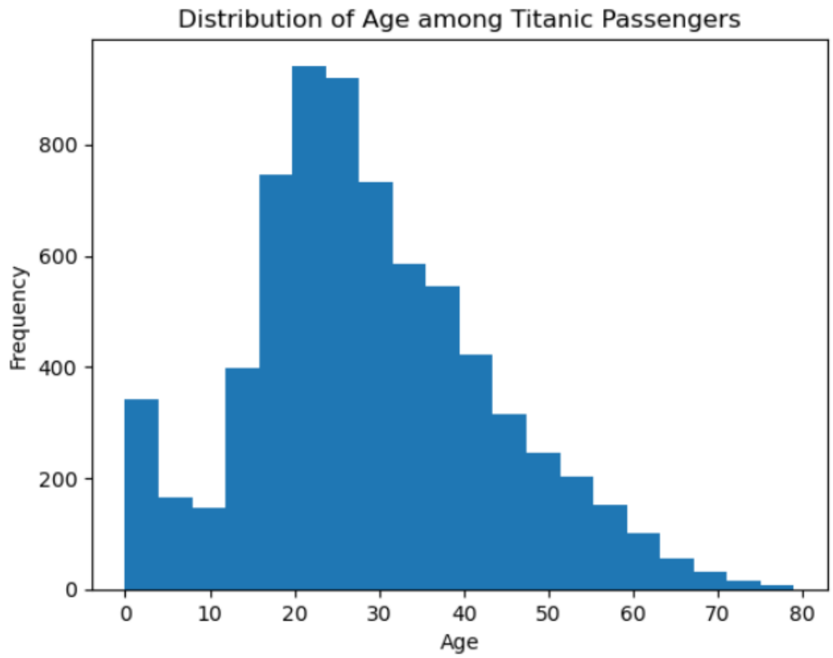


Figure 24 Chart Representing the Age group of passengers

Properties associated with survival

One of the possible explorations is passenger survival. We look over various factors that might influence the passenger's survival.

Problem 1: How is embarking from or traveling to a specific location associated with survival?

First, we explored the number of unique Home Planets and the number of passengers related to each.

```
print(df.groupby('HomePlanet').size())
```

```
HomePlanet
Earth     3803
Europa    1802
Mars      1467
dtype: int64
```

Figure 25 Home planets of passengers

There are three unique Home Planets. We can see that about half of the passengers come from Earth while Europa and Mars have similar numbers of passengers.

```
# Calculate the survival rate for each HomePlanet
survival_rates = df.groupby(['HomePlanet']).mean()['Transported']

# Create a bar chart showing the survival rates for each
plot = sns.barplot(x=survival_rates.index, y=survival_rates.values)

# Set the y-axis limit to 0.7 to better estimate data, add title and label
plot.set(ylim=(0, 0.7), title="Survival Rate by Home Planet", ylabel="Survival Rate" )

print(plot)
```

```
AxesSubplot(0.125,0.11;0.775x0.77)
```

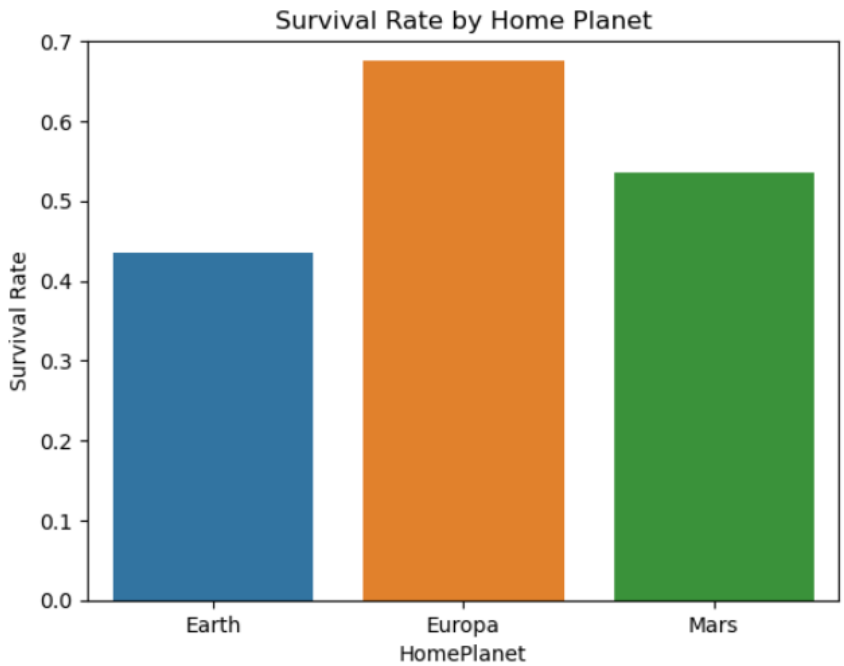


Figure 26 Survival Rate of Each HomePlanet

Here from the figure 26, we can see that the Europa had the highest rate of survival and Earth had the lowest.

```
print(df.groupby('Destination').size())
```

```
Destination
55 Cancri e      1510
PSO J318.5-22     676
TRAPPIST-1e      4886
dtype: int64
```

Figure 27 Destination feature

There are three unique Destinations. We can see that of the passengers are overwhelmingly traveling to TRAPPIST-1e, which is interesting. We can see the plot the survival rates for each of them.

```
# Calculate the survival rate for each Destination
survival_rates = df.groupby(['Destination']).mean()['Transported']

# Create a bar chart showing the survival rates for each
plot = sns.barplot(x=survival_rates.index, y=survival_rates.values)

# Set the y-axis limit to 0.7 to better estimate data, add title and label
plot.set(ylim=(0, 0.7), title="Survival Rate by Destination", ylabel="Survival Rate")

print(plot)
```

```
AxesSubplot(0.125,0.11;0.775x0.77)
```

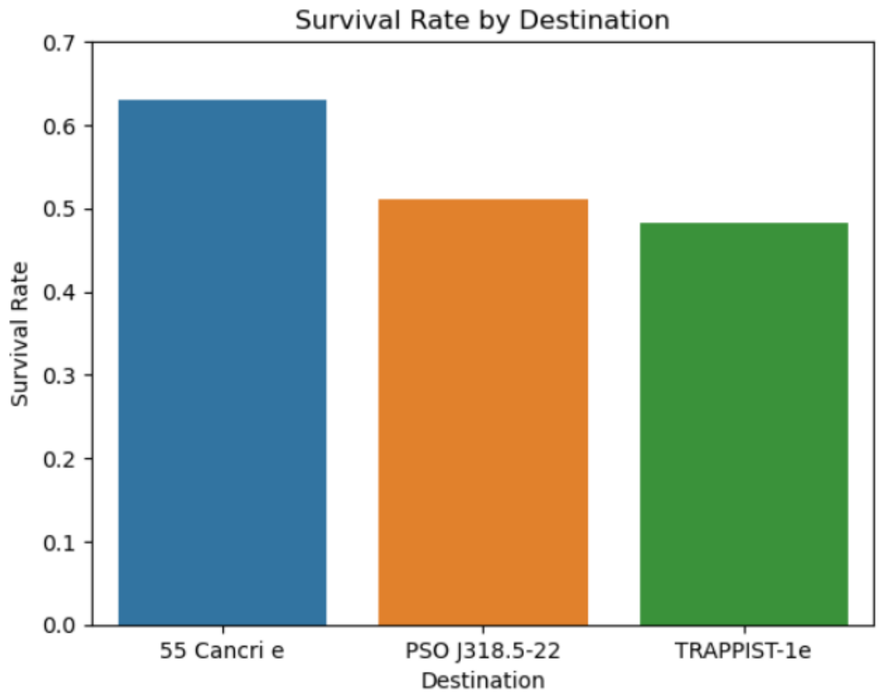


Figure 28 Survival Rate by Destination

Interestingly, TRAPPIST-1e has the lowest survival rate amongst the destinations and was also the most traveled to destination by far, so it seems like there may be some underlying trend there. That said, the range of survival rates is much smaller for Destination than it was for HomePlanet, so Destination may not be as strongly associated with survival as HomePlanet.

Now, as a sort of summary, we can plot the survival rates of HomePlanet/Destination combinations and see if there are any emergent trends.

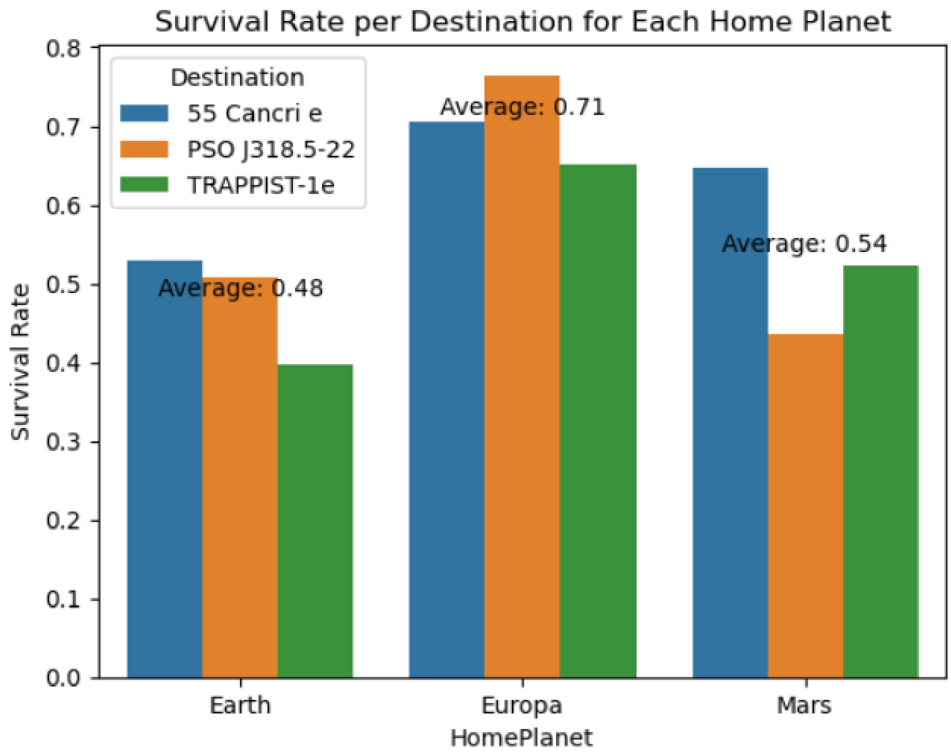


Figure 29 Survival Rate per Destination for each HomePlanet

From figure 29, we can see that nearly 80% passengers from Europa who were traveling to PSO J318.5-22 survived. On the flip side, just under 40% of passengers from Earth traveling to TRAPPIS-1e survived. Additionally, while the mean survival rate for Mars and Earth is roughly 50%, Europa's is much higher at 70%.

As we indicated earlier, the Destination TRAPPIST-1e tends much lower than mean for Earth and Europa, but it's much closer to the average for Mars, where the other two destinations are much more distant from the mean.

Summary: Problem 1

Problem 1 asks "How is embarking from or traveling to a specific location associated with survival?"

We found that the distribution shows that most passengers come from Earth and are headed to TRAPPIST-1e. Passengers from Earth have the lowest survival rate at about 0.44 while passengers from Europa have the highest at about 0.68, so HomePlanet appears to be slightly correlated with survival.

On the flip side, Destination appears to be less strongly linked to survival. Passengers headed to TRAPPIST-1e and PSO J318.5-22 had survival rates of about 0.5 while 55 Cancri e has a rate of about 0.64. It's worth noting that the sample size for Destination is heavily skewed toward TRAPPIST-1e, and less than 8% of passengers were headed to PSO J318.5-22, so conclusions from this data may be of dubious significance.

Problem 2: Is passenger socioeconomic status related to survival?

The data set contains some information we can use to approximate the socioeconomic status of passengers. There are features that contain the amount spent at the amenities aboard the Spaceship Titanic, as well as the VIP feature, which is a boolean containing whether or not the passenger paid extra for the VIP package. Together, these represent a sort of proxy for socioeconomic status.

In order to perform this analysis, we'll need to divide the passengers up by their presumed socioeconomic class. As such, we'll create three categories, lower, middle, and upper to represent the socioeconomic status of passengers.

TotalSpent Feature

As we are not interested in analyzing spending at individual amenities as they relate to survival, we can combine the amenities spending features into a single feature called TotalSpent. Afterward, we can drop the original columns.

```
# Create TotalSpent as a sum of the amenities columns
df['TotalSpent'] = df['RoomService'] + df['FoodCourt'] + df['ShoppingMall'] + df['Spa'] + df['VRDeck']

# Drop amenities columns
df.drop(['RoomService', 'FoodCourt', 'ShoppingMall', 'Spa', 'VRDeck'], axis=1, inplace=True)
```

```
# Verify creation of TotalSpent and dropping of amenities columns
df.head(2)
```

	HomePlanet	CryoSleep	Destination	Age	VIP	Transported	Deck	Side	Group	TotalSpent
0	Europa	False	TRAPPIST-1e	39.0	False	False	B	P	0001	0.0
1	Earth	False	TRAPPIST-1e	24.0	False	True	F	S	0002	736.0

Figure 30 Creating TotalSpent Feature

Now, we used quantiles to differentiate passengers according to their total spending on amenities, as indicated by the newly-created TotalSpent feature. Importantly, passengers in CryoSleep would not be able to spend money at amenities, so they need to be excluded from this analysis first before going further. We'll create a subset of the data containing only passengers not in CryoSleep for this part.

```
# Get a subset of the data containing only passengers not in CryoSleep
awake = df[df['CryoSleep'] == False].copy()
print(f"Number of passengers not in CryoSleep: {awake.shape[0]}")
awake.head(1)
```

```
lumber of passengers not in CryoSleep: 4384
```

	HomePlanet	CryoSleep	Destination	Age	VIP	Transported	Deck	Side	Group	TotalSpent
0	Europa	False	TRAPPIST-1e	39.0	False	False	B	P	0001	0.0

Figure 31 Creating passengers not in CryoSleep

In Figure 31, we can see that we have created a subset of the data which contains only passengers but which are not in CryoSleep.

```python
# Plot histogram of passenger spending
sns.histplot(awake['TotalSpent'])
```

```
<AxesSubplot: xlabel='TotalSpent', ylabel='Count'>
```

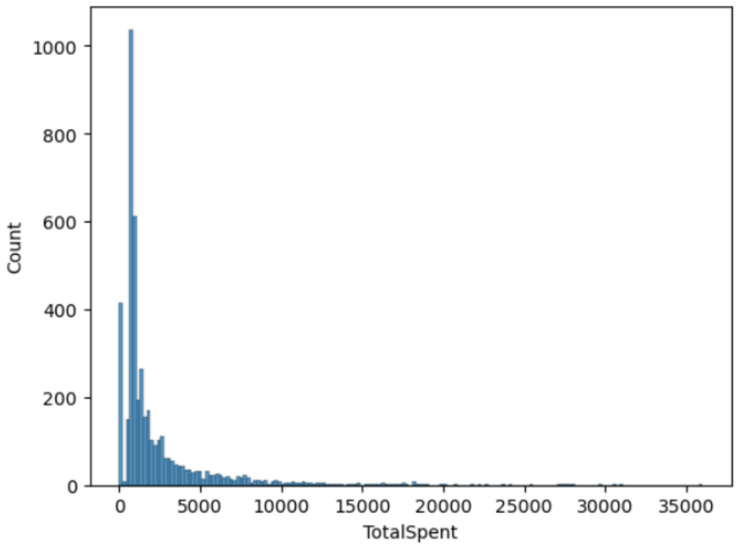


Figure 32 Passengers Spending

It appears that most passengers spend less than about 2,500, but there are still quite a few that spend a bit more (and a few that spend quite a bit more). Let's get more detail about the quartiles using df.describe

```python
awake['TotalSpent'].describe()
```

```
count     4384.000000
mean      2319.061131
std       3294.501584
min          0.000000
25%        761.750000
50%       1051.000000
75%       2507.750000
max      35987.000000
Name: TotalSpent, dtype: float64
```

Figure 33 Total Spent Description

We can see from the table above that 50th percentile is still a bit far from the feature's mean, but it does jump up quite a bit by the 75th percentile. Even still, the presence of large outliers is apparent, as the maximum value is orders of magnitude higher than the mean. Let's see what the data looks like without the top 10% of spenders

```
# Exclude the top 10% of spenders
subset = awake.where(awake['TotalSpent'] < awake['TotalSpent'].quantile(0.90))
subset.describe()
```

	Age	TotalSpent
count	3945.000000	3945.000000
mean	29.094550	1439.632446
std	14.161895	1226.483888
min	0.000000	0.000000
25%	20.000000	738.000000
50%	27.000000	923.000000
75%	38.000000	1822.000000
max	79.000000	5746.000000

Figure 34 Excluding top 10% of spenders

We can determine the quantiles we will use for our class thresholds from the table above:

Lower class: TotalSpent <= 800 (<30%)

Middle class: 800 < TotalSpent < 2500 (~31-75%)

Upper Class: TotalSpent >= 2000 (>75%)

We can use this information to make a new feature,

```
# Define the conditions and class labels
conditions = [
    awake['TotalSpent'] <= 800,
    (awake['TotalSpent'] > 800) & (awake['TotalSpent'] < 2500),
    awake['TotalSpent'] >= 2500
]
class_labels = ['Lower', 'Middle', 'Upper']

# Create a new column 'Class' based on the conditions and labels
awake['Class'] = np.select(conditions, class_labels, default='Unknown')
```

```
# Verify Class was created
awake.head()
```

	HomePlanet	CryoSleep	Destination	Age	VIP	Transported	Deck	Side	Group	TotalSpent	Class
0	Europa	False	TRAPPIST-1e	39.0	False	False	B	P	0001	0.0	Lower
1	Earth	False	TRAPPIST-1e	24.0	False	True	F	S	0002	736.0	Lower
2	Europa	False	TRAPPIST-1e	58.0	True	False	A	S	0003	10383.0	Upper
3	Europa	False	TRAPPIST-1e	33.0	False	False	A	S	0003	5176.0	Upper
4	Earth	False	TRAPPIST-1e	16.0	False	True	F	S	0004	1091.0	Middle

Figure 35 Creating a new class

It's important to keep in mind that passengers can also purchase a VIP package, which we can use as a modifier for our determined classes. Before we do that, we can explore this data independently.

```
# Calculate the survival rate for each Class
survival_rates = awake.groupby(['Class']).mean()['Transported']

# Create a bar chart showing the survival rates for each
plot = sns.barplot(x=survival_rates.index, y=survival_rates.values)

# Set the y-axis limit to 0.7 to better estimate data, add title and label
plot.set(title="Survival Rate by Class", ylabel="Survival Rate")

print(plot)
```

```
AxesSubplot(0.125,0.11;0.775x0.77)
```

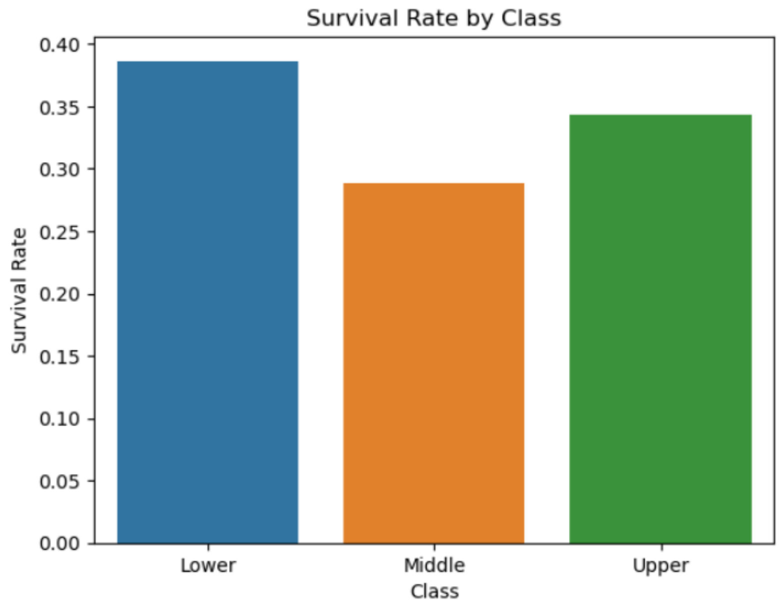


Figure 36 Survival Rate by Class

Interestingly, it appears that all three classes have a survival rate roughly between 0.3 and 0.4. In order to do so, we are going to establish the criteria that purchasing the VIP package automatically excludes the passenger from the lower-class tier. We'll also need to double check that no passengers in CryoSleep paid for the VIP package.

Problem 2: Summary

Question 2 asks "Is passenger socioeconomic status related to survival?" The data set provides a few features we can use to estimate the socioeconomic status of passengers. After combining the amenities spending into a single feature called TotalSpent, a Class feature was created containing the approximate socioeconomic class of the passenger.

Ultimately, socioeconomic status does not track closely with passenger survival. Based on the information above, it appears that passenger spending is not a strong predictor of passenger survival. However, this analysis could necessarily only be performed on passenger not in CryoSleep

Problem 3: Is CryoSleep related to survival?

Next, we will explore the effect of another important feature on survival: CryoSleep. As mentioned previously, CryoSleep is a boolean indicating whether or not the passenger elected to be frozen for the duration of the trip. This is a unique feature in general, but it's nature also means its effect on survival is hard to hypothesize. Naturally, this is definitely the type of feature worth exploring.

We'll start by getting a basic count:

```
df['CryoSleep'].value_counts()
```

```
False    4384
True     2688
Name: CryoSleep, dtype: int64
```

Figure 37 Getting basic count

Given this information, it appears there may be enough passengers where CryoSleep == True to perform some meaningful analysis. We can check the survival rates for each group:

```
# Calculate the survival rate for each CryoSleep status
survival_rates = df.groupby(['CryoSleep'])['Transported'].mean()

# Create a bar chart showing the survival rates for each
plot = sns.barplot(x=survival_rates.index, y=survival_rates.values)

# Set the y-axis limit to 0.7 to better estimate data, add title and label
plot.set(title="Survival Rate by CryoSleep Status", ylabel="Survival Rate")

print(plot)
```

```
AxesSubplot(0.125,0.11;0.775x0.77)
```

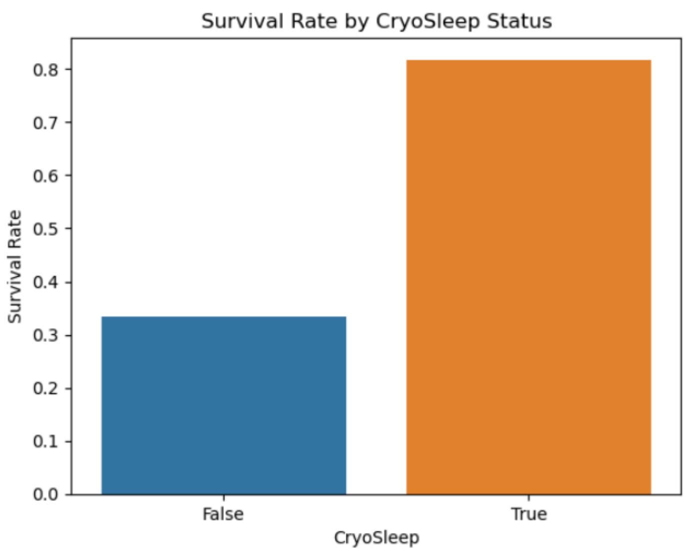


Figure 38 Survival rate by CryoSleep Status

The graph above shows that about 80% of CryoSleep passengers survived while only about 35% of passenger not in CryoSleep survived.

Problem 3: Summary

Question 3 asks "Is CryoSleep related to survival?" Of the 7,072 passengers in the data set, 2,688 of them were in CryoSleep (about 30% of the data set). Clearly, the difference in survival rates between these two categories is quite dramatic. Roughly four out five passengers in CryoSleep survived while only about 35% of passengers not in CryoSleep survived. As such, it appears that CryoSleep is likely a good predictor of passenger survival.

Problem 4: Does cabin location (Deck, Side) correlate to Survival?

Curiously, the point at which the anomaly came into contact with the Space Titanic wasn't reported. Assuming that proximity to the anomaly meant a higher chance of being Transported, by answering this question we not only gain insight into the location of impact but also can determine whether passengers on a certain Side or a specific Deck were more likely to be Transported than others. We can accomplish this by using the value_counts() function on our Sides feature.

```
df['Side'].value_counts()
```

```
S    3576
P    3496
Name: Side, dtype: int64
```

Figure 39 Counting side, Deck of the Ship

As expected, there are roughly the same number of passengers on each side of the ship. This works to our advantage since the data would be skewed if there was a larger difference in these values, though the difference should be noted, nonetheless.

```
In [47]:
# Calculate the survival rate for each Side of the Ship
survival_rates = df.groupby(['Side'])['Transported'].mean()

# Create a bar chart showing the survival rates for each
plot = sns.barplot(x=survival_rates.index, y=survival_rates.values)

# Add title and label y-axis
plot.set(title="Survival Rate by Side", ylabel="Survival Rate")

print(plot)
```

```
AxesSubplot(0.125,0.11;0.775x0.77)
```

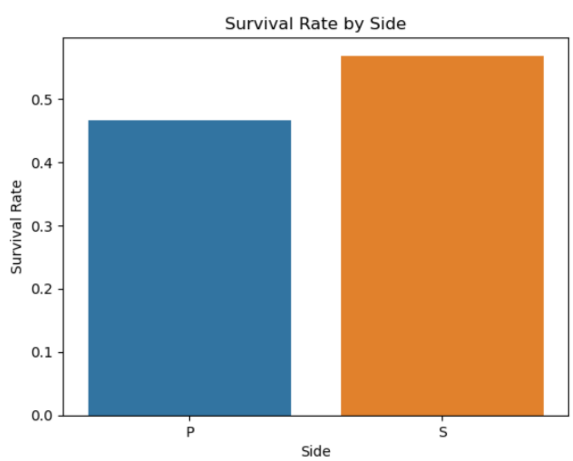


Figure 40 Survival by Side

Based on our results, it appears that passengers on the Port Side were more likely to be transported than those on the Starboard Side of the ship. Now let's find if a certain Deck was more affected than the others.

```
df['Deck'].value_counts()
```

```
F    2265
G    2161
E     716
B     688
C     627
D     395
A     217
T       3
Name: Deck, dtype: int64
```

Figure 41 Deck Comparison

Unfortunately, there is a much greater variation in passenger numbers between Decks than there was for the Sides.

```
df['Deck'].value_counts().describe()
```

```
count       8.000000
mean      884.000000
std       856.133334
min         3.000000
25%       350.500000
50%       657.500000
75%      1077.250000
max      2265.000000
Name: Deck, dtype: float64
```

Figure 42 Detail analysis of Deck

Regardless we can still plot the data and see if a pattern appears.

```
# Calculate the survival rate for each Deck
survival_rates = df.groupby(['Deck'])['Transported'].mean()

# Create a bar chart showing the survival rates for each
plot = sns.barplot(x=survival_rates.index, y=survival_rates.values)

# Add title and label y-axis
plot.set(title="Survival Rate by Deck", ylabel="Survival Rate")

print(plot)
```

```
AxesSubplot(0.125,0.11;0.775x0.77)
```

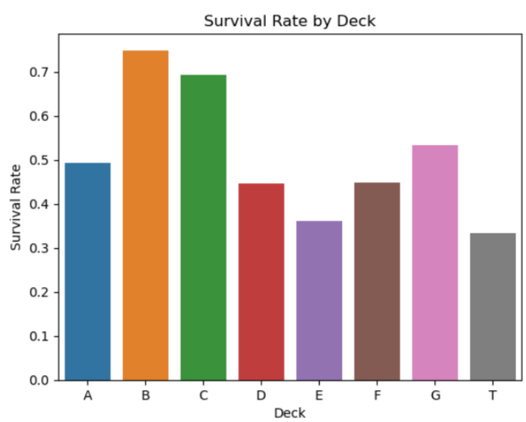


Figure 43 Survival Rate by Deck

It appears that the `Decks` with the lowest survival rates are the E and T `Decks`. However, seeing as the T Deck only has 3 passengers it is definitely an outlier, and we can safely exclude it from our analysis. The E `Deck` on the other hand not only has the third largest number of passengers, it also has a similar number of passengers as B and C `Deck`. I think that it can be reasonably concluded that it was the most affected by the anomaly. Another thing to note is that the B and C `Decks` had the highest survival rates.

Problem 4 Summary

Question 4 asks whether or not the location of the passenger's cabin is related to their survival. Without a clear map of the ship's layout, it would be difficult to pinpoint the point of impact with the anomaly. From our plotting, we were able to see that the Port `Side` had a lower survival rate. While `Deck` E appears to be the most impacted, A, D, and F all had survival rates below 50%. `Deck` G fared a bit better with a roughly 55% survival rate. The only thing that is clear from this analysis is that the survival rates for `Decks` B and C were much higher than those of the others. It could be possible that `Decks` B and C were the point of impact and the anomaly had a wide range of effect around them instead of directly that point. Further information would be required regarding the layout of the ship and possible other factors such as these decks being reserved for individuals in CryoSleep before any definite conclusions could be drawn.

Passenger Survival Summary

In this part, we explored the relationship between several features and passenger survival. An analysis of Destination and HomePlanet showed that HomePlanet is likely a stronger predictor of Transported status than Destination. Interestingly, TRAPPIST-1e had the lowest survival rate despite being the most popular destination.

When examining socioeconomic status, the survival rates were relatively similar across different socioeconomic classes. As such, no clear correlation was found between TotalSpent/Class (out proxy for socioeconomic status) and Transported status.

The analysis also revealed that passengers in CryoSleep had a significantly higher survival rate compared to those not in CryoSleep, suggesting that CryoSleep was a reliable predictor of survival.

Similarly, while the sample sizes leave a bit to be desired, Deck may be somewhat strongly correlated to Transported, while the effect of Side is less pronounced.

Recommendations

As performing a logistic regression analysis is more Machine Learning and prediction based so we would've needed to convert all the categorical values to integers so that the model could understand it and then feedback in the test.csv to see how accurately our model could predict survivability. That's the actual goal of the challenge on Kaggle. It should be done for the analysis but since, we only wanted to explore and find out various relationships between the features and perform analysis. The recommendation for future is to perform logistic regression analysis.

Future Work

As, the scope of this project was exploratory in nature, so there are certainly more insights to be gleaned from the data set. For example, more features could be extracted from the data set. The group portion of PassengerId and Name could be used to create a feature exploring the number of family members a passenger has on board. Similarly, further analyzing the Group feature as it relates to survival could be an interesting area of study.

Additionally, the analysis presented here would greatly benefit from employment of more advance statistical methods. For example, since determining Transported for a given passenger is a binary problem, logistic regression may be used to determine the statistical significance of features as they relate to Transported (i.e., survival).

Conclusion

In the sections above, we explored the Spaceship Titanic data set provided by Kaggle. The aim of the project was to explore the data set, but to also perform rudimentary analysis with regard to passenger survival (as indicated by Transported status).

Before moving on to the analysis, however, we performed some data cleaning operations such as dropping NaN values from the data. We extracted the Group, Deck, Side, and TotalSpent features, dropping Cabin, VIP, PassengerId, Name, and the amenities spending features in the process. We then analyzed the distribution of categorical and numerical features before proceeding to survival analysis.

Ultimately, the survival analysis focused on four questions, attempting to determine a correlation between Transported and each of the following features: HomePlanet, Destination, TotalSpent, CryoSleep, Deck, and Side (we also plotted survival rates by VIP status as part of our VIP exploration). Of these features, we found that CryoSleep had the strongest relationship to Transported while Deck might also be a strong indicator of survival. Further analysis of the other features will be required to draw more meaningful conclusions with respect to those features' effects on Transported status.

Clearly, there are multiple factors that influence passenger survival, and this analysis did not focus on the combinative effect of features on passenger survival. This and other directions for future research are discussed in the next section.

References

I. Howard, A., Chow, A., & Holbrook, R. (n.d.). *Spaceship Titanic*. Kaggle. Retrieved May 7, 2023, from https://www.kaggle.com/c/spaceship-titanic

II. *Data Cleaning Time has come: Make your business clearer*. Dataconomy. (2022, April 11). Retrieved May 7, 2023, from https://dataconomy.com/2022/04/11/what-is-data-cleaning-how-to-clean-6-steps/

III. Pandas. (2023, April 24). *User guide#*. User Guide - pandas 2.0.1 documentation. Retrieved May 7, 2023, from https://pandas.pydata.org/docs/user_guide/index.html#user-guide

IV. Oliphant, T. E. (2006). *A guide to NumPy* (Vol. 1, p. 85). USA: Trelgol Publishing.

V. J. D. Hunter, "Matplotlib: A 2D Graphics Environment," in Computing in Science & Engineering, vol. 9, no. 3, pp. 90-95, May-June 2007, doi: 10.1109/MCSE.2007.55.

VI. Reiss, F., Cutler, B., & Eichenberger, Z. (2021). Natural language processing with pandas dataframes. In *Proc. Of The 20th Python In Science Conf.(Scipy 2021)* (pp. 49-58).

VII. Eaton, D. A. (2020). Toytree: A minimalist tree visualization and manipulation library for Python. *Methods in Ecology and Evolution*, *11*(1), 187-191.

VIII. Brownlee, J. (2020). *Data preparation for machine learning: data cleaning, feature selection, and data transforms in Python*. Machine Learning Mastery.

IX. McKinney, W. (2012). *Python for data analysis: Data wrangling with Pandas, NumPy, and IPython*. " O'Reilly Media, Inc.".