```
#  1. Creating a DataFrame

import pandas as pd

# Create a DataFrame from a dictionary
data = {
    'Name': ['Alice', 'Bob', 'Charlie'],
    'Age': [25, 30, 35],
    'City': ['New York', 'Los Angeles', 'Chicago']
}
df = pd.DataFrame(data)

print(df)
```

```
      Name  Age         City
0    Alice   25     New York
1      Bob   30  Los Angeles
2  Charlie   35      Chicago
```

```
# 2. Access by Index Label Using loc
# The loc accessor allows you to access rows by their label or index name. This is useful if you have a custom index
# Set a custom index
df.set_index('Name', inplace=True)

# Access the row with the index 'Alice'
alice_row = df.loc['Alice']

#print(alice_row)
print(df)
```

```
          Age         City
Name
Alice      25     New York
Bob        30  Los Angeles
Charlie    35      Chicago
```

```
#3. Access Multiple Rows by Position or Label
#You can also access multiple rows at once using both iloc and loc.
# Access the first two rows by position
first_two_rows = df.iloc[0:2]
print(first_two_rows)
# Access rows by index labels
rows = df.loc[['Alice', 'Bob']]
print(rows)
```

```
        Age         City
Name
Alice    25     New York
Bob      30  Los Angeles
        Age         City
Name
Alice    25     New York
Bob      30  Los Angeles
```

```
# 4. Access Rows by Boolean Indexing
# Boolean indexing allows you to filter rows based on a condition.
# Filter rows where Age is greater than 25
filtered_rows = df[df['Age'] > 25]
print(filtered_rows)
```

```
          Age         City
Name
Bob        30  Los Angeles
Charlie    35      Chicago
```

```
#5. Access Rows Using query Method
# The query method allows you to access rows based on a query string.
# Use query to filter rows where Age is greater than 25
queried_rows = df.query('Age > 25')
print(queried_rows)
```

```
          Age         City
Name
Bob        30  Los Angeles
Charlie    35      Chicago
```

```
#6. Access Rows with at and iat
# The at and iat accessors are useful for getting a single value from a DataFrame, but you can use them to access specific rows and colu
# Access the value at the first row and 'Age' column
value_at = df.at['Alice', 'Age']  # When using a custom index
value_at_position = df.iat[0, 1]  # Using integer position
print(value_at)
print(value_at_position)
```

```
25
New York
```

Let's use a more complex dataset, such as the Titanic dataset, which includes various types of data such as numerical, categorical, and text. This dataset is commonly used for demonstration purposes and is available through several sources, including Kaggle and the Seaborn library.

Steps:

Load Data

Inspect Data

Clean Data

Feature Engineering

Normalize/Standardize Data

Split Data

```
import pandas as pd

# Load the Titanic dataset from seaborn
import seaborn as sns
df = pd.read_csv('/content/titanic.csv')
```

```
print(df.head())
```

```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0      1
2                             Heikkinen, Miss. Laina  female  26.0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                           Allen, Mr. William Henry    male  35.0      0

   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
3      0            113803  53.1000  C123        S
4      0            373450   8.0500   NaN        S
```

```
print(df.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
None
```

```
df.describe()
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

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

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

The columns 'Age' and 'Cabin' contains more null values.

'Survived' is the target column/variable.

'PassengerId', 'Name' and 'Ticket' doesn't contribute to the target variable

'Survived'. So, we can remove it from the data.

'Age' and 'Embarked' has less number of missing value. We have to impute them using different techniques.

As there are a lot of missing values in the column 'Cabin', we can remove it from the training data.

'Pclass', 'Sex', 'SibSp', 'Parch', 'Fare' doesn't have any missing values.

We can also create new variable like 'total size of the family' from the columns 'SibSp' and 'Parch'.

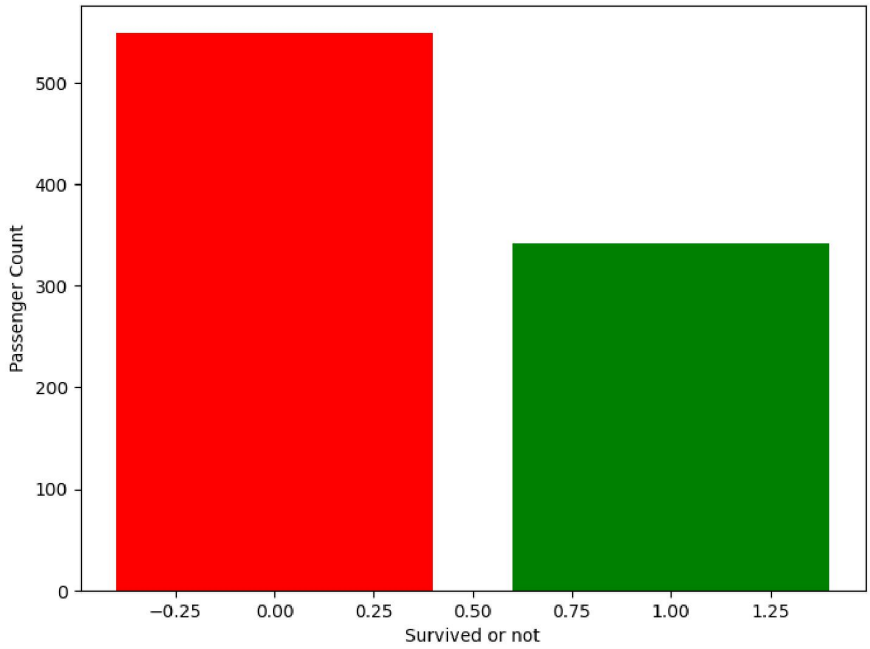
```
survival_counts = df['Survived'].value_counts()
survival_counts
```

Survived	count
0	549
1	342

dtype: int64

```
import matplotlib.pyplot as plt
plt.figure(figsize=(8, 6))
plt.bar(survival_counts.index, survival_counts.values, color=['red', 'green'])
plt.xlabel('Survived or not')
plt.ylabel('Passenger Count')
```

```
Text(0, 0.5, 'Passenger Count')
```



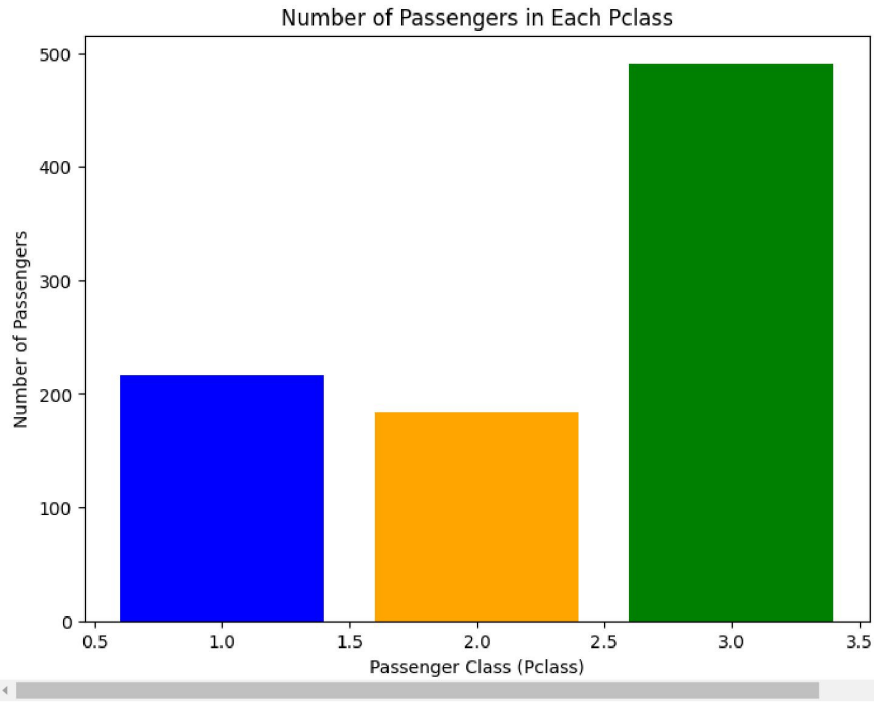

```
Survived=df[['Pclass', 'Survived']].groupby('Pclass').count()
Survived
```

Pclass	Survived
1	216
2	184
3	491

```
plt.figure(figsize=(8, 6))
plt.bar(Survived.index, Survived['Survived'], color=['blue', 'orange', 'green'])

# Adding title and labels
plt.title('Number of Passengers in Each Pclass')
plt.xlabel('Passenger Class (Pclass)')
plt.ylabel('Number of Passengers')
```

```
Text(0, 0.5, 'Number of Passengers')
```



```
df[['Pclass', 'Survived']].groupby('Pclass').sum()
```

Pclass	Survived
1	136
2	87
3	119

```
prob=df[['Pclass', 'Survived']].groupby('Pclass').mean()
prob
```

Pclass	Survived
1	0.629630
2	0.472826
3	0.242363

```
df.Sex.value_counts().sort_index()
```

Sex	count
female	314
male	577

dtype: int64

```
probs=df[['Sex', 'Survived']].groupby('Sex').mean()
probs
```

Sex	Survived
female	0.742038
male	0.188908

```
#We can remove 'Ticket' and 'PassengerId', as they don't contribute to target class.
#Remove 'Cabin' as it has a lot of missing values in both train and test data
df.drop(columns=['Ticket', 'PassengerId', 'Cabin'], inplace=True)
df.head()
```

	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Fare	Embarked
0	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	7.2500	S
1	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	71.2833	C
2	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	7.9250	S
3	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	53.1000	S
4	0	3	Allen, Mr. William Henry	male	35.0	0	0	8.0500	S

```
#fill missing values
# Fill missing values for 'age' with the mean
df['Age'].fillna(df['Age'].mean(), inplace=True)


# Fill missing values for 'embark_town' with the most frequent value
df['Embarked'].fillna(df['Embarked'].mode()[0], inplace=True)


df.drop_duplicates(inplace=True)


df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 9 columns):
 #   Column    Non-Null Count  Dtype
```

```
 ---  ------    --------------  -----
  0   Survived  891 non-null    int64
  1   Pclass    891 non-null    int64
  2   Name      891 non-null    object
  3   Sex       891 non-null    object
  4   Age       891 non-null    float64
  5   SibSp     891 non-null    int64
  6   Parch     891 non-null    int64
  7   Fare      891 non-null    float64
  8   Embarked  891 non-null    object
dtypes: float64(2), int64(4), object(3)
memory usage: 62.8+ KB
```

```
df['Title'] = df.Name.str.extract(' ([A-Za-z]+)\.', expand=False)
```

```
df = df.drop(columns='Name')
```

```
df['Title'] = df['Title'].replace(['Dr', 'Rev', 'Col', 'Major', 'Countess', 'Sir', 'Jonkheer', 'Lady', 'Capt', 'Don'], 'Others')
df['Title'] = df['Title'].replace('Ms', 'Miss')
df['Title'] = df['Title'].replace('Mme', 'Mrs')
df['Title'] = df['Title'].replace('Mlle', 'Miss')
df.Title.value_counts()
```

Title	count
Mr	517
Miss	185
Mrs	126
Master	40
Others	23

dtype: int64

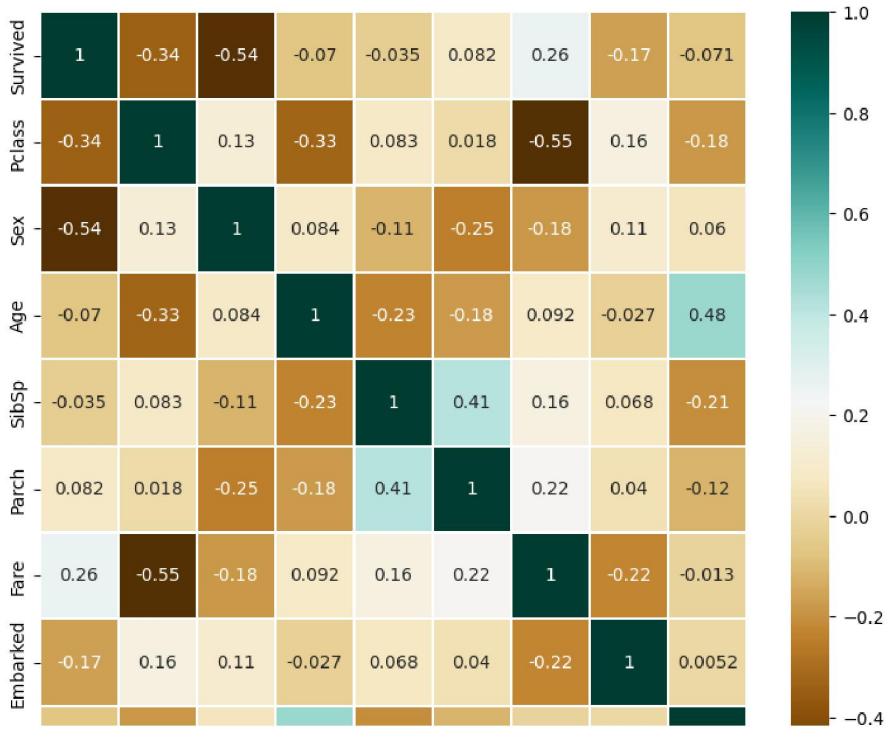
```
#use map or label encoder
#train_data['Title'] = train_data['Title'].map({'Master':0, 'Miss':1, 'Mr':2, 'Mrs':3, 'Others':4})
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
# Convert 'Sex' column to numerical
df['Sex'] = le.fit_transform(df['Sex'])
df['Embarked'] = le.fit_transform(df['Embarked'])
df['Title'] = le.fit_transform(df['Title'])

print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 9 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Survived  891 non-null    int64
 1   Pclass    891 non-null    int64
 2   Sex       891 non-null    int64
 3   Age       891 non-null    float64
 4   SibSp     891 non-null    int64
 5   Parch     891 non-null    int64
 6   Fare      891 non-null    float64
 7   Embarked  891 non-null    int64
 8   Title     891 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 62.8 KB
None
```

```
corr_matrix = df.corr()
plt.figure(figsize=(9, 8))
sns.heatmap(data = corr_matrix,cmap='BrBG', annot=True, linewidths=0.2)
```

<Axes: >


Survived
Pclass
Sex
Age
SibSp
Parch
Fare
Embarked
1 -0.34 -0.54 -0.07 -0.035 0.082 0.26 -0.17 -0.071
-0.34 1 0.13 -0.33 0.083 0.018 -0.55 0.16 -0.18
-0.54 0.13 1 0.084 -0.11 -0.25 -0.18 0.11 0.06
-0.07 -0.33 0.084 1 -0.23 -0.18 0.092 -0.027 0.48
-0.035 0.083 -0.11 -0.23 1 0.41 0.16 0.068 -0.21
0.082 0.018 -0.25 -0.18 0.41 1 0.22 0.04 -0.12
0.26 -0.55 -0.18 0.092 0.16 0.22 1 -0.22 -0.013
-0.17 0.16 0.11 -0.027 0.068 0.04 -0.22 1 0.0052
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4