Vibhu Vanjari Module 4 Assignment 1 – Titanic

Code for this assignment can be found here: LINK1 LINK2

Data preparation:

Cabin – although ultimately I decided to drop this variable since imputation was not possible, I realized that the first letter of the Cabin corresponded to the deck of the ship which was for the most part assigned by Pclass was an interesting observation.

I broke down the Ticket feature into Ticket_Number and Ticket_Group. Ticket_Group was based on the alphanumeric characteristics of the ticket itself to find various patterns.

I also extracted the "Title" out of the "Name" feature, for example Mr, Miss, Mrs, Master, etc. this variable in combination with the Age variable was quite helpful later on in helping split SibSp to Siblings and Spouse variables and split Parch into Parents and Children variables.

Predicting missing data:

One passenger had missing fare data, upon analysis of similar Pclass and Embarkation ports, a fare of $8.05 was predicted for PassengerId 1044. Two passengers had missing Embarked data, this was a bit more complex but ultimately I chose to set their embarkation port to C for PassengerId 62 and 830. This was based on similar fares in that Pclass and at a similar Fare level.

The most complex prediction was predicting missing Age data. The process I took as is as follows: 1) build a regression model (Regression 4) to predict Age without using newly created Sibling, Spouse, Parents and Children variables, 2) use that model to estimate a

preliminary age, 3) use the estimated age to predict Sibling, Spouse, Parents and Children variables and 4) use a more comprehensive model (Regression 3) to predict Age.

Predicting survival:

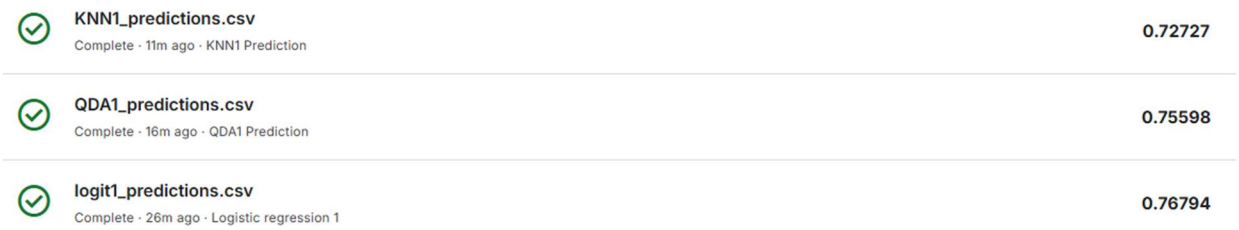
I built 3 models a logistic regression, QDA model and a KNN model. Through trial and error I picked the set of variables that gave me the best p-values on the logistic regression and continued to use those variables across the other two models.

The features that were most predictive were Sex and Pclass. Family characteristics did play a role in survival rate as well but it was quite complex to model. Having a spouse on the ship didn't seem to matter but having Siblings or Parents was quite significant predictor. The table below compares fit metrics and AUC across the three different models used.

Model	Accuracy	Precision	Recall / TPR	FPR	AUC	Kaggle Score
Logistic Regression	81.4%	79.0%	70.5%	11.7%	0.86	0.76794
QDA	81.9%	77.0%	75.4%	14.0%	0.86	0.75598
KNN	86.0%	87.8%	73.7%	6.4%	0.91	0.72727

It's quite clear from this that the logistic regression does the best job generalizing. Since I used the same variables it's a nice way to compare apples-to-apples. The KNN is particularly overfitting the training data and performs significantly worse on the test data even with higher accuracy, precision and AUC. Simpler models like Logistic Regression,

even if they are slightly less accurate on training data, can perform better on new, unseen passengers.



Introduction

Link to access this code - https://colab.research.google.com/drive/1alhR1osGKoX7YIvuWhXdu1Gcpa0RPhid

Data taken from - https://www.kaggle.com/c/titanic/data

Part 2: https://colab.research.google.com/drive/1OhL-lseBrsHbzX33VEzRQuRTYuQXHu5s#scrollTo=9HWrWlpE3gFQ

Import modules and data files

[ ] ↳ 5 cells hidden

EDA

Establishing base survival rates across different features

[ ] ↳ 4 cells hidden

Cabin

TAKEAWAY: There is not much information the variable "Cabin" gives us in addition to Pclass. The deck level doesn't or room number doesn't neccessarily give us much information on survival rate. There is a lot of missing samples here so it may be best to eliminate this variable.

```
# Determine if Cabin has importance in predicting survival
df_train_original['Cabin'].value_counts()

# Group by 'Cabin' and 'Survived', then count
cabin_survival_counts = df_train_original.groupby(['Cabin', 'Survived']).size().unstack(fill_value=0)

# Rename the columns for clarity
cabin_survival_counts.columns = ['Survived_0', 'Survived_1']

# Reset index for a clean table
cabin_survival_counts = cabin_survival_counts.reset_index()

print(cabin_survival_counts)
```

Show hidden output

No clear pattern in cabin names versus survival.

Further research about the Titanic's layout gave me the following information:

- The letter indicated the deck level.
  - First class was located on the upper decks: A Deck (Promenade), B Deck (Bridge), C Deck (Shelter), some on D Deck (Saloon).
  - Second class was located mainly on D Deck and E Deck (midship, closer to the center for stability).
  - Third class was located lower down on F Deck and G Deck.
  - Many Third Class passengers had no cabin number listed, they were assigned general spaces. This explains the large amount of missing cabin data.
- The number suggested location along the length of the ship but it wasn't strictly sequential or geographic.
  - Lower numbers were usually further forward (closer to the bow/front).
  - Higher numbers were usually toward the back (stern).

```
# Group by 'Pclass' and count missing 'Cabin' values
cabin_missing_by_pclass = df_train_original.groupby('Pclass')['Cabin'].apply(lambda x: x.isnull().sum()).reset_index(name='Cabin_Missing_Count')

# Calculate total people in each Pclass
pclass_counts = df_train_original['Pclass'].value_counts().sort_index().reset_index()
pclass_counts.columns = ['Pclass', 'Total_People']

# Merge the two DataFrames
cabin_missing_by_pclass = pd.merge(cabin_missing_by_pclass, pclass_counts, on='Pclass')

# Calculate the ratio
cabin_missing_by_pclass['Missing_Ratio'] = cabin_missing_by_pclass['Cabin_Missing_Count'] / cabin_missing_by_pclass['Total_People']

# Display the results
print(cabin_missing_by_pclass)
```

```
   Pclass  Cabin_Missing_Count  Total_People  Missing_Ratio
0       1                   40           216       0.185185
1       2                  168           184       0.913043
2       3                  479           491       0.975560
```

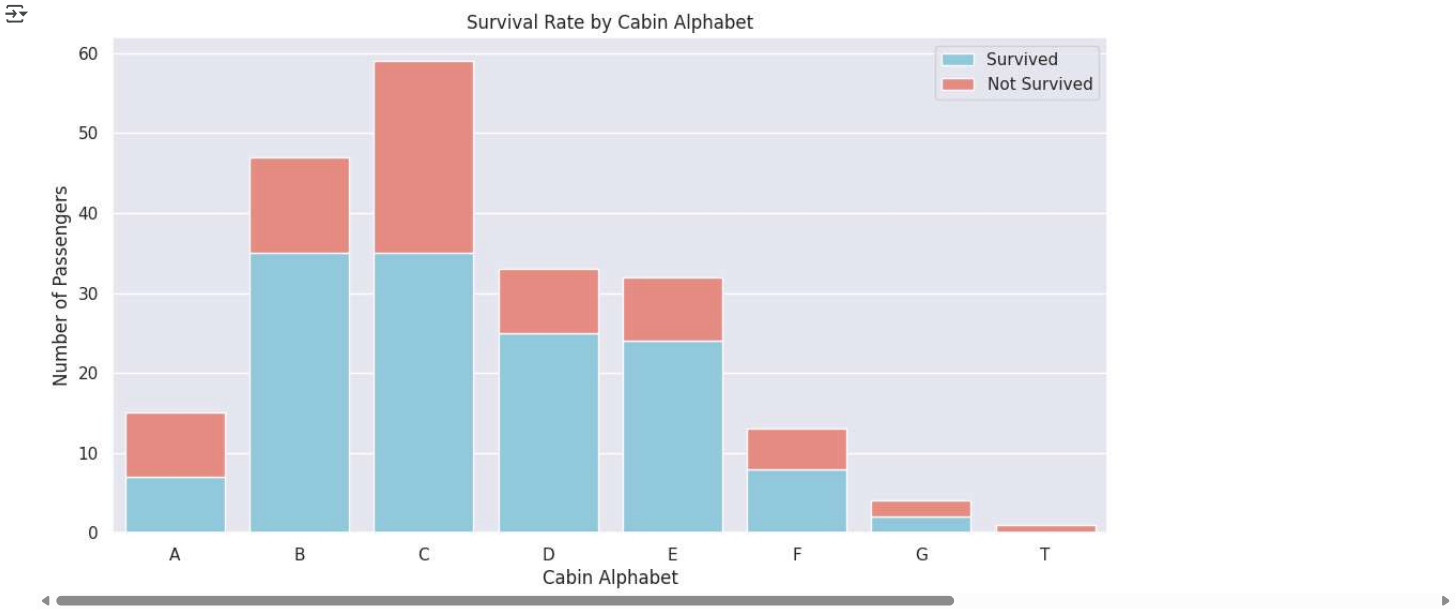
From the data it is not possible to conclude everyone with missing Cabin data travelled in third class.

```
# Split cabin data into two parts, alphabet data and numeric data
df_train_original['Cabin_Alphabet'] = df_train_original['Cabin'].str.extract(r'([A-Za-z]+)')
df_train_original['Cabin_Number'] = df_train_original['Cabin'].str.extract(r'(\d+)')

# Convert 'Cabin_Number' to numeric, handling errors
```

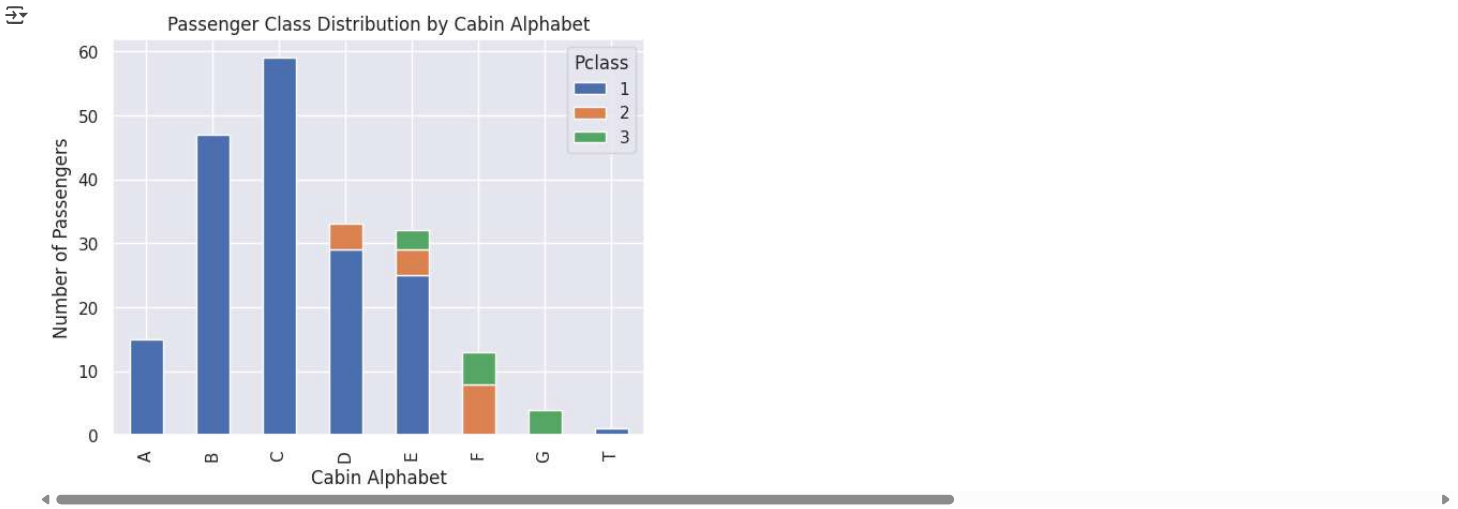
```
df_train_original['Cabin_Number'] = pd.to_numeric(df_train_original['Cabin_Number'], errors='coerce')
df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 13 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
 12  TestYes      891 non-null    int64
dtypes: float64(2), int64(6), object(5)
memory usage: 90.6+ KB
```

```
# Checking to see if deck level has a significant impact on survival rate.

# Group by 'Cabin_Alphabet' and 'Survived', then count
cabin_alphabet_survival_counts = df_train_original.groupby(['Cabin_Alphabet', 'Survived']).size().unstack(fill_value=0)

# Rename the columns for clarity
cabin_alphabet_survival_counts.columns = ['Survived_0', 'Survived_1']

# Reset index for a clean table
cabin_alphabet_survival_counts = cabin_alphabet_survival_counts.reset_index()

# Plotting
plt.figure(figsize=(12, 6))
sns.barplot(x='Cabin_Alphabet', y='Survived_1', data=cabin_alphabet_survival_counts, color='skyblue', label='Survived')
sns.barplot(x='Cabin_Alphabet', y='Survived_0', data=cabin_alphabet_survival_counts, color='salmon', label='Not Survived', bottom=cabin_alphabet_survival_counts['Survived_1'])

plt.title('Survival Rate by Cabin Alphabet')
plt.xlabel('Cabin Alphabet')
plt.ylabel('Number of Passengers')
plt.legend()
plt.show()
```




```
# Checking to see if deck level has a correlation with Pclass

# Group by 'Cabin_Alphabet' and 'Pclass', then count
cabin_alphabet_pclass_counts = df_train_original.groupby(['Cabin_Alphabet', 'Pclass']).size().unstack(fill_value=0)

# Reset index for a clean table
cabin_alphabet_pclass_counts = cabin_alphabet_pclass_counts.reset_index()

# Plotting
cabin_alphabet_pclass_counts.plot(x='Cabin_Alphabet', kind='bar', stacked=True,
                                  title='Passenger Class Distribution by Cabin Alphabet')
plt.xlabel('Cabin Alphabet')
plt.ylabel('Number of Passengers')
plt.legend(title='Pclass')
plt.show()
```

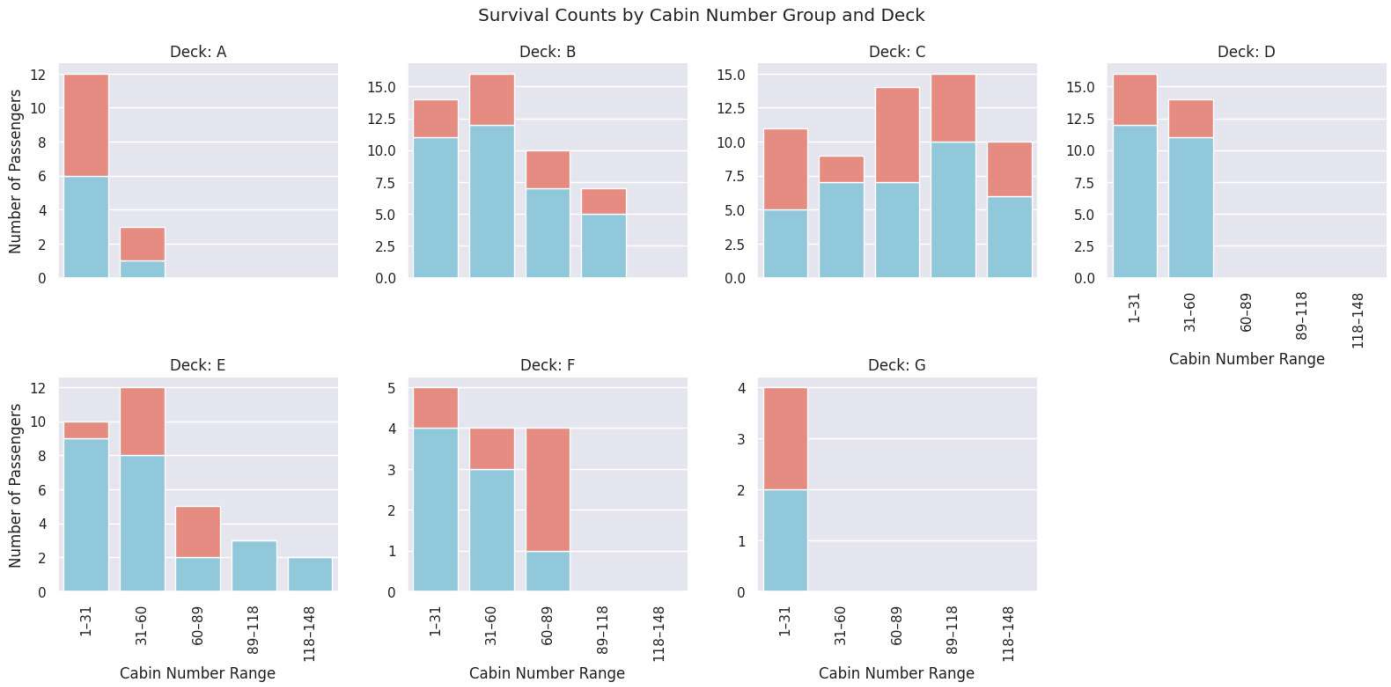


- Everyone in decks A, B and C were 1st class.
- Everyone in deck G was 3rd class.

```
# Checking to see if cabin number has an impact on survival rate

# Create 5 groups for Cabin_Number using pd.cut
df_train_original['Cabin_Number_Group'] = pd.cut(df_train_original['Cabin_Number'], bins=5, labels=False)

# Group by 'Cabin_Number_Group' and 'Survived', then count
cabin_number_group_survival_counts = df_train_original.groupby(['Cabin_Number_Group', 'Survived']).size().unstack(fill_value=0)

# Rename the columns for clarity
cabin_number_group_survival_counts.columns = ['Survived_0', 'Survived_1']

# Reset index for a clean table
cabin_number_group_survival_counts = cabin_number_group_survival_counts.reset_index()

# Plotting
plt.figure(figsize=(12, 6))
sns.barplot(x='Cabin_Number_Group', y='Survived_1', data=cabin_number_group_survival_counts, color='skyblue', label='Survived')
sns.barplot(x='Cabin_Number_Group', y='Survived_0', data=cabin_number_group_survival_counts, color='salmon', label='Not Survived', bottom=cabin_number_group_survival_counts['Sur

plt.title('Survival Rate by Cabin Number Group')
plt.xlabel('Cabin Number Group')
plt.ylabel('Number of Passengers')
plt.legend()
plt.show()
```

Show hidden output

```
# Checking to see if cabin number has an impact on survival rate, faceted by deck level.

# 1. Ensure Cabin_Number is numeric
df_train_original['Cabin_Number'] = pd.to_numeric(df_train_original['Cabin_Number'], errors='coerce')

# 2. Create bins and labels
bin_cut = pd.cut(df_train_original['Cabin_Number'], bins=5)
bin_edges = bin_cut.cat.categories  # Correct way to access bin ranges

bin_labels = [f"{int(edge.left)}-{int(edge.right)}" for edge in bin_edges]

# Assign bin labels
df_train_original['Cabin_Number_Group'] = pd.cut(df_train_original['Cabin_Number'], bins=5, labels=bin_labels)

# 3. Group by Cabin_Alphabet, Cabin_Number_Group, and Survived
cabin_grouped = df_train_original.groupby(['Cabin_Alphabet', 'Cabin_Number_Group', 'Survived'], observed=True).size().unstack(fill_value=0).reset_index()

# Rename columns for clarity
cabin_grouped.columns = ['Cabin_Alphabet', 'Cabin_Number_Group', 'Survived_0', 'Survived_1']

# Optional: Remove rows with missing Cabin_Alphabet
cabin_grouped = cabin_grouped[cabin_grouped['Cabin_Alphabet'].notna()]

# 4. Plotting with FacetGrid
g = sns.FacetGrid(cabin_grouped, col='Cabin_Alphabet', col_wrap=4, height=4, sharey=False)

def facet_barplot(data, color, **kwargs):
    sns.barplot(x='Cabin_Number_Group', y='Survived_1', data=data, color='skyblue', **kwargs)
    sns.barplot(x='Cabin_Number_Group', y='Survived_0', data=data, color='salmon', bottom=data['Survived_1'], **kwargs)
    plt.xticks(rotation=90)

g.map_dataframe(facet_barplot)
g.add_legend()
g.set_axis_labels("Cabin Number Range", "Number of Passengers")
g.set_titles(col_template="Deck: {col_name}")
plt.subplots_adjust(top=0.9)
g.fig.suptitle('Survival Counts by Cabin Number Group and Deck')
plt.show();
```



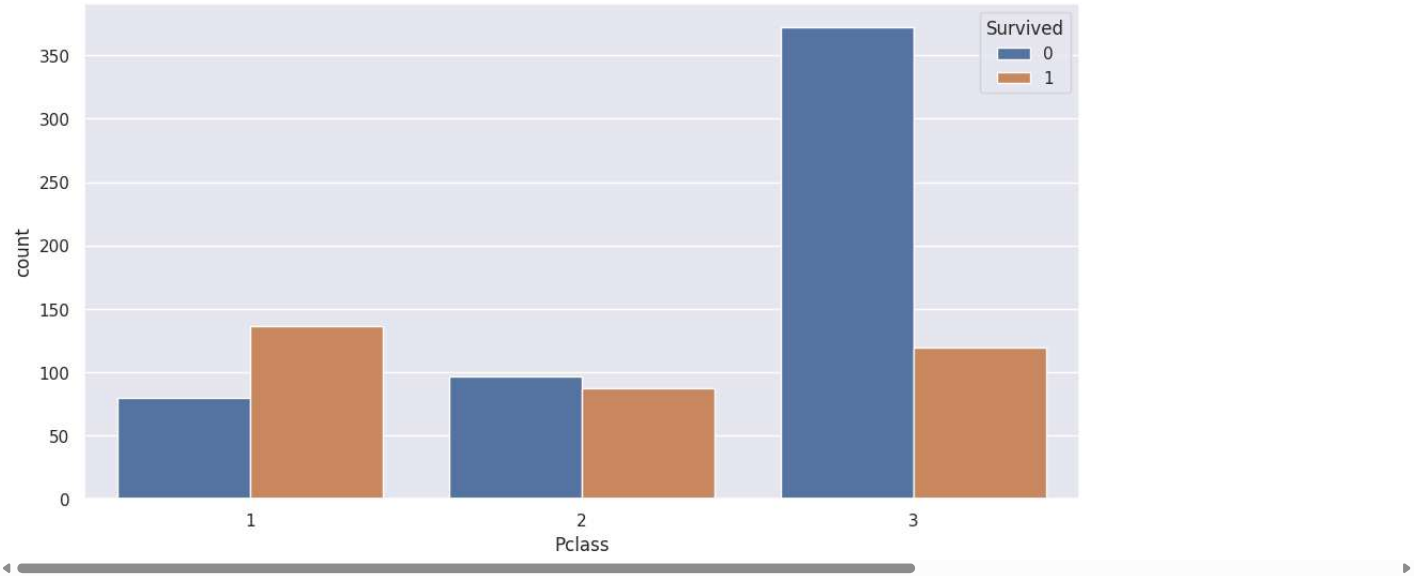
There is no clear pattern here and not enough samples to conclude anything with confidence.

```
# Restoring original train data.
df_train_original = pd.read_csv('train.csv')
```

Other EDA

```
# Plot Pclass vs Survived
plt.figure(figsize=(12, 6))
sns.countplot(x='Pclass', hue='Survived', data=df_train_original)
```

```
<Axes: xlabel='Pclass', ylabel='count'>
```

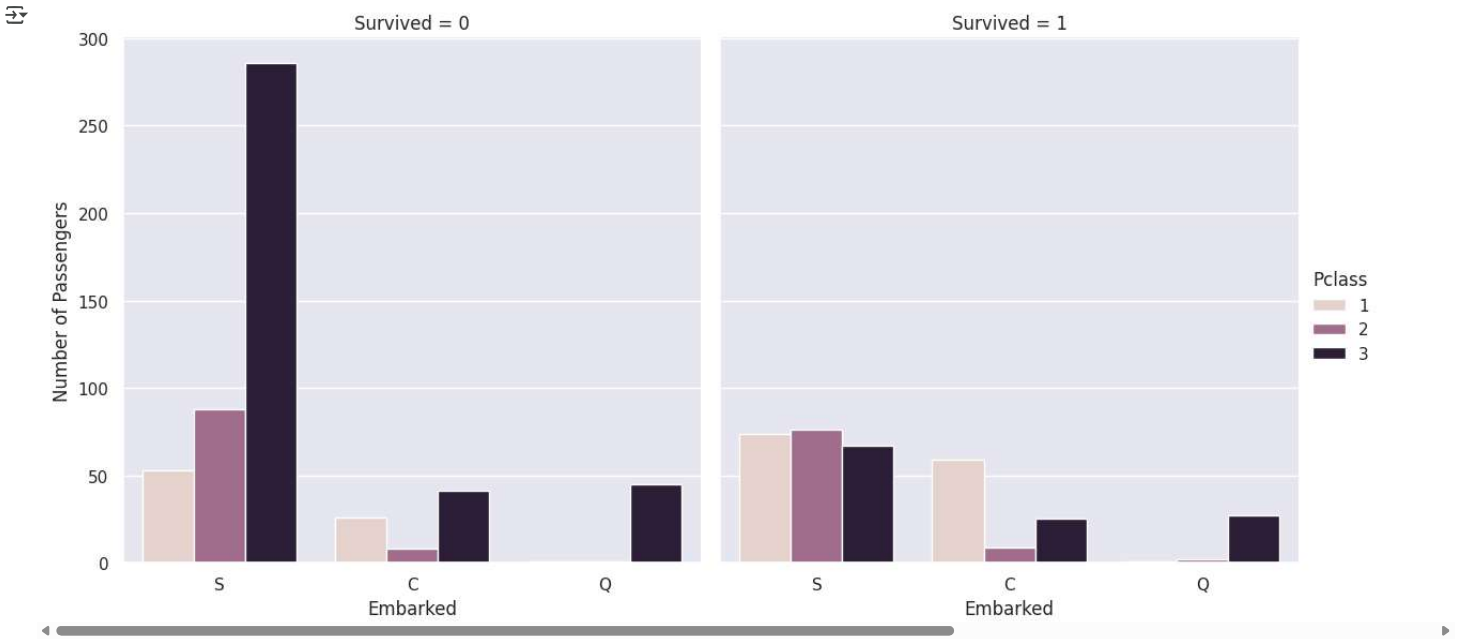



```
# Plot Embarked vs Survived
plt.figure(figsize=(12, 6))
sns.countplot(x='Embarked', hue='Survived', data=df_train_original)
```

Show hidden output

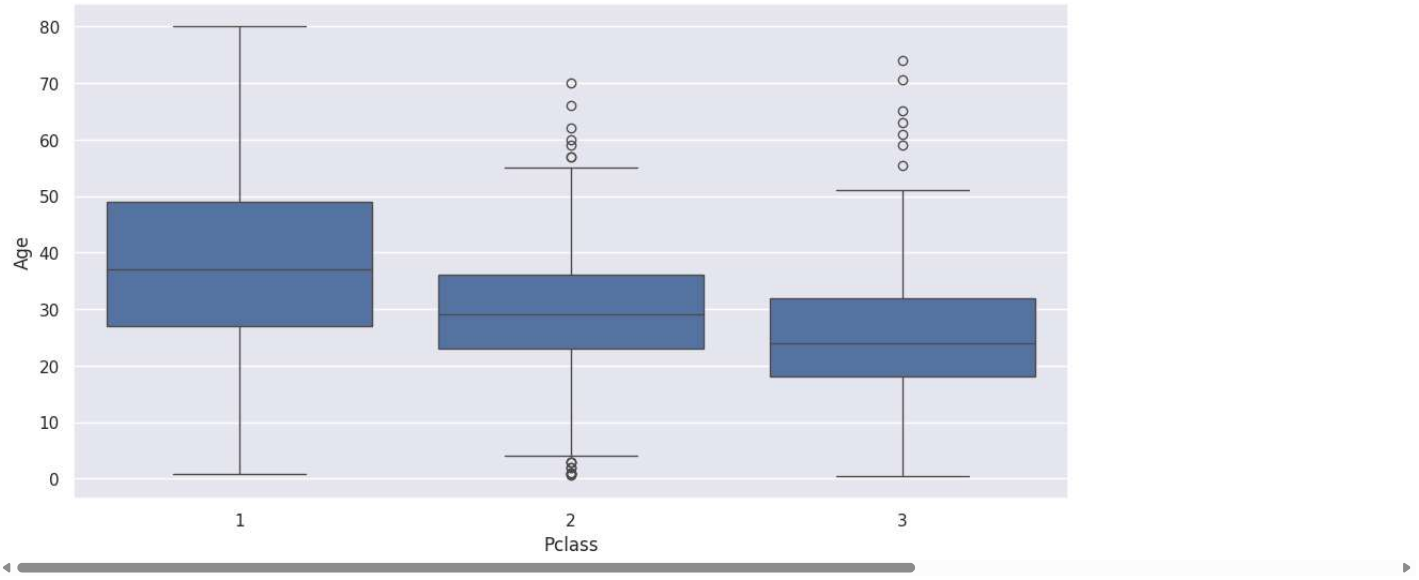
```
# Plot Embarked vs Survived faceted by Survived
g = sns.catplot(
```

```
    x='Embarked', hue='Pclass', col='Survived',
    kind='count', data=df_train_original,
    height=6, aspect=1
)
g.set_titles("Survived = {col_name}")
g.set_axis_labels("Embarked", "Number of Passengers")
plt.show()
```



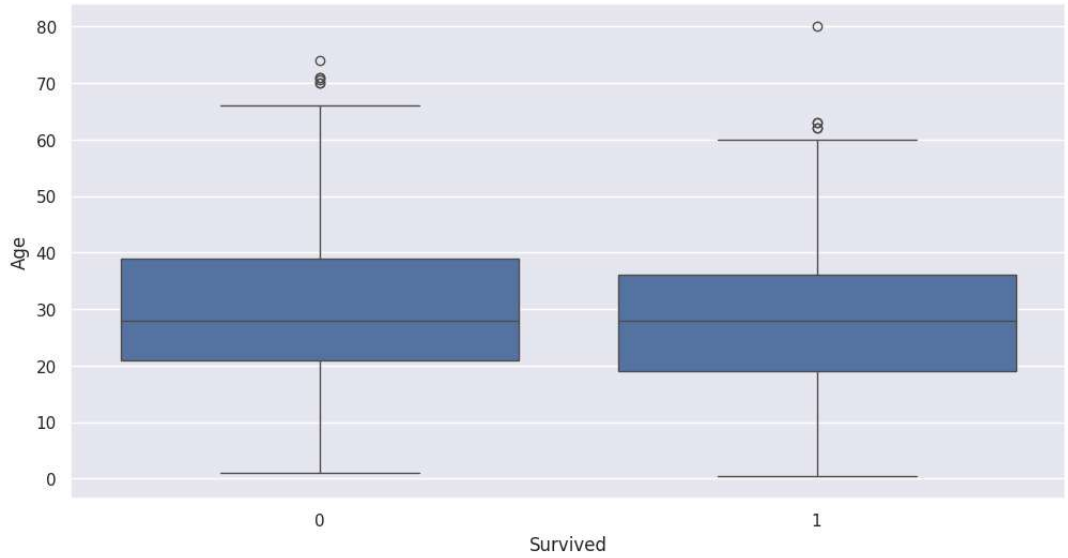

```
# Plot Age vs Pclass
plt.figure(figsize=(12, 6))
sns.boxplot(x='Pclass', y='Age', data=df_train_original)
```

```
<Axes: xlabel='Pclass', ylabel='Age'>
```

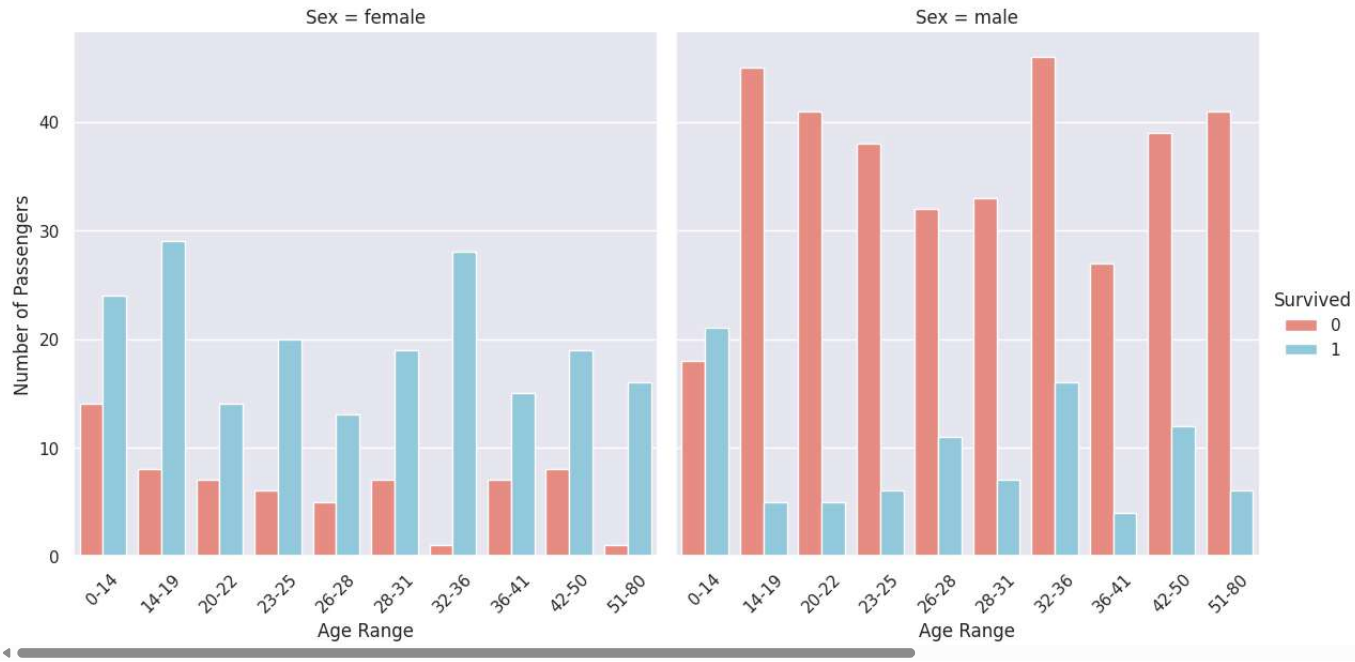



```
# Plot Age vs Survived
plt.figure(figsize=(12, 6))
sns.boxplot(x='Survived', y='Age', data=df_train_original)
```

```
<Axes: xlabel='Survived', ylabel='Age'>
```

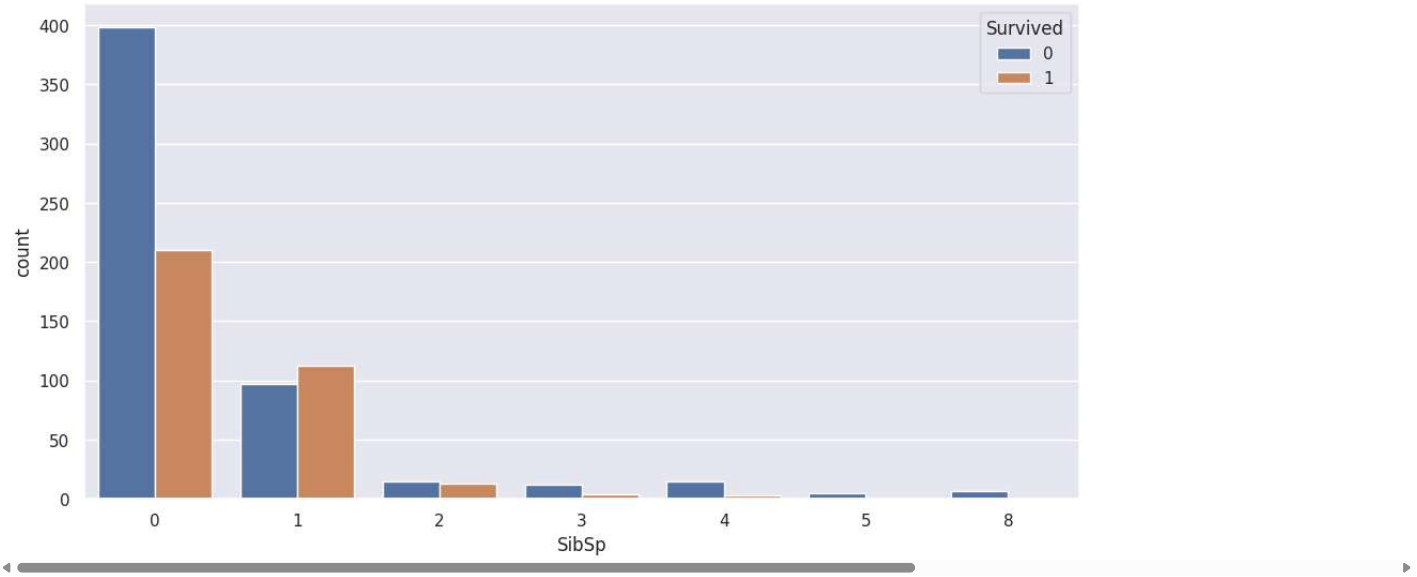



```
# Plot Age (decile) vs Survived faceted by Sex

# 1. Create Age Deciles
df_train_original['Age_Decile'] = pd.qcut(df_train_original['Age'], 10, labels=False, duplicates='drop')

# 2. Get Age Range Labels
age_ranges = df_train_original.groupby('Age_Decile')['Age'].agg(['min', 'max'])
x_labels = [f"{int(row['min'])}-{int(row['max'])}" for _, row in age_ranges.iterrows()]

# 3. Group by Sex, Age_Decile, and Survived
age_survival_counts = df_train_original.groupby(['Sex', 'Age_Decile', 'Survived'], observed=True).size().reset_index(name='Count')

# 4. Plotting
g = sns.catplot(
    data=age_survival_counts, kind='bar',
    x='Age_Decile', y='Count', hue='Survived', col='Sex',
    height=6, aspect=1, palette={0: 'salmon', 1: 'skyblue'}
)

# 5. Customize
g.set_titles("Sex = {col_name}")
g.set_axis_labels("Age Range", "Number of Passengers")
g.set_xticklabels(x_labels, rotation=45)
plt.show()
```




```
# Plot SibSp vs Survived
plt.figure(figsize=(12, 6))
sns.countplot(x='SibSp', hue='Survived', data=df_train_original)
```

```
<Axes: xlabel='SibSp', ylabel='count'>
```




```
# Restoring original train data.
df_train_original = pd.read_csv('train.csv')
```

Merge and clean data

Merge data

```
# Create working datasets
df_train = df_train_original
df_test = df_test_original

# Add a variable to both df_train and df_test indicating whether train or test dataset
df_train['TestYes'] = 0
df_test['TestYes'] = 1

# Create merged dataset used to cleanup data
df_merged = pd.concat([df_train, df_test], ignore_index=True)


df_merged.info()
```

Show hidden output

Creating a variable for ticket type

```
# Split numeric text into a seperate Ticket feature
df_merged['Ticket_Number'] = df_merged['Ticket'].str.extract(r'(\d+)$')
df_merged['Ticket_Group'] = df_merged['Ticket'].str.replace(r'\d+$', '', regex=True)
df_merged['Ticket_Group'] = df_merged['Ticket_Group'].str.replace(' ', '', regex=False)


# If 'Ticket_Number' is N/A set to 0
df_merged['Ticket_Number'] = df_merged['Ticket_Number'].fillna(0)

# If 'Ticket_Group' is " " set to "Regular"
df_merged['Ticket_Group'] = df_merged['Ticket_Group'].replace(r'^\s*$', 'Regular', regex=True)


# Count by Ticket_Group
df_merged['Ticket_Group'].value_counts()
```

Show hidden output

```
# Merge Ticket_Group "A/5", "A/5.", "A.5.", "A./5." into "A/5"
df_merged['Ticket_Group'] = df_merged['Ticket_Group'].replace(['A/5', 'A/5.', 'A.5.', 'A./5.'], 'A/5')

# Merge Ticket_Group "A/4", "A/4.", "A4.", "S.C./A.4.", "SC/A4" into "A/4"
df_merged['Ticket_Group'] = df_merged['Ticket_Group'].replace(['A/4', 'A/4.', 'A4.', 'S.C./A.4.'], 'A/4')

# Merge Ticket_Group "C.A.", "CA.", "CA", "C.A./SOTON" into "CA"
df_merged['Ticket_Group'] = df_merged['Ticket_Group'].replace(['C.A.', 'CA.', 'CA'], 'CA')

# Merge Ticket_Group "SOTON/O.Q.", "SOTON/O2" and "SOTON/OQ" into "SOTON/OQ"
df_merged['Ticket_Group'] = df_merged['Ticket_Group'].replace(['SOTON/O.Q.', 'SOTON/OQ', 'SOTON/O2'], 'SOTON/OQ')

# Merge all other Ticket_Group values into "OTHER"
other_ticket_groups = df_merged['Ticket_Group'].unique()
other_ticket_groups = other_ticket_groups[~pd.Series(other_ticket_groups).isin(['A/5', 'A/4', 'CA', 'SOTON/OQ', 'Regular'])]
df_merged['Ticket_Group'] = df_merged['Ticket_Group'].replace(other_ticket_groups, 'OTHER')
```

Creating a variable for Title

```
# Look for "Mr.", "Master.", "Mrs.", "Miss." in the Name feature and extract to a new feature called "Title"
df_merged['Title'] = df_merged['Name'].str.extract(r' ([A-Za-z]+)\.', expand=False)
df_merged['Title'] = df_merged['Title'].str.replace(' ', '', regex=False)

# Count by Title
df_merged['Title'].value_counts()
```

Show hidden output

```
# Standardize titles
df_merged['Title'] = df_merged['Title'].replace({
    'Mlle': 'Miss',
    'Ms': 'Miss',
    'Mme': 'Mrs',
    'Dr': 'Officer',
    'Rev': 'Officer',
    'Col': 'Officer',
    'Major': 'Officer',
    'Capt': 'Officer',
    'Sir': 'Royalty',
    'Lady': 'Royalty',
    'Countess': 'Royalty',
    'Don': 'Royalty',
    'Dona': 'Royalty',
    'Jonkheer': 'Royalty'
})

# Count by Title
df_merged['Title'].value_counts()
```

Title	count
Mr	757
Miss	264
Mrs	198
Master	61
Officer	23
Royalty	6

dtype: int64

Splitting SibSp into two variables

```
# Create variables Siblings, Spouse
df_merged['Siblings'] = 0
df_merged['Spouse'] = 0

# If Title is Master, Miss or Age < 18 then Siblings = SibSp (this is a child or unmarried person)
df_merged.loc[((df_merged['Title'].isin(['Master', 'Miss'])) | (df_merged['Age'] < 18)), 'Siblings'] = df_merged['SibSp']
df_merged.loc[((df_merged['Title'].isin(['Master', 'Miss'])) | (df_merged['Age'] < 18)), 'Spouse'] = 0

# If Title is Mr, Mrs, Royalty or Officer and Age >= 18, SibSp > 0, then Spouse = 1
df_merged.loc[((df_merged['Title'].isin(['Mr', 'Mrs', 'Royalty', 'Officer'])) & (df_merged['Age'] >= 18) & (df_merged['SibSp'] > 0)), 'Spouse'] = 1
df_merged.loc[((df_merged['Title'].isin(['Mr', 'Mrs', 'Royalty', 'Officer'])) & (df_merged['Age'] >= 18) & (df_merged['SibSp'] > 0)), 'Siblings'] = df_merged['SibSp'] - df_merg
```

Splitting Parch into two variables

```
# Create variables Siblings, Spouse
df_merged['Parents'] = 0
df_merged['Children'] = 0

# If Title is Master, Miss or Age <= 20 then Parents = Parch (this is a child/young adult traveling with parents)
df_merged.loc[((df_merged['Title'].isin(['Master', 'Miss'])) | (df_merged['Age'] <= 20)), 'Parents'] = df_merged['Parch']
df_merged.loc[((df_merged['Title'].isin(['Master', 'Miss'])) | (df_merged['Age'] <= 20)), 'Children'] = 0

# If Title is Mr, Mrs, Royalty or Officer and Age > 20, Parch > 0 then Children = Parch
df_merged.loc[((df_merged['Title'].isin(['Mr', 'Mrs', 'Royalty', 'Officer'])) & (df_merged['Age'] > 20) & (df_merged['Parch'] > 0)), 'Children'] = df_merged['Parch']
df_merged.loc[((df_merged['Title'].isin(['Mr', 'Mrs', 'Royalty', 'Officer'])) & (df_merged['Age'] > 20) & (df_merged['Parch'] > 0)), 'Parents'] = 0
```

Predicting missing Embarked data

[ ] ↳ 8 cells hidden

Predicting missing Fare data

```
# Filter to missing Fare data
missing_fare = df_merged[df_merged['Fare'].isnull()]
```

```
# Print rows
print(missing_fare)
```

```
      PassengerId  Survived  Pclass                Name   Sex   Age  SibSp  \
1043         1044       NaN       3  Storey, Mr. Thomas  male  60.5      0

      Parch Ticket  Fare Cabin Embarked  TestYes Ticket_Number Ticket_Group  \
1043      0   3701   NaN   NaN        S        1          3701      Regular

     Title  Siblings  Spouse  Parents  Children
1043    Mr         0       0        0         0
```

```
# Filter to other passengers in 3rd class, embarking from Southampton
filtered_df = df_merged[(df_merged['Pclass'] == 3) & (df_merged['Embarked'] == 'S')]

# Calculate fare statistics
fare_stats = filtered_df['Fare'].describe()
fare_stats['median'] = filtered_df['Fare'].median()

# Print the results
print(fare_stats)
```

```
count     494.000000
mean       14.435422
std        13.118281
min         0.000000
25%         7.854200
50%         8.050000
75%        15.900000
max        69.550000
median      8.050000
Name: Fare, dtype: float64
```

Missing fare for this passenger can be set to the median fare for 3rd class passengers embarking from Southampton.

Clean data

```
# Remove feature "Cabin" from df_merged dataset
df_merged.drop('Cabin', axis=1, inplace=True)

# Set Embarked = C for PassengerId 62 and 830
df_merged.loc[df_merged['PassengerId'].isin([62, 830]), 'Embarked'] = 'C'

# Set Fare =8.05 for PassengerId 1044
df_merged.loc[df_merged['PassengerId'] == 1044, 'Fare'] = 80.5


df_merged.info()
```

Show hidden output

```
# Download df_merged as csv
df_merged.to_csv('df_merged.csv', index=False)


# Convert Sex, Embarked, Ticket_Group and Title to numeric categorical variables
df_merged['Sex_code'] = pd.Categorical(df_merged['Sex']).codes
df_merged['Embarked_code'] = pd.Categorical(df_merged['Embarked']).codes
df_merged['Ticket_Group_code'] = pd.Categorical(df_merged['Ticket_Group']).codes
df_merged['Title_Code'] = pd.Categorical(df_merged['Title']).codes
```

Predicting age

Preliminary Prediction

```
# Regress Age on Fare, Pclass
age_model1 = smf.ols(formula='Age ~ Fare + C(Pclass)', data=df_merged).fit()
print(age_model1.summary())
```

Show hidden output

```
# Regress Age on Fare, Pclass, SibSp and Parch
age_model2 = smf.ols(formula='Age ~ Fare + C(Pclass) + SibSp + Parch', data=df_merged).fit()
print(age_model2.summary())
```

Show hidden output

```
# Regression3: trial and error to get lowest p-values, however Parents and Children variables cannot be used to predict missing age values
age_model3 = smf.ols(formula='Age ~ C(Pclass) + Parents + Children + C(Embarked_code) + C(Title_Code)', data=df_merged).fit()
print(age_model3.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                    Age   R-squared:                       0.478
Model:                            OLS   Adj. R-squared:                  0.472
Method:                 Least Squares   F-statistic:                     86.05
Date:                Mon, 28 Apr 2025   Prob (F-statistic):          1.35e-137
Time:                        07:20:54   Log-Likelihood:                -3934.7
No. Observations:                1046   AIC:                             7893.
Df Residuals:                    1034   BIC:                             7953.
Df Model:                          11
```

```
Covariance Type:            nonrobust
================================================================================================
                               coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------------------------
Intercept                25.1294      2.001     12.556      0.000      21.202      29.057
C(Pclass)[T.2]           -9.9003      0.957    -10.346      0.000     -11.778      -8.023
C(Pclass)[T.3]          -13.2734      0.861    -15.423      0.000     -14.962     -11.585
C(Embarked_code)[T.1]     5.4144      1.738      3.116      0.002       2.005       8.824
C(Embarked_code)[T.2]     1.9337      0.876      2.208      0.027       0.215       3.652
C(Title_Code)[T.1]        7.5012      1.754      4.278      0.000       4.060      10.942
C(Title_Code)[T.2]       14.8072      1.866      7.935      0.000      11.145      18.469
C(Title_Code)[T.3]       15.6186      2.045      7.638      0.000      11.606      19.631
C(Title_Code)[T.4]       23.3451      2.916      8.006      0.000      17.623      29.067
C(Title_Code)[T.5]       15.3927      4.700      3.275      0.001       6.170      24.615
Parents                  -7.3015      0.786     -9.289      0.000      -8.844      -5.759
Children                  2.1990      0.497      4.425      0.000       1.224       3.174
==============================================================================
Omnibus:                      102.920   Durbin-Watson:                   1.941
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              144.171
Skew:                           0.751   Prob(JB):                     4.94e-32
Kurtosis:                       4.025   Cond. No.                         25.5
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

```
# Regress Age on Fare + C(Pclass) + SibSp + Parch + Sex_code + C(Embarked_code) + C(Ticket_Group_code) + C(Title_Code)
age_model4 = smf.ols(formula='Age ~ C(Pclass) + Parch + C(Embarked_code) + C(Title_Code)', data=df_merged).fit()
print(age_model4.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                    Age   R-squared:                       0.424
Model:                            OLS   Adj. R-squared:                  0.419
Method:                 Least Squares   F-statistic:                     76.28
Date:                Mon, 28 Apr 2025   Prob (F-statistic):          7.92e-117
Time:                        07:20:54   Log-Likelihood:                -3985.8
No. Observations:                1046   AIC:                             7994.
Df Residuals:                    1035   BIC:                             8048.
Df Model:                          10
Covariance Type:            nonrobust
================================================================================================
                               coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------------------------
Intercept                15.4021      1.852      8.315      0.000      11.767      19.037
C(Pclass)[T.2]           -9.9420      1.004     -9.898      0.000     -11.913      -7.971
C(Pclass)[T.3]          -13.3054      0.903    -14.729      0.000     -15.078     -11.533
C(Embarked_code)[T.1]     7.4771      1.812      4.127      0.000       3.922      11.032
C(Embarked_code)[T.2]     1.8145      0.919      1.974      0.049       0.011       3.618
C(Title_Code)[T.1]       13.4462      1.738      7.736      0.000      10.035      16.857
C(Title_Code)[T.2]       24.7408      1.677     14.751      0.000      21.450      28.032
C(Title_Code)[T.3]       27.4713      1.775     15.481      0.000      23.989      30.953
C(Title_Code)[T.4]       33.4773      2.882     11.618      0.000      27.823      39.132
C(Title_Code)[T.5]       25.1597      4.832      5.207      0.000      15.679      34.641
Parch                    -0.4780      0.445     -1.075      0.283      -1.350       0.394
==============================================================================
Omnibus:                       66.145   Durbin-Watson:                   1.967
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               79.412
Skew:                           0.607   Prob(JB):                     5.70e-18
Kurtosis:                       3.588   Cond. No.                         24.3
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

```
# Make age predictions for missing data
df_merged['Age_Predicted1'] = age_model1.predict(df_merged)
df_merged['Age_Predicted2'] = age_model2.predict(df_merged)
df_merged['Age_Predicted4'] = age_model4.predict(df_merged)

# Download df_merged as csv
df_merged.to_csv('df_merged.csv', index=False)
```

```
# Plot Age_Predicted4 vs Age for samples where Age is not missing
plt.figure(figsize=(12, 6))
sns.scatterplot(x='Age_Predicted4', y='Age', data=df_merged[~df_merged['Age'].isnull()])
plt.title('Age vs Age_Predicted4')
plt.xlabel('Age_Predicted4')
plt.ylabel('Age')
plt.show()
```

Show hidden output

Updating Siblings, Spouse, Parents and Children Using Preliminary Predictions

```
# If Title is Master, Miss or Age < 18 then Siblings = SibSp (this is a child or unmarried person)
df_merged.loc[((df_merged['Title'].isin(['Master', 'Miss'])) | (df_merged['Age_Predicted4'] < 18)), 'Siblings'] = df_merged['SibSp']
df_merged.loc[((df_merged['Title'].isin(['Master', 'Miss'])) | (df_merged['Age_Predicted4'] < 18)), 'Spouse'] = 0

# If Title is Mr, Mrs, Royalty or Officer and Age >= 18, SibSp > 0, then Spouse = 1
df_merged.loc[((df_merged['Title'].isin(['Mr', 'Mrs', 'Royalty', 'Officer'])) & (df_merged['Age_Predicted4'] >= 18) & (df_merged['SibSp'] > 0)), 'Spouse'] = 1
df_merged.loc[((df_merged['Title'].isin(['Mr', 'Mrs', 'Royalty', 'Officer'])) & (df_merged['Age_Predicted4'] >= 18) & (df_merged['SibSp'] > 0)), 'Siblings'] = df_merged['SibSp'

# If Title is Master, Miss or Age <= 20 then Parents = Parch (this is a child/young adult traveling with parents)
df_merged.loc[((df_merged['Title'].isin(['Master', 'Miss'])) | (df_merged['Age_Predicted4'] <= 20)), 'Parents'] = df_merged['Parch']
df_merged.loc[((df_merged['Title'].isin(['Master', 'Miss'])) | (df_merged['Age_Predicted4'] <= 20)), 'Children'] = 0

# If Title is Mr, Mrs, Royalty or Officer and Age > 20, Parch > 0 then Children = Parch
```

```
df_merged.loc[((df_merged['Title'].isin(['Mr', 'Mrs', 'Royalty', 'Officer'])) & (df_merged['Age_Predicted4'] > 20) & (df_merged['Parch'] > 0)), 'Children'] = df_merged['Parch']
df_merged.loc[((df_merged['Title'].isin(['Mr', 'Mrs', 'Royalty', 'Officer'])) & (df_merged['Age_Predicted4'] > 20) & (df_merged['Parch'] > 0)), 'Parents'] = 0
```

```
# Download df_merged as csv
df_merged.to_csv('df_merged.csv', index=False)
```

Rerunning Best Age Prediction

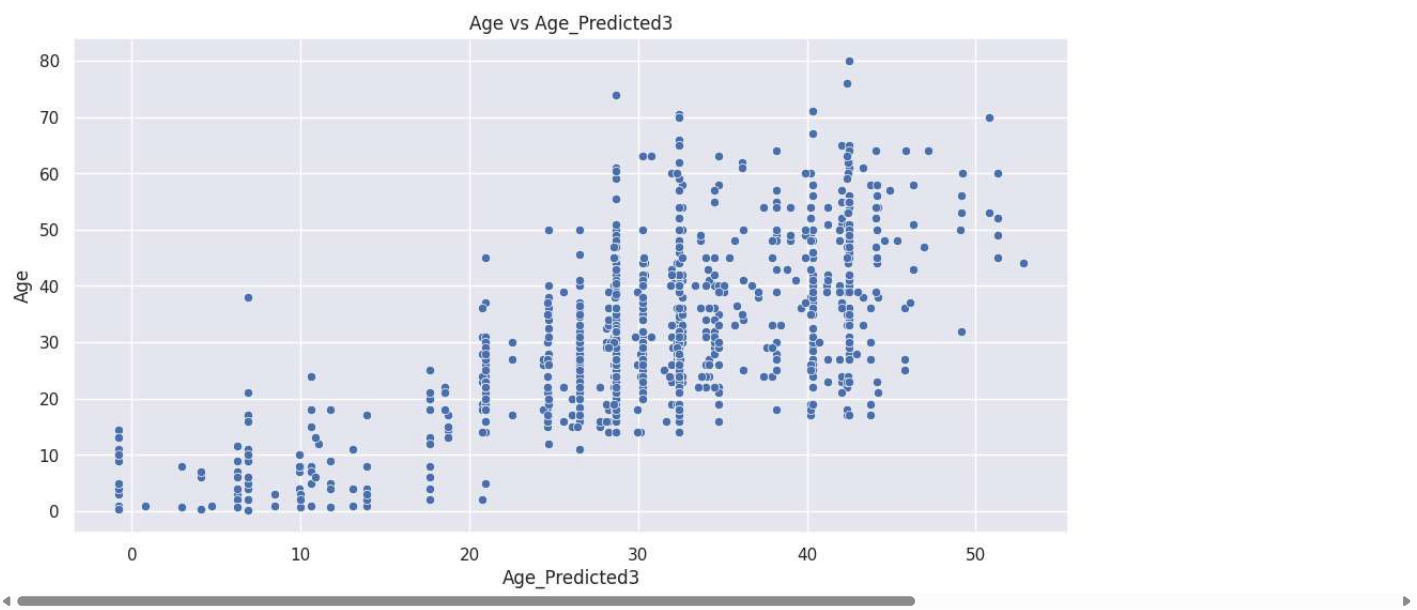
```
# Regression3: trial and error to get lowest p-values, however Parents and Children variables cannot be used to predict missing age values
age_model3 = smf.ols(formula='Age ~ C(Pclass) + Parents + Children + Spouse + C(Embarked_code) + C(Title_Code)', data=df_merged).fit()
print(age_model3.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                    Age   R-squared:                       0.464
Model:                            OLS   Adj. R-squared:                  0.457
Method:                 Least Squares   F-statistic:                     74.37
Date:                Mon, 28 Apr 2025   Prob (F-statistic):          1.43e-130
Time:                        07:20:54   Log-Likelihood:                -3948.9
No. Observations:                1046   AIC:                             7924.
Df Residuals:                    1033   BIC:                             7988.
Df Model:                          12
Covariance Type:            nonrobust
================================================================================================
                                   coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------------------------
Intercept                       24.9161      2.110     11.811      0.000      20.776      29.056
C(Pclass)[T.2]                 -10.0812      0.974    -10.353      0.000     -11.992      -8.171
C(Pclass)[T.3]                 -13.8246      0.883    -15.652      0.000     -15.558     -12.091
C(Embarked_code)[T.1]            5.8653      1.761      3.330      0.001       2.410       9.321
C(Embarked_code)[T.2]            2.1620      0.889      2.431      0.015       0.417       3.907
C(Title_Code)[T.1]               7.6660      1.814      4.226      0.000       4.106      11.226
C(Title_Code)[T.2]              15.4404      2.003      7.708      0.000      11.510      19.371
C(Title_Code)[T.3]              17.5038      2.211      7.918      0.000      13.166      21.841
C(Title_Code)[T.4]              24.2196      3.025      8.008      0.000      18.285      30.155
C(Title_Code)[T.5]              16.2600      4.806      3.383      0.001       6.830      25.690
Parents                         -7.0072      0.892     -7.852      0.000      -8.758      -5.256
Children                         1.7089      0.498      3.430      0.001       0.731       2.687
Spouse                          -2.1905      0.889     -2.464      0.014      -3.935      -0.446
==============================================================================
Omnibus:                       87.800   Durbin-Watson:                   1.958
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              115.979
Skew:                           0.691   Prob(JB):                     6.54e-26
Kurtosis:                       3.866   Cond. No.                         26.6
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

```
df_merged['Age_Predicted3'] = age_model3.predict(df_merged)
```

```
# Download df_merged as csv
df_merged.to_csv('df_merged.csv', index=False)
```

```
# Plot Age_Predicted3 vs Age for samples where Age is not missing
plt.figure(figsize=(12, 6))
sns.scatterplot(x='Age_Predicted3', y='Age', data=df_merged[~df_merged['Age'].isnull()])
plt.title('Age vs Age_Predicted3')
plt.xlabel('Age_Predicted3')
plt.ylabel('Age')
plt.show()
```




```
# For missing values of Age, replace Age with Age_Predicted3
df_merged['Age'] = df_merged['Age'].fillna(df_merged['Age_Predicted3'])
```

```
# Remove Age_Predicted1, Age_Predicted2, Age_Predicted3 and Age_Predicted4 columns
df_merged.drop(['Age_Predicted1', 'Age_Predicted2', 'Age_Predicted3', 'Age_Predicted4'], axis=1, inplace=True)

# Download df_merged as csv
df_merged.to_csv('df_merged.csv', index=False)


df_merged.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1309 entries, 0 to 1308
Data columns (total 23 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   PassengerId        1309 non-null   int64  
 1   Survived           891 non-null    float64
 2   Pclass             1309 non-null   int64  
 3   Name               1309 non-null   object 
 4   Sex                1309 non-null   object 
 5   Age                1309 non-null   float64
 6   SibSp              1309 non-null   int64  
 7   Parch              1309 non-null   int64  
 8   Ticket             1309 non-null   object 
 9   Fare               1309 non-null   float64
 10  Embarked           1309 non-null   object 
 11  TestYes            1309 non-null   int64  
 12  Ticket_Number      1309 non-null   object 
 13  Ticket_Group       1309 non-null   object 
 14  Title              1309 non-null   object 
 15  Siblings           1309 non-null   int64
```

Introduction

Link to access this code - https://colab.research.google.com/drive/1OhL-lseBrsHbzX33VEzRQuRTYuQXHu5s

Data taken from - https://www.kaggle.com/c/titanic/data

Part 1: https://colab.research.google.com/drive/1alhR1osGKoX7YIvuWhXdu1Gcpa0RPhid#scrollTo=9HWrWlpE3gFQ

Import modules and data files

[ ] ↳ 3 cells hidden

Split Train and Test data

```
# Split merged dataset df_merged into df_train and df_test based on the TestYes feature
df_train = df_merged[df_merged['TestYes'] == 0]
df_test = df_merged[df_merged['TestYes'] == 1]

# Dropping TestYes feature from both datasets
df_train = df_train.drop(['TestYes'], axis=1)
df_test = df_test.drop(['TestYes'], axis=1)


df_train.info()
```

Show hidden output

Logistic Regression Models

```
feature_cols = ['Pclass', 'Age', 'Embarked', 'Sex', 'Siblings', 'Children', 'Parents']
# Prepare X and y
X_train = df_train[feature_cols]
y_train = df_train['Survived']

# One-hot encode
X_train = pd.get_dummies(X_train, columns=['Embarked', 'Sex'], drop_first=True)

# Add constant
X_train_sm = sm.add_constant(X_train)

# Ensure all columns in X_train_sm and y_train are numeric
X_train_sm = X_train_sm.astype(float)
y_train = y_train.astype(float)

# Fit logistic regression
logit_model = sm.Logit(y_train, X_train_sm)
result = logit_model.fit()

# View summary
print(result.summary())
```

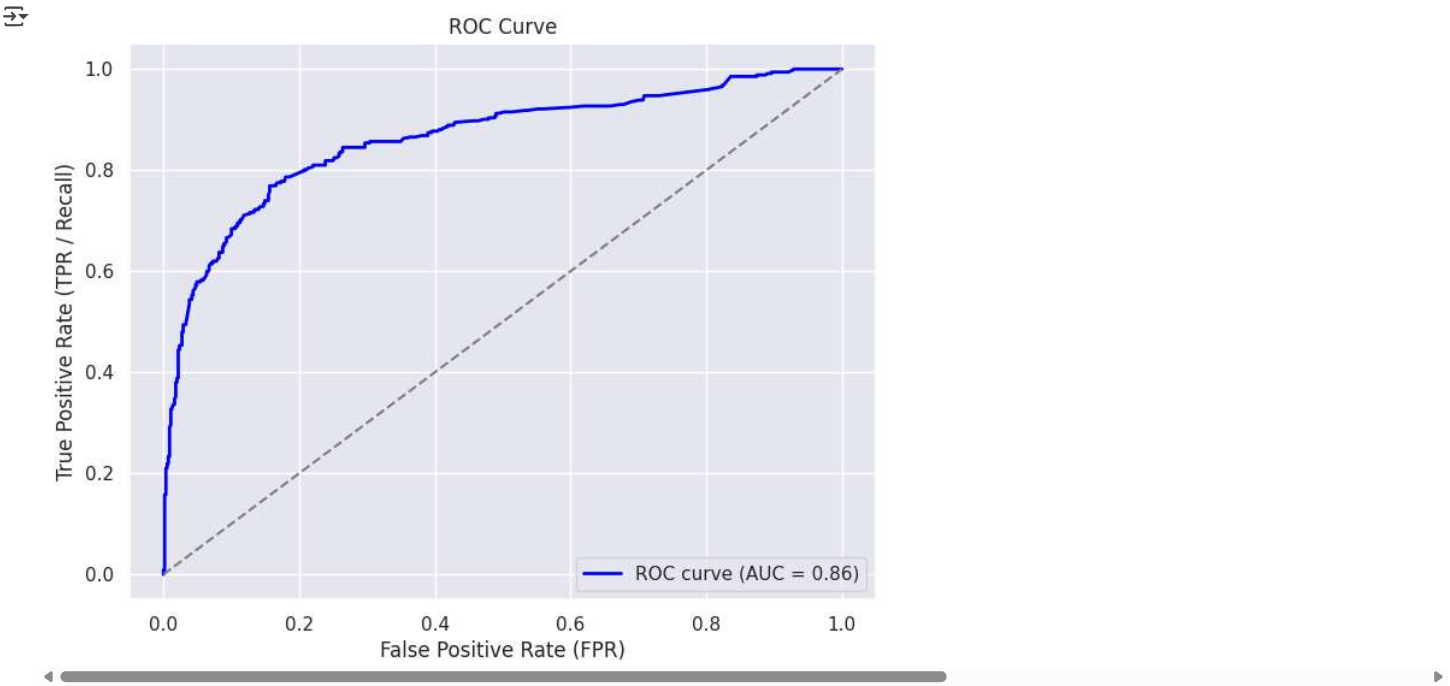
```
Optimization terminated successfully.
         Current function value: 0.433979
         Iterations 7
                           Logit Regression Results
==============================================================================
Dep. Variable:               Survived   No. Observations:                  891
Model:                          Logit   Df Residuals:                      882
Method:                           MLE   Df Model:                            8
Date:                Mon, 28 Apr 2025   Pseudo R-squ.:                 0.3483
Time:                        08:26:41   Log-Likelihood:                -386.68
converged:                       True   LL-Null:                       -593.33
Covariance Type:            nonrobust   LLR p-value:                2.666e-84
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          5.3422      0.558      9.572      0.000       4.248       6.436
Pclass        -1.1623      0.135     -8.586      0.000      -1.428      -0.897
Age           -0.0370      0.009     -3.998      0.000      -0.055      -0.019
Siblings      -0.6477      0.167     -3.886      0.000      -0.974      -0.321
Children      -0.2137      0.131     -1.631      0.103      -0.470       0.043
Parents        0.5850      0.309      1.894      0.058      -0.020       1.190
Embarked_Q    -0.0191      0.387     -0.049      0.961      -0.778       0.740
Embarked_S    -0.4495      0.231     -1.944      0.052      -0.903       0.004
Sex_male      -2.6965      0.202    -13.355      0.000      -3.092      -2.301
==============================================================================
```

```
# Predict probabilities on the training data
y_pred_probs = result.predict(X_train_sm)
```

```
# Convert probabilities to 0/1 predictions
y_pred_train = (y_pred_probs >= 0.5).astype(int)

accuracy = accuracy_score(y_train, y_pred_train)
print(f"Accuracy on training data: {accuracy:.4f}")

# Display a confusion matrix
cm = confusion_matrix(y_train, y_pred_train)
print("\nConfusion Matrix:")
print(cm)

# Display a classification report
print("\nClassification Report:")
print(classification_report(y_train, y_pred_train))
```

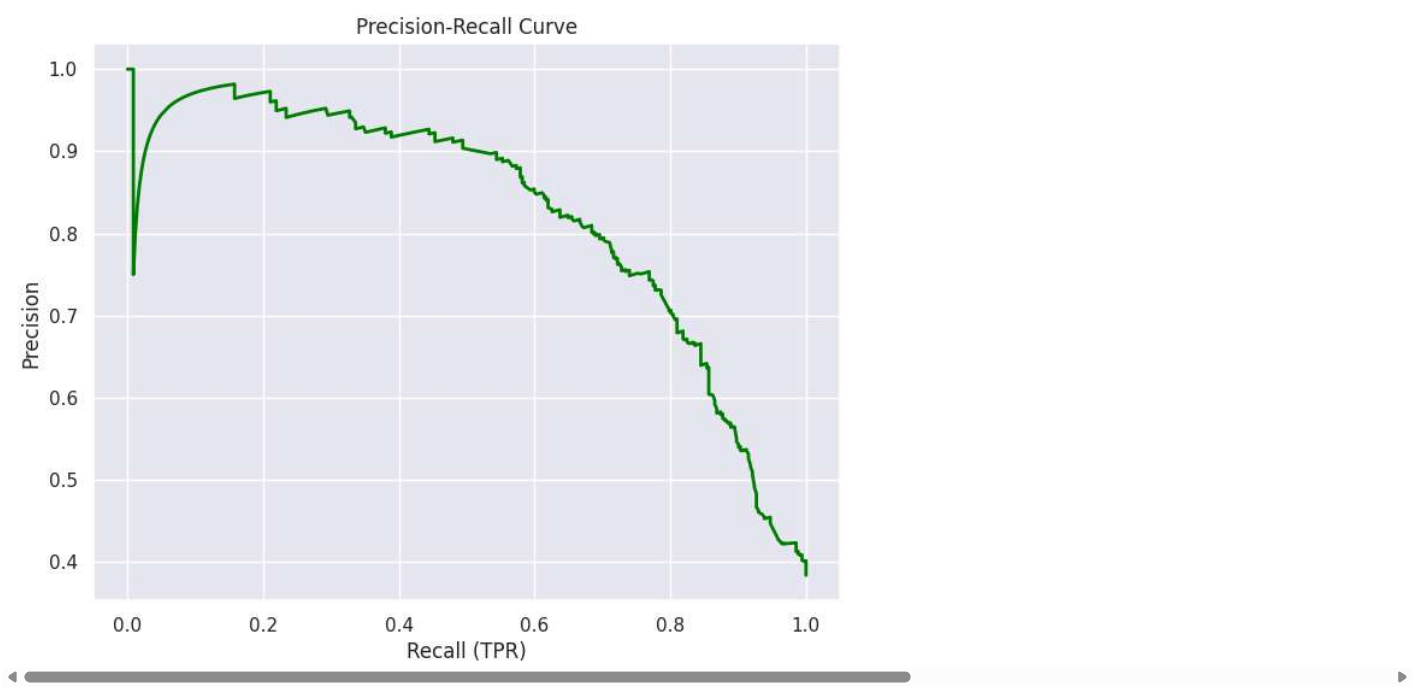
```
Accuracy on training data: 0.8148

Confusion Matrix:
[[485  64]
 [101 241]]

Classification Report:
              precision    recall  f1-score   support

         0.0       0.83      0.88      0.85       549
         1.0       0.79      0.70      0.74       342

    accuracy                           0.81       891
   macro avg       0.81      0.79      0.80       891
weighted avg       0.81      0.81      0.81       891
```

```
# Get predicted probabilities
y_pred_probs = result.predict(X_train_sm)

# ROC Curve
fpr, tpr, thresholds = roc_curve(y_train, y_pred_probs)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8,6))
plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')  # Diagonal line for random guess
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR / Recall)')
plt.title('ROC Curve')
plt.legend(loc="lower right")
plt.grid(True)
plt.show()
```




```
# Precision-Recall Curve
precision, recall, thresholds_pr = precision_recall_curve(y_train, y_pred_probs)

plt.figure(figsize=(8,6))
plt.plot(recall, precision, color='green', lw=2)
```

```
plt.xlabel('Recall (TPR)')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.grid(True)
plt.show()
```




```
# Step 1: Prepare X_test
X_test = df_test[feature_cols]

# Step 2: One-hot encode categorical features
X_test = pd.get_dummies(X_test, columns=['Embarked', 'Sex'], drop_first=True)

# Step 3: Reindex to match X_train columns (before adding constant!)
X_test = X_test.reindex(columns=X_train.columns, fill_value=0)

# Step 4: Add constant manually
X_test_sm = sm.add_constant(X_test)

# Step 5: Ensure numeric
X_test_sm = X_test_sm.astype(float)

# Step 6: Predict
y_pred_probs_test = result.predict(X_test_sm)

# Step 7: Threshold to 0/1
y_pred_test = (y_pred_probs_test >= 0.5).astype(int)

# Step 8: Add prediction back to df_test
df_test['Survived_Predicted'] = y_pred_test



# Export df_test PassengerId and Survived
df_test[['PassengerId', 'Survived_Predicted']].to_csv('logit1_predictions.csv', index=False)
```

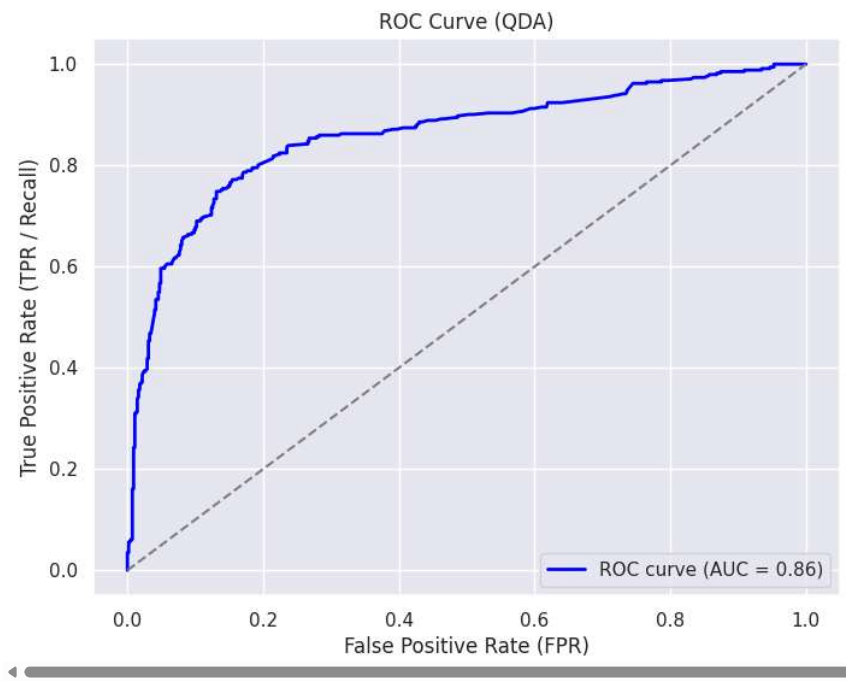
QDA Model

```
# 1. Define features
feature_cols = ['Pclass', 'Age', 'Embarked', 'Sex', 'Siblings', 'Children', 'Parents']

# 2. Prepare X_train and y_train
X_train = df_train[feature_cols]
y_train = df_train['Survived']

# 3. One-hot encode categorical variables
X_train = pd.get_dummies(X_train, columns=['Embarked', 'Sex'], drop_first=True)

# 4. Ensure everything is numeric (you probably already are, but good practice)
X_train = X_train.astype(float)
y_train = y_train.astype(float)

# 5. Create and fit QDA model
qda = QuadraticDiscriminantAnalysis()
```

```
qda.fit(X_train, y_train)

# 6. Predict on training data
y_pred_train = qda.predict(X_train)

# 7. Evaluate model
accuracy = accuracy_score(y_train, y_pred_train)
print(f"Training Accuracy: {accuracy:.4f}")

print("\nConfusion Matrix:")
print(confusion_matrix(y_train, y_pred_train))

print("\nClassification Report:")
print(classification_report(y_train, y_pred_train))
```

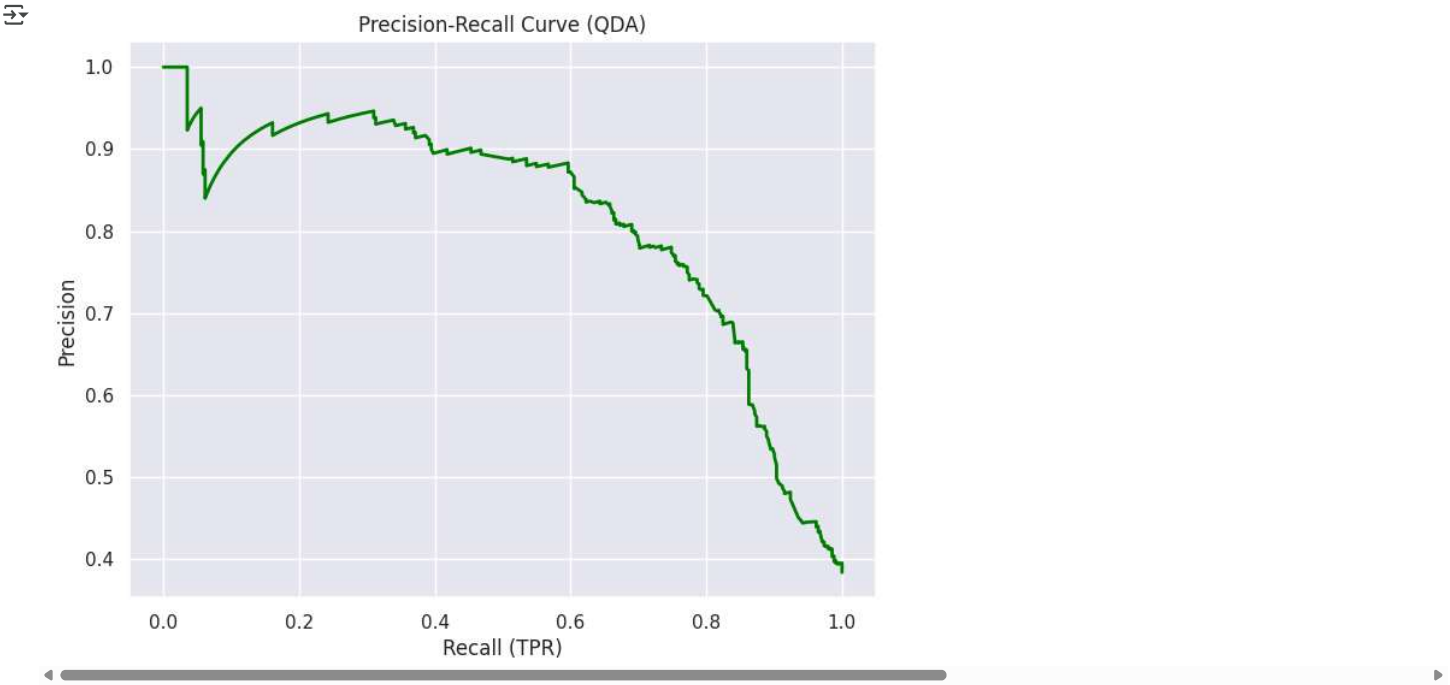
```
Training Accuracy: 0.8193

Confusion Matrix:
[[472  77]
 [ 84 258]]

Classification Report:
              precision    recall  f1-score   support

         0.0       0.85      0.86      0.85       549
         1.0       0.77      0.75      0.76       342

    accuracy                           0.82       891
   macro avg       0.81      0.81      0.81       891
weighted avg       0.82      0.82      0.82       891
```

```
# Get predicted probabilities
y_pred_probs_train = qda.predict_proba(X_train)[:, 1]  # probability of class 1 (Survived)

# ROC Curve
fpr, tpr, thresholds = roc_curve(y_train, y_pred_probs_train)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8,6))
plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')  # Random guess line
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR / Recall)')
plt.title('ROC Curve (QDA)')
plt.legend(loc="lower right")
plt.grid(True)
plt.show()
```




```
# Precision-Recall Curve
precision, recall, thresholds_pr = precision_recall_curve(y_train, y_pred_probs_train)

plt.figure(figsize=(8,6))
plt.plot(recall, precision, color='green', lw=2)
plt.xlabel('Recall (TPR)')
```

```
plt.ylabel('Precision')
plt.title('Precision-Recall Curve (QDA)')
plt.grid(True)
plt.show()
```




```
# Step 1: Prepare X_test
X_test = df_test[feature_cols]

# Step 2: One-hot encode categorical features
X_test = pd.get_dummies(X_test, columns=['Embarked', 'Sex'], drop_first=True)

# Step 3: Reindex to match X_train columns (NO constant yet!)
X_test = X_test.reindex(columns=X_train.columns, fill_value=0)

# Step 4: Ensure numeric
X_test = X_test.astype(float)

# Step 5: Predict 0/1 directly using QDA
y_pred_test = qda.predict(X_test)

# Step 6: Add predictions back to df_test
df_test['Survived_Predicted_QDA'] = y_pred_test


# Export df_test PassengerId and Survived
df_test[['PassengerId', 'Survived_Predicted_QDA']].to_csv('QDA1_predictions.csv', index=False)
```

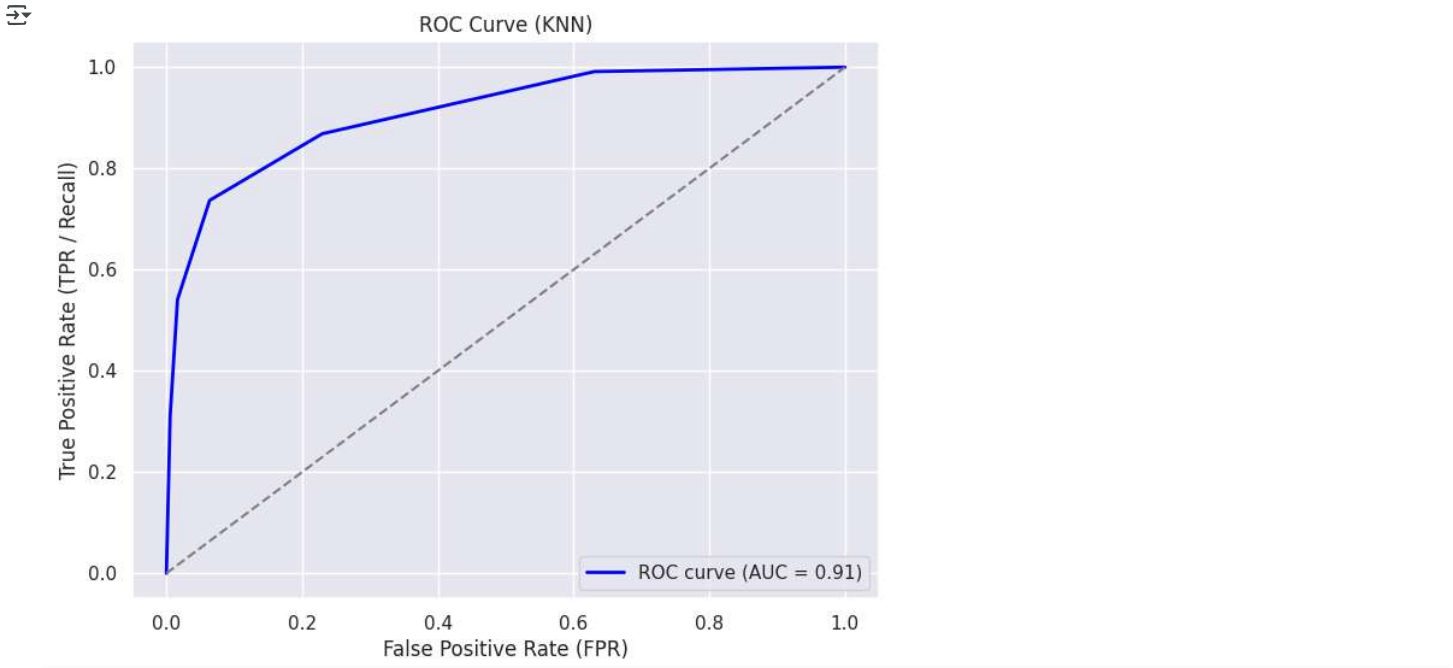
KNN Model

```
# 1. Define feature columns
feature_cols = ['Pclass', 'Age', 'Embarked', 'Sex', 'Siblings', 'Children', 'Parents']

# 2. Prepare X_train and y_train
X_train = df_train[feature_cols]
y_train = df_train['Survived']

# 3. One-hot encode categorical variables
X_train = pd.get_dummies(X_train, columns=['Embarked', 'Sex'], drop_first=True)

# 4. Ensure numeric
X_train = X_train.astype(float)
y_train = y_train.astype(float)

# 5. Create and fit KNN model
knn = KNeighborsClassifier(n_neighbors=5)  # You can adjust k=5,7,9 etc.
knn.fit(X_train, y_train)

# 6. Predict on training data
y_pred_train = knn.predict(X_train)

# 7. Evaluate model
accuracy = accuracy_score(y_train, y_pred_train)
print(f"Training Accuracy: {accuracy:.4f}")
```

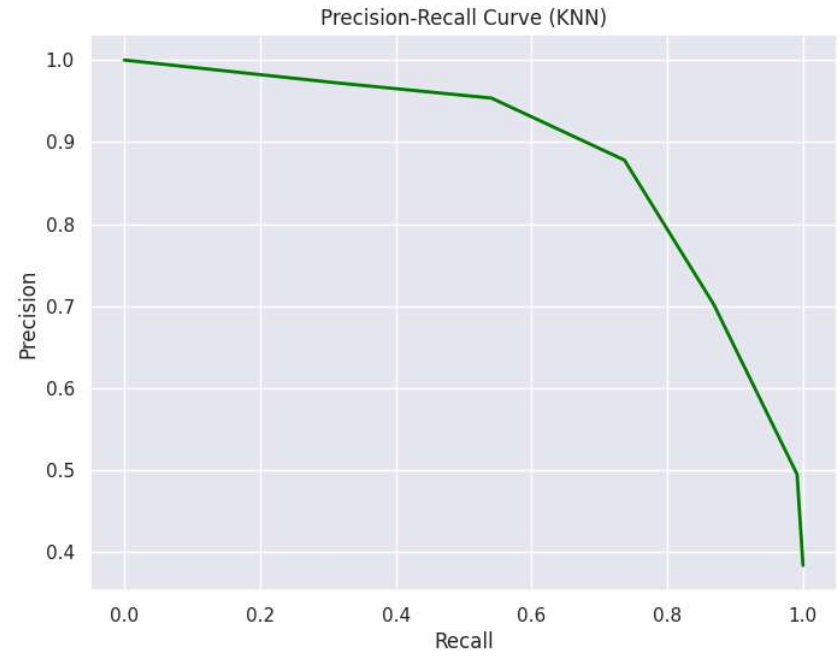
```
print("\nConfusion Matrix:")
print(confusion_matrix(y_train, y_pred_train))

print("\nClassification Report:")
print(classification_report(y_train, y_pred_train))
```

```
Training Accuracy: 0.8597

Confusion Matrix:
[[514  35]
 [ 90 252]]

Classification Report:
              precision    recall  f1-score   support

         0.0       0.85      0.94      0.89       549
         1.0       0.88      0.74      0.80       342

    accuracy                           0.86       891
   macro avg       0.86      0.84      0.85       891
weighted avg       0.86      0.86      0.86       891
```

```
# Get predicted probabilities
y_pred_probs_train = knn.predict_proba(X_train)[:, 1]  # Probability of Survived = 1

# ROC Curve
fpr, tpr, thresholds = roc_curve(y_train, y_pred_probs_train)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8,6))
plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR / Recall)')
plt.title('ROC Curve (KNN)')
plt.legend(loc="lower right")
plt.grid(True)
plt.show()
```




```
# Precision-Recall Curve
precision, recall, thresholds_pr = precision_recall_curve(y_train, y_pred_probs_train)

plt.figure(figsize=(8,6))
plt.plot(recall, precision, color='green', lw=2)
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve (KNN)')
plt.grid(True)
plt.show()
```




```
# Step 1: Prepare X_test
X_test = df_test[feature_cols]

# Step 2: One-hot encode
X_test = pd.get_dummies(X_test, columns=['Embarked', 'Sex'], drop_first=True)

# Step 3: Reindex to match X_train columns
X_test = X_test.reindex(columns=X_train.columns, fill_value=0)

# Step 4: Ensure numeric
X_test = X_test.astype(float)

# Step 5: Predict
y_pred_test = knn.predict(X_test)

# Step 6: Add predictions back to df_test
df_test['Survived_Predicted_KNN'] = y_pred_test


# Export df_test PassengerId and Survived
df_test[['PassengerId', 'Survived_Predicted_KNN']].to_csv('KNN1_predictions.csv', index=False)
```