

```
In [ ]: import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import chi2_contingency
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
# import numpy as np
from sklearn.ensemble import BaggingClassifier
import torch.nn as nn
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
import torch
import torch.optim as optim
import torch.nn as nn

import xgboost as xgb
import itertools

from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

In [2]: filepath = 'data/titanic/train.csv'
train = pd.read_csv(filepath)
filepath = 'data/titanic/test.csv'
test = pd.read_csv(filepath)

In [3]: train, val = train_test_split(train, test_size=0.2, random_state=42)
```

Problem 1 - Classification Model 1:

After doing some initial analysis of the various input features and target label, (show your work on this), run an experiment in Python using any of the techniques you learned about in lectures/materials/background reading. Clearly show your code in your IPython notebook file (ultimately submitted as a pdf file) along with the necessary comments showing the input features used in your model and the hyperparameter settings (which depend on the model technique you are using). For example, if you're using a random forest, one of the hyperparameters would be the maximum tree depth, (there are many other hyperparameter settings, as well).

- Pre-processing Techniques: Feature selection, feature creation, dimensionality reduction, noise reduction, attribute discretization, . . .
- Classification Techniques: decision trees for classification, rule-based classifiers, instance-based classifiers (including KNN's), Bayesian classifiers, artificial neural networks (ANN's), support vector machines (SVM's), and finally, and perhaps most importantly, ensemble methods (bagging, boosting, random forests) for combining the methods listed above.

Provide code/comments/appropriate description for the following:

1. Code and results for any preprocessing/visualization of input features and label (passenger survivorship).

In [3]: `train.head()`

Out[3]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500

In [95]:

```

print(f'There are {train.shape[0]} samples and {train.shape[1]} features in the tra
print('-----')
print('Results of train.info():')
print(train.info())
print('-----')
print('Results of train.describe():')
print(train.describe())

```

There are 712 samples and 16 features in the training set.

```
-----
Results of train.info():
<class 'pandas.core.frame.DataFrame'>
Index: 712 entries, 331 to 102
Data columns (total 16 columns):
#   Column                Non-Null Count  Dtype
---  -
0   PassengerId            712 non-null    int64
1   Survived                712 non-null    int64
2   Name                   712 non-null    object
3   Age                    712 non-null    float64
4   SibSp                  712 non-null    int64
5   Parch                  712 non-null    int64
6   Ticket                 712 non-null    object
7   Fare                   712 non-null    float64
8   Parch_binary           712 non-null    int64
9   Pclass_binary          712 non-null    int64
10  Embarked_Q             712 non-null    bool
11  Embarked_S             712 non-null    bool
12  Pclass_2               712 non-null    bool
13  Pclass_3               712 non-null    bool
14  Sex_male               712 non-null    bool
15  Fare_binary            712 non-null    int64
dtypes: bool(5), float64(2), int64(7), object(2)
memory usage: 70.2+ KB
None
-----
```

```
Results of train.describe():
```

	PassengerId	Survived	Age	SibSp	Parch \
count	712.000000	712.000000	712.000000	712.000000	712.000000
mean	448.234551	0.376404	29.069553	0.553371	0.379213
std	256.731423	0.484824	13.174172	1.176404	0.791669
min	1.000000	0.000000	0.420000	0.000000	0.000000
25%	224.750000	0.000000	22.000000	0.000000	0.000000
50%	453.500000	0.000000	26.000000	0.000000	0.000000
75%	673.500000	1.000000	36.000000	1.000000	0.000000
max	891.000000	1.000000	80.000000	8.000000	6.000000

	Fare	Parch_binary	Pclass_binary	Fare_binary
count	712.000000	712.000000	712.000000	712.000000
mean	32.586276	0.240169	0.441011	0.619382
std	51.969529	0.427486	0.496857	0.485880
min	0.000000	0.000000	0.000000	0.000000
25%	7.925000	0.000000	0.000000	0.000000
50%	14.454200	0.000000	0.000000	1.000000
75%	30.500000	0.000000	1.000000	1.000000
max	512.329200	1.000000	1.000000	1.000000

All of the feature datatypes seem to make sense. The 'Cabin' feature is 77% null so I will likely remove/transform that. The 'Age' feature is ~20% null so that will also need to be dealt with.

General observations about the feature stats: The mean age is lower than I would have imagined. I'm also surprised that 'SibSp' stats were so low. I imagined that most people were

on the Titanic with a spouse! The stats for the 'Parch' (number of parents or children aboard) feature are very low. From the little I know of the Titanic it didn't seem like something people would go on with their parents/kids, so this isn't particularly surprising. Still, I'm interested to see the details.

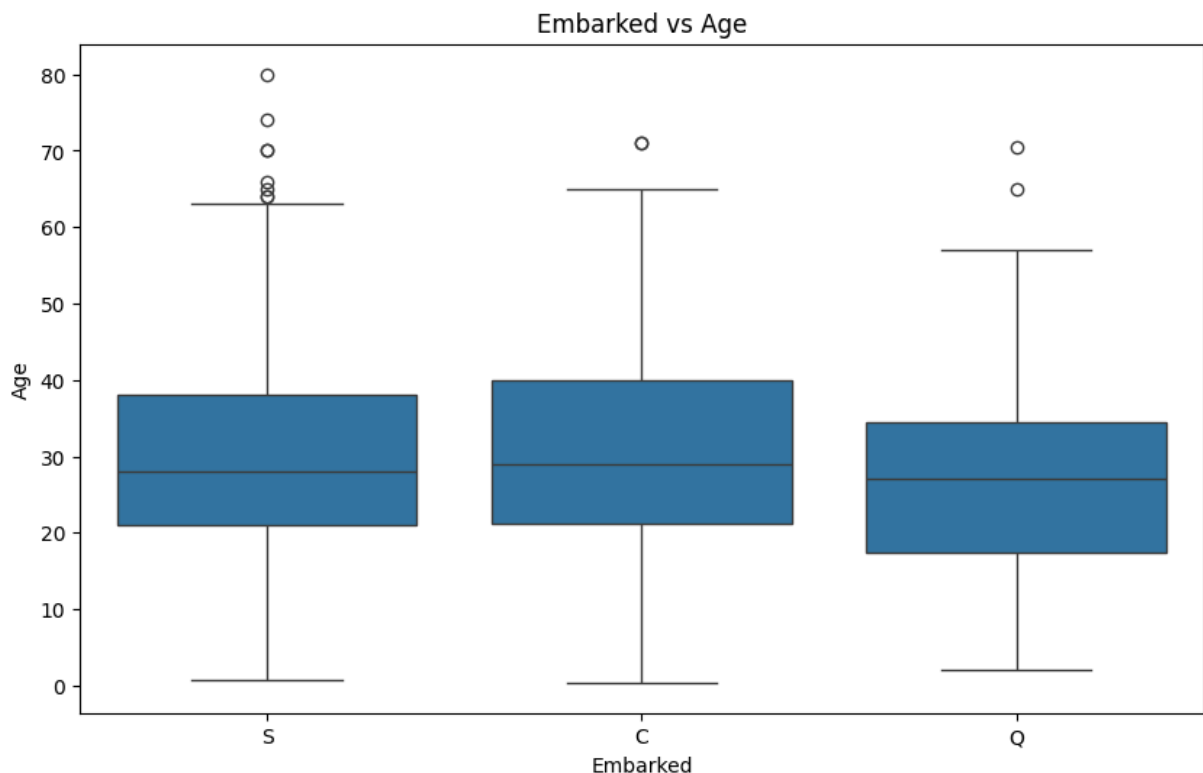
Dealing with Null Values

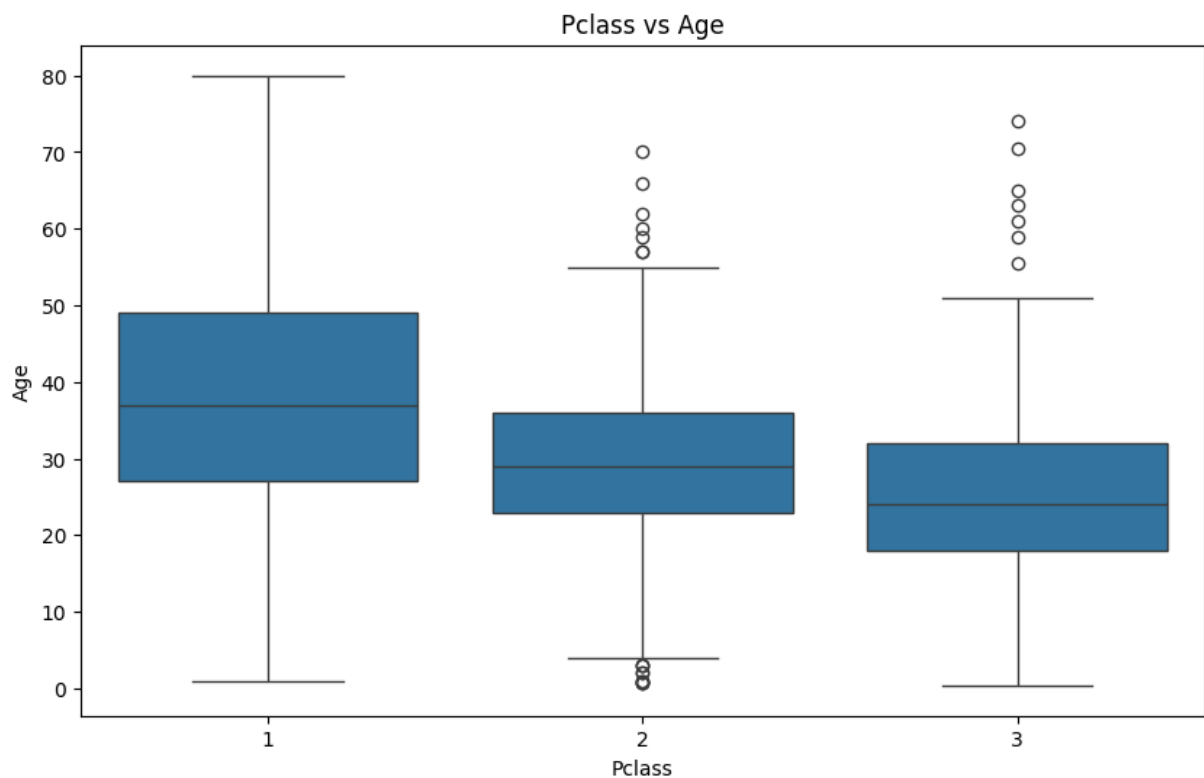
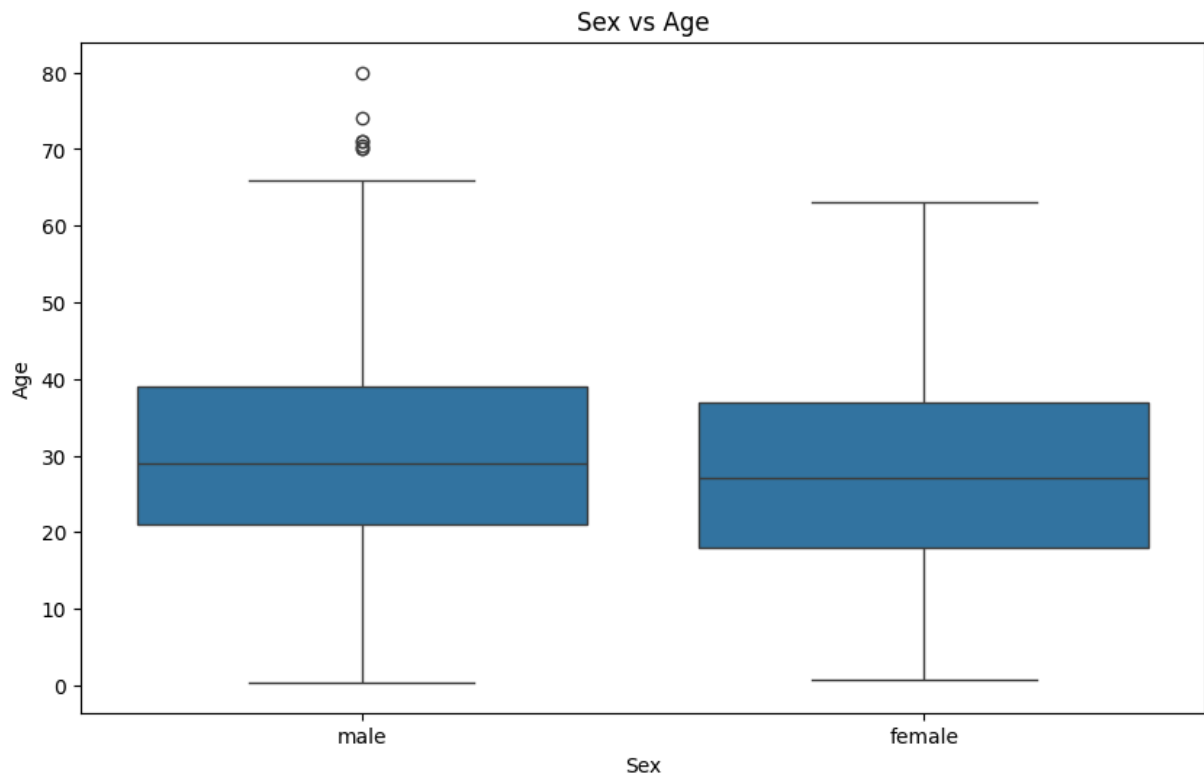
I'm taking different approaches to treating the nulls in 'Cabin' and 'Age'. Since 'Cabin' is mostly null I will just drop the column.

```
In [4]: train.drop('Cabin', axis=1, inplace=True)
        val.drop('Cabin', axis=1, inplace=True)
        test.drop('Cabin', axis=1, inplace=True)
```

For 'Age' I will fill the nulls. I'd rather fill them with group means than the col mean so I'll see if there is a good variable to group 'Age' by.

```
In [6]: cols = ['Embarked', 'Sex', 'Pclass']
        for col in cols:
            plt.figure(figsize=(10, 6))
            x = col
            y = 'Age'
            sns.boxplot(x=x, y=y, data=train)
            plt.title(f'{x} vs {y}')
            plt.show()
```





Based on this I think 'Pclass' makes sense as a grouping variable for 'Age'.

```
In [5]: train['Age'] = train.groupby('Pclass')['Age'].transform(lambda x: x.fillna(x.mean()))
val['Age'] = val.groupby('Pclass')['Age'].transform(lambda x: x.fillna(x.mean()))
test['Age'] = test.groupby('Pclass')['Age'].transform(lambda x: x.fillna(x.mean()))
```

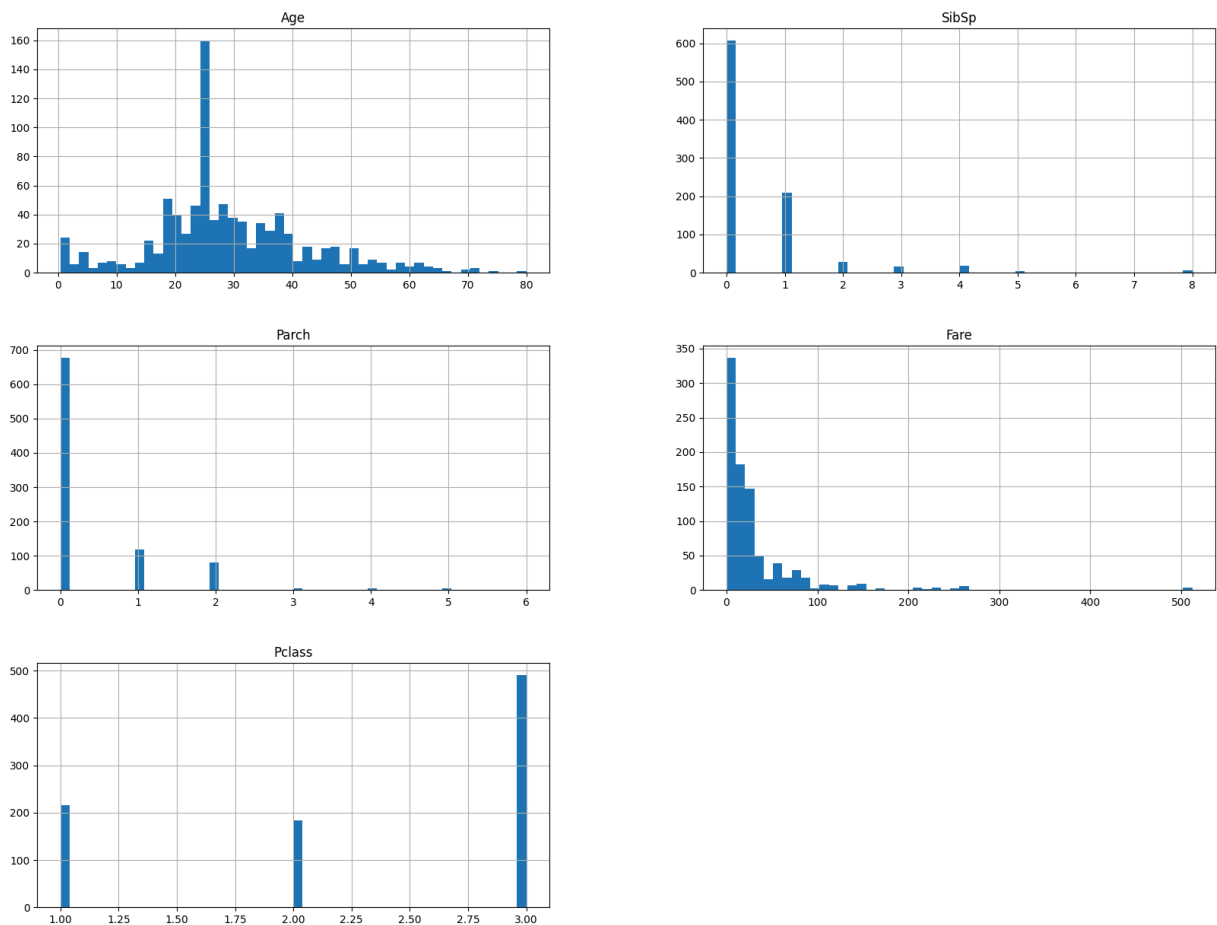
EDA Continued

```
In [8]: value_counts = train['Parch'].value_counts()
        for i in range(len(value_counts)):
            print(f'{value_counts.iloc[i]} passengers had {value_counts.index[i]} parents or children aboard.
```

678 passengers had 0 parents or children aboard.
 118 passengers had 1 parents or children aboard.
 80 passengers had 2 parents or children aboard.
 5 passengers had 5 parents or children aboard.
 5 passengers had 3 parents or children aboard.
 4 passengers had 4 parents or children aboard.
 1 passengers had 6 parents or children aboard.

Plotting first the numerical then the categorical features for more details on what we saw in .describe().

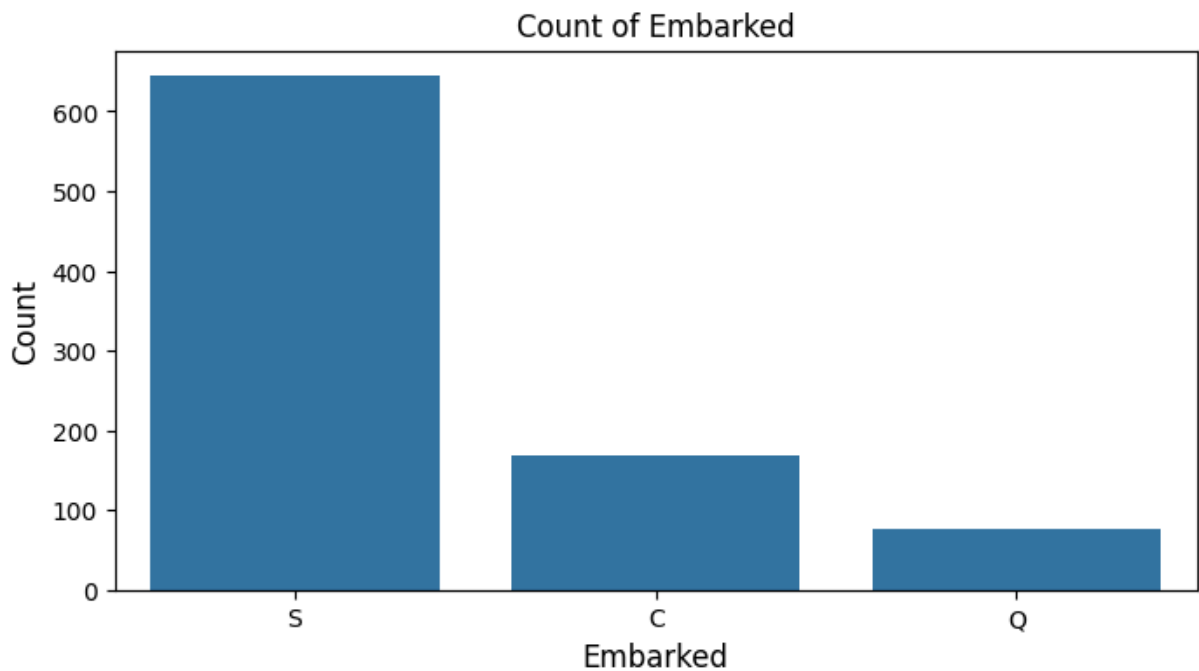
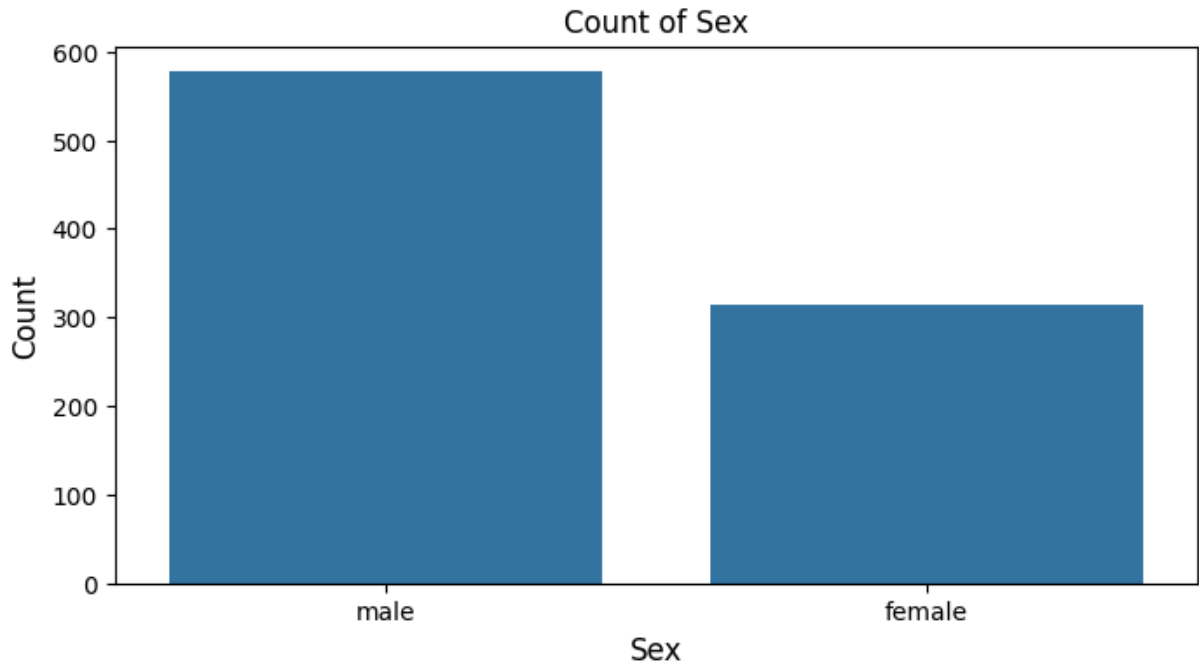
```
In [9]: numerical_features = ['Age', 'SibSp', 'Parch', 'Fare', 'Pclass']
        train[numerical_features].hist(bins=50, figsize=(20, 15))
        plt.show()
```



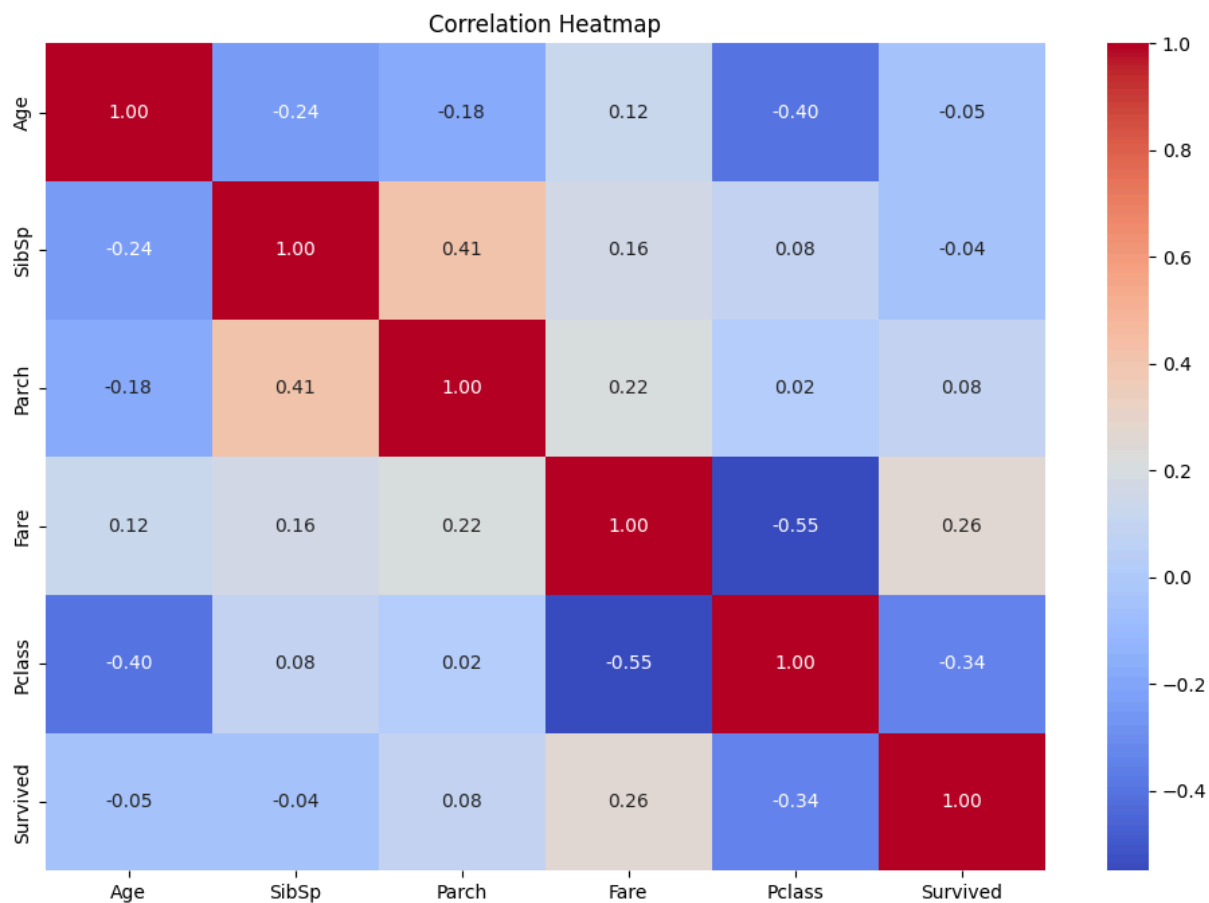
```
In [10]: categorical_features = ['Sex', 'Embarked']

        for col in train[categorical_features].columns:
            plt.figure(figsize=(8, 4))
            sns.countplot(x=col, data=train)
```

```
plt.title(f'Count of {col}')  
plt.xlabel(col, fontsize=12)  
plt.ylabel('Count', fontsize=12)  
plt.show()
```



```
In [11]: corr_features = ['Age', 'SibSp', 'Parch', 'Fare', 'Pclass', 'Survived',]  
plt.figure(figsize=(12, 8))  
sns.heatmap(train[corr_features].corr(), annot=True, fmt=".2f", cmap='coolwarm')  
plt.title('Correlation Heatmap')  
plt.show()
```

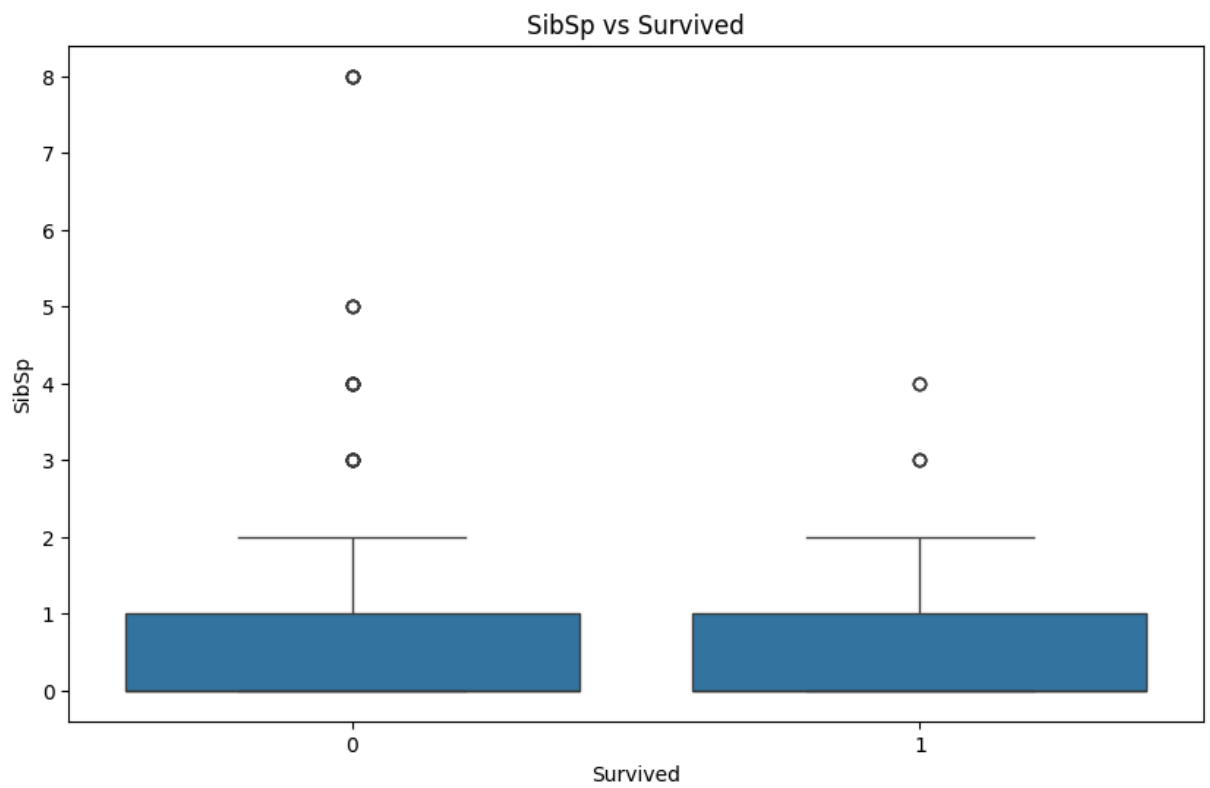
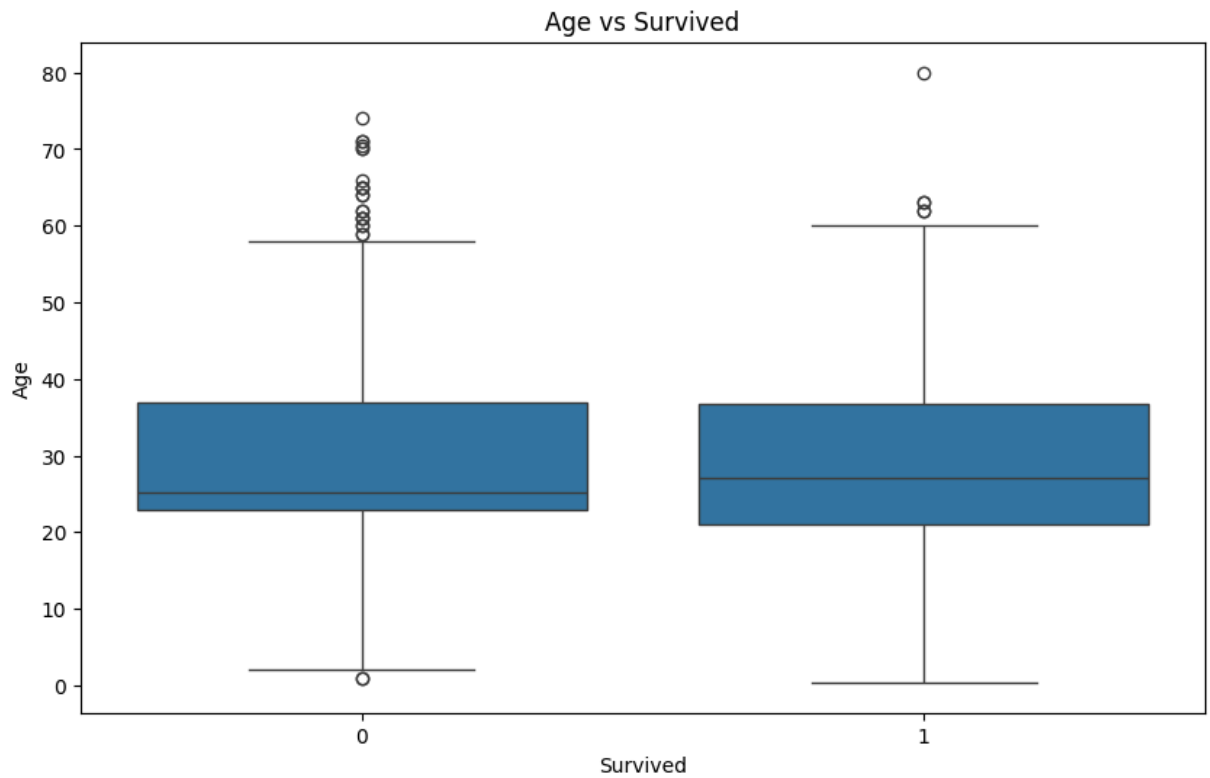


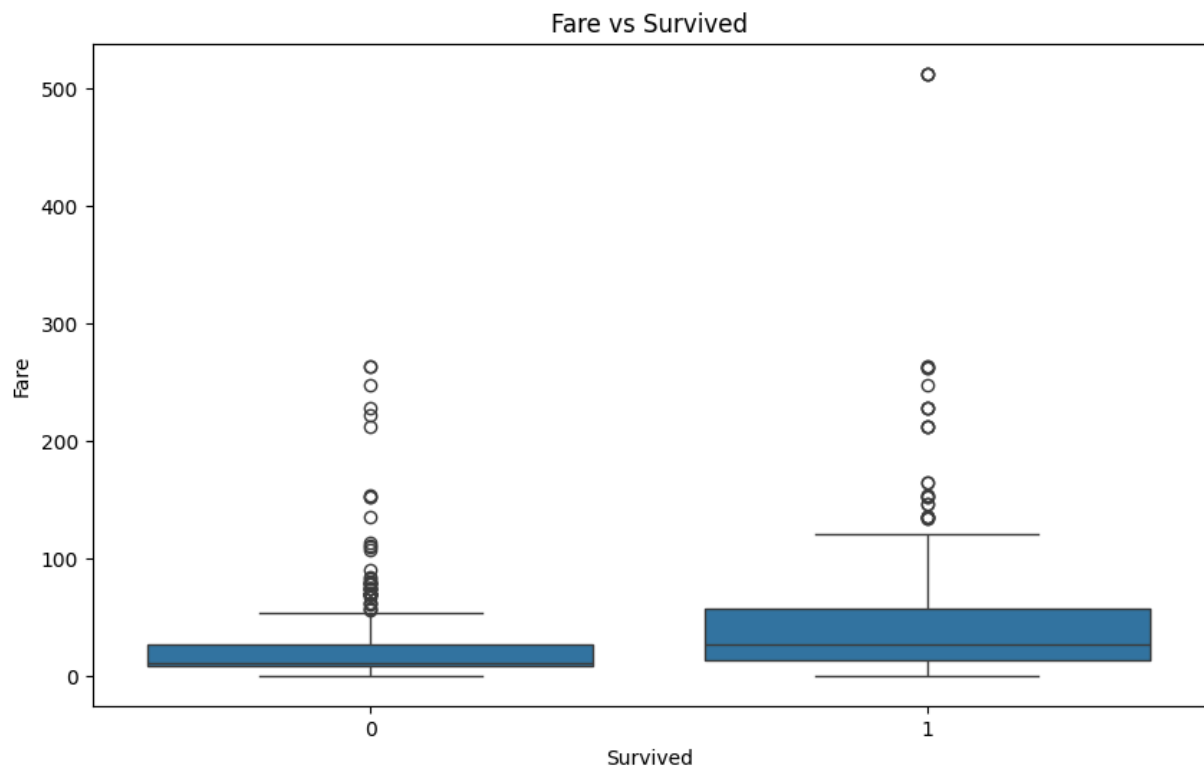
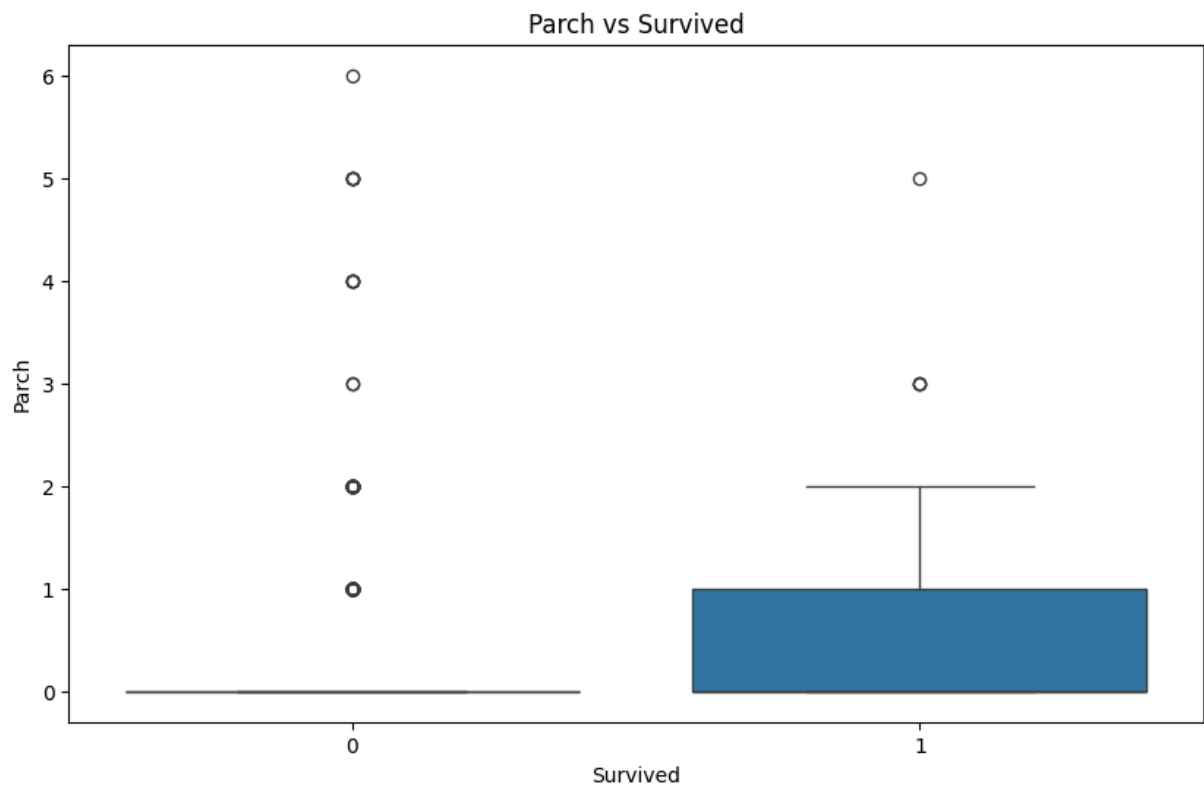
We can see that 'Fare' and 'Pclass' have a negative correlation. Might be worth combining those into one feature.

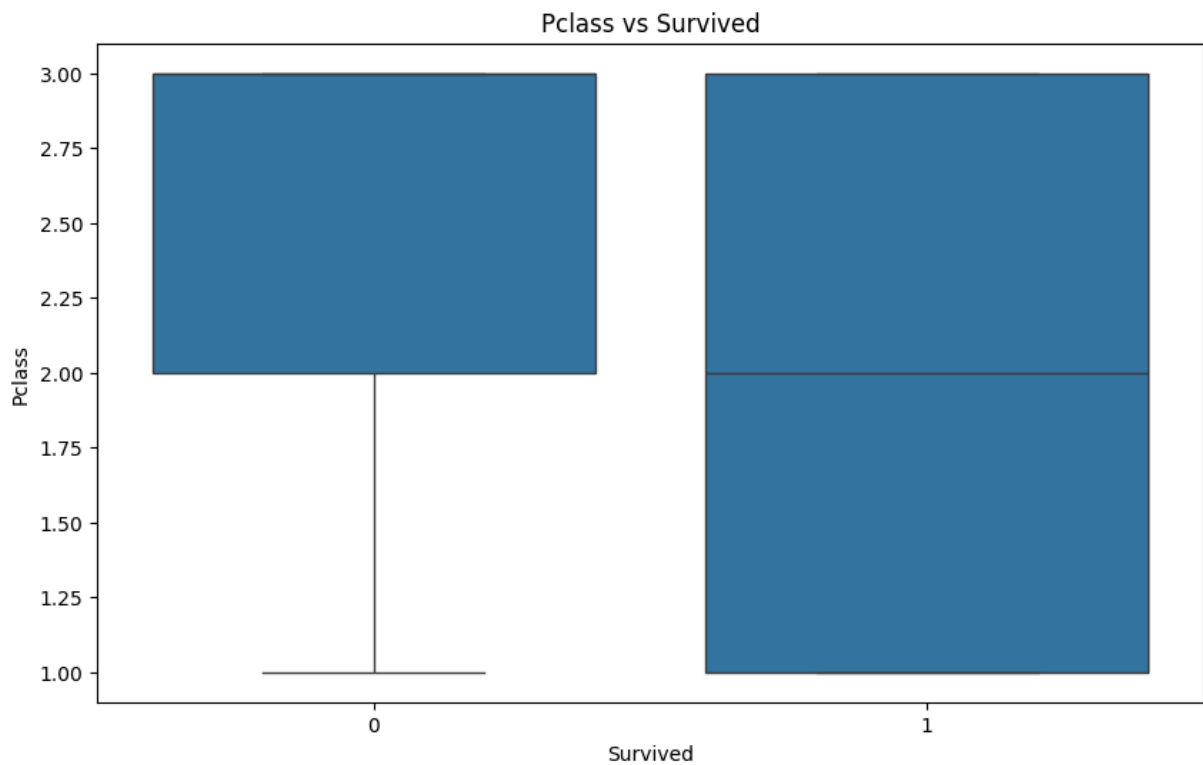
```
In [97]: mean_age_survived = train.groupby('Survived')['Age'].median()
print(mean_age_survived)
```

```
Survived
0    25.162917
1    27.000000
Name: Age, dtype: float64
```

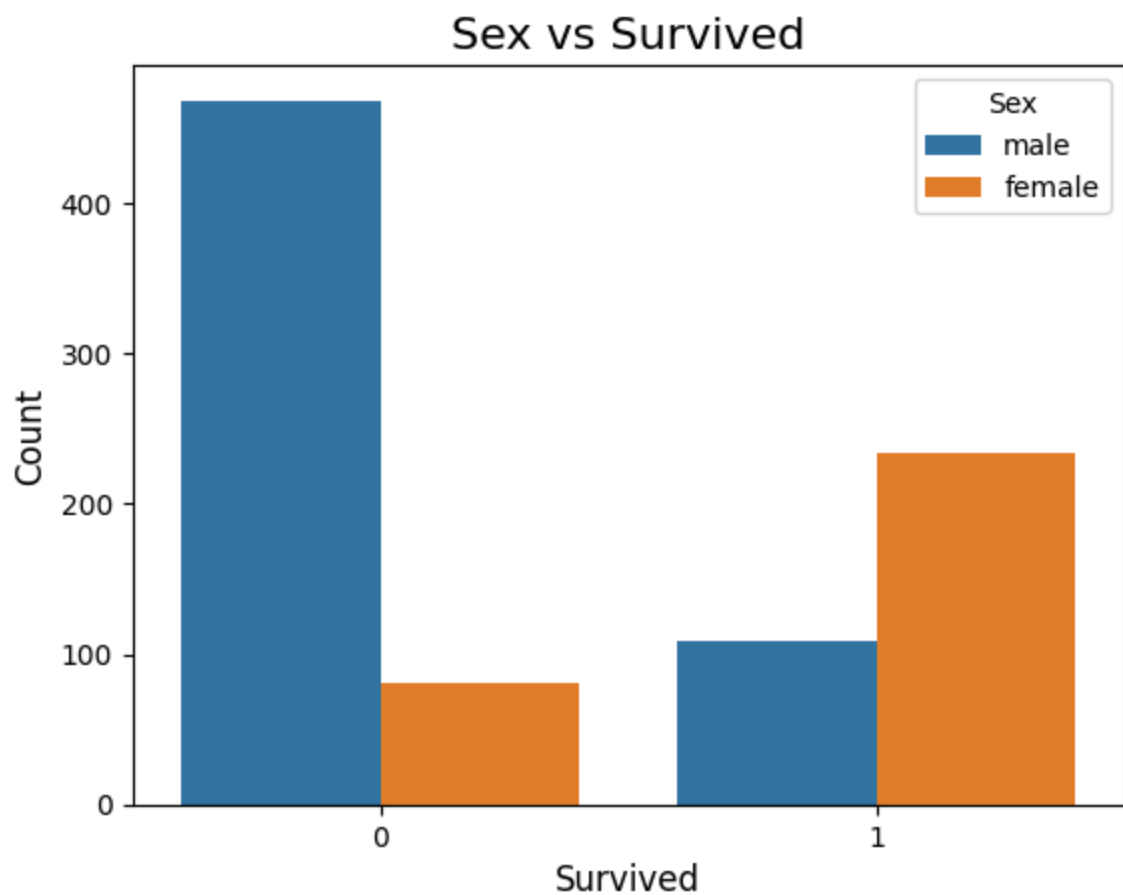
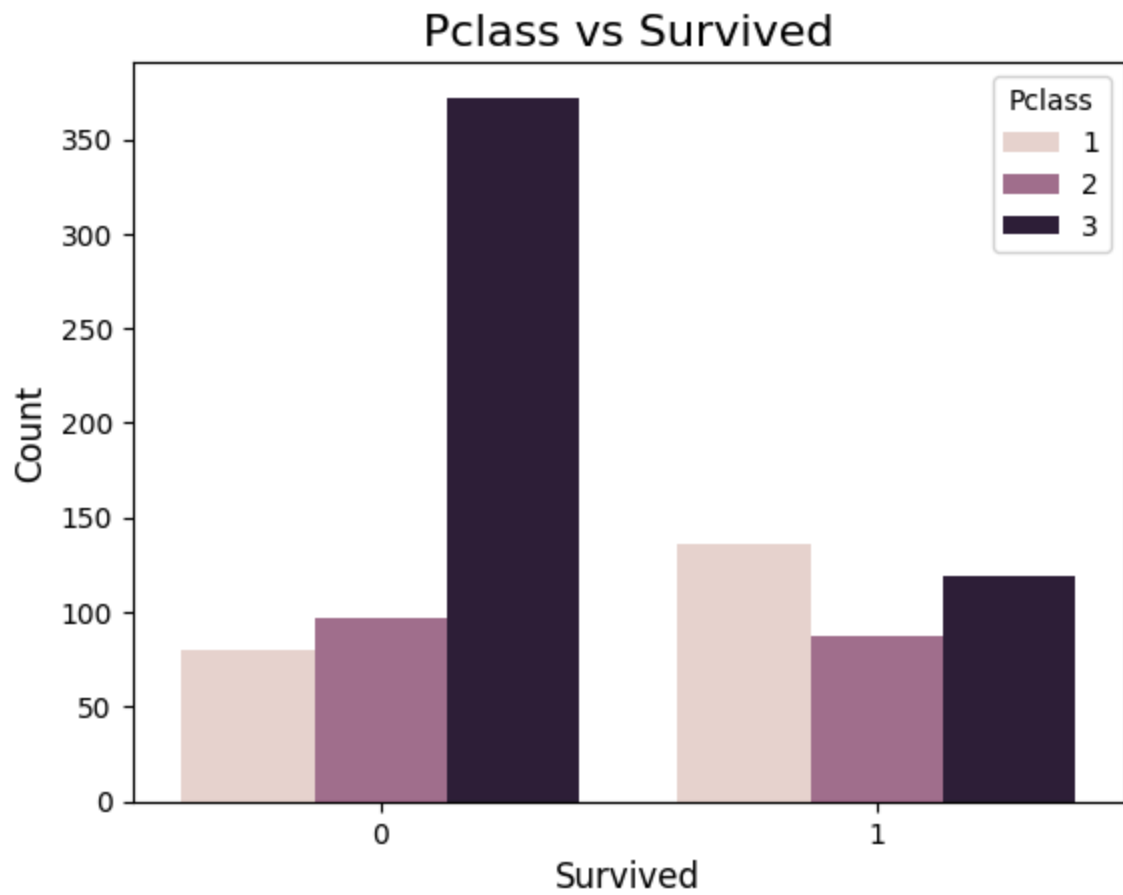
```
In [12]: for column in train[corr_features].columns:
if column != 'Survived':
    plt.figure(figsize=(10, 6))
    sns.boxplot(x='Survived', y=column, data=train)
    plt.title(f'{column} vs Survived')
    plt.show()
```

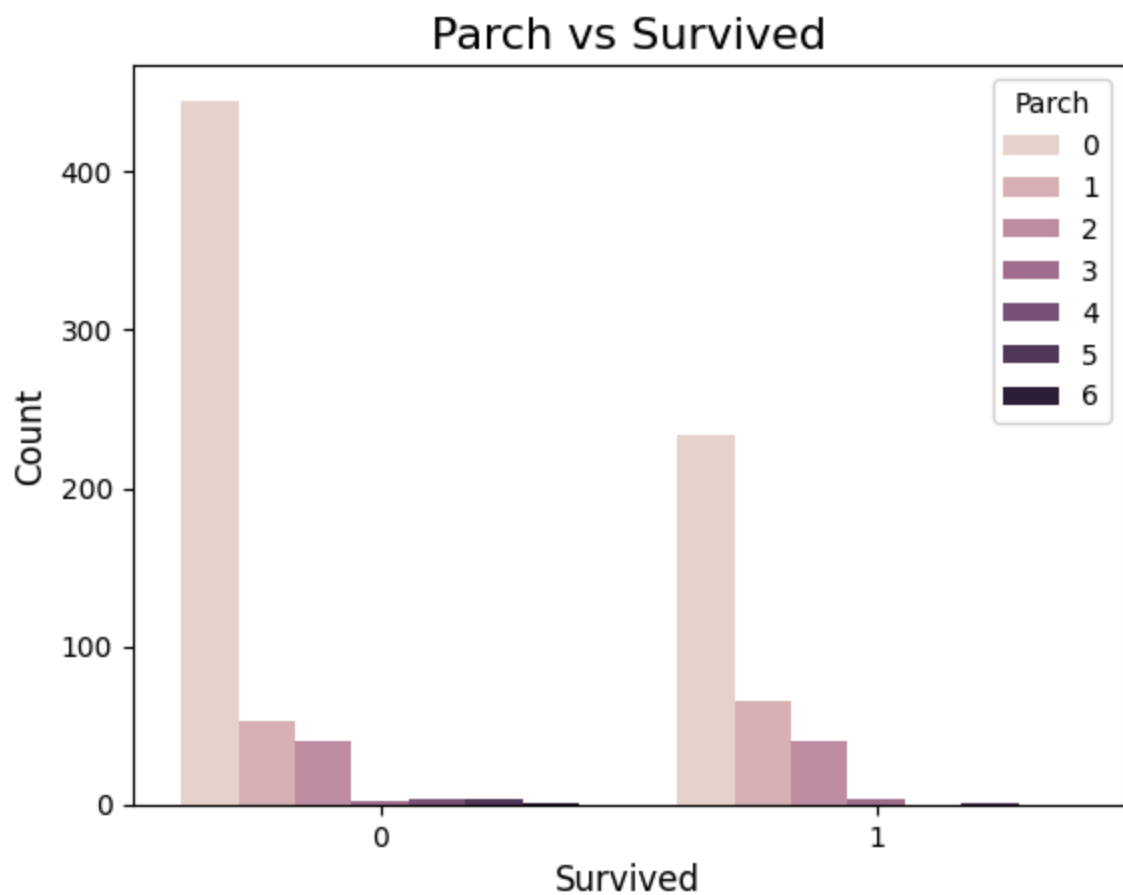
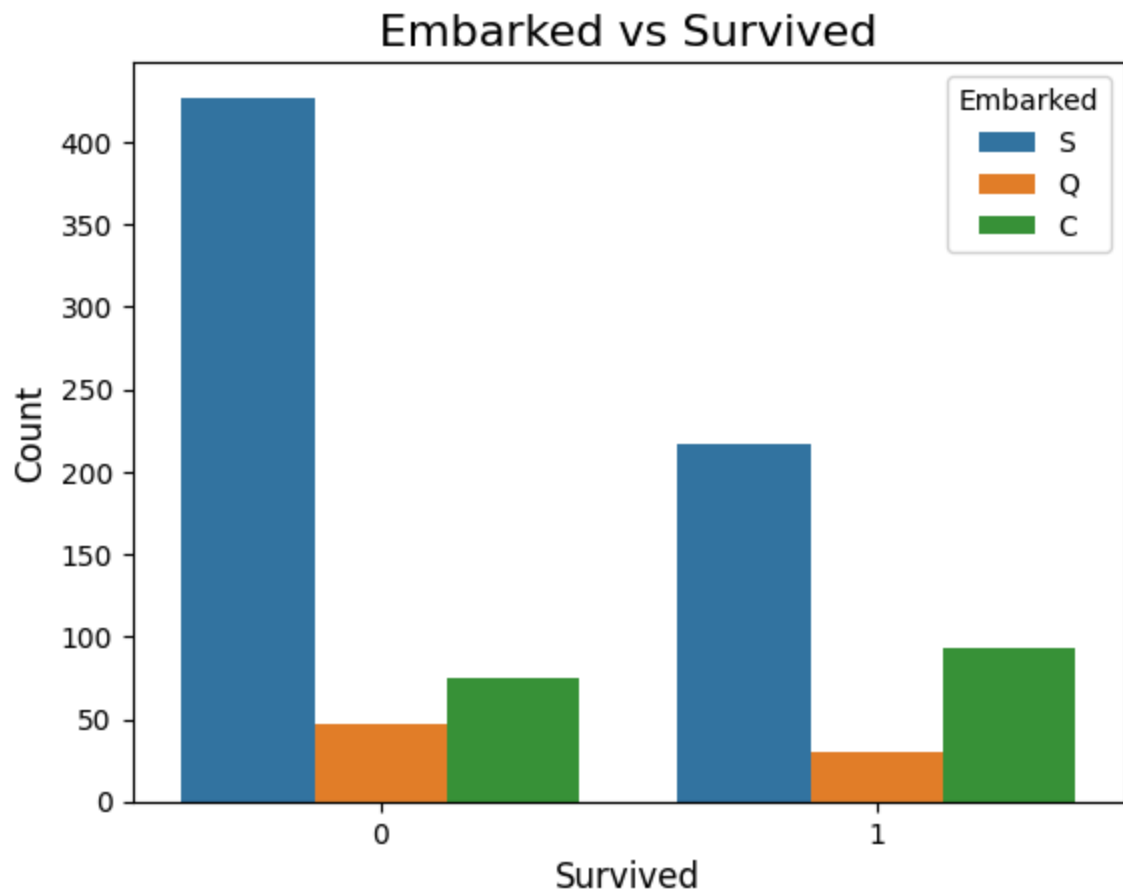







```
In [13]: cat_features = ['Pclass', 'Sex', 'Embarked', 'Parch', 'SibSp']
for col in cat_features:
    sns.countplot(x='Survived', hue=col, data=train)
    plt.title(f'{col} vs Survived', fontsize=16)
    plt.xlabel('Survived', fontsize=12)
    plt.ylabel('Count', fontsize=12)
    plt.legend(title=col)
    plt.show()
```







In [14]: `cat_features = ['Pclass', 'Sex', 'Embarked', 'Parch', 'SibSp']#, 'Parch_binary']`

```
for col in cat_features:
    contingency_table = pd.crosstab(train[col], train['Survived'])
    chi2, p, dof, expected = chi2_contingency(contingency_table)
    if p < 0.05:
        print(f'Chi-squared test for {col} vs Survived:')
        print(f"Chi-squared Statistic: {chi2}")
        print(f"P-value: {p}")
        print(f"Degrees of Freedom: {dof}")
        print(f"Expected Frequencies: \n{expected}")
        print('-----')
    else:
        print(f'{col} is not associated with Survived.')
```

Chi-squared test for Pclass vs Survived:
Chi-squared Statistic: 102.88898875696056
P-value: 4.549251711298793e-23
Degrees of Freedom: 2
Expected Frequencies:
[[133.09090909 82.90909091]
 [113.37373737 70.62626263]
 [302.53535354 188.46464646]]

Chi-squared test for Sex vs Survived:
Chi-squared Statistic: 260.71702016732104
P-value: 1.1973570627755645e-58
Degrees of Freedom: 1
Expected Frequencies:
[[193.47474747 120.52525253]
 [355.52525253 221.47474747]]

Chi-squared test for Embarked vs Survived:
Chi-squared Statistic: 26.48914983923762
P-value: 1.769922284120912e-06
Degrees of Freedom: 2
Expected Frequencies:
[[103.7480315 64.2519685]
 [47.5511811 29.4488189]
 [397.7007874 246.2992126]]

Chi-squared test for Parch vs Survived:
Chi-squared Statistic: 27.925784060236168
P-value: 9.703526421039996e-05
Degrees of Freedom: 6
Expected Frequencies:
[[4.17757576e+02 2.60242424e+02]
 [7.27070707e+01 4.52929293e+01]
 [4.92929293e+01 3.07070707e+01]
 [3.08080808e+00 1.91919192e+00]
 [2.46464646e+00 1.53535354e+00]
 [3.08080808e+00 1.91919192e+00]
 [6.16161616e-01 3.83838384e-01]]

Chi-squared test for SibSp vs Survived:
Chi-squared Statistic: 37.2717929152043
P-value: 1.5585810465902147e-06
Degrees of Freedom: 6
Expected Frequencies:
[[374.62626263 233.37373737]
 [128.77777778 80.22222222]
 [17.25252525 10.74747475]
 [9.85858586 6.14141414]
 [11.09090909 6.90909091]
 [3.08080808 1.91919192]
 [4.31313131 2.68686869]]

```
In [ ]: # train['Female'] = (train['Sex'] == 'female').astype(int)
# val['Female'] = (val['Sex'] == 'female').astype(int)
# test['Female'] = (test['Sex'] == 'female').astype(int)
```

```
In [16]: numerical_features = ['Age', 'SibSp', 'Fare', 'Pclass', 'Parch', 'Female']# 'Parch_

scaler = StandardScaler()
scaled = scaler.fit_transform(train[numerical_features])

pca = PCA(n_components=2)
pca.fit(scaled)

components = pca.components_

loadings = pd.DataFrame(components, columns=numerical_features)

importance = loadings.iloc[0].abs()
importance = importance.sort_values(ascending=False)

print("\nFeature Importance for the First Principal Component:")
print(importance)
```

Feature Importance for the First Principal Component:

```
Fare      0.615065
Pclass    0.534100
Female     0.351838
Parch     0.349243
SibSp     0.244704
Age       0.175473
Name: 0, dtype: float64
```

We can see something really interesting for 'Parch' when we compare the heatmap to the boxplots: while the heatmap shows low correlation between 'Parch' and the target variable, the box plot seems to suggest a relationship.

```
In [17]: n_survived = train[train['Survived'] == 1].shape[0]
n_not_survived = train[train['Survived'] == 0].shape[0]
print(f'{round(n_survived / train.shape[0] * 100)}% of passengers survived.')
```

38% of passengers survived.

```
In [18]: break_feature = 'Survived'
break_two_feature = 'Parch'
subset = train[train[break_feature] == 1]
n = subset.shape[0]

# for class_num in [1,]:
num_class_total = train[train[break_two_feature] == 0].shape[0]
pct_of_total = num_class_total/train.shape[0]
num_class_subset = subset[subset[break_two_feature] == 0].shape[0]
pct_class = num_class_subset/n
pct_of_class = num_class_subset/num_class_total
print(f'{round(pct_of_total*100)}% of passengers did not have parents or children o
print(f'{round(pct_class*100)}% of surviving passengers did not have parents or chi
print(f'{round(pct_of_class*100)}% of passengers who did not have parents or childr
```



```

print('-----')

num_class_total = train[train[break_two_feature] > 0].shape[0]
pct_of_total = num_class_total/train.shape[0]
num_class_subset = subset[subset[break_two_feature] > 0].shape[0]
pct_class = num_class_subset/n
pct_of_class = num_class_subset/num_class_total
print(f'{round(pct_of_total*100)}% of passengers had parents or children on board.')
print(f'{round(pct_class*100)}% of surviving passengers had parents or children on board.')
print(f'{round(pct_of_class*100)}% of passengers who had parents or children on board.')
print('-----')

num_class_total = train[train[break_two_feature] > 1].shape[0]
pct_of_total = num_class_total/train.shape[0]
num_class_subset = subset[subset[break_two_feature] > 1].shape[0]
pct_class = num_class_subset/n
pct_of_class = num_class_subset/num_class_total
print(f'{round(pct_of_total*100)}% of passengers had more than one parents or child')
print(f'{round(pct_class*100)}% of surviving passengers had more than one parents or child')
print(f'{round(pct_of_class*100)}% of passengers who had more than one parents or child')
print('-----')

for class_num in range(1,7):
    num_class_total = train[train[break_two_feature] == class_num].shape[0]
    pct_of_total = num_class_total/train.shape[0]
    num_class_subset = subset[subset[break_two_feature] == class_num].shape[0]
    pct_class = num_class_subset/n
    pct_of_class = num_class_subset/num_class_total
    print(f'{round(pct_of_total*100)}% of passengers had {class_num} parents or child')
    print(f'{round(pct_class*100)}% of surviving passengers had {class_num} parents or child')
    print(f'{round(pct_of_class*100)}% of passengers who had {class_num} parents or child')
    print('-----')

```

76% of passengers did not have parents or children on board.
 68% of surviving passengers did not have parents or children on board.
 34% of passengers who did not have parents or children on board survived.

 24% of passengers had parents or children on board.
 32% of surviving passengers had parents or children on board.
 51% of passengers who had parents or children on board survived.

 11% of passengers had more than one parents or children on board.
 13% of surviving passengers had more than one parents or children on board.
 46% of passengers who had more than one parents or children on board survived.

 13% of passengers had 1 parents or children on board.
 19% of surviving passengers had 1 parents or children on board.
 55% of passengers who had 1 parents or children on board survived.

 9% of passengers had 2 parents or children on board.
 12% of surviving passengers had 2 parents or children on board.
 50% of passengers who had 2 parents or children on board survived.

 1% of passengers had 3 parents or children on board.
 1% of surviving passengers had 3 parents or children on board.
 60% of passengers who had 3 parents or children on board survived.

 0% of passengers had 4 parents or children on board.
 0% of surviving passengers had 4 parents or children on board.
 0% of passengers who had 4 parents or children on board survived.

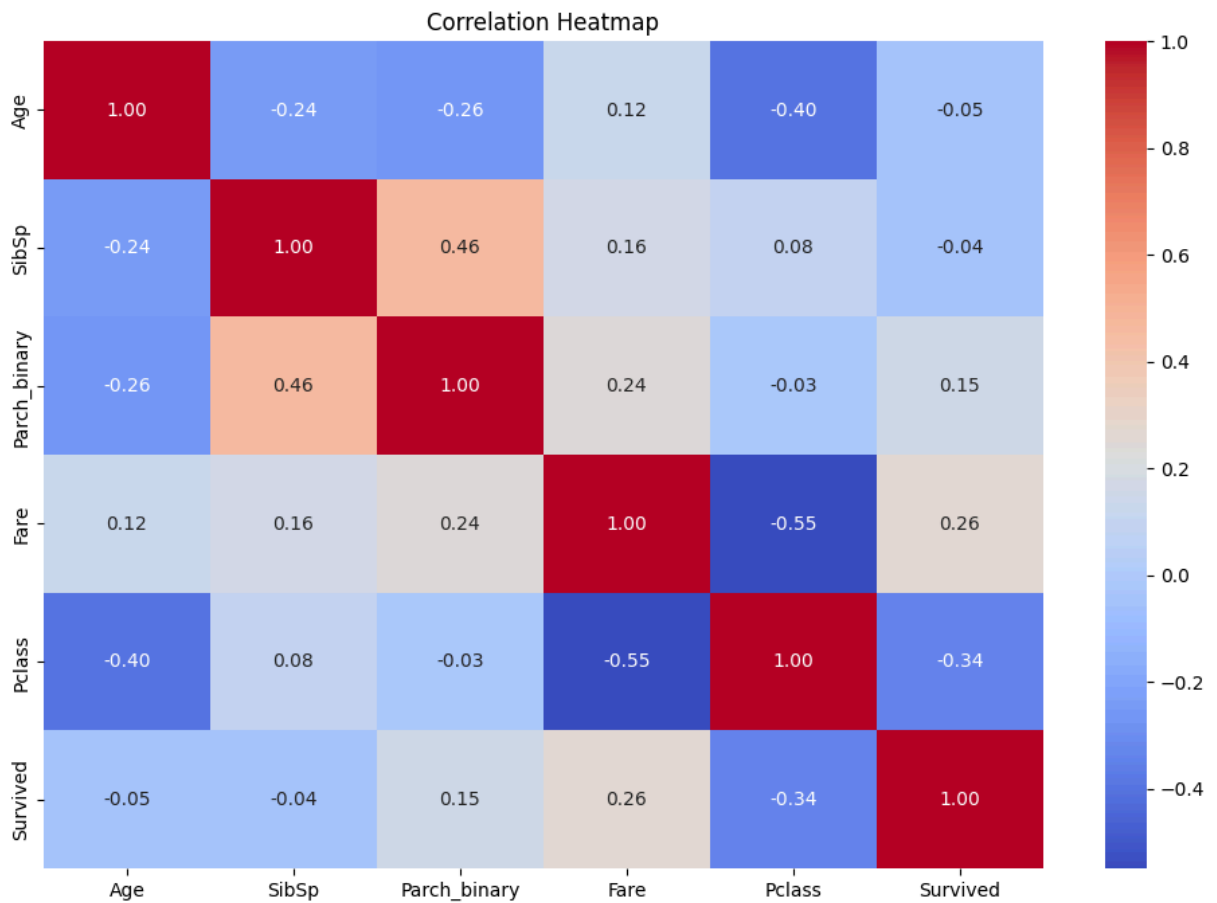
 1% of passengers had 5 parents or children on board.
 0% of surviving passengers had 5 parents or children on board.
 20% of passengers who had 5 parents or children on board survived.

 0% of passengers had 6 parents or children on board.
 0% of surviving passengers had 6 parents or children on board.
 0% of passengers who had 6 parents or children on board survived.

Since survival rate was about the same for passengers who 1 to 3 parents or children on board (and very very few passengers had more than three parents or children on board), I think it makes sense to make 'Parch' into a binary [0='did not have children or parents', 1='had children or parents'] variable.

```
In [6]: train['Parch_binary'] = (train['Parch'] > 0).astype(int)
val['Parch_binary'] = (val['Parch'] > 0).astype(int)
test['Parch_binary'] = (test['Parch'] > 0).astype(int)
```

```
In [20]: corr_features = ['Age', 'SibSp', 'Parch_binary', 'Fare', 'Pclass', 'Survived']
plt.figure(figsize=(12, 8))
sns.heatmap(train[corr_features].corr(), annot=True, fmt=".2f", cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```



```
In [21]: break_feature = 'Survived'
break_two_feature = 'Pclass'
subset = train[train[break_feature] == 1]
n = subset.shape[0]

for class_num in range(1,4):
    num_class_total = train[train[break_two_feature] == class_num].shape[0]
    pct_of_total = num_class_total/train.shape[0]
    num_class_subset = subset[subset[break_two_feature] == class_num].shape[0]
    pct_class = num_class_subset/n
    pct_of_class = num_class_subset/num_class_total
    print(f'{round(pct_of_total*100)}% of passengers had class {class_num} tickets.
    print(f'{round(pct_class*100)}% of surviving passengers had class {class_num} t
    print(f'{round(pct_of_class*100)}% of passengers with class {class_num} tickets
    print('-----')
```

24% of passengers had class 1 tickets.
 40% of surviving passengers had class 1 tickets.
 63% of passengers with class 1 tickets survived.

 21% of passengers had class 2 tickets.
 25% of surviving passengers had class 2 tickets.
 47% of passengers with class 2 tickets survived.

 55% of passengers had class 3 tickets.
 35% of surviving passengers had class 3 tickets.
 24% of passengers with class 3 tickets survived.

```
In [7]: train['Pclass_binary'] = (train['Pclass'] != 3).astype(int)
val['Pclass_binary'] = (val['Pclass'] != 3).astype(int)
test['Pclass_binary'] = (test['Pclass'] != 3).astype(int)
```

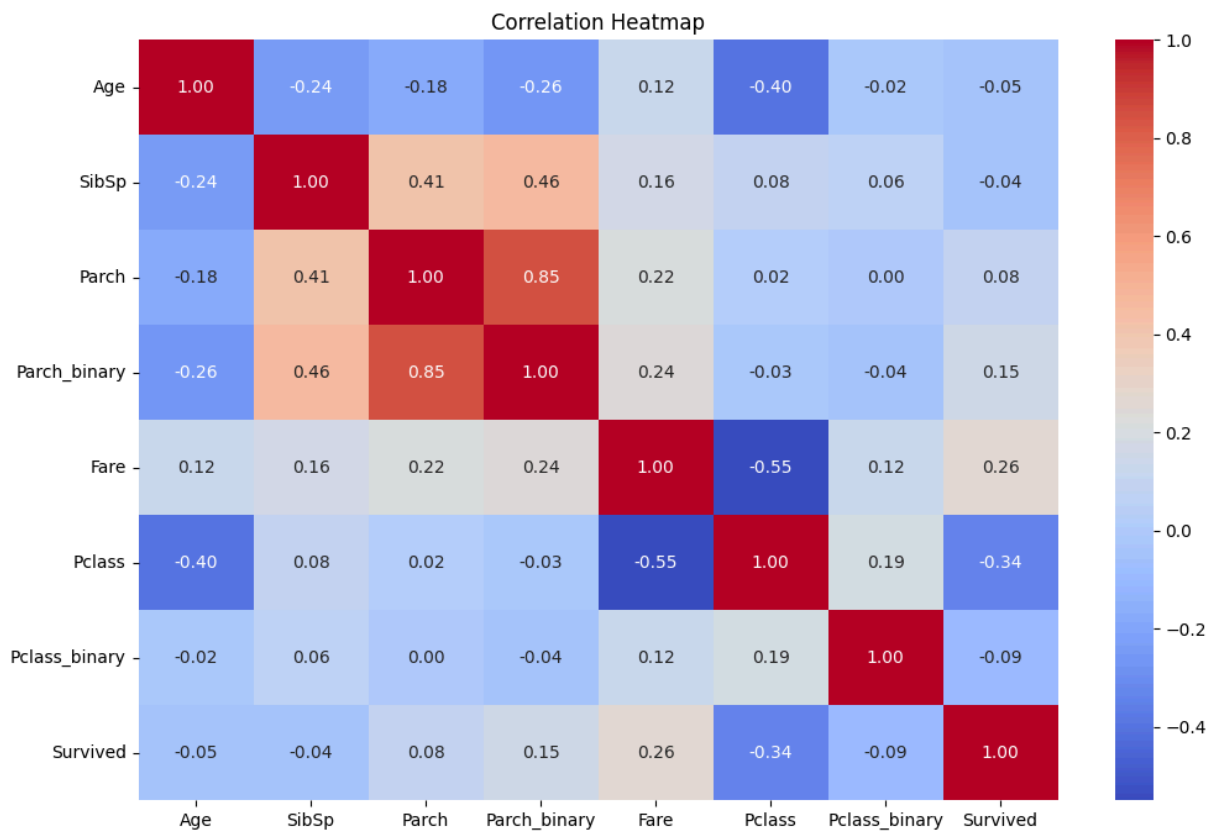
```
In [170]: train.head()
```

```
Out[170]:
```

	PassengerId	Survived	Name	Age	SibSp	Parch	Ticket	Fare	Parch_binary
331	332	0	Partner, Mr. Austen	45.5	0	0	113043	28.5000	(
733	734	0	Berriman, Mr. William John	23.0	0	0	28425	13.0000	(
382	383	0	Tikkanen, Mr. Juho	32.0	0	0	STON/O 2. 3101293	7.9250	(
704	705	0	Hansen, Mr. Henrik Juil	26.0	1	0	350025	7.8542	(
813	814	0	Andersson, Miss. Ebba Iris Alfrida	6.0	4	2	347082	31.2750	1



```
In [23]: corr_features = ['Age', 'SibSp', 'Parch', 'Parch_binary', 'Fare', 'Pclass', 'Pclass_binary']
plt.figure(figsize=(12, 8))
sns.heatmap(train[corr_features].corr(), annot=True, fmt=".2f", cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```



Since several of the features have minimal correlation with the target variable it might make sense to do PCA.

2. Code and results for the model, including performance/accuracy on the training set and on your validation set.

```
In [8]: train.head()
```

Out[8]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fa
331	332	0	1	Partner, Mr. Austen	male	45.5	0	0	113043	28.50
733	734	0	2	Berriman, Mr. William John	male	23.0	0	0	28425	13.00
382	383	0	3	Tikkanen, Mr. Juho	male	32.0	0	0	STON/O 2. 3101293	7.92
704	705	0	3	Hansen, Mr. Henrik Juul	male	26.0	1	0	350025	7.85
813	814	0	3	Andersson, Miss. Ebba Iris Alfrida	female	6.0	4	2	347082	31.27



```
In [10]: train['Fare_binary'] = (train['Fare'] > 10).astype(int)
val['Fare_binary'] = (val['Fare'] > 10).astype(int)
test['Fare_binary'] = (test['Fare'] > 10).astype(int)
```

```
In [8]: train = pd.get_dummies(train, columns=['Embarked'], drop_first=True)
val = pd.get_dummies(val, columns=['Embarked'], drop_first=True)
test = pd.get_dummies(test, columns=['Embarked'], drop_first=True)

train = pd.get_dummies(train, columns=['Pclass'], drop_first=True)
val = pd.get_dummies(val, columns=['Pclass'], drop_first=True)
test = pd.get_dummies(test, columns=['Pclass'], drop_first=True)

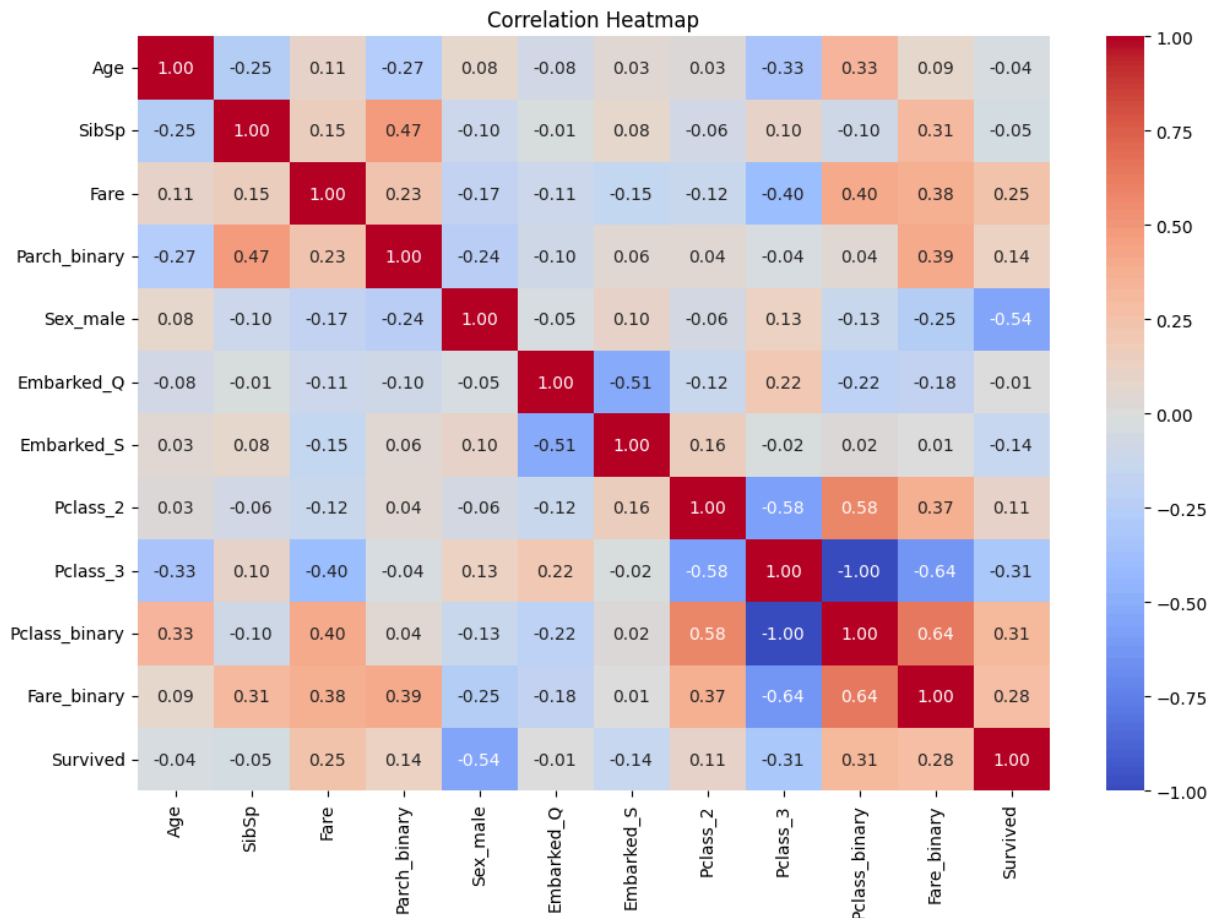
train = pd.get_dummies(train, columns=['Sex'], drop_first=True)
val = pd.get_dummies(val, columns=['Sex'], drop_first=True)
test = pd.get_dummies(test, columns=['Sex'], drop_first=True)
train.head()
```

Out[8]:

	PassengerId	Survived	Name	Age	SibSp	Parch	Ticket	Fare	Parch_binary
331	332	0	Partner, Mr. Austen	45.5	0	0	113043	28.5000	(
733	734	0	Berriman, Mr. William John	23.0	0	0	28425	13.0000	(
382	383	0	Tikkanen, Mr. Juho	32.0	0	0	STON/O 2. 3101293	7.9250	(
704	705	0	Hansen, Mr. Henrik Juul	26.0	1	0	350025	7.8542	(
813	814	0	Andersson, Miss. Ebba Iris Alfrida	6.0	4	2	347082	31.2750	1



```
In [52]: corr_features = ['Age', 'SibSp', 'Fare', 'Parch_binary', 'Sex_male', 'Embarked_Q',
plt.figure(figsize=(12, 8))
sns.heatmap(train[corr_features].corr(), annot=True, fmt=".2f", cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```



```
In [11]: # X_train = train[['Age', 'SibSp', 'Fare', 'Parch_binary', 'Sex_male', 'Embarked_Q']
X_train = train[['Age', 'SibSp', 'Fare_binary', 'Parch_binary', 'Sex_male', 'Embarked_Q']
y_train = train['Survived']

# X_val = val[['Age', 'SibSp', 'Fare', 'Parch_binary', 'Sex_male', 'Embarked_Q', 'Embarked_S']
X_val = val[['Age', 'SibSp', 'Fare_binary', 'Parch_binary', 'Sex_male', 'Embarked_Q', 'Embarked_S']
y_val = val['Survived']

# X_test = test[['Age', 'SibSp', 'Fare', 'Parch_binary', 'Sex_male', 'Embarked_Q', 'Embarked_S']
X_test = test[['Age', 'SibSp', 'Fare_binary', 'Parch_binary', 'Sex_male', 'Embarked_Q', 'Embarked_S']
```

```
In [94]: dtrain = xgb.DMatrix(X_train, label=y_train)
dval = xgb.DMatrix(X_val, label=y_val)

model_hyperparams = {
    'eta': [.01, 0.1, 0.3],
    'max_depth': [3, 6, 9],
    'subsample' : [0.8, 1],
    'colsample_bytree' : [0.8, 1],
}

keys = model_hyperparams.keys()
values = model_hyperparams.values()

# Generate all parameter combinations from model_hyperparams using itertools.product
combinations = itertools.product(*values)
```



```

num_boost_round = 100

best_val_accuracy = 0
best_combo = None

# Iterate through each parameter combination and run model
for combo in combinations:
    combo = dict(zip(keys, combo))
    combo['objective']='binary:logistic'
    combo['eval_metric'] = 'logloss'
    combo['seed'] = 42
    model = xgb.train(combo, dtrain, num_boost_round)
    y_pred = model.predict(dval)
    y_pred_binary = [1 if pred > 0.5 else 0 for pred in y_pred]

    val_accuracy = accuracy_score(y_val, y_pred_binary)

    if val_accuracy > best_val_accuracy:
        print(f'New best validation accuracy: {val_accuracy * 100:.2f}%')
        print('      Hyperparameters:', combo)
        best_combo = combo
        best_val_accuracy = val_accuracy

    # print(classification_report(y_val, y_pred_binary))

    # cm = confusion_matrix(y_val, y_pred_binary)
    # print("Confusion Matrix:\n", cm)
    print('-----')

```

```

New best validation accuracy: 80.45%
      Hyperparameters: {'eta': 0.01, 'max_depth': 3, 'subsample': 0.8, 'colsample_bytree': 0.8, 'objective': 'binary:logistic', 'eval_metric': 'logloss', 'seed': 42}
-----
New best validation accuracy: 82.68%
      Hyperparameters: {'eta': 0.01, 'max_depth': 9, 'subsample': 1, 'colsample_bytree': 1, 'objective': 'binary:logistic', 'eval_metric': 'logloss', 'seed': 42}
-----
New best validation accuracy: 83.24%
      Hyperparameters: {'eta': 0.1, 'max_depth': 6, 'subsample': 0.8, 'colsample_bytree': 0.8, 'objective': 'binary:logistic', 'eval_metric': 'logloss', 'seed': 42}
-----
New best validation accuracy: 83.80%
      Hyperparameters: {'eta': 0.1, 'max_depth': 6, 'subsample': 1, 'colsample_bytree': 1, 'objective': 'binary:logistic', 'eval_metric': 'logloss', 'seed': 42}
-----
New best validation accuracy: 84.36%
      Hyperparameters: {'eta': 0.1, 'max_depth': 9, 'subsample': 1, 'colsample_bytree': 0.8, 'objective': 'binary:logistic', 'eval_metric': 'logloss', 'seed': 42}
-----
New best validation accuracy: 84.92%
      Hyperparameters: {'eta': 0.3, 'max_depth': 3, 'subsample': 1, 'colsample_bytree': 0.8, 'objective': 'binary:logistic', 'eval_metric': 'logloss', 'seed': 42}
-----

```

```
In [ ]: model = xgb.train(best_combo, dtrain, num_boost_round)
```

```
dtest = xgb.DMatrix(X_test)
y_pred = model.predict(dtest)
y_pred_binary = [1 if pred > 0.5 else 0 for pred in y_pred]


print(y_pred_binary[:10])
```

```
[0, 0, 0, 0, 0, 0, 1, 0, 1, 0]
```





```
In [81]: model_name = 'XGBClassifier'
test_predictions = pd.DataFrame()
test_predictions['PassengerId'] = test['PassengerId']
test_predictions['Survived'] = y_pred_binary
test_predictions.to_csv(f'data/titanic/{model_name}_test_predictions.csv', index=False)
```

3. A snapshot of the submission including your team name (as an individual, you're a team of 1 person), as shown on the Kaggle leaderboard.

As a note, I changed my last name a couple years ago so my name still shows up in some locations (such as Kaggle) as Cate Merfeld even though my name at WPI has been updated to Cate Dunham.

14812	Cate Merfeld		0.72727	1	11s
-------	--------------	---	---------	---	-----

I was able to increase the test accuracy by changing the prediction binarization threshold up to .9, but I found that increasing it above .7 increased the rate of false negatives. The image below shows results for thresholds of .6,.7,.8 and .9:

	XGBClassifier_test_predictions.csv Complete · now	0.75837
	XGBClassifier_test_predictions.csv Complete · 2m ago	0.75598
	XGBClassifier_test_predictions.csv Complete · 3m ago	0.74401
	XGBClassifier_test_predictions.csv Complete · 6m ago	0.73205

4. Commentary on the relative performance of your model, relative to that of other teams. It is not necessary to have a high performing model at this point, but you should have some idea about next steps for improving the model.

My first submission was number 14,812 out of 15,341 total entries, meaning it was in the bottom 3% of entries. I can't see where my other XGB submissions fell in the leaderboard since I submitted them after I had submitted other models for problems 2 and 3 that had higher scores, but in terms of accuracy I can see that my XGB model with the highest accuracy had a higher accuracy score than my random forest model, meaning it would probably have come in somewhere above the 13,105 that the RF was ranked at.

Problem 2 - Classification Model 2:

Repeat Problem 1, but this time with a different model. It is sufficient to try the same technique but with different hyperparameter values. However, you will need to provide comments about the relative performance of your new model compared to your Classification Model 1. Provide all of the information as listed for Classification Model 1 (code, results, performance, etc.).

For my second model I tried a random forest, which outperformed my XGB models with prediction binarization thresholds under .7.

```
In [ ]: rf = RandomForestClassifier(n_estimators=10, random_state=42)
rf.fit(X_train, y_train)
print(f'Training Accuracy: {rf.score(X_train, y_train)*100:.2f}%')

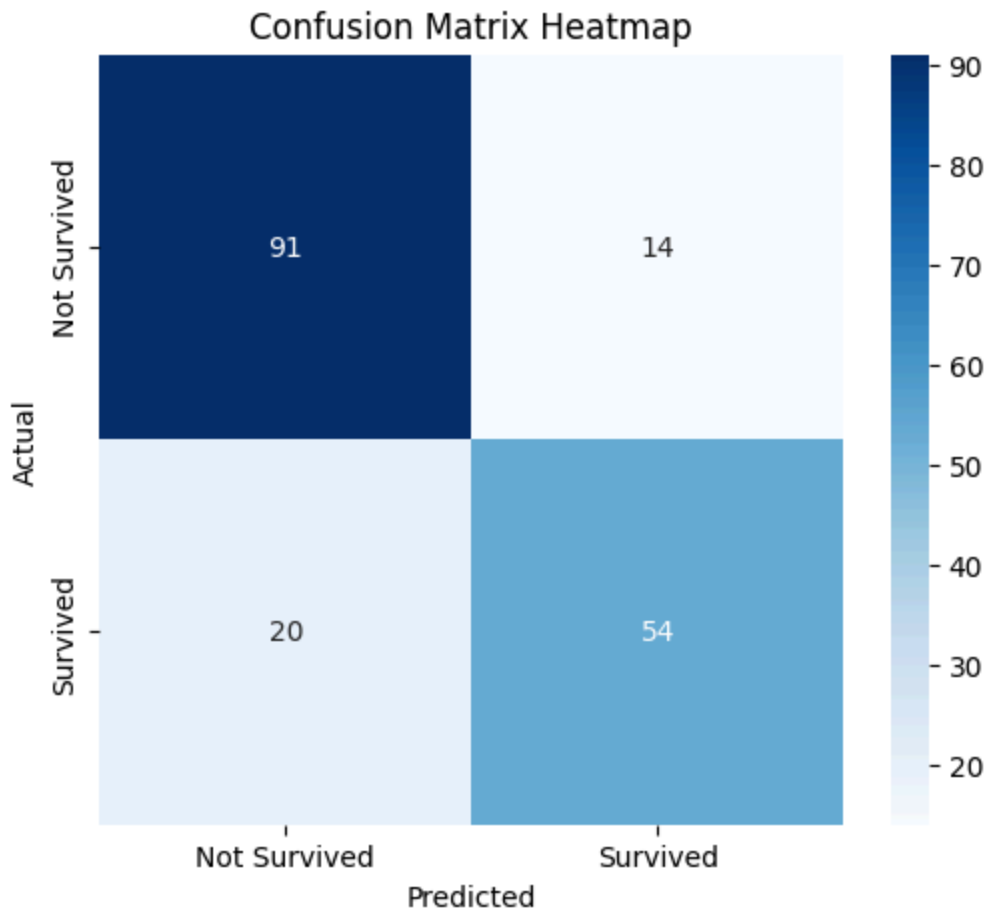
print(f'Validation Accuracy: {rf.score(X_val, y_val)*100:.2f}%')
```

Training Accuracy: 93.40%
Validation Accuracy: 81.01%

```
In [ ]: from sklearn.metrics import confusion_matrix
# import numpy as np

y_pred = rf.predict(X_val)
cm = confusion_matrix(y_val, y_pred)

# Plot heatmap
plt.figure(figsize=(6,5))
sns.heatmap(cm, annot=True, cmap="Blues", xticklabels=['Not Survived', 'Survived'],
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix Heatmap")
plt.show()
```



```
In [ ]: test_preds = pd.DataFrame()
test_preds['PassengerId'] = test['PassengerId']
test_preds['Survived'] = rf.predict(X_test)
test_preds.to_csv('data/titanic/rf_test_preds.csv', index=False)
```

13105	Cate Merfeld		0.75119	2	13s
-------	--------------	--	---------	---	-----

Although an improvement from my original XGB model, my random forest was still in the bottom 15% of entries.

Problem 3 - Classification Model 3:

Repeat Problem 1 yet again, but this time with a different model from either Problem 1 or 2. You should be looking not for perfection, but for improvement in accuracy on your validation set, and ultimately on your position on the leaderboard (although this is not absolutely necessary). If you find the accuracy going down, you should comment as to possible reasons why. Submit all code, results, comments on relative performance as you did in Problem 2.

```

In [ ]: max_depths = [7, 8, 9, 10]
        n_estimators = [25, 50, 100, 200]

        best_val_accuracy = 0
        best_depth = None
        best_estimators = None

        for depth in max_depths:
            for n in n_estimators:
                base_model = DecisionTreeClassifier(max_depth=depth)

                # Initialize and train BaggingClassifier
                bagging_model = BaggingClassifier(estimator=base_model, n_estimators=n, random_state=42)
                bagging_model.fit(X_train, y_train)

                print(f'Results for Bagging Classifier with Base Model Max Depth of {depth}')

                # Print training accuracy
                print(f'      Training Accuracy: {bagging_model.score(X_train, y_train) * 100:.8f}%')

                # Make predictions on validation set
                bagging_predictions = bagging_model.predict(X_val)

                val_accuracy = bagging_model.score(X_val, y_val)
                # Print validation accuracy
                print(f'      Validation Accuracy: {val_accuracy * 100:.8f}%')

                if val_accuracy > best_val_accuracy:
                    print(f'      New best validation accuracy: {val_accuracy * 100:.8f}%')
                    best_depth = depth
                    best_estimators = n
                    best_val_accuracy = val_accuracy
                print('-----')

        print(f'Best Validation Accuracy: {best_val_accuracy * 100:.8f}%')
        print(f'Best Max Depth: {best_depth}')
        print(f'Best Number of Estimators: {best_estimators}')

```

Results for Bagging Classifier with Base Model Max Depth of 7 and 25 Estimators:
Training Accuracy: 88.20%
Validation Accuracy: 79.88826816%
New best validation accuracy: 79.88826816%

Results for Bagging Classifier with Base Model Max Depth of 7 and 50 Estimators:
Training Accuracy: 88.34%
Validation Accuracy: 79.88826816%

Results for Bagging Classifier with Base Model Max Depth of 7 and 100 Estimators:
Training Accuracy: 89.04%
Validation Accuracy: 77.65363128%

Results for Bagging Classifier with Base Model Max Depth of 7 and 200 Estimators:
Training Accuracy: 88.34%
Validation Accuracy: 78.77094972%

Results for Bagging Classifier with Base Model Max Depth of 8 and 25 Estimators:
Training Accuracy: 90.45%
Validation Accuracy: 79.88826816%

Results for Bagging Classifier with Base Model Max Depth of 8 and 50 Estimators:
Training Accuracy: 90.87%
Validation Accuracy: 79.32960894%

Results for Bagging Classifier with Base Model Max Depth of 8 and 100 Estimators:
Training Accuracy: 90.59%
Validation Accuracy: 79.88826816%

Results for Bagging Classifier with Base Model Max Depth of 8 and 200 Estimators:
Training Accuracy: 90.17%
Validation Accuracy: 79.88826816%

Results for Bagging Classifier with Base Model Max Depth of 9 and 25 Estimators:
Training Accuracy: 91.99%
Validation Accuracy: 78.21229050%

Results for Bagging Classifier with Base Model Max Depth of 9 and 50 Estimators:
Training Accuracy: 91.99%
Validation Accuracy: 80.44692737%
New best validation accuracy: 80.44692737%

Results for Bagging Classifier with Base Model Max Depth of 9 and 100 Estimators:
Training Accuracy: 91.85%
Validation Accuracy: 79.88826816%

Results for Bagging Classifier with Base Model Max Depth of 9 and 200 Estimators:
Training Accuracy: 91.99%
Validation Accuracy: 81.56424581%
New best validation accuracy: 81.56424581%

Results for Bagging Classifier with Base Model Max Depth of 10 and 25 Estimators:
Training Accuracy: 93.12%
Validation Accuracy: 77.65363128%

Results for Bagging Classifier with Base Model Max Depth of 10 and 50 Estimators:

Training Accuracy: 92.56%
 Validation Accuracy: 79.32960894%

 Results for Bagging Classifier with Base Model Max Depth of 10 and 100 Estimators:
 Training Accuracy: 92.84%
 Validation Accuracy: 81.56424581%

 Results for Bagging Classifier with Base Model Max Depth of 10 and 200 Estimators:
 Training Accuracy: 92.84%
 Validation Accuracy: 81.56424581%


 Best Validation Accuracy: 81.56424581%
 Best Max Depth: 9
 Best Number of Estimators: 200

```
In [ ]: base_model = DecisionTreeClassifier(max_depth=best_depth)

# Initialize and train BaggingClassifier
bagging_model = BaggingClassifier(estimator=base_model, n_estimators=best_estimator)
bagging_model.fit(X_train, y_train)

model_name = 'BaggingClassifier'
test_predictions = pd.DataFrame()
test_predictions['PassengerId'] = test['PassengerId']
test_predictions['Survived'] = bagging_model.predict(X_test)
test_predictions.to_csv(f'data/titanic/{model_name}_test_predictions.csv', index=False)
```

With a test accuracy of ~78%, my bagging model outperformed any of my other models. It ranked in the top 24% of entries.

3617	Cate Merfeld		0.77990	3	11s
------	--------------	---	---------	---	-----

Compiling Each Model's Results for the Assignment:

Each student will submit a pdf document that shows their code, output, comments, visualizations etc.. This will be accomplished by building a IPython/Jupyter notebook to complete the assignment and then after completion, converting the document to pdf format.

As part of this assignment you will not only download the titanic dataset but you'll also join the competition (by clicking on the "Join the Competition" button on the Kaggle/Titanic webpage).

For each model, you'll use the training dataset to train a model, but also estimate the accuracy of your model. You'll accomplish this by splitting the training set (in the training file

downloaded from Kaggle) into two subsets - an actual training set on which you'll train the model, and then a validation set (say, an 80/20 split, but that's just a suggestion) on which you'll run the trained model to see how accurate it is. You'll be able to get an accurate performance reading as you'll know what the actual results are for these records (they were part of the original training set you downloaded).

Next, you'll run your model on the actual test set data file you downloaded from Kaggle. The label values (whether the passenger survived, 1 or 0), for the records in the test file are not given. So, you are not able to determine the accuracy of your model against the test set by simply running your model on these records. Instead, you'll submit the test file with your model's predictions for these passengers to Kaggle, where the platform will compare your model's predicted values with the actual, (hidden) values to give you an accuracy score. Note, you won't be given the correct/incorrect breakdown of the individual test records for your model - you'll only see the overall accuracy score. It will be left to you to take this coarse-grained accuracy info and think of ways to improve your model.

After the Kaggle platform determines your model's accuracy, it will post your "team" name (since you're each doing this individually, you each constitute a team of 1, and you can name your team however you like), along with your accuracy at the appropriate position on the leaderboard. Take a screenshot of your team position on the leaderboard and include it for each of the 3 models you develop as part of the assignment.

Figures 1, 2 and 3, below, show images of the Kaggle leaderboard, including individual entrants, the number of submissions they've made, and the accuracy of their last submission on the Kaggle test set.

Please understand I am not looking for you to have a 100% accuracy for any of your models. The important aspect is after starting with a simple model, you start thinking about ways to improve the model, record your reasoning for making these changes, determining whether the model has improved after submitting against the Kaggle test set, and providing further reasoning about any change to performance (positive or negative).