Titanic - Who will survive?

CSE 351

Joseph Jeong

Cleaning Data

```
# mapping values in the 'Embarked' column to numerical values
embarked_mapping = {'C': 1, 'Q': 2, 'S': 3}
df['Embarked'] = df['Embarked'].map(embarked_mapping)

# mapping values in the 'Sex' column to numerical values
sex_mapping = {'male': 0, 'female': 1}
df['Sex'] = df['Sex'].map(sex_mapping)
```

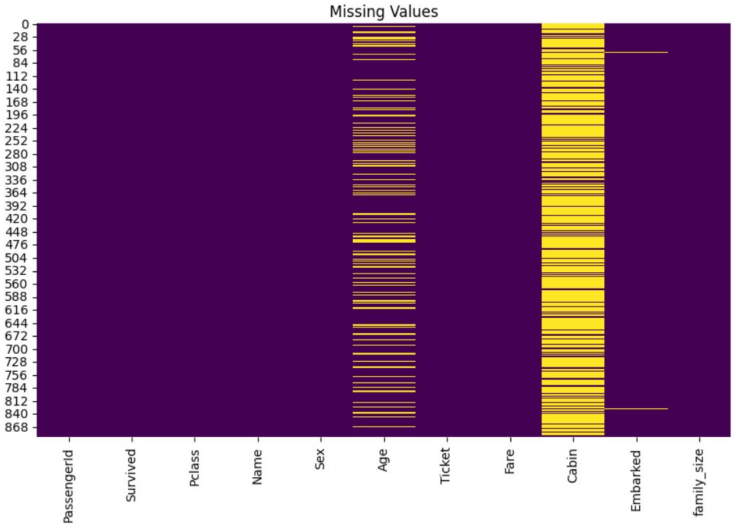
```
# create a new column 'family_size' by combining 'SibSp' and 'Parch'
# they're both family stats, so i thought that i should combine them to simplify the calculations
df['family_size'] = df['SibSp'] + df['Parch']

# thereby droppinig the 'SibSp' and 'Parch' columns
df = df.drop(['SibSp', 'Parch'], axis=1)
```

- Mapped values in ‘Embarked’ and ‘Sex’ column to numerical values
- ‘family_size’ column created by combining ‘SibSp’ and ‘Parch’ for easier analysis and low dimensionality

Checking for missing values

```
# checking for missing values
missing_values = df.isnull().sum()
print("₩nMissing Values:")
print(missing_values)

# visualize missing values
plt.figure(figsize=(10, 6))
sns.heatmap(df.isnull(), cmap='viridis', cbar=False)
plt.title('Missing Values')
plt.show()
```

```
Missing Values:
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
Ticket           0
Fare             0
Cabin          687
Embarked         2
family_size      0
```



Imputing missing values

```
# impute missing values in the 'Age' column with the median age
median_age = df['Age'].median()
df['Age'].fillna(median_age, inplace=True)

# verify that there are no more missing values in the 'Age' column
print("Missing Values in Age Column After Imputation:", df['Age'].isnull().sum(), "₩n")
```

```
Missing Values in Age Column After Imputation: 0
```

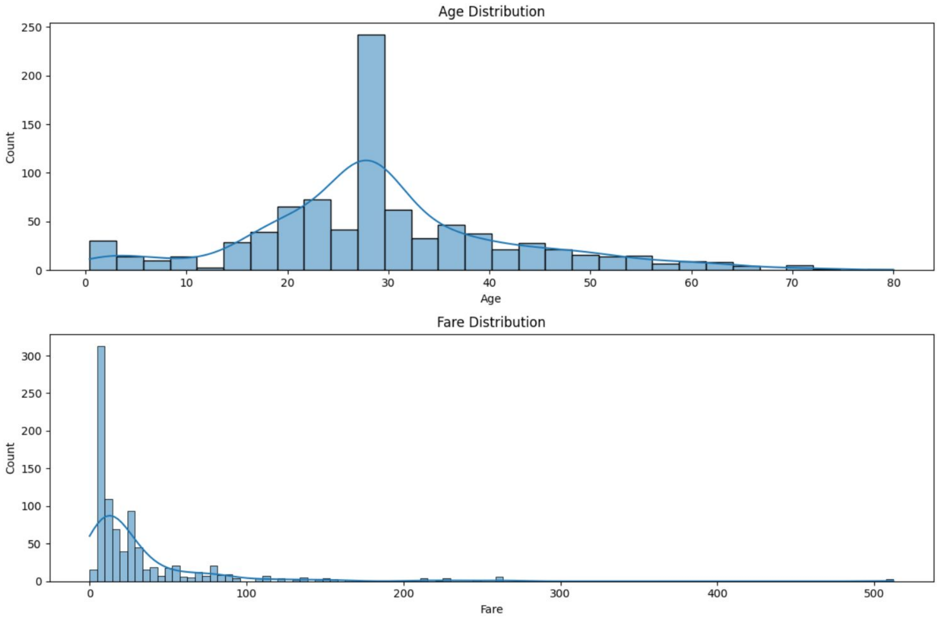
- I didn’t impute any missing values in Cabin because I wasn’t going to use that in my analysis

Outliers

```python
# dealing with outliers
# define function to detect outliers using IQR method for each numerical feature
def detect_outliers(df, features):
    outlier_indices = {}

    for col in features:
        Q1 = np.percentile(df[col], 25)
        Q3 = np.percentile(df[col], 75)
        IQR = Q3 - Q1
        outlier_step = 1.5 * IQR
        outlier_list_col = df[(df[col] < Q1 - outlier_step) | (df[col] > Q3 + outlier_step)].index
        outlier_indices[col] = len(outlier_list_col)

    return outlier_indices

# detect outliers in numerical features
numerical_features = ['Age','Fare']
outliers_counts = detect_outliers(df, numerical_features)

print("Number of outliers for each numerical feature:")
for feature, count in outliers_counts.items():
    print(f"{feature}: {count}")
```

```
Number of outliers for each numerical feature:
Age: 66
Fare: 116
```

```python
plt.figure(figsize=(12, 8))

# plot age
plt.subplot(2, 1, 1)
sns.histplot(df['Age'], kde=True)
plt.title('Age Distribution')

# plot fare
plt.subplot(2, 1, 2)
sns.histplot(df['Fare'], kde=True)
plt.title('Fare Distribution')

plt.tight_layout()
plt.show()
```



Dealing with ‘Fares’ outlier

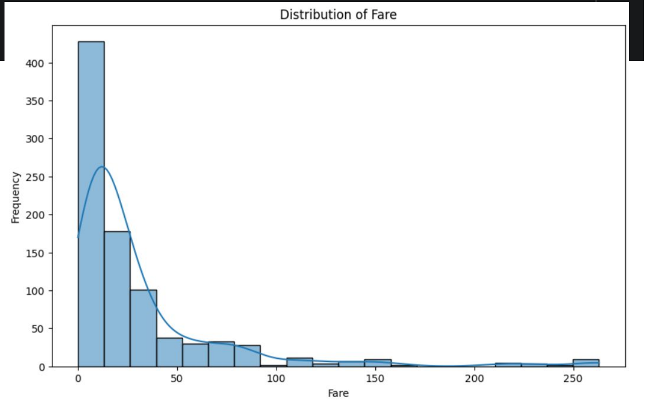
```python
# sort the dataset by 'Fare' column in decreasing order
highest_fares = df.sort_values(by='Fare', ascending=False)

# display the passengers with the highest fares
print("Passengers with the highest fares (in decreasing order):")
print(highest_fares.head(15))
```

```
                                          Name  Sex   Age    Ticket  \
258                           Ward, Miss. Anna    1  35.0  PC 17755
737                      Lesurer, Mr. Gustave J    0  35.0  PC 17755
679         Cardeza, Mr. Thomas Drake Martinez    0  36.0  PC 17755
88                  Fortune, Miss. Mabel Helen    1  23.0     19950
27                Fortune, Mr. Charles Alexander    0  19.0     19950
341               Fortune, Miss. Alice Elizabeth    1  24.0     19950
438                          Fortune, Mr. Mark    0  64.0     19950
311                  Ryerson, Miss. Emily Borie    1  18.0  PC 17608
742        Ryerson, Miss. Susan Parker "Suzette"    1  21.0  PC 17608
118                      Baxter, Mr. Quigg Edmond    0  24.0  PC 17558
299   Baxter, Mrs. James (Helene DeLaudeniere Chaput)    1  50.0  PC 17558
557                        Robbins, Mr. Victor    0  28.0  PC 17757
700  Astor, Mrs. John Jacob (Madeleine Talmadge Force)    1  18.0  PC 17757
380                         Bidois, Miss. Rosalie    1  42.0  PC 17757
716                   Endres, Miss. Caroline Louise    1  38.0  PC 17757

         Fare        Cabin  Embarked  family_size
258  512.3292          NaN       1.0            0
737  512.3292         B101       1.0            0
679  512.3292  B51 B53 B55       1.0            1
88   263.0000  C23 C25 C27       3.0            5
27   263.0000  C23 C25 C27       3.0            5
341  263.0000  C23 C25 C27       3.0            5
438  263.0000  C23 C25 C27       3.0            5
311  262.3750  B57 B59 B63 B66   1.0            4
742  262.3750  B57 B59 B63 B66   1.0            4
118  247.5208      B58 B60       1.0            1
299  247.5208      B58 B60       1.0            1
557  227.5250          NaN       1.0            0
700  227.5250      C62 C64       1.0            1
380  227.5250          NaN       1.0            0
716  227.5250          C45       1.0            0
```

- Among other higher and equally priced tickets, they are typically from the same family or have a clear(er) relation to each other.
- Not only that, the ticket price was so much larger than the closer prices below that I felt very compelled to change/remove these values

Dealing with ‘Fares’ outlier

```python
# set the fare price to 263 for PassengerIds 259, 738, and 680
df.loc[df['PassengerId'].isin([259, 738, 680]), 'Fare'] = 263

# verifying the changes
print("Updated DataFrame with Fare values changed:")
print(df[df['PassengerId'].isin([259, 738, 680])][['PassengerId', 'Name', 'Fare']])

print(df.describe())

# plot histogram of Fare to verify changes
plt.figure(figsize=(10, 6))
sns.histplot(df['Fare'], bins=20, kde=True)
plt.title('Distribution of Fare')
plt.xlabel('Fare')
plt.ylabel('Frequency')
plt.show()
```




```
Updated DataFrame with Fare values changed:
     PassengerId                                Name   Fare
258          259                    Ward, Miss. Anna  263.0
679          680  Cardeza, Mr. Thomas Drake Martinez  263.0
737          738              Lesurer, Mr. Gustave J  263.0
       PassengerId    Survived      Pclass         Sex         Age  \
count   891.000000  891.000000  891.000000  891.000000  891.000000
mean    446.000000    0.383838    2.308642    0.352413   29.361582
std     257.353842    0.486592    0.836071    0.477990   13.019697
min       1.000000    0.000000    1.000000    0.000000    0.420000
25%     223.500000    0.000000    2.000000    0.000000   22.000000
50%     446.000000    0.000000    3.000000    0.000000   28.000000
75%     668.500000    1.000000    3.000000    1.000000   35.000000
max     891.000000    1.000000    3.000000    1.000000   80.000000

             Fare    Embarked  family_size
count  891.000000  889.000000   891.000000
mean    31.364716    2.535433     0.904602
std     43.257927    0.792088     1.613459
min      0.000000    1.000000     0.000000
25%      7.910400    2.000000     0.000000
50%     14.454200    3.000000     0.000000
75%     31.000000    3.000000     1.000000
max    263.000000    3.000000    10.000000
```

Why not change ‘Age’?

After looking at the ‘Age’ distribution, only a handful of passengers were older as the vast majority of them were within their late 20s and early 30s

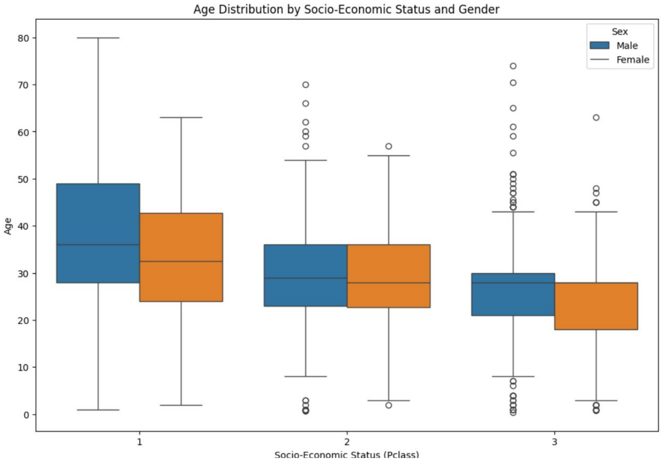
- Data about this will be shown later

In addition, none of the values didn’t seem to be mistakenly inputted. Everything seemed to be accurate.

Analysis of Data

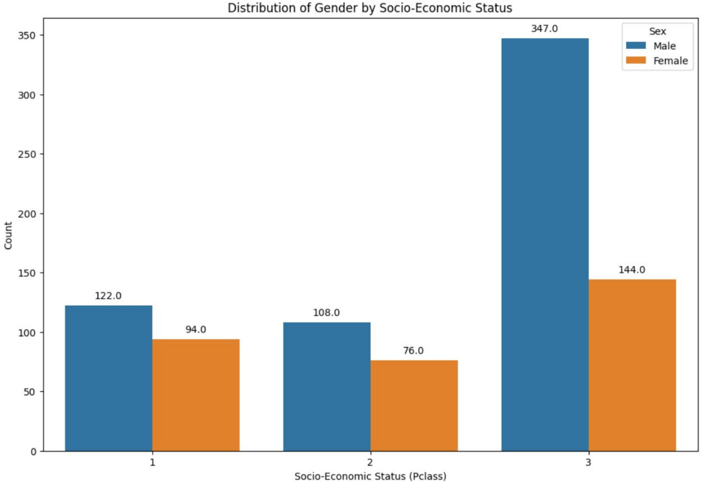
Exploring socio-economic status with varying features

```
# age compared to pclass and gender
plt.figure(figsize=(12, 8))
sns.boxplot(x='Pclass', y='Age', hue='Sex', data=df)
plt.title('Age Distribution by Socio-Economic Status and Gender')
plt.xlabel('Socio-Economic Status (Pclass)')
plt.ylabel('Age')
plt.legend(title='Sex', labels=['Male', 'Female'])
plt.show()

# gender compared to pclass
plt.figure(figsize=(12, 8))
ax = sns.countplot(x='Pclass', hue='Sex', data=df)
```



Notable features:

- 1st class passengers are typically older men, 3rd class younger
  - Male dominated society
- 3rd class is VASTLY dominated by male population
  - 1st + 2nd class combined is only 53 more passengers

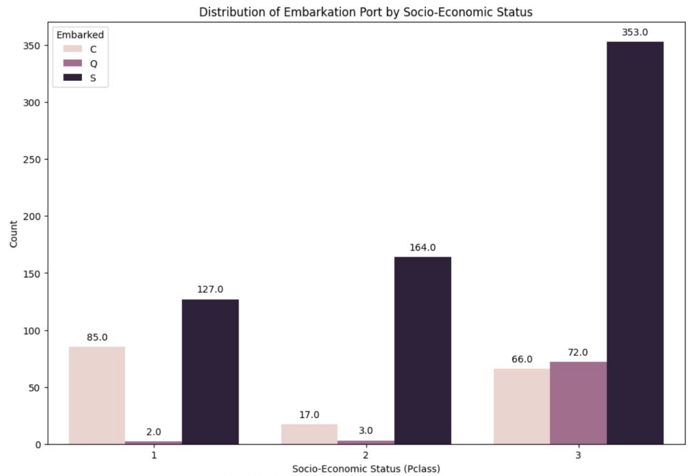


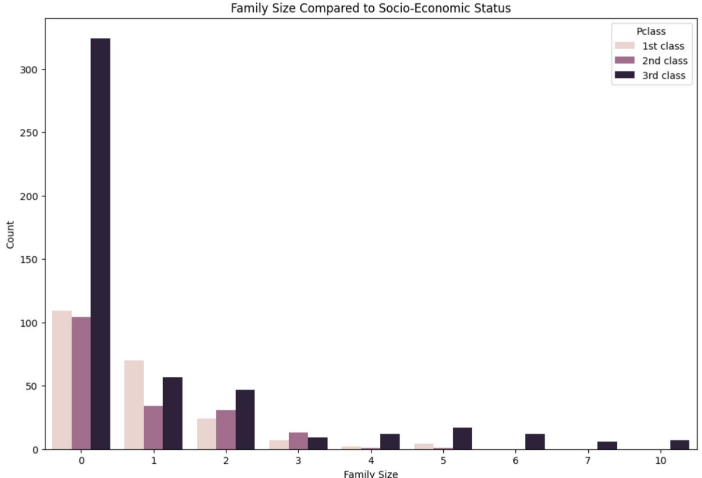
Exploring socio-economic status with varying features

```
# embarkation location compared to pclass
plt.figure(figsize=(12, 8))
ax = sns.countplot(x='Pclass', hue='Embarked', data=df)
```

```
# family size compared to pclass
plt.figure(figsize=(12, 8))
sns.countplot(x='family_size', hue='Pclass', data=df)
plt.title('Family Size Compared to Socio-Economic Status')
plt.xlabel('Family Size')
plt.ylabel('Count')
plt.legend(title='Pclass', loc='upper right', labels=['1st class', '2nd class', '3rd class'])
plt.show()
```

Key: C = Cherbourg = 1, Q = Queenstown = 2, S = Southampton = 3



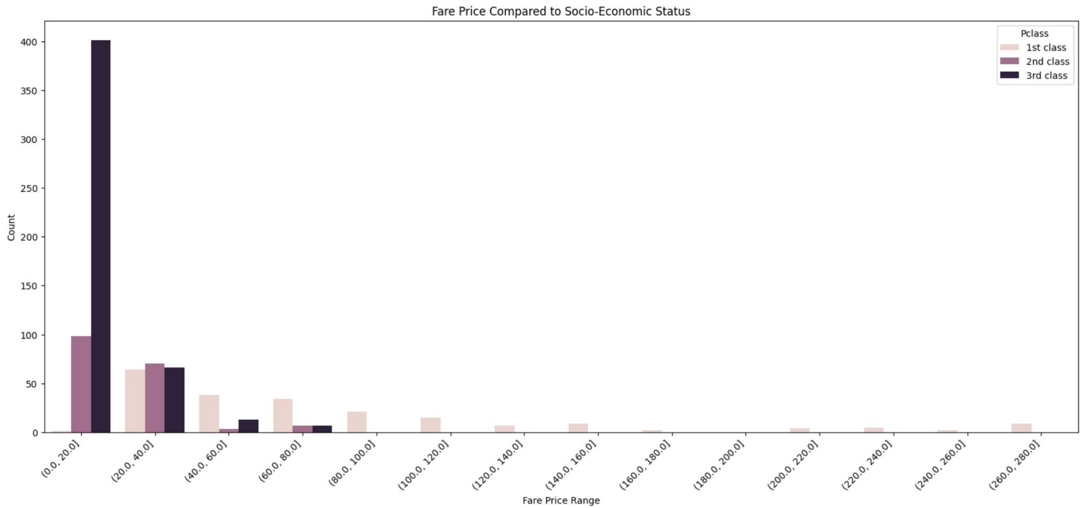


Notable features:

- 1st class mostly came from Cherbourg and Southampton
- Queenstown mostly consists of 3rd class
- Southampton is by far the most populated town that had Titanic passengers
- Most passengers had 0 family along
- 3rd class passengers had the most amount of family along

Exploring socio-economic status with varying features

```
# define fare price bins
fare_bins = np.arange(0, df['Fare'].max() + 20, 20)

# create a new column 'fare_range' to categorize fare prices into bins
df['fare_range'] = pd.cut(df['Fare'], bins=fare_bins)

# plot fare price compared to Pclass using fare ranges
plt.figure(figsize=(20, 8))
sns.countplot(x='fare_range', hue='Pclass', data=df)
plt.title('Fare Price Compared to Socio-Economic Status')
plt.xlabel('Fare Price Range')
plt.ylabel('Count')
plt.xticks(rotation=45, ha='right')
plt.legend(title='Pclass', loc='upper right', labels=['1st class', '2nd class', '3rd class'])
plt.show()
```



Notable features:

- 1st class passengers had the most expensive tickets, 3rd class passengers (makes sense)
- 2nd class and 3rd class passengers had similarly priced tickets.
- Tickets were expensive all around: 3rd class today would be over $1000

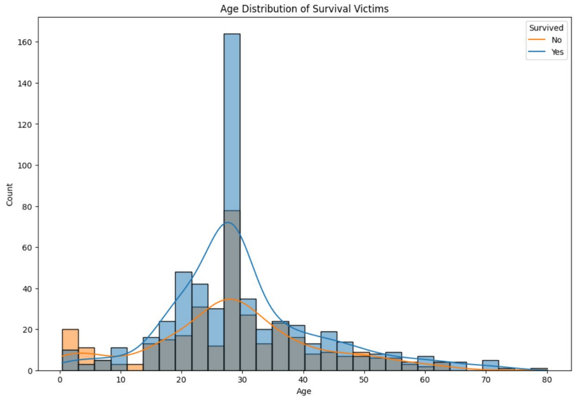
Class	Price in 1912 (£)	Price in 1912 ($)	Price today (£)	Price today ($)
First Class Suite	£870	$4,350	£105,883	$133,132
First Class Berth	£30	$150	£3,651	$4,591
Second Class	£12	$60	£1,460	$1,834
Third Class	£7	$35	£852	$1,071

https://www.cruisemummy.co.uk/titanic-ticket-prices/

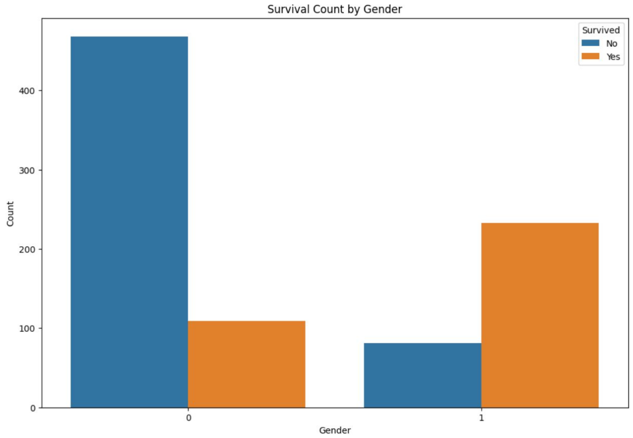
Exploring distribution of survival victims

```
# survival count compared to age
plt.figure(figsize=(12, 8))
sns.histplot(x='Age', hue='Survived', data=df, kde=True)
plt.title('Age Distribution of Survival Victims')
plt.xlabel('Age')
plt.ylabel('Count')
plt.legend(title='Survived', labels=['No', 'Yes'])
plt.show()

# survival count compared to Gender
plt.figure(figsize=(12, 8))
sns.countplot(x='Sex', hue='Survived', data=df)
plt.title('Survival Count by Gender')
plt.xlabel('Gender')
plt.ylabel('Count')
plt.legend(title='Survived', labels=['No', 'Yes'])
plt.show()

# survival count compared to family_size
plt.figure(figsize=(12, 8))
sns.countplot(x='family_size', hue='Survived', data=df)
plt.title('Survival Count by Family Size')
plt.xlabel('Family Size')
plt.ylabel('Count')
plt.legend(title='Survived', loc='upper right', labels=['No', 'Yes'])
plt.show()
```

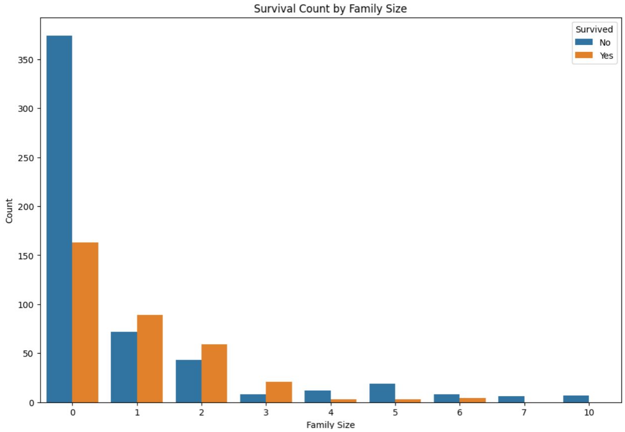
1



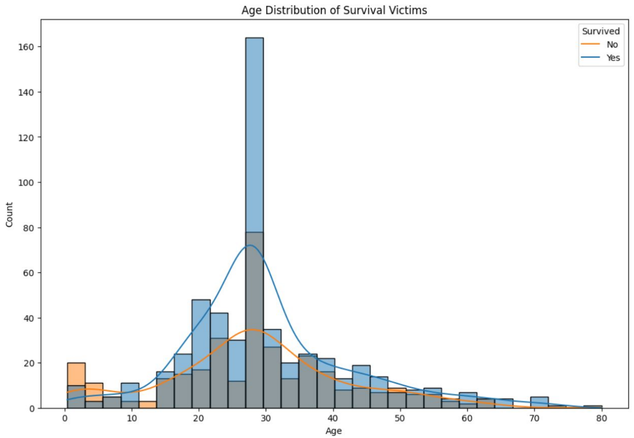
2



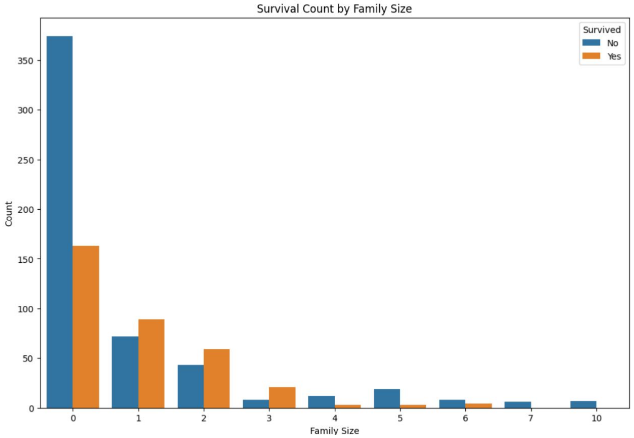
3



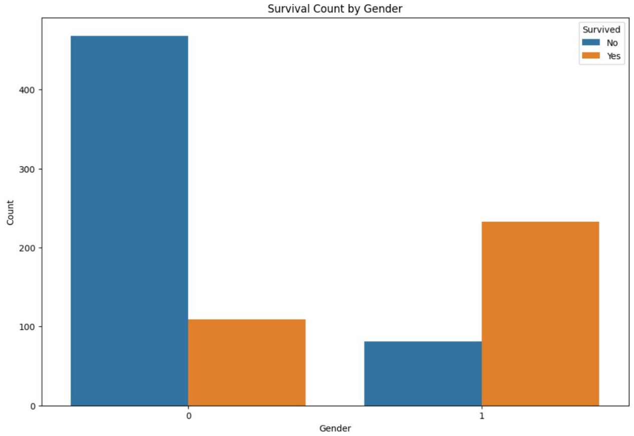
Exploring distribution of survival victims



Most surviving and dying passengers were in late 20s



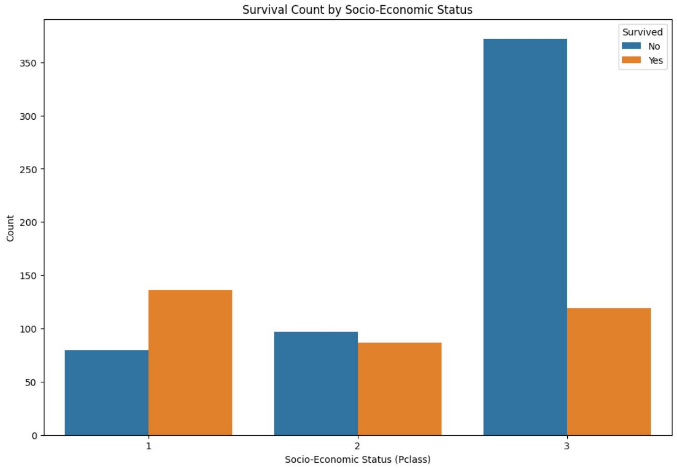
Passengers with 1-3 family members survived more often, hard to find correlation



“Save women and children first!”

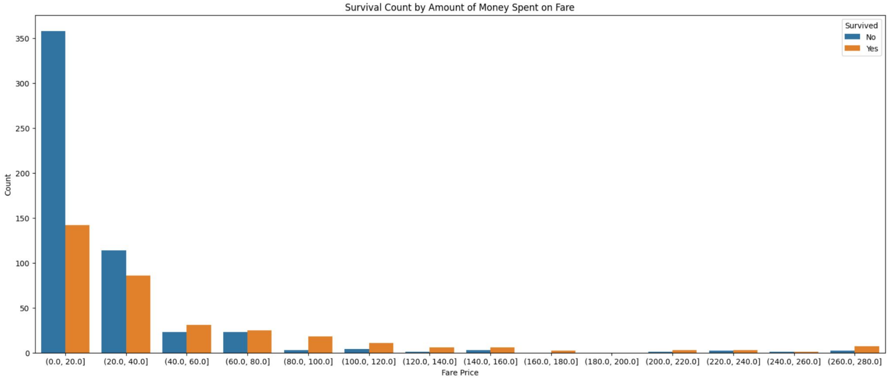
Exploring distribution of survival victims

```
# survival count compared to pclass
plt.figure(figsize=(12, 8))
sns.countplot(x='Pclass', hue='Survived', data=df)
plt.title('Survival Count by Socio-Economic Status')
plt.xlabel('Socio-Economic Status (Pclass)')
plt.ylabel('Count')
plt.legend(title='Survived', labels=['No', 'Yes'])
plt.show()

# survival count compared to fare price spent using the same range used above
plt.figure(figsize=(20, 8))
sns.countplot(x='fare_range', hue='Survived', data=df)
plt.title('Survival Count by Amount of Money Spent on Fare')
plt.xlabel('Fare Price')
plt.ylabel('Count')
plt.legend(title='Survived', labels=['No', 'Yes'])
plt.show()
```



Notable features:

- 1st class passengers survived more often compared to other classes
- They seemed to have more priority as they were probably higher leading citizens
- Unfortunately, most of the passengers who passed away were of the 3rd class

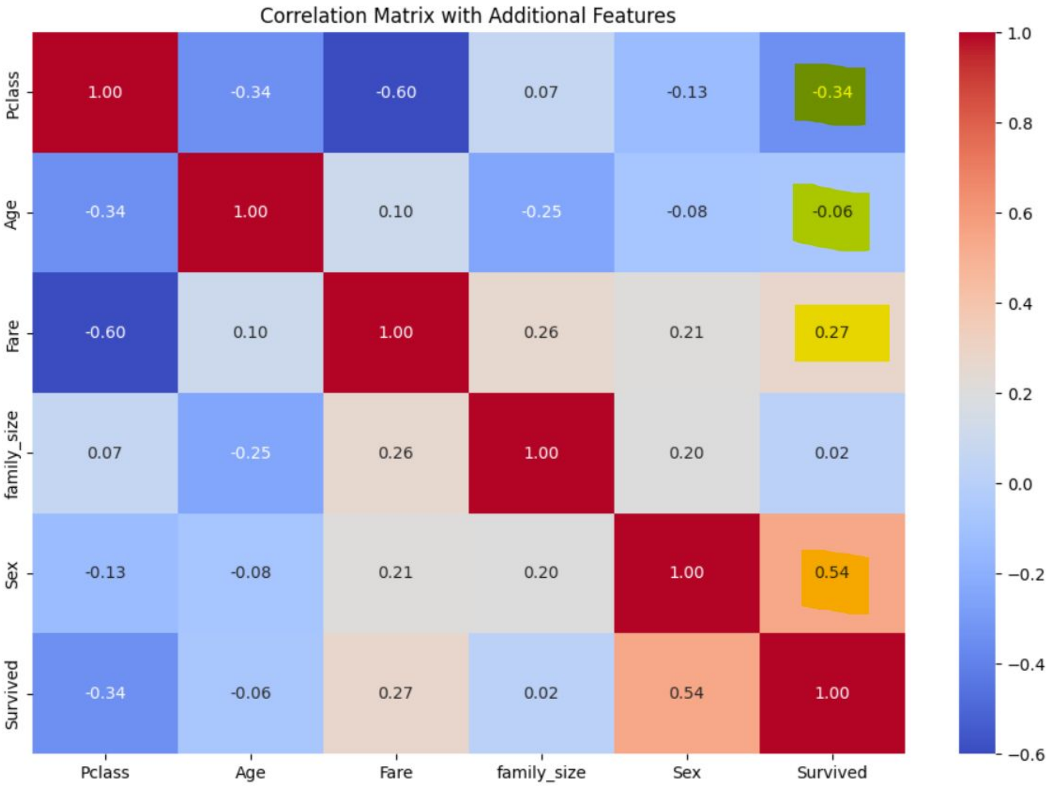


Correlation Analysis

```python
# important features added to the correlation matrix
additional_features = ['Pclass','Age', 'Fare', 'family_size', 'Sex', 'Survived']
correlation_matrix_with_additional = df[additional_features].corr()

# plot the correlation matrix as a heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix_with_additional, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix with Additional Features')
plt.show()
```

Notable features:

- Pclass, Age, Fare, Sex
- Keeping Age so that the data remains complex otherwise the algorithms output the same data



Modeling

Preprocess Data

```
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.model_selection import train_test_split, cross_val_score, cross_validate, cross_val_predict

# tdf is test_df for test.csv
tdf = pd.read_csv("test.csv")
```

```
# preprocess the data

# mapping string data to numerical data
sex_mapping = {'male': 0, 'female': 1}
tdf['Sex'] = tdf['Sex'].map(sex_mapping)

embarked_mapping = {'C': 1, 'Q': 2, 'S': 3}
tdf['Embarked'] = tdf['Embarked'].map(embarked_mapping)

# inputting missing or NaN data
tdf['Age'].fillna(tdf['Age'].median(), inplace=True)
tdf['Fare'].fillna(tdf['Fare'].median(), inplace=True)


# combining "family traits" to a single column
tdf['family_size'] = tdf['SibSp'] + tdf['Parch']
tdf = tdf.drop(['SibSp', 'Parch'], axis=1)
```

Mapping and Imputations are the same process as with the training data

Choosing a Model

Chose the ones we learned/briefly mention in class:

- Logistic Regression: binary classification algorithm
  - e.g. predict if email is spam or not, if patient has disease or not
- Linear Regression: supervised learning algorithm for regression
  - finds “best-fitting” linear relationship between input features and target by iteratively minimizing the sum of the squared differences between the observed and predicted values
  - gets a “precise” value, so added a rounding feature.

```
# load, train, and predict with Linear Regression model
# Linear Regression predicts with more specific values to their decimal place. I didn't want that, so I decided
# to make it so that if your "survival" is more than 0.5, you survived.
linear_model = LinearRegression()
linear_model.fit(X_train, y_train)
linear_predictions = linear_model.predict(X_test)
linear_predictions = [1 if x > 0.5 else 0 for x in linear_predictions]
```

- K Neighbors Classification: supervised learning algorithm for classification
  - doesn’t learn model → memorizes training instances and makes predictions based on the similarity of new instances to the training data
  - on each new data point, identifies the k-nearest neighbors (most similar) from the training set
- Random Forest Classification: supervised learning method for classification and regression
  - creates multiple trees and merges their prediction to improve accuracy and reduce overfitting

Choosing a Model

```python
# define key features and target variable
features = ['Pclass', 'Sex', 'Age', 'family_size', 'Fare']
X_train = df[features]
y_train = df['Survived']

X_test = tdf[features]

# load, train, and predict with Logistic Regression model
logistic_model = LogisticRegression(max_iter=1000)
logistic_model.fit(X_train, y_train)
logistic_predictions = logistic_model.predict(X_test)

# load, train, and predict with Linear Regression model
# Linear Regression predicts with more specific values to their decimal place. I didn't want that, so I decided
# to make it so that if your "survival" is more than 0.5, you survived.
linear_model = LinearRegression()
linear_model.fit(X_train, y_train)
linear_predictions = linear_model.predict(X_test)
linear_predictions = [1 if x > 0.5 else 0 for x in linear_predictions]

# load, train, and predict with K Neighbors Classifier model
knn_model = KNeighborsClassifier()
knn_model.fit(X_train, y_train)
knn_predictions = knn_model.predict(X_test)

# load, train, and predict with Random Forest Classifier model
rf_model = RandomForestClassifier()
rf_model.fit(X_train, y_train)
rf_predictions = rf_model.predict(X_test)

# printing the predictions for visibility
print("Logistic Regression predictions:", logistic_predictions)
print("Linear Regression predictions:", linear_predictions)
print("KNeighbors Classification predictions:", knn_predictions)
print("Random Forest Classification predictions:", rf_predictions)
```

```python
true_data = pd.read_csv("idkiftrue_submission.csv")
true_survival = true_data['Survived']

# evaluating models
def evaluate_model(predictions, true_survival, model_name):
    print(f"Metrics for {model_name}:")
    print("Accuracy:", accuracy_score(true_survival, predictions))
    print("Precision:", precision_score(true_survival, predictions))
    print("Recall:", recall_score(true_survival, predictions))
    print("F1 Score:", f1_score(true_survival, predictions), "₩n")

print("Evaluation on True Data:")
evaluate_model(logistic_predictions, true_survival, "Logistic Regression")
evaluate_model(linear_predictions, true_survival, "Linear Regression")
evaluate_model(knn_predictions, true_survival, "K Neighbors Classifier")
evaluate_model(rf_predictions, true_survival, "Random Forest Classifier")
```

```
Evaluation on True Data:
Metrics for Logistic Regression:
Accuracy: 0.9354066985645934
Precision: 0.8805031446540881
Recall: 0.9459459459459459
F1 Score: 0.9120521172638437

Metrics for Linear Regression:
Accuracy: 0.9665071770334929
Precision: 0.9466666666666667
Recall: 0.9594594594594594
F1 Score: 0.9530201342281879

Metrics for K Neighbors Classifier:
Accuracy: 0.6483253588516746
Precision: 0.5029940119760479
Recall: 0.5675675675675675
F1 Score: 0.5333333333333332

Metrics for Random Forest Classifier:
Accuracy: 0.8325358851674641
Precision: 0.756578947368421
Recall: 0.777027027027027
F1 Score: 0.7666666666666666
```

Analysis of scores

```
Evaluation on True Data:
Metrics for Logistic Regression:
Accuracy: 0.9354066985645934
Precision: 0.8805031446540881
Recall: 0.9459459459459459
F1 Score: 0.9120521172638437

Metrics for Linear Regression:
Accuracy: 0.9665071770334929
Precision: 0.9466666666666667
Recall: 0.9594594594594594
F1 Score: 0.9530201342281879

Metrics for K Neighbors Classifier:
Accuracy: 0.6483253588516746
Precision: 0.5029940119760479
Recall: 0.5675675675675675
F1 Score: 0.5333333333333332

Metrics for Random Forest Classifier:
Accuracy: 0.8325358851674641
Precision: 0.756578947368421
Recall: 0.777027027027027
F1 Score: 0.7666666666666666
```

Logistic and Linear Regression have by far the largest scores.

KNN is much lower. Here's a couple reasons why:

- Number of features was too high– it complicated the algorithm
- Could lead to <u>overfitting</u>
- With more dimensions, defining a set decision boundary becomes more challenging

Random Forest is slightly lower:

- Possibly, data is not properly normalized → overfitting

HOW TO IMPROVE?

Might be beneficial to use less features

Less number of features → Better results?

```python
# define NEW key features and target variable
features = ['Pclass', 'Sex', 'Fare', 'Age']
X_train = df[features]
y_train = df['Survived']

X_test = tdf[features]

# load, train, and predict with Logistic Regression model
logistic_model = LogisticRegression(max_iter=1000)
logistic_model.fit(X_train, y_train)
logistic_predictions = logistic_model.predict(X_test)

# load, train, and predict with Linear Regression model
linear_model = LinearRegression()
linear_model.fit(X_train, y_train)
linear_predictions = linear_model.predict(X_test)
linear_predictions = [1 if x > 0.5 else 0 for x in linear_predictions]

# load, train, and predict with K Neighbors Classifier model
knn_model = KNeighborsClassifier()
knn_model.fit(X_train, y_train)
knn_predictions = knn_model.predict(X_test)

# load, train, and predict with Random Forest Classifier model
rf_model = RandomForestClassifier()
rf_model.fit(X_train, y_train)
rf_predictions = rf_model.predict(X_test)

# printing the predictions for visibility
print("Logistic Regression predictions:", logistic_predictions)
print("Linear Regression predictions:", linear_predictions)
print("KNeighbors Classification predictions:", knn_predictions)
print("Random Forest Classification predictions:", rf_predictions)
```

```python
print("Evaluation on True Data:")
evaluate_model(logistic_predictions, true_survival, "Logistic Regression")
evaluate_model(linear_predictions, true_survival, "Linear Regression")
evaluate_model(knn_predictions, true_survival, "K Neighbors Classifier")
evaluate_model(rf_predictions, true_survival, "Random Forest Classifier")
```

```
Evaluation on True Data:
Metrics for Logistic Regression:
Accuracy: 0.9234449760765551
Precision: 0.8625
Recall: 0.9324324324324325
F1 Score: 0.8961038961038962

Metrics for Linear Regression:
Accuracy: 0.9641148325358851
Precision: 0.934640522875817
Recall: 0.9662162162162162
F1 Score: 0.9501661129568106

Metrics for K Neighbors Classifier:
Accuracy: 0.638755980861244
Precision: 0.49122807017543857
Recall: 0.5675675675675675
F1 Score: 0.5266457680250783

Metrics for Random Forest Classifier:
Accuracy: 0.8277511961722488
Precision: 0.740506329113924
Recall: 0.7905405405405406
F1 Score: 0.7647058823529411
```

Before vs. After

```
features = ['Pclass', 'Sex', 'Age', 'family_size', 'Fare']
```

```
Evaluation on True Data:
Metrics for Logistic Regression:
Accuracy: 0.9354066985645934
Precision: 0.8805031446540881
Recall: 0.9459459459459459
F1 Score: 0.9120521172638437

Metrics for Linear Regression:
Accuracy: 0.9665071770334929
Precision: 0.9466666666666667
Recall: 0.9594594594594594
F1 Score: 0.9530201342281879

Metrics for K Neighbors Classifier:
Accuracy: 0.6483253588516746
Precision: 0.5029940119760479
Recall: 0.5675675675675675
F1 Score: 0.5333333333333332

Metrics for Random Forest Classifier:
Accuracy: 0.8325358851674641
Precision: 0.756578947368421
Recall: 0.777027027027027
F1 Score: 0.7666666666666666
```

```
features = ['Pclass', 'Sex', 'Fare', 'Age']
```

```
Evaluation on True Data:
Metrics for Logistic Regression:
Accuracy: 0.9234449760765551
Precision: 0.8625
Recall: 0.9324324324324325
F1 Score: 0.8961038961038962

Metrics for Linear Regression:
Accuracy: 0.9641148325358851
Precision: 0.934640522875817
Recall: 0.9662162162162162
F1 Score: 0.9501661129568106

Metrics for K Neighbors Classifier:
Accuracy: 0.638755980861244
Precision: 0.49122807017543857
Recall: 0.5675675675675675
F1 Score: 0.5266457680250783

Metrics for Random Forest Classifier:
Accuracy: 0.8277511961722488
Precision: 0.740506329113924
Recall: 0.7905405405405406
F1 Score: 0.7647058823529411
```

Cross Validation

Cross Validation Set

```
# redefine features
features = ['Pclass', 'Sex', 'Age', 'family_size', 'Fare']

# split the original data into training and testing sets (60% training, 40% testing)
X_train, X_test, y_train, y_test = train_test_split(df[features], df['Survived'], test_size=0.4, random_state=42)

# logistic regression
logistic_model.fit(X_train, y_train)
# eval
logistic_regression_predictions = logistic_model.predict(X_test)
accuracy_log = accuracy_score(y_test, logistic_regression_predictions)
precision_log = precision_score(y_test, logistic_regression_predictions)
recall_log = recall_score(y_test, logistic_regression_predictions)
f1_log = f1_score(y_test, logistic_regression_predictions)

# linear regression
linear_model.fit(X_train, y_train)
# eval
linear_regression_predictions = linear_model.predict(X_test)
accuracy_lr = accuracy_score(y_test, linear_regression_predictions.round())
precision_lr = precision_score(y_test, linear_regression_predictions.round())
recall_lr = recall_score(y_test, linear_regression_predictions.round())
f1_lr = f1_score(y_test, linear_regression_predictions.round())

# knn
knn_model.fit(X_train, y_train)
# eval
knn_predictions = knn_model.predict(X_test)
accuracy_knn = accuracy_score(y_test, knn_predictions)
precision_knn = precision_score(y_test, knn_predictions)
recall_knn = recall_score(y_test, knn_predictions)
f1_knn = f1_score(y_test, knn_predictions)

# random forest
rf_model.fit(X_train, y_train)
# eval
rf_predictions = rf_model.predict(X_test)
accuracy_rf = accuracy_score(y_test, rf_predictions)
precision_rf = precision_score(y_test, rf_predictions)
recall_rf = recall_score(y_test, rf_predictions)
f1_rf = f1_score(y_test, rf_predictions)
```

```
print("Logistic Regression - Accuracy:", accuracy_log)
print("Logistic Regression - Precision:", precision_log)
print("Logistic Regression - Recall:", recall_log)
print("Logistic Regression - F1 Score:", f1_log)

print("Linear Regression - Accuracy:", accuracy_lr)
print("Linear Regression - Precision:", precision_lr)
print("Linear Regression - Recall:", recall_lr)
print("Linear Regression - F1 Score:", f1_lr)

print("KNN - Accuracy:", accuracy_knn)
print("KNN - Precision:", precision_knn)
print("KNN - Recall:", recall_knn)
print("KNN - F1 Score:", f1_knn)

print("Random Forest - Accuracy:", accuracy_rf)
print("Random Forest - Precision:", precision_rf)
print("Random Forest - Recall:", recall_rf)
print("Random Forest - F1 Score:", f1_rf)
```

```
Logistic Regression - Accuracy: 0.7927170868347339
Logistic Regression - Precision: 0.7815126050420168
Logistic Regression - Recall: 0.6595744680851063
Logistic Regression - F1 Score: 0.7153846153846154
Linear Regression - Accuracy: 0.7759103641456583
Linear Regression - Precision: 0.7520661157024794
Linear Regression - Recall: 0.6453900709219859
Linear Regression - F1 Score: 0.6946564885496184
KNN - Accuracy: 0.7002801120448179
KNN - Precision: 0.6370967741935484
KNN - Recall: 0.5602836879432624
KNN - F1 Score: 0.5962264150943396
Random Forest - Accuracy: 0.7927170868347339
Random Forest - Precision: 0.7557251908396947
Random Forest - Recall: 0.7021276595744681
Random Forest - F1 Score: 0.7279411764705883
```

Comparing with and without Cross Validation

```
Evaluation on True Data:
Metrics for Logistic Regression:
Accuracy: 0.9354066985645934
Precision: 0.8805031446540881
Recall: 0.9459459459459459
F1 Score: 0.9120521172638437

Metrics for Linear Regression:
Accuracy: 0.9665071770334929
Precision: 0.9466666666666667
Recall: 0.9594594594594594
F1 Score: 0.9530201342281879

Metrics for K Neighbors Classifier:
Accuracy: 0.6483253588516746
Precision: 0.5029940119760479
Recall: 0.5675675675675675
F1 Score: 0.5333333333333332

Metrics for Random Forest Classifier:
Accuracy: 0.8349282296650717
Precision: 0.7548387096774194
Recall: 0.7905405405405406
F1 Score: 0.7722772277227724
```

```
Logistic Regression - Accuracy: 0.7927170868347339
Logistic Regression - Precision: 0.7815126050420168
Logistic Regression - Recall: 0.6595744680851063
Logistic Regression - F1 Score: 0.7153846153846154
Linear Regression - Accuracy: 0.7759103641456583
Linear Regression - Precision: 0.7520661157024794
Linear Regression - Recall: 0.6453900709219859
Linear Regression - F1 Score: 0.6946564885496184
KNN - Accuracy: 0.7002801120448179
KNN - Precision: 0.6370967741935484
KNN - Recall: 0.5602836879432624
KNN - F1 Score: 0.5962264150943396
Random Forest - Accuracy: 0.7927170868347339
Random Forest - Precision: 0.7557251908396947
Random Forest - Recall: 0.7021276595744681
Random Forest - F1 Score: 0.7279411764705883
```

Should be higher, I believe the work was implemented incorrectly.