Titanic - Who will survive?

By: Terry Shvartsman (114311609)

Original Data

```
test = pd.read_csv(r'OneDrive\Desktop\test.csv')
train = pd.read_csv(r'OneDrive\Desktop\train.csv')
```

Test

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	892	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	NaN	Q
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	S
2	894	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	NaN	Q
3	895	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	NaN	S
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875	NaN	S
...	...	...	...	...	...	...	...	...	...	...	...
413	1305	3	Spector, Mr. Woolf	male	NaN	0	0	A.5. 3236	8.0500	NaN	S
414	1306	1	Oliva y Ocana, Dona. Fermina	female	39.0	0	0	PC 17758	108.9000	C105	C
415	1307	3	Saether, Mr. Simon Sivertsen	male	38.5	0	0	SOTON/O.Q. 3101262	7.2500	NaN	S
416	1308	3	Ware, Mr. Frederick	male	NaN	0	0	359309	8.0500	NaN	S
417	1309	3	Peter, Master. Michael J	male	NaN	1	1	2668	22.3583	NaN	C

418 rows × 11 columns

Train

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
...	...	...	...	...	...	...	...	...	...	...	...	...
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	NaN	S
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	B42	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	NaN	S
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	C148	C
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500	NaN	Q

891 rows × 12 columns

Analyzing the Data (EDA)

Preparing to Clean The Data

Using pandas functions, like the .describe() and .info() function, I was able to get an understanding of the spread of the data of each column. Determining the mean, median, q1, q3, min, and max is informative when considering factors like outliers and data distribution. In addition, the .info() function tells me how many non-null values, and hence how many null-values, each column has. This was useful in understanding which columns may need to be dropped, as well as which columns I needed to exercise caution with while mutating.

```
train.describe()
```

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

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object
 11  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

Cleaning the Data

Using the functions described before, I was able to use my understanding of the data to determine what data to remove. First, I determined that there were approximately 20 outliers in the “Fare” column with a fare of more than or equal to 200. However if I were to remove these rows, it would remove 18 rows, nearly 5%, of the rows in the test dataset. As such, using the .query() function, I removed the outliers than had a fare more than or equal to 400, which removed a minimal number of rows from the test data-set while also removing some odd rows from the training set. Furthermore, I determined that the columns “Passenger Id”, “Name”, “Ticket” (ticket number), were unnecessary columns and dropped them as well.

```
np.where(train.Fare >= 200, 1, 0).sum()
```

20

```
np.where(test.Fare >= 200, 1, 0).sum()
```

18

```
np.where(train.Fare >= 400, 1, 0).sum()
```

3

```
np.where(test.Fare >= 400, 1, 0).sum()
```

1

```
#Removing outliers and removing unnecessary columns (cleaning the data)
test = test.query("Fare < 400").drop(columns=['PassengerId', 'Name', 'Ticket'])
train = train.query("Fare < 400").drop(columns=['PassengerId', 'Name', 'Ticket'])
```

Data After Cleaning

Train

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Cabin	Embarked
0	0	3	male	22.0	1	0	7.2500	NaN	S
1	1	1	female	38.0	1	0	71.2833	C85	C
2	1	3	female	26.0	0	0	7.9250	NaN	S
3	1	1	female	35.0	1	0	53.1000	C123	S
4	0	3	male	35.0	0	0	8.0500	NaN	S
...	...	...	...	...	...	...	...	...	...
886	0	2	male	27.0	0	0	13.0000	NaN	S
887	1	1	female	19.0	0	0	30.0000	B42	S
888	0	3	female	NaN	1	2	23.4500	NaN	S
889	1	1	male	26.0	0	0	30.0000	C148	C
890	0	3	male	32.0	0	0	7.7500	NaN	Q

888 rows × 9 columns

Test

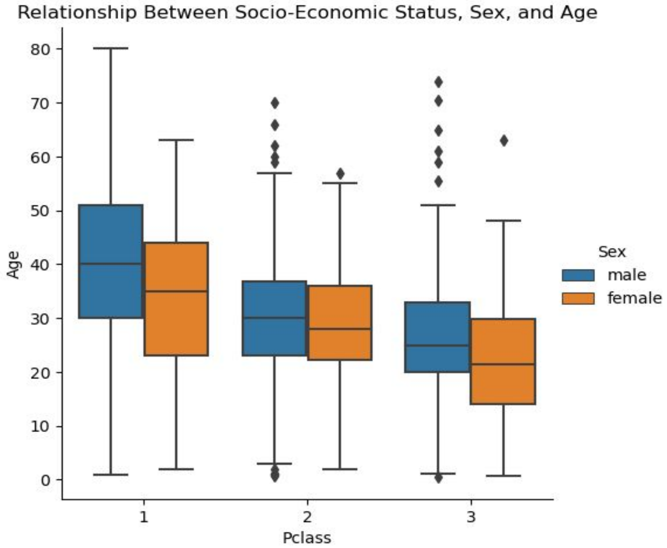
	Pclass	Sex	Age	SibSp	Parch	Fare	Cabin	Embarked
0	3	male	34.5	0	0	7.8292	NaN	Q
1	3	female	47.0	1	0	7.0000	NaN	S
2	2	male	62.0	0	0	9.6875	NaN	Q
3	3	male	27.0	0	0	8.6625	NaN	S
4	3	female	22.0	1	1	12.2875	NaN	S
...	...	...	...	...	...	...	...	...
413	3	male	NaN	0	0	8.0500	NaN	S
414	1	female	39.0	0	0	108.9000	C105	C
415	3	male	38.5	0	0	7.2500	NaN	S
416	3	male	NaN	0	0	8.0500	NaN	S
417	3	male	NaN	1	1	22.3583	NaN	C

416 rows × 8 columns

Analyzing Relationship between Socioeconomic Status and Other Features

There are a number of interesting observations that can be made through modeling the relationships between the features. It can be noticed that the median age decreases, for both men and women, as the “Pclass” increases. In other words, the higher the class, the higher the median age of the people. Additionally, as the catplot demonstrates, there are age outliers in Pclass 2 and 3, demonstrating a larger spread of ages in Pclass 1 compared to Pclass 2 and 3. This means that there were more very old and very young people in Pclass 1 than in 2 and 3. Furthermore, the catplot, along with the “Pclass vs Sex” table, demonstrates that there were more men than women in every Pclass, as well as that there is a massive jump in the difference between the number of males and females in the lowest Pclass compared to Pclass 1 and 2.

```
sns.catplot(data=train, x="Pclass", y="Age", kind="box", hue="Sex")
plt.show()
```




```
train.groupby(['Pclass','Sex']).size()

Pclass  Sex
1       female     93
        male      120
2       female     76
        male      108
3       female    144
        male      347
dtype: int64
```

Analyzing Relationship between Socioeconomic Status and Other Features Continued

Finally, using the groupby function, I created a table that compared the average fare to the number of family members in each Pclass. Interestingly, Pclass 2 and 3 had no gaps in the number of family members, while Pclass 1 had no cases of 3 family members. Additionally, it can be noticed that Pclass 1 or 2 came with only three of four family members, while Pclass 3 had up to 6. Also, although somewhat obvious, I decided to compare the average fare by Pclass. As expected, the higher the class the higher the fare. The interesting thing, however, is how large of a difference the fare of Pclass 1 was from Pclass 2 and 3.

```
train.groupby(['Pclass', 'Parch'])['Fare'].mean()
```

```
Pclass  Parch
1       0         68.639980
        1        115.125135
        2        150.343648
        4        263.000000
2       0         17.467132
        1         27.609506
        2         33.499488
        3         20.875000
3       0         10.023412
        1         19.408033
        2         33.809300
        3         29.336100
        4         25.625000
        5         32.550000
        6         46.900000
```

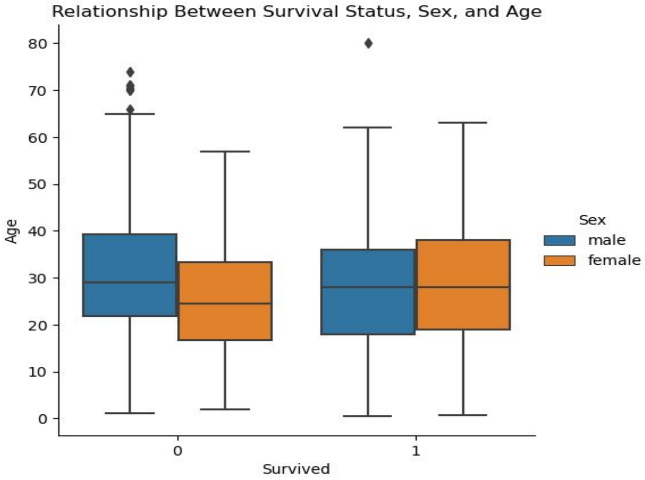
```
train.groupby(['Pclass'])['Fare'].mean()
```

```
Pclass
1    78.124061
2    20.662183
3    13.675550
Name: Fare, dtype: float64
```

Analyzing Relationship between Survival Status and Other Features

There are a number of interesting observations between the survival status of passengers and their other features. Firstly, as demonstrated by the catplot and table, the vast majority of survivors were women, which makes sense since women and children were put onto the lifeboats first. However, it is interesting to consider that the ratio of women surviving out of total women is 233/314, which is almost 75%, while only 109/577, not even 20%, of men survived. Additionally, it can be noticed that the average fare paid by the survivors was significantly higher than those who died, even though the number of survivors was significantly less than the number of people that died. This demonstrates how the majority of survivors were from Pclass 1. As for age, it seems that the ages of both the survivors and the dead passengers was relatively distributed. However, as there are outliers in the males that died portion of the catplot, it can be deduced that more old people died than young people. This, again, makes sense since they put women and children on the lifeboats before adult males.

```
sns.catplot(data=train, x="Survived", y="Age", kind="box", hue="Sex")
plt.show()
```




```
train.groupby(['Survived'])['Fare'].mean()
```

```
Survived
0    22.117887
1    48.395408
Name: Fare, dtype: float64
```

```
train.groupby(['Survived','Sex']).size()
```

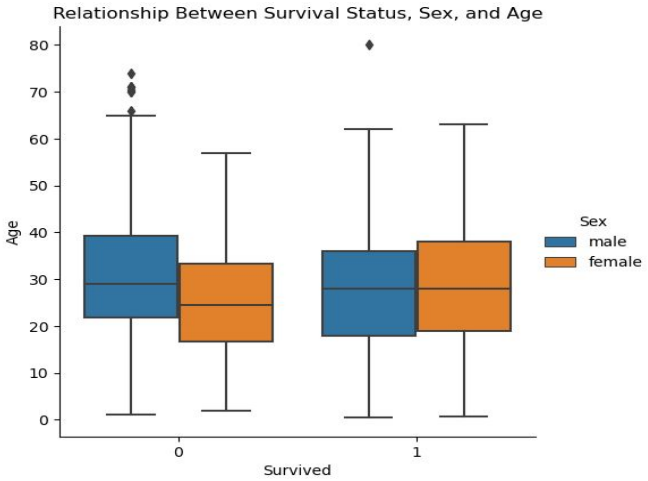
```
Survived  Sex
0         female     81
          male      468
1         female    233
          male      109
dtype: int64
```

Correlation and Importance Analysis

Before doing any analysis, I hypothesized that the age, sex, and pclass will be the most important columns, and hence have the largest correlation (by magnitude). Unfortunately, as Sex is a non-numerical feature, I was unable to use the .corr() function to determine the correlation that it had. Instead, I used a catplot as shown below. Upon computing the correlation table, I was surprised to see how low of a correlation Pclass and Age has on Survived. The age had almost 0 correlation, the same as something I assumed insignificant like number of family members (Parch). Nevertheless, as expected, Pclass, and subsequently Fare, had the highest correlation with surviving. As such, Pclass and Fare are the most “important” features.

```
#CORRELATION STUFF
train.corr()
```

	Survived	Pclass	Age	SibSp	Parch	Fare
Survived	1.000000	-0.334068	-0.079472	-0.033395	0.082157	0.261742
Pclass	-0.334068	1.000000	-0.368625	0.080937	0.018212	-0.604960
Age	-0.079472	-0.368625	1.000000	-0.307639	-0.189194	0.100396
SibSp	-0.033395	0.080937	-0.307639	1.000000	0.415141	0.211816
Parch	0.082157	0.018212	-0.189194	0.415141	1.000000	0.263910
Fare	0.261742	-0.604960	0.100396	0.211816	0.263910	1.000000



Extracting Information from Non-Numerical-Features

As mentioned in the slide above, finding the correlation between a non-numeric feature like Sex was impossible to do with the .corr() function. As such, in order to extract information from non-numeric features, you could use Seaborn's plotting functions, such as pairplot and catplot, to model the information inside these non-numeric features. Additionally, it is possible to create tables using the groupby or sortby functions that display the non-numeric features in a numeric way. Finally, it is possible to use a pandas function called get_dummies() that converts the non-numeric feature into subcategories and then places a 1 or 0 depending on whether that subcategory is true or false. For instance, for the sex-dummies, it turned the Sex column into Male and Female and gave a checked whether the person was male. Thus, every male would get a 1, and every female a 0.

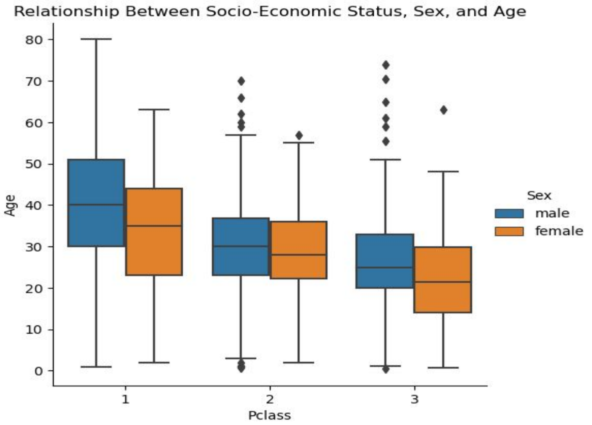
get_dummies

male
1
0
0
0
0
...
0
1
0
0
1

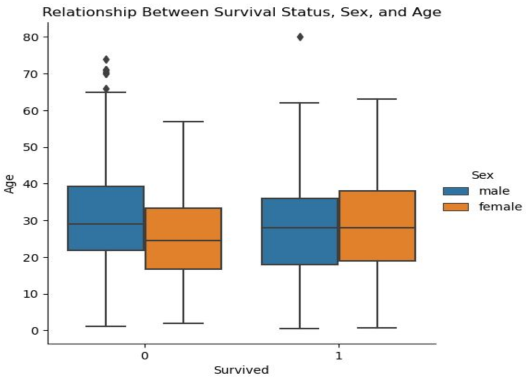
```
sex_dummies = pd.get_dummies(droped_train.Sex, drop_first=True)
embarked_dummies = pd.get_dummies(droped_train.Embarked, drop_first=True)
```

Cat-Plots

```
sns.catplot(data=train, x="Pclass", y="Age", kind="box", hue="Sex")
plt.show()
```



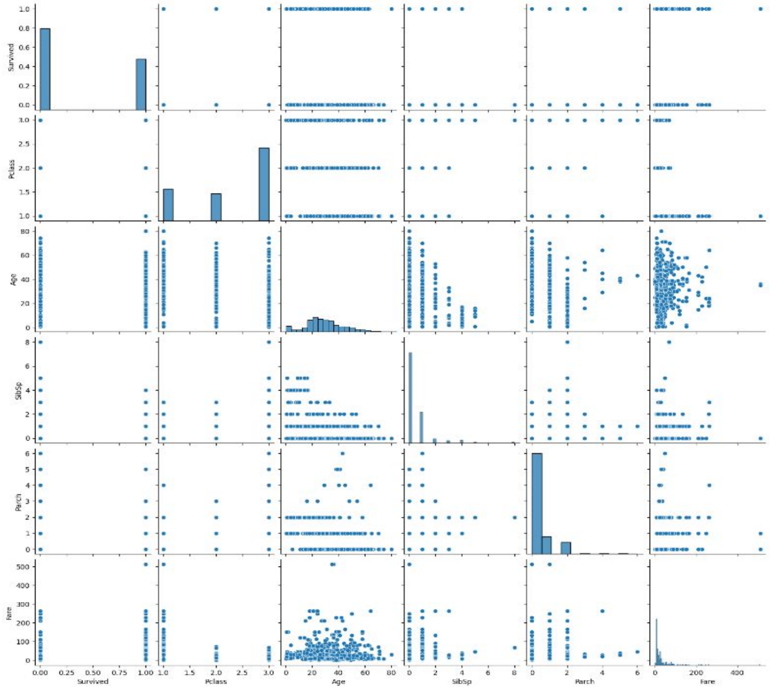

```
sns.catplot(data=train, x="Survived", y="Age", kind="box", hue="Sex")
plt.show()
```



Extracting Information from Non-Numerical-Features Continued

Pair-Plot

```
sns.pairplot(train.drop(columns=['PassengerId']))
plt.show()
```



Group-by

```
train.groupby(['Pclass','Sex']).size()
```

```
Pclass  Sex
1       female     93
        male      120
2       female     76
        male      108
3       female    144
        male      347
dtype: int64
```

Modeling and Question Answering

Splitting Data

The original data did not have the survived column in the test data-set, making it impossible to determine the accuracy of our model. As such, I split the original train data into two new sets called my_train, the new training data, and my_test, the new testing data.

```
prepped_data = prep_data(train)
prepped_data_x = prepped_data.drop(columns = ['Survived'])
prepped_data_y = prepped_data['Survived']

train_test_list = train_test_split(prepped_data)
my_train = train_test_list[0]
my_test = train_test_list[1]
```

Functions Used

```
def prep_data(data):
    droped_train = data.drop(columns=['Cabin']).dropna()
    sex_dummies = pd.get_dummies(droped_train.Sex, drop_first=True)
    embarked_dummies = pd.get_dummies(droped_train.Embarked, drop_first=True)

    new_train = pd.concat([droped_train.drop(columns=['Sex','Embarked']), sex_dummies, embarked_dummies], axis = 1)
    return new_train

def build_model(train_x, train_y, test_x, model_type, data_for_cv_x, data_for_cv_y):
    if model_type == "logistic_regression":
        model = LogisticRegression()
        cv_scores = cross_val_score(model, data_for_cv_x, data_for_cv_y, cv=5)

    elif model_type == "KNN":
        model  = KNeighborsClassifier(n_neighbors=3)
        cv_scores = cross_val_score(model, data_for_cv_x, data_for_cv_y, cv=5)
    else:
        model = RandomForestClassifier(random_state = 0)
        cv_scores = cross_val_score(model, data_for_cv_x, data_for_cv_y, cv=5)

    model.fit(train_x, train_y)
    res = model.predict(test_x)
    return res, cv_scores
```

Model 1: Logistic Regression

The first model I used was the logistic regression model. Mathematically, the logistic regression model takes the input and transforms the data, using log, to create a line to represent the data. Using the created line, the model checks the difference between each input's estimated value, as is derived by this line, and the actual value. The model takes the points with the smallest difference and uses that to predict future input values. By calculating the f1 value, recall value, precision value, and accuracy, the accuracy of the model is actually pretty high, since all the values are above 0.7.

```
my_train_x = my_train.drop(columns=['Survived'])
my_train_y = my_train['Survived']

my_test_x = my_test.drop(columns=['Survived'])
my_test_y = my_test['Survived']
```

```
lr_model = build_model(my_train_x, my_train_y, my_test_x, "logistic_regression", prepped_data_x, prepped_data_y)
lr_pred, lr_cv_scores = lr_model
```

```
print_all(my_test_y, lr_pred)
```

```
f1_score: 0.7273
recall_score: 0.7089
precision_score: 0.7467
accuracy_score: 0.764
```

```
lr_pred
```

```
array([0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1,
       0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0,
       0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0,
       1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1,
       0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1,
       0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0,
       1, 0], dtype=int64)
```

Model 2: K-Nearest-Neighbors

My second model is the K-nearest neighbors model. This model takes the inputted point and then the K nearest point to that point (the K is specified by the programmer; in this case I used 3). It then averages the output of the K inputs and uses that as a prediction. The f1, recall, precision, and accuracy score here was actually lower than the logistic regression. This is likely due to the knn model essentially cherry picking data, allowing a couple points to explain all the point, which is quite unreliable.

```
my_train_x = my_train.drop(columns=['Survived'])
my_train_y = my_train['Survived']

my_test_x = my_test.drop(columns=['Survived'])
my_test_y = my_test['Survived']
```

```
knn_model = build_model(my_train_x, my_train_y, my_test_x, "KNN", prepped_data_x, prepped_data_y)
knn_pred, knn_cv_scores = knn_model
```

```
print_all(my_test_y, knn_pred)
```

```
f1_score: 0.5839
recall_score: 0.5063
precision_score: 0.6897
accuracy_score: 0.6798
```

```
knn_pred
```

```
array([0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1,
       1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0,
       1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0,
       0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1,
       1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0,
       1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0,
       1, 0], dtype=int64)
```

Model 3: Random Forest

My final model was the random forest model. This model takes a large number of decision trees, each trained on a different portion of the data, and uses the average of each decision tree's output as a predictor for future inputs.This model worked quite well, with the f1, recall, precision, and accuracy score all being above 70. In fact, with a .75 accuracy score, this was the best out of the three models.

```
my_train_x = my_train.drop(columns=['Survived'])
my_train_y = my_train['Survived']

my_test_x = my_test.drop(columns=['Survived'])
my_test_y = my_test['Survived']
```

```
rf_model = build_model(my_train_x, my_train_y, my_test_x, "Random Forest", prepped_data_x, prepped_data_y)
rf_pred, rf_cv_scores = rf_model
```

```
print_all(my_test_y, rf_pred)
```

```
f1_score: 0.7261
recall_score: 0.7215
precision_score: 0.7308
accuracy_score: 0.7584
```

```
rf_pred
```

```
array([0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0,
       0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0,
       1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0,
       1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1,
       0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1,
       0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0,
       0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0,
       1, 0], dtype=int64)
```

Re-Splitting and Cross Validation Set

In the build_model function, I calculated the cross validation score of each model using a sklearn built in function. It can be noticed that although the logistic regression and random forest models were already quite accurate, the cross validation set for both models had a higher mean accuracy. This means that the cross validation set actually improved these models by nearly 5%. However, even after creating a cross validation set, the KNN model's accuracy didn't change much. This is likely due to the unreliable nature of the model.

```
#Accuracy of Model with Cross Validation
lr_cv_mean = round(np.mean(lr_cv_scores),4)
knn_cv_mean = round(np.mean(knn_cv_scores),4)
rf_cv_mean = round(np.mean(rf_cv_scores),4)

print('Mean Accuracy with Cross Validation Set of Logistic Regression Model: {}'.format(lr_cv_mean))
print('Mean Accuracy with Cross Validation Set of K-Nearest-Neighbors Model: {}'.format(knn_cv_mean))
print('Mean Accuracy with Cross Validation Set of Random Forest Model: {}'.format(rf_cv_mean))
```

```
Mean Accuracy with Cross Validation Set of Logistic Regression Model: 0.7885
Mean Accuracy with Cross Validation Set of K-Nearest-Neighbors Model: 0.6926
Mean Accuracy with Cross Validation Set of Random Forest Model: 0.7913
```

```
print_all(my_test_y, lr_pred)
```

```
f1_score: 0.7273
recall_score: 0.7089
precision_score: 0.7467
accuracy_score: 0.764
```

```
print_all(my_test_y, knn_pred)
```

```
f1_score: 0.5839
recall_score: 0.5063
precision_score: 0.6897
accuracy_score: 0.6798
```

```
print_all(my_test_y, rf_pred)
```

```
f1_score: 0.7261
recall_score: 0.7215
precision_score: 0.7308
accuracy_score: 0.7584
```