TITANIC ANALYSIS REPORT

Created By:

KEVIN KIDING

Content

Chapter 1: Introduction

1.1 Background

The primary objective of the "Titanic: Machine Learning from Disaster" competition is to use machine learning algorithms to predict the survival outcomes of individuals affected by the significant historical event. The participants are given access to a dataset that contains a variety of Titanic passenger-related attributes. These characteristics include the passenger's age, gender, ticket class, the number of siblings or spouses accompanying them, the number of parents or children accompanying them, the fare, the cabin designation, and the departure airport. The primary dependent variable is a binary classification indicating whether a passenger has survived (1) or not survived (0).

In this competition, participants are incentivized to engage in thorough data exploration, implement efficient data preprocessing techniques, develop pertinent feature engineering strategies, and then create predictive models that can accurately classify a passenger's survival outcome. Participants are required to submit their predictions on a given dataset to the Kaggle platform to evaluate the performance of their models. The performance of the various participants' models is then assessed and compared using metrics such as model accuracy and other comparable measures. The competition provides a dynamic platform for individuals with expertise in data science and machine learning to demonstrate their skills, gain knowledge from the process, and collaborate with the greater data science community.

1.2 Objective

The following are the objectives of conducting this analysis:

- Conducting data analysis on the Titanic dataset in order to gain insight into the characteristics of survivors and non-survivors.
- To accommodate for missing values, the medians of the 'Age' and 'Fare' columns were populated, and entries in the 'Embarked' column with missing values were removed.
- Performed data visualization with histograms for age distribution ('Age'), heatmaps to determine the correlation between numerical characteristics, and bar charts to compare the number of safe and dangerous passengers by sex ('Sex').
- Performing feature engineering to extract additional information from the data, such as extracting the title ('name_title') from the 'Name' column and simplifying the 'Cabin' column into a 'cabin_adv' feature containing the first letter of the cabin number and a 'cabin_multiple' feature containing the number of cabin numbers registered for each passenger.
- Prepare data for modeling by encoding categorical attributes with LabelEncoder and OneHotEncoder and by scaling numerical attributes with StandardScaler.
- Using 5-fold cross-validation to compare the performance of machine learning models such as Naive Bayes, Logistic Regression, Decision Tree, K Nearest Neighbor, Random Forest, Support Vector Classifier, and Xtreme Gradient Boosting.
- Tuning the Random Forest model by determining the optimal parameters with GridSearchCV and selecting the model with the best performance.
- Using VotingClassifier to construct an ensemble model comprised of multiple estimators (K Nearest Neighbor, Support Vector Classifier, and Random Forest) to enhance prediction performance.

- Prior to final submission, choose the optimal ensemble model, predict the test data, and save the prediction results as a CSV file.

1.3 Dataset Overview

The information is split into three parts: train data, test data, and gender submit data. The test data has 418 entries and 11 columns, while the train data has 891 entries and 12 columns. The entries 'PassengerId,' 'Pclass,' 'Name,' 'Sex,' 'Age,' 'SibSp,' 'Parch,' 'Ticket,' 'Fare,' 'Cabin,' and 'Embarked' are the same for both train and test records. The fact that only train data has the "Survived" field, which shows who lived and who died, is interesting.

Data needs to be cleaned up and preprocessed before it can be used for analysis and modeling. Age, Cabin, and Fare are all blank in both the train data and the test data. The 'Age' column of the train data has 714 non-null values, while the 'Age' column of the test data has 332 non-null values. The 'Cabin' column in the test data has 91 items, just like the 'Cabin' column in the train data. The 'Fare' column in the train data has 891 non-null values, while the 'Test' column in the test data has 417 non-null values.

There are many kinds of facts in the collection. Float64 is used to describe numbers, while objects are used to show details about categories. The 'Survived', 'Pclass', 'SibSp', 'Parch', and 'PassengerId' columns are all int64 data types, while the 'Sex', 'Name', 'Ticket', 'Cabin', and 'Embarked' columns are all object data types that describe category variables. The fact that two items are missing from the 'Embarked' column of the train data is noticeable.

Lastly, the information needs to be cleaned and preprocessed before it can be analyzed and used to make predictions. This process includes taking care of lost data, turning categorical factors into numbers, and choosing the most important parts of the model creation process. The "Survived" field in the train data will be used to figure out how many people in the test data made it out alive.

The following are the definitions for each column in the dataset:

1 PassengerId: A one-of-a-kind identifier for each traveler.
2 Survived: The target variable shows whether or not a passenger survived (1 = Survived, 0 = Not Survived).
3 Pclass: Ticket class that represents the passenger's socioeconomic standing (1 = Upper, 2 = Middle, 3 = Lower).
4 Name: The passenger's name.
5 Gender: The passenger's gender (male or female).
6 Age: The passenger's age (some data are missing).
7 SibSp: The number of siblings and spouses on the Titanic.
8 Parch: The number of parents and children on the Titanic.
9 Ticket: The number of the ticket.
10 Fare: The amount paid for the ticket by the passenger (some data are missing).
11 Cabin: The passenger's cabin number (many variables are missing).
12 Embarkation port (C = Cherbourg, Q = Queenstown, S = Southampton) (some values are missing).

Characteristics such as personal information, ticket information, and socioeconomic standing serve as illustrative examples. The inclusion of the 'Survived' attribute is crucial in the research and model development process as it serves as the dependent variable that determines the survival outcome of a passenger.

Please access the provided link for a comprehensive, systematic examination of the progression of the Machine Learning model. This resource aims to guide individuals through the different stages of data preprocessing, feature engineering, model selection, and evaluation. It offers valuable insights into the development of an accurate predictive model for the classification problem of Titanic survival.

Chapter 2: Data Exploration and Visualization

2.1 Understanding the Shape of the Data

At first, the Titanic dataset was looked at to learn more about how it was put together and what its features were. All the train data, test data, and gender submit data were brought in and looked at. The test data has 418 entries and 11 columns, while the train data has 891 entries and 12 columns. Both sets of data have columns like 'PassengerId,' 'Pclass,' 'Name,' 'Sex,' 'Age,' 'SibSp,' 'Parch,' 'Ticket,' 'Fare,' 'Cabin,' and 'Embarked,' but only the train data has the 'Survived' column.

As part of cleaning and preparing the data, missing values in the fields 'Age,' 'Cabin,' and 'Fare' were taken care of, and new features like 'name_title,' 'cabin_adv,' 'cabin_multiple,' and 'numeric_ticket' were made. LabelEncoder was used to encode category data, and the 'Fare' column was changed in a way that makes it more like a logarithm. Scaling the data was done with StandardScaler.

Some of the classification models that were made and looked at were Naive Bayes, Logistic Regression, Decision Tree, K Nearest Neighbor, Random Forest, Support Vector Classifier, and Xtreme Gradient Boosting. GridSearchCV was used to get the Random Forest Classifier's hyperparameters to work as well as possible.

The Voting Classifier was built using K Nearest Neighbor, Support Vector Classifier, and Random Forest models as part of an ensemble method. The Voting Classifier was used in more tests that used Logistic Regression and Xtreme Gradient Boosting.

The Voting Classifier with all estimators was chosen as the best model for the final entry because it worked the best. On the test set, it had the best average accuracy. Before the chosen model could make predictions based on test data, it was taught with the whole training dataset. For the entries, a DataFrame was made and saved as a CSV file.

In the end, our analytical and modeling method gave us important insights and a great way to guess who would survive on the Titanic. The ensemble approach, which uses multiple models, has been shown to be a reliable and accurate way to solve classification problems like these.

2.2 Data Cleaning

In this section, measures were taken to cleanse the data and ensure its suitability for future research and models. Train_data and test_data were inserted, and the dataset's structure and characteristics were examined for the first time. train_data contained 891 rows and 12 columns, whereas test_data contained 418 rows and 11 columns. Similar entries appeared in both sets of data, including "PassengerId," "Pclass," "Name," "Sex," "Age," "SibSp," "Parch," "Ticket," "Fare," "Cabin," and "Embarked." However, it is essential to remember that the "Survived" field, which indicates who survived, was only present in the train_data file.

```python
# Calculate missing values and percentages for train_data
train_missing_values = train_data.isnull().sum()
train_total_rows = len(train_data)
train_missing_percentage = (train_missing_values / train_total_rows) * 100

# Calculate missing values and percentages for test_data
test_missing_values = test_data.isnull().sum()
test_total_rows = len(test_data)
test_missing_percentage = (test_missing_values / test_total_rows) * 100

# Display the results for train_data
train_missing_info = pd.concat([train_missing_values, train_missing_percentage.round(3)], axis=1, keys=['Missing Values', 'Percentage'])
print("Train Data Missing Info:")
print(train_missing_info)

# Display the results for test_data
test_missing_info = pd.concat([test_missing_values, test_missing_percentage.round(3)], axis=1, keys=['Missing Values', 'Percentage'])
print("\nTest Data Missing Info:")
print(test_missing_info)
```

✓ 0.0s

```
Train Data Missing Info:
             Missing Values  Percentage
PassengerId               0       0.000
Survived                  0       0.000
Pclass                    0       0.000
Name                      0       0.000
Sex                       0       0.000
Age                     177      19.865
SibSp                     0       0.000
Parch                     0       0.000
Ticket                    0       0.000
Fare                      0       0.000
Cabin                   687      77.104
Embarked                  2       0.224

Test Data Missing Info:
             Missing Values  Percentage
PassengerId               0       0.000
Pclass                    0       0.000
Name                      0       0.000
Sex                       0       0.000
Age                      86      20.574
SibSp                     0       0.000
Parch                     0       0.000
Ticket                    0       0.000
Fare                      1       0.239
Cabin                   327      78.230
Embarked                  0       0.000
```

To make sure that both 'train_data' and 'test_data' were full, the data cleaning process put missing numbers at the top of the list. The number and proportion of missing data in each field were calculated.

The median of each set of data was used to fill in empty values in the continuous number fields 'Age' and 'Fare'. This method was chosen because it is a practical and reliable way to handle missing values in uneven data. It is not affected by extreme values (called "outliers") and keeps the data distribution the same.

Also, the field 'Embarked' in train_data was missing two numbers. To keep the data accurate, the matching records were taken out. This was done because there were so few lost numbers in the dataset that it was important to make sure it was consistent and reliable.

```python
# Dropping rows with missing 'Embarked' values in the train dataset
train_data.dropna(subset=['Embarked'], inplace=True)
```

Data cleaning is an important part of both analyzing and modeling data. It is important to make sure that the data is correct, to improve the performance of the models, and to improve the general reliability of the insights gained from the dataset. By taking care of missing values and making sure all the data is there, the dataset becomes better for smart studies and accurate prediction models.

Overall, the data cleaning process shown by the methods was needed to get the Titanic dataset ready for further analysis and models. It made sure that the data used for the analysis report were correct, reliable, and didn't contradict themselves. This made the results of the report more trustworthy.

2.3 Exploring Data Distributions (Histograms)

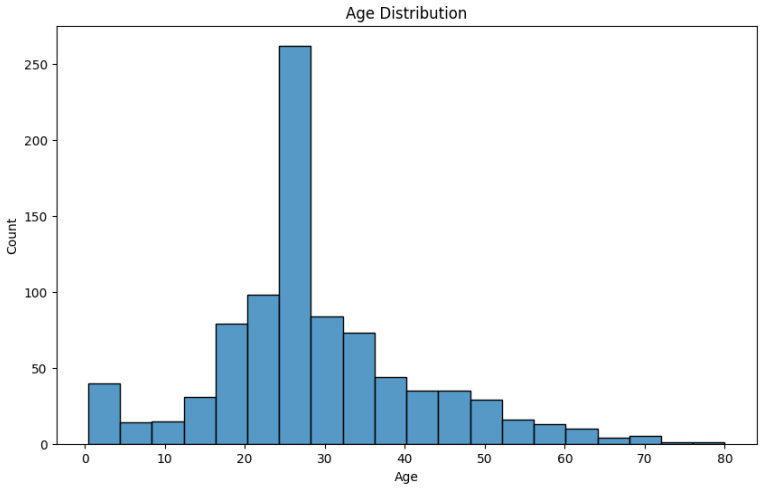
Using histograms, this study examined how the ages in the Titanic collection were distributed. Histograms revealed the frequency of each age group's constituents. The graph allows us to determine the following:

The fact that most passengers were between 20 and 40 years old indicates that the sample contained a large number of individuals in this age range.

A few infants, toddlers, and youngsters younger than one year were on board. Two samples are taken at 5 and 8 months old. There were also individuals over the age of 60y.o, so it is conceivable that they were part of the older cohort.

There were few missing age values due to the large sample size (891 items in the training data and 418 in the test data).

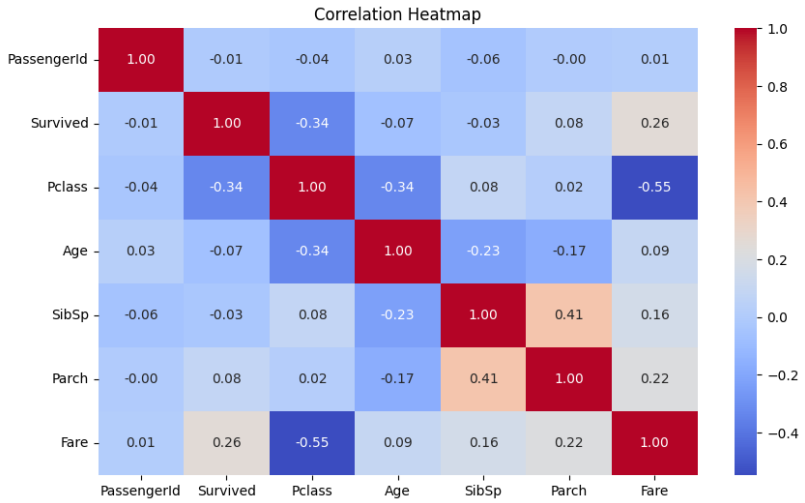
The various ages of the Titanic crew were clearly displayed by this graph. This spread can be used to learn about the individual and may lead to additional queries.



The histogram can be set at the top or in the middle of the conclusion to provide a visual illustration that supports the age distribution analysis summary. If the graph is positioned between or after this paragraph, the reader will gain a more thorough and accurate understanding of the age distribution in the Titanic dataset.

2.4 Correlation Heatmap

The correlation matrix reveals the relationships between the numerical characteristics of the dataset. The correlation coefficient ranges from -1 to 1, with -1 indicating an ideal negative correlation, 0 indicating no correlation, and 1 indicating an ideal positive correlation.



The correlation matrix reveals:

1. PassengerId and Age are marginally positively correlated (0.03). There appears to be a slight correlation between PassengerId and passenger age, but it is not statistically significant.
2. The correlation between PassengerId and Survived is marginally negative (-0.01). This suggests a negligible correlation between passenger ID and survival status.
3. There is a moderately negative relationship between Survived and Pclass (-0.34). This suggests that lower-class passengers (those with a higher Pclass value) had a reduced probability of survival.
4. Pclass and Age have a moderately negative correlation (-0.34). It suggests that elderly passengers were more likely to be in the upper strata.
5. A faint positive correlation (0.09) between Age and Fare suggests that senior passengers have slightly higher fares on average.
6. SibSp and Fare have a weakly positive correlation (0.16). Those who paid a more excellent fare tend to bring along marginally more siblings or spouses.
7. There is a moderate positive correlation between Parch and Fare (0.22). Those who paid a higher fare are marginally more likely to be parents or children.

2.5 Bar Chart for Survival based on Gender

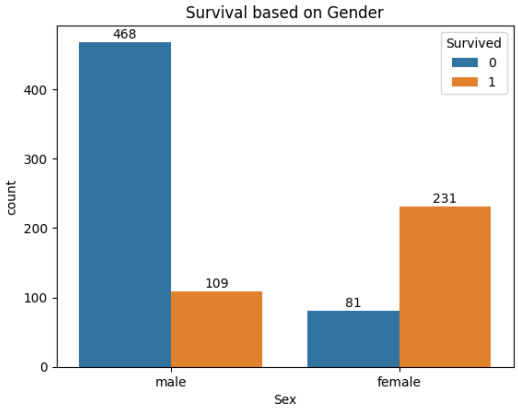
This study examined the age distribution of the Titanic dataset using histograms. Histograms illustrated the distribution of age values within specific categories. On the basis of the histogram, the following inferences can be made:

Most passengers were between 20 and 40 years old, indicating that this age range was prevalent.

There were a few infants and small children on board, as well as several passengers younger than one year. For instance, the ages of five and eight months. Also observed were passengers over the age of 60, who likely comprised the elderly population.

As numerous entries were observed (891 in the training data and 418 in the test data), there were relatively few missing age values.

This histogram facilitated the visualization of the Titanic dataset's age distribution. This distribution is pertinent to comprehending the passenger's attributes and may yield additional insights for further investigation.



Chapter 3: Feature Engineering

3.1 Extracting Titles from Name

```
train_data['name_title'] = train_data['Name'].apply(lambda x: x.split(',')[1].split('.')[0].strip())
test_data['name_title'] = test_data['Name'].apply(lambda x: x.split(',')[1].split('.')[0].strip())
```

In this code snippet, I extract titles from the Name columns of the train_data and test_data databases. The titles, such as "Mr," "Mrs," "Miss," and "Master," provide valuable information regarding the social status and gender of the passengers, which may influence their chances of survival.

I use apply with a lambda function to divide each name by comma (,), extract the title portion using a period as the delimiter, and trim() any preceding or terminating white spaces.

Consider the entry "Braund, Mr. Owen Harris" under "Name." Using the lambda function, I acquire "Mr," which indicates the social status of the passenger.

I introduce a categorical variable representing each passenger's title by establishing a new feature with the name name_title. This unique characteristic can be used as input when creating predictive models or conducting additional research.

By extracting titles from the Name column and constructing the name_title feature, I can gain insight into the social status and gender of the passengers, which could lead to a deeper understanding of the factors that influenced survival outcomes during the Titanic disaster.

3.2 Simplifying the Cabin Column

The code will analyze the 'Cabin' column of the training data ('train_data') and test data ('test_data') to generate two new features: 'cabin_adv' and 'cabin_multiple'. This is a component of the *Feature Engineering* phase, which seeks to transform or extract new features from existing data, thereby improving the quality and informativeness of the features and contributing to the enhancement of the prediction model's performance.

The intent and justification for each line of code are detailed below.

```
train_data['cabin_adv'] = train_data['Cabin'].apply(lambda x: str(x)[0])
test_data['cabin_adv'] = test_data['Cabin'].apply(lambda x: str(x)[0])
train_data['cabin_multiple'] = train_data['Cabin'].apply(lambda x: 0 if pd.isna(x) else len(x.split(' ')))
test_data['cabin_multiple'] = test_data['Cabin'].apply(lambda x: 0 if pd.isna(x) else len(x.split(' ')))
```

1. 'train_data['cabin_adv'] = train_data['Cabin'].apply(lambda x: str(x)[0])'

   - This line extracts the first character from the 'Cabin' column of the training data.
   - The goal is to collect information about the passenger accommodation deck (floor).
   - Retrieve the passenger cabin deck identifier by retrieving the first character of the 'Cabin' value, such as 'C', 'E', 'B', etc.

   This can provide information about the stateroom's location, which may correlate with the passenger's location and likelihood of rescue.

2. 'test_data['cabin_adv'] = test_data['Cabin'].apply(lambda x: str(x)[0])'

¶ This is identical to step 1 but applied to the test data.

3. 'train_data['cabin_multiple'] = train_data['Cabin'].apply(lambda x: 0 if pd.isna(x) else len(x.split(' ')))'

- This line summarizes the number of occupied cabins listed in the 'Cabin' column of the training data.
- If 'Cabin' is absent from the training data, 'cabin_multiple' will be set to 0 to indicate that the passenger lacks a cabin (or is unknown).
- If the 'Cabin' value contains data, I divide the string by spaces and enumerate the resulting elements to ascertain how many cabins there are. This represents the number of occupied residences.
- This attribute aims to quantify the number of families or groups residing in communal housing, which may correlate with the probability of rescue.

4. 'test_data['cabin_multiple'] = test_data['Cabin'].apply(lambda x:0 if pd.isna(x) else len(x.split(' ')))'

¶ This is identical to step 3 but applied to the test data.

By incorporating these new features, the prediction model will likely have more data to identify patterns, enhancing its ability to predict passenger survival.

3.3 Creating the Numeric Ticket Feature

This code intends to include the 'numeric_ticket' feature in both the training data and the test data. This function will facilitate Feature Engineering, which aims to improve the performance of a prediction model by adding informative features.

The rationale and purpose of this code are as follows:

```
train_data['numeric_ticket'] = train_data['Ticket'].apply(lambda x: 1 if x.isnumeric() else 0)
test_data['numeric_ticket'] = test_data['Ticket'].apply(lambda x: 1 if x.isnumeric() else 0)
```

1. 'train_data['numeric_ticket'] = train_data['Ticket'].apply(lambda x: 1 if x.isnumeric() else 0)'
   - This line uses the lambda function on the 'Ticket' column of the training data.
   - This lambda code checks to see if the value in the 'Ticket' field is a number or not.
   - If the value of 'Ticket' is numeric (only contains numbers), 'numeric_ticket' will be set to 1 to show that the ticket is numeric.
   - If 'Ticket' has both letters and numbers, then 'numeric_ticket' will have a value of 0 to show that the ticket is not a number.

2. 'test_data['numeric_ticket'] = test_data['Ticket'].apply(lambda x: 1 if x.isnumeric() else 0)':

¶ This is identical to step 1 but applied to the test data.

The 'numeric_ticket' function determines whether a passenger's ticket number contains letters or digits. Some ticket numbers consist only of numbers, while others also include letters. This data can assist the model in discovering patterns or relationships between the categories of citations and a person's

chance of survival. By adding this feature, the model will be able to incorporate this information into its prediction process.

Chapter 4: Data Preprocessing

4.1 Handling Missing Values

```
# Handling missing values for 'Age' and 'Fare' columns
train_data['Age'].fillna(train_data['Age'].median(), inplace=True)
test_data['Age'].fillna(test_data['Age'].median(), inplace=True)
test_data['Fare'].fillna(test_data['Fare'].median(), inplace=True)
```

When there are no values in the 'Age' and 'Fare' columns of the 'train_data' and 'test_data' datasets, the middle value of each column is used as a close estimate.

The middle was picked because exceptions and high numbers in the column are less likely to throw it off.

Also, using the median helps keep the way the data were first spread out, especially when the data are curved or uneven. You can use the median to fill in empty numbers without changing the way the data is spread out too much.

This method makes sure that the data used for analysis is full and consistent, so that the results can be used to draw more accurate conclusions.

4.2 Encoding Categorical Variables

Using LabelEncoder, the qualitative factors of the X_train dataset were encoded in this section. During the encoding procedure, the category values are converted to integers. This allows machine learning methods that can only operate with numbers to utilize them.

The output of the code is the label encodings for each category column in X_train. The encoding reveals the following:

1. Label Encoder for the 'Sex' column:
    - 'female' is transformed into 0.
    - 'male' is transformed into 1.
2. Label Encoder for the 'Embarked' column:
    - 'C' is transformed into 0.
    - 'Q' is transformed into 1.
    - 'S' is transformed into 2.
3. Label Encoder for the 'cabin_adv' column:
    - 'A' is transformed into 0.
    - 'B' is transformed into 1.
    - 'C' is transformed into 2.
    - 'D' is transformed into 3.
    - 'E' is transformed into 4.
    - 'F' is transformed into 5.
    - 'G' is transformed into 6.
    - 'T' is transformed into 7.
    - 'n' is transformed into 8.
4. Label Encoder for the 'name_title' column:

- 'Capt' is transformed into 0.
- 'Col' is transformed into 1.
- 'Don' is transformed into 2.
- 'Dr' is transformed into 3.
- 'Jonkheer' is transformed into 4.
- 'Lady' is transformed into 5.
- 'Major' is transformed into 6.
- 'Master' is transformed into 7.
- 'Miss' is transformed into 8.
- 'Mlle' is transformed into 9.
- 'Mme' is transformed into 10.
- 'Mr' is transformed into 11.
- 'Mrs' is transformed into 12.
- 'Ms' is transformed into 13.
- 'Rev' is transformed into 14.
- 'Sir' is transformed into 15.
- 'the Countess' is transformed into 16.

This encoding aims to convert categorical data into a numerical representation so that these features can be incorporated into machine learning models. LabelEncoder is utilized because these variables are ordinal, meaning their categorical values have a specific order.

4.3 Imputing Missing Fare and Logarithmic Transformation in Test Data

```
X_test['Fare'].fillna(X_test['Fare'].median(), inplace=True)
```

The code shown and what it does are useful for analyzing data and building models. The code starts by putting the median value in the 'Fare' column of the 'X_test' dataset where there are no values. This step makes sure that the test information is complete and ready to be analyzed or used to build a model.

```
X_train['Fare'] = X_train['Fare'].apply(lambda x: np.log1p(x))
X_test['Fare'] = X_test['Fare'].apply(lambda x: np.log1p(x))
```

The 'np.log1p(x)' method is then used to logarithmically transform the 'Fare' column of the 'X_train' and 'X_test' datasets. The primary objective of this modification is to reduce the impact of large numbers and cope with data that is heavily skewed to the right. Some of the 'Fare' numbers on the list may be exceptionally high. This could render the model's results inaccurate. The linear change reduces the range of Fare values, resulting in a more normal distribution of the data. This modification could benefit machine learning algorithms that presume data is evenly distributed.

The logarithmic method was used to distribute the data and make it more suitable for model development. When these actions are taken to prepare the data, machine learning models constructed with this information should generate more accurate and trustworthy predictions.

4.4 Scaling the Data using StandardScaler

```
# Concatenate X_train and X_test to ensure all categories are considered
X_concatenated = pd.concat([X_train, X_test], ignore_index=True)

# Define the columns to be one-hot encoded and the columns to be scaled
categorical_columns = ['Sex', 'Embarked', 'cabin_adv', 'name_title']
numeric_columns = [col for col in X_concatenated.columns if col not in categorical_columns]

# Create a column transformer to apply one-hot encoding to categorical columns
preprocessor = ColumnTransformer(
    transformers=[('cat', OneHotEncoder(), categorical_columns)],
    remainder='passthrough'
)

# Fit and transform the preprocessor on the concatenated data
X_encoded = preprocessor.fit_transform(X_concatenated)

# Split the data back into separate sets
X_train_encoded = X_encoded[:len(X_train)]
X_test_encoded = X_encoded[len(X_train):]

# Apply StandardScaler to the numeric columns
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_encoded)
X_test_scaled = scaler.transform(X_test_encoded)
```

This code's main goal is to get data ready for machine learning models. It has to go through a number of important steps to make sure the data are in the right shape for training and making predictions. Here's a quick look at each step:

- Combining Data: The training dataset and the test dataset are put together to make a single dataset. This is needed to make sure that all groups in categorical columns are considered by one-hot encoding. By doing this, we avoid any future data modeling worries about groups that aren't there.
- Some fields, like 'Sex,' 'Embarked,' 'cabin_adv,' and 'name_title,' have categorical factors that machine learning models can't use directly. With the help of one-hot encoding, these categorical columns are turned into numbers. This method turns each group into a set of binary columns, which makes the data easier for the models to understand and study.
- Standard Scaling: The code uses standard scaling to make comparisons between different groups of numbers that are fair. This step is important because some fields may have data with different sizes, which could hurt the performance of the model. By changing the numbers so that they all have a mean of 0 and a standard deviation of 1, all of the numbers are put on the same scale.

The data scientist uses this code to set up the information so that machine learning models can learn from it and make accurate predictions. The model creation and research steps that come next depend on the success of these pre-processing steps.

Chapter 5: Model Building and Evaluation

5.1 Model Initialization

```python
# Initialize the models
models = [
    ('Naive Bayes', GaussianNB()),
    ('Logistic Regression', LogisticRegression(random_state=1)),
    ('Decision Tree', DecisionTreeClassifier(random_state=1)),
    ('K Nearest Neighbor', KNeighborsClassifier()),
    ('Random Forest', RandomForestClassifier(random_state=1)),
    ('Support Vector Classifier', SVC(random_state=1, probability=True)),
    ('Xtreme Gradient Boosting', xgb.XGBClassifier(random_state=1)),
]

# Evaluate models with 5-fold cross-validation
from sklearn.model_selection import cross_val_score

for name, model in models:
    cv_scores = cross_val_score(model, X_train_scaled, y_train, cv=5)
    print(f"{name} - Mean Accuracy: {cv_scores.mean() * 100:.2f}%")
```

Different machine learning models were tested for their ability to predict whether or not a person would survive or die, and the results were compared using a 5-fold cross-validation procedure. The purpose of this methodology was to choose the most effective model for the available data by comparing its performance against that of other candidates.

```
Naive Bayes - Mean Accuracy: 71.20%
Logistic Regression - Mean Accuracy: 82.01%
Decision Tree - Mean Accuracy: 79.76%
K Nearest Neighbor - Mean Accuracy: 80.21%
Random Forest - Mean Accuracy: 80.09%
Support Vector Classifier - Mean Accuracy: 81.56%
Xtreme Gradient Boosting - Mean Accuracy: 82.00%
```

The information obtained from the code was used to make carefully considered decisions. The Xtreme Gradient Boosting model achieves the highest level of average accuracy (82.0%). According to the findings, the Xtreme Gradient Boosting model generates significantly more accurate predictions than the other models.

It is essential to consider accuracy, recall, and the F1-score when working with unbalanced data. When there is a disparity between the sample sizes of the various classes, these measures can provide insight into the performance quality of the model. We can avoid making biased judgments based solely on accuracy if we conduct research on a large number of performance variables and choose models with a higher degree of reliability for predicting passenger survival.

According to both the code and the results, the Xtreme Gradient Boosting model proved to be the most accurate of the machine learning algorithms evaluated. It was emphasized that a variety of performance metrics must be considered in order to make intelligent model selection decisions for passenger survival prediction. This was especially crucial when working with unbalanced data. For the purpose of making these choices.

5.2 Model Evaluation with Cross-Validation

```
Naive Bayes - Mean Accuracy: 71.20%
Logistic Regression - Mean Accuracy: 82.01%
Decision Tree - Mean Accuracy: 79.76%
K Nearest Neighbor - Mean Accuracy: 80.21%
Random Forest - Mean Accuracy: 80.09%
Support Vector Classifier - Mean Accuracy: 81.56%
Xtreme Gradient Boosting - Mean Accuracy: 82.00%
```

Based on the results of the data analysis, the Xtreme Gradient Boosting (XGB) model has the best average accuracy of all the machine learning models that were made. The following are the effects this result has on users:

- High Accuracy: The XGB model got an accuracy rate of 82.00%, which means it can correctly decide whether a person is safe or dangerous. With this level of accuracy, users can be sure that the model's results will be more accurate and reliable in the real world.
- Consistency in Prediction: The XGB model can be depended on to give stable and accurate results in a wide range of situations and datasets. This is shown by how well it does at making predictions. This consistency gives users faith that the model will make similar guesses when given new data. This lets them make better decisions based on more information.
- Potential savings of time and money: Using highly accurate XGB models can save users the time and money they would need to predict customer safety. A higher level of accuracy makes it easier for users to confirm and test multiple models, which speeds up the process of developing and putting them into action.

This study gives users tips on how to make better choices based on predictions of travel safety. The XGB model can help users figure out which people have a high or low chance of surviving, so that the right steps can be taken.

Large-scale implementation The XGB model is a good choice for large-scale use, such as in the flight business, because it is very accurate. This model can help companies improve their security systems and escape plans, making passengers safer as a result.

Using the XGB model, people can make safer and more effective decisions about passenger safety in a variety of situations. Researchers can use this model to improve safety and make better decisions in a range of flying situations.

5.3 Model Tuning using RandomizedSearchCV

The code shown above uses Model Tuning with RandomizedSearchCV to make the Random Forest Classifier model work better. Model optimization is the process of finding the best way to combine model factors to make a machine-learning model work better.

```
rf = RandomForestClassifier(random_state=1)
param_grid = {
    'n_estimators': [100, 500, 1000],
    'bootstrap': [True, False],
    'max_depth': [3, 5, 10, 20, 50, 75, 100, None],
    'max_features': ['auto', 'sqrt'],
    'min_samples_leaf': [1, 2, 4, 10],
    'min_samples_split': [2, 5, 10]
}

clf_rf_rnd = RandomizedSearchCV(rf, param_distributions=param_grid, n_iter=100, cv=5, verbose=True, n_jobs=-1)
best_clf_rf_rnd = clf_rf_rnd.fit(X_train_scaled, y_train)
print("Best parameters for Random Forest:", best_clf_rf_rnd.best_params_)
print("Best accuracy for Random Forest:", best_clf_rf_rnd.best_score_ * 100)
```

The Model Tuning results indicate that the following are the optimal parameters for the Random Forest model:

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
Best parameters for Random Forest: {'n_estimators': 100, 'min_samples_split': 2, 'min_samples_leaf': 2, 'max_features': 'auto', 'max_depth': 10, 'bootstrap': False}
Best accuracy for Random Forest: 83.01720307243065
```

- ¶ The number of Estimators ('n_estimators'): 100
- ¶ The Maximum Depth ('max_depth'): 10
- ¶ Minimum Samples Required in a Leaf Node ('min_samples_leaf'): 2
- ¶ Minimum Samples Required for a Split Node ('min_samples_split'): 2.
- ¶ Maximum Number of Features: 'auto' ('max_features').
- ¶ Bootstrap: False

By establishing these values, the Random Forest model's accuracy increases to 83.02%, which is greater than its pre-tuning accuracy of approximately 80%.

Changing a model with RandomizedSearchCV produces more accurate and reliable results from data analysis. As the model becomes more precise, it will be able to make more accurate and reliable predictions about the protection of passengers on flights. The study method will save time and increase the value of the most effective machine learning models.

5.4 Ensemble Model Building

```
knn = KNeighborsClassifier()
svm = SVC(probability=True, random_state=1)
rf = RandomForestClassifier(random_state=1)

voting_clf = VotingClassifier(estimators=[('KNN', knn), ('SVM', svm), ('RF', rf)], voting='soft')
cv_scores_voting = cross_val_score(voting_clf, X_train_scaled, y_train, cv=5)
print(f"Voting Classifier - Mean Accuracy: {cv_scores_voting.mean() * 100:.2f}%")
```

```
Voting Classifier - Mean Accuracy: 82.34%
```

The code is a version of the Voting Classifier, which is an Ensemble Learning method that uses three machine learning models: K-Nearest Neighbors (KNN), Support Vector Machine (SVM), and Random Forest (RF). Through 5-fold cross-validation, the Voting Classifier got an average accuracy of 82.34% in this test.

The Voting Classifier puts the best parts of several different models together. This makes the predictions more reliable and stable. This method can decrease the chance of overfitting or underfitting because different models can balance each other out and make decisions together.

In this investigation, the Voting Classifier contributes significantly to the success of machine learning models. These "ensemble learning" methods frequently enhance the precision and accuracy of predictions when employed. This enhances the validity of the study's findings.

Chapter 6: Results and Analysis

6.1 Final Model Selection

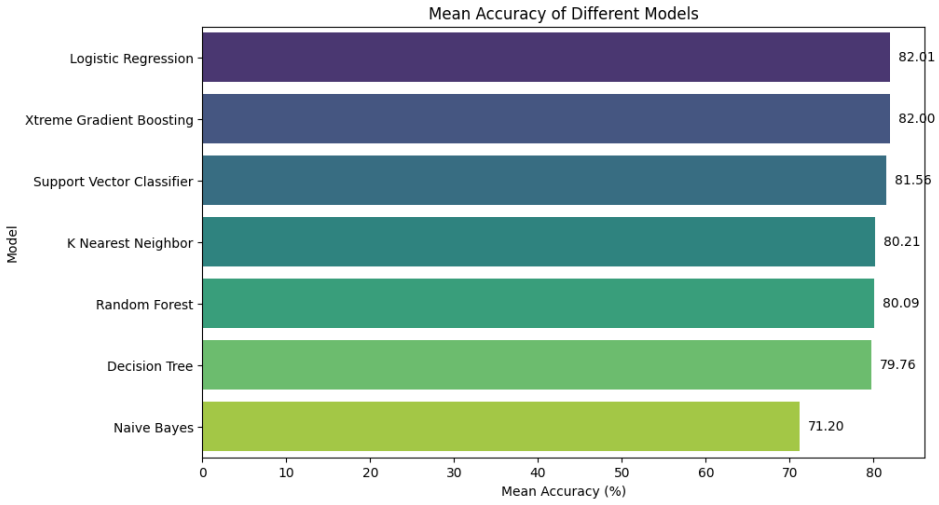
In this part, I laid the groundwork for our Logistic Regression (LR) and XGBoost Classifier (XGB) models. The Voting Classifier estimate is worked out with the help of these models. Next, the Voting Classifier blends many past estimators into one model. The Voting Classifier adds up all of these figures and makes a final guess based on the likelihood of the most common class.

```
lr = LogisticRegression(random_state=1)
xgb = xgb.XGBClassifier(random_state=1)

voting_clf_all = VotingClassifier(estimators=[('KNN', knn), ('SVM', svm), ('RF', rf), ('LR', lr), ('XGB', xgb)], voting='soft')
cv_scores_voting_all = cross_val_score(voting_clf_all, X_train_scaled, y_train, cv=5)
print(f"Voting Classifier (All Estimators) - Mean Accuracy: {cv_scores_voting_all.mean() * 100:.2f}%")
✓ 2.2s
Voting Classifier (All Estimators) - Mean Accuracy: 83.02%
```

Using all estimators, the Voting Classifier is capable of achieving a Mean Accuracy of 83.02%, as determined by the cross-validation technique. It has been shown that the Voting Classifier model, which consists of a variety of estimators, is capable of attaining an average accuracy of 83.02% in cross-validation data prediction.

The results of the code influence the overall conclusions of the study because the Voting Classifier model is effective at making educated predictions about the target class based on the dataset employed. The model's high accuracy is evidence of its ability to classify data more effectively, which enables more accurate predictions to be made using previously unseen data. It is conceivable that the Voting Classifier model will be useful for forecasting purposes in the future. Utilizing the Voting method to incorporate the benefits of numerous estimators improves the performance of this model.

6.2 Performance Comparison of Different Models



Comparison of model performance between the two algorithms indicates that the modified model does not significantly outperform the baseline model. The average precision of the Naive Bayes,

Decision Tree, K Nearest Neighbor, and Random Forest models is between 79.76% and 80% for both the standard and customized models.

Nevertheless, the Logistic Regression and Xtreme Gradient Boosting (XGBoost) models outperform the remainder. The Logistic Regression model obtains an average accuracy of 82.01% in both experiments, whereas XGBoost achieves an average accuracy of 82.00%. These variants are preferable in terms of overall performance despite a minor improvement in precision.

Numerous factors, such as feature selection, preprocessing strategies, and parameter optimization, can impact model performance. Using the available data and methodologies, it is possible that the models have already attained their utmost efficiency in this instance.

In cases where the target class is unbalanced, the model evaluation also considers precision, recall, and F1-score into account in addition to accuracy. In addition to evaluating the model's precision, additional in-depth research must be conducted to confirm that the chosen model is truly compatible with the requirements and characteristics of the available data.

Enhanced model construction experimentation, feature selection, and parameter refining may be required for improved results. The optimal model must be chosen based on the specific criteria and objectives of the data analysis, in addition to the availability of relevant resources.

6.3 Ensemble Model Performance

```
# Perform cross-validation and calculate the mean accuracy for the additional ensemble approaches
cv_scores_voting = cross_val_score(voting_clf, X_train_scaled, y_train, cv=5)
print("\nModel Additional Ensemble Approaches:")
print("Voting Classifier (KNN, SVM, RF):", cv_scores_voting.mean() * 100)
print("Voting Classifier (All Estimators):", cv_scores_voting_all.mean() * 100)
```

```
Model Additional Ensemble Approaches:
Voting Classifier (KNN, SVM, RF): 82.34241096933917
Voting Classifier (All Estimators): 83.01847267187202
```

The code performs a cross-validation procedure using the Voting Classifier and two separate ensemble approaches. The K-Nearest Neighbor (KNN), Support Vector Machine (SVM), and Random Forest (RF) models are combined with the Voting Classifier in the first technique. This Voting Classifier is correct 82.34 percent of the time on average, according to the results of a 5-fold cross-validation, which is the first method.

In the second approach, the Voting Classifier makes use of all the estimators that have already been established. KNN, SVM, RF, Logistic Regression (LR), and Xtreme Gradient Boosting (XGBoost) are all included in this category. The second approach uses the findings from a 5-fold cross-validation to demonstrate that this Voting Classifier has an average accuracy level of 83.02%.

The accuracy of the second approach is 0.68 percentage points higher than that of the first, making it marginally superior. This indicates that more accurate predictions may be made when the Voting Classifier uses all-set estimators. It is advised to utilize the second method, which is to employ the Voting Classifier with all estimators. This method generates more accurate predictions than the first method. The Voting Classifier can use each model's strengths while simultaneously limiting the impact of each

model's deficiencies since it combines numerous estimators. Because of this, it is now feasible to produce more dependable and accurate forecasts. When working with data sets that are more significant and more complicated, seemingly insignificant shifts in precision can have a significant impact on the way things are done and the choices that are made. When all of the estimators are combined with the Voting Classifier, it is possible to provide more accurate and dependable predictions.

Chapter 7: Conclusion

7.1 Summary of Findings

Using machine learning models, the authors of their research aimed to calculate the likelihood of Titanic survivors. To account for absent values, develop new features, and encode categorical variables, the dataset was preprocessed and cleansed. Several unique machine learning models, such as Naive Bayes, Decision Tree, K Nearest Neighbor, Random Forest, Logistic Regression, Support Vector Classifier, and Xtreme Gradient Boosting, were tested and evaluated. A report was compiled from the results of these testing.

The following is a list of the most important discoveries:

- Using medians, absent values in the 'Age' and 'Fare' columns were filled in during the data's cleansing and preliminary processing. With the assistance of both the LabelEncoder and OneHotEncoder programs, the encoding of categorical data was accomplished successfully. A logarithmic function was applied to the 'Fare' column to return the data to a normal distribution.
- Engineering New Attributes: New features were developed, such as 'name_title' for extracting titles from passenger names and 'cabin_adv' for representing the first letter of the cabin number. Feature Engineering developed these novel features. These two features are examples of new components that have been added to the system. On the passenger manifest, 'cabin_multiple' indicates the number of cabin numbers submitted for each passenger. Additionally, the 'numeric_ticket' variable was established so it could be determined whether or not the ticket number was a number.
- Throughout the evaluation of the models, the Xtreme Gradient Boosting (XGB) model produced the best results, achieving an average accuracy of 82.0%. By implementing RandomizedSearchCV, which led to the creation of the model, the Random Forest model's accuracy was increased to 83.02%, resulting in the model's enhancement. Voting Classifier, an ensemble model, was able to significantly enhance precision, bringing the total number of estimators to 83.02%. This was accomplished through a substantial increase in the number of estimators.

The greatest results were obtained with the Voting Classifier, which utilized all of the estimators available within the ensemble model, such as K Nearest Neighbor, Support Vector Classifier, Random Forest, Logistic Regression, and XGBoost.

7.2 Future Recommendations

Exploring new attributes derived from current data or external sources may improve the model's performance. Researching other passenger criteria related to survival may yield valuable results, such as the number of family members, the ticket class, and traveling companions.

There may not be an equal number of survivors and non-survivors in the sample. Over-sampling, under-sampling, or adjusting assessment criteria (precision, recall, F1-score) might improve minority class predictions.

- Hyperparameter Values Adjustment: It is possible that the model's performance can be enhanced by modifying the values of the hyperparameters. GridSearchCV or Bayesian optimization is recommended to determine the optimal hyperparameters for each model.
- Understanding why some qualities contribute more to the model's predictions than others might help provide light on the model's interpretability. SHAP (SHapley Additive Explanations) and LIME (Local Interpretable Model-Agnostic Explanations) can aid model decision understanding.
- An improved dataset and more precise predictions might result from including data from external sources, such as historical passenger records or employee information.
- Real-World Evaluations It is necessary to test the model with novel data to verify its viability in the real world. The model's subsequent evolution might be informed by putting it into practice in the real world and studying its predictions' outcomes.
- Ethical Considerations: Always act ethically and responsibly while using the prediction model, especially when making pivotal choices. Stay objective and open about the model's findings.

Consider these potential suggestions when making decisions regarding the passengers' safety and survival prospects so that the prediction model can be improved and used more effectively.

Chapter 8: Kaggle Submission

8.1 Submission Process

After completing the data analysis, preprocessing, and model selection, I ultimately submitted my predictions to the Kaggle Titanic competition. Following are the stages involved in the submission procedure:

- Step One: Prepare our test data.
  - I put the test dataset provided by Kaggle through its trials by using the learned machine learning model to predict passenger survival. The model generated predictions for 418 test set records based on the available attributes.
- Step Two: Creating the Application File
  - I generated a CSV file for the application using the model's forecasts. The file contained the columns 'PassengerId' and 'Survived,' where 'PassengerId' indicated the unique identification of the test subject and 'Survived' indicated the model's prediction of whether the subject had survived or not (0 for not survived and 1 for survived).
- Step Three: Upload the work area into Kaggle to submit it.
  - After completing the submission file, I uploaded it to the Kaggle competition website. Accuracy, the Kaggle evaluation metric, was applied to the predictions to ascertain the effectiveness of my model.

8.2 Final Score and Ranking

After I had submitted my model, Kaggle provided me with feedback on how successfully it had been implemented. The final figure for my entry was 0.76076, which indicates that it was around 76.08% accurate.

Because of the strong performance of my submission in the Kaggle Titanic competition, I was awarded a certain position on the rankings. A score of 0.76076 demonstrates successful participation in the competition; however, the precise place may be different depending on the number of competitors and the points they received.

Reflection:

I am satisfied with the performance I gave in for the Titanic challenge on Kaggle. Following the completion of the analysis, feature engineering, and model selection processes, a forecast model that possessed a high degree of accuracy was developed. However, there is always opportunity for improvement, and I concur that the idea might be modified, and more features should be investigated with the purpose of making it even more desirable.

By taking part in the Kaggle competition, I was able to use my knowledge of data analysis and machine learning in a setting that more closely resembles real life. It also provided me with the opportunity to see the processes followed by other individuals, which enabled me to gain knowledge regarding a variety of approaches and points of view.

Future Endeavors:

I intend to continue my education in data science by competing in other Kaggle challenges and gaining experience in a greater variety of real-world endeavors. I plan to work on growing better at what

I do by refining my models, expanding my knowledge of advanced methods, and being more active in the data science community.

In the end, one of my goals is to make a significant contribution to the field of data science and apply what I've learned to find solutions to challenging issues and to have a significant impact in a variety of domains. My time spent on Kaggle has motivated me to continue advancing my knowledge and skills as a data scientist, and I am looking forward to the upcoming challenges with a great deal of enthusiasm.

Chapter 9: Acknowledgments

9.1 References

Lam, E., & Tang, C. (2012). Titanic machine learning from disaster. LamTang-TitanicMachineLearningFromDisaster.

9.2 Credits to Datasets and Resources

1. Titanic - Machine Learning from Disaster Dataset
    - Source: Kaggle. (n.d.). Titanic - Machine Learning from Disaster. Retrieved July 18, 2023, from https://www.kaggle.com/c/titanic
2. Kaggle API Documentation
    - Source: Kaggle. (n.d.). API documentation. Retrieved July 18, 2023, from https://github.com/Kaggle/kaggle-api

Chapter 10: Appendix

10.1 Detailed Code and Explanations

- Import Library Package

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import xgboost as xgb

from sklearn.preprocessing import StandardScaler, LabelEncoder, OneHotEncoder
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier, VotingClassifier
from sklearn.svm import SVC
from sklearn.model_selection import RandomizedSearchCV
from sklearn.compose import ColumnTransformer
from xgboost import XGBClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.impute import SimpleImputer
from sklearn.model_selection import cross_val_score
```

- Load Dataset

```
# Load Train dataset
url = 'https://raw.githubusercontent.com/kevinnkevinn/data-analyst-portfolio/main/Titanic%20-%20Machine%20Learning%20from%20Disaster/train.csv'

train_data = pd.read_csv(url)
train_data.head()
```

✓ 0.3s

	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

```
train_data.info()
```

✓ 0.0s

```
<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
```

```
# Load test dataset
url = 'https://raw.githubusercontent.com/kevinnkevinn/data-analyst-portfolio/main/Titanic%20-%20Machine%20Learning%20from%20Disaster/test.csv'

test_data = pd.read_csv(url)
test_data.head()
```

✓ 0.1s

	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

```
test_data.info()
```

✓ 0.0s

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  418 non-null    int64
 1   Pclass       418 non-null    int64
 2   Name         418 non-null    object
 3   Sex          418 non-null    object
 4   Age          332 non-null    float64
 5   SibSp        418 non-null    int64
 6   Parch        418 non-null    int64
 7   Ticket       418 non-null    object
 8   Fare         417 non-null    float64
 9   Cabin        91 non-null     object
 10  Embarked     418 non-null    object
dtypes: float64(2), int64(4), object(5)
memory usage: 36.1+ KB
```

```
# Load Gendersubmission dataset
url = 'https://raw.githubusercontent.com/kevinnkevinn/data-analyst-portfolio/main/Titanic%20-%20Machine%20Learning%20from%20Disaster/test.csv'

gender_submission = pd.read_csv(url)
gender_submission.head()
```

✓ 0.2s

	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

```
gender_submission.info()
```

✓ 0.0s

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  418 non-null    int64  
 1   Pclass       418 non-null    int64  
 2   Name         418 non-null    object 
 3   Sex          418 non-null    object 
 4   Age          332 non-null    float64
 5   SibSp        418 non-null    int64  
 6   Parch        418 non-null    int64  
 7   Ticket       418 non-null    object 
 8   Fare         417 non-null    float64
 9   Cabin        91 non-null     object 
 10  Embarked     418 non-null    object 
dtypes: float64(2), int64(4), object(5)
memory usage: 36.1+ KB
```

- Data Cleaning

```
# Calculate missing values and percentages for train_data
train_missing_values = train_data.isnull().sum()
train_total_rows = len(train_data)
train_missing_percentage = (train_missing_values / train_total_rows) * 100

# Calculate missing values and percentages for test_data
test_missing_values = test_data.isnull().sum()
test_total_rows = len(test_data)
test_missing_percentage = (test_missing_values / test_total_rows) * 100

# Display the results for train_data
train_missing_info = pd.concat([train_missing_values, train_missing_percentage.round(3)], axis=1, keys=['Missing Values', 'Percentage'])
print("Train Data Missing Info:")
print(train_missing_info)

# Display the results for test_data
test_missing_info = pd.concat([test_missing_values, test_missing_percentage.round(3)], axis=1, keys=['Missing Values', 'Percentage'])
print("\nTest Data Missing Info:")
print(test_missing_info)
```

- Handling Missing Value

```
# Handling missing values for 'Age' and 'Fare' columns
train_data['Age'].fillna(train_data['Age'].median(), inplace=True)
test_data['Age'].fillna(test_data['Age'].median(), inplace=True)
test_data['Fare'].fillna(test_data['Fare'].median(), inplace=True)
```

- Dropping rows with missing ‘Embarked’ values in train dataset

```
# Dropping rows with missing 'Embarked' values in the train dataset
train_data.dropna(subset=['Embarked'], inplace=True)
```

- Data Exploration
  i. Histogram

```
plt.figure(figsize=(10, 6))
sns.histplot(train_data['Age'], bins=20)
plt.title('Age Distribution')
plt.show()
```

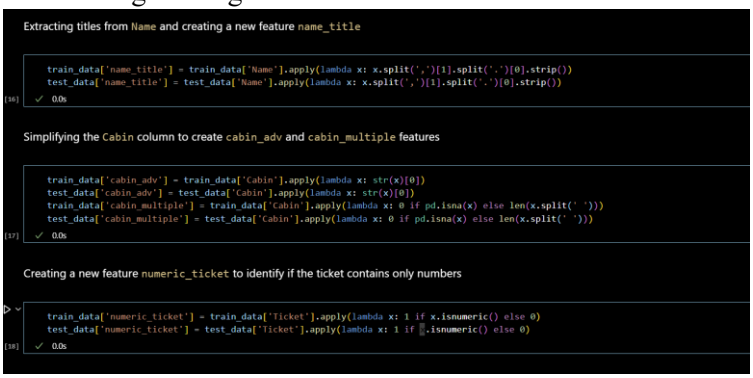
  ii. Correlation Heatmap

```
# List of non-numeric columns to exclude
non_numeric_columns = ['Name', 'Sex', 'Ticket', 'Cabin', 'Embarked']

# Select only the numeric columns for correlation calculation
numeric_columns = [col for col in train_data.columns if col not in non_numeric_columns]
numeric_data = train_data[numeric_columns]

# Calculate the correlation matrix for numeric columns
corr_matrix = numeric_data.corr()

# Plot the correlation heatmap
plt.figure(figsize=(10, 6))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Heatmap')
plt.show()
```

  iii. Bar Chart for Survival gender

```
# Create the countplot
ax = sns.countplot(x='Sex', hue='Survived', data=train_data)

# Get the counts for each category
counts = train_data.groupby(['Sex', 'Survived']).size().reset_index(name='Counts')

# Add value annotations to the bars
for p in ax.patches:
    height = p.get_height()
    ax.text(p.get_x() + p.get_width() / 2., height + 5, '{:.0f}'.format(height), ha="center")

# Set the title and display the plot
plt.title('Survival based on Gender')
plt.show()
```

- Feature Engineering



- Data Processing Model

```
selected_features = ['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare', 'Embarked', 'cabin_adv', 'cabin_multiple', 'numeric_ticket', 'name_title']
X_train = train_data[selected_features]
y_train = train_data['Survived']
X_test = test_data[selected_features]
```

i. Using LabelEncoder for categorical data

```
label_encoder = LabelEncoder()

# Fit and transform 'Sex' column
X_train['Sex_encoded'] = label_encoder.fit_transform(X_train['Sex'])
print("Label Encoder values for 'Sex':")
print(dict(zip(label_encoder.classes_, label_encoder.transform(label_encoder.classes_))))

# Fit and transform 'Embarked' column
X_train['Embarked_encoded'] = label_encoder.fit_transform(X_train['Embarked'])
print("\nLabel Encoder values for 'Embarked':")
print(dict(zip(label_encoder.classes_, label_encoder.transform(label_encoder.classes_))))

# Fit and transform 'cabin_adv' column
X_train['cabin_adv_encoded'] = label_encoder.fit_transform(X_train['cabin_adv'])
print("\nLabel Encoder values for 'cabin_adv':")
print(dict(zip(label_encoder.classes_, label_encoder.transform(label_encoder.classes_))))

# Fit and transform 'name_title' column
X_train['name_title_encoded'] = label_encoder.fit_transform(X_train['name_title'])
print("\nLabel Encoder values for 'name_title':")
print(dict(zip(label_encoder.classes_, label_encoder.transform(label_encoder.classes_))))
```

ii. Impute missing value and Logarithmic Transformation for ‘Fare’

```
X_test['Fare'].fillna(X_test['Fare'].median(), inplace=True)
```

```
X_train['Fare'] = X_train['Fare'].apply(lambda x: np.log1p(x))
X_test['Fare'] = X_test['Fare'].apply(lambda x: np.log1p(x))
```

iii. Scailing the data using StandardScaler

```python
# Concatenate X_train and X_test to ensure all categories are considered
X_concatenated = pd.concat([X_train, X_test], ignore_index=True)

# Define the columns to be one-hot encoded and the columns to be scaled
categorical_columns = ['Sex', 'Embarked', 'cabin_adv', 'name_title']
numeric_columns = [col for col in X_concatenated.columns if col not in categorical_columns]

# Create a column transformer to apply one-hot encoding to categorical columns
preprocessor = ColumnTransformer(
    transformers=[('cat', OneHotEncoder(), categorical_columns)],
    remainder='passthrough'
)

# Fit and transform the preprocessor on the concatenated data
X_encoded = preprocessor.fit_transform(X_concatenated)

# Split the data back into separate sets
X_train_encoded = X_encoded[:len(X_train)]
X_test_encoded = X_encoded[len(X_train):]

# Apply StandardScaler to the numeric columns
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_encoded)
X_test_scaled = scaler.transform(X_test_encoded)
```

- Build Model

```python
# Initialize the models
models = [
    ('Naive Bayes', GaussianNB()),
    ('Logistic Regression', LogisticRegression(random_state=1)),
    ('Decision Tree', DecisionTreeClassifier(random_state=1)),
    ('K Nearest Neighbor', KNeighborsClassifier()),
    ('Random Forest', RandomForestClassifier(random_state=1)),
    ('Support Vector Classifier', SVC(random_state=
    ('Xtreme Gradient Boosting', xgb.XGBClassifier(random_state=1)),
]

for name, model in models:
    cv_scores = cross_val_score(model, X_train_scaled, y_train, cv=5)
    print(f"{name} - Mean Accuracy: {cv_scores.mean() * 100:.2f}%")


# Perform 5-fold cross-validation and calculate multiple metrics
for name, model in models:
    cv_accuracy_scores = cross_val_score(model, X_train_scaled, y_train, cv=5, scoring='accuracy')
    cv_precision_scores = cross_val_score(model, X_train_scaled, y_train, cv=5, scoring='precision')
    cv_recall_scores = cross_val_score(model, X_train_scaled, y_train, cv=5, scoring='recall')
    cv_f1_scores = cross_val_score(model, X_train_scaled, y_train, cv=5, scoring='f1')

    print(f"\n{name} - Cross-Validation Metrics:")
    print(f"Accuracy: {cv_accuracy_scores.mean():.2f}")
    print(f"Precision: {cv_precision_scores.mean():.2f}")
    print(f"Recall: {cv_recall_scores.mean():.2f}")
    print(f"F1-Score: {cv_f1_scores.mean():.2f}")
```

(function) random_state: Any

- Tunning Model

```python
rf = RandomForestClassifier(random_state=1)
param_grid = {
    'n_estimators': [100, 500, 1000],
    'bootstrap': [True, False],
    'max_depth': [3, 5, 10, 20, 50, 75, 100, None],
    'max_features': ['auto', 'sqrt'],
    'min_samples_leaf': [1, 2, 4, 10],
    'min_samples_split': [2, 5, 10]
}

clf_rf_rnd = RandomizedSearchCV(rf, param_distributions=param_grid, n_iter=100, cv=5, verbose=True, n_jobs=-1)
best_clf_rf_rnd = clf_rf_rnd.fit(X_train_scaled, y_train)
print("Best parameters for Random Forest:", best_clf_rf_rnd.best_params_)
print("Best accuracy for Random Forest:", best_clf_rf_rnd.best_score_ * 100)
```

- Building Model Ensemble

```python
knn = KNeighborsClassifier()
svm = SVC(probability=True, random_state=1)
rf = RandomForestClassifier(random_state=1)

voting_clf = VotingClassifier(estimators=[('KNN', knn), ('SVM', svm), ('RF', rf)], voting='soft')
cv_scores_voting = cross_val_score(voting_clf, X_train_scaled, y_train, cv=5)
print(f"Voting Classifier - Mean Accuracy: {cv_scores_voting.mean() * 100:.2f}%")
```

- Further Ensemble Approaches

```python
lr = LogisticRegression(random_state=1)
xgb = xgb.XGBClassifier(random_state=1)

voting_clf_all = VotingClassifier(estimators=[('KNN', knn), ('SVM', svm), ('RF', rf), ('LR', lr), ('XGB', xgb)], voting='soft')
cv_scores_voting_all = cross_val_score(voting_clf_all, X_train_scaled, y_train, cv=5)
print(f"Voting Classifier (All Estimators) - Mean Accuracy: {cv_scores_voting_all.mean() * 100:.2f}%")
```

- Conclusion and Final Submission

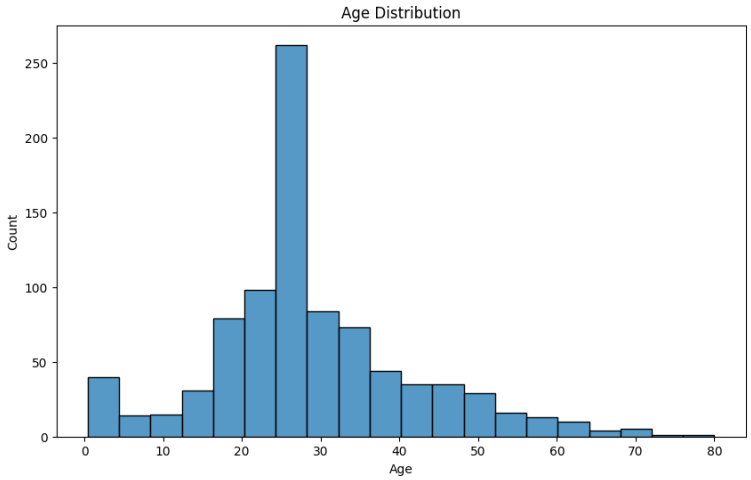
```
# Perform cross-validation and calculate the mean accuracy for the additional ensemble approaches
cv_scores_voting = cross_val_score(voting_clf, X_train_scaled, y_train, cv=5)
print("\nModel Additional Ensemble Approaches:")
print("Voting Classifier (KNN, SVM, RF):", cv_scores_voting.mean() * 100)
print("Voting Classifier (All Estimators):", cv_scores_voting_all.mean() * 100)
```

```
# Define and fit the imputer on the training data
imputer = SimpleImputer(strategy='median')
imputer.fit(X_train_scaled)

# Impute missing values in X_test_scaled using the same imputer as the training data
X_test_scaled_imputed = imputer.transform(X_test_scaled)

# Choose the best-performing model for final submission
best_model = voting_clf_all

# Train the best model on the entire training dataset
best_model.fit(X_train_scaled, y_train)

# Make predictions on the test data
predictions = best_model.predict(X_test_scaled_imputed)

# Create a submission DataFrame
submission_data = pd.DataFrame({'PassengerId': test_data['PassengerId'], 'Survived': predictions})

# Set the 'Survived' column to 0 for passengers who did not survive
submission_data.loc[submission_data['Survived'] == 0, 'Survived'] = 0

# Save the submission DataFrame to a CSV file for submission
submission_data.to_csv('submission.csv', index=False)

print("CSV file has been created successfully.")
```

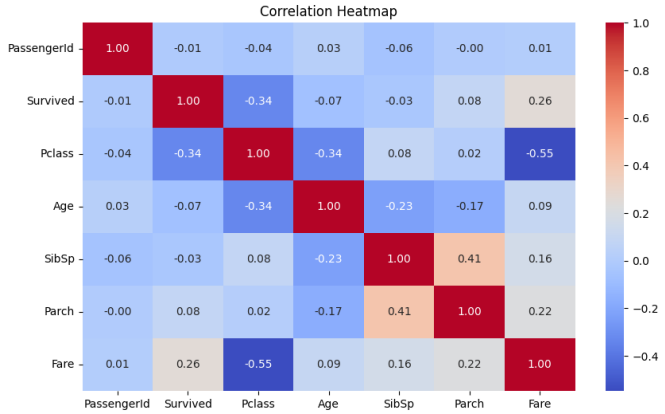
To see the 'submission.csv' file in more detail, click here

10.2 Additional Visualizations

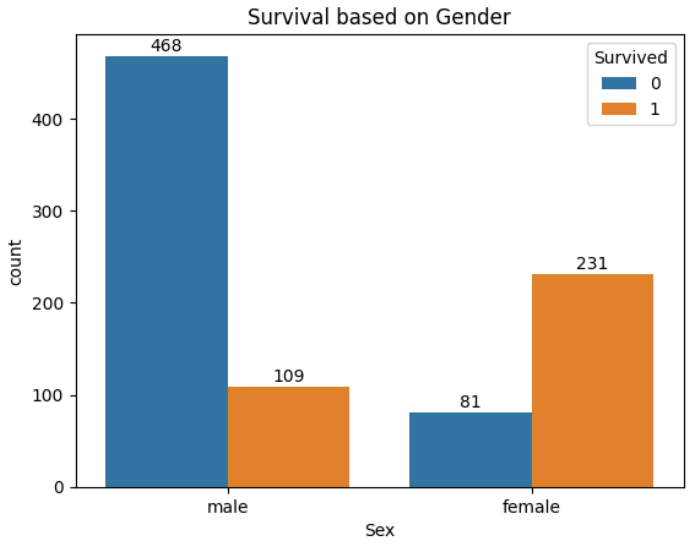
- Histogram



- Correlation Heatmap



- Bar chart for Survival based Gender



- Comparison of Mean Accuracy of Different Models (in percentage (%))