Artificial Intelligence CS361

Titanic survivor prediction using Random Forest

May 24, 2023

Team Members: Islam Abd-Elhady
Hussein Abd-Elkader
Ramez Nasser
Ahmed Alaa
Enas Ragab

Contents

Chapter I

Introduction/Executive Summary

The sinking of the RMS Titanic on April 15, 1912, remains one of the most infamous maritime disasters in history. The tragedy resulted in the loss of over 1,500 lives, prompting extensive investigations and raising questions about the factors that influenced the survival of passengers aboard the ship. In this project, we aim to build a machine-learning model, specifically the Random Forest algorithm, to predict the survival outcomes of passengers on the Titanic.

The goal of this project is to analyze the Titanic dataset, which contains information about a subset of the passengers, and develop a predictive model that can classify passengers as survivors or non-survivors based on various features. By leveraging the Random Forest algorithm and other machine learning techniques, we can uncover patterns and relationships in the data that may contribute to a better understanding of the factors that affected survival during the disaster.

The Titanic dataset consists of a range of features for each passenger, such as age, sex, passenger class, fare, and family relationships aboard the ship. These features offer valuable insights into the demographics and circumstances of the passengers and provide an opportunity to explore the impact of different factors on survival probabilities.

In summary, this project aims to utilize the Random Forest algorithm and other machine learning techniques to analyze the Titanic dataset, build a model for survival prediction, and gain insights into the factors that influenced passenger survival during the Titanic disaster.

Chapter II

Methodology

Description of Random Forest Algorithms

A Random Forest Algorithm is a supervised machine learning algorithm that is extremely popular and is used for Classification and Regression problems in Machine Learning. We know that a forest comprises numerous trees, and the more trees more it will be robust. Similarly, the greater the number of trees in a Random Forest Algorithm, the higher its accuracy and problem-solving ability. Random Forest is a classifier that contains several decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset. It is based on the concept of ensemble learning which is a process of combining multiple classifiers to solve a complex problem and improve the performance of the model.

The trees are trained using a process called bagging, which involves randomly sampling the data with replacement. Additionally, each tree is trained on a random subset of features, which helps to introduce diversity in the model. This randomness and diversity in the trees are what make Random Forest powerful.

To make a prediction using Random Forest, the algorithm combines the predictions of all the individual decision trees. For classification problems, the most common class predicted by the trees is chosen as the final prediction. For regression problems, the average or median of the predictions from the trees is taken as the final prediction.

Users have a lot of data and can train your models. Supervised learning further falls into two groups: classification and regression.

With supervised training, the training data contains the input and target values. The algorithm picks up a pattern that maps the input values to the output and uses this pattern to predict values in the future.

The following steps explain the working Random Forest Algorithm:

I. Select random samples from a given data or training set.

II. This algorithm will construct a decision tree for every training data.

III. Voting will take place by averaging the decision tree.

IV. Finally, select the most voted prediction result as the final prediction result.

Main Features of Random Forest Algorithms

I. Miscellany: Each tree has a unique attribute, variety and features concerning other trees. Not all trees are the same.

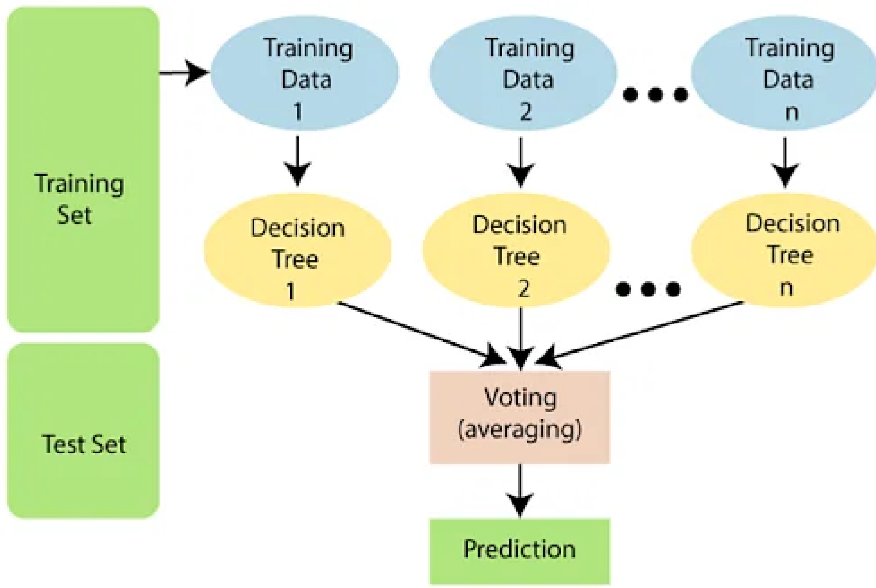


Figure II.1: Random Forest Algorithm Steps

II. Immune to the curse of dimensionality: Since a tree is a conceptual idea, it requires no features to be considered. Hence, the feature space is reduced.

III. Parallelization: We can fully use the CPU to build random forests since each tree is created autonomously from different data and features.

IV. Train-Test split: In a Random Forest, we don't have to differentiate the data for train and test because the decision tree never sees 30

V. Stability: The final result is based on Bagging, meaning the result is based on majority voting or average.

Random Forest pseudocode

I. Randomly select "k" features from total "m" features, where k $\ll$ m.

II. Among the "k" features, calculate the node "d" using the best split point.

III. Split the node into daughter nodes using the best split.

IV. Repeat steps 1 to 3 until "l" number of nodes has been reached.

V. Build the forest by repeating steps 1 to 4 for "n" number of times to create "n" number of trees.

The beginning of random forest algorithm starts with randomly selecting "k" features out of the total "m" features. In the image, you can observe that we are randomly taking features and observations. In the next stage, we are using the randomly selected "k" features to find the root node by using the best-split approach. In the next stage, We will be calculating the daughter nodes using the same best-split approach. Will the first 3 stages until we form the tree with a root node and the target as the leaf node. Finally, we repeat 1 to 4 stages to create "n" randomly created trees. These randomly created trees form a random forest.

Random forest prediction pseudocode

To perform prediction using the trained random forest algorithm uses the below pseudocode.

I. Takes the test features and uses the rules of each randomly created decision tree to predict the outcomes and stores the predicted outcome (target).

II. Calculate the votes for each predicted target.

III. Consider the highly voted predicted target as the final prediction from the random forest algorithm.

To perform the prediction using the trained random forest algorithm we need to pass the test features through the rules of each randomly created tree. Suppose let's say we formed 100 random decision trees to from the random forest. Each random forest will predict different targets (outcomes) for the same test feature. Then by considering each predicted target votes will be calculated. Suppose the 100 random decision trees are predicting some 3 unique targets x, y, z then the votes of x is nothing but out of 100 random decision trees how many trees prediction is x? Likewise for the other 2 targets (y, z). If x is getting high votes. Let's say out of 100 random decision trees 60 trees are predicting the target will be x. Then the final random forest returns the x as the predicted target. This concept of voting is known as majority voting.

Random Forest has several advantages that make it suitable for solving various problems

- Robustness to overfitting: Random Forest reduces the risk of overfitting compared to individual decision trees. Combining multiple trees and using random subsets of data and features, it helps to mitigate

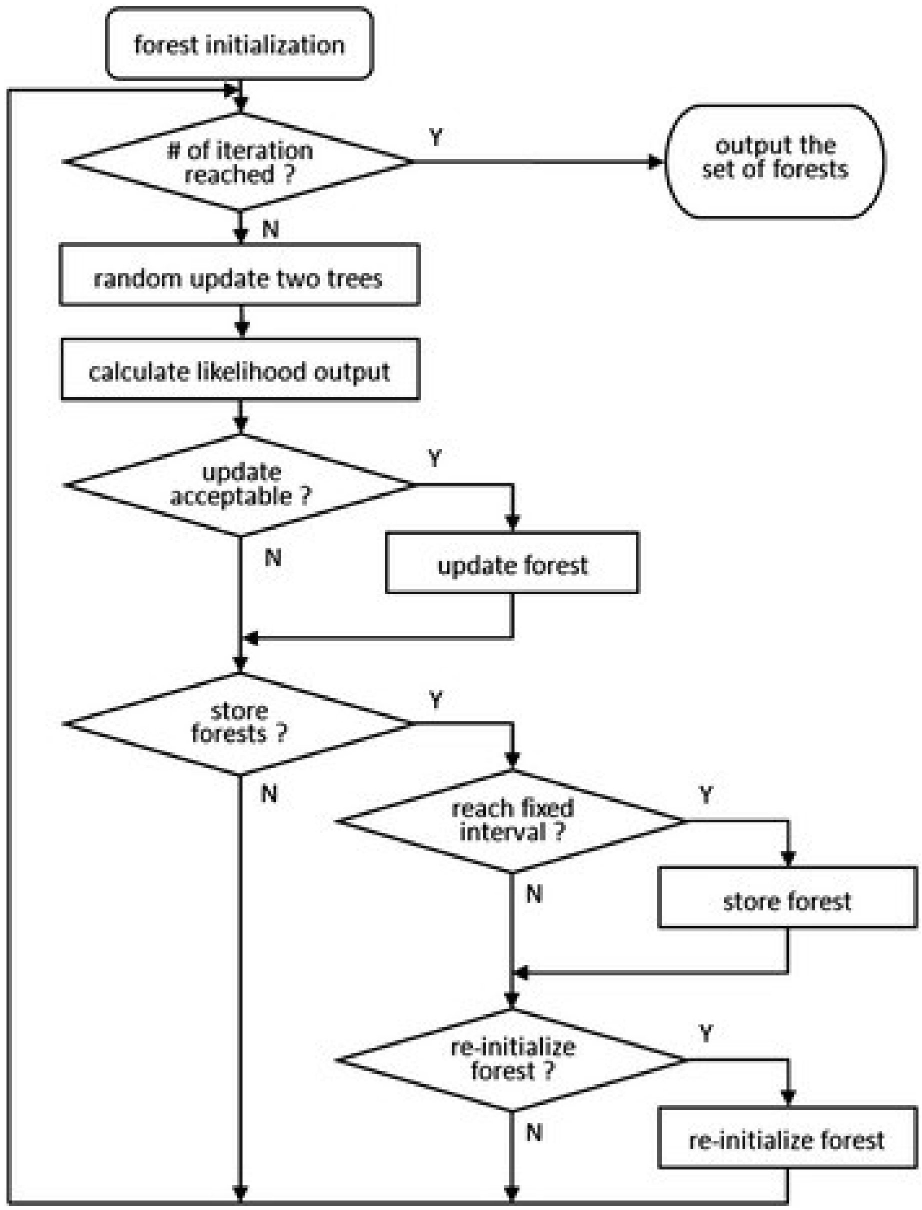


Figure II.2: Flowchart of Random Forest Algorithm the impact of outliers and noise in the data.

- Feature importance: Random Forest provides a measure of feature importance. By examining the average depth or information gained across all the trees, we can identify which features are the most informative for making predictions by estimating missing data.

- Handles high-dimensional data: Random Forest performs well even with a large number of features. It can effectively handle high-dimensional data and does not require feature selection or dimensionality reduction techniques in most cases.

- Out-of-bag (OOB) error estimation: During the training process, Random Forest uses a subset of data for each tree and leaves out a portion of the data. This out-of-bag data can be used to estimate the performance of the model without the need for additional cross-validation.

The time complexity of Random Forest primarily depends on two factors

- Building the trees: The time complexity of building a decision tree is O(n * m * log(n)), where n is the number of samples and m is the number of features. Random Forest builds multiple decision trees, so the overall time complexity is O(k * n * m * log(n)), where k is the number of trees.

- Making predictions: The time complexity of making a prediction using a decision tree is O(m), where m is the number of features. Random Forest combines the predictions of multiple trees, so the overall time complexity is O(k * m), where k is the number of trees.

Therefore, the overall time complexity of training a Random Forest is represented by O(k * n * m * log(n)), where k denotes the number of trees, n is the number of instances, and m represents the number of features. For making predictions, the time complexity is O(k * m).

The number of trees, k, is typically a hyperparameter that can be adjusted based on the specific problem being addressed. The time complexity can be further influenced by various implementation details, including the efficiency of data structures and the utilization of parallelization techniques.

Regarding space complexity, it can be expressed as O(k * depth of the tree). This signifies that the amount of memory required by Random Forest is proportional to the number of trees multiplied by the depth of each tree.

In practical scenarios, Random Forest has demonstrated computational efficiency across a wide range of problem sizes. It can effectively handle large datasets and high-dimensional feature spaces while delivering robust predictive performance. Consequently, Random Forest tends to outperform other algorithms in terms of speed and efficiency.

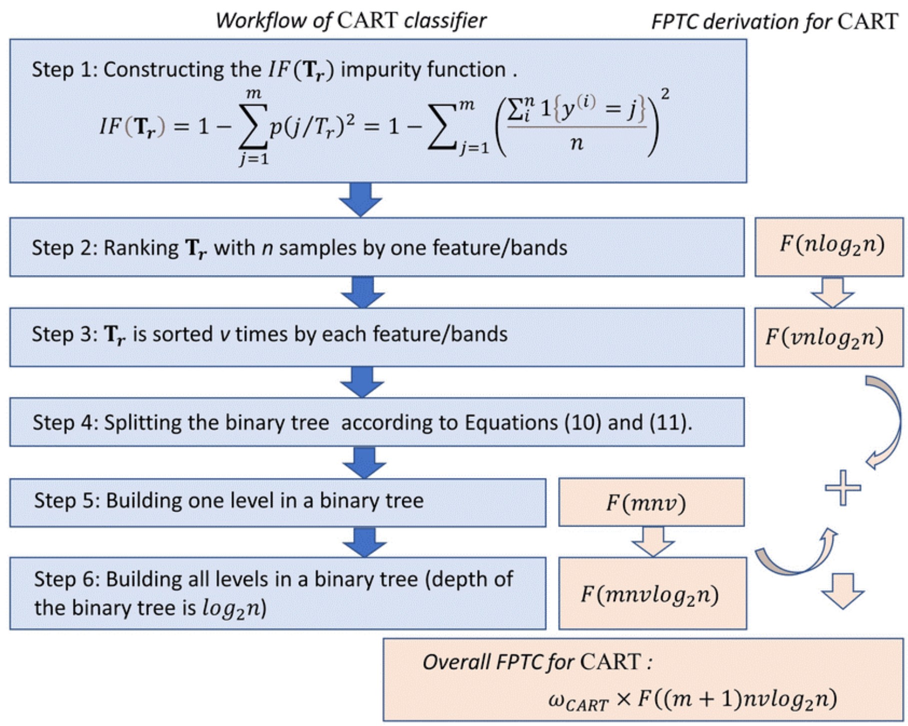


Figure II.3: Analyze the time complexity of Random Forest, Deriving full parameter time complexity (FPTC) for random forest (RF)

Chapter III

Experimental Simulation

III. Experimental Simulation:

1. Describe the programming languages and environments used in the project.

In the experimental simulation project, the programming language used is Python, which is a versatile and popular language known for its simplicity and readability. Python is widely used in scientific computing, data analysis, and machine learning due to its extensive libraries and frameworks.

Python offers a wide range of libraries and tools that are useful for simulations, such as NumPy for numerical computations, Pandas for data manipulation, Matplotlib for data visualization, and SciPy for scientific computing. These libraries provide efficient and convenient functions for handling complex mathematical operations, data processing, and plotting.

The project also utilizes the Colaboratory (Colab) environment, which is a cloud-based platform provided by Google. Colab allows users to write and execute Python code in a browser-based environment without the need for any local installation. It provides a Jupyter notebook interface that integrates code, text, and visualizations in a single document, making it convenient for prototyping, experimentation, and collaboration.

Colab offers several advantages for experimental simulation projects. It provides access to powerful hardware resources, including GPUs and TPUs, which can significantly accelerate computations. It also allows seamless integration with other Google services like Google Drive and Google Cloud, enabling easy access to data and resources. Additionally, Colab supports sharing and collaboration, making it suitable for team projects or sharing research findings.

Overall, Python and Colab provide a robust and accessible combination for developing and running experimental simulations, offering a wide range of libraries, tools, and resources to support the project's goals.

2. Discuss the details of programming the primary function and its procedures used to implement the introduced algorithms in Section II.

To implement the Random Forest Classifier algorithm in Python, you can utilize the scikit-learn library, which provides a comprehensive set of machine learning tools and algorithms. The primary function used for implementing the Random Forest Classifier is the `RandomForestClassifier` class from the `sklearn.ensemble` module.

Here's an overview of the primary function and its procedures used in implementing the Random Forest Classifier algorithm:

I. Import the necessary libraries:

```
from sklearn.ensemble import RandomForestClassifier
```

II. Create an instance of the RandomForestClassifier class:

```
clf = RandomForestClassifier()
```

III. Fit the classifier to the training data:

```
clf.fit(X_train, y_train)
```

Here, X_train represents the feature matrix of the training data, and y_train represents the corresponding target labels.

IV. Predict the class labels for the test data:

```
y_pred = clf.predict(X_test)
```

Here, X_test represents the feature matrix of the test data, and y_pred contains the predicted class labels.

V. Evaluate the performance of the classifier:

```
accuracy = clf.score(X_test, y_test)
```

The score method calculates the accuracy of the classifier by comparing the predicted labels (y_pred) with the true labels (y_test).

The Random Forest Classifier algorithm is an ensemble method that combines multiple decision trees to make predictions. Each decision tree is built using a random subset of the training data and a random subset of the features. Randomization helps in reducing overfitting and improving generalization.

The `RandomForestClassifier` class in scikit-learn provides various parameters that can be tuned to control the behavior of the algorithm. Some commonly used parameters include:

- `n_estimators`: The number of decision trees to be used in the random forest. - `max_depth`: The maximum depth of each decision tree. - `min_samples_split`: The minimum number of samples required to split an internal node. - `min_samples_leaf`: The minimum number of samples required to be at a leaf node. - `max_features`: The number of features to consider when looking for the best split.

These parameters can be set during the initialization of the `RandomForestClassifier` instance or modified after the instance is created.

By following these steps and customizing the parameters as per your requirements, you can effectively implement the Random Forest Classifier algorithm in Python using the scikit-learn library.

3. Explain the test cases used to test the programmed codes and how to set the program parameters and constants.

When testing the programmed code for the Random Forest Classifier, we use various test cases to ensure its correctness and effectiveness:

I. Large-scale dataset test: Random Forest Classifier is known for its scalability, so it is important to test the code with a large dataset to ensure it can handle large-scale problems efficiently. Use a dataset with a significant number of samples and features to assess the classifier's performance and its ability to handle computational requirements.

II. Cross-validation test: Cross-validation is a commonly used technique to evaluate the performance of machine learning models. Divide your dataset into multiple folds and perform cross-validation using the Random Forest Classifier. This helps in assessing the model's generalization capabilities and detecting any overfitting issues.

When setting the program parameters and constants in Random Forest, we consider the following:

- Number of estimators: This parameter (`n_estimators`) represents the number of decision trees in the forest. Higher values can improve performance but at the cost of increased computational resources. Start with a moderate value and adjust based on the size and complexity of your dataset.

Chapter IV

Results and Technical Discussion

Results and Technical Discussion:

1. Report the main program results and outputs.

When using the model with the Random Forest algorithm, the following results were produced:

The best accuracy rate: 81.6%

6.2.3 Ramdom Forest

```
[297] clf = RandomForestClassifier(n_estimators=13)
      scoring = 'accuracy'
      score = cross_val_score(clf, train_data, target, cv=k_fold, n_jobs=1, scoring=scoring)
      print(score)

[0.8        0.87640449 0.80898876 0.79775281 0.85393258 0.78651685
 0.80898876 0.80898876 0.79775281 0.82022472]
```

```
# Random Forest Score
round(np.mean(score)*100, 2)

81.6
```

```
[152] clf = RandomForestClassifier(n_estimators=13)
      scoring = 'accuracy'
      score = cross_val_score(clf, train_data, target, cv=k_fold, n_jobs=1, scoring=scoring)
      print(score)

      [0.82222222 0.80898876 0.7752809  0.80898876 0.85393258 0.83146067
       0.79775281 0.79775281 0.7752809  0.83146067]

[153] # Random Forest Score
      round(np.mean(score)*100, 2)

      81.03
```

```
[159] clf = RandomForestClassifier(n_estimators=13)
      scoring = 'accuracy'
      score = cross_val_score(clf, train_data, target, cv=k_fold, n_jobs=1, scoring=scoring)
      print(score)

      [0.81111111 0.87640449 0.75280899 0.80898876 0.82022472 0.80898876
       0.78651685 0.80898876 0.78651685 0.78651685]

      # Random Forest Score
      round(np.mean(score)*100, 2)

      80.47
```

```
[163] clf = RandomForestClassifier(n_estimators=13)
      scoring = 'accuracy'
      score = cross_val_score(clf, train_data, target, cv=k_fold, n_jobs=1, scoring=scoring)
      print(score)

      [0.78888889 0.84269663 0.79775281 0.78651685 0.84269663 0.78651685
       0.80898876 0.84269663 0.79775281 0.82022472]

      # Random Forest Score
      round(np.mean(score)*100, 2)

      81.15
```

2. Test/Evaluation experimental procedure and analysis of results.

We performed the test process of the model and obtained the following data:

```
submission.head(15)
```

	PassengerId	Survived
0	892	0
1	893	0
2	894	1
3	895	1
4	896	1
5	897	0
6	898	0
7	899	0
8	900	1
9	901	0
10	902	0
11	903	0
12	904	1
13	905	0
14	906	1

3. Discuss the main results and their quality.

The model produced an accuracy rate of 81.6%, and this accuracy depends on the features and inputs in the dataset. These results are approximate, but they are among the best results among most algorithms. Additionally, when using more than one algorithm such as Support Vector Machine, Decision Tree, and Naive Bayes, this accuracy was the best.

Chapter V

References

I. SimpliLearn

II. Dataaspirant

III. ResearchGate

IV. Kaggle

V. Secondary Analysis of Electronic Health Records, From Chapter 27 Signal Processing: False Alarm Reduction

VI. Mukesh ChapagainTitanic Solution: A Beginner's Guide

VII. How to score 0.8134 in Titanic Kaggle Challenge

VIII. Titanic: factors to survive

IX. Titanic Survivors Dataset and Data Wrangling

Appendix A

Project Source Codes

In this section, you will find the source codes related to the project. These source codes encompass various aspects and components of the project, providing insights into the implementation details and functionality.

The source codes are available for download and exploration. They can be accessed through the following link:

Source Codes Link

Feel free to explore and utilize these source codes for reference, further development, or any related purposes.

Titanic: Machine Learning from Disaster

Predict survival on the Titanic

- Defining the problem statement
- Collecting the data
- Exploratory data analysis
- Feature engineering
- Modelling
- Testing

1. Defining the problem statement

Complete the analysis of what sorts of people were likely to survive.
In particular, we ask you to apply the tools of machine learning to predict which passengers survived the Titanic tragedy.

```
from IPython.display import Image
Image(url=
"https://static1.squarespace.com/static/5006453fe4b09ef2252ba068/5095e
abce4b06cb305058603/5095eabce4b02d37bef4c24c/
1352002236895/100_anniversary_titanic_sinking_by_esai8mellows-
d4xbme8.jpg")
```

```
<IPython.core.display.Image object>
```

2. Collecting the data

training data set and testing data set are given by Kaggle you can download from
my github https://github.com/minsuk-heo/kaggle-titanic/tree/master
or you can download from kaggle directly kaggle

load train, test dataset using Pandas

```
import pandas as pd

train = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')
```

3. Exploratory data analysis

Printing first 5 rows of the train dataset.

```
train.head(80)
```

```
    PassengerId  Survived  Pclass  \
0             1         0       3
1             2         1       1
2             3         1       3
```

```
3             4         1       1
4             5         0       3
..          ...       ...     ...
75           76         0       3
76           77         0       3
77           78         0       3
78           79         1       2
79           80         1       3

                                                 Name     Sex    Age
SibSp  \
0                             Braund, Mr. Owen Harris    male  22.00
1
1   Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.00
1
2                              Heikkinen, Miss. Laina  female  26.00
0
3        Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.00
1
4                            Allen, Mr. William Henry    male  35.00
0
..                                                ...     ...    ...
...
75                            Moen, Mr. Sigurd Hansen    male  25.00
0
76                                  Staneff, Mr. Ivan    male    NaN
0
77                           Moutal, Mr. Rahamin Haim    male    NaN
0
78                      Caldwell, Master. Alden Gates    male   0.83
0
79                           Dowdell, Miss. Elizabeth  female  30.00
0

    Parch            Ticket     Fare  Cabin Embarked
0       0         A/5 21171   7.2500    NaN        S
1       0          PC 17599  71.2833    C85        C
2       0  STON/O2. 3101282   7.9250    NaN        S
3       0            113803  53.1000   C123        S
4       0            373450   8.0500    NaN        S
..    ...               ...      ...    ...      ...
75      0            348123   7.6500  F G73        S
76      0            349208   7.8958    NaN        S
77      0            374746   8.0500    NaN        S
78      2            248738  29.0000    NaN        S
79      0            364516  12.4750    NaN        S

[80 rows x 12 columns]
```

Data Dictionary

- Survived: 0 = No, 1 = Yes

- pclass: Ticket class 1 = 1st, 2 = 2nd, 3 = 3rd

- sibsp: # of siblings / spouses aboard the Titanic

- parch: # of parents / children aboard the Titanic

- ticket: Ticket number
- cabin: Cabin number
- embarked: Port of Embarkation C = Cherbourg, Q = Queenstown, S = Southampton

Total rows and columns

We can see that there are 891 rows and 12 columns in our training dataset.

```
test.head()
```

```
   PassengerId  Pclass                                          Name
Sex  \
0          892       3                              Kelly, Mr. James
male
1          893       3              Wilkes, Mrs. James (Ellen Needs)
female
2          894       2                     Myles, Mr. Thomas Francis
male
3          895       3                              Wirz, Mr. Albert
male
4          896       3  Hirvonen, Mrs. Alexander (Helga E Lindqvist)
female

    Age  SibSp  Parch   Ticket     Fare Cabin Embarked
0  34.5      0      0   330911   7.8292   NaN        Q
1  47.0      1      0   363272   7.0000   NaN        S
2  62.0      0      0   240276   9.6875   NaN        Q
3  27.0      0      0   315154   8.6625   NaN        S
4  22.0      1      1  3101298  12.2875   NaN        S
```

```
train.shape
```

```
(891, 12)
```

```
test.shape
```

```
(418, 11)
```

```
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
```

```
test.info()
```

```
<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: 36.0+ KB
```

We can see that *Age* value is missing for many rows.

Out of 891 rows, the *Age* value is present only in 714 rows.

Similarly, *Cabin* values are also missing in many rows. Only 204 out of 891 rows have *Cabin* values.

```
train.isnull().sum()
```

```
PassengerId      0
Survived         0
```

```
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

```
test.isnull().sum()
```

```
PassengerId      0
Pclass           0
Name             0
Sex              0
Age             86
SibSp            0
Parch            0
Ticket           0
Fare             1
Cabin          327
Embarked         0
dtype: int64
```

There are 177 rows with missing *Age*, 687 rows with missing *Cabin* and 2 rows with missing *Embarked* information.

import python lib for visualization

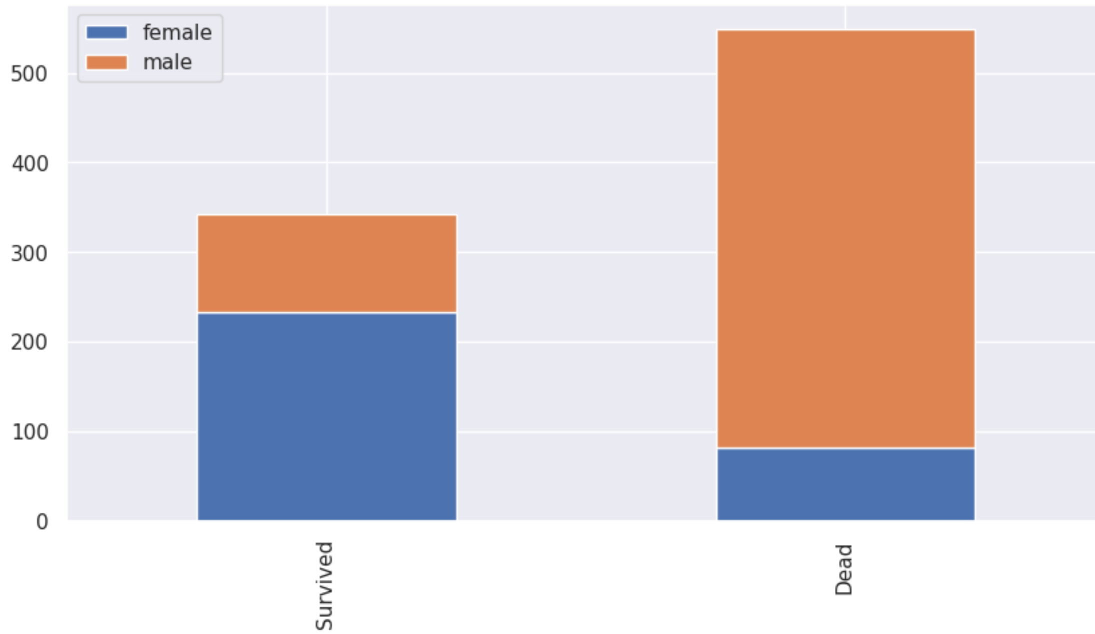
```
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
sns.set() # setting seaborn default for plots
```

Bar Chart for Categorical Features

- Pclass
- Sex
- SibSp (# of siblings and spouse)
- Parch (# of parents and children)
- Embarked
- Cabin

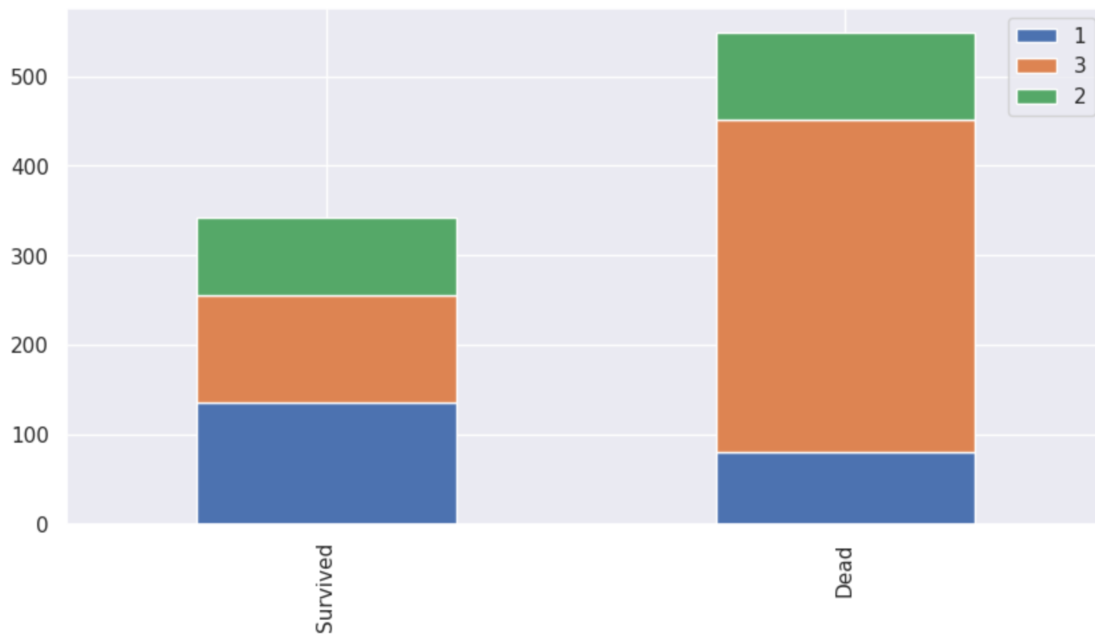
```
def bar_chart(feature):
    survived = train[train['Survived']==1][feature].value_counts()
    dead = train[train['Survived']==0][feature].value_counts()
    df = pd.DataFrame([survived,dead])
    df.index = ['Survived','Dead']
    df.plot(kind='bar',stacked=True, figsize=(10,5))
```

```python
bar_chart('Sex')
```



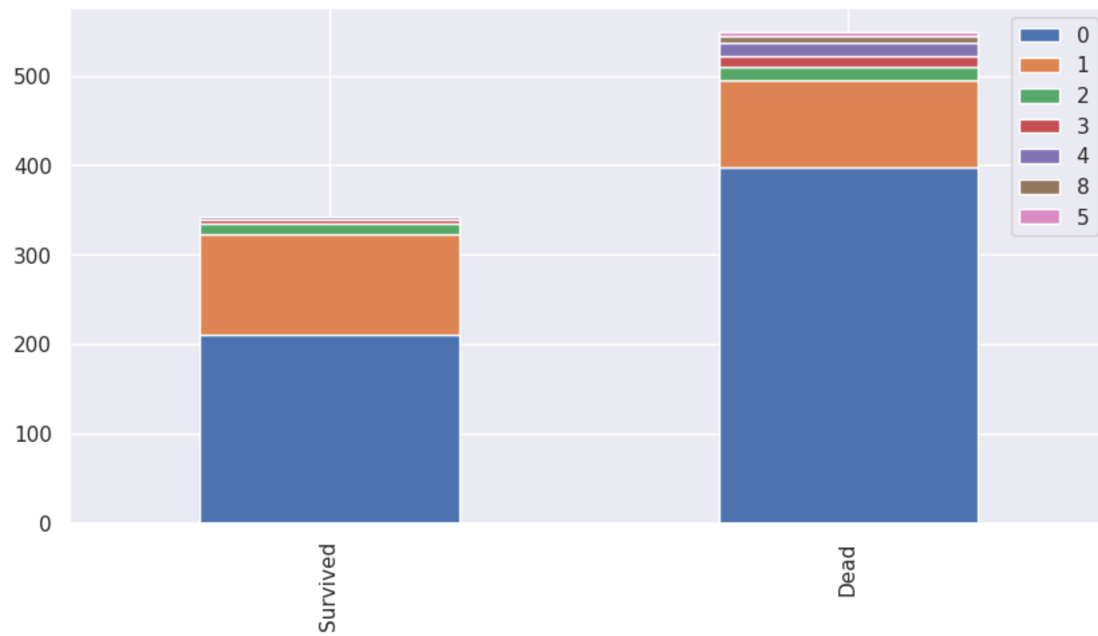
The Chart confirms **Women** more likely survivied than **Men**

```python
bar_chart('Pclass')
```



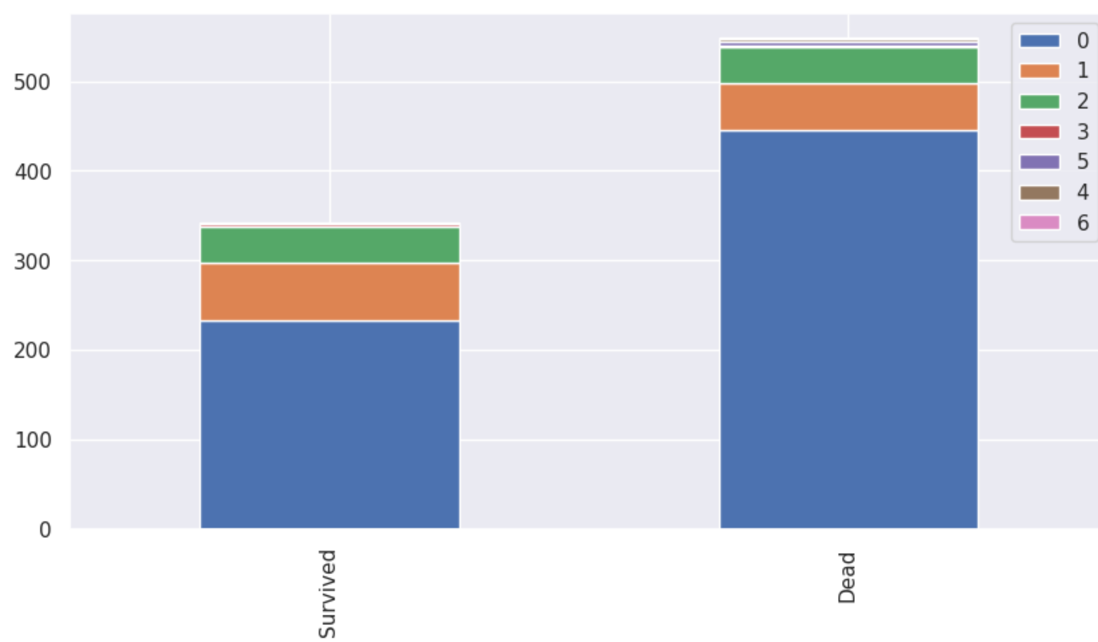
The Chart confirms **1st class** more likely survivied than **other classes**
The Chart confirms **3rd class** more likely dead than **other classes**

```python
bar_chart('SibSp')
```



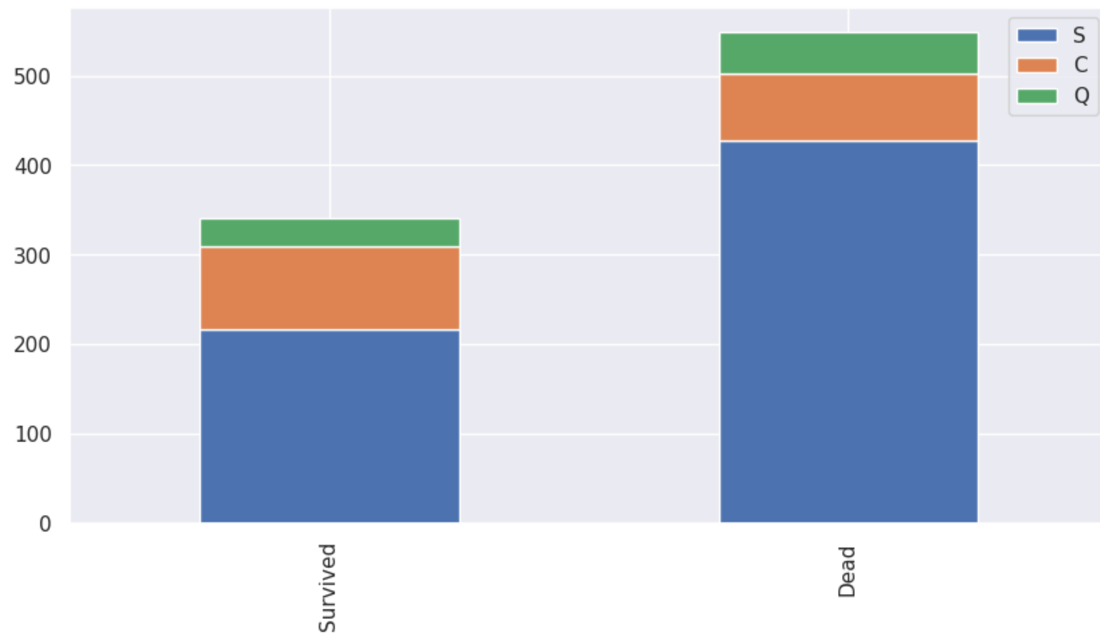
The Chart confirms **a person aboarded with more than 2 siblings or spouse** more likely survived
The Chart confirms ** a person aboarded without siblings or spouse** more likely dead

```
bar_chart('Parch')
```



The Chart confirms **a person aboarded with more than 2 parents or children** more likely survived
The Chart confirms ** a person aboarded alone** more likely dead

```
bar_chart('Embarked')
```



The Chart confirms **a person aboarded from C** slightly more likely survived
The Chart confirms **a person aboarded from Q** more likely dead
The Chart confirms **a person aboarded from S** more likely dead

4. Feature engineering

Feature engineering is the process of using domain knowledge of the data to create features (**feature vectors**) that make machine learning algorithms work.

feature vector is an n-dimensional vector of numerical features that represent some object. Many algorithms in machine learning require a numerical representation of objects, since such representations facilitate processing and statistical analysis.

```
train.head()
```

```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name     Sex   Age
SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0
1
1  Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0
1
2                             Heikkinen, Miss. Laina  female  26.0
0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0
```

```
1
4                            Allen, Mr. William Henry    male  35.0
0

   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
3      0            113803  53.1000  C123        S
4      0            373450   8.0500   NaN        S
```

4.1 how titanic sank?

sank from the bow of the ship where third class rooms located
conclusion, Pclass is key feature for classifier

```
Image(url=
"https://static1.squarespace.com/static/5006453fe4b09ef2252ba068/t/
5090b249e4b047ba54dfd258/1351660113175/TItanic-Survival-
Infographic.jpg?format=1500w")
```

```
<IPython.core.display.Image object>
```

```
train.head(10)
```

```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3
5            6         0       3
6            7         0       1
7            8         0       3
8            9         1       3
9           10         1       2

                                                Name     Sex   Age
SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0
1
1  Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0
1
2                             Heikkinen, Miss. Laina  female  26.0
0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0
1
4                           Allen, Mr. William Henry    male  35.0
0
5                                   Moran, Mr. James    male   NaN
0
```

```
6                             McCarthy, Mr. Timothy J    male  54.0
0
7                      Palsson, Master. Gosta Leonard    male   2.0
3
8  Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)  female  27.0
0
9                Nasser, Mrs. Nicholas (Adele Achem)  female  14.0
1

   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
3      0            113803  53.1000  C123        S
4      0            373450   8.0500   NaN        S
5      0            330877   8.4583   NaN        Q
6      0             17463  51.8625   E46        S
7      1            349909  21.0750   NaN        S
8      2            347742  11.1333   NaN        S
9      0            237736  30.0708   NaN        C
```

4.2 Name

```
train_test_data = [train, test] # combining train and test dataset

for dataset in train_test_data:
    dataset['Title'] = dataset['Name'].str.extract(' ([A-Za-z]+)\.',
expand=False)
```

```
train['Title'].value_counts()
```

```
Mr          517
Miss        182
Mrs         125
Master       40
Dr            7
Rev           6
Mlle          2
Major         2
Col           2
Countess      1
Capt          1
Ms            1
Sir           1
Lady          1
Mme           1
Don           1
Jonkheer      1
Name: Title, dtype: int64
```

```
test['Title'].value_counts()
```

```
Mr        240
Miss       78
Mrs        72
Master     21
Col         2
Rev         2
Ms          1
Dr          1
Dona        1
Name: Title, dtype: int64
```

Title map

Mr : 0
Miss : 1
Mrs: 2
Others: 3

```
title_mapping = {"Mr": 0, "Miss": 1, "Mrs": 2,
                 "Master": 3, "Dr": 3, "Rev": 3, "Col": 3, "Major": 3,
"Mlle": 3,"Countess": 3,
                 "Ms": 3, "Lady": 3, "Jonkheer": 3, "Don": 3, "Dona" :
3, "Mme": 3,"Capt": 3,"Sir": 3 }
for dataset in train_test_data:
    dataset['Title'] = dataset['Title'].map(title_mapping)
```

```
train.head()
```

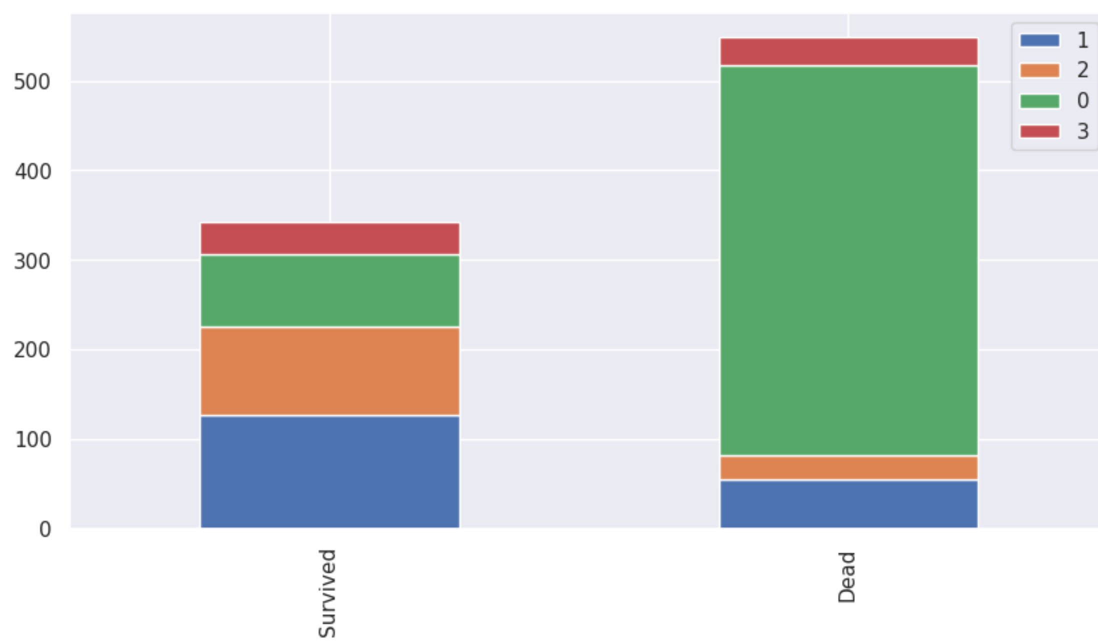
```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name     Sex   Age
SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0
1
1  Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0
1
2                             Heikkinen, Miss. Laina  female  26.0
0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0
1
4                           Allen, Mr. William Henry    male  35.0
0

   Parch            Ticket     Fare Cabin Embarked  Title
0      0         A/5 21171   7.2500   NaN        S      0
1      0          PC 17599  71.2833   C85        C      2
```

```
2      0  STON/O2. 3101282   7.9250   NaN        S      1
3      0            113803  53.1000  C123        S      2
4      0            373450   8.0500   NaN        S      0
```

```
test.head()
```

```
   PassengerId  Pclass                                          Name
Sex  \
0          892       3                              Kelly, Mr. James
male
1          893       3              Wilkes, Mrs. James (Ellen Needs)
female
2          894       2                     Myles, Mr. Thomas Francis
male
3          895       3                              Wirz, Mr. Albert
male
4          896       3  Hirvonen, Mrs. Alexander (Helga E Lindqvist)
female

    Age  SibSp  Parch   Ticket     Fare Cabin Embarked  Title
0  34.5      0      0   330911   7.8292   NaN        Q      0
1  47.0      1      0   363272   7.0000   NaN        S      2
2  62.0      0      0   240276   9.6875   NaN        Q      0
3  27.0      0      0   315154   8.6625   NaN        S      0
4  22.0      1      1  3101298  12.2875   NaN        S      2
```

```
bar_chart('Title')
```




```
# delete unnecessary feature from dataset
train.drop('Name', axis=1, inplace=True)
test.drop('Name', axis=1, inplace=True)
```

```
train.head()
```

```
   PassengerId  Survived  Pclass     Sex   Age  SibSp  Parch  \
0            1         0       3    male  22.0      1      0
1            2         1       1  female  38.0      1      0
2            3         1       3  female  26.0      0      0
3            4         1       1  female  35.0      1      0
4            5         0       3    male  35.0      0      0

             Ticket     Fare Cabin Embarked  Title
0         A/5 21171   7.2500   NaN        S      0
1          PC 17599  71.2833   C85        C      2
2  STON/O2. 3101282   7.9250   NaN        S      1
3            113803  53.1000  C123        S      2
4            373450   8.0500   NaN        S      0
```

```
test.head()
```

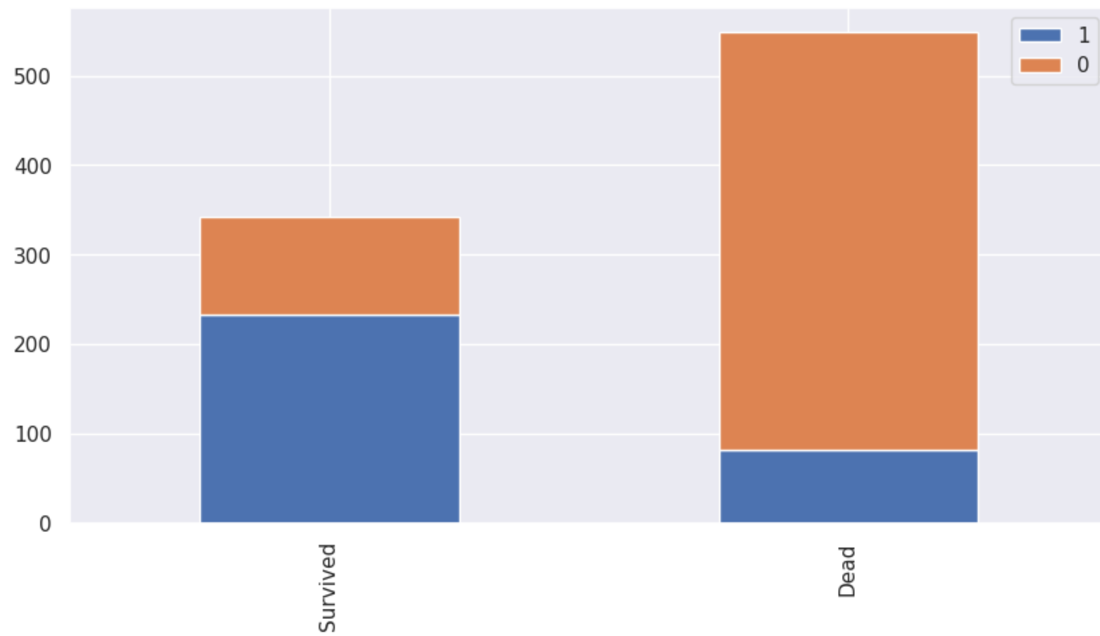
```
   PassengerId  Pclass     Sex   Age  SibSp  Parch   Ticket     Fare
Cabin  \
0          892       3    male  34.5      0      0   330911   7.8292
NaN
1          893       3  female  47.0      1      0   363272   7.0000
NaN
2          894       2    male  62.0      0      0   240276   9.6875
NaN
3          895       3    male  27.0      0      0   315154   8.6625
NaN
4          896       3  female  22.0      1      1  3101298  12.2875
NaN

  Embarked  Title
0        Q      0
1        S      2
2        Q      0
3        S      0
4        S      2
```

4.3 Sex

male: 0 female: 1

```
sex_mapping = {"male": 0, "female": 1}
for dataset in train_test_data:
    dataset['Sex'] = dataset['Sex'].map(sex_mapping)
```

```
bar_chart('Sex')
```



4.4 Age

4.4.1 some age is missing

Let's use Title's median age for missing Age

```
train.head(100)
```

```
    PassengerId  Survived  Pclass  Sex   Age  SibSp  Parch
Ticket  \
0             1         0       3    0  22.0      1      0         A/5
21171
1             2         1       1    1  38.0      1      0          PC
17599
2             3         1       3    1  26.0      0      0  STON/O2.
3101282
3             4         1       1    1  35.0      1      0
113803
4             5         0       3    0  35.0      0      0
373450
..          ...       ...     ...  ...   ...    ...    ...
...
95           96         0       3    0   NaN      0      0
374910
96           97         0       1    0  71.0      0      0          PC
17754
97           98         1       1    0  23.0      0      1          PC
17759
98           99         1       2    1  34.0      0      1
231919
99          100         0       2    0  34.0      1      0
```

```
244367
```

```
        Fare    Cabin Embarked  Title
0    7.2500      NaN        S      0
1   71.2833      C85        C      2
2    7.9250      NaN        S      1
3   53.1000     C123        S      2
4    8.0500      NaN        S      0
..      ...      ...      ...    ...
95   8.0500      NaN        S      0
96  34.6542       A5        C      0
97  63.3583  D10 D12        C      0
98  23.0000      NaN        S      2
99  26.0000      NaN        S      0

[100 rows x 12 columns]
```

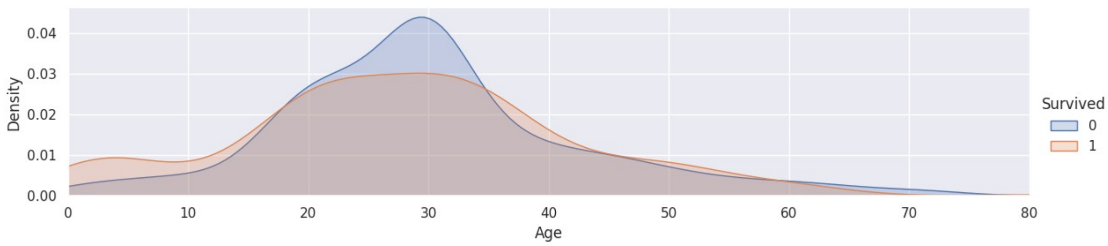
```
# fill missing age with median age for each title (Mr, Mrs, Miss,
Others)
train["Age"].fillna(train.groupby("Title")["Age"].transform("median"),
inplace=True)
test["Age"].fillna(test.groupby("Title")["Age"].transform("median"),
inplace=True)

train.head(30)
train.groupby("Title")["Age"].transform("median")
```

```
0      30.0
1      35.0
2      21.0
3      35.0
4      30.0
       ...
886     9.0
887    21.0
888    21.0
889    30.0
890    30.0
Name: Age, Length: 891, dtype: float64
```

```
facet = sns.FacetGrid(train, hue="Survived",aspect=4)
facet.map(sns.kdeplot,'Age',shade= True)
facet.set(xlim=(0, train['Age'].max()))
facet.add_legend()

plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/seaborn/axisgrid.py:848:
FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
```

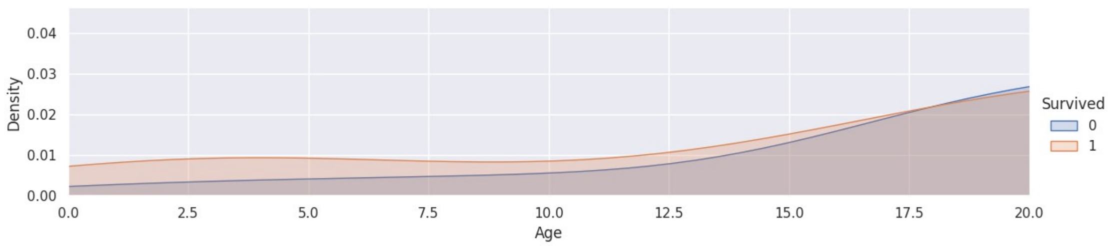
```
This will become an error in seaborn v0.14.0; please update your code.

  func(*plot_args, **plot_kwargs)
/usr/local/lib/python3.10/dist-packages/seaborn/axisgrid.py:848:
FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  func(*plot_args, **plot_kwargs)
```




```
facet = sns.FacetGrid(train, hue="Survived",aspect=4)
facet.map(sns.kdeplot,'Age',shade= True)
facet.set(xlim=(0, train['Age'].max()))
facet.add_legend()
plt.xlim(0, 20)
```

```
/usr/local/lib/python3.10/dist-packages/seaborn/axisgrid.py:848:
FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  func(*plot_args, **plot_kwargs)
/usr/local/lib/python3.10/dist-packages/seaborn/axisgrid.py:848:
FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  func(*plot_args, **plot_kwargs)
```

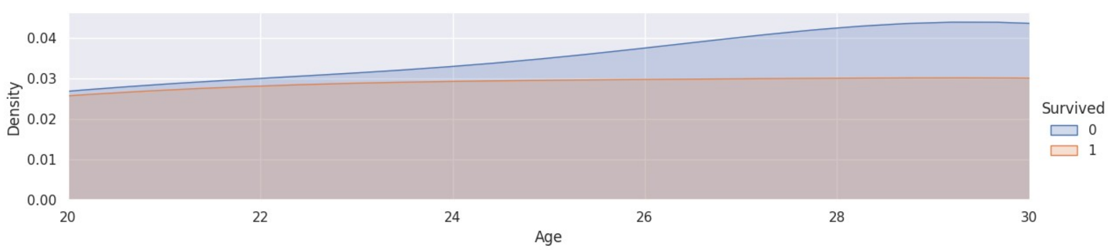
```
(0.0, 20.0)
```



```
facet = sns.FacetGrid(train, hue="Survived",aspect=4)
facet.map(sns.kdeplot,'Age',shade= True)
facet.set(xlim=(0, train['Age'].max()))
facet.add_legend()
plt.xlim(20, 30)
```

```
/usr/local/lib/python3.10/dist-packages/seaborn/axisgrid.py:848:
FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  func(*plot_args, **plot_kwargs)
/usr/local/lib/python3.10/dist-packages/seaborn/axisgrid.py:848:
FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  func(*plot_args, **plot_kwargs)
```

```
(20.0, 30.0)
```

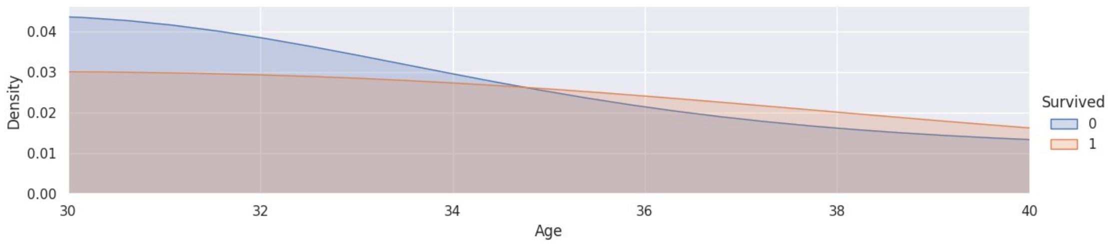



```
facet = sns.FacetGrid(train, hue="Survived",aspect=4)
facet.map(sns.kdeplot,'Age',shade= True)
facet.set(xlim=(0, train['Age'].max()))
facet.add_legend()
plt.xlim(30, 40)
```

```
/usr/local/lib/python3.10/dist-packages/seaborn/axisgrid.py:848:
FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  func(*plot_args, **plot_kwargs)
/usr/local/lib/python3.10/dist-packages/seaborn/axisgrid.py:848:
FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.
```

```
  func(*plot_args, **plot_kwargs)
```

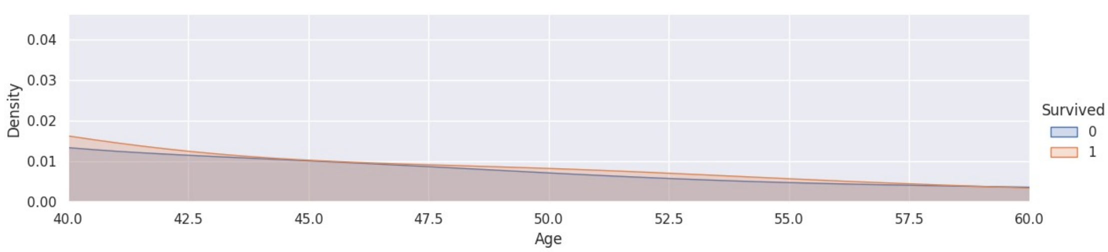
```
(30.0, 40.0)
```




```
facet = sns.FacetGrid(train, hue="Survived",aspect=4)
facet.map(sns.kdeplot,'Age',shade= True)
facet.set(xlim=(0, train['Age'].max()))
facet.add_legend()
plt.xlim(40, 60)
```

```
/usr/local/lib/python3.10/dist-packages/seaborn/axisgrid.py:848:
FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  func(*plot_args, **plot_kwargs)
/usr/local/lib/python3.10/dist-packages/seaborn/axisgrid.py:848:
FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  func(*plot_args, **plot_kwargs)
```

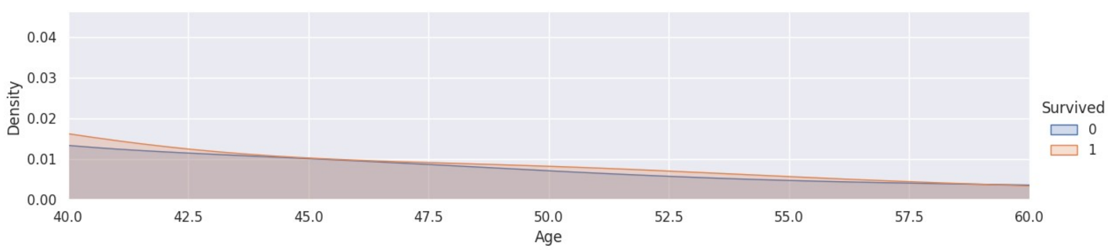
```
(40.0, 60.0)
```




```
facet = sns.FacetGrid(train, hue="Survived",aspect=4)
facet.map(sns.kdeplot,'Age',shade= True)
facet.set(xlim=(0, train['Age'].max()))
facet.add_legend()
plt.xlim(40, 60)
```

```
/usr/local/lib/python3.10/dist-packages/seaborn/axisgrid.py:848:
FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  func(*plot_args, **plot_kwargs)
/usr/local/lib/python3.10/dist-packages/seaborn/axisgrid.py:848:
FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  func(*plot_args, **plot_kwargs)
```

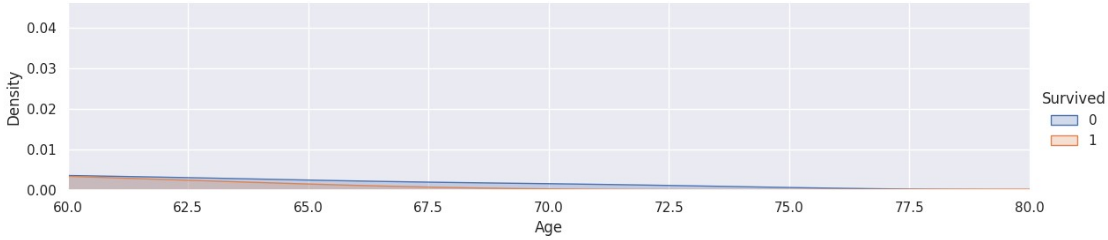
```
(40.0, 60.0)
```




```
facet = sns.FacetGrid(train, hue="Survived",aspect=4)
facet.map(sns.kdeplot,'Age',shade= True)
facet.set(xlim=(0, train['Age'].max()))
facet.add_legend()
plt.xlim(60)
```

```
/usr/local/lib/python3.10/dist-packages/seaborn/axisgrid.py:848:
FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  func(*plot_args, **plot_kwargs)
/usr/local/lib/python3.10/dist-packages/seaborn/axisgrid.py:848:
FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  func(*plot_args, **plot_kwargs)
```

```
(60.0, 80.0)
```




```
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   Sex          891 non-null    int64  
 4   Age          891 non-null    float64
 5   SibSp        891 non-null    int64  
 6   Parch        891 non-null    int64  
 7   Ticket       891 non-null    object 
 8   Fare         891 non-null    float64
 9   Cabin        204 non-null    object 
 10  Embarked     889 non-null    object 
 11  Title        891 non-null    int64  
dtypes: float64(2), int64(7), object(3)
memory usage: 83.7+ KB
```

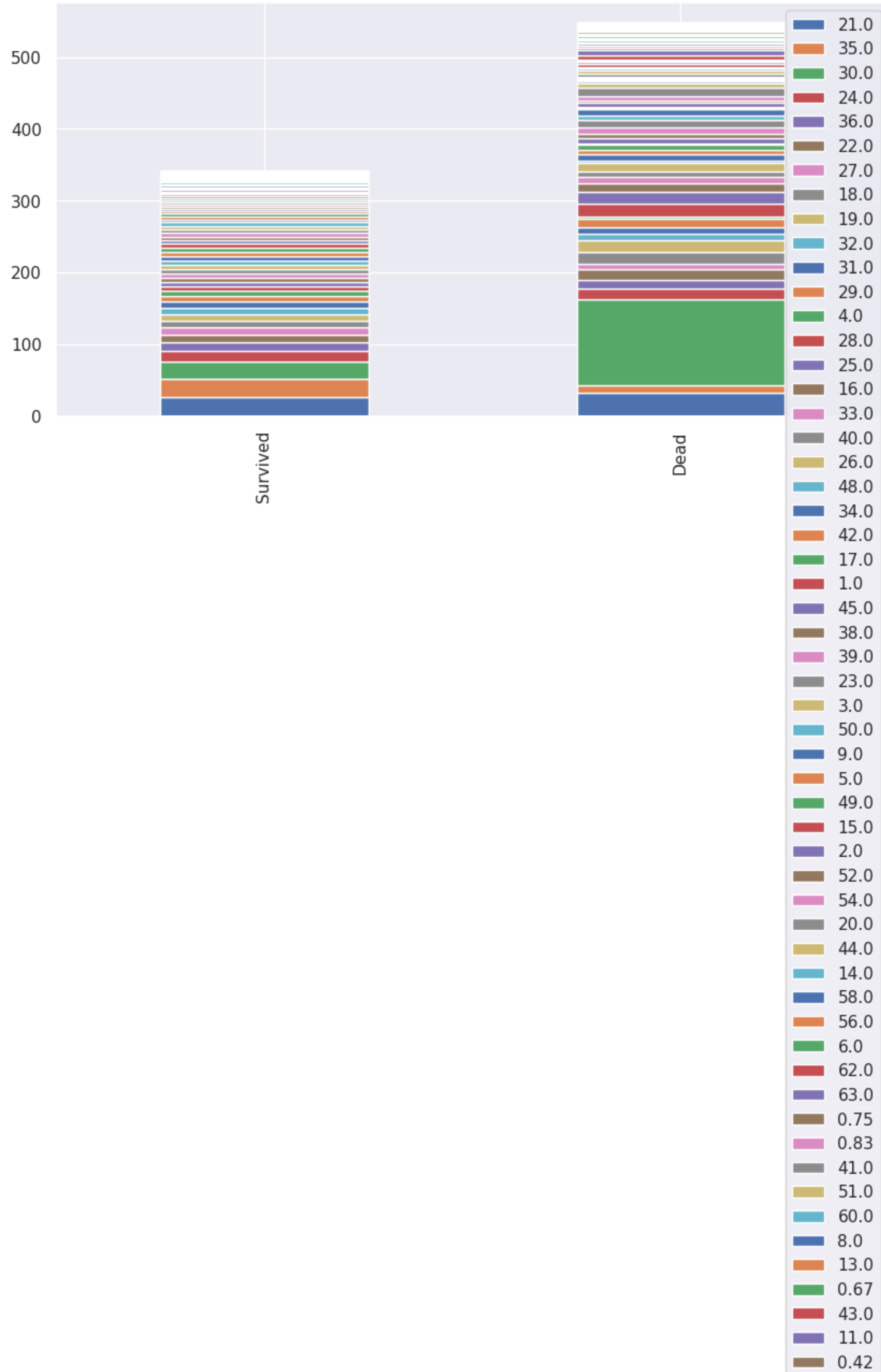
```
test.info()
```

```
<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   Sex          418 non-null    int64  
 3   Age          418 non-null    float64
 4   SibSp        418 non-null    int64  
 5   Parch        418 non-null    int64  
 6   Ticket       418 non-null    object 
 7   Fare         417 non-null    float64
 8   Cabin        91 non-null     object 
 9   Embarked     418 non-null    object 
 10  Title        418 non-null    int64  
dtypes: float64(2), int64(6), object(3)
memory usage: 36.0+ KB
```

```
train.head()
```

```
   PassengerId  Survived  Pclass  Sex   Age  SibSp  Parch            
Ticket  \
0            1         0       3    0  22.0      1      0         A/5 
21171   
1            2         1       1    1  38.0      1      0          PC 
17599   
2            3         1       3    1  26.0      0      0  STON/O2. 
3101282   
3            4         1       1    1  35.0      1      0             
113803   
4            5         0       3    0  35.0      0      0             
373450   

      Fare Cabin Embarked  Title  
0   7.2500   NaN        S      0  
1  71.2833   C85        C      2  
2   7.9250   NaN        S      1  
3  53.1000  C123        S      2  
4   8.0500   NaN        S      0  
```

```
bar_chart('Age')
```



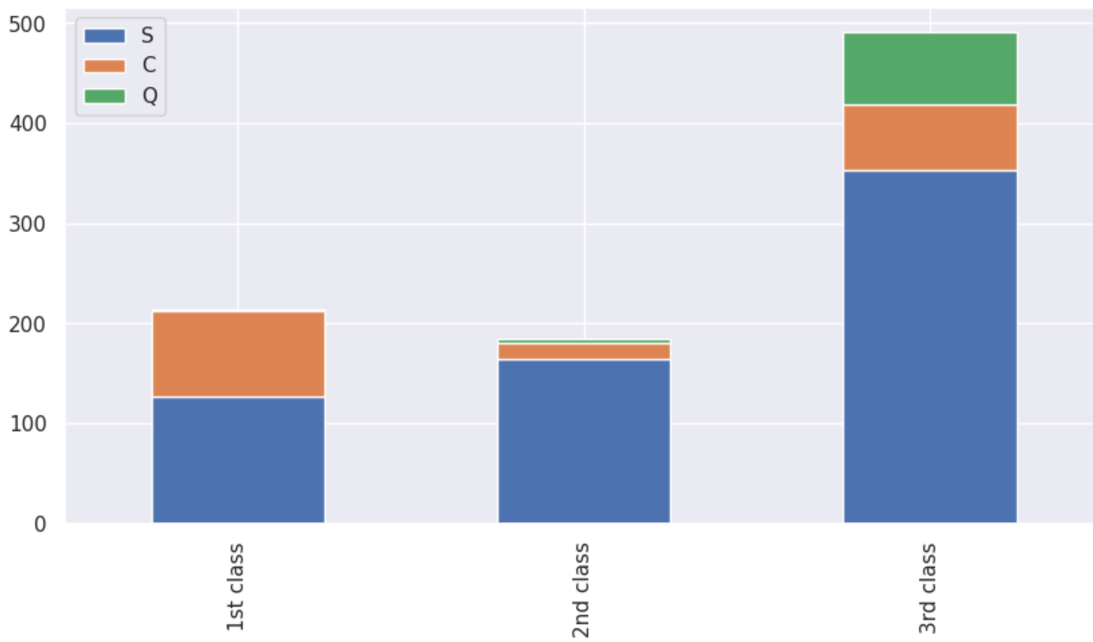
0
100
200
300
400
500
Survived
Dead
21.0
35.0
30.0
24.0
36.0
22.0
27.0
18.0
19.0
32.0
31.0
29.0
4.0
28.0
25.0
16.0
33.0
40.0
26.0
48.0
34.0
42.0
17.0
1.0
45.0
38.0
39.0
23.0
3.0
50.0
9.0
5.0
49.0
15.0
2.0
52.0
54.0
20.0
44.0
14.0
58.0
56.0
6.0
62.0
63.0
0.75
0.83
41.0
51.0
60.0
8.0
13.0
0.67
43.0
11.0
0.42

4.5 Embarked

4.5.1 filling missing values

```
Pclass1 = train[train['Pclass']==1]['Embarked'].value_counts()
Pclass2 = train[train['Pclass']==2]['Embarked'].value_counts()
Pclass3 = train[train['Pclass']==3]['Embarked'].value_counts()
df = pd.DataFrame([Pclass1, Pclass2, Pclass3])
df.index = ['1st class','2nd class', '3rd class']
df.plot(kind='bar',stacked=True, figsize=(10,5))
```

```
<Axes: >
```



more than 50% of 1st class are from S embark
more than 50% of 2nd class are from S embark
more than 50% of 3rd class are from S embark

fill out missing embark with S embark

```
for dataset in train_test_data:
    dataset['Embarked'] = dataset['Embarked'].fillna('S')
```

```
train.head()
```

```
   PassengerId  Survived  Pclass  Sex   Age  SibSp  Parch  
Ticket  \
0            1         0       3    0  22.0      1      0         A/5 
21171   
1            2         1       1    1  38.0      1      0          PC 
17599   
2            3         1       3    1  26.0      0      0  STON/O2. 
3101282   
```

```
3            4         1       1    1  35.0      1      0
113803
4            5         0       3    0  35.0      0      0
373450

      Fare Cabin Embarked  Title
0   7.2500   NaN        S      0
1  71.2833   C85        C      2
2   7.9250   NaN        S      1
3  53.1000  C123        S      2
4   8.0500   NaN        S      0
```

```
embarked_mapping = {"S": 0, "C": 1, "Q": 2}
for dataset in train_test_data:
    dataset['Embarked'] = dataset['Embarked'].map(embarked_mapping)
```

4.6 Fare

```
# fill missing Fare with median fare for each Pclass
train["Fare"].fillna(train.groupby("Pclass")
["Fare"].transform("median"), inplace=True)
test["Fare"].fillna(test.groupby("Pclass")
["Fare"].transform("median"), inplace=True)
train.head(50)
```

```
    PassengerId  Survived  Pclass  Sex   Age  SibSp  Parch
Ticket  \
0             1         0       3    0  22.0      1      0         A/5
21171
1             2         1       1    1  38.0      1      0          PC
17599
2             3         1       3    1  26.0      0      0  STON/O2.
3101282
3             4         1       1    1  35.0      1      0
113803
4             5         0       3    0  35.0      0      0
373450
5             6         0       3    0  30.0      0      0
330877
6             7         0       1    0  54.0      0      0
17463
7             8         0       3    0   2.0      3      1
349909
8             9         1       3    1  27.0      0      2
347742
9            10         1       2    1  14.0      1      0
237736
10           11         1       3    1   4.0      1      1
PP 9549
11           12         1       1    1  58.0      0      0
113783
12           13         0       3    0  20.0      0      0
```

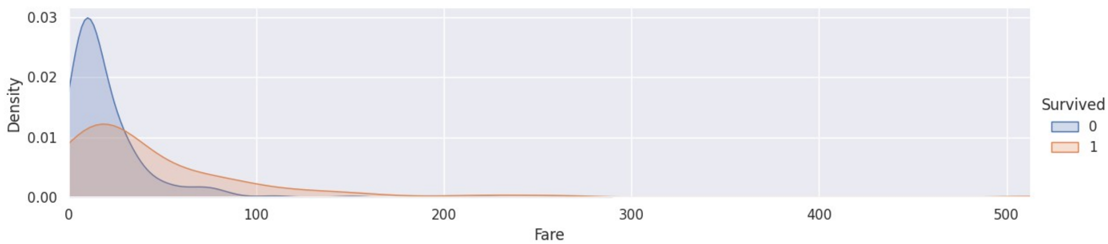
								A/5. 2151
13	14	0	3	0	39.0	1	5	347082
14	15	0	3	1	14.0	0	0	350406
15	16	1	2	1	55.0	0	0	248706
16	17	0	3	0	2.0	4	1	382652
17	18	1	2	0	30.0	0	0	244373
18	19	0	3	1	31.0	1	0	345763
19	20	1	3	1	35.0	0	0	2649
20	21	0	2	0	35.0	0	0	239865
21	22	1	2	0	34.0	0	0	248698
22	23	1	3	1	15.0	0	0	330923
23	24	1	1	0	28.0	0	0	113788
24	25	0	3	1	8.0	3	1	349909
25	26	1	3	1	38.0	1	5	347077
26	27	0	3	0	30.0	0	0	2631
27	28	0	1	0	19.0	3	2	19950
28	29	1	3	1	21.0	0	0	330959
29	30	0	3	0	30.0	0	0	349216
30	31	0	1	0	40.0	0	0	PC 17601
31	32	1	1	1	35.0	1	0	PC 17569
32	33	1	3	1	21.0	0	0	335677
33	34	0	2	0	66.0	0	0	C.A. 24579
34	35	0	1	0	28.0	1	0	PC 17604
35	36	0	1	0	42.0	1	0	113789
36	37	1	3	0	30.0	0	0	2677
37	38	0	3	0	21.0	0	0	

```
A./5. 2152
38          39          0        3    1  18.0      2      0
345764
39          40          1        3    1  14.0      1      0
2651
40          41          0        3    1  40.0      1      0
7546
41          42          0        2    1  27.0      1      0
11668
42          43          0        3    0  30.0      0      0
349253
43          44          1        2    1   3.0      1      2
SC/Paris 2123
44          45          1        3    1  19.0      0      0
330958
45          46          0        3    0  30.0      0      0   S.C./A.4.
23567
46          47          0        3    0  30.0      1      0
370371
47          48          1        3    1  21.0      0      0
14311
48          49          0        3    0  30.0      2      0
2662
49          50          0        3    1  18.0      1      0
349237

        Fare        Cabin  Embarked  Title
0     7.2500          NaN         0      0
1    71.2833          C85         1      2
2     7.9250          NaN         0      1
3    53.1000         C123         0      2
4     8.0500          NaN         0      0
5     8.4583          NaN         2      0
6    51.8625          E46         0      0
7    21.0750          NaN         0      3
8    11.1333          NaN         0      2
9    30.0708          NaN         1      2
10   16.7000           G6         0      1
11   26.5500         C103         0      1
12    8.0500          NaN         0      0
13   31.2750          NaN         0      0
14    7.8542          NaN         0      1
15   16.0000          NaN         0      2
16   29.1250          NaN         2      3
17   13.0000          NaN         0      0
18   18.0000          NaN         0      2
19    7.2250          NaN         1      2
20   26.0000          NaN         0      0
21   13.0000          D56         0      0
22    8.0292          NaN         2      1
```

```
23   35.5000           A6         0      0
24   21.0750          NaN         0      1
25   31.3875          NaN         0      2
26    7.2250          NaN         1      0
27  263.0000  C23 C25 C27         0      0
28    7.8792          NaN         2      1
29    7.8958          NaN         0      0
30   27.7208          NaN         1      3
31  146.5208          B78         1      2
32    7.7500          NaN         2      1
33   10.5000          NaN         0      0
34   82.1708          NaN         1      0
35   52.0000          NaN         0      0
36    7.2292          NaN         1      0
37    8.0500          NaN         0      0
38   18.0000          NaN         0      1
39   11.2417          NaN         1      1
40    9.4750          NaN         0      2
41   21.0000          NaN         0      2
42    7.8958          NaN         1      0
43   41.5792          NaN         1      1
44    7.8792          NaN         2      1
45    8.0500          NaN         0      0
46   15.5000          NaN         2      0
47    7.7500          NaN         2      1
48   21.6792          NaN         1      0
49   17.8000          NaN         0      2
```

```python
facet = sns.FacetGrid(train, hue="Survived",aspect=4)
facet.map(sns.kdeplot,'Fare',shade= True)
facet.set(xlim=(0, train['Fare'].max()))
facet.add_legend()

plt.show()
```

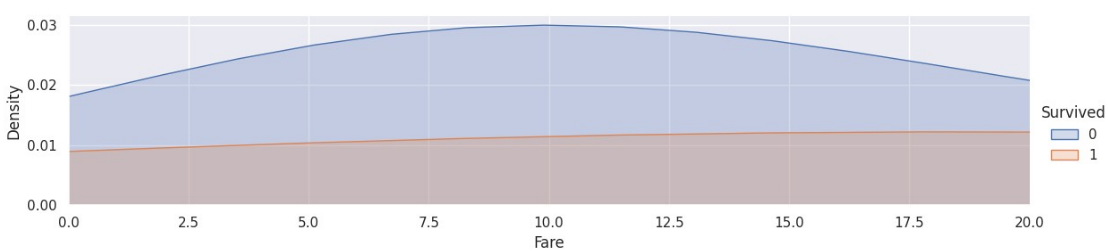
```
/usr/local/lib/python3.10/dist-packages/seaborn/axisgrid.py:848:
FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  func(*plot_args, **plot_kwargs)
/usr/local/lib/python3.10/dist-packages/seaborn/axisgrid.py:848:
FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  func(*plot_args, **plot_kwargs)
```




```
facet = sns.FacetGrid(train, hue="Survived",aspect=4)
facet.map(sns.kdeplot,'Fare',shade= True)
facet.set(xlim=(0, train['Fare'].max()))
facet.add_legend()
plt.xlim(0, 20)
```

```
/usr/local/lib/python3.10/dist-packages/seaborn/axisgrid.py:848:
FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  func(*plot_args, **plot_kwargs)
/usr/local/lib/python3.10/dist-packages/seaborn/axisgrid.py:848:
FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  func(*plot_args, **plot_kwargs)
```

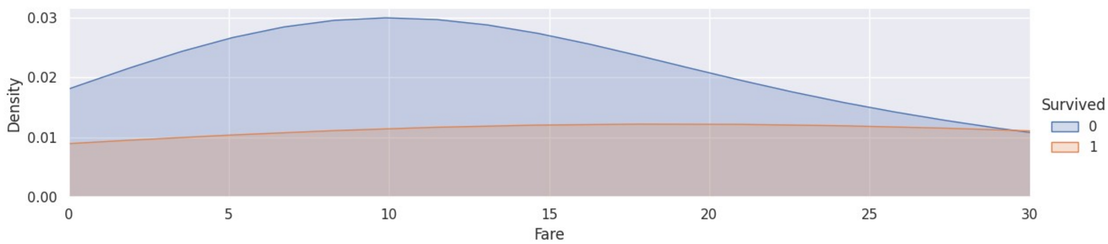
```
(0.0, 20.0)
```




```
facet = sns.FacetGrid(train, hue="Survived",aspect=4)
facet.map(sns.kdeplot,'Fare',shade= True)
facet.set(xlim=(0, train['Fare'].max()))
facet.add_legend()
plt.xlim(0, 30)
```

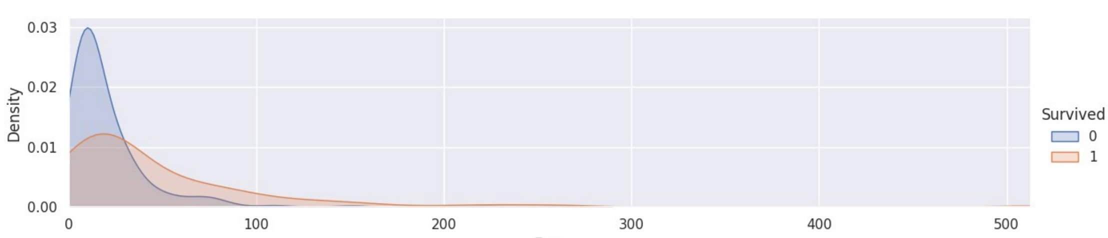
```
/usr/local/lib/python3.10/dist-packages/seaborn/axisgrid.py:848:
FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.
```

```
  func(*plot_args, **plot_kwargs)
/usr/local/lib/python3.10/dist-packages/seaborn/axisgrid.py:848:
FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  func(*plot_args, **plot_kwargs)

(0.0, 30.0)
```




```
facet = sns.FacetGrid(train, hue="Survived",aspect=4)
facet.map(sns.kdeplot,'Fare',shade= True)
facet.set(xlim=(0, train['Fare'].max()))
facet.add_legend()
plt.xlim(0)
```

```
/usr/local/lib/python3.10/dist-packages/seaborn/axisgrid.py:848:
FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  func(*plot_args, **plot_kwargs)
/usr/local/lib/python3.10/dist-packages/seaborn/axisgrid.py:848:
FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  func(*plot_args, **plot_kwargs)

(0.0, 512.3292)
```



```
train.head()
```

```
   PassengerId  Survived  Pclass  Sex   Age  SibSp  Parch            
Ticket  \
0            1         0       3    0  22.0      1      0         A/5 
21171   
1            2         1       1    1  38.0      1      0          PC 
17599   
2            3         1       3    1  26.0      0      0  STON/O2. 
3101282   
3            4         1       1    1  35.0      1      0            
113803   
4            5         0       3    0  35.0      0      0            
373450   

      Fare Cabin  Embarked  Title  
0   7.2500   NaN         0      0  
1  71.2833   C85         1      2  
2   7.9250   NaN         0      1  
3  53.1000  C123         0      2  
4   8.0500   NaN         0      0  
```

4.7 Cabin

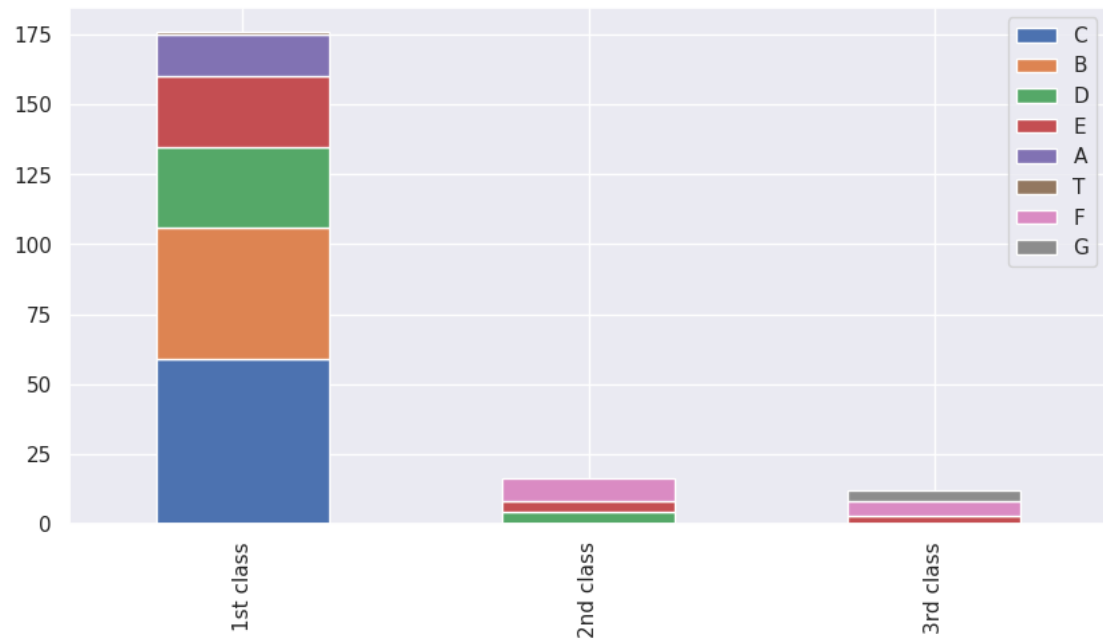
```
train.Cabin.value_counts()
```

```
B96 B98        4
G6             4
C23 C25 C27    4
C22 C26        3
F33            3
              ..
E34            1
C7             1
C54            1
E36            1
C148           1
Name: Cabin, Length: 147, dtype: int64
```

```
for dataset in train_test_data:
    dataset['Cabin'] = dataset['Cabin'].str[:1]
```

```
Pclass1 = train[train['Pclass']==1]['Cabin'].value_counts()
Pclass2 = train[train['Pclass']==2]['Cabin'].value_counts()
Pclass3 = train[train['Pclass']==3]['Cabin'].value_counts()
df = pd.DataFrame([Pclass1, Pclass2, Pclass3])
df.index = ['1st class','2nd class', '3rd class']
df.plot(kind='bar',stacked=True, figsize=(10,5))
```

```
<Axes: >
```




```
cabin_mapping = {"A": 0, "B": 0.4, "C": 0.8, "D": 1.2, "E": 1.6, "F":
2, "G": 2.4, "T": 2.8}
for dataset in train_test_data:
    dataset['Cabin'] = dataset['Cabin'].map(cabin_mapping)

# fill missing Fare with median fare for each Pclass
train["Cabin"].fillna(train.groupby("Pclass")
["Cabin"].transform("median"), inplace=True)
test["Cabin"].fillna(test.groupby("Pclass")
["Cabin"].transform("median"), inplace=True)
```

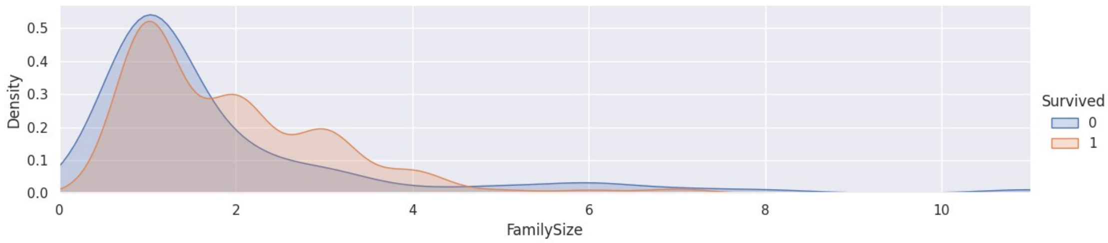
4.8 FamilySize

```
train["FamilySize"] = train["SibSp"] + train["Parch"] + 1
test["FamilySize"] = test["SibSp"] + test["Parch"] + 1

facet = sns.FacetGrid(train, hue="Survived",aspect=4)
facet.map(sns.kdeplot,'FamilySize',shade= True)
facet.set(xlim=(0, train['FamilySize'].max()))
facet.add_legend()
plt.xlim(0)
```

```
/usr/local/lib/python3.10/dist-packages/seaborn/axisgrid.py:848:
FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  func(*plot_args, **plot_kwargs)
/usr/local/lib/python3.10/dist-packages/seaborn/axisgrid.py:848:
FutureWarning:
```

```
`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  func(*plot_args, **plot_kwargs)
```

```
(0.0, 11.0)
```




```
family_mapping = {1: 0, 2: 0.4, 3: 0.8, 4: 1.2, 5: 1.6, 6: 2, 7: 2.4,
8: 2.8, 9: 3.2, 10: 3.6, 11: 4}
for dataset in train_test_data:
    dataset['FamilySize'] = dataset['FamilySize'].map(family_mapping)

train.head()
```

```
   PassengerId  Survived  Pclass  Sex   Age  SibSp  Parch
Ticket  \
0            1         0       3    0  22.0      1      0         A/5
21171
1            2         1       1    1  38.0      1      0          PC
17599
2            3         1       3    1  26.0      0      0  STON/O2.
3101282
3            4         1       1    1  35.0      1      0
113803
4            5         0       3    0  35.0      0      0
373450

      Fare  Cabin  Embarked  Title  FamilySize
0   7.2500    2.0         0      0         0.4
1  71.2833    0.8         1      2         0.4
2   7.9250    2.0         0      1         0.0
3  53.1000    0.8         0      2         0.4
4   8.0500    2.0         0      0         0.0
```

```
train.head()
```

```
   PassengerId  Survived  Pclass  Sex   Age  SibSp  Parch
Ticket  \
0            1         0       3    0  22.0      1      0         A/5
21171
1            2         1       1    1  38.0      1      0          PC
17599
2            3         1       3    1  26.0      0      0  STON/O2.
```

```
3101282
3            4          1       1    1  35.0      1      0
113803
4            5          0       3    0  35.0      0      0
373450

      Fare  Cabin  Embarked  Title  FamilySize
0   7.2500    2.0         0      0         0.4
1  71.2833    0.8         1      2         0.4
2   7.9250    2.0         0      1         0.0
3  53.1000    0.8         0      2         0.4
4   8.0500    2.0         0      0         0.0
```

```
features_drop = ['Ticket', 'SibSp', 'Parch']
train = train.drop(features_drop, axis=1)
test = test.drop(features_drop, axis=1)
train = train.drop(['PassengerId'], axis=1)
```

```
train_data = train.drop('Survived', axis=1)
target = train['Survived']

train_data.shape, target.shape
```

```
((891, 8), (891,))
```

```
train_data.head(10)
```

```
   Pclass  Sex   Age     Fare  Cabin  Embarked  Title  FamilySize
0       3    0  22.0   7.2500    2.0         0      0         0.4
1       1    1  38.0  71.2833    0.8         1      2         0.4
2       3    1  26.0   7.9250    2.0         0      1         0.0
3       1    1  35.0  53.1000    0.8         0      2         0.4
4       3    0  35.0   8.0500    2.0         0      0         0.0
5       3    0  30.0   8.4583    2.0         2      0         0.0
6       1    0  54.0  51.8625    1.6         0      0         0.0
7       3    0   2.0  21.0750    2.0         0      3         1.6
8       3    1  27.0  11.1333    2.0         0      2         0.8
9       2    1  14.0  30.0708    1.8         1      2         0.4
```

5. Modelling

```
# Importing Classifier Modules
from sklearn.ensemble import RandomForestClassifier

import numpy as np
```

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 9 columns):
 #   Column      Non-Null Count  Dtype
```

```
---  ------      -------------  -----
 0   Survived    891 non-null    int64
 1   Pclass      891 non-null    int64
 2   Sex         891 non-null    int64
 3   Age         891 non-null    float64
 4   Fare        891 non-null    float64
 5   Cabin       891 non-null    float64
 6   Embarked    891 non-null    int64
 7   Title       891 non-null    int64
 8   FamilySize  891 non-null    float64
dtypes: float64(4), int64(5)
memory usage: 62.8 KB
```

6.2 Cross Validation (K-fold)

```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
k_fold = KFold(n_splits=10, shuffle=True, random_state=0)
```

6.2.3 Ramdom Forest

```
clf = RandomForestClassifier(n_estimators=13)
scoring = 'accuracy'
score = cross_val_score(clf, train_data, target, cv=k_fold, n_jobs=1,
scoring=scoring)
print(score)
```

```
[0.81111111 0.85393258 0.78651685 0.7752809  0.85393258 0.83146067
 0.79775281 0.7752809  0.76404494 0.80898876]
```

```
# Random Forest Score
round(np.mean(score)*100, 2)
```

```
80.58
```

7. Testing

```
clf = RandomForestClassifier(n_estimators=13)
clf.fit(train_data, target)

test_data = test.drop("PassengerId", axis=1).copy()
prediction = clf.predict(test_data)

submission = pd.DataFrame({
        "PassengerId": test["PassengerId"],
        "Survived": prediction
    })

submission.to_csv('gender_submission.csv', index=False)

submission = pd.read_csv('gender_submission.csv')
submission.head()
```

	PassengerId	Survived
0	892	0
1	893	0
2	894	0
3	895	0
4	896	1

Appendix B

Project Team Plan and Achievement

Project Team Plan for "Titanic Survivor Prediction using Random Forest":

- Define Project Objectives: - Develop a classification model to predict whether a Titanic passenger survived or not. - Create a robust and accurate prediction model using the Random Forest algorithm.
- Team Formation: - Identify team members with expertise in data analysis, machine learning, and programming. - Assign roles and responsibilities, such as data preprocessing, feature engineering, model development, and evaluation.
- Data Collection and Exploration: - Gather the Titanic dataset containing passenger information (e.g., age, gender, ticket class, etc.). - Perform exploratory data analysis (EDA) to gain insights into the dataset and understand the relationships between variables.
- Data Preprocessing and Feature Engineering: - Han-

dle missing data by applying appropriate techniques, such as imputation or removal. - Encode categorical variables into numerical representations for model compatibility. - Perform feature engineering to create new features or transform existing ones to enhance the model's predictive power.

- Model Development: - Implement the Random Forest algorithm using suitable libraries or frameworks (e.g., scikit-learn in Python). - Split the dataset into training and testing sets to evaluate the model's performance. - Tune hyperparameters of the Random Forest algorithm to optimize model accuracy and generalization.

- Model Evaluation: - Assess the model's performance using appropriate evaluation metrics (e.g., accuracy, precision, recall, F1-score). - Utilize cross-validation techniques to validate the model's robustness and identify potential overfitting or underfitting issues.

- Documentation and Reporting: - Document the entire process, including data preprocessing, feature engineering, model development, and evaluation. - Clearly explain the steps taken, choices made, and rationale behind them. - Present the findings, including insights gained from EDA and the model's performance metrics.

Project Achievement for "Titanic Survivor Prediction using Random Forest":

- Successful Model Creation: - Developed a Random Forest classification model that accurately predicts whether a Titanic passenger survived or not. - Achieved a high level of accuracy and precision in the predictions, meeting the project's objective.

- Data Preprocessing and Feature Engineering: - Conducted thorough data preprocessing, handling missing values and encoding categorical variables appropriately. - Performed feature engineering techniques to enhance the model's predictive power and capture relevant information.

- Model Evaluation and Performance: - Evaluated the model's performance using various evaluation metrics, ensuring its effectiveness in predicting survival outcomes. - Conducted cross-validation to assess the model's generalization ability and minimize potential overfitting or underfitting.

- Documentation and Reporting: - Documented the entire project, providing a clear and comprehensive overview of the process followed. - Reported the findings, including insights gained from EDA, the model's performance metrics, and any challenges encountered.

By successfully executing the project plan and achieving the desired objectives, the team has demonstrated their ability to build an accurate Random Forest model for predicting Titanic survivorship. The project's documentation and reporting provide valuable insights and serve as a foundation for future projects in the field of classification modeling.

Appendix C

Link to the Presentation File

In this section, you will find the link to the presentation file. The presentation file provides an overview of our project and includes detailed information about our findings and recommendations. Please click on the following link to access the presentation file: Presentation File Link.