Kaggle Titanic ML

“Solving” the Titanic ML Solution… - Kushagra Bharti

Couple Steps in Solving the Problem

1. Determining the Question
2. Acquiring training and testing data (from here)
3. Cleansing the Data using Pandas (learnt here)
4. Analyze & Explore the Data
5. Create Models & Predict
6. Visualize & Create a Report

Overview

- On April 15, 1912, during her maiden voyage, the Titanic sank after colliding with an iceberg, killing 1502 out of 2224 passengers and crew. Translated 32% survival rate.
- One of the reasons that the shipwreck led to such loss of life was that there were not enough lifeboats for the passengers and crew.
- Although there was some element of luck involved in surviving the sinking, some groups of people were more likely to survive than others, such as women, children, and the upper-class.

Data Dict.

Variable	Definiton	Key
survival	did they survive	0 = No, 1 = Yes
pclass	what class were they in	1 = 2nd, 2 = 2nd, 3 = 3rd
sex	gender	
age	age in years	
sibsp	# of siblings or spouses	
parch	# of parents of children	
ticket	ticket number	
fare	passenger fare	
cabin	cabin number	
embarked	where they left from	C = Cherbourg, Q = Queenstown, S = Southhampton

Prerequisites

Machine learning uses algorithms to learn from data in datasets. They find patterns, develop understanding, make decisions, and evaluate those decisions.

What is the difference between training and testing data? (link)

Training	Testing
Fed into ML model to learn and recognize patterns	Unknown to computer unlike the training
Typically larger than the testing data set	Used to test your ML model
Algo’s help machines solve problems using patterns	Represents real data and large but smaller than training
Used to teach your model	Used to compare to see if your model is useful

LET’S START!!!

Obviously, first we want to import some libraries to make our lives easier

```
# data analysis and wrangling
import pandas as pd
import numpy as np
import random as rnd

# visualization
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

# machine learning

# Fitting a line to a curve
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC, LinearSVC

# Used for sorting data
from sklearn.ensemble import RandomForestClassifier # multiple decision trees (hard)
from sklearn.tree import DecisionTreeClassifier # just a decision tree

# Uses proximity to solve the algorithm (set how many closest points)
from sklearn.neighbors import KNeighborsClassifier

# Constant updates to the equation based on your data (also create a slope at the end)
# Single layer neural network (basically kind of)
from sklearn.naive_bayes import GaussianNB

# Proper Neural Network
from sklearn.linear_model import Perceptron

# Aligned with random forest algorithm (used for optimization)
# (much more efficient than logistic regression)
from sklearn.linear_model import SGDClassifier
```

We import lots of different libraries, all with their own purposes.

- ▼ Pandas

  To work with CSV or XLXS files with python

- ▼ Numpy

  To use number more effectively within python

- ▼ Random

  Just random… lol

- ▼ Seaborn

  Very similar to matpotlib but is used to data visualization with graphs and charts

▼ Matpotlib

  Used to plot data… graphs and charts

▼ Sklearn

  ML library… has bunch of algorithms built into the system to make it convenient

Next we import our data…

```
train_df = pd.read_csv('train.csv')
test_df = pd.read_csv('test.csv')
combine = [train_df, test_df]
```

Importing both training and testing data as you can see…

Combining data sets to that it is more convenient to make changes to both data sets at the same time

Now let’s train our data…

We want to figure out our feature names (column names) (very annoying lingo)

```
print(train_df.columns.values)
train_df.head()
```

both figuring out feature names and giving us a preview of the data (useless???)

We also want to separate our data into three different types: Categorical & Numerical.

▼ Categorical

  Similar or Exactly the same values multiple times

  - Survived (Yes or No)
  - Gender (Male or Female)

▼ Numerical

  Data that has numbers attached to it

  - Age (0-99)
  - Fare ($$$)

▼ Alphanumerical

  Data that has both letters and numbers…

  - Tickets (All sorts of things)
  - Cabin (Alphanumerical)

▼ Errors?

  Any data that we may both be able to use at all.

  - Name (irrelevant and no way to use to train models)

```
train_df.tail()
```

This time we printed the bottom of the data

Now that we know what our data looks like with different values and all… we must start cleaning our data out and all…

```
train_df.info()
print('_'*40) #Just a divider... nothing else
test_df.info()
```

This is to get info on our data with null values and sorts…

General formatting of info function —>

Column 1	Unique Values in Column 1	Any null?	Type of Values
PassengerID	891	non-null	int64 (64 base integers?)
Name	891	non-null	object (name or unkown)
Age	714	non-null	float64 (64 base float?)

```
train_df.describe()
```

VERY similar to .info() function but more mathematical and statistical values and analysis of data

```
train_df[['Pclass', 'Survived']].groupby(['Pclass'], as_index=False).mean().sort_values(by='Survived', ascending=False)
```

Quick test to verify our data…

And from output we can see that it seems to work out…

Now lets do it with gender

```
train_df[["Sex", "Survived"]].groupby(['Sex'], as_index=False).mean().sort_values(by='Survived', ascending=False)
```

And now with SibSp

```
train_df[["SibSp", "Survived"]].groupby(['SibSp'], as_index=False).mean().sort_values(by='Survived', ascending=False)
```

And lastly with Parch

```
train_df[["Parch", "Survived"]].groupby(['Parch'], as_index=False).mean().sort_values(by='Survived', ascending=False)
```

NOW let’s start analyzing the numbers more and more and also figuring out some like guidelines

We know that infants and children had higher rates of survival.

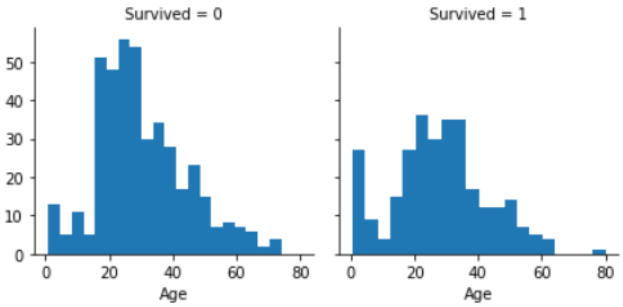
People above the age of 80 survived

People within ages 15-25 did not survive too well

Most passengers are in 15-35 yr old range (standard deviation)

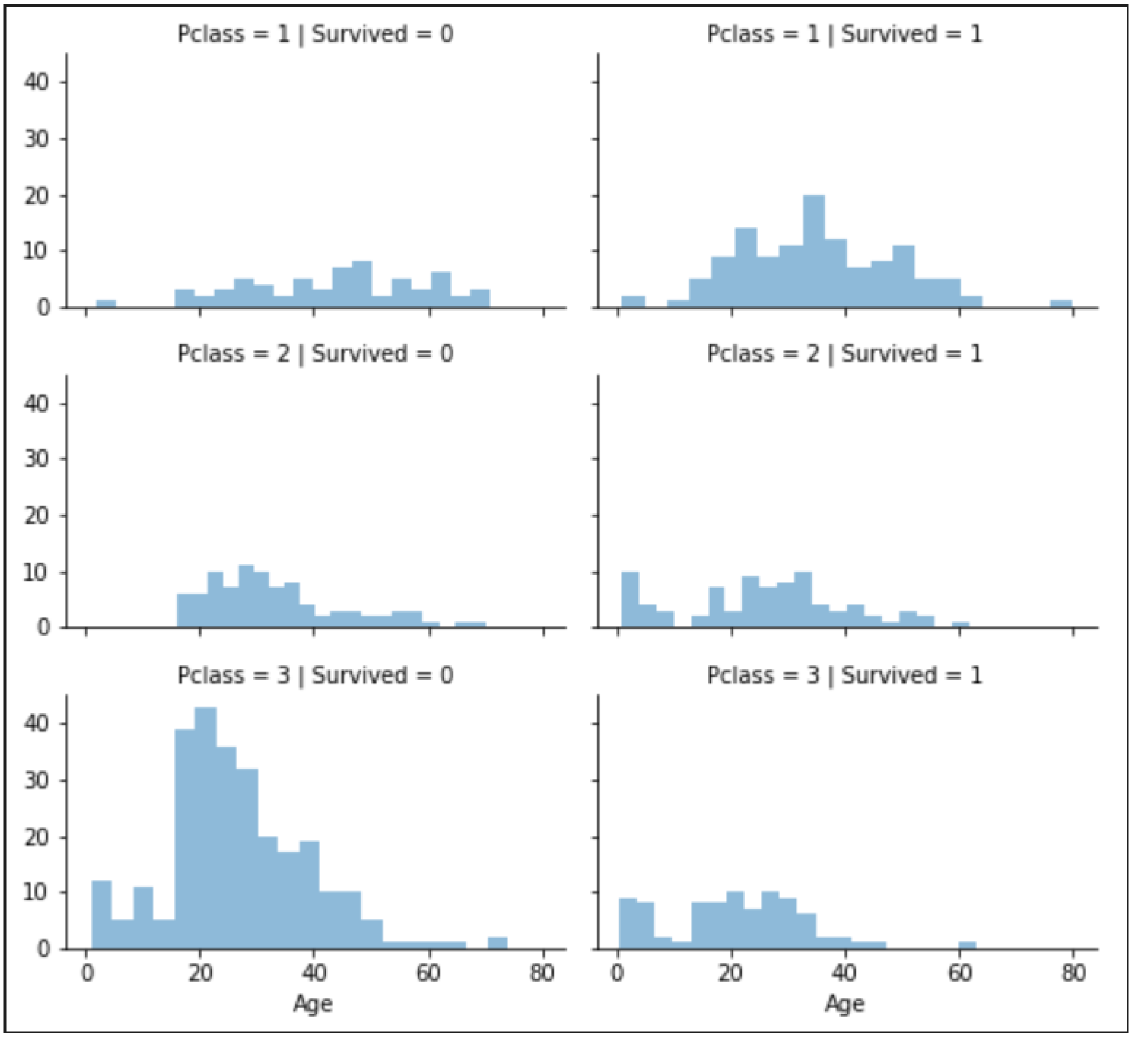
```
g = sns.FacetGrid(train_df, col='Survived')
g.map(plt.hist, 'Age', bins=20)
```

We just graphed the ages of people surviving and not surviving… not the best and ideal graph layout but better than nothing…



Now lets do the same for passenger class for better visualization

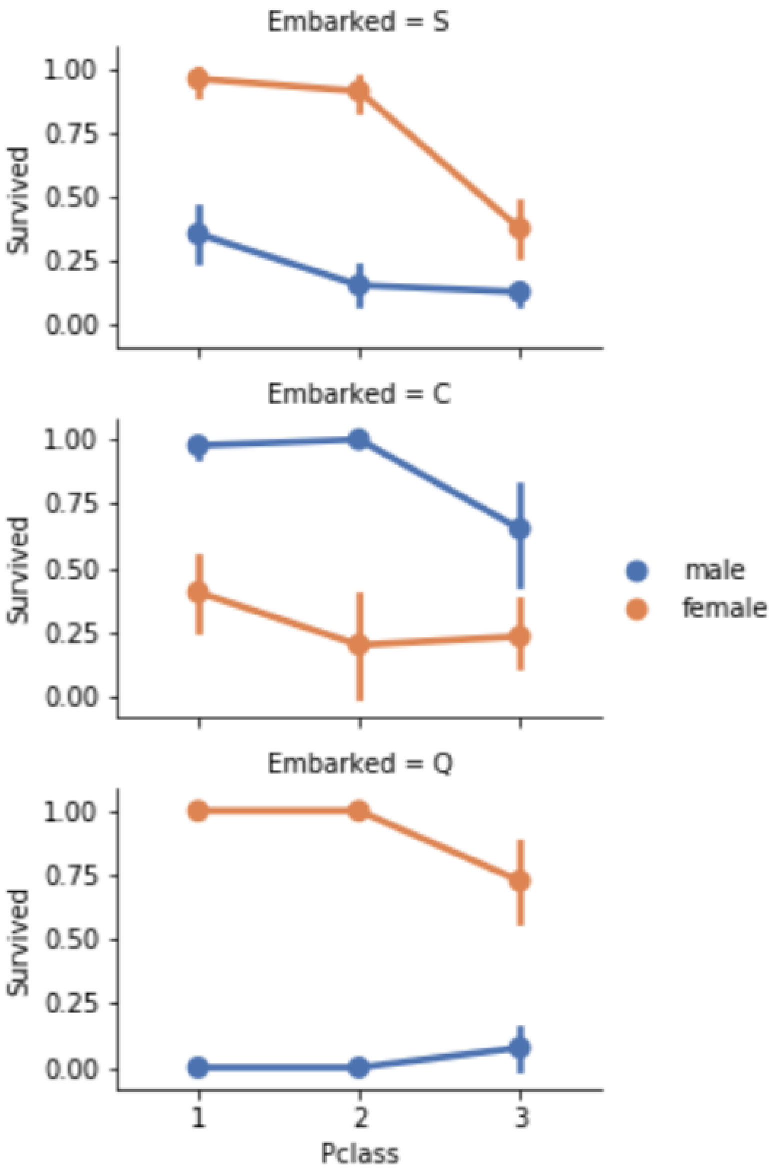
```
# grid = sns.FacetGrid(train_df, col='Pclass', hue='Survived')
grid = sns.FacetGrid(train_df, col='Survived', row='Pclass', size=2.2, aspect=1.6)
grid.map(plt.hist, 'Age', alpha=.5, bins=20)
grid.add_legend();
```



Now let’s include sex into this graph…

```
grid = sns.FacetGrid(train_df, row='Embarked', size=2.2, aspect=1.6)
grid.map(sns.pointplot, 'Pclass', 'Survived', 'Sex', palette='deep')
grid.add_legend()
```

And here are the results…



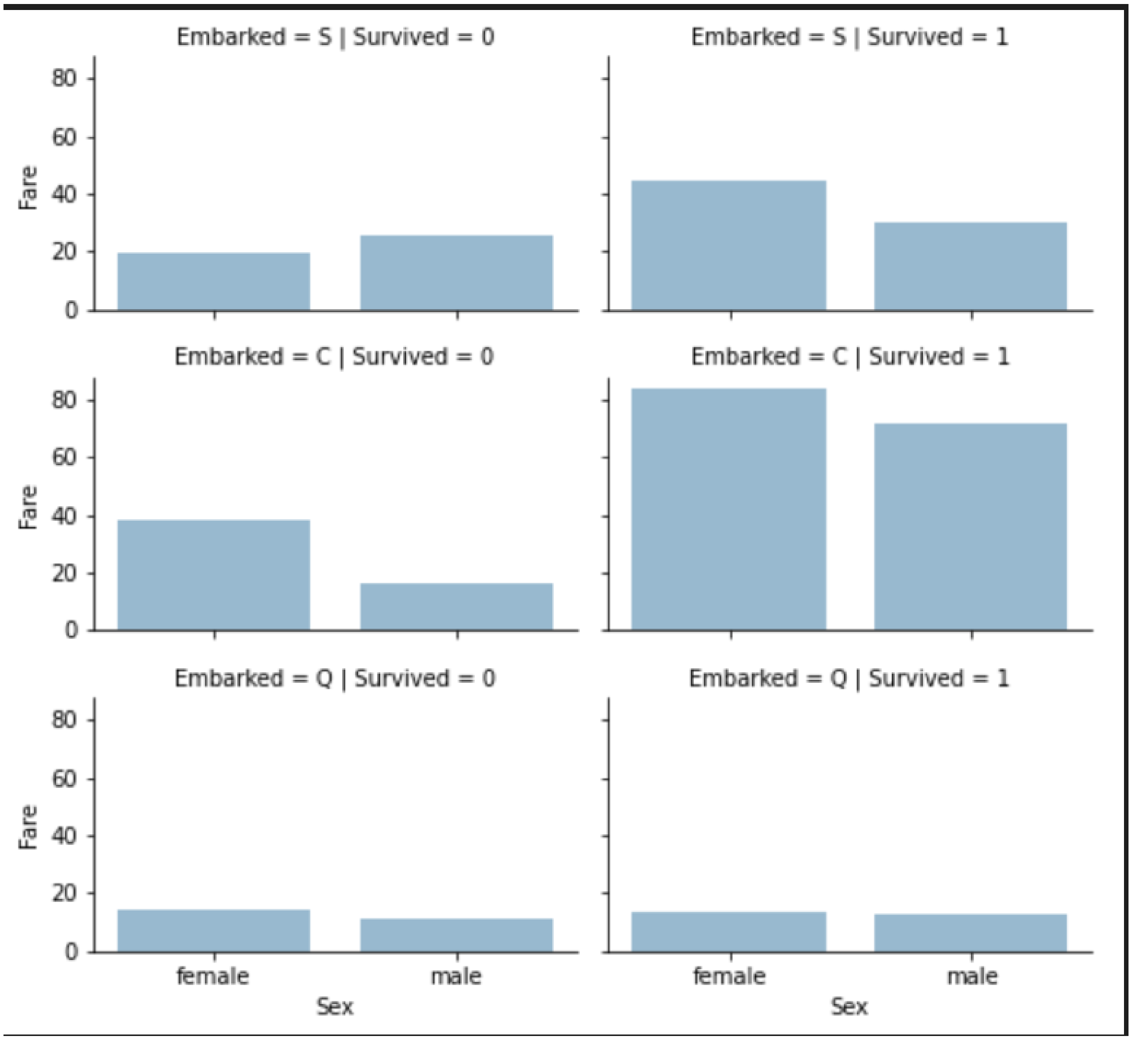
Here are some observations that we can deduce from these graphs…

- Female Survival > Male Survival
    - Except when in lowest class
- Lower end male survived more than the rest of the classes

And lastly, we are going to incorporate the fare price for it…

```
grid = sns.FacetGrid(train_df, row='Embarked', col='Survived', size=2.2, aspect=1.6)
grid.map(sns.barplot, 'Sex', 'Fare', alpha=.5, ci=None)
grid.add_legend()
```

Here are the results…



Here are some more observations made:

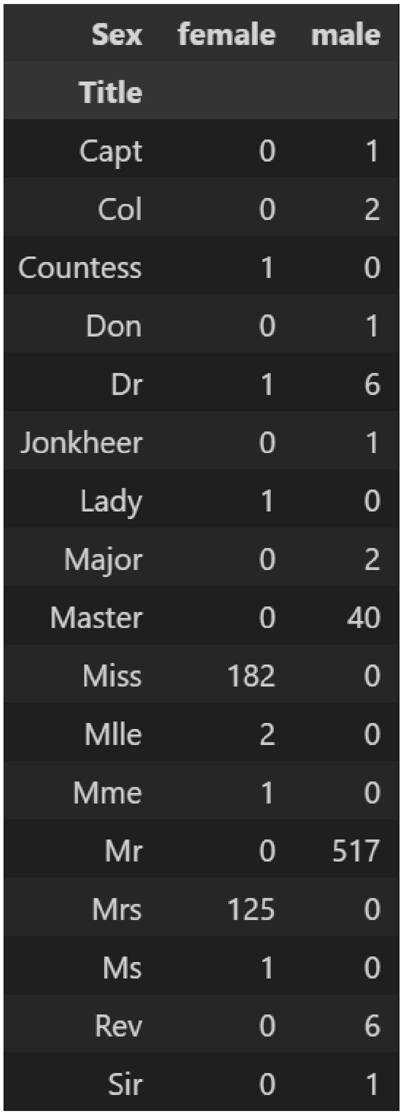
- more fare = more survival
- Embarkment relates to survival (why??? no idea)

Due to our observations… we can drop two of the factors (ticket and cabin)

```
train_df = train_df.drop(['Ticket', 'Cabin'], axis=1)
test_df = test_df.drop(['Ticket', 'Cabin'], axis=1)
combine = [train_df, test_df]
train_df.shape, test_df.shape, combine[0].shape, combine[1].shape
```

Now let’s do some testing with the title class…

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

pd.crosstab(train_df['Title'], train_df['Sex'])
```

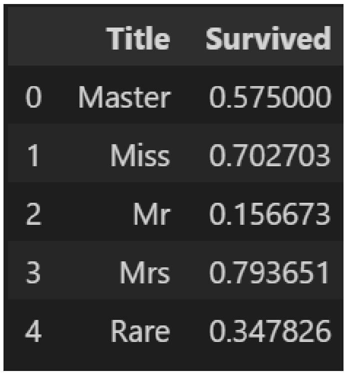


Sex	female	male
Title		
Capt	0	1
Col	0	2
Countess	1	0
Don	0	1
Dr	1	6
Jonkheer	0	1
Lady	1	0
Major	0	2
Master	0	40
Miss	182	0
Mlle	2	0
Mme	1	0
Mr	0	517
Mrs	125	0
Ms	1	0
Rev	0	6
Sir	0	1

Titles give us pretty expected results… But we should check survivability for each title…

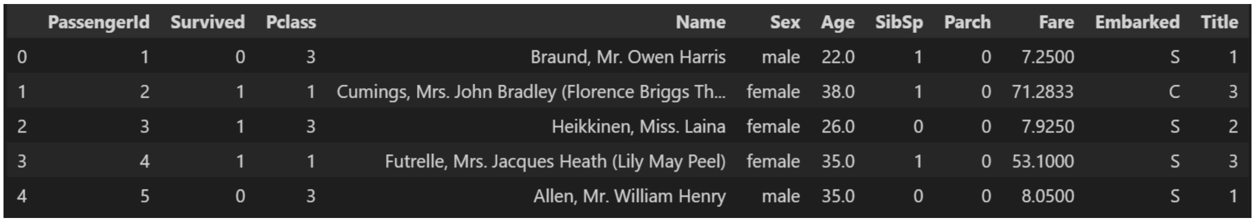
```
for dataset in combine:
    dataset['Title'] = dataset['Title'].replace(['Lady', 'Countess','Capt', 'Col',\
  'Don', 'Dr', 'Major', 'Rev', 'Sir', 'Jonkheer', 'Dona'], 'Rare')

    dataset['Title'] = dataset['Title'].replace('Mlle', 'Miss')
    dataset['Title'] = dataset['Title'].replace('Ms', 'Miss')
    dataset['Title'] = dataset['Title'].replace('Mme', 'Mrs')

train_df[['Title', 'Survived']].groupby(['Title'], as_index=False).mean()
```

And the results are also as expected… (don’t make significant changes to our current model)



	Title	Survived
0	Master	0.575000
1	Miss	0.702703
2	Mr	0.156673
3	Mrs	0.793651
4	Rare	0.347826

```
title_mapping = {"Mr": 1, "Miss": 2, "Mrs": 3, "Master": 4, "Rare": 5}
for dataset in combine:
    dataset['Title'] = dataset['Title'].map(title_mapping)
    dataset['Title'] = dataset['Title'].fillna(0)

train_df.head()
```



	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Fare	Embarked	Title
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	7.2500	S	1
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	71.2833	C	3
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	7.9250	S	2
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	53.1000	S	3
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	8.0500	S	1

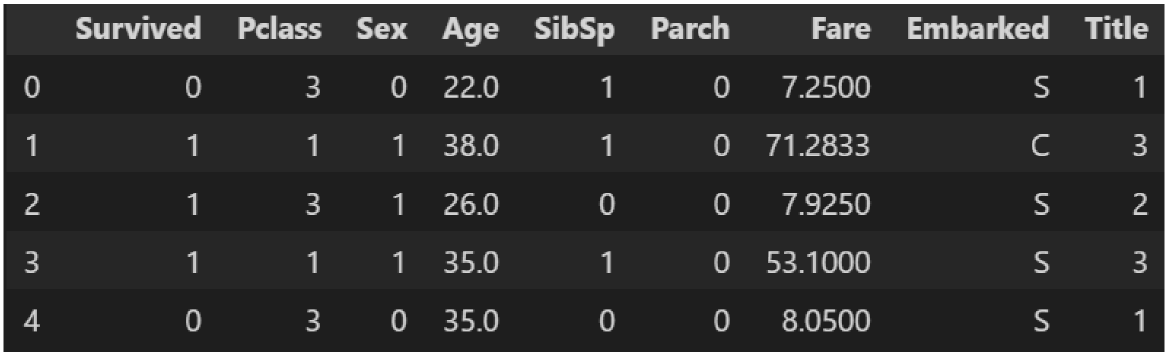
AND LASTLY… we can drop name and passenger ID too… bcuz they are pretty useless…

```
train_df = train_df.drop(['Name', 'PassengerId'], axis=1)
test_df = test_df.drop(['Name'], axis=1)
combine = [train_df, test_df]
train_df.shape, test_df.shapev
```

And now we also make everything binary and numerical to make it easier for computers to understand and just for us in general as well…

```
for dataset in combine:
    dataset['Sex'] = dataset['Sex'].map( {'female': 1, 'male': 0} ).astype(int)

train_df.head()
```

Here’s the results now.

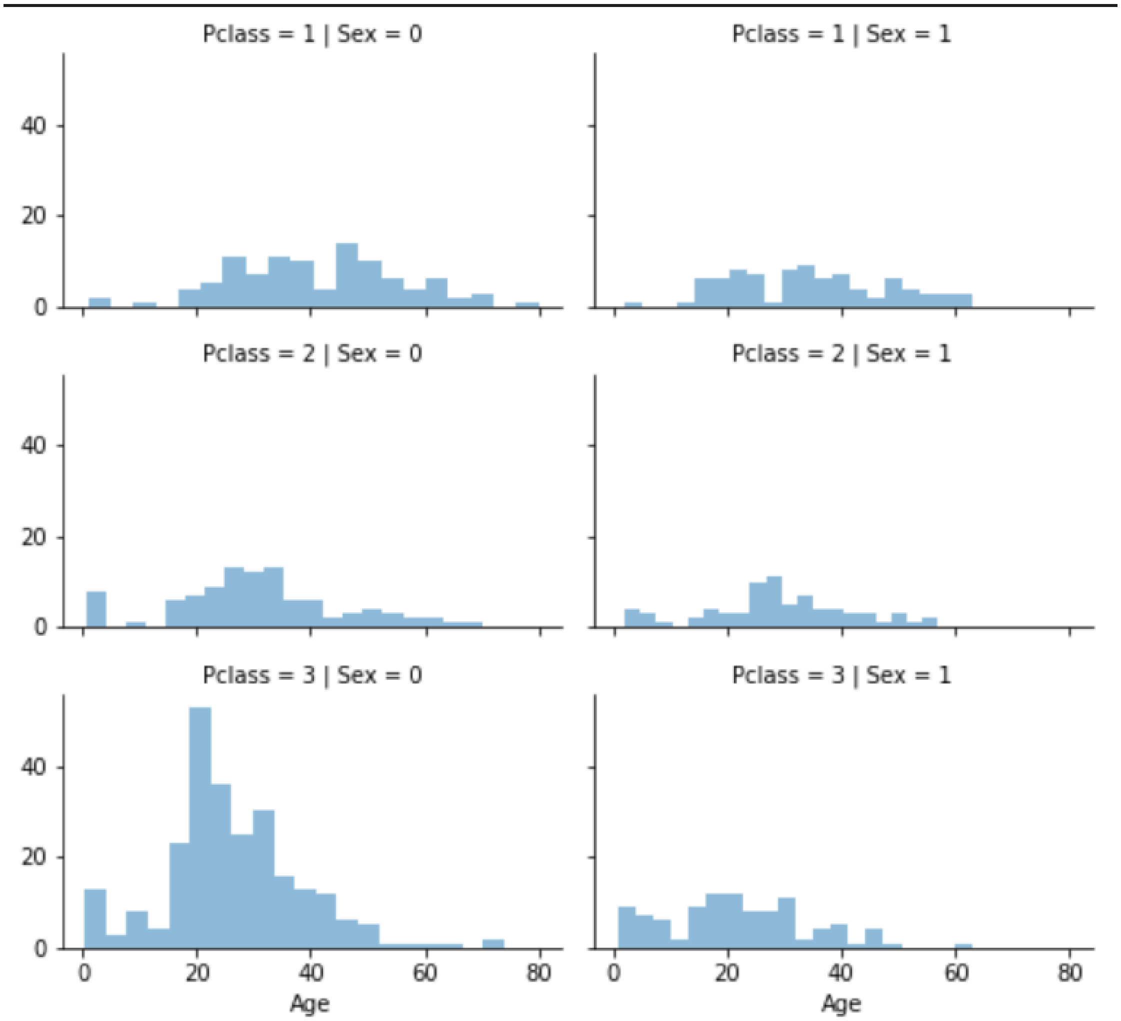


	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	Title
0	0	3	0	22.0	1	0	7.2500	S	1
1	1	1	1	38.0	1	0	71.2833	C	3
2	1	3	1	26.0	0	0	7.9250	S	2
3	1	1	1	35.0	1	0	53.1000	S	3
4	0	3	0	35.0	0	0	8.0500	S	1

Now let's fill in all the data that are null values or incomplete… (using arrays and filling in age, sex, etc…)

```
grid = sns.FacetGrid(train_df, row='Pclass', col='Sex', size=2.2, aspect=1.6)
grid.map(plt.hist, 'Age', alpha=.5, bins=20)
grid.add_legend()
```

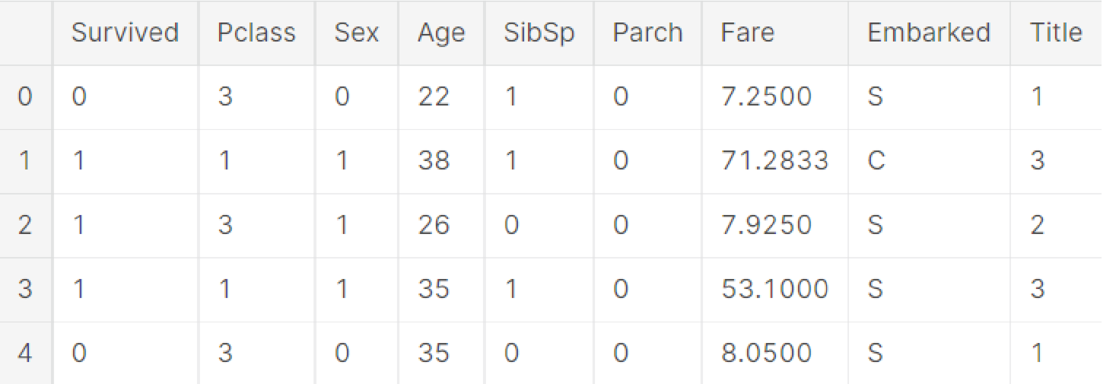
More results…



```
guess_ages = np.zeros((2,3))
guess_ages

for dataset in combine:
    for i in range(0, 2):
        for j in range(0, 3):
            guess_df = dataset[(dataset['Sex'] == i) & \
                                  (dataset['Pclass'] == j+1)]['Age'].dropna()

            age_guess = guess_df.median()

            guess_ages[i,j] = int( age_guess/0.5 + 0.5 ) * 0.5

    for i in range(0, 2):
        for j in range(0, 3):
            dataset.loc[ (dataset.Age.isnull()) & (dataset.Sex == i) & (dataset.Pclass == j+1),\
                    'Age'] = guess_ages[i,j]

    dataset['Age'] = dataset['Age'].astype(int)

train_df.head()
```

Here’s what the training data looks like now:

After we filled in the data together.



	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	Title
0	0	3	0	22	1	0	7.2500	S	1
1	1	1	1	38	1	0	71.2833	C	3
2	1	3	1	26	0	0	7.9250	S	2
3	1	1	1	35	1	0	53.1000	S	3
4	0	3	0	35	0	0	8.0500	S	1

Determing survival for age groups

```
train_df['AgeBand'] = pd.cut(train_df['Age'], 5)
train_df[['AgeBand', 'Survived']].groupby(['AgeBand'], as_index=False).mean().sort_values(by='AgeBand', ascending=True)
```

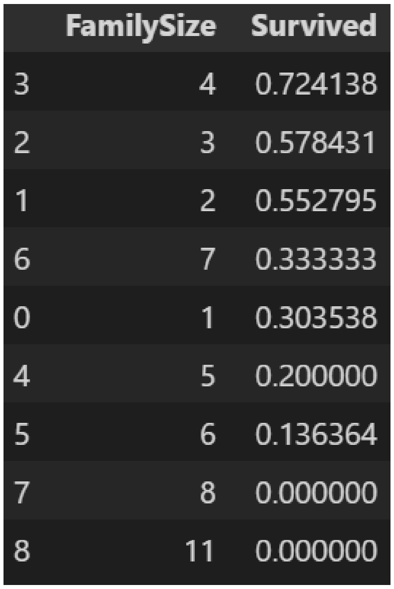
filling in the data…

```
for datasetin combine:
    dataset.loc[ dataset['Age'] <= 16, 'Age'] = 0
    dataset.loc[(dataset['Age'] > 16) & (dataset['Age'] <= 32), 'Age'] = 1
    dataset.loc[(dataset['Age'] > 32) & (dataset['Age'] <= 48), 'Age'] = 2
    dataset.loc[(dataset['Age'] > 48) & (dataset['Age'] <= 64), 'Age'] = 3
    dataset.loc[ dataset['Age'] > 64, 'Age']
train_df.head()
```

Let’s combined sibsp and parch to make a bigger and better “family number” class…

```
for dataset in combine:
    dataset['FamilySize'] = dataset['SibSp'] + dataset['Parch'] + 1

train_df[['FamilySize', 'Survived']].groupby(['FamilySize'], as_index=False).mean().sort_values(by='Survived', ascending=False)
```

here are the results…

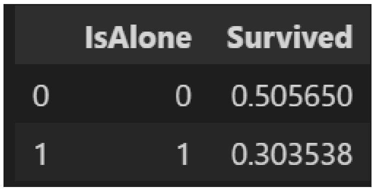


	FamilySize	Survived
3	4	0.724138
2	3	0.578431
1	2	0.552795
6	7	0.333333
0	1	0.303538
4	5	0.200000
5	6	0.136364
7	8	0.000000
8	11	0.000000

But what about for the people who are alone? got it covered!

```
for dataset in combine:
    dataset['IsAlone'] = 0
    dataset.loc[dataset['FamilySize'] == 1, 'IsAlone'] = 1

train_df[['IsAlone', 'Survived']].groupby(['IsAlone'], as_index=False).mean()
```

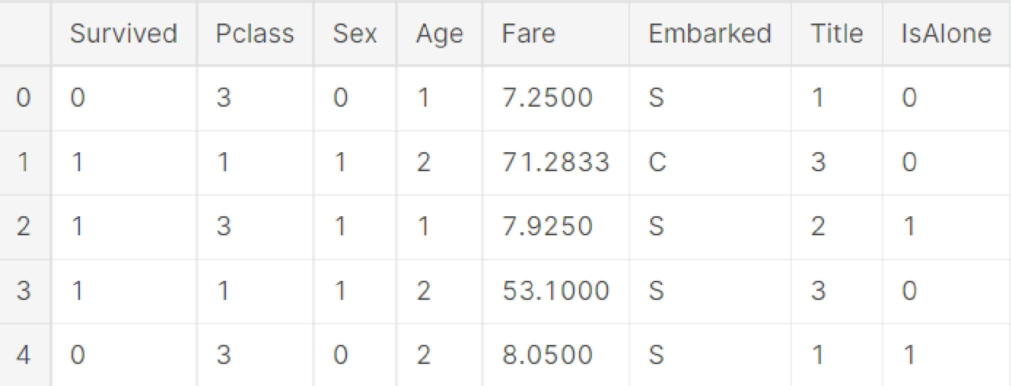
and boom results:



	IsAlone	Survived
0	0	0.505650
1	1	0.303538

dropping parch, sibsp, and family, and just simplifying it to “IsAlone”…

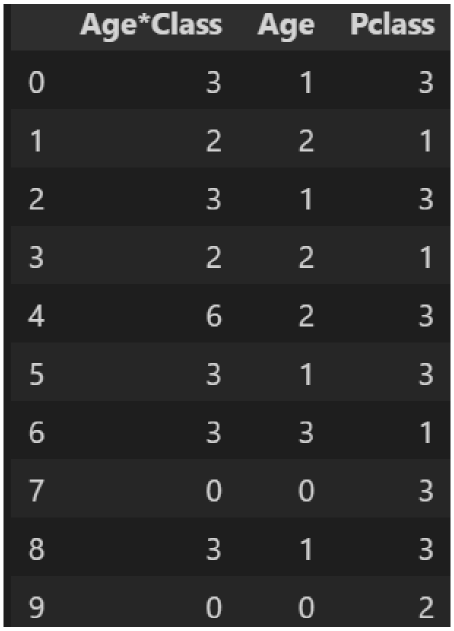
```
train_df = train_df.drop(['Parch', 'SibSp', 'FamilySize'], axis=1)
test_df = test_df.drop(['Parch', 'SibSp', 'FamilySize'], axis=1)
combine = [train_df, test_df]

train_df.head()
```



	Survived	Pclass	Sex	Age	Fare	Embarked	Title	IsAlone
0	0	3	0	1	7.2500	S	1	0
1	1	1	1	2	71.2833	C	3	0
2	1	3	1	1	7.9250	S	2	1
3	1	1	1	2	53.1000	S	3	0
4	0	3	0	2	8.0500	S	1	1

Let’s also combine Pclass and age…

```
for dataset in combine:
    dataset['Age*Class'] = dataset.Age * dataset.Pclass

train_df.loc[:, ['Age*Class', 'Age', 'Pclass']].head(10)
```



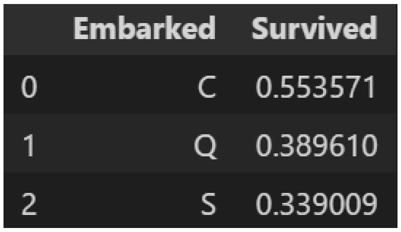
	Age*Class	Age	Pclass
0	3	1	3
1	2	2	1
2	3	1	3
3	2	2	1
4	6	2	3
5	3	1	3
6	3	3	1
7	0	0	3
8	3	1	3
9	0	0	2

The embarked column has a missing value in two spots… we will just fill it by using the most common point of embarkment…

```
freq_port = train_df.Embarked.dropna().mode()[0]
freq_port # Results gives us "S" as the most common
```

Lets check for survivability…

```
for dataset in combine:
    dataset['Embarked'] = dataset['Embarked'].fillna(freq_port)

train_df[['Embarked', 'Survived']].groupby(['Embarked'], as_index=False).mean().sort_values(by='Survived', ascending=False)
```



	Embarked	Survived
0	C	0.553571
1	Q	0.389610
2	S	0.339009

S is the most common but also the place w the lowest survival rates… (skewed data?) I don't think it is skewed but there could be a possibility.

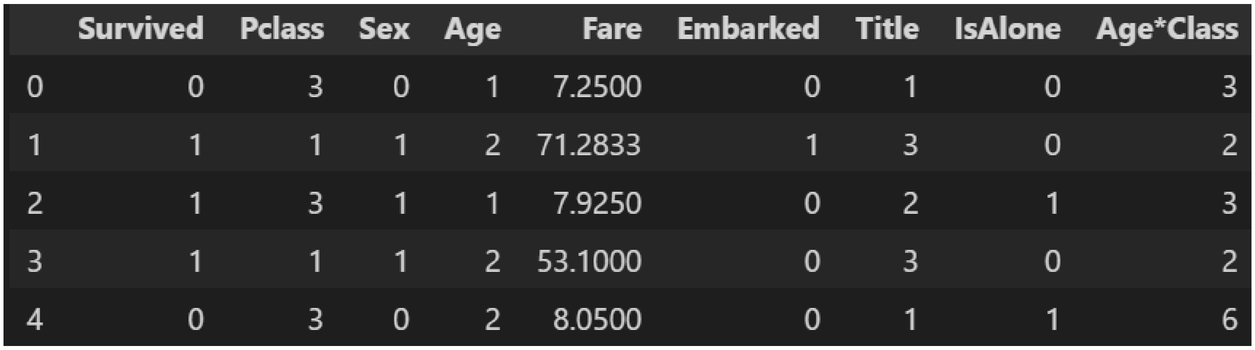
Now time to turn the embarked values into numerical values (easier for computers to understand)

S - 0

C - 1

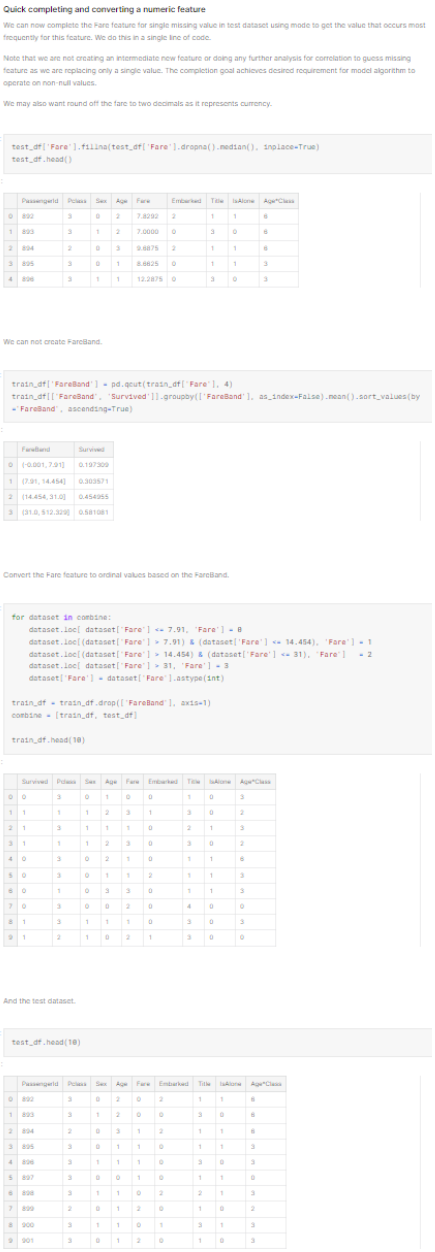
Q - 2

```
for dataset in combine:
    dataset['Embarked'] = dataset['Embarked'].map( {'S': 0, 'C': 1, 'Q': 2} ).astype(int)

train_df.head()
```



	Survived	Pclass	Sex	Age	Fare	Embarked	Title	IsAlone	Age*Class
0	0	3	0	1	7.2500	0	1	0	3
1	1	1	1	2	71.2833	1	3	0	2
2	1	3	1	1	7.9250	0	2	1	3
3	1	1	1	2	53.1000	0	3	0	2
4	0	3	0	2	8.0500	0	1	1	6

From here on… I did not understand the tutorial as well as I would have liked to…



Quick completing and converting a numeric feature

We can now complete the Fare feature for single missing value in test dataset using mode to get the value that occurs most frequently for this feature. We do this in a single line of code.

Note that we are not creating an intermediate new feature or doing any further analysis for correlation to guess missing feature as we are replacing only a single value. The completion goal achieves desired requirement for model algorithm to operate on non-null values.

We may also want round off the fare to two decimals as it represents currency.

```
test_df['Fare'].fillna(test_df['Fare'].dropna().median(), inplace=True)
test_df.head()
```

	PassengerId	Pclass	Sex	Age	Fare	Embarked	Title	IsAlone	Age*Class
0	892	3	0	2	7.8292	2	1	1	6
1	893	3	1	2	7.0000	0	3	0	6
2	894	2	0	3	9.6875	2	1	1	6
3	895	3	0	1	8.6625	0	1	1	3
4	896	3	1	1	12.2875	0	3	0	3

We can not create FareBand.

```
train_df['FareBand'] = pd.qcut(train_df['Fare'], 4)
train_df[['FareBand', 'Survived']].groupby(['FareBand'], as_index=False).mean().sort_values(by
='FareBand', ascending=True)
```

	FareBand	Survived
0	(-0.001, 7.91]	0.197309
1	(7.91, 14.454]	0.303571
2	(14.454, 31.0]	0.454955
3	(31.0, 512.329]	0.581081

Convert the Fare feature to ordinal values based on the FareBand.

```
for dataset in combine:
    dataset.loc[ dataset['Fare'] <= 7.91, 'Fare'] = 0
    dataset.loc[(dataset['Fare'] > 7.91) & (dataset['Fare'] <= 14.454), 'Fare'] = 1
    dataset.loc[(dataset['Fare'] > 14.454) & (dataset['Fare'] <= 31), 'Fare']   = 2
    dataset.loc[ dataset['Fare'] > 31, 'Fare'] = 3
    dataset['Fare'] = dataset['Fare'].astype(int)

train_df = train_df.drop(['FareBand'], axis=1)
combine = [train_df, test_df]

train_df.head(10)
```

	Survived	Pclass	Sex	Age	Fare	Embarked	Title	IsAlone	Age*Class
0	0	3	0	1	0	0	1	0	3
1	1	1	1	2	3	1	3	0	2
2	1	3	1	1	1	0	2	1	3
3	1	1	1	2	3	0	3	0	2
4	0	3	0	2	1	0	1	1	6
5	0	3	0	1	1	2	1	1	3
6	0	1	0	3	3	0	1	1	3
7	0	3	0	0	2	0	4	0	0
8	1	3	1	1	1	0	3	0	3
9	1	2	1	0	2	1	3	0	0

And the test dataset.

```
test_df.head(10)
```

	PassengerId	Pclass	Sex	Age	Fare	Embarked	Title	IsAlone	Age*Class
0	892	3	0	2	0	2	1	1	6
1	893	3	1	2	0	0	3	0	6
2	894	2	0	3	1	2	1	1	6
3	895	3	0	1	1	0	1	1	3
4	896	3	1	1	1	0	3	0	3
5	897	3	0	0	1	0	1	1	0
6	898	3	1	1	0	2	2	1	3
7	899	2	0	1	2	0	1	0	2
8	900	3	1	1	0	1	3	1	3
9	901	3	0	1	2	0	1	0	3

https://www.kaggle.com/code/startupsci/titanic-data-science-solutions

AND FINALLY…

We can start Modeling…

All the code for this is in the .ipynb file that I have shared…

Thing To Do Still:

□ Understand the Fare Band Feature towards the end of the tutorial

□ Understand and Research the Algorithms used for predicting in the end… (I have added basic notes earlier in the report… in the first bit of code… but I haven't been able to research further.)

- Logistic Regression
- KNN or k-Nearest Neighbors
- Support Vector Machines
- Naive Bayes classifier
- Decision Tree
- Random Forrest
- Perceptron
- Artificial neural network
- RVM or Relevance Vector Machine