Exploratory Data Analysis

- **Exploratory Data Analysis** refers to the critical process of performing initial investigations on data so as to **discover pattern**s,to **spot anomalies**,to **test hypothesis** and to check assumptions with the help of summary statistics and graphical representations.
- It is a good practice to **understand the data first** and **try to gather as many insights from it**. EDA is all about making sense of data in hand,before getting them dirty with it.

Charactersitics of EDA:

- Gain insight into the data
- More common ways of summarizing location, spread, and shape analysis
- Used resistant statistics
- From these we could make decisions on test selection and whether the data should be transformed or reexpressed before further analysis
- inspect relationships between and among variables

Logistic Regression with Python

Dataset: [Titanic Data Set from Kaggle](). A famous data set and very often is a student's first step in machine learning! and to enter into kaggle competation.

We'll be trying to predict a classification- survival or deceased.

Import Libraries

Let's import some libraries to get started!

```
import pandas as pd    # To load dataset and preprocess the data.
import numpy as np     # To work with Arrays and neumarical analysis.
import matplotlib.pyplot as plt  # For visualization.
import seaborn as sns  # For visualization and to apply statistical functions.
# For displaying graphs within jupyter notebook.
%matplotlib inline
```

The Data

Let's start by reading in the titanic_train.csv file into a pandas dataframe.

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

```
train.head()
```

	PassengerId	Survived	Pclass	Name	Gender	Age	SibSp	Parch	Ticket	Far
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.250
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th	female	38.0	1	0	PC 17599	71.283

Exploratory Data Analysis

We'll start by checking out missing data.

Missing Data

We can use seaborn to create a simple heatmap to see where we are missing data!

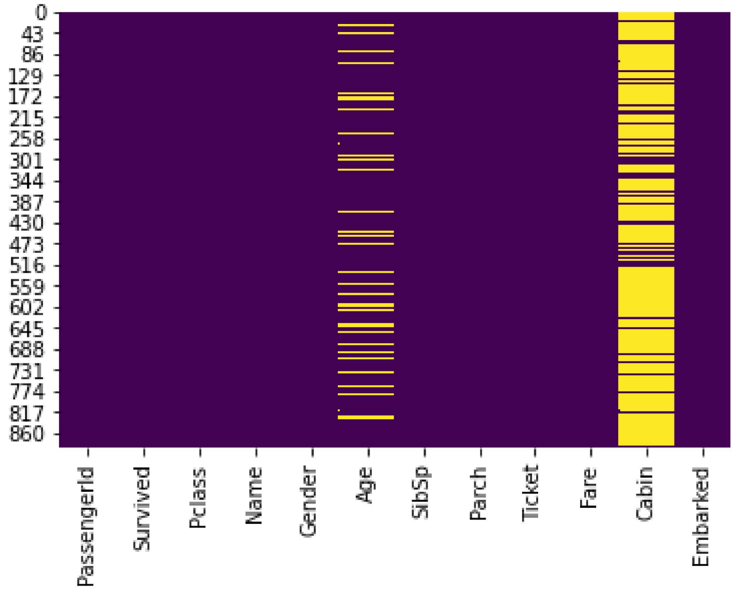
```
train.isnull()
```

	PassengerId	Survived	Pclass	Name	Gender	Age	SibSp	Parch	Ticket	Fare	Ca
0	False	False	False	False	False	False	False	False	False	False	T
1	False	False	False	False	False	False	False	False	False	False	Fa
2	False	False	False	False	False	False	False	False	False	False	T
3	False	False	False	False	False	False	False	False	False	False	Fa
4	False	False	False	False	False	False	False	False	False	False	T
...	...	...	...	...	...	...	...	...	...	...	
886	False	False	False	False	False	False	False	False	False	False	T
887	False	False	False	False	False	False	False	False	False	False	Fa
888	False	False	False	False	False	True	False	False	False	False	T
889	False	False	False	False	False	False	False	False	False	False	Fa
890	False	False	False	False	False	False	False	False	False	False	T

891 rows × 12 columns

```
sns.heatmap(train.isnull(),cbar=False,cmap='viridis')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2304f36e48>
```

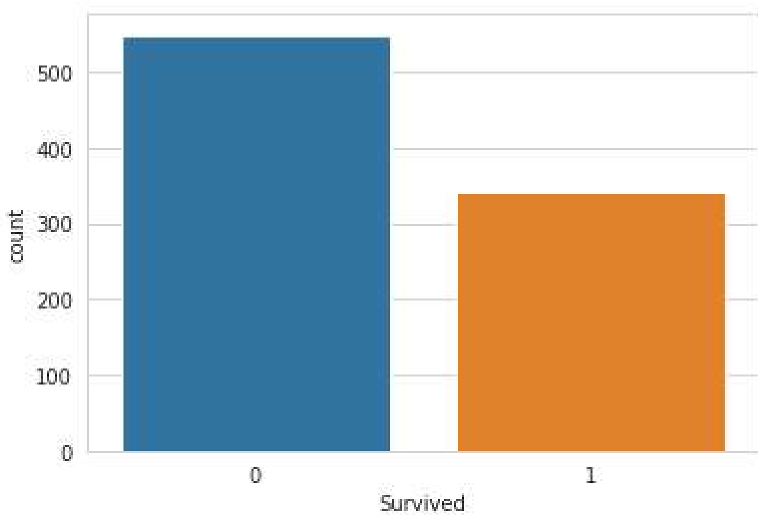


Age; Small percent (10 to 20) of the data is missing. The proportion of Age missing is likely small enough for reasonable replacement with some form of imputation. Cabin: Large percentage of data is missing. We can do 2 things.

- Drop coloum
- change it to another feature like Cabin Known: 1 or 0

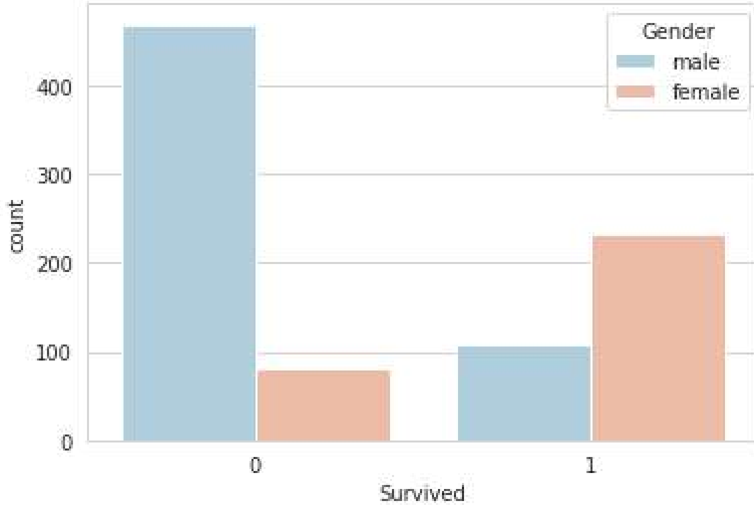
```
sns.set_style('whitegrid')
sns.countplot(x='Survived',data=train)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2304f317b8>
```



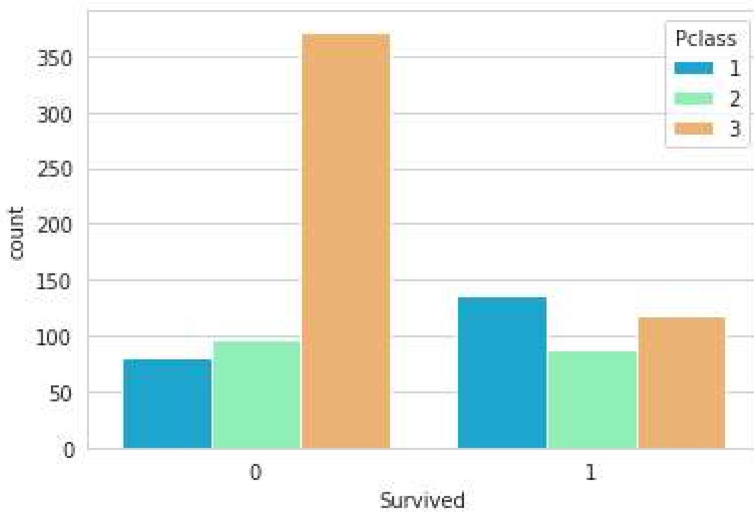

```
sns.set_style('whitegrid')
sns.countplot(x='Survived',hue='Gender',data=train,palette='RdBu_r')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2304e84160>
```



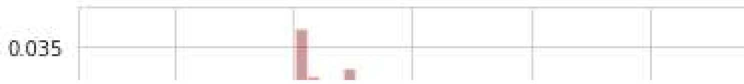

```
sns.set_style('whitegrid')
sns.countplot(x='Survived',hue='Pclass',data=train,palette='rainbow')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2304e0f6a0>
```



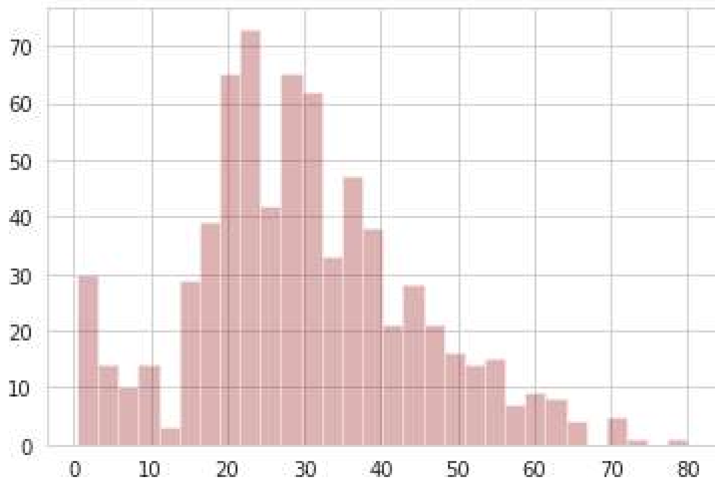

```
#sns.distplot(train['Age'].dropna(),kde=False,color='darkred',bins=40)
sns.distplot(train['Age'].dropna(),kde=True,color='darkred',bins=40)
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/distributions.py:2551: FutureWarning: `di
  warnings.warn(msg, FutureWarning)
<matplotlib.axes._subplots.AxesSubplot at 0x7f230434cbe0>
```



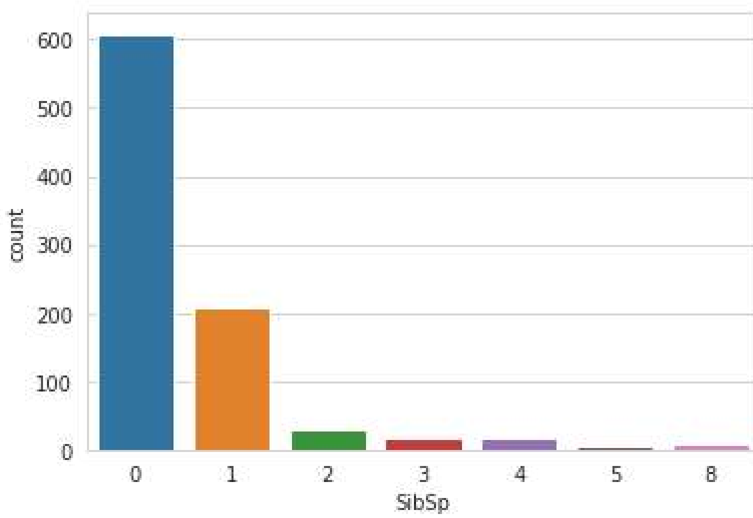

```
train['Age'].hist(bins=30,color='darkred',alpha=0.3)  # using matplot lib
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f23041f9400>
```



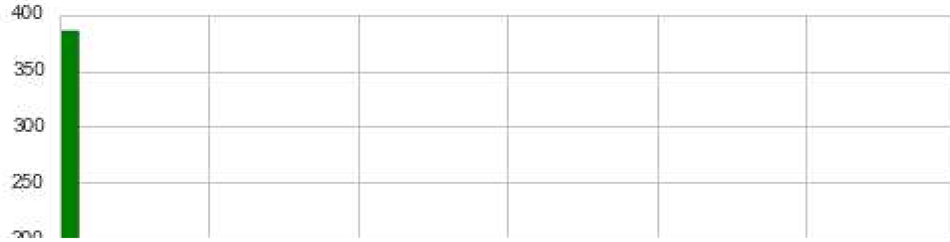

```
sns.countplot(x='SibSp',data=train)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f23040f14e0>
```




```
train['Fare'].hist(color='green',bins=40,figsize=(8,4))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x113893048>
```



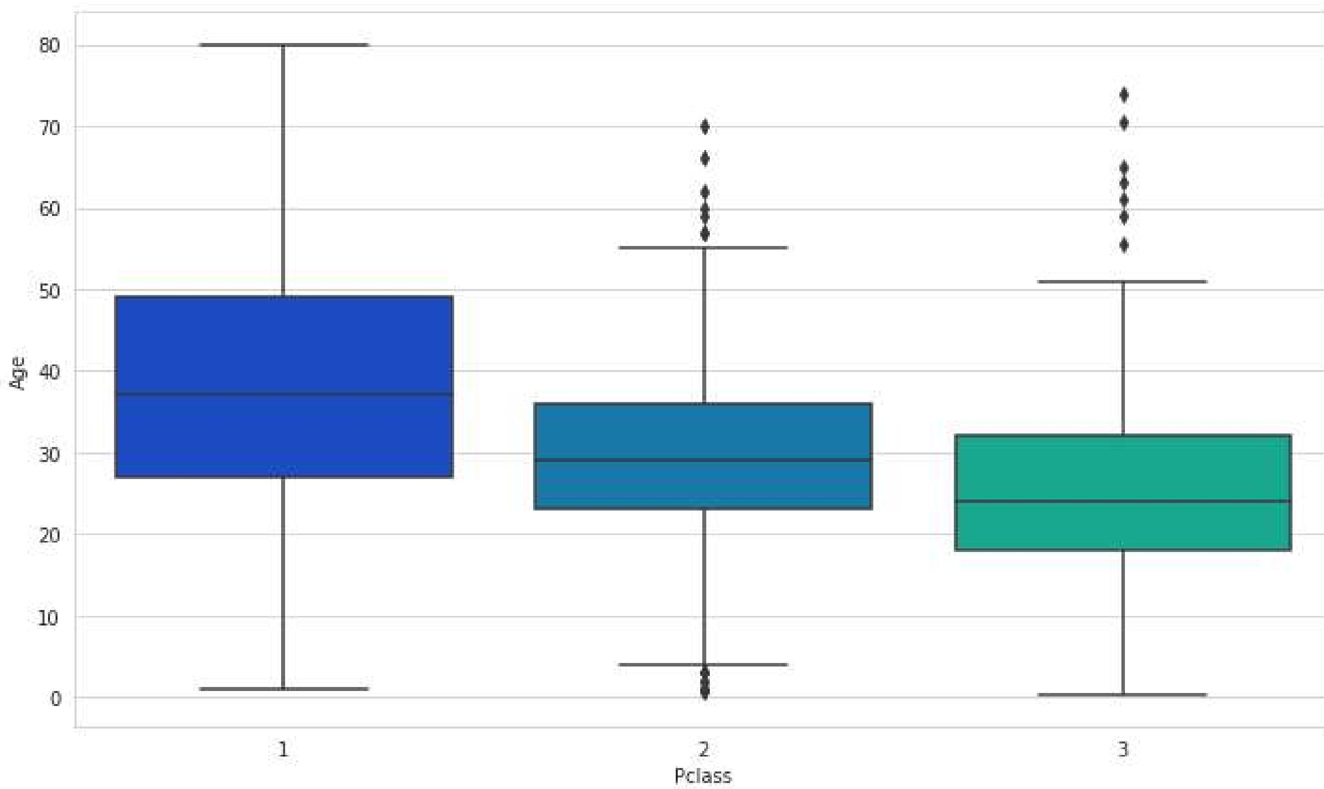
Data Cleaning

We want to fill in missing age data instead of just dropping the missing age data rows. One way to do this is by filling in the mean age of all the passengers (imputation). However we can be smarter about this and check the average age by passenger class. For example:

Link to understand Box/whisker Plot [Box Plot]()

```
plt.figure(figsize=(12, 7))
sns.boxplot(x='Pclass',y='Age',data=train,palette='winter')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f22fecb2d30>
```



We can see the wealthier passengers in the higher classes tend to be older, which makes sense. We'll use these average age values to impute based on Pclass for Age.

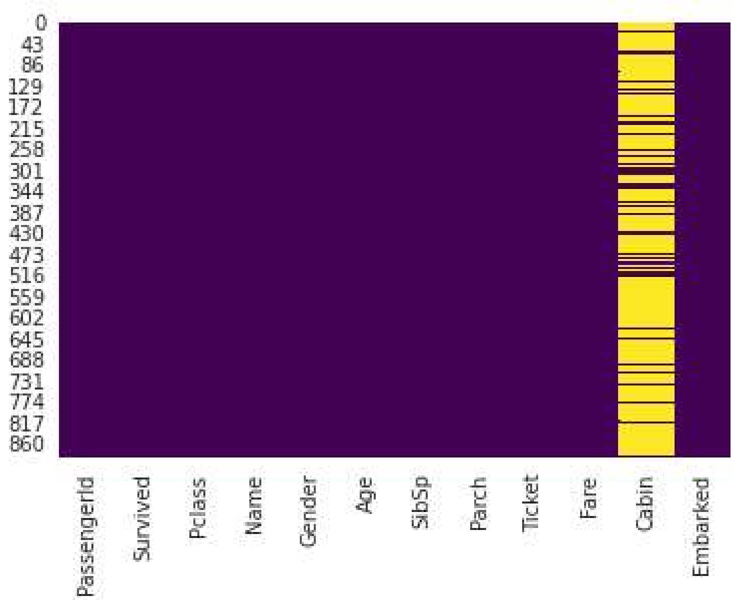
```
def impute_age(cols):
    Age = cols[0]
    Pclass = cols[1]

    if pd.isnull(Age):

        if Pclass == 1:
            return 37

        elif Pclass == 2:
            return 29

        else:
            return 24

    else:
        return Age
```

Now apply that function!

```
train['Age'] = train[['Age','Pclass']].apply(impute_age,axis=1)
```

```
sns.heatmap(train.isnull(),cbar=False,cmap='viridis')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f23040ce198>
```



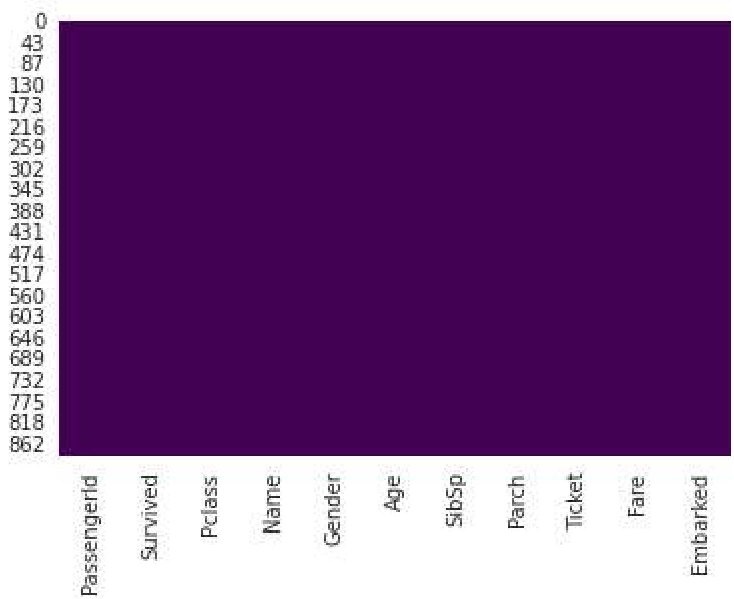
Great! Let's go ahead and drop the Cabin column and the row in Embarked that is NaN.

```
train.drop('Cabin',axis=1,inplace=True)
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-40-985ba4a0cedd> in <module>()
----> 1 train.drop('Cabin',axis=1,inplace=True)

3 frames
/usr/local/lib/python3.6/dist-packages/pandas/core/indexes/base.py in drop(self, labels, errors)
   5282         if mask.any():
   5283             if errors != "ignore":
-> 5284                 raise KeyError(f"{labels[mask]} not found in axis")
   5285             indexer = indexer[~mask]
   5286         return self.delete(indexer)

KeyError: "['Cabin'] not found in axis"
```

```
sns.heatmap(train.isnull(),cbar=False,cmap='viridis')
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f22febb7358>




```
train.head()
```

	PassengerId	Survived	Pclass	Name	Gender	Age	SibSp	Parch	Ticket	Fa
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.25
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th	female	38.0	1	0	PC 17599	71.28

```
train.dropna(inplace=True)
```

Converting Categorical Features

We'll need to convert categorical features to dummy variables using pandas! Otherwise our machine learning algorithm won't be able to directly take in those features as inputs.

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 889 entries, 0 to 890
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  889 non-null    int64
 1   Survived     889 non-null    int64
 2   Pclass       889 non-null    int64
 3   Name         889 non-null    object
 4   Gender       889 non-null    object
 5   Age          889 non-null    float64
 6   SibSp        889 non-null    int64
 7   Parch        889 non-null    int64
 8   Ticket       889 non-null    object
 9   Fare         889 non-null    float64
 10  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(4)
memory usage: 83.3+ KB
```

```
#pd.get_dummies(train['Embarked'],drop_first=True).head() #To avoid Dummy Variable Trap
pd.get_dummies(train['Embarked']).head()
```

	C	Q	S
0	0	0	1
1	1	0	0
2	0	0	1
3	0	0	1
4	0	0	1

```
gender = pd.get_dummies(train['Gender'],drop_first=True)
embark = pd.get_dummies(train['Embarked'],drop_first=True)
```

```
train.drop(['Gender','Embarked','Name','Ticket'],axis=1,inplace=True)
```

```
train.head()
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
0	1	0	3	22.0	1	0	7.2500
1	2	1	1	38.0	1	0	71.2833
2	3	1	3	26.0	0	0	7.9250
3	4	1	1	35.0	1	0	53.1000
4	5	0	3	35.0	0	0	8.0500

```
train = pd.concat([train,gender,embark],axis=1)
```

```
train.head()
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare	male	Q	S
0	1	0	3	22.0	1	0	7.2500	1	0	1
1	2	1	1	38.0	1	0	71.2833	0	0	0
2	3	1	3	26.0	0	0	7.9250	0	0	1
3	4	1	1	35.0	1	0	53.1000	0	0	1
4	5	0	3	35.0	0	0	8.0500	1	0	1

Now our data is ready for developing Model.

Here Dependent Variable = Survived and Independent Variable = rest of the variables.

Building a Logistic Regression model

Let's splitt data into a training set and test set

Train Test Split

```
train.drop('Survived',axis=1).head()
```

	PassengerId	Pclass	Age	SibSp	Parch	Fare	male	Q	S
0	1	3	22.0	1	0	7.2500	1	0	1
1	2	1	38.0	1	0	71.2833	0	0	0
2	3	3	26.0	0	0	7.9250	0	0	1
3	4	1	35.0	1	0	53.1000	0	0	1
4	5	3	35.0	0	0	8.0500	1	0	1

```
#Labelled Output
train['Survived'].head()
```

```
0    0
1    1
2    1
3    1
4    0
Name: Survived, dtype: int64
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(train.drop('Survived',axis=1),
                                                    train['Survived'], test_size=0.30,
                                                    random_state=101)
```

Training and Predicting

```
from sklearn.linear_model import LogisticRegression
```

+ Code + Text

```
logmodel = LogisticRegression()
logmodel.fit(X_train,y_train)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/_logistic.py:940: Convergen

lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

```
predictions = logmodel.predict(X_test)
```

```
from sklearn.metrics import confusion_matrix
```

```
accuracy=confusion_matrix(y_test,predictions)
```

```
accuracy
```

```
array([[149,  14],
       [ 39,  65]])
```

```
from sklearn.metrics import accuracy_score
```

```
accuracy=accuracy_score(y_test,predictions)
accuracy
```

```
0.8014981273408239
```

```
predictions
```

```
array([0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1,
       1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1,
       0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0,
       0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0,
       0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0,
       0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0,
       1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,
       0, 1, 1])
```

Let's move on to evaluate our model!

Evaluation

We can check precision,recall,f1-score using classification report!

```
from sklearn.metrics import classification_report
```

```
print(classification_report(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.79      0.91      0.85       163
           1       0.82      0.62      0.71       104

    accuracy                           0.80       267
   macro avg       0.81      0.77      0.78       267
```

```
weighted avg       0.80      0.80      0.80       267
```

Not so bad! You might want to explore other feature engineering and the other titanic_text.csv file, some suggestions for feature engineering:

- Try grabbing the Title (Dr.,Mr.,Mrs,etc..) from the name as a feature
- Maybe the Cabin letter could be a feature
- Is there any info you can get from the ticket?