CS580L : MACHINE LEARNING SPRING 2018

Machine Learning Algorithm for Predicting Disaster Rating

Manish Kumar, B00715673

Email: mkumar7@binghamton.edu

DESCRIPTION:-

- The sinking of the RMS Titanic is one of the most infamous shipwrecks in history. On April 15, 1912, during her maiden voyage, the Titanic sank after colliding with an iceberg, killing 1502 out of 2224 passengers and crew. This sensational tragedy shocked the international community and led to better safety regulations for ships.
- 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.
- In this project, I will complete the analysis of what sorts of people were likely to survive. In particular, I will apply the tools of machine learning to predict which passengers survived the tragedy.

FILE DESCRIPTION:-

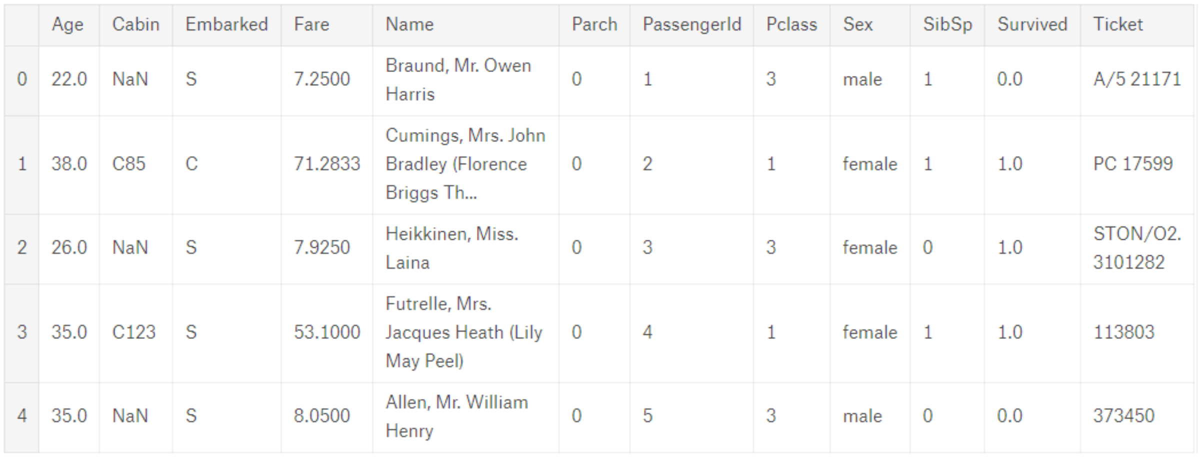
The data has been split into two groups:

- training set (train.csv)
- test set (test.csv)
- The training set should be used to build your machine learning models. For the training set, the outcome (also known as the "ground truth") for each passenger. My model will be based on "features" like passengers' gender and class. I can also use feature engineering to create new features.
- The test set should be used to see how well my model performs on unseen data. For the test set, There should not be any ground truth for each passenger. It is my job to predict these outcomes. For each

passenger in the test set, use the model you trained to predict they survived or not on the sinking of the Titanic.

Statistical summaries and Visualisations:

Titanic.head()



	Age	Cabin	Embarked	Fare	Name	Parch	PassengerId	Pclass	Sex	SibSp	Survived	Ticket
0	22.0	NaN	S	7.2500	Braund, Mr. Owen Harris	0	1	3	male	1	0.0	A/5 21171
1	38.0	C85	C	71.2833	Cumings, Mrs. John Bradley (Florence Briggs Th...	0	2	1	female	1	1.0	PC 17599
2	26.0	NaN	S	7.9250	Heikkinen, Miss. Laina	0	3	3	female	0	1.0	STON/O2. 3101282
3	35.0	C123	S	53.1000	Futrelle, Mrs. Jacques Heath (Lily May Peel)	0	4	1	female	1	1.0	113803
4	35.0	NaN	S	8.0500	Allen, Mr. William Henry	0	5	3	male	0	0.0	373450

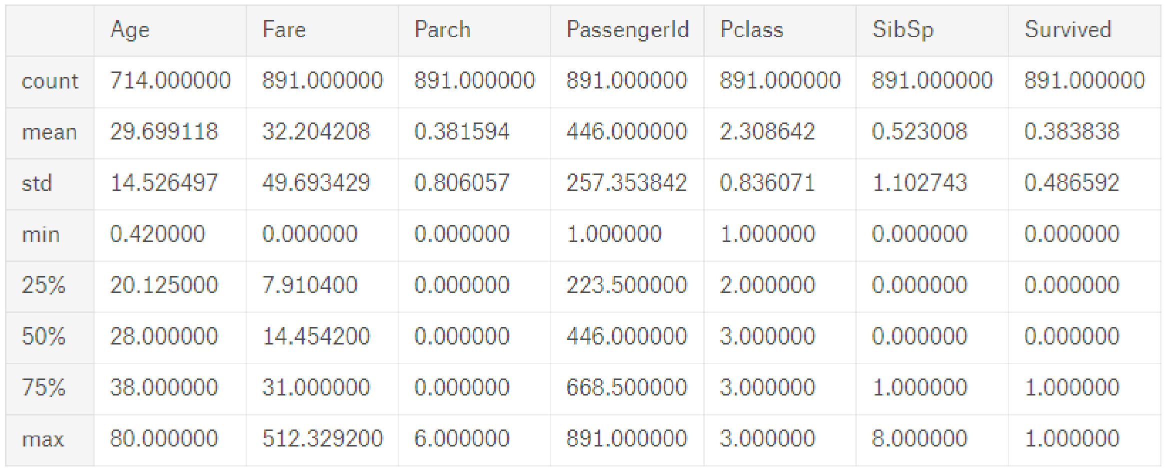
We've got a sense of our variables, their class type, and the first few observations of each. We know we're working with 1309 observations of 12 variables. To make things a bit more explicit since a couple of the variable names aren't 100% illuminating, here's what we've got to deal with:

Survived: Survived (1) or died (0)

- Pclass: Passenger's class
- Name: Passenger's name
- Sex: Passenger's sex
- Age: Passenger's age
- SibSp: Number of siblings/spouses aboard
- Parch: Number of parents/children aboard
- Ticket: Ticket number
- Fare: Fare

- Cabin: Cabin
- Embarked: Port of embarkation

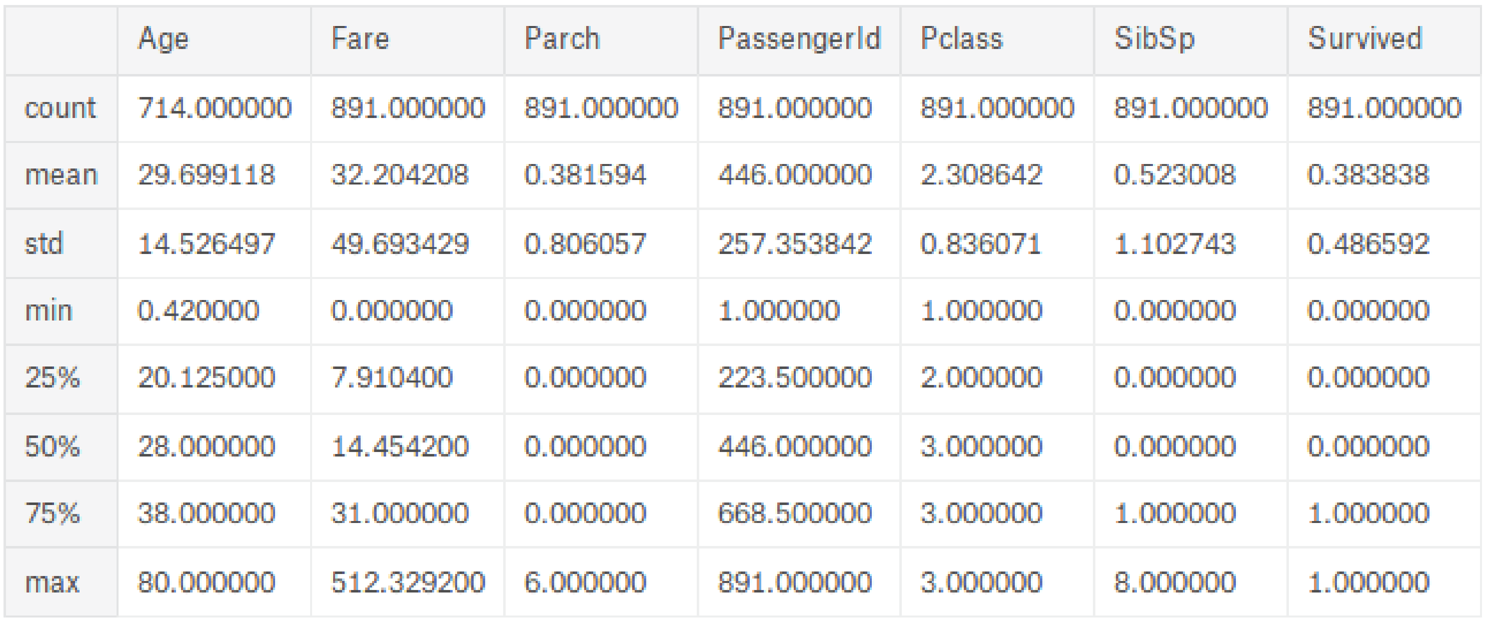
Titanic.describe()



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

A numeric variable is one with values of integers or real numbers while a categorical variable is a variable that can take on one of a limited, and usually fixed, number of possible values, such as blood type.

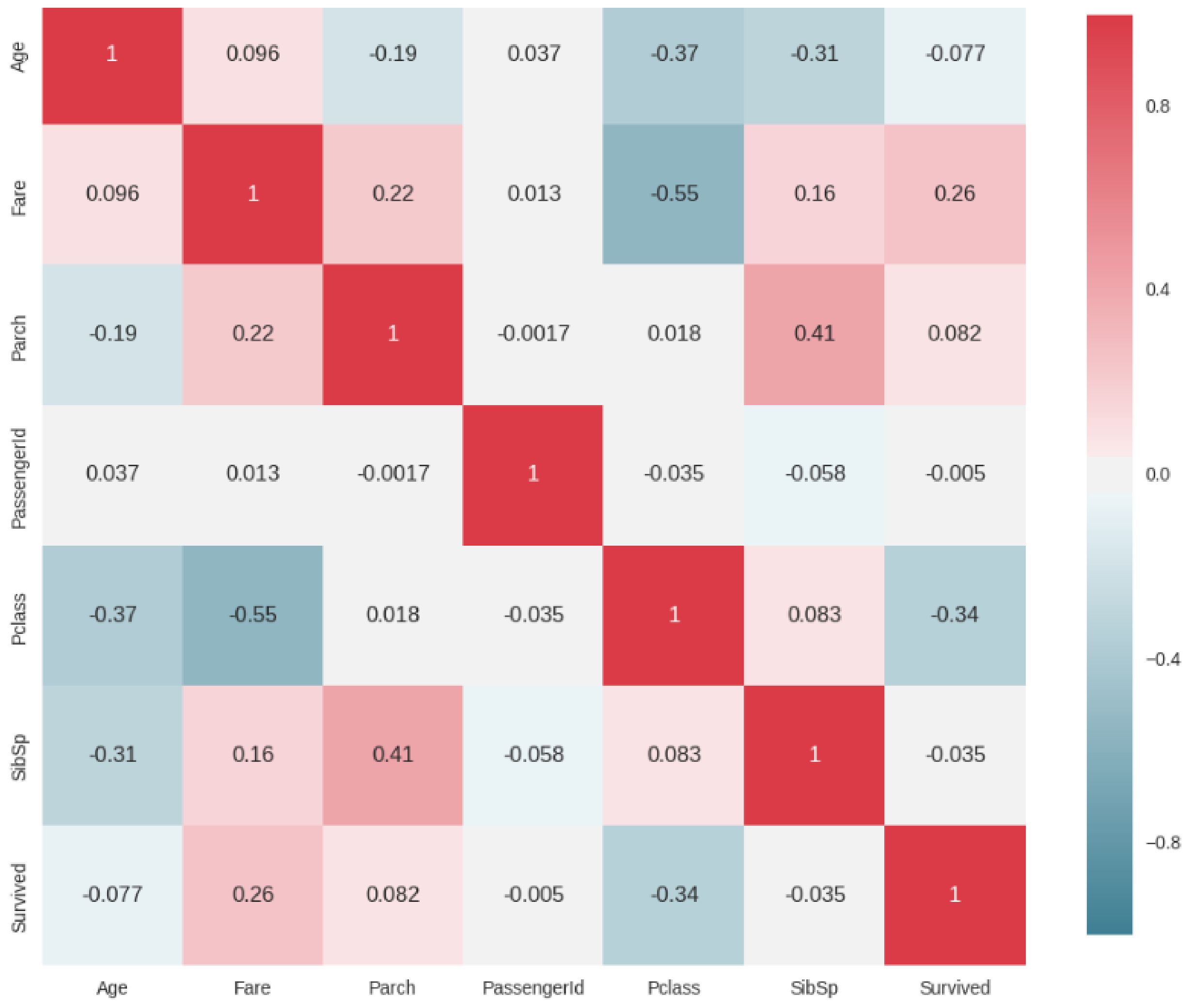
We need to notice especially what type of variable each is, how many observations there are and some of the variable values.



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

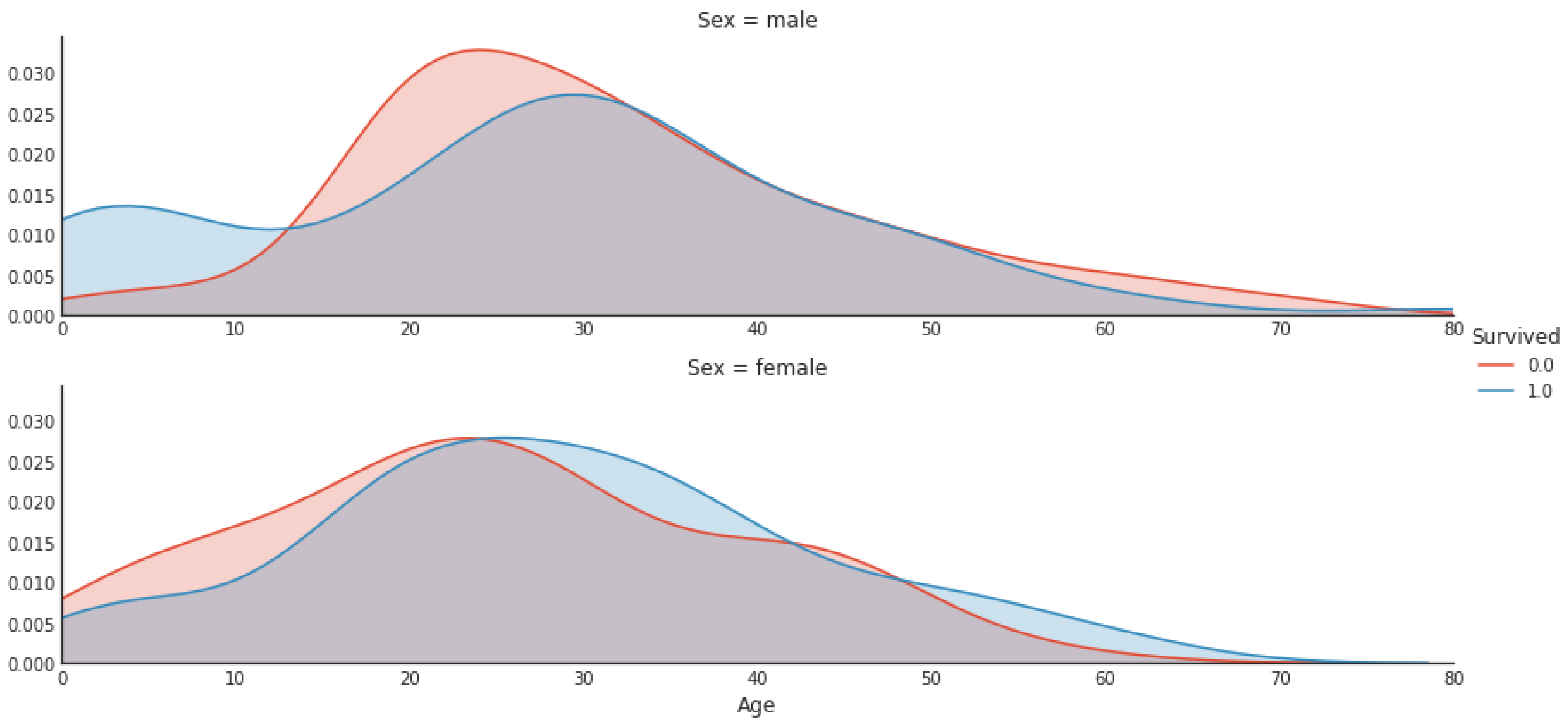
A heat map of correlation may give us a understanding of which variables are important

Plot_correlation_map(titanic)



Consider the graphs below. Differences between survival for different values is what will be used to separate the target variable (survival in this case) in the model. If the two lines had been about the same, then it would not have been a good variable for our predictive model.

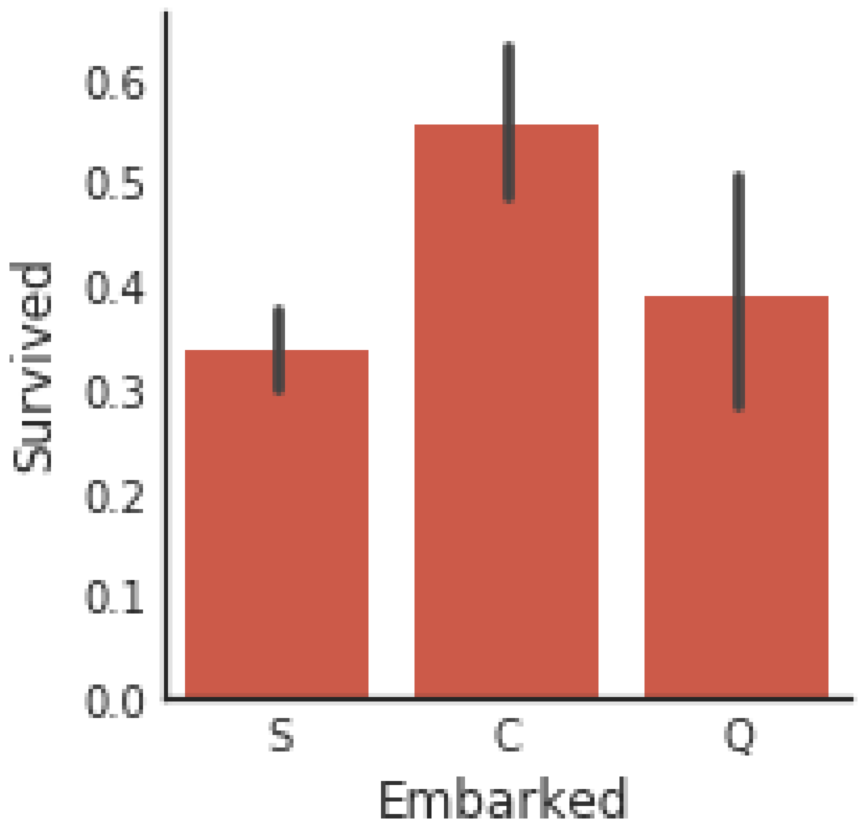
plot_distribution(titanic , var = 'Age' , target = 'Survived' , row = 'Sex')



We can also look at categorical variables like Embarked and their relationship with survival.

- C = Cherbourg
- Q = Queenstown
- S = Southampton

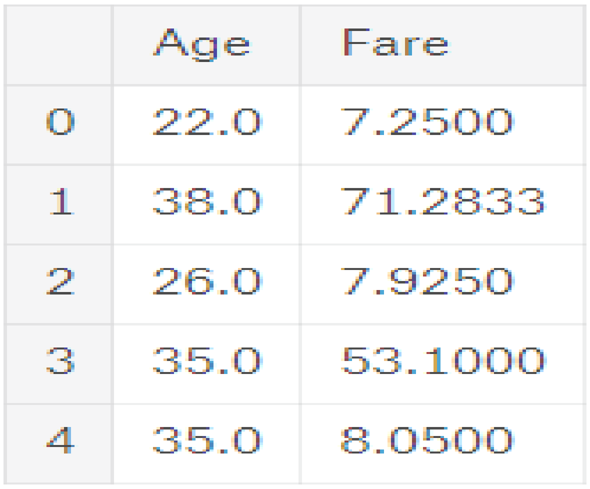
plot_categories(titanic , cat = 'Embarked' , target = Survived')



Fill missing values in variables

Most machine learning algorithms require all variables to have values in order to use it for training the model. The simplest method is to fill missing values with the average of the variable across all observations in the training set.

```
# Create dataset

imputed = pd.DataFrame()



# Fill missing values of Age with the average of Age (mean)

imputed[ 'Age' ] = full.Age.fillna( full.Age.mean() )



# Fill missing values of Fare with the average of Fare (mean)

imputed[ 'Fare' ] = full.Fare.fillna( full.Fare.mean() )



imputed.head()
```



	Age	Fare
0	22.0	7.2500
1	38.0	71.2833
2	26.0	7.9250
3	35.0	53.1000
4	35.0	8.0500

Create datasets

```
# Create all datasets that are necessary to train, validate and test models

train_valid_X = full_X[ 0:891 ]

train_valid_y = titanic.Survived

test_X = full_X[ 891: ]

train_X , valid_X , train_y , valid_y = train_test_split( train_valid_X ,
train_valid_y , train_size = .7 )



print (full_X.shape , train_X.shape , valid_X.shape , train_y.shape ,
valid_y.shape , test_X.shape)
```

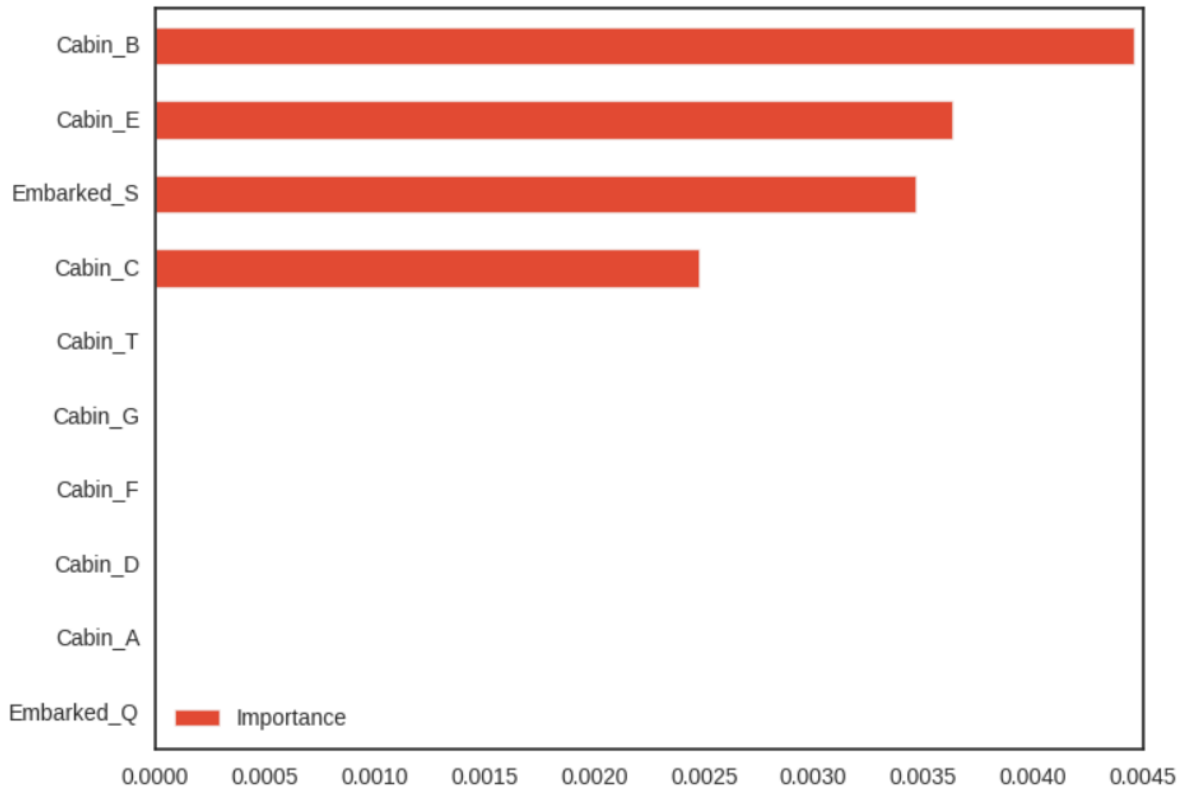
```
(1309, 15) (623, 15) (268, 15) (623,) (268,) (418, 15)
```

Feature importance:

Selecting the optimal features in the model is important. We will now try to evaluate what the most important variables are for the model to make the prediction.

```
plot_variable_importance(train_X, train_y)
```

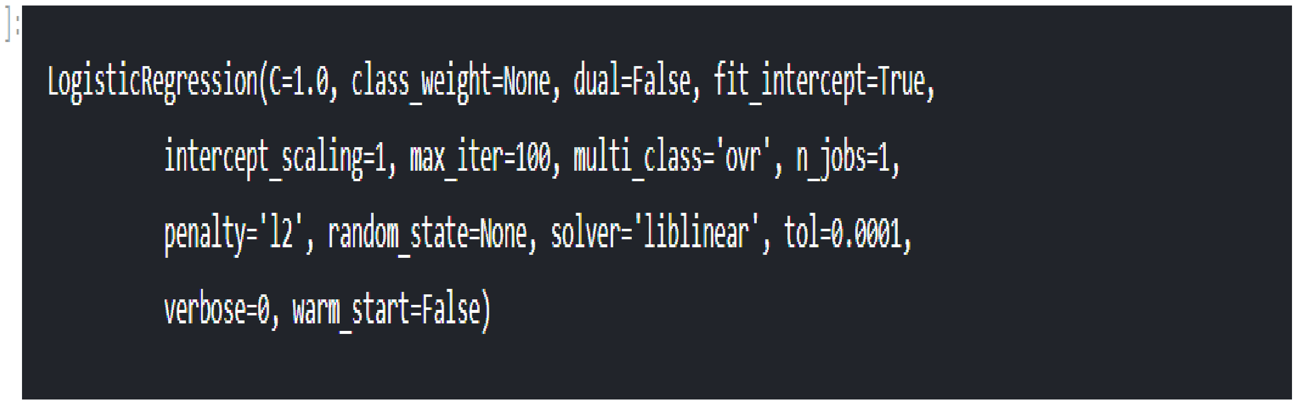
```
0.991974317817
```



Train the selected model

When you have selected a dataset with the features you want and a model you would like to try it is now time to train the model. After all our preparation model training is simply done with the one line below.

model.fit(train_X , train_y)



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

Evaluation:

We can evaluate the accuracy of the model by using the validation set where we know the actual outcome. This data set have not been used for training the model, so it's completely new to the model.

We then compare this accuracy score with the accuracy when using the model on the training data. If the difference between these are significant this is an indication of overfitting. We try to avoid this because it means the model will not generalize well to new data and is expected to perform poorly.

- Predict if a passenger survived the sinking of the Titanic or not. For each PassengerId in the test set, you must predict a 0 or 1 value for the Survived variable.
- submit a csv file with exactly 418 entries plus a header row. The submission will show an error if have extra columns (beyond PassengerId and Survived) or rows.

The file should have exactly 2 columns:

- PassengerId (sorted in any order)
- Survived (contains your binary predictions: 1 for survived, 0 for deceased)

- PassengerId Survived

```
 892              0
 893              1
 894              0
 Etc.
```

# Score the model

```
print (model.score( train_X , train_y ) , model.score( valid_X , valid_y ))
```

```
0.794542536116 0.772388059701
```

Automagic

It's also possible to automatically select the optimal number of features and visualize this. This is uncommented and can be tried in the competition part of the tutorial.

```
rfecv = RFECV( estimator = model , step = 1 , cv = StratifiedKFold( train_y , 2 ) , scoring = 'accuracy' )

rfecv.fit( train_X , train_y )



#print (rfecv.score( train_X , train_y ) , rfecv.score( valid_X , valid_y ))

#print( "Optimal number of features : %d" % rfecv.n_features_ )
```

# Plot number of features VS. cross-validation scores

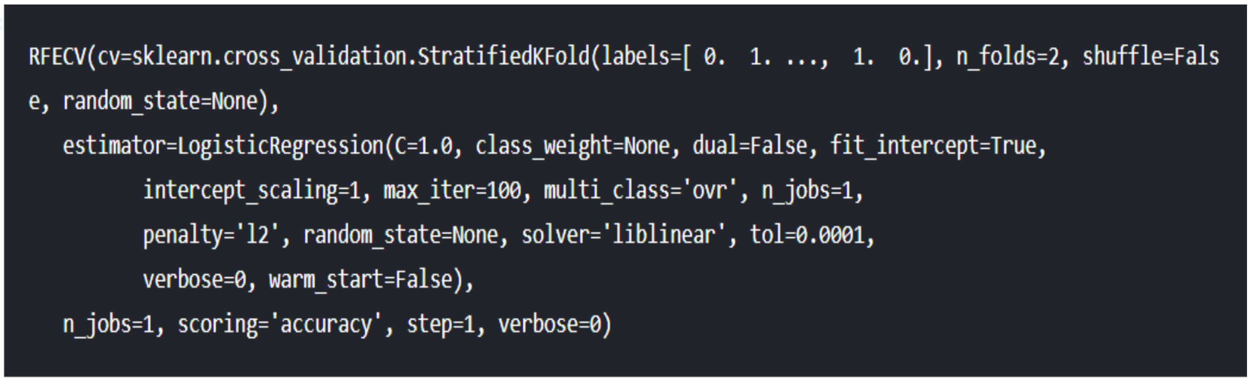
#plt.figure()

#plt.xlabel("Number of features selected")

#plt.ylabel("Cross validation score (nb of correct classifications)")

#plt.plot(range(1 , len(rfecv.grid_scores_) + 1) , rfecv.grid_scores_)

#plt.show()



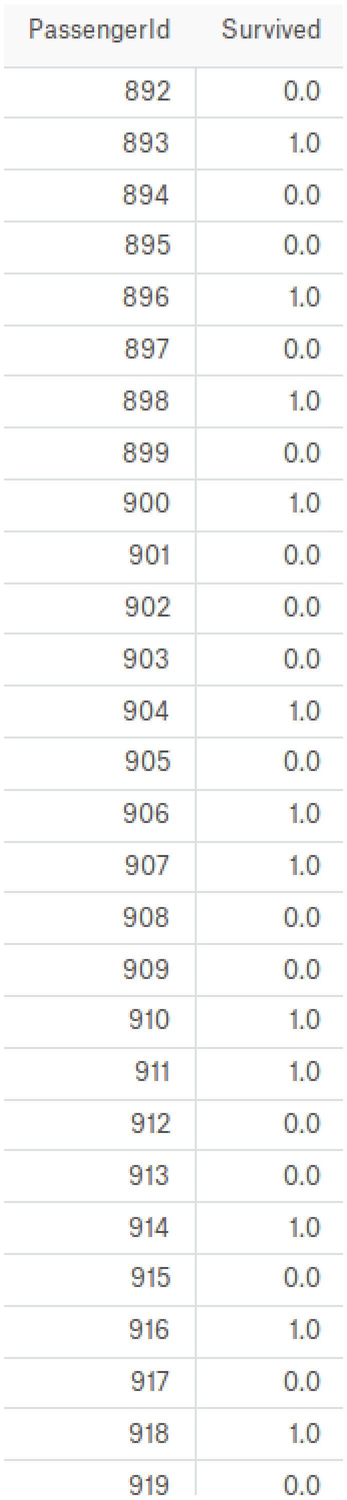
```
RFECV(cv=sklearn.cross_validation.StratifiedKFold(labels=[ 0.  1. ...,  1.  0.], n_folds=2, shuffle=Fals
e, random_state=None),
   estimator=LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False),
   n_jobs=1, scoring='accuracy', step=1, verbose=0)
```

HOW TO PRINT THE OUTPUT:

```
test_Y = model.predict( test_X )

passenger_id = full[891:].PassengerId

test = pd.DataFrame( { 'PassengerId': passenger_id , 'Survived': test_Y } )

test.shape

test.head()

test.to_csv( 'titanic_pred.csv' , index = False )
```

This will print the output to the path specifies above.

OUTPUT:



PassengerId	Survived
892	0.0
893	1.0
894	0.0
895	0.0
896	1.0
897	0.0
898	1.0
899	0.0
900	1.0
901	0.0
902	0.0
903	0.0
904	1.0
905	0.0
906	1.0
907	1.0
908	0.0
909	0.0
910	1.0
911	1.0
912	0.0
913	0.0
914	1.0
915	0.0
916	1.0
917	0.0
918	1.0
919	0.0

TUTORIAL :-

- Titanic Data Science Solutions Python Notebook
- Use pandas for data manipulation
- Use matplotlib and seaborn for data visualization
- Learn to build models with scikit-learn
- These Python kernels cover more advanced techniques and complex approaches:
- An Interactive Data Science Tutorial
- Get familiar with using Jupyter notebooks
- Learn the importance of feature selection in machine learning
- Machine Learning from Start to Finish with Scikit-Learn
- Use cross-validation to make sure your model generalizes to new data (i.e., it doesn’t “overfit”)
- Use parameter tuning and grid search to select the
-
- best performing model out of several different classification algorithms
- An Introduction to Ensembling/Stacking in Python:
- Use the fundamental skill of “Ensembling” to combine the predictions of several models
- The basics of feature engineering and data visualization
- How to deal with missing values in the dataset
- How to train a random forest classifier to make a prediction