Assignment 3: Part 1 | Titanic

David Thor - Practical Maching Learning and AI (MSDS 422 - Winter 2022)

Part 0 - Importing Data

In [1]:

```
# What does this do? This will render HTML content.
from IPython.display import HTML
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
```

In [2]:

```
# note the use of 'consensual' package nicknames.
import os
import numpy as np
import pandas as pd
import math
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn import linear_model
from scipy import stats
import matplotlib.gridspec as gridspec
from sklearn.compose import ColumnTransformer
from sklearn.linear_model import Ridge
from sklearn.model_selection import KFold
from sklearn.linear_model import Lasso
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RepeatedKFold
from sklearn.linear_model import ElasticNet
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import LogisticRegression
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.neighbors import KNeighborsClassifier
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

In [3]:

```
##Importing Data
titanic_trainDat = pd.read_csv('train.csv')
titanic_testDat =  pd.read_csv('test.csv')
```

In [4]:

```
titanic_trainDat.shape
titanic_testDat.shape
```

Out[4]:

```
(891, 12)
```

Out[4]:

```
(418, 11)
```

In [5]:

```
# checking variables to make sure survivied is the y variable
set(titanic_trainDat.columns).difference(set(titanic_testDat.columns))
```

Out[5]:

```
{'Survived'}
```

In [6]:

```
# Are the features in the test data a subset of the train data features?
set(titanic_testDat.columns).issubset(set(titanic_trainDat.columns))
```

Out[6]:

```
True
```

In [7]:

```
## Understanding the data
titanic_trainDat.info()
titanic_trainDat.head(3)
titanic_trainDat.describe()
```

```
<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
```

Out[7]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S

Out[7]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208

std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

In [8]:

```
## Isolating numerical and categorical variables for potential use later
titanic_categoricals = titanic_trainDat[['Survived','Pclass','Sex',\
                                         'Ticket','Cabin','Embarked']]
titanic_numericals = titanic_trainDat[['Age','SibSp','Parch','Fare']]
```

In [9]:

```
#Commented for cleaner PDF export

# Understanding data distributions
#import warnings
#warnings.filterwarnings("ignore")

#for i in titanic_numericals:
    #plt.hist(titanic_numericals[i])
    #plt.title(i)
    #plt.show()
```

In [10]:

```
#Commeneted for cleaner PDF export

## Understanding data distributions of categoricals
#for i in titanic_categoricals:
    #sns.barplot(titanic_categoricals[i].value_counts().index,\
               #titanic_categoricals[i].value_counts()).set_title(i)
    #plt.show()
```

EDA

- After some initial analysis of the data, below is my plan.
  - Drop unnecessary columns (PassengerID, Name)
  - I am also going to drop ticket as I don't think it has any revalence. Data seems rather sporatic.
  - I am going to feature engineer Cabin section (letter)
  - Going to impute numerical data with either mean/median
  - Going to impute most categorical with mode

In [11]:

```
total = titanic_trainDat.isnull().sum().sort_values(ascending = False)
pcg = (total / titanic_trainDat.isnull().count()).sort_values\
(ascending = False)
miss_val = pd.concat([total, pcg], axis = 1, keys = ['Total', 'Percentage'])
miss_val.head(20)
```

Out[11]:

	Total	Percentage
Cabin	687	0.771044
Age	177	0.198653
Embarked	2	0.002245
PassengerId	0	0.000000
Survived	0	0.000000
Pclass	0	0.000000
Name	0	0.000000
Sex	0	0.000000
SibSp	0	0.000000
Parch	0	0.000000
Ticket	0	0.000000
Fare	0	0.000000

In [12]:

```
## Created a function for easier use later with test data
def titanic_eda(df):
    df['Cabin']=df['Cabin'].fillna('NA')
    #Feature Engineer a new variable with only cabin area(letter)
    df['cabinLetter'] = df['Cabin'].str[:1]
    df['Embarked']=df['Embarked'].fillna(df['Embarked'].mode()[0])
    df['Pclass']=df['Pclass'].fillna(df['Pclass'].mode()[0])
    df['Sex']=df['Sex'].fillna(df['Sex'].mode()[0])
    df['SibSp']=df['SibSp'].fillna(df['SibSp'].median())
    df['Parch']=df['Parch'].fillna(df['Parch'].median())
    df['Age']=df['Age'].fillna(df['Age'].median())
    df['Fare']=df['Fare'].fillna(df['Fare'].mean())
    df.drop(['PassengerId','Ticket','Name','Cabin'],axis=1,inplace=True)
    return(df)
```

In [13]:

```
titanic_trainDat = titanic_eda(titanic_trainDat)
```

In [14]:

```
sum(titanic_trainDat.isnull().sum())
```

Out[14]:

```
0
```

In [15]:

```
titanic_trainDat.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    object
 3   Age          891 non-null    float64
 4   SibSp        891 non-null    int64
 5   Parch        891 non-null    int64
 6   Fare         891 non-null    float64
 7   Embarked     891 non-null    object
 8   cabinLetter  891 non-null    object
dtypes: float64(2), int64(4), object(3)
memory usage: 62.8+ KB
```

Transformation

- I have selected the appropriate numericals and categoricals to standardize and encode

In [16]:

```
### Encoding
# Col transform specs

titanic_ct = ColumnTransformer([('standardized',\
                                 preprocessing.StandardScaler(),[2,3,4,5]),
       ('oneHotter', preprocessing.OneHotEncoder\
        (handle_unknown='ignore'),[0,1,6,7])])

titanic_ct
```

Out[16]:

```
ColumnTransformer(transformers=[('standardized', StandardScaler(),
                                 [2, 3, 4, 5]),
                                ('oneHotter',
                                 OneHotEncoder(handle_unknown='ignore'),
                                 [0, 1, 6, 7])])
```

In [17]:

```
#Separate the target from the variables
titanic_y=titanic_trainDat.Survived.to_numpy(copy=True)
titanic_X=titanic_trainDat.loc[:,titanic_trainDat.columns!='Survived']\
.to_numpy(copy=True)
titanic_X.shape   # size
titanic_y.shape   # size
#titanic_X
```

Out[17]:

```
(891, 8)
```

Out[17]:

```
(891,)
```

In [18]:

```
#For easier accuracy calcuations.
#Selected this metric for performance measurement
from sklearn.metrics import accuracy_score
```

In [19]:

```
# Random split using a scikit-learn preprocessing method
# I selected 85% because I want a small yet reasonable for final_test
# 134 is a good 'final exam' size to validate my ensemble against
```

```
Xtrain, Xtest, ytrain, ytest = train_test_split(titanic_X, titanic_y, \
        train_size=0.85, random_state=9)

Xtrain.shape
Xtest.shape
ytrain.shape
ytest.shape
```

Out[19]:

```
(757, 8)
```

Out[19]:

```
(134, 8)
```

Out[19]:

```
(757,)
```

Out[19]:

```
(134,)
```

Modeling Summary

- ### Random Forest
  - First off, my base RF model accuracy for the training data was 98%. The base model RF for the test data was 77%
  - This indicated that there was some overfitting.
  - Secondly, I used RandomizedSearchCV to randomly select hyperparameters for a basis to start hyperparameter tuning. This was randomly testing ~4320 combination of hyperparameter settings.
  - The best random parameter model's accuracy score for the training and test data were 87% and 78%, respectively.
  - Secondly, I used GridSearchCV to further tune the 'best' hyperparameters from the RandomSearchCV to see if it made a difference.
  - The GridSearchCV hyperparameters yield an accuracy score for the training and testing data were 86% and 80%.
  - In short, I couldn't get much improvement from my results from tuning the RF model. I thought maybe this was the limitation (due to my EDA, etc.) or RF wasn't just a good model for this dataset. I proceeded to Gradient Boosting.
- ### Gradient Boosting
  - Since I spent sometime analyzing some of the parameters for the RF model, I manually selected certain intervals to for my GridSearchCV parameters.
  - One key parameter I was sure to test out was the learning rate as this differs from the RF model
  - The accuracy from my Gradient Boosting model was 87% for the training data, and 81%
  - I was happy with the results as this indicating an improvement in score without overfitting/leakage.

In [20]:

```
# I wanted to look at the possible hyperparatmers to test with
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(random_state = 10)
from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(rf.get_params())
```

```
Parameters currently in use:

{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
```

```
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': 10,
 'verbose': 0,
 'warm_start': False}
```

In [21]:

```
#After reading/understanding some of the hyperparameters, I was ready to test.
#I used RandomizedSearchCV, this will randmoly select different combinations
#  of hyper parameters to test
#This randomly test out 4320 settings

from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in \
                np.linspace(start = 200, stop = 2000, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
pprint(random_grid)
```

```
{'bootstrap': [True, False],
 'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, None],
 'max_features': ['auto', 'sqrt'],
 'min_samples_leaf': [1, 2, 4],
 'min_samples_split': [2, 5, 10],
 'n_estimators': [200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000]}
```

In [22]:

```
# Use the random grid to search for best hyperparameters
# First create the base model to tune
rf = RandomForestClassifier()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = rf, param_distributions = \
                               random_grid, n_iter = 100, cv = 3, \
                               verbose=2, random_state=42, n_jobs = -1)
# Fit the random search model
```

```
rf_random.fit(titanic_ct.fit_transform(Xtrain), ytrain)
```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits
```

Out[22]:

```
RandomizedSearchCV(cv=3, estimator=RandomForestClassifier(), n_iter=100,
                   n_jobs=-1,
                   param_distributions={'bootstrap': [True, False],
                                        'max_depth': [10, 20, 30, 40, 50, 60,
                                                      70, 80, 90, 100, 110,
                                                      None],
                                        'max_features': ['auto', 'sqrt'],
                                        'min_samples_leaf': [1, 2, 4],
                                        'min_samples_split': [2, 5, 10],
                                        'n_estimators': [200, 400, 600, 800,
                                                         1000, 1200, 1400, 1600,
                                                         1800, 2000]},
                   random_state=42, verbose=2)
```

In [23]:

```
# These were the best hyper parameters from the random 4320 settings.
rf_random.best_params_
```

Out[23]:

```
{'n_estimators': 400,
 'min_samples_split': 10,
 'min_samples_leaf': 4,
 'max_features': 'auto',
 'max_depth': 70,
 'bootstrap': True}
```

In [49]:

```
# Below is a base RF model
base_model = RandomForestClassifier(random_state = 42)
base_model_fit = base_model.fit(titanic_ct.fit_transform(Xtrain), ytrain)
y_pred_base_train = base_model_fit.predict(titanic_ct.transform(Xtrain))
base_accuracy_trained_data = accuracy_score(ytrain,y_pred_base_train)
print (f'Base Model Training Data Accuracy: ',base_accuracy_trained_data)
# It's important to see how this base model compares to hypertuned models
y_pred_test_base = base_model_fit.predict(titanic_ct.transform(Xtest))
base_accuracy_test_data = accuracy_score(ytest,y_pred_test_base)
print (f'Base Model Final Exam Data Accuracy: ', base_accuracy_test_data)
```

```
Base Model Training Data Accuracy:  0.9841479524438573
Base Model Final Exam Data Accuracy:  0.7761194029850746
```

In [52]:

```
# I am now fitting the best randomized parameters from earlier.
best_random_model = rf_random.best_estimator_
best_random_model_fit = \
best_random_model.fit(titanic_ct.fit_transform(Xtrain), ytrain)
y_pred_trained_random = best_random_model_fit.predict\
(titanic_ct.transform(Xtrain))
accuracy_trained_data_random = accuracy_score(ytrain,\
                                               y_pred_trained_random)
print (f'Best Random Model Training Data Accuracy: ', \
       accuracy_trained_data_random)
#This will show us the accuracy against the test/final exam data
y_pred_test_random = best_random_model_fit.predict(\
                                         titanic_ct.transform(Xtest))
accuracy_test_data_random = accuracy_score(ytest,y_pred_test_random)
print (f'Best Random Model Final Exam Data Accuracy: ', \
```

```
        accuracy_test_data_random)
```

```
Best Random Model Training Data Accuracy:  0.869220607661823
Best Random Model Final Exam Data Accuracy:  0.7835820895522388
```

In [26]:

```python
# After analyzing some of the hyper parameters from the model above
# I decided to further tune the parameters in hopes to better the model
# I had selected a cross-fold of 3 because with limited amount of records(800)
# I want to make sure the validation records is sufficient.
from sklearn.model_selection import GridSearchCV
# Create the parameter grid based on the results of random search
param_grid = {
    'bootstrap': [True],
    'max_depth': [80, 90, 100, 110],
    'max_features': ['sqrt'],
    'min_samples_leaf': [3, 4, 5],
    'min_samples_split': [2, 4, 8],
    'n_estimators': [1000, 1400, 1800, 2200]
}
# Create a based model
rf = RandomForestClassifier()
# Instantiate the grid search model
grid_search = GridSearchCV(estimator = rf, param_grid = param_grid,
                          cv = 3, n_jobs = -1, verbose = 2)
```

In [27]:

```python
#This will fit and show the best parameters for the RF model
grid_search.fit(titanic_ct.fit_transform(Xtrain), ytrain)
grid_search.best_params_
```

```
Fitting 3 folds for each of 144 candidates, totalling 432 fits
```

Out[27]:

```
GridSearchCV(cv=3, estimator=RandomForestClassifier(), n_jobs=-1,
             param_grid={'bootstrap': [True], 'max_depth': [80, 90, 100, 110],
                         'max_features': ['sqrt'],
                         'min_samples_leaf': [3, 4, 5],
                         'min_samples_split': [2, 4, 8],
                         'n_estimators': [1000, 1400, 1800, 2200]},
             verbose=2)
```

Out[27]:

```
{'bootstrap': True,
 'max_depth': 80,
 'max_features': 'sqrt',
 'min_samples_leaf': 5,
 'min_samples_split': 4,
 'n_estimators': 1400}
```

In [90]:

```python
# This will show predict and show us the accuracy of the training/test data.
best_grid = grid_search.best_estimator_
best_grid_fit = best_grid.fit(titanic_ct.fit_transform(Xtrain), ytrain)
y_pred_best_train = best_grid_fit.predict(titanic_ct.transform(Xtrain))
accuracy_trained_data_best = accuracy_score(ytrain,y_pred_best_train)
print (f'Best Parameter Model Training Data Accuracy: ', \
       accuracy_trained_data_best)

y_pred_best_test = best_grid_fit.predict(titanic_ct.transform(Xtest))
accuracy_test_data_best = accuracy_score(ytest,y_pred_best_test)
```

```
print (f'Best Parameter Model Final Exam Data Accuracy: ', \
       accuracy_test_data_best)
```

```
Best Parameter Model Training Data Accuracy:  0.8586525759577279
Best Parameter Model Final Exam Data Accuracy:  0.7985074626865671
```

Gradient Boosting Section

In [29]:

```
##Gradient Boosting possible parameters to tune
from sklearn.ensemble import GradientBoostingClassifier
gbc = GradientBoostingClassifier(random_state = 10)
from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(gbc.get_params())
```

```
Parameters currently in use:

{'ccp_alpha': 0.0,
 'criterion': 'friedman_mse',
 'init': None,
 'learning_rate': 0.1,
 'loss': 'deviance',
 'max_depth': 3,
 'max_features': None,
 'max_leaf_nodes': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_iter_no_change': None,
 'random_state': 10,
 'subsample': 1.0,
 'tol': 0.0001,
 'validation_fraction': 0.1,
 'verbose': 0,
 'warm start': False}
```

In [30]:

```
# Instead of trying the randomsearchCV, I am going to just test parameters
# I have an idea which ones to tune.
# I selected cross fold of 3 because with only ~700 records,
# I want to make sure number of records are sufficient for the validation.
from sklearn.model_selection import GridSearchCV
param_grid = {
    "loss":["deviance"],
    "learning_rate": [0.01, 0.05, 0.1, 0.15, 0.2],
    "min_samples_split": [2, 4, 8],
    "min_samples_leaf": [3, 4, 5],
    "max_depth":[3,5,8],
    "max_features":["log2","sqrt"],
    "criterion": ["friedman_mse",  "mae"],
    "subsample":[0.5, 0.8, 0.9, 1.0],
    "n_estimators":[8,10,12]
}
# Create a based model
gbc = GradientBoostingClassifier()
# Instantiate the grid search model
```

```
grid_search = GridSearchCV(estimator = gbc, param_grid = param_grid,
                           cv = 3, n_jobs = -1, verbose = 2)
```

In [31]:

```
# The Best parameters for the gradient boosted model
grid_search.fit(titanic_ct.fit_transform(Xtrain), ytrain)
grid_search.best_params_
```

```
Fitting 3 folds for each of 6480 candidates, totalling 19440 fits
```

Out[31]:

```
GridSearchCV(cv=3, estimator=GradientBoostingClassifier(), n_jobs=-1,
             param_grid={'criterion': ['friedman_mse', 'mae'],
                         'learning_rate': [0.01, 0.05, 0.1, 0.15, 0.2],
                         'loss': ['deviance'], 'max_depth': [3, 5, 8],
                         'max_features': ['log2', 'sqrt'],
                         'min_samples_leaf': [3, 4, 5],
                         'min_samples_split': [2, 4, 8],
                         'n_estimators': [8, 10, 12],
                         'subsample': [0.5, 0.8, 0.9, 1.0]},
             verbose=2)
```

Out[31]:

```
{'criterion': 'friedman_mse',
 'learning_rate': 0.15,
 'loss': 'deviance',
 'max_depth': 5,
 'max_features': 'log2',
 'min_samples_leaf': 4,
 'min_samples_split': 4,
 'n_estimators': 12,
 'subsample': 0.8}
```

In [80]:

```
# This will fit, predict and show accuracy for the trainng/test data
best_gbc_grid = grid_search.best_estimator_
best_gbc_grid_fit = best_gbc_grid.fit(titanic_ct.fit_transform(Xtrain)\
                                      , ytrain)
y_pred_gbc_train = best_gbc_grid_fit.predict(titanic_ct.transform(Xtrain))
accuracy_trained_data_gbc = accuracy_score(ytrain,y_pred_gbc_train)
print (f'Best GradientBoosting Parameter Model Training Data Accuracy: ', \
       accuracy_trained_data_gbc)

y_pred_test_gbc = best_gbc_grid_fit.predict(titanic_ct.transform(Xtest))
accuracy_test_data_gbc = accuracy_score(ytest,y_pred_test_gbc)
print (f'Best GradientBoosting Parameter Model Final Exam Data Accuracy: ', \
       accuracy_test_data_gbc)
```

```
Best GradientBoosting Parameter Model Training Data Accuracy:  0.8665785997357992
Best GradientBoosting Parameter Model Final Exam Data Accuracy:  0.8059701492537313
```

Implement/Submission

- I have decided to use the gradient boosting model because it had the highest 'test/final exam' accuracy score

In [81]:

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

In [82]:

```
# Called the EDA function above before transformations
```

```
titanic_testDat = titanic_eda(titanic_testDat)
```

In [83]:

```
sum(titanic_testDat.isnull().sum())
```

Out[83]:

```
0
```

In [84]:

```
titanic_final_exam=titanic_testDat.to_numpy(copy=True)
```

In [85]:

```
# Predict results from the gradient boosted model
y_pred = best_gbc_grid_fit.predict(titanic_ct.transform(titanic_final_exam))
```

In [86]:

```
df_test = pd.read_csv('test.csv')
Regresult = pd.DataFrame(y_pred, columns=['Survived'])
df_test['PassengerId'].shape, Regresult.shape
test_t = pd.DataFrame(df_test["PassengerId"])
```

Out[86]:

```
((418,), (418, 1))
```

In [87]:

```
my_submission = pd.concat([test_t, Regresult ], axis=1)
```

In [88]:

```
my_submission
```

Out[88]:

	PassengerId	Survived
0	892	0
1	893	0
2	894	0
3	895	0
4	896	0
...	...	...
413	1305	0
414	1306	1
415	1307	0
416	1308	0
417	1309	0

418 rows × 2 columns

In [89]:

```
my_submission.to_csv('Assignment3-Titanic.csv', index=False)
```