Supervised Learning Capstone: Predicting Survival on the Titanic

By Jacob Knopping

The Sinking of the Titanic

- RMS Titanic was a British passenger liner
  - Largest ship of its time
- The ship struck an iceberg during its maiden voyage from Southampton to NYC
- Eventually splitting in half and sinking to the ocean floor
  - early morning hours of April 15, 1912 in the North Atlantic Ocean
- ~2,224 passengers and crew were onboard
  - ~1,500 passengers died
  - Marks one of modern history's deadliest commercial disasters
- The shipwreck was discovered in 1985 during a US military mission

Link

About the Data

- Dataset was obtained from Kaggle (link)
  - Provides training and test data
- Variables can be defined as follows:
  - PassengerId: unique id for each passenger
  - Survived: 1= survived, 0 = did not survive
  - Pclass: passenger class
  - Name: the name of the passenger
  - Sex: the sex of the passenger
  - Age: passenger age
  - ***SibSp: number of siblings and spouses***
  - ***ParCh: number of parents and children***
  - Ticket: ticket number
  - Fare: price paid for ticket
  - Cabin: the cabin number of the passenger
  - ***Embarked***: where the passenger boarded from (S = Southampton; England; C = Cherbourg, France; Q = Queenstown, Ireland)

Goal

- Our goal: build a classifier model, using supervised learning
  - Build multiple models
    - Decision Tree
    - Random Forest Classifier
    - KNN Classifier
    - Support Vector Machine
    - Gradient Boosting Classifier
  - Select the model with the best results
- Predict whether or not a passenger would survive the Titanic, using the training data

Data Cleaning

```
percent_miss = 100*titanic_df.isnull().sum()/len(titanic_df)
print(percent_miss)
```

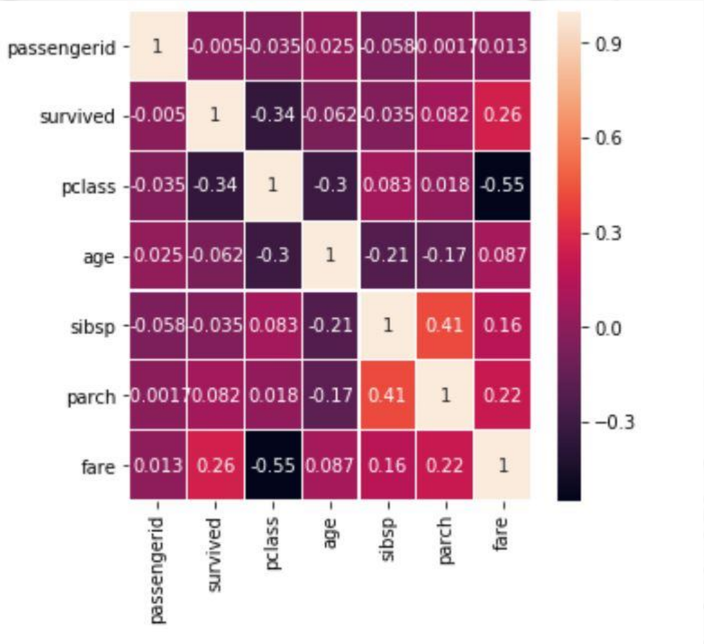
```
passengerid     0.000000
survived        0.000000
pclass          0.000000
name            0.000000
sex             0.000000
age            19.865320
sibsp           0.000000
parch           0.000000
ticket          0.000000
fare            0.000000
cabin          77.104377
embarked        0.224467
dtype: float64
```

```
#Dealing with missing values

#Drop cabin, interpolate age, fill embarked with mode
for dataset in datasets:
    dataset.drop(columns = 'cabin', inplace=True)
    dataset.interpolate(inplace=True)
    dataset.fillna(dataset.mode().iloc[0], inplace=True)

print(titanic_df.isnull().sum())
```

```
passengerid    0
survived       0
pclass         0
name           0
sex            0
age            0
sibsp          0
parch          0
ticket         0
fare           0
embarked       0
dtype: int64
```

- ‘Cabin’ variable was missing 77% of data → dropped
- ‘Age’ variable was missing 19.9% of data → filled using interpolation
- ‘Embarked’ variable was only missing two data points → filled using mode

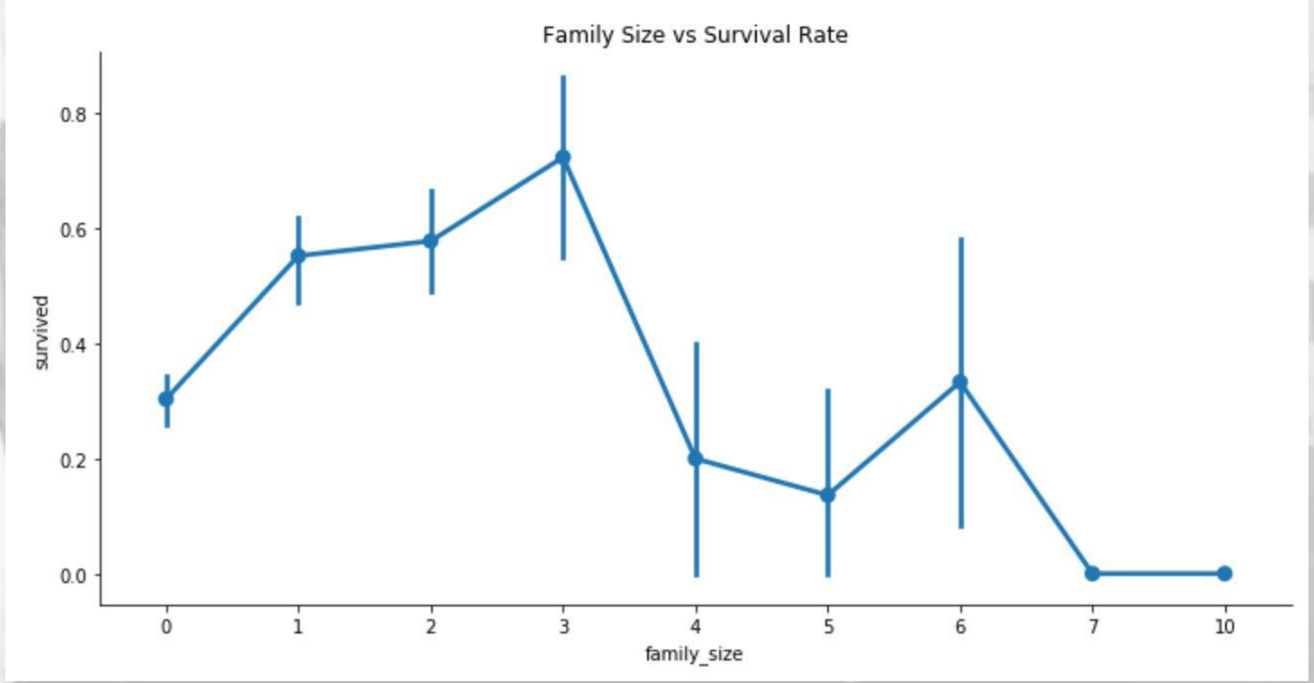
EDA and Feature Engineering (pt. 1)



- Correlations do not appear to be particularly high for this dataset
- ‘Fare’ and ‘pclass’ have some negative correlation (-0.55)
- ‘Sibsp’ and ‘parch’ have a somewhat positive correlation (0.41)
  - Combine the ‘parch’ and ‘sibsp’ features into a feature ‘family_size’
  - Remember: ‘parch’ is the number of parents/children, and ‘sibsp’ is the number of siblings/spouses

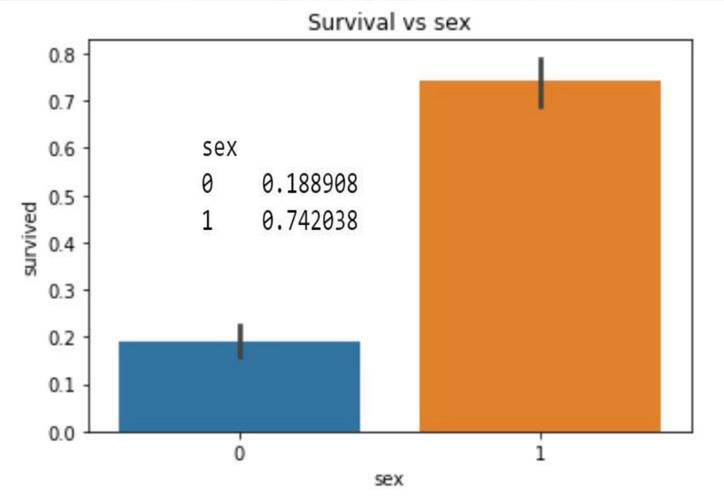
EDA and Feature Engineering (pt. 2)

```
#Create a new feature that combines parch and sibsp
for dataset in datasets:
    dataset['family_size'] = dataset.sibsp + dataset.parch
```

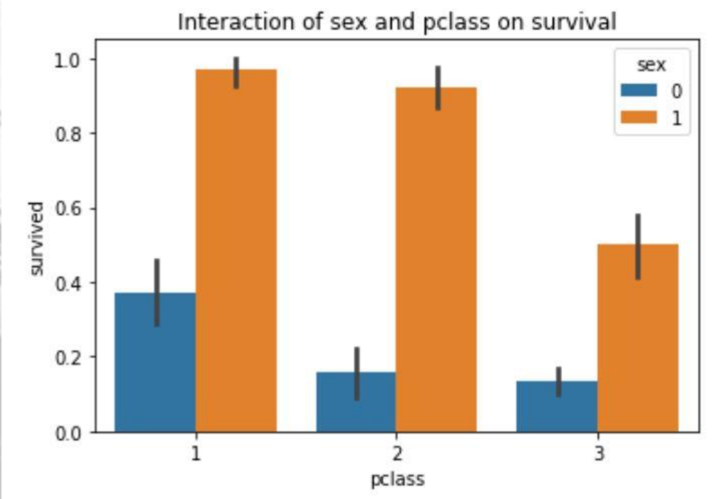


- Family size appears to have an impact
- Smaller families have a higher survival rate than larger ones
- Family_size will be used as a feature

EDA and Feature Engineering (pt. 3)



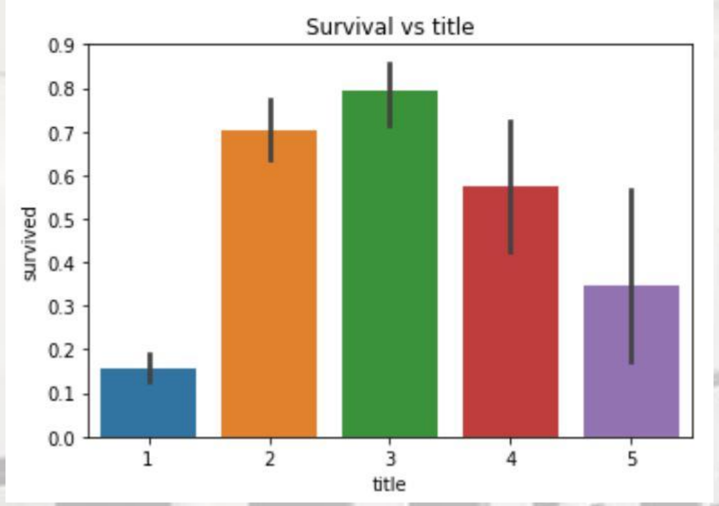
- 74.2% of females survived, compared to 18.9% of males
- ‘sex’ will be a selected feature
- Interaction between ‘sex’ and ‘pclass’ appears to be significant
    - Create new interaction variable ‘sex_pclass’

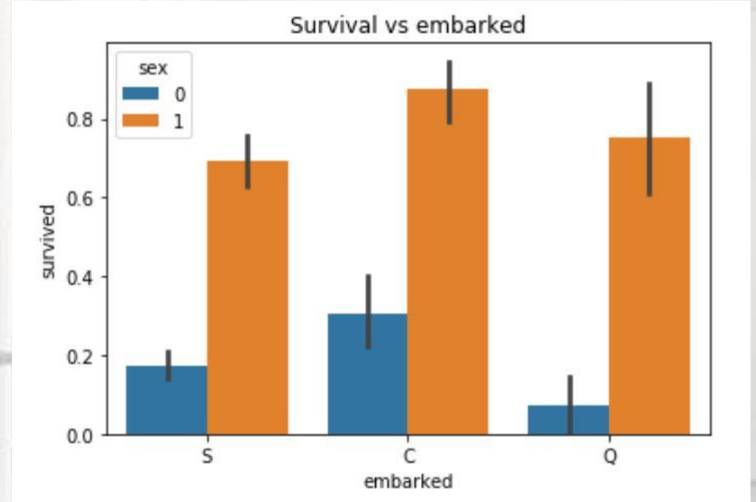


pclass	1	2	3
sex			
0	0.368852	0.157407	0.135447
1	0.968085	0.921053	0.500000

```
#Create interaction feature
for dataset in datasets:
    dataset['sex_pclass']= dataset['sex']* dataset['pclass']
```

EDA and Feature Engineering (pt. 4)






```
embarked
C    0.553571
Q    0.389610
S    0.339009
```

```
#Get count of unique names
titanic_df.name.nunique()
```

```
891
```

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

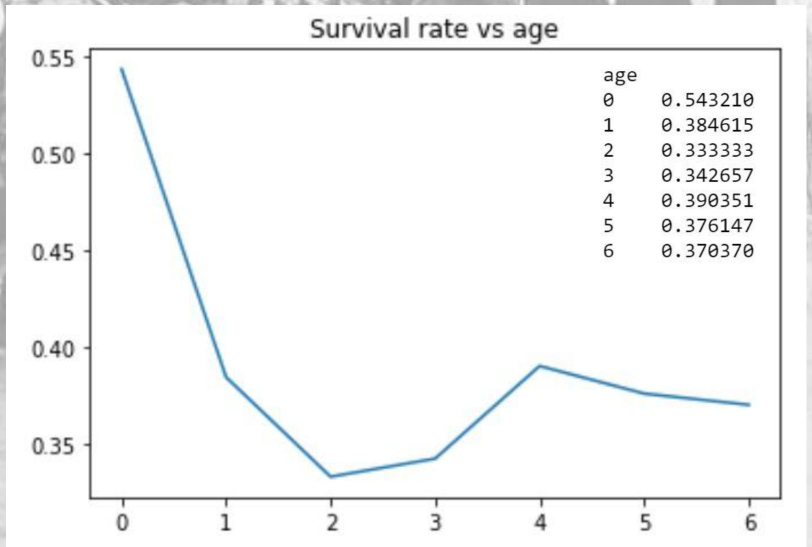
- Every passenger in the dataset has a unique name (not useful)
  - Titles for each name can be extracted (useful)
  - Mr, Miss, and Mrs titles are expectedly high (Master is also frequent)
  - The rest of the titles fall into a 'unique' category

```
titles = {"Mr": 1, "Miss": 2, "Mrs": 3, "Master": 4, "unique": 5}
```

- The embarked variable appears to have some useful information, and will be kept as a feature

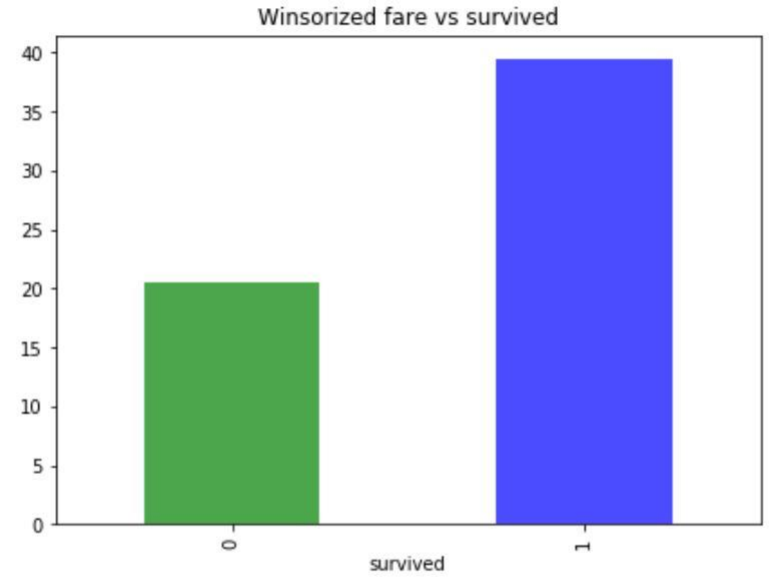
EDA and Feature Engineering (pt. 5)

```python
for dataset in datasets:
    dataset['age'] = dataset['age'].astype(int)
    #child
    dataset.loc[ dataset['age'] <= 12, 'age'] = 0
    #teenager
    dataset.loc[(dataset['age'] > 12) & (dataset['age'] <= 19), 'age'] = 1
    #young adult
    dataset.loc[(dataset['age'] > 19) & (dataset['age'] <= 24), 'age'] = 2
    #young adult 2
    dataset.loc[(dataset['age'] > 24) & (dataset['age'] <= 29), 'age'] = 3
    #30s
    dataset.loc[(dataset['age'] > 29) & (dataset['age'] <= 39), 'age'] = 4
    #40s
    dataset.loc[(dataset['age'] > 39) & (dataset['age'] <= 49), 'age'] = 5
    #50-60s
    dataset.loc[(dataset['age'] > 49) & (dataset['age'] <= 69), 'age'] = 6
    #elderly
    dataset.loc[ dataset['age'] > 69, 'age'] = 6
```



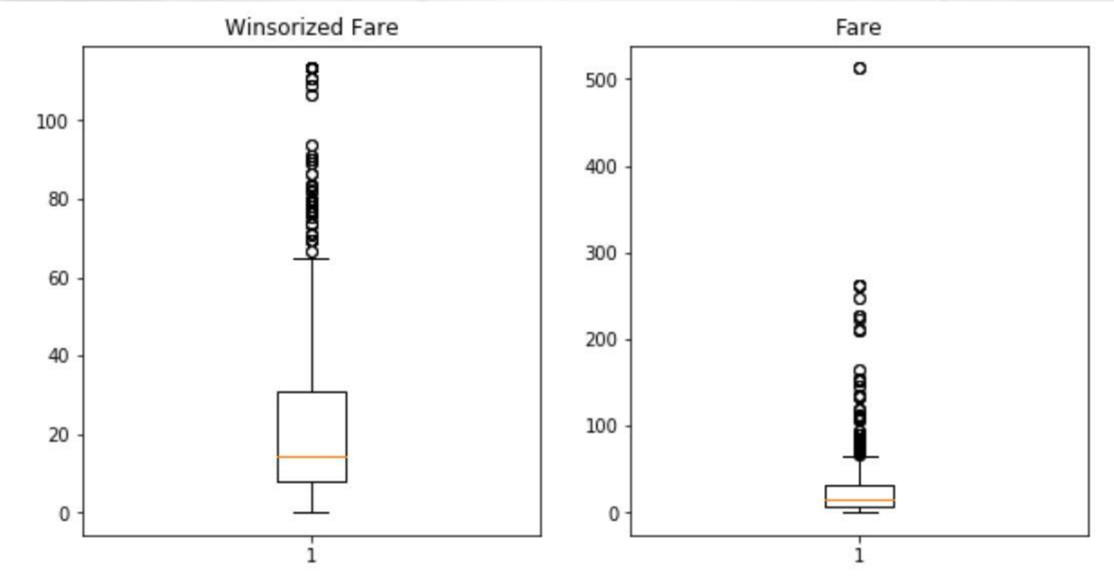
- There are many unique values for age, making it difficult to draw conclusions on a binary result (0-died, 1-survived)
  - Solution: assign age ranges, and then evaluate data
- There is a clear relationship between survival and age ranges
  - Children were the most likely to survive (age 12 and under)

EDA and Feature Engineering (pt. 6)




```
survived
0    20.489889
1    39.484845
```

- Winsorization was used on fare to account for outliers
- Some potenital outliers remain
  - Valuable info → keep
- Higher fare = higher chance of survival
- Select winsorized_fare as a feature




```
titanic_df.winsorized_fare.describe()
```

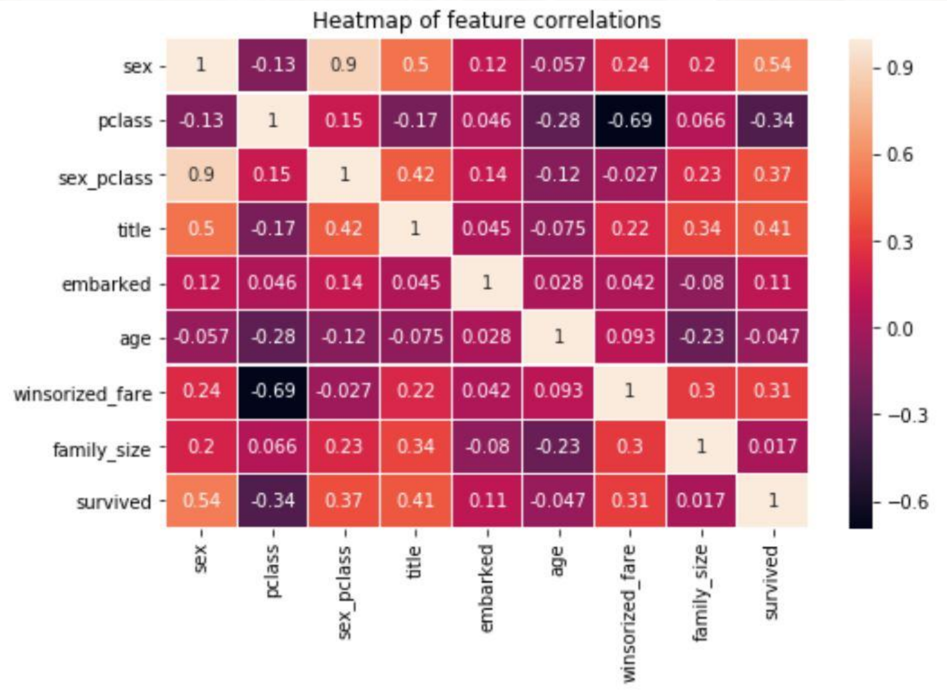
```
count    891.000000
mean      27.780882
std       29.400264
min        0.000000
25%        7.910400
50%       14.454200
75%       31.000000
max      113.275000
Name: winsorized_fare, dtype: float64
```

```
titanic_df.fare.describe()
```

```
count    891.000000
mean      32.204208
std       49.693429
min        0.000000
25%        7.910400
50%       14.454200
75%       31.000000
max      512.329200
Name: fare, dtype: float64
```

Features for building a model

- The interaction of ‘sex’ and ‘pclass’ has a very high correlation with ‘sex’
  - Therefore, the interaction is dropped
- List of selected features:
  - sex
  - pclass
  - title
  - embarked
  - age
  - winsorized_fare
  - family_size

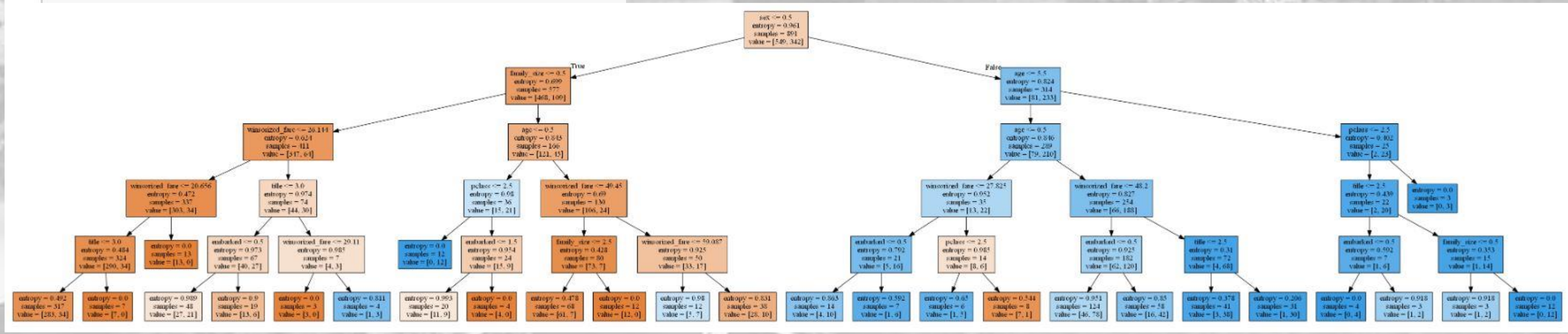



```
X_train = titanic_df[feature_list]
Y_train = titanic_df.survived
```

Building Supervised Learning Models (pt. 1)

Decision Tree

```
#Decision Tree
decision_tree = tree.DecisionTreeRegressor()

decision_tree = tree.DecisionTreeClassifier(
    criterion='entropy',
    max_features=1,
    max_depth=5)

decision_tree.fit(X_train, Y_train)
```



Random Forest

```
#Random Forest Model

from sklearn import ensemble
random_forest = ensemble.RandomForestClassifier(n_estimators=100,
                                                 criterion='entropy',
                                                max_features=1,
                                                max_depth=5)
random_forest.fit(X_train, Y_train)
```

Building Supervised Learning Models (pt. 2)

- Support vector machine
- Gradient booster
- KNN Classifier

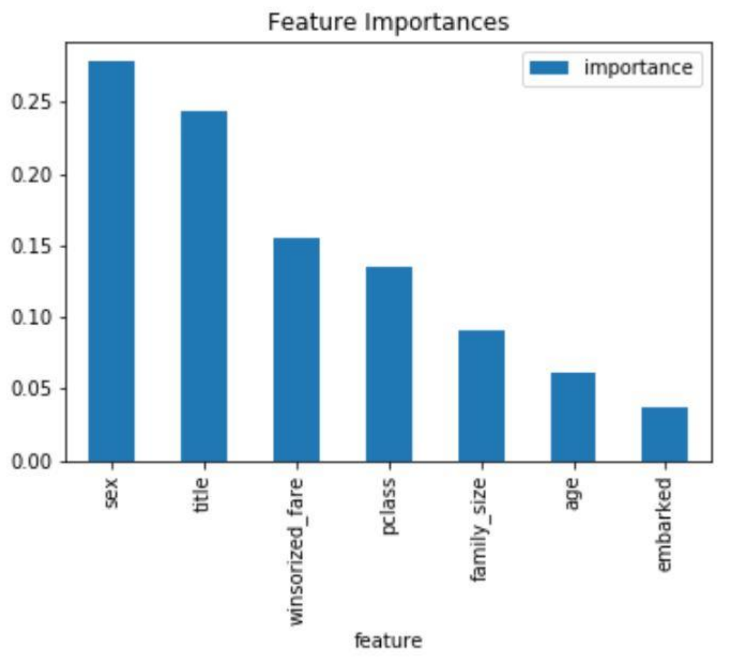
```python
#Gradient Boosting Classifier

params = {'n_estimators': 500,
          # 'max_depth': 2,
           'loss': 'deviance'}

clf = ensemble.GradientBoostingClassifier(**params)
clf.fit(X_train, Y_train)
```

```python
#SVC

from sklearn.svm import SVC, LinearSVC

linear_svc = LinearSVC()
linear_svc.fit(X_train, Y_train)

linear_svc.fit(X_train, Y_train)
```

```python
#KNN

from sklearn import neighbors

knn = neighbors.KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, Y_train)
```

Comparing Models and Feature Importance

Model	Score (training set)	Cross validation score (training set)	Score (test set)
Random Forest	0.851852	0.830470	0.935407
Support Vector Machines	0.793490	0.767672	0.925837
KNN Classifier	0.842873	0.766750	0.787081
Gradient Boosting Classifier	0.842873	0.766750	0.787081
Decision Tree	0.814815	0.761143	0.866029



- According to the chart (left) the Random Forest model appears to be the best choice
  - highest cross validation scores (no overfitting)
  - best performance for the test set
- Sex, title, and winsorized_fare are the top three features for importance (Random Forest model)

Tuning Hyperparameters

- Now that the Random Forest Classifier has been selected, we can try out different parameters to improve the model
  - Create a function adjust_parameters
- Chosen parameters:
  - N_estimators = 300
  - Criterion = ‘gini’
  - Max_depth = 6
  - Remaining set to default
- Accuracy/time tradeoff of n_estimators

```
def adjust_parameters(X, Y, n_estimators, criterion, max_features, max_depth):
    parameters = {
        'n_estimators': n_estimators,
        'criterion': criterion,
        'max_features': max_features,
        'max_depth': max_depth
    }
    random_forest = ensemble.RandomForestClassifier(**parameters)
    random_forest.fit(X, Y)
    rfc_score = random_forest.score(X, Y)
    print('Score:', rfc_score)
    rfc_cvs = cross_val_score(random_forest, X, Y, cv=10)
    print('Cross validation score', rfc_cvs.mean())
```

```
from datetime import datetime
start_time = datetime.now()

#Original
adjust_parameters(X_train, Y_train, 100, 'entropy', 1, 5)

end_time = datetime.now()
print('Duration: {}'.format(end_time - start_time))
```

```
Score: 0.8451178451178452
Cross validation score 0.83165645216207
Duration: 0:00:00.790937
```

```
start_time = datetime.now()

#Tuned hyperparameters
adjust_parameters(X_train, Y_train, 300, 'gini', None, 6)

end_time = datetime.now()
print('Duration: {}'.format(end_time - start_time))
```

```
Score: 0.8866442199775533
Cross validation score 0.8439779820678697
Duration: 0:00:03.401806
```

Making Predictions

- Would I survive the Titanic?

```
#Predictions for fun
#sex,pclass,title,embarked,age,fare,family_size
#Assuming I'm traveling with my dad and brother, second class
Jacob_survival = [[0, 2, 1, 1, 26, 20, 2]]
Jacob_pred = random_forest.predict_proba(Jacob_survival)
print(Jacob_pred)
```

```
[[0.82387264 0.17612736]]
```

- How about Jack and Rose from *Titanic (1997)*?

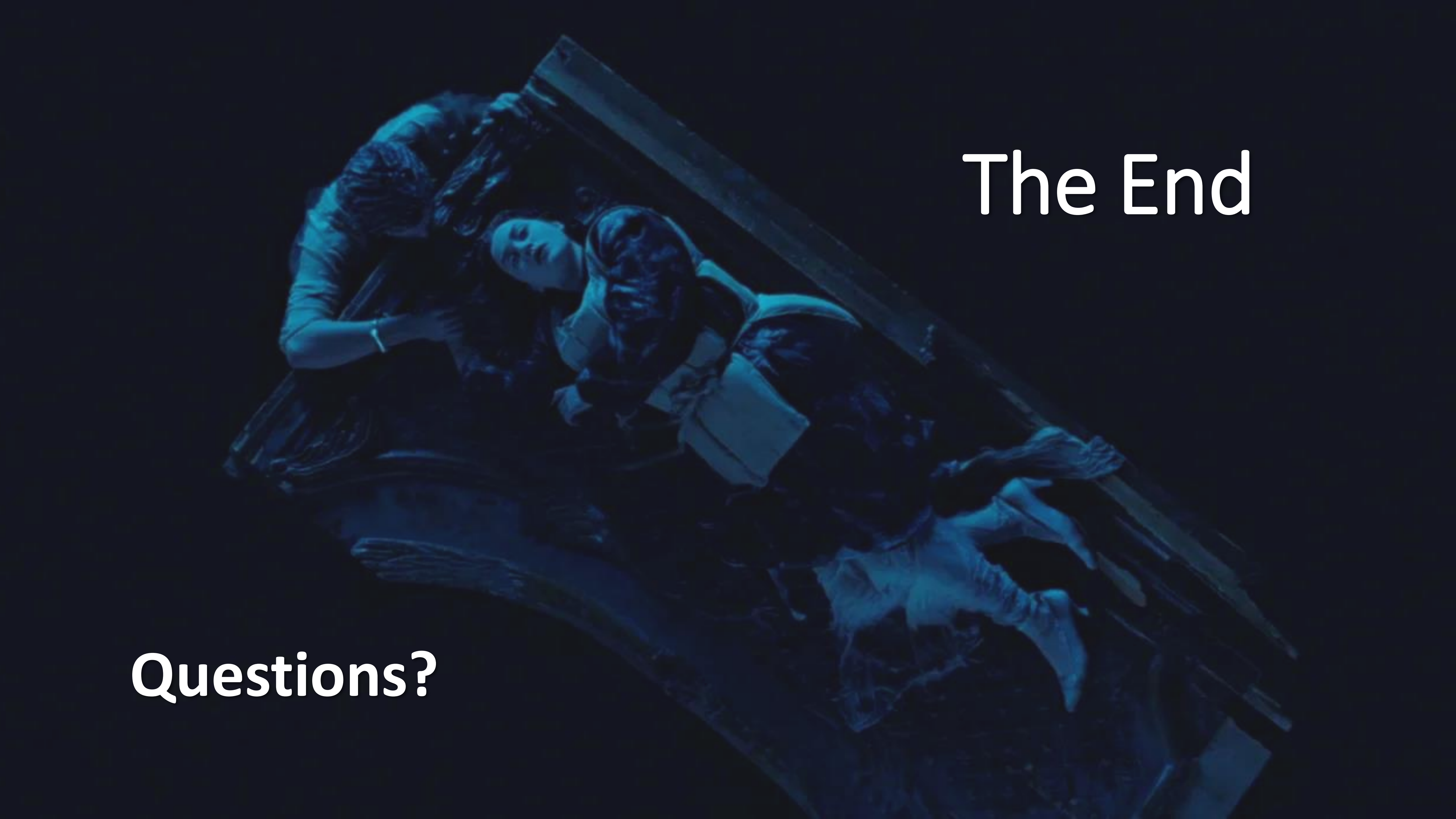
```
#Titanic characters

#Jack= male, third class, mr, Southhampton England, 20, cheap, alone
Jack_Dawson_survival = [[0, 3, 1, 0, 20, 10, 0]]
Jack_Dawson_pred = random_forest.predict_proba(Jack_Dawson_survival)
print('Jack Dawson:', Jack_Dawson_pred)

#Rose= female, first class, miss, Southhampton, 17, expensive, parents plus fiance(3)
Rose_DeWitt_Bukater_survival = [[1, 1, 2, 0, 17, 150, 3]]
Rose_DeWitt_Bukater_pred = random_forest.predict_proba(Rose_DeWitt_Bukater_survival)
print('Rose DeWitt Bukater:', Rose_DeWitt_Bukater_pred)
```

```
Jack Dawson: [[0.89705125 0.10294875]]
Rose DeWitt Bukater: [[0.14758536 0.85241464]]
```

Results:

- Jacob survival rate: 17.6%
- Jack survival rate: 10.3%
- Rose survival rate: 85%

Any room for Jack and I on that door, Rose?


The End
Questions?

Links

https://www.buzzfeednews.com/article/gabrielsanchez/haunting-pictures-from-the-titanic-before-she-sank
https://en.wikipedia.org/wiki/RMS_Titanic
https://www.kaggle.com/c/titanic/overview
https://www.historic-uk.com/HistoryUK/HistoryofBritain/RMS-Titanic-the-unsinkable-ship/