Miesjel van Rooij E-B31 2234475

Who Survived the Titanic Disaster?

1. Introduction

In this data prediction I will use decision tree and random forest to determine if a passenger of the Titanic would have survived. Based on the features age, passenger class and sex.

The reason why I used these features is because very specific features (name is an extreme case) could result in overfitting (consider a tree that just asks if the name is X, she survived). Features for which there are a small number of instances with each value present a similar problem. They might not be useful for generalization. I will use class, age, and sex because a priori, I expect them to have possibly influenced the passenger's survival.

2. Preparing Dataset with Pandas

2.1. Load Dataset

In [164]:
```
# Importing libraries to use
import pandas as pd
import numpy as np

# Checking versions
print('numpy version:', np.__version__)
print('matplotlib version:', pd.__version__)

## Reading csv file
df_titanic = pd.read_csv("Titanic dataset.csv")
```

```
numpy version: 1.16.5
matplotlib version: 0.25.1
```

In [165]:
```
# Checking top 10 rows of the dataframe
df_titanic.head(10)
```

Out[165]:

	row.names	pclass	survived	name	age	embarked	home.dest	room	ticket	bo
0	1	1st	1	Allen, Miss Elisabeth Walton	29.0000	Southampton	St Louis, MO	B-5	24160 L221	
1	2	1st	0	Allison, Miss Helen Loraine	2.0000	Southampton	Montreal, PQ / Chesterville, ON	C26	NaN	Na
2	3	1st	0	Allison, Mr Hudson Joshua Creighton	30.0000	Southampton	Montreal, PQ / Chesterville, ON	C26	NaN	(13
3	4	1st	0	Allison, Mrs Hudson J.C. (Bessie Waldo Daniels)	25.0000	Southampton	Montreal, PQ / Chesterville, ON	C26	NaN	Na
4	5	1st	1	Allison, Master Hudson Trevor	0.9167	Southampton	Montreal, PQ / Chesterville, ON	C22	NaN	
5	6	1st	1	Anderson, Mr Harry	47.0000	Southampton	New York, NY	E-12	NaN	
6	7	1st	1	Andrews, Miss Kornelia Theodosia	63.0000	Southampton	Hudson, NY	D-7	13502 L77	
7	8	1st	0	Andrews, Mr Thomas, jr	39.0000	Southampton	Belfast, NI	A-36	NaN	Na
8	9	1st	1	Appleton, Mrs Edward Dale (Charlotte Lamson)	58.0000	Southampton	Bayside, Queens, NY	C-101	NaN	
9	10	1st	0	Artagaveytia, Mr Ramon	71.0000	Cherbourg	Montevideo, Uruguay	NaN	NaN	(2

2.2. Investigating Dataset

In [166]:
```
# Checking all data types of all rows
df_titanic.dtypes
```

Out[166]:
```
row.names      int64
pclass        object
survived       int64
name          object
age          float64
embarked      object
home.dest     object
room          object
ticket        object
boat          object
sex           object
dtype: object
```

In [167]:
```
# Checking info of each column. Showing amount of entries, data types and total c
df_titanic.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1313 entries, 0 to 1312
Data columns (total 11 columns):
row.names    1313 non-null int64
pclass       1313 non-null object
survived     1313 non-null int64
name         1313 non-null object
age          633 non-null float64
embarked     821 non-null object
home.dest    754 non-null object
room         77 non-null object
ticket       69 non-null object
boat         347 non-null object
sex          1313 non-null object
dtypes: float64(1), int64(2), object(8)
memory usage: 113.0+ KB
```

As seen in the result above, the age column (1 of the 3 features I will use), has alot of NaN values.

In [168]:
```
# Statistical aggregated functions on the int and float type columns.
df_titanic.describe()
```

Out[168]:

	row.names	survived	age
count	1313.000000	1313.000000	633.000000
mean	657.000000	0.341965	31.194181
std	379.174762	0.474549	14.747525
min	1.000000	0.000000	0.166700
25%	329.000000	0.000000	21.000000
50%	657.000000	0.000000	30.000000
75%	985.000000	1.000000	41.000000
max	1313.000000	1.000000	71.000000

In [169]:
```
# Average age of all passengers
df_titanic.age.mean()
```

Out[169]:
```
31.19418104265403
```

In [170]:
```
# The mode of age
df_titanic.age.mode()
```

Out[170]:
```
0    30.0
dtype: float64
```

As seen above the average age is 31.19, whereas the mode is 30. I chose to use the average age to fill up the missing age entries later on.

In [171]:
```
# Creating new variables for the columns sex, age and pclass which I will use as
sex_column = df_titanic.sex
age_column = df_titanic.age
pclass_column = df_titanic.pclass
```

In [172]:
```
# Checking for total amount of null values in age column
totalMissingAge = age_column.isnull().sum()
print("Total amount of null values in the age column: " + str(totalMissingAge))
```

```
Total amount of null values in the age column: 680
```

In [173]:
```
# Checking for total amount of null values in sex column
totalMissingSex = sex_column.isnull().sum()
print("Total amount of null values in the age column: " + str(totalMissingSex))
```

```
Total amount of null values in the age column: 0
```

In [174]:
```
# Checking for total amount of null values in pclass column
totalMissingPclass = pclass_column.isnull().sum()
print("Total amount of null values in the age column: " + str(totalMissingPclass
```

```
Total amount of null values in the age column: 0
```

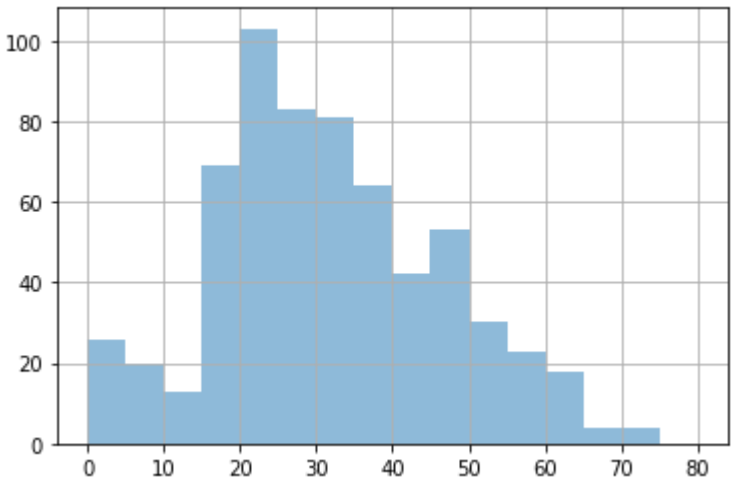
In [175]:
```
# Printing all rows with missing age entrie.
df_titanic[df_titanic['age'].isnull()][['sex', 'pclass', 'age']]
```

Out[175]:

	sex	pclass	age
12	female	1st	NaN
13	male	1st	NaN
14	male	1st	NaN
29	male	1st	NaN
32	male	1st	NaN
...	...	...	...
1308	male	3rd	NaN
1309	male	3rd	NaN
1310	male	3rd	NaN
1311	female	3rd	NaN
1312	male	3rd	NaN

680 rows × 3 columns

In [176]:
```
# Creating a histogram to show the distibution of the passenger's age. Age 0 to 8
df_titanic['age'].dropna().hist(bins=16, range=(0,80), alpha = .5)
pyl.show()
```



3. Data Munging

3.1. Transforming the Data

In this part I will transform the values in the dataframe into the shape needed for machine learning. In this case on a tree based algorithm. Therefor it is needed to transfor all data in binary features.

First of all, it's hard to run analysis on the string values of "male" and "female". I will store this transformation into a new column "Gender".

In [177]:
```
# Creating a new column for the gender, so the original sex is not changed.
# 0 is for female and 1 is for male
df_titanic['Gender'] = df_titanic['sex'].map( {'female': 0, 'male': 1} ).astype(
df_titanic.head(3)
```

Out[177]:

	row.names	pclass	survived	name	age	embarked	home.dest	room	ticket	boat	
0	1	1st	1	Allen, Miss Elisabeth Walton	29.0	Southampton	St Louis, MO	B-5	24160 L221	2	fer
1	2	1st	0	Allison, Miss Helen Loraine	2.0	Southampton	Montreal, PQ / Chesterville, ON	C26	NaN	NaN	fer
2	3	1st	0	Allison, Mr Hudson Joshua Creighton	30.0	Southampton	Montreal, PQ / Chesterville, ON	C26	NaN	(135)	r

In [178]:
```
# I will do the same for passenger class and make it numeric. This is stored in t
# 1 is for 1st class
# 2 is for 2nd class
# 3 is for 3rd class
df_titanic['Class_numeric'] = df_titanic['pclass'].map( {'1st': 1, '2nd': 2, '3r
df_titanic.head(3)
```

Out[178]:

	row.names	pclass	survived	name	age	embarked	home.dest	room	ticket	boat	
0	1	1st	1	Allen, Miss Elisabeth Walton	29.0	Southampton	St Louis, MO	B-5	24160 L221	2	fer
1	2	1st	0	Allison, Miss Helen Loraine	2.0	Southampton	Montreal, PQ / Chesterville, ON	C26	NaN	NaN	fer
2	3	1st	0	Allison, Mr Hudson Joshua Creighton	30.0	Southampton	Montreal, PQ / Chesterville, ON	C26	NaN	(135)	r

In [179]:
```
# To test and check whether it work I will create a list of all males in 2nd clas
df_titanicMales2nd = df_titanic[(df_titanic.Class_numeric == 2) & (df_titanic.Ge
df_titanicMales2nd
```

Out[179]:

	row.names	pclass	survived	name	age	embarked	home.dest	room	ticket	boat
322	323	2nd	0	Abelson, Mr Samuel	30.0	Cherbourg	Russia New York, NY	NaN	NaN	NaN
324	325	2nd	0	Andrew, Mr Edgar Samuel	18.0	Southampton	Buenos Aires, Argentina / New Jersey, NJ	NaN	NaN	NaN
325	326	2nd	0	Andrew, Mr Frank	NaN	Southampton	Cornwall, England Houghton, MI	NaN	NaN	NaN
326	327	2nd	0	Angle, Mr William A.	34.0	Southampton	Warwick, England	NaN	NaN	NaN
328	329	2nd	0	Ashby, Mr John	57.0	Southampton	West Hoboken, NJ	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...
593	594	2nd	1	Wilhelms, Mr Charles	32.0	Southampton	London, England	NaN	NaN	9
595	596	2nd	1	Williams, Mr Charles Eugene	NaN	Southampton	Harrow, England	NaN	NaN	14
598	599	2nd	0	Aldworth, Mr Charles Augustus	30.0	Southampton	Bryn Mawr, PA, USA	NaN	248744 L13	NaN
600	601	2nd	0	Pernot, Mr Rene	NaN	Cherbourg	NaN	2131	L15 1s	NaN
601	602	2nd	0	Swane, Mr George	18.0	Southampton	NaN	F-?	NaN	(294)

173 rows × 13 columns

3.2. Handling missing values

The age histogram did seem positively skewed. Therefor I will use the average age to fill in the missing age entries

In [180]:
```
# Replacing the NaN age values with the average age in a new column "AgeFill".
df_titanic['age'].fillna((df_titanic['age'].mean()), inplace=True)
df_titanic['AgeFill'] = df_titanic.age.mean()
df_titanic.head(50)
```

				(Margaret Molly" Tobin)"				
37	38	1st	1	Brown, Mrs John Murray (Caroline Lane Lamson)	59.000000	Southampton	Belmont, MA	C-101
38	39	1st	1	Bucknell, Mrs William Robert (Emma Eliza Ward)	60.000000	Cherbourg	Philadelphia, PA	NaN
39	40	1st	0	Butt, Major Archibald Willingham	45.000000	Southampton	Washington, DC	NaN
40	41	1st	1	Calderhead, Mr Edward P.	31.194181	Southampton	New York, NY	NaN
				Candee, Mrs				

3.3 Hot Encoding

The catergorical feature attribute pclass is already converted into 0, 1 and 2. Below here I will further transform this into three new binary features. This is done by hot encoding, which is a way of managing categorical attributes for real-based methods.

In [181]:
```
df_titanic['FirstClass'] = df_titanic['pclass'].map( {'1st': 1, '2nd': 0, '3rd':
df_titanic['SecondClass'] = df_titanic['pclass'].map( {'1st': 0, '2nd': 1, '3rd'
df_titanic['ThirdClass'] = df_titanic['pclass'].map( {'1st': 0, '2nd': 0, '3rd':

df_titanic.head(5)
```

Out[181]:

	row.names	pclass	survived	name	age	embarked	home.dest	room	ticket	boat
0	1	1st	1	Allen, Miss Elisabeth Walton	29.0000	Southampton	St Louis, MO	B-5	24160 L221	2
1	2	1st	0	Allison, Miss Helen Loraine	2.0000	Southampton	Montreal, PQ / Chesterville, ON	C26	NaN	NaN
2	3	1st	0	Allison, Mr Hudson Joshua Creighton	30.0000	Southampton	Montreal, PQ / Chesterville, ON	C26	NaN	(135)
3	4	1st	0	Allison, Mrs Hudson J.C. (Bessie Waldo Daniels)	25.0000	Southampton	Montreal, PQ / Chesterville, ON	C26	NaN	NaN
4	5	1st	1	Allison, Master Hudson Trevor	0.9167	Southampton	Montreal, PQ / Chesterville, ON	C22	NaN	11

4. Finalising dataset for analysis

In [182]:
```
# Finalising pre-processing by turning this into a numerical feature set (datafr
# (dataframe titanic_y) and a numerical target column
titanic_X = df_titanic[['age','Gender','FirstClass','SecondClass','ThirdClass']]
titanic_y = df_titanic['survived']
```

In [183]:
```
titanic_X.head(10)
```

Out[183]:

	age	Gender	FirstClass	SecondClass	ThirdClass
0	29.0000	0	1	0	0
1	2.0000	0	1	0	0
2	30.0000	1	1	0	0
3	25.0000	0	1	0	0
4	0.9167	1	1	0	0
5	47.0000	1	1	0	0
6	63.0000	0	1	0	0
7	39.0000	1	1	0	0
8	58.0000	0	1	0	0
9	71.0000	1	1	0	0

In [184]:
```
titanic_y.head(10)
```

Out[184]:
```
0    1
1    0
2    0
3    0
4    1
5    1
6    1
7    0
8    1
9    0
Name: survived, dtype: int64
```

5. Analyse Dataset

5.1. Training a Decision Tree Classifier

In [185]:
```
# Importing from sklearn to split training and testing data
from sklearn.model_selection import train_test_split

# Splitting testing and training data from the dataset
X_train, X_test, y_train, y_test = train_test_split(titanic_X, titanic_y, test_s
```

In [186]:
```
# Importing decision tree classifer
from sklearn import tree
clf_entropy = tree.DecisionTreeClassifier(criterion = "entropy", random_state = 1

# Fitting the training data into the model
clf_entropy.fit(X_train, y_train)
```

Out[186]:
```
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=5,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=5, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=100, splitter='best')
```

In [187]:
```
# Setting the predict values from the test set to the y_pred_en variable
y_pred_en = clf_entropy.predict(X_test)

# Displaying the preidcitons doen with the testing set
y_pred_en
```

Out[187]:
```
array([0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1,
       0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0,
       1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1,
       1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0,
       0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0,
       1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0],
      dtype=int64)
```

5.2. Evaluation Metrics Function

In [188]:
```
# Importing evaluation metrics from sklearn to show the results
from sklearn.metrics import accuracy_score

# Showing the accuracy of the tree based model
print ("Accuracy is ", accuracy_score(y_test,y_pred_en)*100)
```

```
Accuracy is  85.17110266159695
```

6. Implementing Random Forest

In [189]:
```
# Implementing a Random Forest classifier.

# Importing the library in which RandomForest Classifier is located.
import sklearn.ensemble as ensemble

# Useing N-estimators to decide the amount of trees in the forest
model = ensemble.RandomForestClassifier(n_estimators = 100)

# fitting the model with the training data
model.fit(titanic_X, titanic_y)

# Printing the model
print(model)
```

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                       max_depth=None, max_features='auto', max_leaf_nodes=Non
e,
                       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=None,
                       verbose=0, warm_start=False)
```

In [190]:
```
# Setting the predict values from the test set to the y_pred_en variable
y_pred_en2 = model.predict(X_test)

# Displaying the predictions done with the testing set
y_pred_en2
```

Out[190]:
```
array([0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1,
       0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0,
       1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1,
       1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0,
       0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0,
       1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0],
      dtype=int64)
```

In [191]:
```
print ("Accuracy is ", accuracy_score(y_test,y_pred_en2)*100)
```

```
Accuracy is  86.31178707224335
```

To conclude this analyse the decision tree model has an accuracy of 85.17, whereas the random forest model has an accuracy of 86.31 to predict whether someone would survive the titanic disaster. This is based on the features age, class and gender.