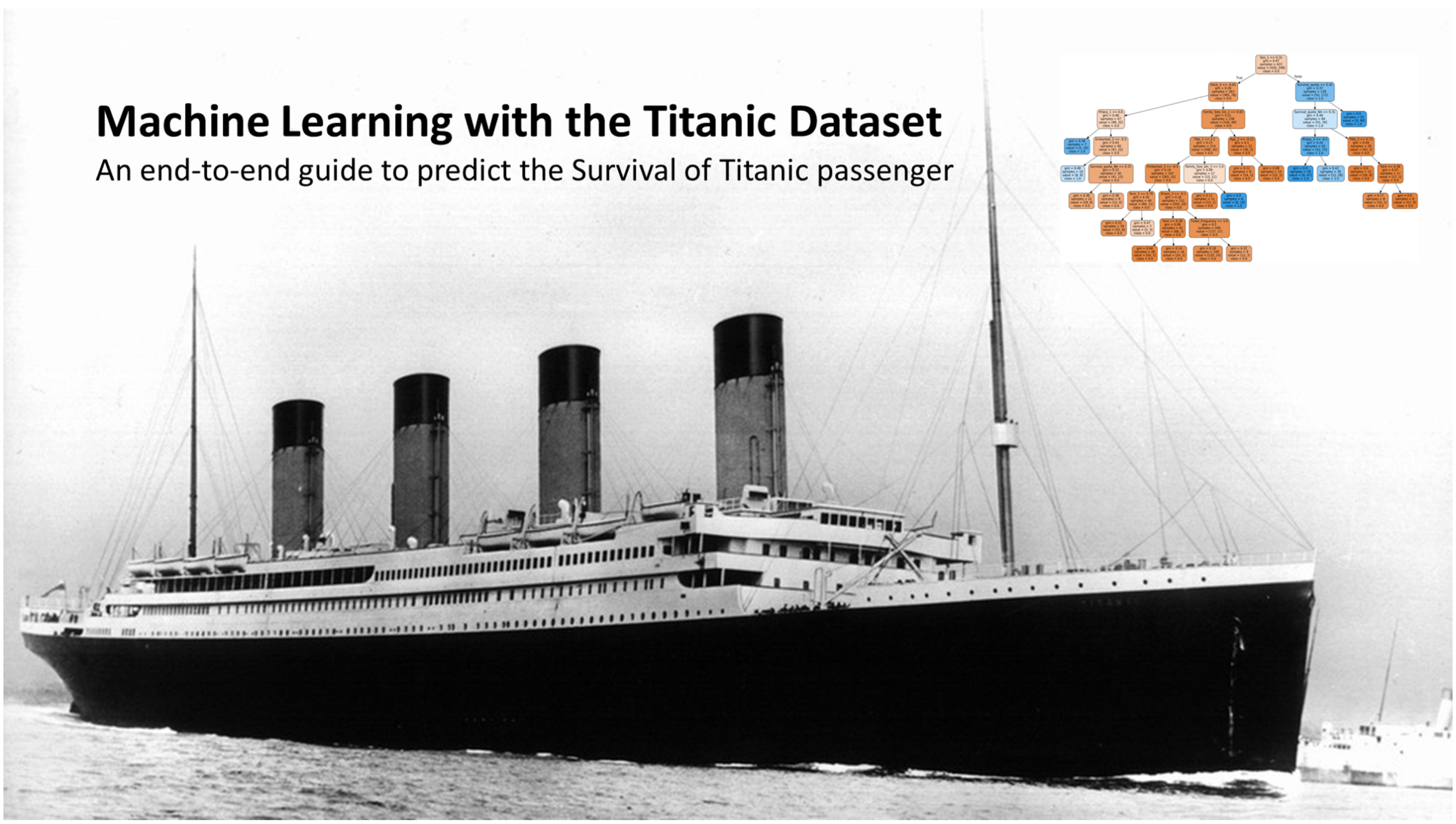
Machine Learning with the Titanic Dataset

An end-to-end guide to predict the Survival of Titanic passenger

Benedikt Droste Apr 10, 2020 · 11 min read ★



From my point of view tutorials for beginners should bring the reader in the position to go on *score at the end but I will also show up some categories where you can easily improve the score. After you have finished reading you can take the model and improve it by yourself. If you are interested in machine learning, the dramatic sinking of the Titanic is a good starting point for your own data science journey. Good luck!*

Getting started

If you are completely new to Kaggle, check out this tutorial for the set up process. You will find the data set and so on here.

After you can loading the files in the Kaggle kernel:

```
def concat_df(train_data, test_data):
    return pd.concat([train_data, test_data], sort=True).reset_index(drop=True)

def divide_df(all_data):
    return all_data.loc[:890], all_data.loc[891:].drop(['Survived'], axis=1)
```

+ Code + Markdown

```
train_data = pd.read_csv("/kaggle/input/titanic/train.csv")
test_data = pd.read_csv("/kaggle/input/titanic/test.csv")

df_all = concat_df(train_data, test_data)
dfs = [train_data, test_data]
```

Which variables are in our dataset:

pclass	Ticket class	1 = 1st, 2 = 2nd, 3 = 3rd
sex	Sex	
Age	Age in years	
sibsp	# of siblings / spouses aboard the Titanic	
parch	# of parents / children aboard the Titanic	
ticket	Ticket number	
fare	Passenger fare	
cabin	Cabin number	
embarked	Port of Embarkation	C = Cherbourg, Q = Queenstown, S = Southampton

Kaggle notes:

pclass: A proxy for socio-economic status (SES)
1st = Upper
2nd = Middle
3rd = Lower

sibsp: The dataset defines family relations in this way…
Sibling = brother, sister, stepbrother, stepsister
Spouse = husband, wife (mistresses and fiancés were ignored)

parch: The dataset defines family relations in this way…
Parent = mother, father
Child = daughter, son, stepdaughter, stepson
Some children travelled only with a nanny, therefore parch=0 for them.

1. Checks in term of data quality

In a first step we will investigate the titanic data set. Kaggle provides a train and a test data set. The train data set contains all the features (possible predictors) and the target (the variable which outcome we want to predict). The test data set is used for the submission, therefore the target variable is missing. Let´s have a look at the data sets:

```
Train data contains: 891 rows and 12 columns
Test data contains: 418 rows and 11 columns
```

+ Code + Markdown

```
print("First 3 rows of the train data:")
display(train_data.head(3))
print("First 3 rows of the test data:")
display(test_data.head(3))
```

```
First 3 rows of the train data:
```

	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

How I already wrote in the introduction, the target variable “Survived” is missing in the test data set. All other columns appears in both dataframs. In sum we have 11 different variables which can be used as features to predict the outcome of our target. You can see at first sight that there are missings for “Cabin”. Missings can irritate our algorithms, so it is important task to clean up the data in a first step.

```
print("Missings in the test data:")
display(test_data.isnull().sum())
```

```
Missings in the train data:
```

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

```
Missings in the test data:
```

```
PassengerId      0
Pclass           0
Name             0
Sex              0
Age             86
SibSp            0
Parch            0
Ticket           0
Fare             1
Cabin          327
Embarked         0
dtype: int64
```

In the training data we have missings in the age, cabin and embarked column. In the test data set are missings in the age, fare and cabin column. We will concat both data sets and perform the data cleansing for the entire data set.

```
df_all = concat_df(train_data, test_data)
```

2. Data cleansing

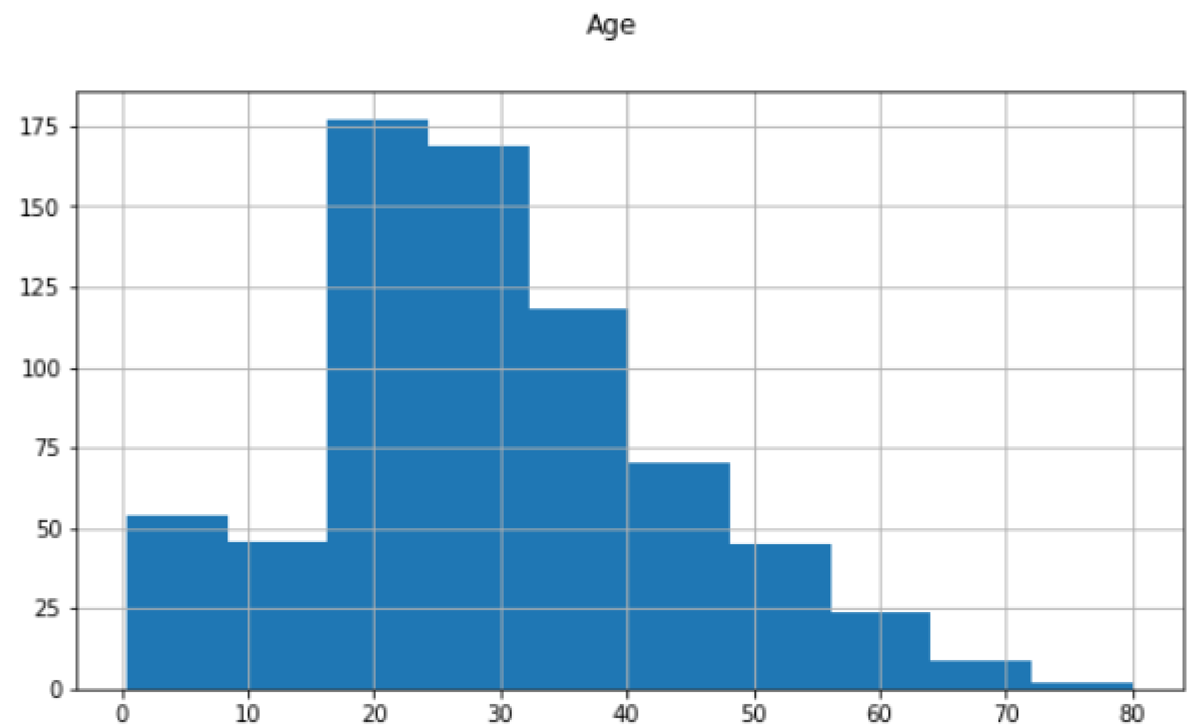
2.1 Age

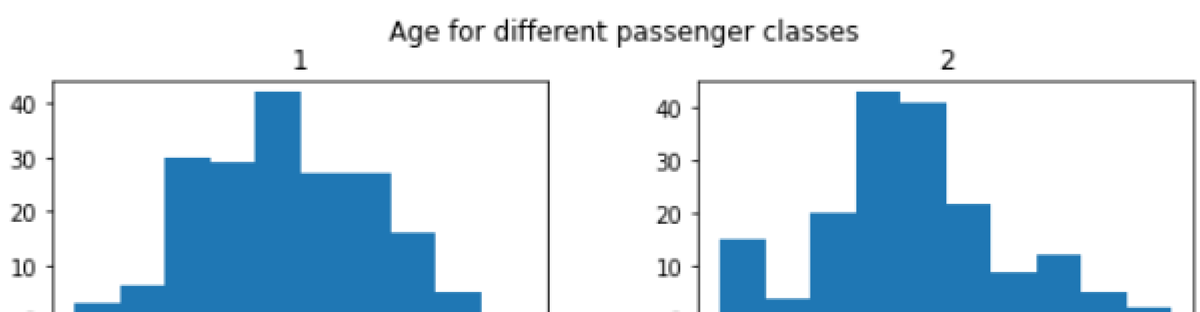
```
print("Missings for Age in the entire data set: " + str(df_all['Age'].isnull().sum()))
print("Missings in percentage: " + str(round(df_all['Age'].isnull().sum()/len(df_all)*100,0)) + " %")
```

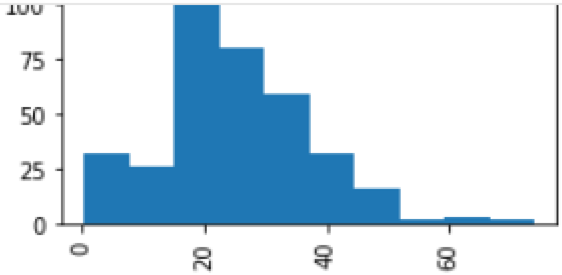
```
Missings for Age in the entire data set: 263
Missings in percentage: 20.0 %
```

+ Code + Markdown

20% of our age column is missings. Let´s have a look at the distribution:








```
print('Median for Age seperated by Pclass:')
display(train_data.groupby('Pclass')['Age'].median())
print('Median for Age seperated by Pclass and Sex:')
display(train_data.groupby(['Pclass','Sex'])['Age'].median())
print('Number of cases:')
display(train_data.groupby(['Pclass','Sex'])['Age'].count())
```

```
Median for Age seperated by Pclass:
```

```
Pclass
1    37.0
2    29.0
3    24.0
Name: Age, dtype: float64
```

```
Median for Age seperated by Pclass and Sex:
```

```
Pclass  Sex
1       female    35.0
        male      40.0
2       female    28.0
        male      30.0
3       female    21.5
        male      25.0
Name: Age, dtype: float64
```

```
Number of cases:
```

```
Pclass  Sex
1       female     85
        male      101
2       female     74
        male       99
3       female    102
        male      253
Name: Age, dtype: int64
```

We don´t want to delete all rows with missing age values, therefore we will replace the missings. As you can see we have a right-skrewed distribution for age and the median should a good choice for substitution.

One thesis was that the median of age differs for the passenger classes. Professional advancement usually comes with increasing age and experience. Therefore, people with a higher socio-economic status are older on average. If we split up by sex we see that there is still a difference because women are younger in general. In a last step I have checked the number of cases to ensure that there are still enough cases in each category. We will use these median values to replace the missings.

```
#replace the missings values with the medians of each group
df_all['Age'] = df_all.groupby(['Pclass','Sex'])['Age'].apply(lambda x: x.fillna(x.median()))
```

2.2 Fare

```
df_all.loc[df_all['Fare'].isnull()]
```

:

	Age	Cabin	Embarked	Fare	Name	Parch	PassengerId	Pclass	Sex	SibSp	Survived	Ticket
1043	60.5	NaN	S	NaN	Storey, Mr. Thomas	0	1044	3	male	0	NaN	3701

We have just one missing fare value in the whole data set. Mr. Thomas was in passenger class 3, travelled alone and embarked in Southhampton. We will take other cases from people in this category and replace the missing Fare with the median of this group.

```
#loc cases which are similiar to Mr. Thomas and use the median of fare to replace the missing for his data set
mr_thomas = df_all.loc[(df_all['Pclass'] == 3) & (df_all['SibSp'] == 0) & (df_all['Embarked'] == 'S')]['Fare'].median()
print(mr_thomas)
df_all.loc[df_all['Fare'].isnull(), 'Fare'] = mr_thomas
```

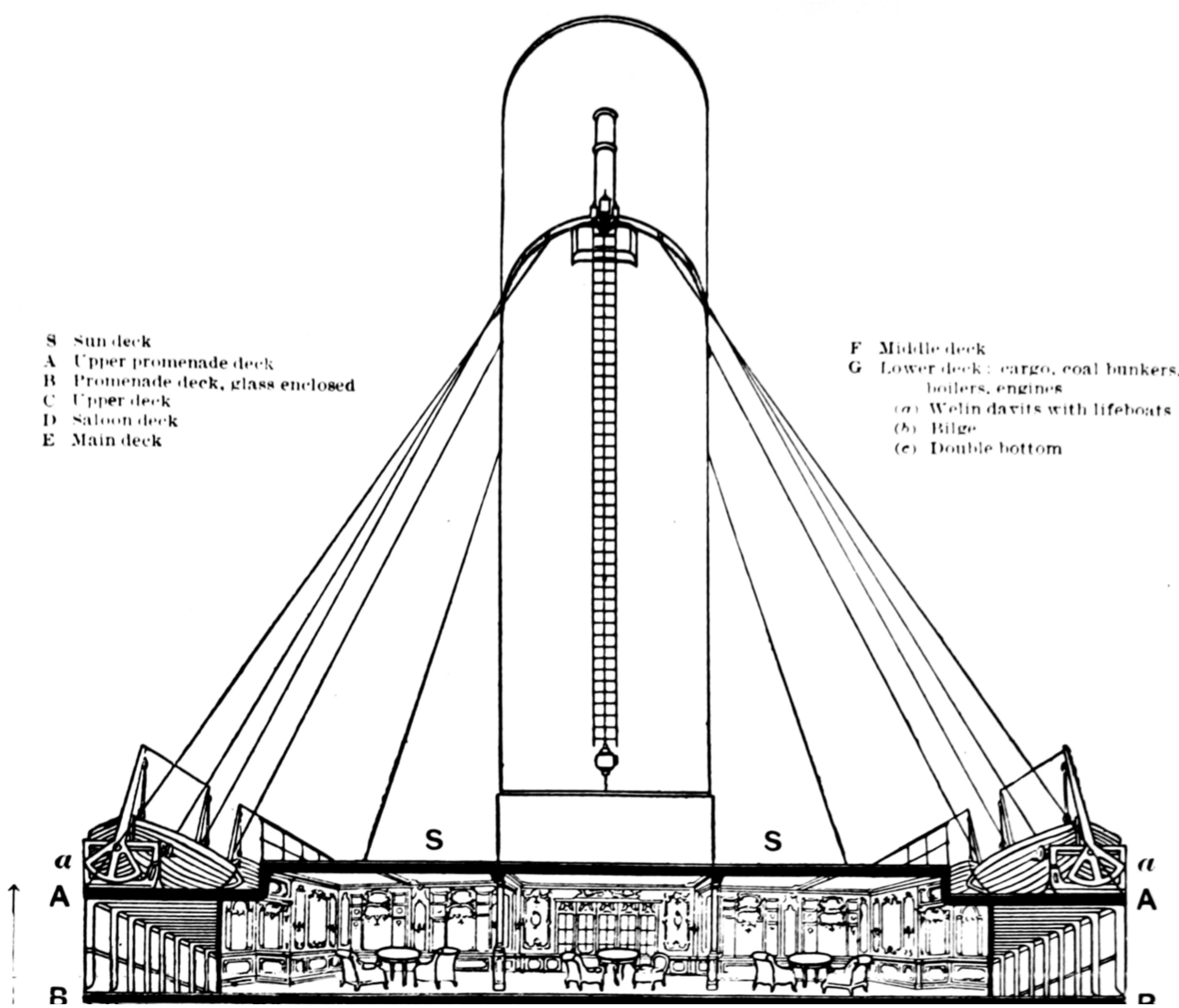
```
7.925
```

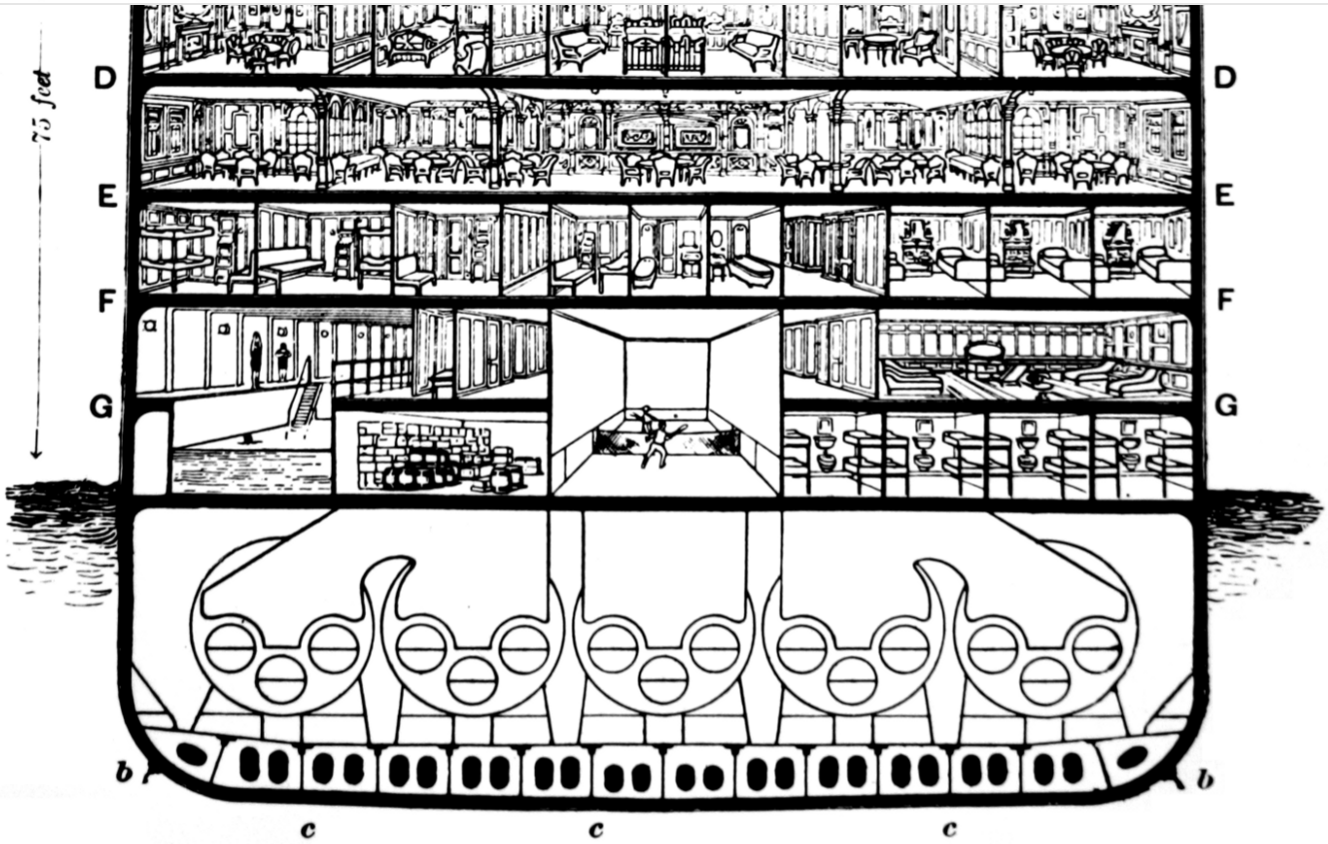
2.3 Cabin

```
       'C23 C25 C27', 'B78', 'D33', 'B30', 'C52', 'B28', 'C83', 'F33',
       'F G73', 'E31', 'A5', 'D10 D12', 'D26', 'C110', 'B58 B60', 'E101',
       'F E69', 'D47', 'B86', 'F2', 'C2', 'E33', 'B19', 'A7', 'C49', 'F4',
       'A32', 'B4', 'B80', 'A31', 'D36', 'D15', 'C93', 'C78', 'D35',
       'C87', 'B77', 'E67', 'B94', 'C125', 'C99', 'C118', 'D7', 'A19',
       'B49', 'D', 'C22 C26', 'C106', 'C65', 'E36', 'C54',
       'B57 B59 B63 B66', 'C7', 'E34', 'C32', 'B18', 'C124', 'C91', 'E40',
       'T', 'C128', 'D37', 'B35', 'E50', 'C82', 'B96 B98', 'E10', 'E44',
       'A34', 'C104', 'C111', 'C92', 'E38', 'D21', 'E12', 'E63', 'A14',
       'B37', 'C30', 'D20', 'B79', 'E25', 'D46', 'B73', 'C95', 'B38',
       'B39', 'B22', 'C86', 'C70', 'A16', 'C101', 'C68', 'A10', 'E68',
       'B41', 'A20', 'D19', 'D50', 'D9', 'A23', 'B50', 'A26', 'D48',
       'E58', 'C126', 'B71', 'B51 B53 B55', 'D49', 'B5', 'B20', 'F G63',
       'C62 C64', 'E24', 'C90', 'C45', 'E8', 'B101', 'D45', 'C46', 'D30',
       'E121', 'D11', 'E77', 'F38', 'B3', 'D6', 'B82 B84', 'D17', 'A36',
       'B102', 'B69', 'E49', 'C47', 'D28', 'E17', 'A24', 'C50', 'B42',
       'C148'], dtype=object)
```

```
There are 147 different values for Cabin and 687 cases are missing.
```

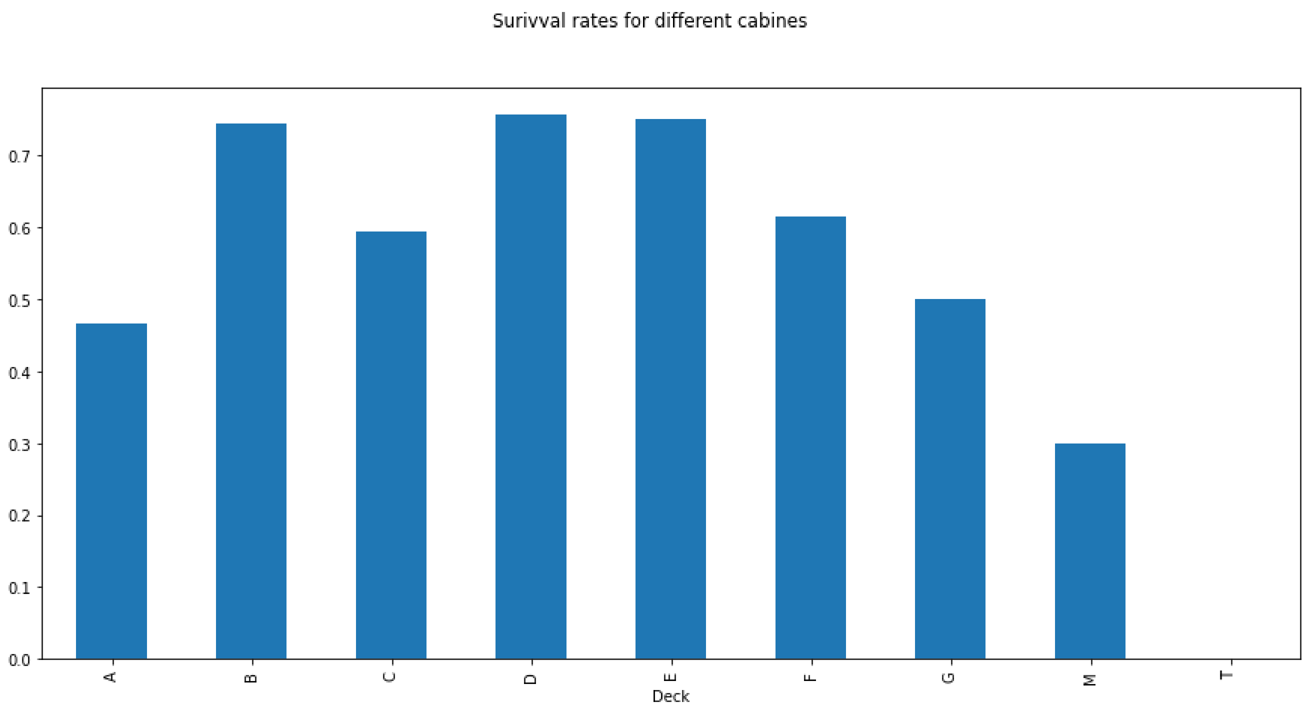
There are a lot of missing values but we should use the cabin variable because it can be an important predictor. As you can see in the following picture, the first class had the cabins on deck A, B or C, a mix of it was on D or E and the third class was mainly on f or g. We can identify the deck by the first letter.




75 feet
D
D
E
E
F
F
G
G
b
b
c
c
c

```
df_all[['Deck','Survived']].groupby('Deck')['Survived'].mean().plot(kind='bar', figsize=(15,7))
pl.suptitle('Surivval rates for different cabines')
```

```
Text(0.5, 0.98, 'Surivval rates for different cabines')
```



There are significant differences in survival rates because guests on the upper decks were quicker on the lifeboats. We will group up some decks.

```
idx = df_all[df_all['Deck'] == 'T'].index
df_all.loc[idx, 'Deck'] = 'A'
df_all['Deck'] = df_all['Deck'].replace(['A', 'B', 'C'], 'ABC')
df_all['Deck'] = df_all['Deck'].replace(['D', 'E'], 'DE')
df_all['Deck'] = df_all['Deck'].replace(['F', 'G'], 'FG')

df_all['Deck'].value_counts()
```

```
M      1014
ABC     182
DE       87
FG       26
Name: Deck, dtype: int64
```

```
df_all.loc[df_all['Embarked'].isnull()]
```

	Age	Cabin	Embarked	Fare	Name	Parch	PassengerId	Pclass	Sex	SibSp	Survived	Ticket	Deck
61	38.0	B28	NaN	80.0	Icard, Miss. Amelie	0	62	1	female	0	1.0	113572	ABC
829	62.0	B28	NaN	80.0	Stone, Mrs. George Nelson (Martha Evelyn)	0	830	1	female	0	1.0	113572	ABC

```
#check for passengers who were in passenger class 1, on deck abc and paid 80 or less for the tickets
df_all.loc[(df_all['Pclass'] == 1) & (df_all['Fare'] <= 80) & (df_all['Deck'] == 'ABC')]['Embarked'].value_counts()
```

```
S    50
C    42
Name: Embarked, dtype: int64
```

There are just two missings for embarked. As we already tried for the fare case we can look up similiar cases to replace the missing value. It stands to reason that people who paid a similar amount, also had a class 1 ticket and were on the same deck, embarked from the same location. I also read in the Kaggle forum that you can google individual passengers, so i gave it a try:

Miss Amelie: https://www.encyclopedia-titanica.org/titanic-survivor/amelia-icard.html

Mrs. George Nelson: https://www.encyclopedia-titanica.org/titanic-survivor/martha-evelyn-stone.html

Regarding to the linked articles both embarked in Southhampton. Data science is about research, too!

```
df_all.loc[df_all['Embarked'].isnull(), 'Embarked'] = 'S'
```

2.5 Conclusion

We have filled every missing value in our data set and didn´t drop a column yet. We used statistical methods for age and fare, created a new category for cabin and did some research for the missings in embarked. Let´s have a double check if everything is fine now.

```
Missings in the data:
```

```
Age              0
Cabin         1014
Embarked         0
Fare             0
Name             0
Parch            0
PassengerId      0
Pclass           0
Sex              0
SibSp            0
Survived       418
Ticket           0
Deck             0
dtype: int64
```

3. Feature engineering

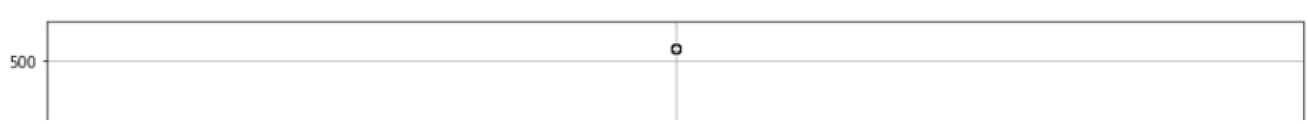
Feature engineering is an art and one of the most exciting things in the broad field of machine learning. I really enjoy to study the Kaggle subforums to explore all the great ideas and creative approaches. The titanic data set offers a lot of possibilities to try out different methods and to improve your prediction score. We will focus on some standards and I will explain every step in detail.

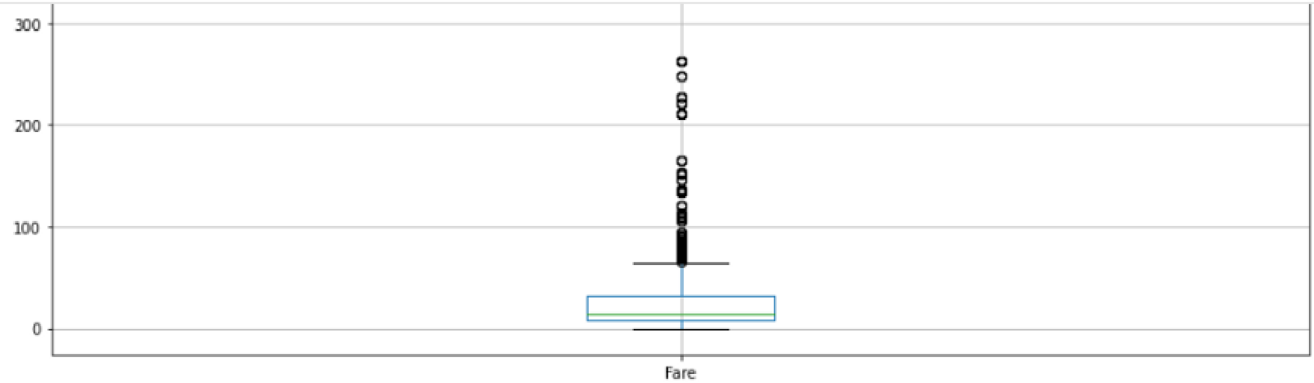
Techniques we will use so far:
- Binning continous variables (e.g. Age)
- Create new features out of existing variables (e.g. Title)
- Label encoding for non numeric features (e.g. Sex)
- One hot encoding for categorial features (e.g. Pclass)

3.1 Binning

```
df_all.boxplot(column=['Fare'], figsize=(15,7))
```

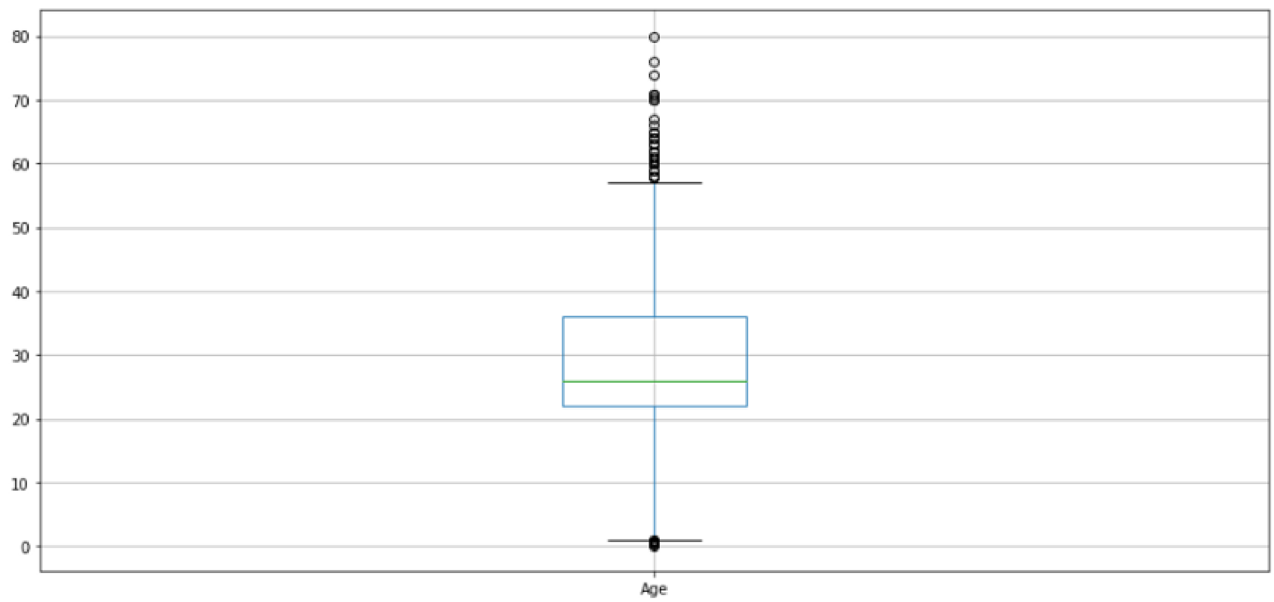
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8ff5566d68>
```






```
df_all.boxplot(column=['Age'], figsize=(15,7))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8ff54db7b8>
```



As you can see, there are outliers for both age and fare. The range of values is much higher for fare compared to age. We will cut the distribution into pieces so that the outliers do not irritate our algorithm. For fare we will assign the same number of cases to each category and for Age we will build the categories based on the values of the variable. This is also the difference between cut and qcut. With cut, the bins are formed based on the values of the variable, regardless of how many cases fall into a category. With qcut we decompose a distribution so that there are the same number of cases in each category.

+ Code + Markdown

```
print("For age, each category has a different number of cases:")
df_all['Age'].value_counts()
```

```
For age, each category has a different number of cases:
```

```
(16.0, 32.0]     752
(32.0, 48.0]     304
(-0.08, 16.0]    134
(48.0, 64.0]     106
(64.0, 80.0]      13
Name: Age, dtype: int64
```

```
print("For fare, each category has nearly a same number of cases:")
df_all['Fare'].value_counts()
```

```
For fare, each category has nearly a same number of cases:
```

```
(-0.001, 7.854]      275
(21.558, 41.579]     265
(41.579, 512.329]    259
(10.5, 21.558]       255
(7.854, 10.5]        255
Name: Fare, dtype: int64
```

On average, younger passengers have a higher chance of survival and so do people with higher ticket prices. Young people were probably rescued first and the people with higher ticket prices had access to the lifeboats first.

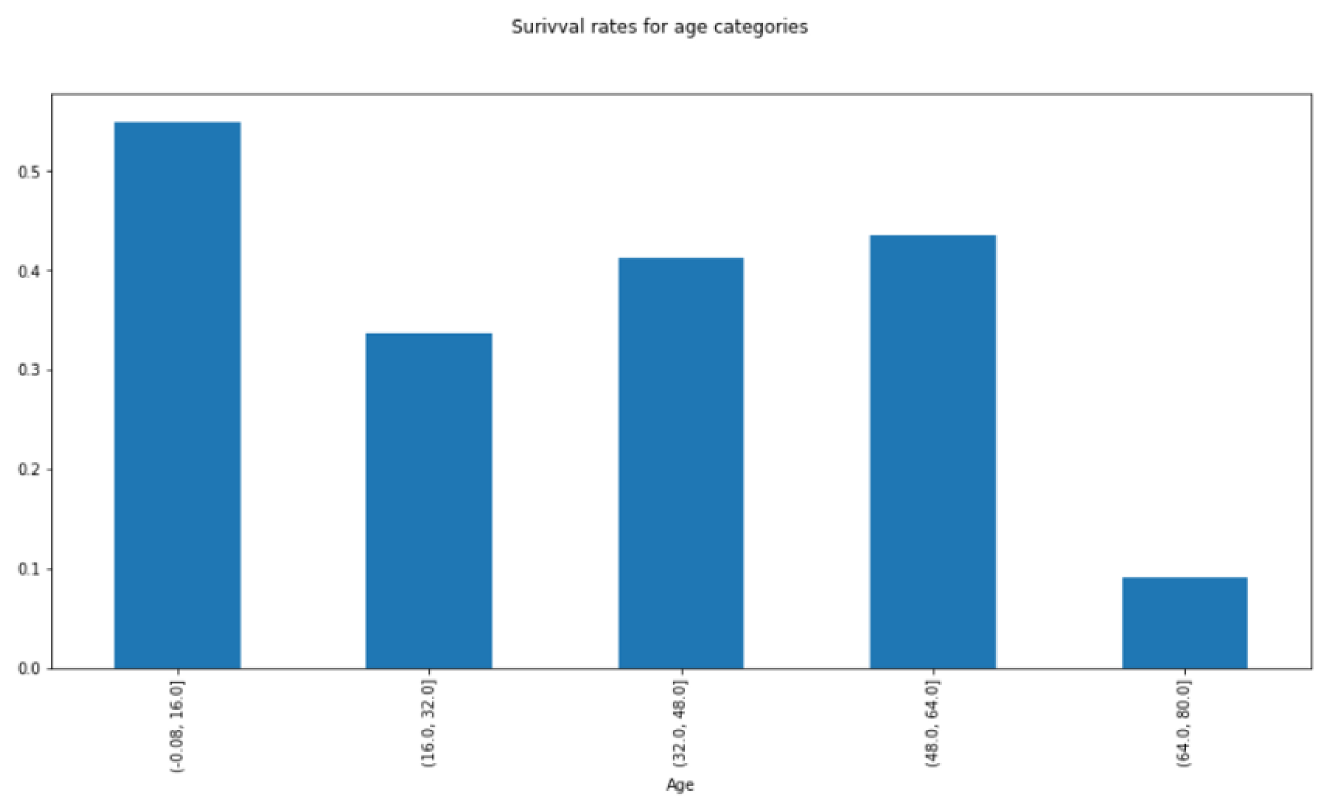
```
df_all[['Age','Survived']].groupby('Age')['Survived'].mean()
```

```
Age
(-0.08, 16.0]    0.550000
(16.0, 32.0]     0.337374
(32.0, 48.0]     0.412037
(48.0, 64.0]     0.434783
(64.0, 80.0]     0.090909
Name: Survived, dtype: float64
```

```
Fare
(-0.001, 7.854]      0.217877
(7.854, 10.5]        0.201087
(10.5, 21.558]       0.426901
(21.558, 41.579]     0.443243
(41.579, 512.329]    0.645349
Name: Survived, dtype: float64
```

```
df_all[['Age','Survived']].groupby('Age')['Survived'].mean().plot(kind='bar', figsize=(15,7))
pl.suptitle('Surivval rates for age categories')
```

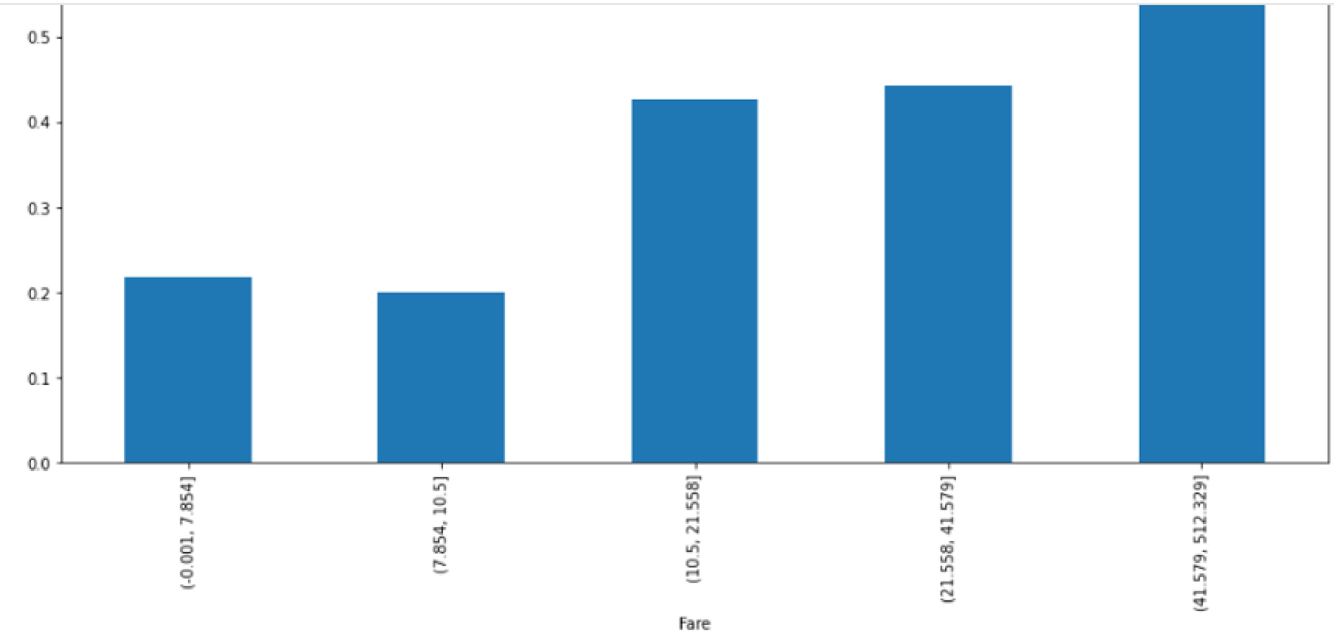
```
Text(0.5, 0.98, 'Surivval rates for age categories')
```




```
df_all[['Fare','Survived']].groupby('Fare')['Survived'].mean().plot(kind='bar', figsize=(15,7))
pl.suptitle('Surivval rates for fare categories')
```

```
Text(0.5, 0.98, 'Surivval rates for fare categories')
```

Surivval rates for fare categories



3.2 Create new features out of existing variables

3.2.1 Family Size

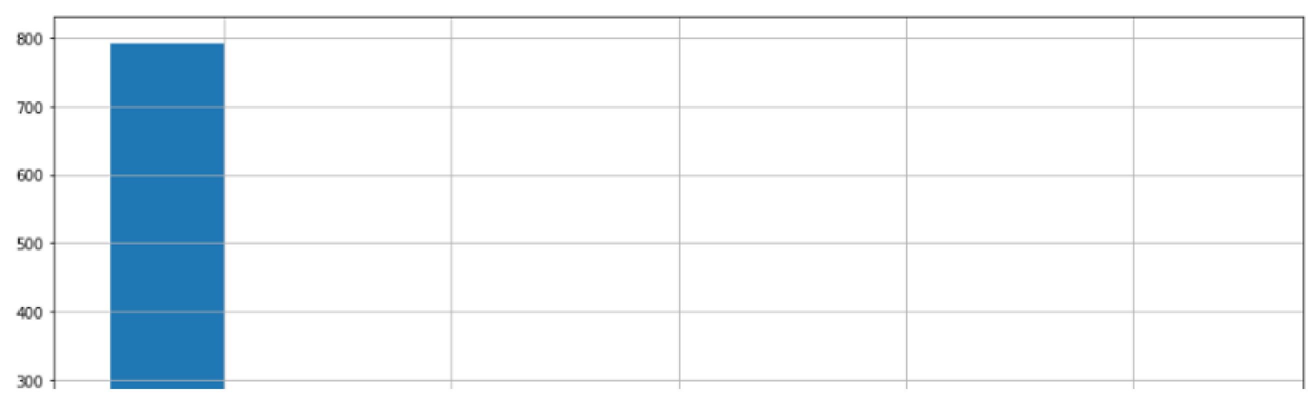
There are two interesting variables in our data set which tells us something about family size. SibSp defines how many siblings and spouses a passenger had and parch how many parents and childrens. We can summarize these variables and add 1 (for each passer-by) to get the family size.

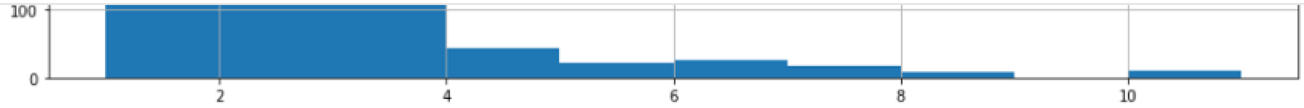
```
df_all['Family_Size'] = df_all['SibSp'] + df_all['Parch'] + 1
```

+ Code + Markdown

```
df_all['Family_Size'].hist(figsize=(15,7))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8ff55a9668>
```





```
df_all['Family_Size_bin'] = df_all['Family_Size'].map(lambda s: 1 if s == 1 else (2 if s == 2 else (3 if 3 <= s <= 4 else (4 if s >= 5 else 0))))
```

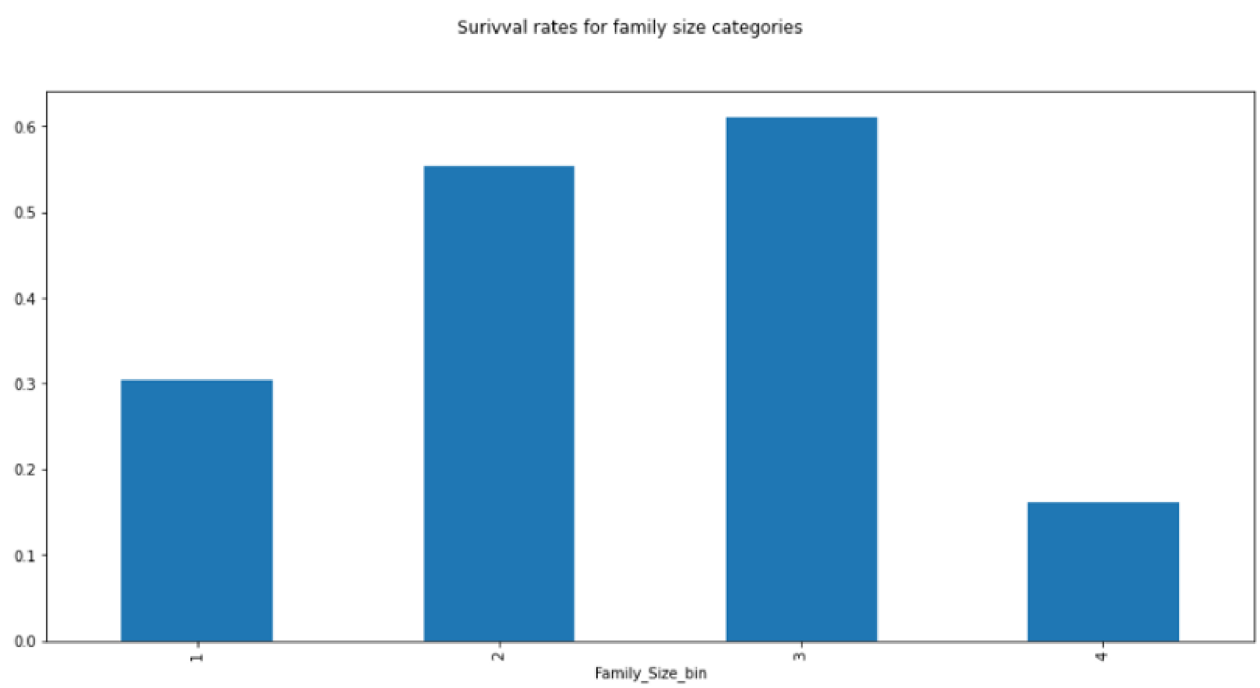
```
df_all['Family_Size_bin'].value_counts()
```

```
1    790
2    235
3    202
4     82
Name: Family_Size_bin, dtype: int64
```

One thesis is that families have a higher chance of survival than singles because they are better able to support themselves and were rescued with priority. However, if the families are too large, coordination is likely to be very difficult in an exceptional situation.

```
df_all[['Family_Size_bin','Survived']].groupby('Family_Size_bin')['Survived'].mean().plot(kind='bar', figsize=(15,7))
pl.suptitle('Surivval rates for family size categories')
```

```
Text(0.5, 0.98, 'Surivval rates for family size categories')
```



```
df_all['Ticket_Frequency'] = df_all.groupby('Ticket')['Ticket'].transform('count')
```

We expect a correlation between ticket frequencies and survival rates, because identical ticket numbers are an indicator that people have travelled together.

```
df_all[['Ticket_Frequency','Survived']].groupby('Ticket_Frequency').mean()
```

]:

Ticket_Frequency	Survived
1	0.270270
2	0.513812
3	0.653465
4	0.727273
5	0.333333
6	0.210526
7	0.208333
8	0.384615
11	0.000000

As expected there are some differences between the survival rates for each ticket frequency.

3.2.3 Title

The name provides us very important information about the socioeconomic status of a passenger. We can answer the question if someone is married or not or if someone has a formal title which could be an indicator for a higher social status.

```
df_all['Title'] = df_all['Name'].str.split(', ', expand=True)[1].str.split('.', expand=True)[0]
df_all['Is_Married'] = 0
df_all['Is_Married'].loc[df_all['Title'] == 'Mrs'] = 1
```

```
18
```

There are quite a lot of different titles in our data set. We only consider title with more than 10 cases, all others we will assign to the category “misc”.

```
title_names = (df_all['Title'].value_counts() < 10)

df_all['Title'] = df_all['Title'].apply(lambda x: 'Misc' if title_names.loc[x] == True else x)

df_all.groupby('Title')['Title'].count()
```

```
Title
Master     61
Misc       34
Miss      260
Mr        757
Mrs       197
Name: Title, dtype: int64
```

3.2.4 Survival rates

This Kaggle Competetion allows us to use information from the test data set. At this point we would like to point out that for high scores you have to be creative with the data. It is almost like a hackathon. In a Realworld task, you would not normally have the opportunity to do this.
We will identify family names of passengers. Then we can see if there are any family members that are present in both the training and the test data set.

```
import string

def extract_surname(data):
    families = []
    for i in range(len(data)):
        name = data.iloc[i]
        if '(' in name:
            name_no_bracket = name.split('(')[0]
        else:
```

```
        for c in string.punctuation:
            family = family.replace(c, '').strip()
        families.append(family)
    return families

df_all['Family'] = extract_surname(df_all['Name'])
```

+ Code + Markdown

```
df_all['Family'].nunique()
```

```
875
```

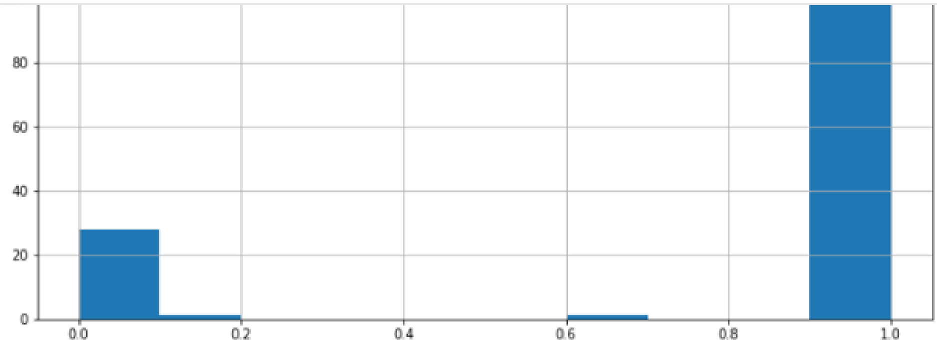
People with a Master's degree and women have survived significantly more often and, on average, have larger families at the same time. We assume that if a master or woman is marked as a survivor in the training data set, family members in the test data set will also have survived.

```
df_all[['Title','Survived','Family_Size']].groupby('Title').mean()
```

Title	Survived	Family_Size
Master	0.575000	4.426230
Misc	0.444444	1.441176
Miss	0.697802	2.169231
Mr	0.156673	1.442536
Mrs	0.792000	2.492386

```
print("Survival rates grouped by families of women in dataset:")
df_all.loc[(df_all['Sex'] == 'female') & (df_all['Family_Size'] > 1)].groupby('Family')['Survived'].mean().hist(figsize=(12,5))
```

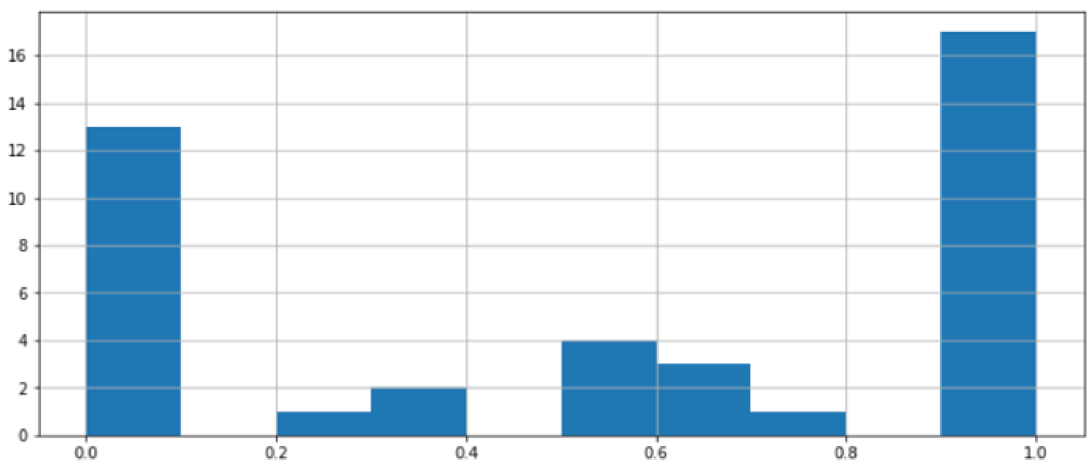
```
Survival rates grouped by families of women in dataset:
```



In women with a family size of 2 or more, most often all or none of them die.

```
master_families = df_all.loc[df_all['Title'] == 'Master']['Family'].tolist()
df_all.loc[df_all['Family'].isin(master_families)].groupby('Family')['Survived'].mean().hist(figsize=(12,5))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8ff53af7f0>
```



The same applies for families of passengers with master in their title.

```
combined_rate = women_rate.append(master_rate)
#It is possible that a women has the family as a master and vice versa, so duplicates have to been dropped
combined_rate_df = combined_rate.to_frame().reset_index().rename(columns={'Survived':'Survival_quota'}).drop_duplicates(subset='Family')
#Merge the new dataframe
df_all = pd.merge(df_all,combined_rate_df, how='left')
```

```
#We have calculated a survival rate for only a part of the cases, the other cases we set to 0 in the dummy variable
df_all['Survival_quota_NA'] = 1
df_all.loc[df_all['Survival_quota'].isnull(), 'Survival_quota_NA'] = 0
df_all['Survival_quota'] = df_all['Survival_quota'].fillna(0)
```

3.3 Label- and One Hot Encoding

Most algorithms cannot do anything with strings, so the variables are often recoded before modeling. Label Encoding maps non-numerical values to numbers. For sex, for example, 0 and len(sex)-1, which is, 1.

This leads to another problem. Many algorithms assume that there is a logical sequence within a column. However, this is not always expressed by the numerical ratio. Therefore it is needed to one hot encoding the variables afterwards. The column Sex then becomes two columns Sex_1 and Sex_2, in which it is binary coded whether someone was male or female. So the algorithm can usually process the information better.

```
    df_all[feature] = LabelEncoder().fit_transform(df_all[feature])


cat_features = ['Pclass', 'Sex', 'Embarked', 'Title', 'Deck', 'Family_Size_bin','Age','Fare']

encoded_features = []

for feature in cat_features:
    encoded_feat = OneHotEncoder().fit_transform(df_all[feature].values.reshape(-1, 1)).toarray()
    n = df_all[feature].nunique()
    cols = ['{}_{}'.format(feature, n) for n in range(1, n + 1)]
    encoded_df = pd.DataFrame(encoded_feat, columns=cols)
    encoded_df.index = df_all.index
    encoded_features.append(encoded_df)

df_all = pd.concat([df_all, *encoded_features], axis=1)
```

+ Code + Markdown

```
df_train, df_test = divide_df(df_all)
```

4. Modelling and prediction

For our first prediction we choose a Random Forrest Classifier. RFCs are easy to understand and proven tools for classification tasks.

We still define the columns that we do not need to consider for modelling. For Embarked, for example, we have created dummy columns, so we can drop the original Embarked column. As training/test split we choose 75% and 25%. We train the algorithm with the training data set and then test predictive power with the test data set.

The criteria in brackets for RFC are not mandatory, if you leave them out, default settings are used. The given parameters are already optimized so that our classifier works better than with the default parameters.

```
#Define columns which can be dropped for the modelling part because we created new label and one hot encoded variants out of them
drop_cols =  ['Embarked', 'Family','Family_Size', 'Survived', 'Family_Size_bin', 'Deck', 'Age',
             'Name', 'Parch', 'PassengerId', 'Pclass', 'Sex', 'SibSp', 'Title', 'Ticket','Cabin']

drop_cols_2 =  ['Embarked', 'Family','Family_Size', 'Family_Size_bin', 'Deck', 'Fare',
             'Name', 'Parch', 'PassengerId', 'Pclass', 'Sex', 'SibSp', 'Title', 'Ticket','Cabin']
```

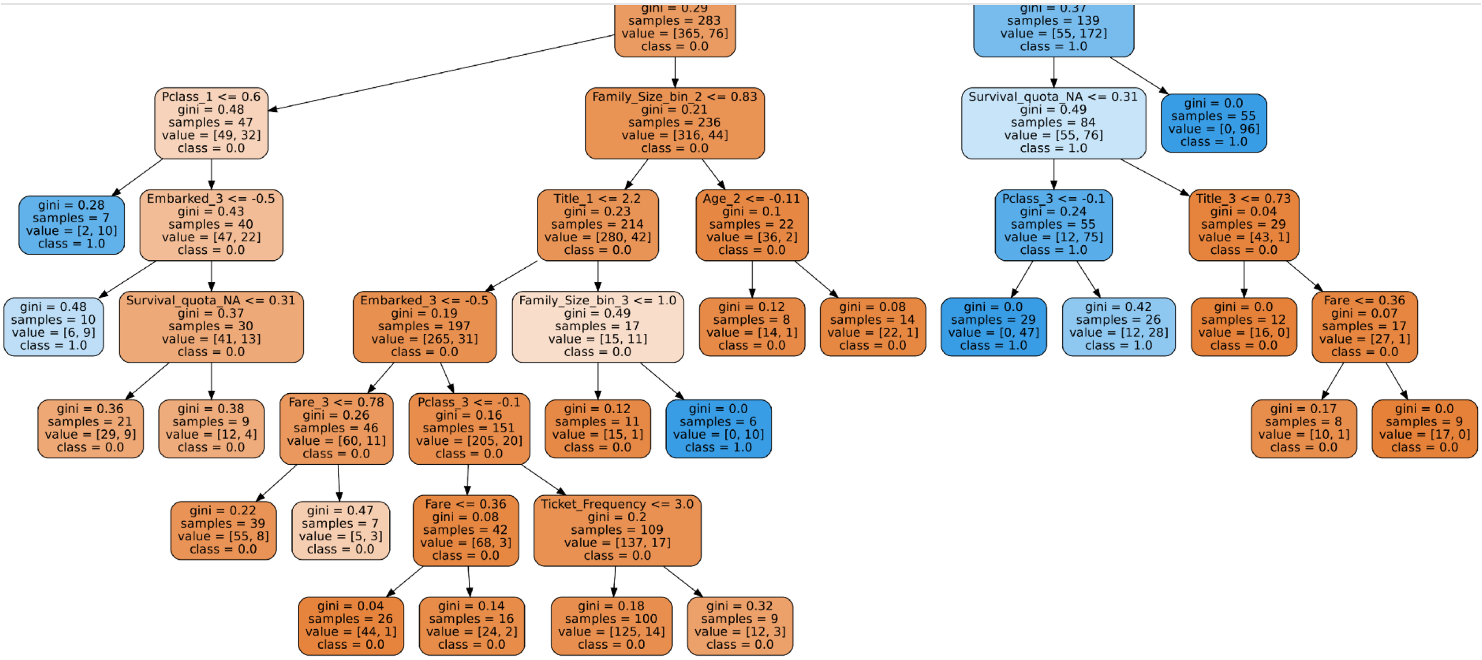
```
#setting up a random forest classifier
#standardization of the variables
X = StandardScaler().fit_transform(df_train.drop(columns=drop_cols))
y = df_train['Survived'].values
X_test = StandardScaler().fit_transform(df_test.drop(columns=drop_cols_2))
```

```
                                             max_depth=7,
                                             min_samples_split=6,
                                             min_samples_leaf=6,
                                             max_features='auto',
                                             oob_score=True,
                                             random_state=42,
                                             n_jobs=-1,
                                             verbose=1)
model.fit(X_train, y_train)
predictions = model.predict(X_test)
print(model.score(X_test1, y_test1))
output = pd.DataFrame({'PassengerId': test_data.PassengerId, 'Survived': predictions})
output['Survived'] = output['Survived'].astype(int)
output.to_csv('2020_04_09_bd_final_v3.csv', index=False)
```

```
[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:    0.1s
[Parallel(n_jobs=-1)]: Done 192 tasks      | elapsed:    0.6s
[Parallel(n_jobs=-1)]: Done 442 tasks      | elapsed:    1.3s
[Parallel(n_jobs=-1)]: Done 792 tasks      | elapsed:    2.3s
[Parallel(n_jobs=-1)]: Done 1242 tasks      | elapsed:    3.5s
[Parallel(n_jobs=-1)]: Done 1750 out of 1750 | elapsed:    5.0s finished
[Parallel(n_jobs=4)]: Using backend ThreadingBackend with 4 concurrent workers.
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=4)]: Done 192 tasks      | elapsed:    0.1s
[Parallel(n_jobs=4)]: Done 442 tasks      | elapsed:    0.2s
[Parallel(n_jobs=4)]: Done 792 tasks      | elapsed:    0.4s
[Parallel(n_jobs=4)]: Done 1242 tasks      | elapsed:    0.6s
[Parallel(n_jobs=4)]: Done 1750 out of 1750 | elapsed:    0.8s finished
[Parallel(n_jobs=4)]: Using backend ThreadingBackend with 4 concurrent workers.
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=4)]: Done 192 tasks      | elapsed:    0.1s
[Parallel(n_jobs=4)]: Done 442 tasks      | elapsed:    0.2s
[Parallel(n_jobs=4)]: Done 792 tasks      | elapsed:    0.3s
[Parallel(n_jobs=4)]: Done 1242 tasks      | elapsed:    0.5s
[Parallel(n_jobs=4)]: Done 1750 out of 1750 | elapsed:    0.7s finished
```

```
0.8654708520179372
```

Our predicting score is almost 86%, which means that we have correctly predicted our target, i.e. the survival rate, in 86% of cases. This is already a good value, which you can now further optimize. Please find below a viszualization of our random forrest tree.



5. Conclusion

We made the entire journey in a small data science project. We explored the data, cleaned up the data, then we modified features and created new ones and in a last step we made a prediction with a random forest tree classifier. But there is still a lot to do, next you can test the following things:
- Do other algorithms perform better?
- Can you choose the bins for Age and Fare better?
- Can the ticket variable be used more reasonable?
- Is it possible to further adjust the survival rate?
- Do we really need all features or do we create unnecessary noise that interferes with our algorithm?

Below you find some great resources to start with.

6. Further reading & resources:

Titanic Data Science Solutions

Explore and run machine learning code with Kaggle Notebooks | Using data from Titanic: Machine Learning from Disaster

www.kaggle.com

A Data Science Framework: To Achieve 99% Accuracy

Explore and run machine learning code with Kaggle Notebooks | Using data from Titanic: Machine Learning from Disaster

www.kaggle.com

Titanic - Advanced Feature Engineering Tutorial

Explore and run machine learning code with Kaggle Notebooks | Using data from Titanic: Machine Learning from Disaster

www.kaggle.com

If you like the article, I would be glad if you follow me. I regularly publish new articles related to Data Science. If you have any questions, feel free to leave me a message or a comment. Check my profile for other tutorials.

Data Science