01-kaggle-titanic

December 21, 2018

```
<span style="font-size: 32px; padding: 0px"> Kaggle Competition Nr. 1 </span>
<span style="font-size: 24px; padding: 0px"> Titanic Challenge am 8.12.18 </span>
```

1 Einleitung

1.1 Ein Vorgeschmack von KI

Spannende Links zum Thema ML
-Genetic Algorithm
Leseliste:

- Data Types in Statistics

Dokumentation zu den Paketen:

- Seaborn
- Tensorflow
- Tensorflow Playground

1.2 Vorbereitung : Python lernen

1.3 Was ist Kaggle ?

Titanic

1.4 Ziel

2 Titanic Challenge

2.1 Wie fängt man an ?

2.1.1 Welche Schritte müssen generell gemacht werden ?

2.1.2 Tipps zum Umgang mit Jupyter Notebook

- **Tab**: zur Code IntelliSense
- **Shift + Tab**: ruft Dokumentation auf
- **Shift + Enter**: Zelle ausführen

Shortcuts zum Editor:

- CTRL + B : Linkes Menü verstecken
- CTRL + SHIFT + D: Single Document Mode - versteckt Tabs für mehrere Dateien

2.1.3 Hilfreiche Links mit Ideen zur Lösung von Titanic

Kurse und Tipps: - Kaggle Tutorials Section - Kaggle Learn Center - Kaggle ML Kurs (4h)
Walkthrough: - titanic-eda-to-ml-beginner - titanic-survival-seaborn-and-ensembles - how to score 81.34 in titanic

2.2 Lets do it !

2.2.1 Daten visualisieren und verstehen

Dieser Schritt ist auch bekannt als **Exploratory Data Analysis**

Daten laden und erste Erkenntnisse Bevor wir unser Datenset visualisieren können müssen wir die Daten herunterladen und in Jupyter importieren.

Pandas ist eine Python Bibliothek, die hauptsächlich für das Erforschen und manipulieren der Daten genutzt wird. Üblicherweise wird Pandas mit *pd* abgekürzt.

```
In [20]: import pandas as pd

         data_url_train = "./train.csv"
         data_url_test = "./test.csv"

         data = pd.read_csv(data_url_train)
```

Das ging schnell und einfach, also schauen wir uns das Objekt (*dataframe*) mal an.

```
In [21]: print("Datensätze: " + str(data.shape[0]))
         print("Spalten: " + str(data.shape[1]))
```

```
Datensätze: 891
Spalten: 12
```

```
In [22]: data.columns
```

```
Out[22]: Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
                'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'],
               dtype='object')
```

```
In [23]: data.head()
         # data.head(10)
```

```
Out[23]:    PassengerId  Survived  Pclass  \
         0            1         0       3
         1            2         1       1
         2            3         1       3
         3            4         1       1
```

```
4            5         0       3

                                                Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0      1
2                             Heikkinen, Miss. Laina  female  26.0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                           Allen, Mr. William Henry    male  35.0      0

   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
3      0            113803  53.1000  C123        S
4      0            373450   8.0500   NaN        S
```

```
In [24]: data.describe()
```

```
Out[24]:        PassengerId    Survived      Pclass         Age       SibSp  \
count   891.000000  891.000000  891.000000  714.000000  891.000000
mean    446.000000    0.383838    2.308642   29.699118    0.523008
std     257.353842    0.486592    0.836071   14.526497    1.102743
min       1.000000    0.000000    1.000000    0.420000    0.000000
25%     223.500000    0.000000    2.000000   20.125000    0.000000
50%     446.000000    0.000000    3.000000   28.000000    0.000000
75%     668.500000    1.000000    3.000000   38.000000    1.000000
max     891.000000    1.000000    3.000000   80.000000    8.000000

            Parch        Fare
count  891.000000  891.000000
mean     0.381594   32.204208
std      0.806057   49.693429
min      0.000000    0.000000
25%      0.000000    7.910400
50%      0.000000   14.454200
75%      0.000000   31.000000
max      6.000000  512.329200
```

Was sehen wir in den Daten ?

Unser Modell soll die Überlebenswahrscheinlichkeit an Hand der Features bestimmen. Die entscheidende Spalte **Survived** ist unser **Output Y**. Alle anderen Spalten können wir als **Features** oder **Input-Werte** betrachten.

AuSSerdem können wir mit der `info()` Funktion uns noch schnell alle Datentypen anzeigen lassen, die aktuell in unserem Trainingsdaten vorliegen

```
In [25]: data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
PassengerId    891 non-null int64
Survived       891 non-null int64
Pclass         891 non-null int64
Name           891 non-null object
Sex            891 non-null object
Age            714 non-null float64
SibSp          891 non-null int64
Parch          891 non-null int64
Ticket         891 non-null object
Fare           891 non-null float64
Cabin          204 non-null object
Embarked       889 non-null object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.6+ KB
```

Fehlende Werte:

Wenn man sich die *count* der einzelnen *columns* genauer ansieht, fällt auf, dass im Datenset für bspw. das *Alter* nicht immer 891 Datensätze vorhanden sind.

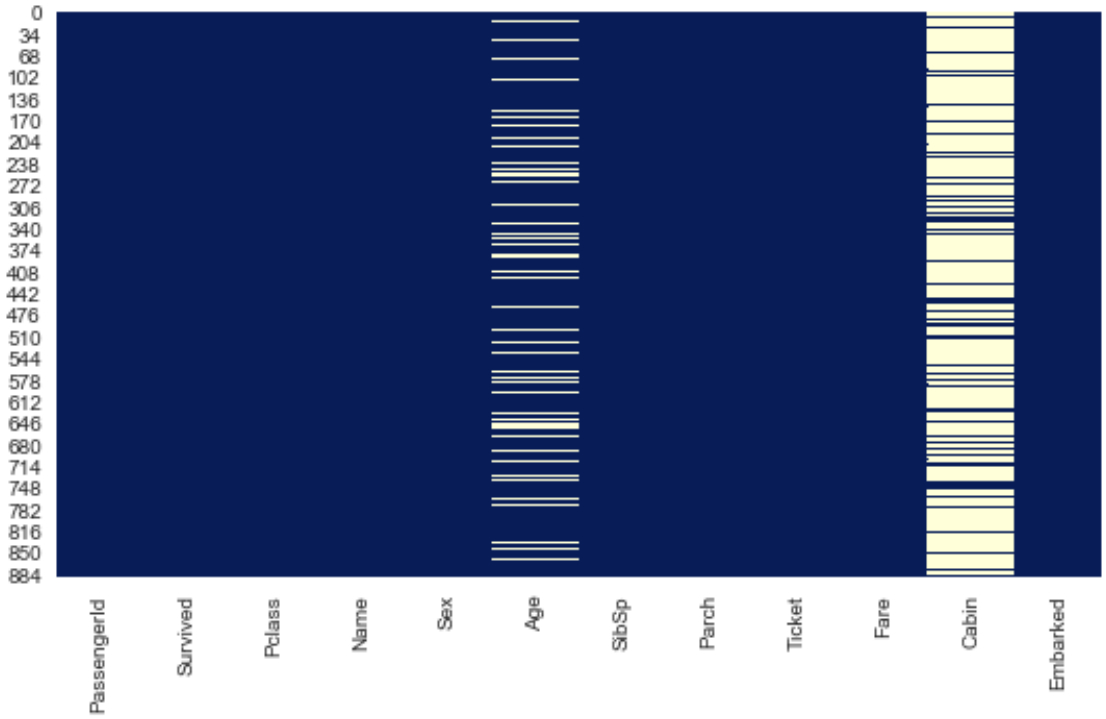
Wir haben also fehlende Werte, dafür müssen wir uns später eine Strategie überlegen.

Daten visualisieren

```
In [31]: import matplotlib.pyplot as plt
         import seaborn as sns

         %matplotlib inline
         sns.set()
```

Fehlende Daten anzeigen als Heatmap

```
In [32]: fig, ax = plt.subplots(figsize=(9,5))
         sns.heatmap(data.isnull(), cbar=False, cmap="YlGnBu_r")
         plt.show()
```



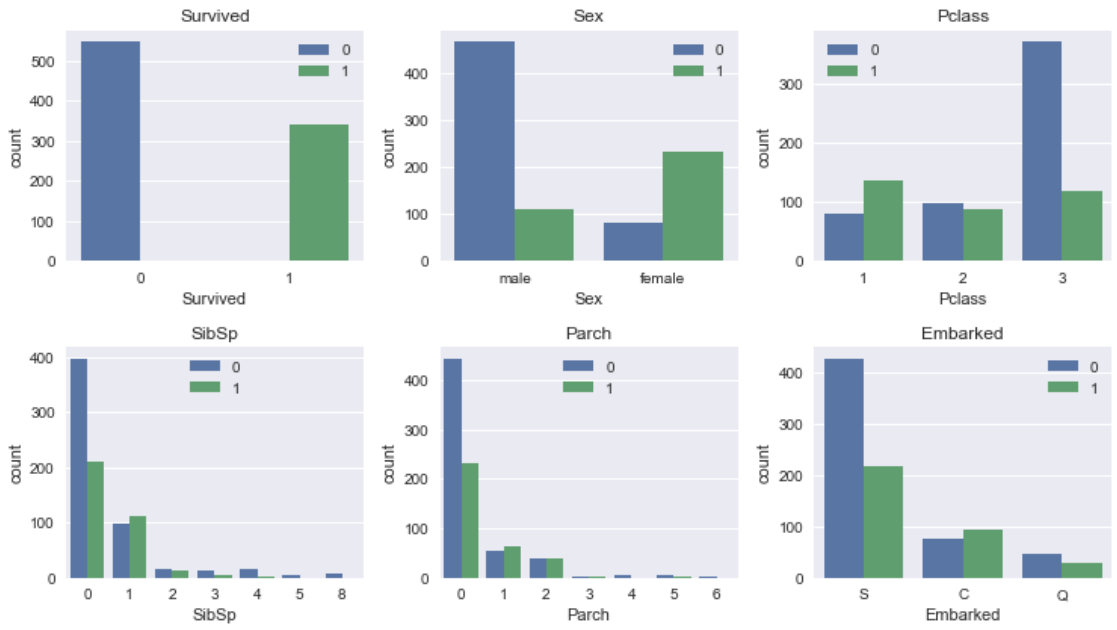
Plotten von *categorical data*

categorical data - ist ein statistischer Datentyp - nominal skalierte Variablen (bspw. Familiennamen, Blutgruppen) - ordinal skalierte Variablen (bspw. Leistung 1-6, Militärrang) - beinhaltet manchmal (je nach Ausprägung und Autor) - metrische Variablen - metrische kategorisierte Variablen (bspw. Altersgruppen 0-6, 7-12)

Im folgenden werden direkt mehrere Plots erzeugt und angezeigt.

```
In [55]: cols = ['Survived', 'Sex', 'Pclass', 'SibSp', 'Parch', 'Embarked']

         nr_rows = 2
         nr_cols = 3

         fig, axs = plt.subplots(nr_rows, nr_cols, figsize=(nr_cols*3.5,nr_rows*3))

         for r in range(0,nr_rows):
             for c in range(0,nr_cols):

                 i = r*nr_cols+c
                 ax = axs[r][c]
                 sns.countplot(data[cols[i]], hue=data["Survived"], ax=ax)
                 ax.set_title(cols[i])
                 ax.legend()
```

```
plt.tight_layout()
```

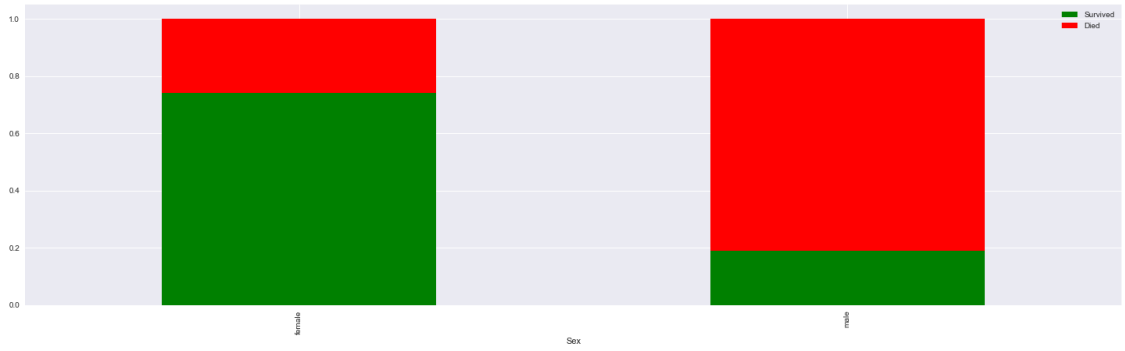


Was sehen wir in den Plots?

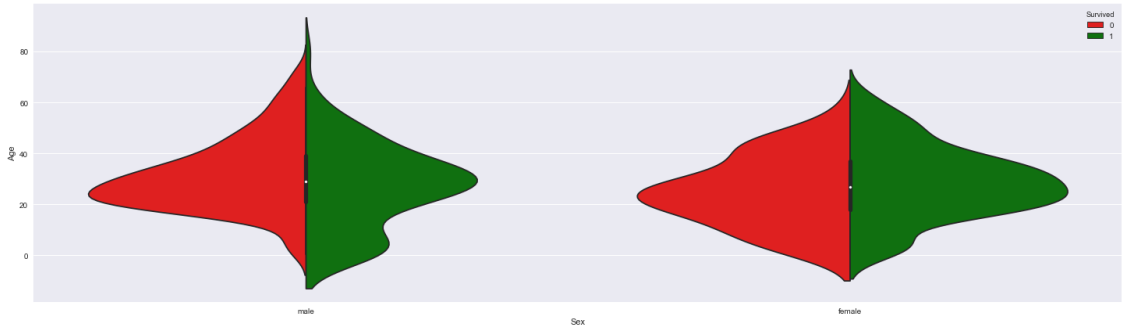
1. Es haben von den ca. 900 Passagieren aus den Trainingsdaten weniger als 350 überlebt.
2. Mehr Frauen als Männer haben überlebt
3. In der *Upper, Middle* Klasse ist die Überlebenswahrscheinlichkeit deutlich höher.
4. SibSp + Parch: Bei der Reise mit Eltern oder Verwandschaft ist die Wahrscheinlichkeit höher als für Alleinreisende.
5. Embarked: Passagiere die in Cherbourg (C) eingestiegen sind haben eine bessere Überlebenschance als Queenstown (Q) und Southhampton (S).

Detailansicht: Überlebenswahrscheinlichkeit Frau-Mann Um noch mehr Diagrammtypen zu testen & die Daten tiefergehend zu betrachten werden wir uns einige Diagramme von oben nochmal einzelnd vornehmen.

```
In [42]: data['Died'] = 1 - data['Survived']
         data.groupby('Sex').agg('mean')[['Survived', 'Died']].plot(kind='bar', figsize=(25, 7),
```




```
In [43]: fig = plt.figure(figsize=(25, 7))
         sns.violinplot(x='Sex', y='Age',
                        hue='Survived', data=data,
                        split=True,
                        palette={0: "r", 1: "g"}
                       );
```



Im oberen Diagramm können wir auSSerdem sehen, dass das Alter für Männer scheinbar einen Einfluss hat (20-30 Jahre alt), da dort eine erhöhte Menge eher stirbt als überlebt.

Bei den Frauen hingegen sind die beiden Seiten für das Alter ausgeglichener und scheint die Überlebenswahrscheinlichkeit nicht zu beinflussen.

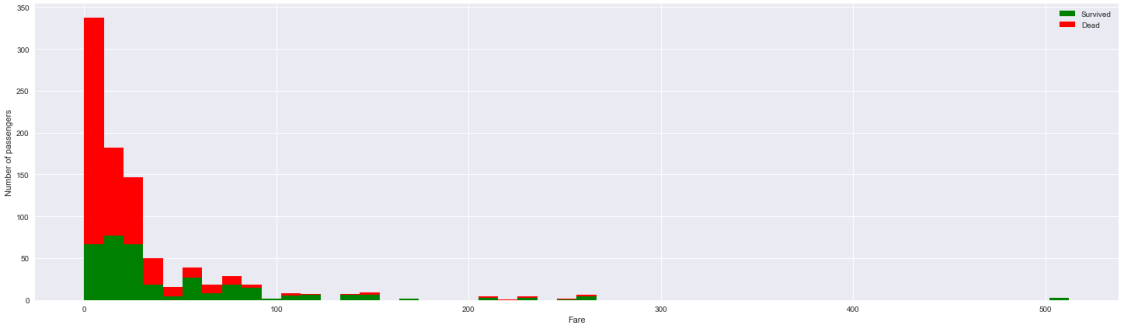
AuSSerdem sieht man im unteren Bereich der männlichen Kinder (0-20), dass die Wahrscheinlichkeit zu überleben deutlich steigt. Bei den weiblichen, bereits bevorzugten Teilnehmern lässt sich die Tendenz auch schwach erkennen.

Rettet Frauen und Kinder zu erst !

Einfluss des Ticketpreises

```
In [44]: figure = plt.figure(figsize=(25, 7))
         plt.hist([data[data['Survived'] == 1]['Fare'], data[data['Survived'] == 0]['Fare']],
                  stacked=True, color = ['g','r'],
                  bins = 50, label = ['Survived','Dead'])
```

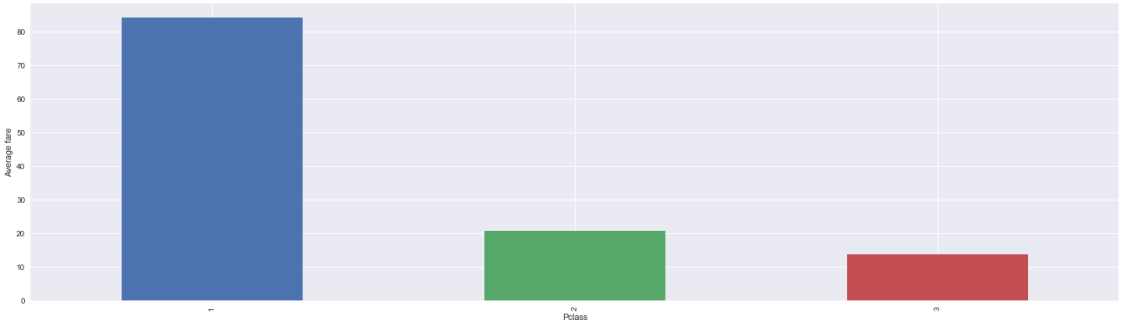
```
plt.xlabel('Fare')
plt.ylabel('Number of passengers')
plt.legend();
```



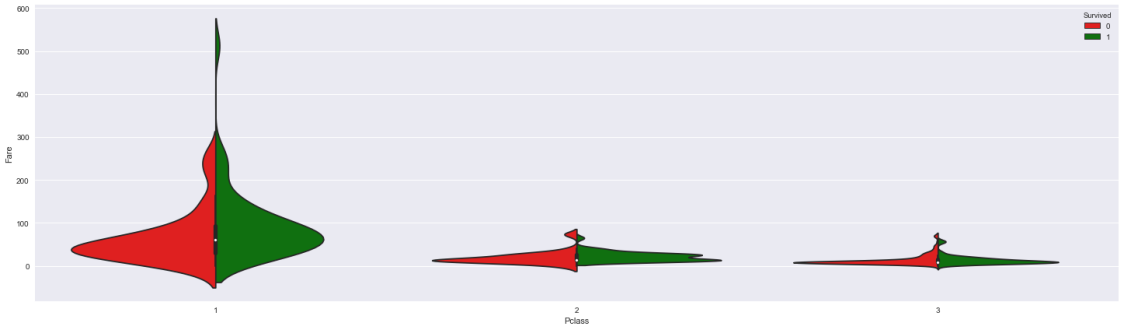
Passagiere, die wenig bezahlt haben sind eher gestorben als Passagiere in den höheren Preisklassen.

Korrelation zwischen Pclass & Fare.

```
In [45]: ax = plt.subplot()
         ax.set_ylabel('Average fare')
         data.groupby('Pclass').mean()['Fare'].plot(kind='bar', figsize=(25, 7), ax = ax);
```



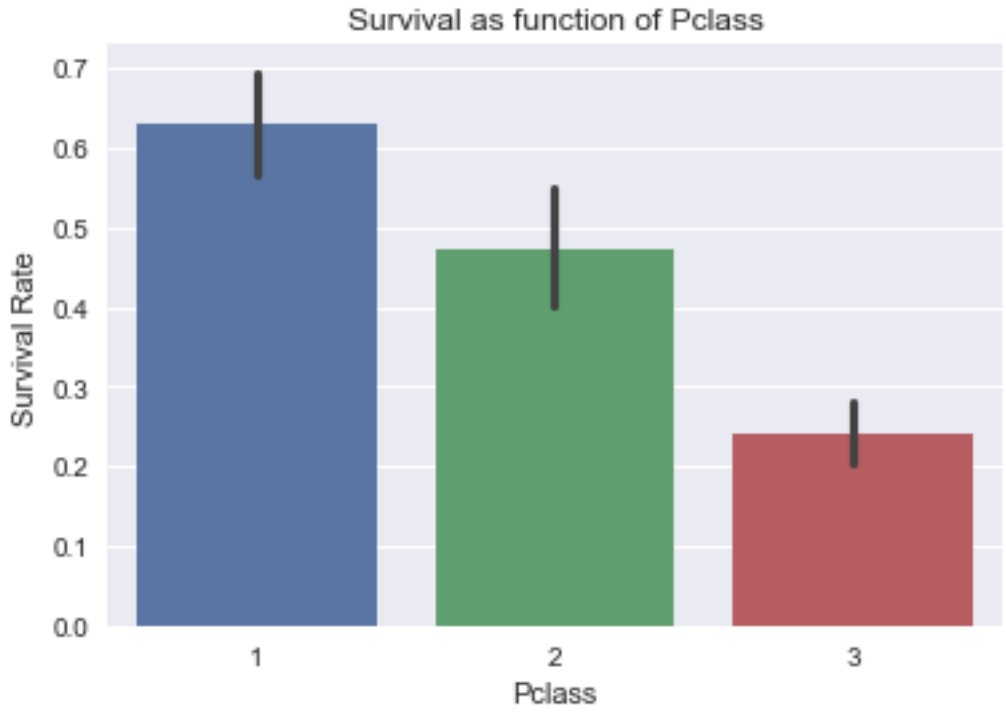

```
In [49]: fig = plt.figure(figsize=(25, 7))
         sns.violinplot(x='Pclass', y='Fare', hue='Survived', data=data, split=True, palette={0:
```



In den beiden oberen Diagrammen erkennen wir eine Korrelation zwischen den Ticketpreisen und den Ticketklassen.

Überlebensrate von Pclass

```
In [51]: sns.barplot(x='Pclass', y='Survived', data=data)
         plt.ylabel("Survival Rate")
         plt.title("Survival as function of Pclass")
         plt.show()
```



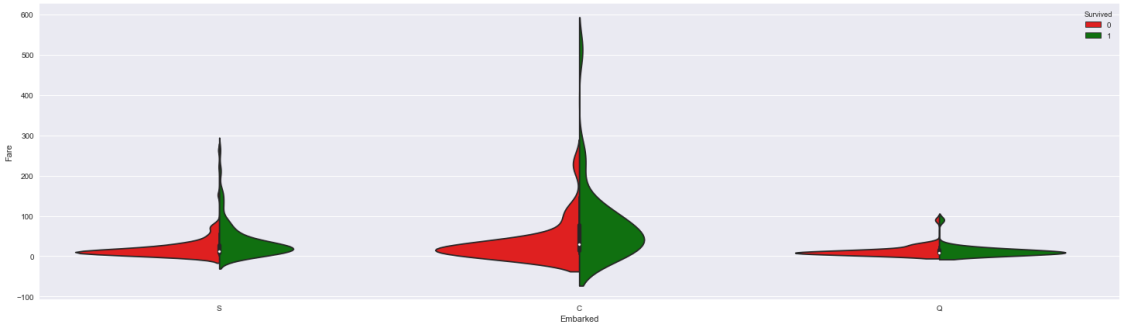
Wenn wir zusätzlich, dass Geschlecht betrachten:

```
In [53]: sns.barplot(x='Sex', y='Survived', hue='Pclass', data=data)
         plt.ylabel("Survival Rate")
         plt.title("Survival as function of Pclass and Sex")
         plt.show()
```

- Höchste Überlebensrate (> 90%) für Frauen aus den Klassen 1 oder 2
- Niedrigste Überlebensrate (< 20%) für Männer aus der 3. Klasse

Einstiegshafen (Embarked)

```
In [54]: fig = plt.figure(figsize=(25, 7))
         sns.violinplot(x='Embarked', y='Fare', hue='Survived', data=data, split=True, palette={
```

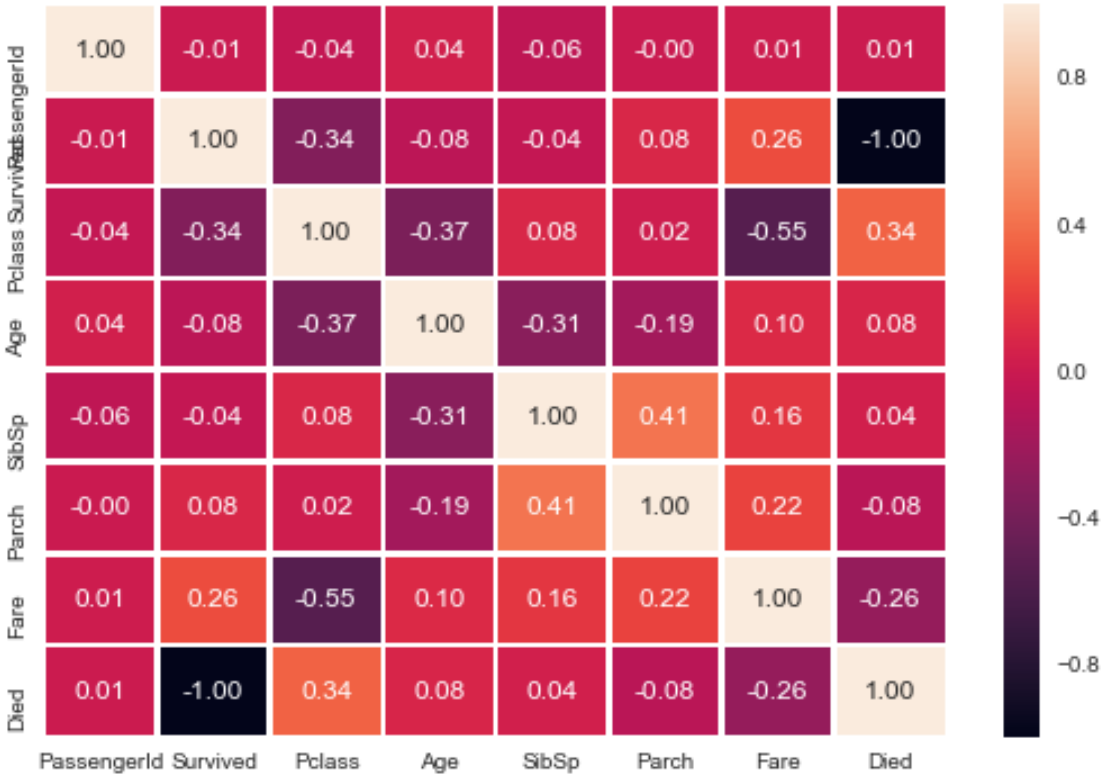


- Standort Queenstown hat die niedrigsten Ticketpreise
- Standort Cherbourg die höchste Spanne an Ticketpreisen und den höchsten Durchschnittspreis

Die Route des Schiffs sollte Southhampton-Cherbourg-Queenstown sein, also lässt sich der niedrigste Durchschnittspreis evtl dadurch erklären, dass die Route dort am kürzesten nach New York war.

Korrealations Matrix

```
In [57]: corr = data.corr()

         f,ax = plt.subplots(figsize=(9,6))
         sns.heatmap(corr, annot = True, linewidths=1.5 , fmt = '.2f',ax=ax)
         plt.show()
```



- Survived and Fare positively correlated
- Survived and Sex_male negatively correlated.
- Survived and Pclass_3 negatively correlated
- SibSp and Parch correlated

2.2.2 Daten vorbereiten

Nicht notwendige Features entfernen Einige Features haben für das Modell überhaupt keinen Einfluss und man kann diese logischerweise entfernen.

- PassengerId

- Name
- Ticket

Jedoch bleiben immer noch sehr viele Features übrig, für unser erstes Modell lassen wir bewusst einige Informationen weg, die wir im späteren Kapitel dann zum Verbessern des Modells einsetzen.

- Cabin (zu viele fehlende Werte)
- SibSp & Parch: -> Mehr dazu später im erweiterten Kaptitel **Feature Engineering**

```
In [168]: data = pd.read_csv(data_url_train)
          test = pd.read_csv(data_url_test)

          # or also
          # data.drop(['Died']) # Just used for plotting

          def dropFeatures(data): # as function to also do it for test set later.
              data.drop(['Ticket', 'Name', 'PassengerId', 'Cabin'], axis=1, inplace=True)
             # data.drop(['SibSp','Parch'], axis=1, inplace=True) # Will be used later

          dropFeatures(data)
          dropFeatures(test)
          data.head()
```

Out[168]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	0	3	male	22.0	1	0	7.2500	S
1	1	1	female	38.0	1	0	71.2833	C
2	1	3	female	26.0	0	0	7.9250	S
3	1	1	female	35.0	1	0	53.1000	S
4	0	3	male	35.0	0	0	8.0500	S

Fehlende Werte behandeln Wir müssen einige fehlende Werte füllen:

- Age
- Fare
- Embarked

Der Median - Wert eignet sich oftmals besser als der Mittelwert, da *Outliers* so nicht den Wert beinflussen

```
In [157]: data.info()
          test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 10 columns):
Survived      891 non-null int64
Age           891 non-null float64
```

```
SibSp          891 non-null int64
Parch          891 non-null int64
Fare           891 non-null float64
Sex_male       891 non-null uint8
Embarked_Q     891 non-null uint8
Embarked_S     891 non-null uint8
Pclass_2       891 non-null uint8
Pclass_3       891 non-null uint8
dtypes: float64(2), int64(3), uint8(5)
memory usage: 39.2 KB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 9 columns):
Age            418 non-null float64
SibSp          418 non-null int64
Parch          418 non-null int64
Fare           417 non-null float64
Sex_male       418 non-null uint8
Embarked_Q     418 non-null uint8
Embarked_S     418 non-null uint8
Pclass_2       418 non-null uint8
Pclass_3       418 non-null uint8
dtypes: float64(2), int64(2), uint8(5)
memory usage: 15.2 KB
```

```
In [158]: data.fillna({'Age': data['Age'].median()}, inplace=True)
          test.fillna({'Age': data['Age'].median()}, inplace=True)

          data.fillna({'Fare': data['Fare'].median()}, inplace=True)
          test.fillna({'Fare': data['Fare'].median()}, inplace=True)

          data.info()
          test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 10 columns):
Survived       891 non-null int64
Age            891 non-null float64
SibSp          891 non-null int64
Parch          891 non-null int64
Fare           891 non-null float64
Sex_male       891 non-null uint8
Embarked_Q     891 non-null uint8
Embarked_S     891 non-null uint8
Pclass_2       891 non-null uint8
Pclass_3       891 non-null uint8
```

```
dtypes: float64(2), int64(3), uint8(5)
memory usage: 39.2 KB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 9 columns):
Age           418 non-null float64
SibSp         418 non-null int64
Parch         418 non-null int64
Fare          418 non-null float64
Sex_male      418 non-null uint8
Embarked_Q    418 non-null uint8
Embarked_S    418 non-null uint8
Pclass_2      418 non-null uint8
Pclass_3      418 non-null uint8
dtypes: float64(2), int64(2), uint8(5)
memory usage: 15.2 KB
```

Daten konvertieren Bevor wir nun unser Modell trainieren können, müssen wir noch einige in numerische Werte umwandeln.

- Age
- Fare

Und auch - Sex - Embarked - Pclass

categorical data -> numerical

```
In [149]: data = pd.get_dummies(data, columns=['Sex', 'Embarked', 'Pclass'], drop_first=True)
          test = pd.get_dummies(test, columns=['Sex', 'Embarked', 'Pclass'], drop_first=True)

In [150]: data.head()

Out[150]:    Survived   Age  SibSp  Parch     Fare  Sex_male  Embarked_Q  Embarked_S  \
          0         0  22.0      1      0   7.2500         1           0           1
          1         1  38.0      1      0  71.2833         0           0           0
          2         1  26.0      0      0   7.9250         0           0           1
          3         1  35.0      1      0  53.1000         0           0           1
          4         0  35.0      0      0   8.0500         1           0           1

             Pclass_2  Pclass_3
          0         0         1
          1         0         0
          2         0         1
          3         0         0
          4         0         1
```

2.2.3 Modell trainieren

Define Type of Model Wir beginnen mit einem **Decision Tree**.

```
In [133]: X = data.drop('Survived', axis=1)
          y = data['Survived']
```

Daten in Train Test splitten

```
In [134]: from sklearn.model_selection import train_test_split

          test_size = 0.3
          X_train, X_cv, y_train, y_cv = train_test_split(X, y, test_size=test_size, random_stat

In [135]: from sklearn.tree import DecisionTreeClassifier

          titanic_dt_model = DecisionTreeClassifier(random_state=1);
          titanic_dtgini_model = DecisionTreeClassifier(criterion='gini', max_depth=3)
```

Fit: Patterns in Daten finden

```
In [136]: titanic_dt_model.fit(X_train, y_train);
          titanic_dtgini_model.fit(X_train, y_train);
```

Predict

```
In [151]: predict_y_cv = titanic_dt_model.predict(X_cv)
          predict_y_cv_gini = titanic_dtgini_model.predict(X_cv)
          y_cv.head()

Out[151]: 331    0
          700    1
          748    0
          751    1
          481    0
          Name: Survived, dtype: int64
```

Evaluate

```
In [152]: from sklearn.metrics import classification_report
          print(classification_report(y_test, predict_y_cv.round()))
          print(classification_report(y_test, predict_y_cv_gini))

                       precision    recall  f1-score   support

                    0       0.78      0.85      0.81       154
                    1       0.77      0.67      0.71       114

          avg / total       0.77      0.77      0.77       268
```

```
             precision    recall  f1-score   support

          0       0.79      0.90      0.84       154
          1       0.84      0.67      0.74       114

avg / total       0.81      0.80      0.80       268
```

```
In [153]: from sklearn.model_selection import cross_val_score

          scores_dt = cross_val_score(titanic_dt_model, X, y, cv=10, scoring='accuracy')
          scores_dt_gini = cross_val_score(titanic_dtgini_model, X, y, cv=10, scoring='accuracy'

          print (scores_dt.mean())
          print (scores_dt_gini.mean())
```

```
0.7879483032572919
0.8181855067529223
```

Confusion Matrix

```
In [154]: from sklearn.metrics import confusion_matrix

          print(confusion_matrix(y_test, predict_y_cv.round()))
          print(confusion_matrix(y_test, predict_y_cv_gini.round()))
```

```
[[131  23]
 [ 38  76]]
[[139  15]
 [ 38  76]]
```

2.2.4 Kaggle Upload

```
In [159]: test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 9 columns):
Age           418 non-null float64
SibSp         418 non-null int64
Parch         418 non-null int64
Fare          418 non-null float64
Sex_male      418 non-null uint8
Embarked_Q    418 non-null uint8
Embarked_S    418 non-null uint8
```

```
Pclass_2      418 non-null uint8
Pclass_3      418 non-null uint8
dtypes: float64(2), int64(2), uint8(5)
memory usage: 15.2 KB
```

```
In [160]: predicted_test_dt = titanic_dt_model.predict(test)
          predicted_test_dt_gini = titanic_dtgini_model.predict(test)
```

```
In [172]: def saveCSV(ids, predictedData, name):
              sub_dt = pd.DataFrame()
              sub_dt['PassengerId'] = ids
              sub_dt['Survived'] = predictedData
              sub_dt.to_csv(name,index=False)

          test = pd.read_csv(data_url_test) # get original data

          passengerIds = test['PassengerId'];
          # saveCSV(passengerIds, predicted_test_dt,'predicted_dt.csv') -> Scores : 0.69377
          # saveCSV(passengerIds, predicted_test_dt_gini,'predicted_dt_gini.csv') -> Scores: 0.7
```

3 Erweiterte Lösung

3.1 Mehr Input Daten nutzen

3.1.1 Familien

```
In [ ]: def addFamilySize(df):
            df['FamilySize'] = df['SibSp'] + df['Parch'] +1

            df['Alone']=0
            df.loc[(df.FamilySize==1),'Alone'] = 1

        addFamilySize(data)
```

```
In [ ]: plt.subplots(figsize=(10,6))
        sns.barplot(x='FamilySize' , y='Survived' , data = df_train)
        plt.ylabel("Survival Rate")
        plt.title("Survival as function of FamilySize")
        plt.show()
```

```
In [ ]:     df.drop(['SibSp'], axis=1, inplace=True)
            df.drop(['Parch'], axis=1, inplace=True)
            df.drop(['Alone'], axis=1, inplace=True)
```

3.1.2 Titel aus dem Namen extrahieren

```
In [ ]: def addTitle(df):
            df['Title']=0
```

```
            df['Title']=df.Name.str.extract(r'([A-Za-z]+)\.') #lets extract the Salutations
            df['Title'].replace(['Mlle','Mme','Ms','Dr','Major','Lady','Countess','Jonkheer','Co
                                 ['Miss','Miss','Miss','Mr','Mr','Mrs','Mrs','Other','Other','Other',

        addTitle(data)
```

```
In [ ]: grps_title_survrate = df_train.groupby(['Title'])['Survived'].mean().to_frame()
        grps_title_survrate
```

```
In [ ]:     # convert Title to numerical
            df['Title'] = df['Title'].map( {'Other':0, 'Mr': 1, 'Master':2, 'Miss': 3, 'Mrs': 4
            # fill na with maximum frequency mode
            df['Title'] = df['Title'].fillna(df['Title'].mode().iloc[0])
            df['Title'] = df['Title'].astype(int)
```

3.1.3 Namenslänge

```
In [ ]:     df['NameLen'] = df.Name.apply(lambda x : len(x))
            df['NameLenBin']=np.nan
            for i in range(20,0,-1):
                df.loc[ df['NameLen'] <= i*5, 'NameLenBin'] = i
```

```
In [ ]: grps_namelenbin_survrate = df_train.groupby(['NameLenBin'])['Survived'].mean().to_frame(
        grps_namelenbin_survrate
```

```
In [ ]:     df.drop(['NameLen'], axis=1, inplace=True)
```

3.1.4 Alter

In Altersgruppen aufteilen

bessere Idee zum Füllen der fehlenden Daten

```
In [ ]:  # Age: use Title to fill missing values
            df.loc[(df.Age.isnull())&(df.Title=='Mr'),'Age']= df.Age[df.Title=="Mr"].mean()
            df.loc[(df.Age.isnull())&(df.Title=='Mrs'),'Age']= df.Age[df.Title=="Mrs"].mean()
            df.loc[(df.Age.isnull())&(df.Title=='Master'),'Age']= df.Age[df.Title=="Master"].mea
            df.loc[(df.Age.isnull())&(df.Title=='Miss'),'Age']= df.Age[df.Title=="Miss"].mean()
            df.loc[(df.Age.isnull())&(df.Title=='Other'),'Age']= df.Age[df.Title=="Other"].mean(
            df = df.drop('Name', axis=1)
```

3.2 Feature Scaling benutzen

```
In [ ]: from sklearn.preprocessing import StandardScaler
        scaler = StandardScaler()

        # for df_train_ml
        scaler.fit(df_train_ml.drop(['Survived'],axis=1))
        scaled_features = scaler.transform(df_train_ml.drop(['Survived'],axis=1))
```

```
        df_train_ml_sc = pd.DataFrame(scaled_features) # columns=df_train_ml.columns[1::])

        # for df_test_ml
        df_test_ml.fillna(df_test_ml.mean(), inplace=True)
        #scaler.fit(df_test_ml)
        scaled_features = scaler.transform(df_test_ml)
        df_test_ml_sc = pd.DataFrame(scaled_features) # , columns=df_test_ml.columns)
```

```
In [ ]: df_test_ml_sc.head()
```

```
In [ ]: df_train_ml.head()
```

```
In [1]: X_sc = df_train_ml_sc
        y_sc = df_train_ml['Survived']
        X_test_sc = df_test_ml_sc
```

```
        ---------------------------------------------------------------------------

        NameError                                 Traceback (most recent call last)

        <ipython-input-1-66b790bc8021> in <module>
    ----> 1 X_sc = df_train_ml_sc
          2 y_sc = df_train_ml['Survived']
          3 X_test_sc = df_test_ml_sc


        NameError: name 'df_train_ml_sc' is not defined
```

3.3 Verschiedene Algorithmen testen und bewerten

3.4 Modell optimieren

3.5 Test mit einem Neuralen Netzwerk mit Tensorflow

4 Auswertung Kaggle Challenge Nr. 1