W_M_LR

April 22, 2019

1 TITANIC

1.1 Wstęp

W tym projekcie postaram się przeanalizować zbiór danych reprezentujący pasażerów uczestniczących w tragicznym rejsie Titanica. Statku nazwanego niezatapialnym, który zatanął podczas swojego dziewiczego rejsu po zderzeniu z górą lodową.

Postaram się sprawdzić jakie czynniki, mogły zwiększyć bądź zmniejszyć szansę przeżycia pasażerów. Analizowany zbiór zawiera następujące kolumny:

- Id Pasażera
- Czy pasażer przeżył
- Klasę jaką podróżował
- Imię i Nazwisko
- Płeć
- Wiek
- Relacje rodzinne(Mąż, Żona, Syn, Córka, Mąż, Żona, Brat, Siostra)
- Bilet
- Wysokość opłaty
- Numer kabiny
- Kod portu

1.2 Plan analizy

W swojej pracy przeanalizuję wszystkie kolumny i sprawdzę jaki dana kolumna miałą wpływ na przeżywalność pasażerów. W międzyczasie w przypadku brakujących wartości zdecyduję czy dana kolumna będzie brała udział w dalszej analizie czy powinna zostać usunięta.

Następnie wybrane w czasie analizy kolumny uwzględnimy w modelu regresji logistycznej. Zbiór titanic_train będzie służył do uczenia naszego modelu, aby następnie przypisać wartości dla atrybutów zbioru titanic_test. Nastepnie podzielimy zbiór titanic_train na zbiór treningowy i testowy, a także sprawdzimy który model zwróci najlepsze wyniki ucząc się na zbiorze treningowym.

In [ ]:

```
In [1707]: import pandas as pd
           import numpy as np
           import matplotlib.pyplot as plt
```

```
import seaborn as sns
%matplotlib inline
sns.set_style('whitegrid')
```

```
In [1708]: tit_test = pd.read_csv('titanic_test.csv')
```

```
In [1709]: tit_train = pd.read_csv('titanic_train.csv')
```

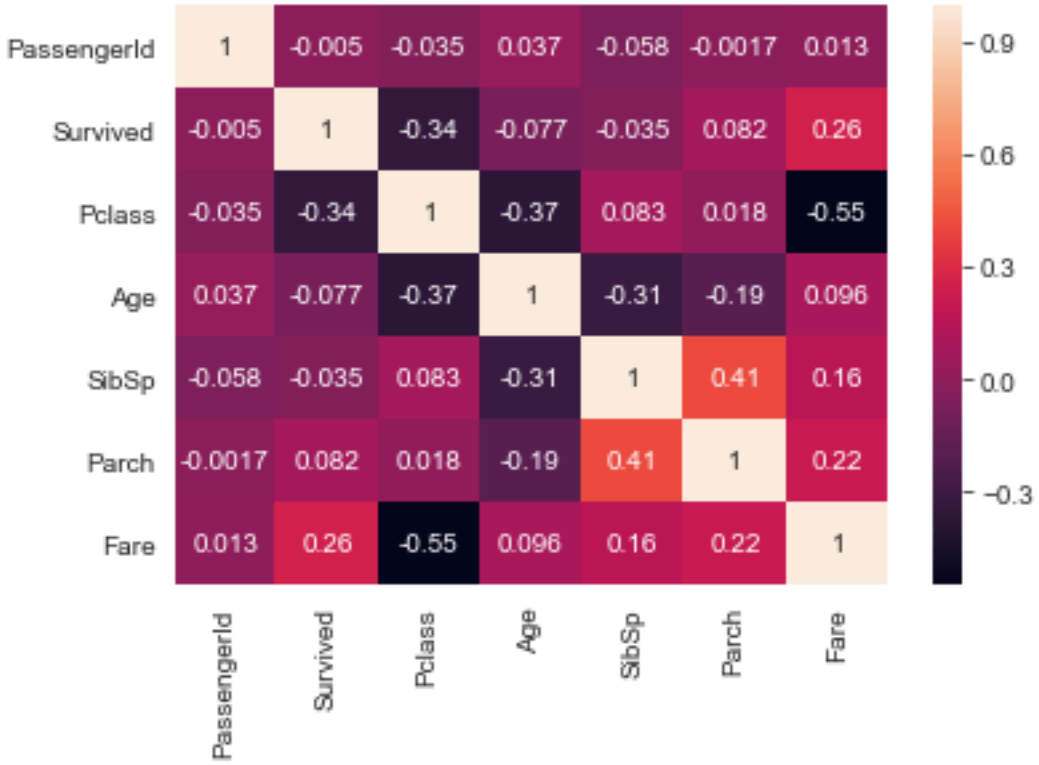
```
In [1710]: tit_train.head()
```

Out[1710]:

	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
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

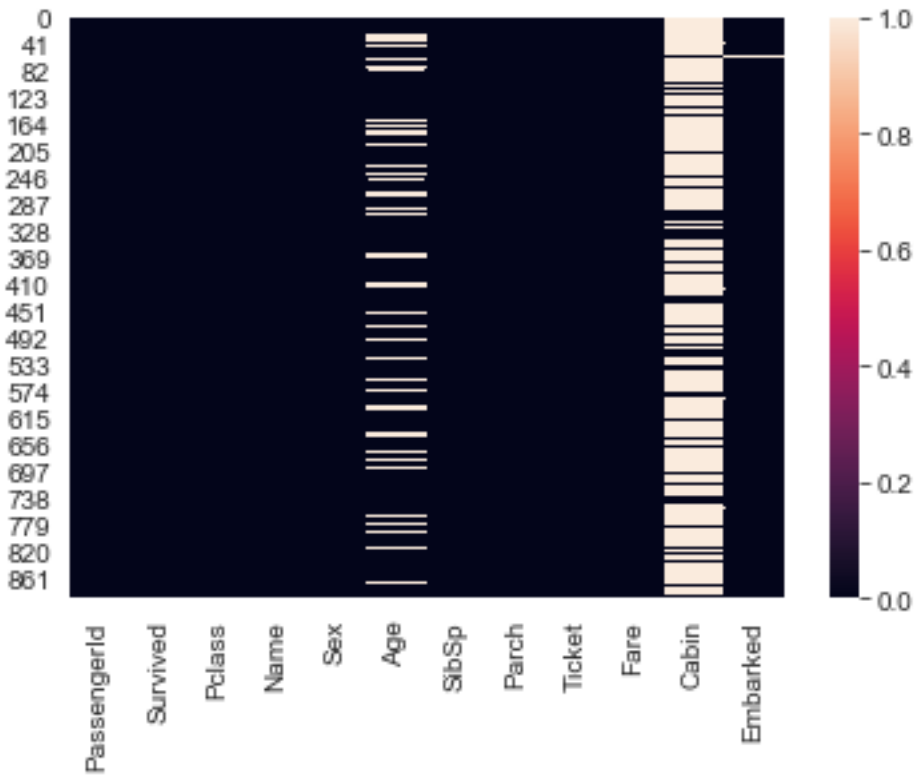
```
In [1711]: sns.heatmap(tit_train.corr(), annot=True)
```

```
Out[1711]: <matplotlib.axes._subplots.AxesSubplot at 0x39e7f01860>
```



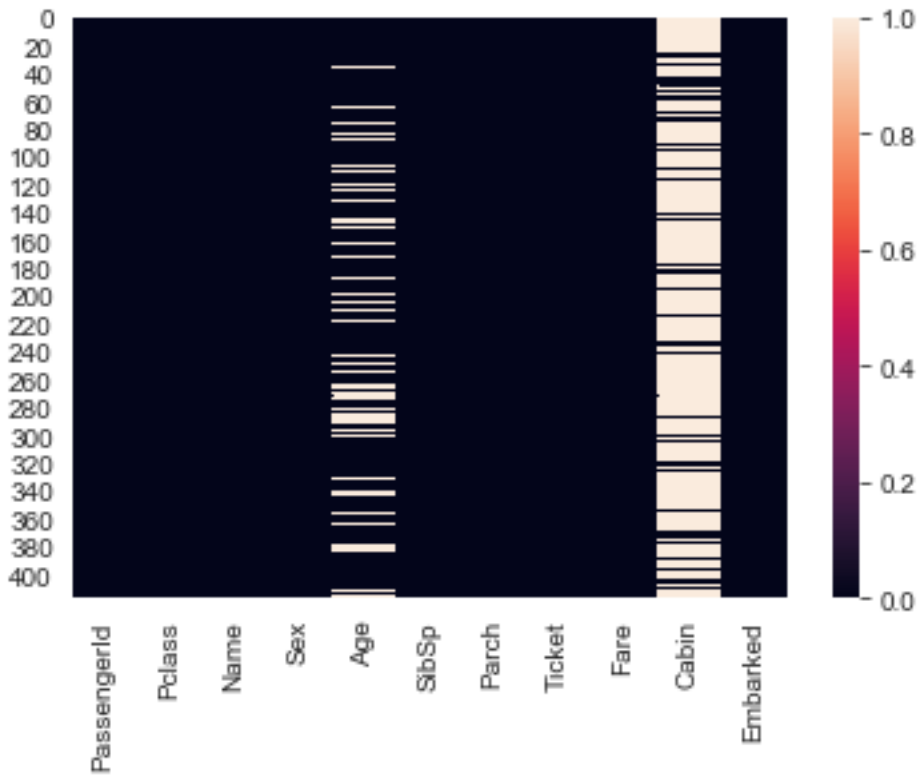

```
In [1712]: sns.heatmap(tit_train.isnull())
```

```
Out[1712]: <matplotlib.axes._subplots.AxesSubplot at 0x39e7e44b00>
```




```
In [1713]: sns.heatmap(tit_test.isnull())
```

```
Out[1713]: <matplotlib.axes._subplots.AxesSubplot at 0x39e90cb0b8>
```



Jak widzimy wystepuje wiele wartości Null/NaN, w jakis sposób trzeba sobie z tym poradzić. Mozliwe rozwiązanie to ustawienie średnego wieku np po płci natomiast kolumnę Cabin można usunąć, albo po prostu ustawić inną wartość zamiast Null/NaN

Teraz przeanalizujemy kolejne kolumny pod względem przydatności w naszym modelu

```
In [1714]: tit_train.describe()
```

```
Out[1714]:        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
```

Jak widzimy przeżyło tylko ok. 38% pasażerów

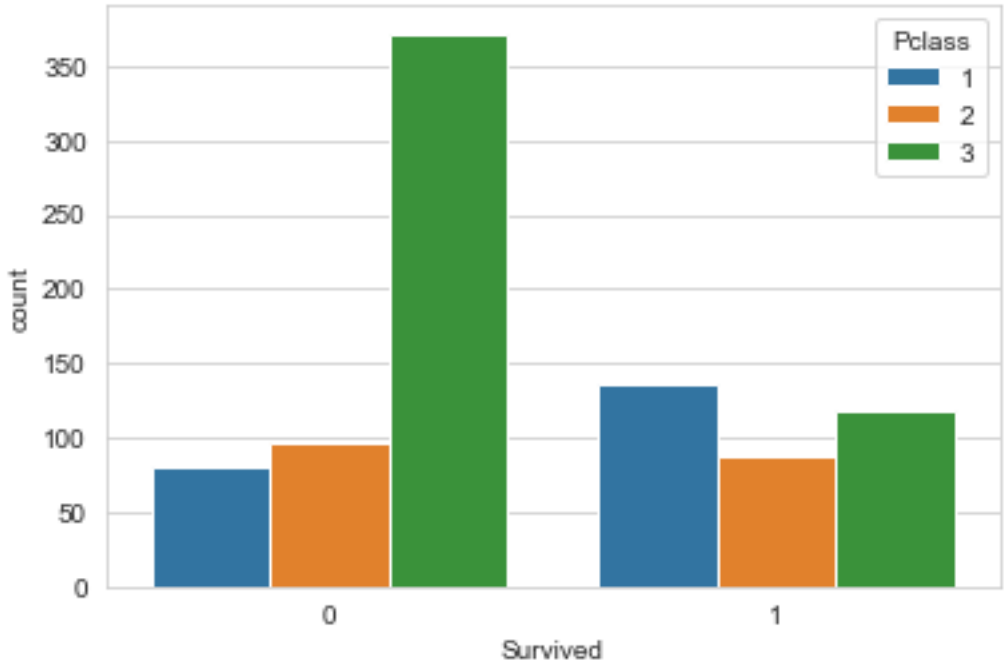
```
In [1715]: tit_train.info()
```

```
<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
```

Teraz przeanalizujemy kolejne kolumny pod kątem przeżywalności pasażerów i zdecydujemy które z nich uwzględnić w naszym modelu. Na początku przeanalizujemy klasy jakimi podróżowali pasażerowie

```
In [1716]: sns.countplot(data=tit_train, x='Survived', hue='Pclass')
```

```
Out[1716]: <matplotlib.axes._subplots.AxesSubplot at 0x39e91ac3c8>
```

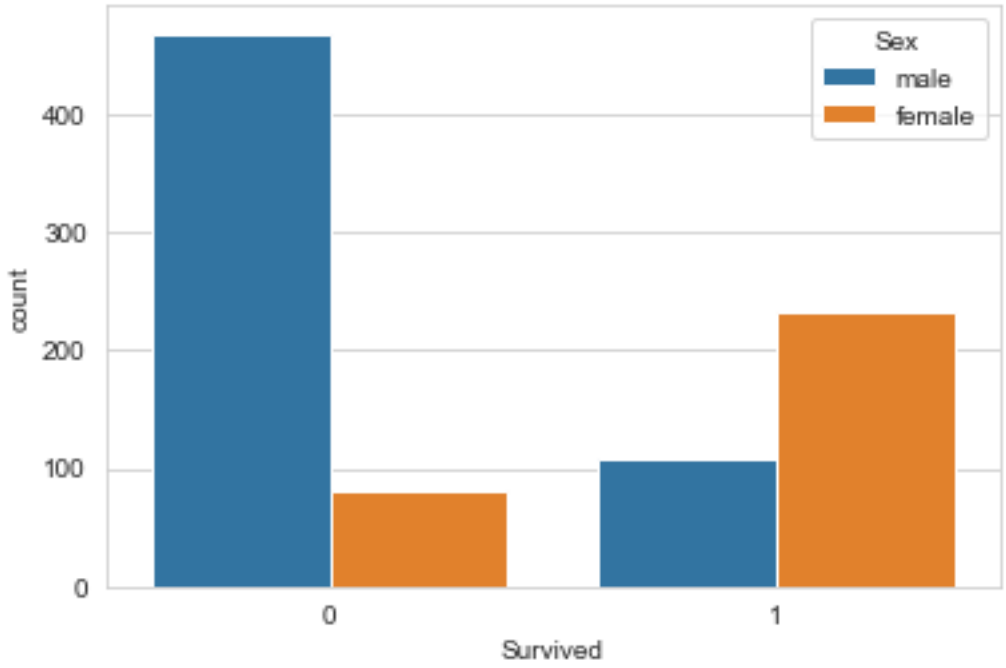



```
In [1717]: tit_train.groupby('Pclass', as_index=False)['Survived'].mean()

Out[1717]:    Pclass  Survived
           0       1  0.629630
           1       2  0.472826
           2       3  0.242363
```

Pasażerowie z pierwszej klasy mieli większe szanse na przeżycie, około 63% z nich przeżyło wypadek, ponadto im lepsza klasa, tym więcej pasażerów przeżyło wypadek - Uwzględniamy Pclass w naszym modelu

Teraz przejdziemy do analizy płci pasażerów

```
In [1718]: sns.countplot(data=tit_train, x='Survived', hue='Sex')

Out[1718]: <matplotlib.axes._subplots.AxesSubplot at 0x39e91fb550>
```




```
In [1719]: tit_train.groupby('Sex', as_index=False)['Survived'].mean().sort_values(by='Survived'

Out[1719]:       Sex  Survived
           0  female  0.742038
           1    male  0.188908
```

Jak widać na wykresie oraz w przedstawionej tabelce, około 74% kobiet przezyło wypadek, natomiast ocalonych mężczyzn było tylko niecałe 19%. Zapewne wynikało to z zasady ratowania najpeirw kobiet i dzieci. O słuszności tej hipotezy przekonamy się także przy analizie wieku pasażerów

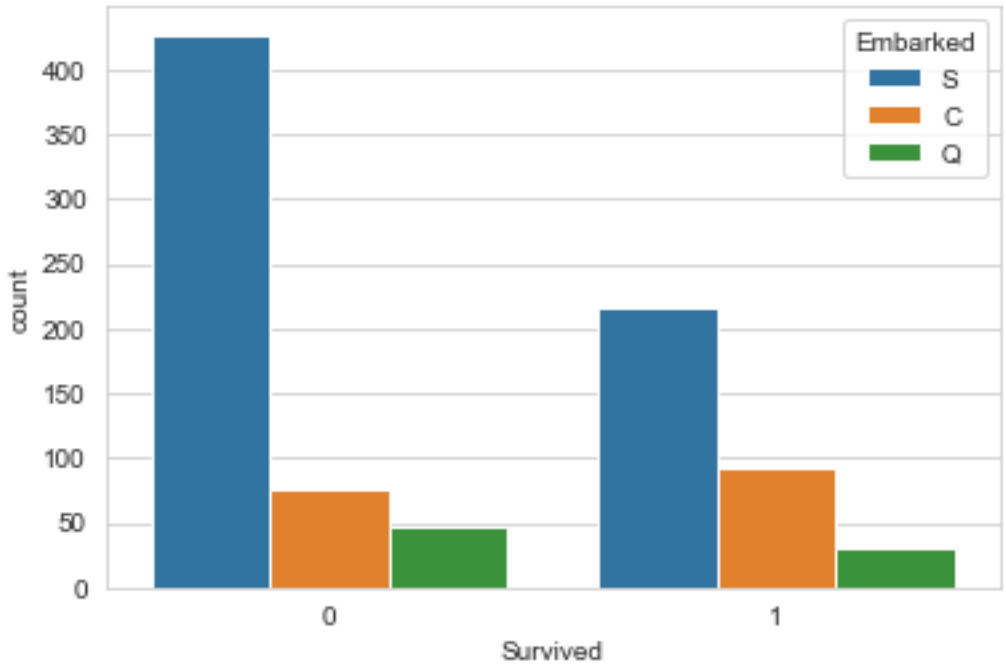
Teraz zamienimy wartości tekstowe na numeryczne abysmy mogli uwzględnić je np w modelu regresji logistycznej

```
In [1720]: sex_train = pd.get_dummies(tit_train['Sex'],  drop_first=True)
           tit_train = pd.concat([tit_train, sex_train], axis=1)
           tit_train.drop('Sex', axis=1, inplace=True)

           sex_test = pd.get_dummies(tit_test['Sex'],  drop_first=True)
           tit_test = pd.concat([tit_test, sex_test], axis=1)
           tit_test.drop('Sex', axis=1, inplace=True)


           # tit_train['Sex'] = tit_train['Sex'].map( {'female': 1, 'male': 0} ).astype(int)
           # tit_test['Sex'] = tit_test['Sex'].map( {'female': 1, 'male': 0} ).astype(int)
```

Teraz zamienimy wartości w 'Embarked' na wartości liczbowe

```
In [1721]: sns.countplot(data=tit_train, x='Survived', hue='Embarked')
```

```
Out[1721]: <matplotlib.axes._subplots.AxesSubplot at 0x39e91a3198>
```




```
In [1722]: tit_train.groupby('Embarked', as_index=False)['Survived'].mean().sort_values(by='Surv
```

```
Out[1722]:   Embarked  Survived
           0        C  0.553571
           1        Q  0.389610
           2        S  0.336957
```

```
In [1723]: tit_train.Embarked[tit_train['Embarked'] == 'C'] = 1
           tit_train.Embarked[tit_train['Embarked'] == 'S'] = 2
           tit_train.Embarked[tit_train['Embarked'] == 'Q'] = 3
           tit_test.Embarked[tit_test['Embarked'] == 'C'] = 1
           tit_test.Embarked[tit_test['Embarked'] == 'S'] = 2
           tit_test.Embarked[tit_test['Embarked'] == 'Q'] = 3
```

```
C:\Users\acer\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#
  """Entry point for launching an IPython kernel.
C:\Users\acer\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: SettingWithCopyWarning:
```

```
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#

C:\Users\acer\Anaconda3\lib\site-packages\ipykernel_launcher.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#
  This is separate from the ipykernel package so we can avoid doing imports until
C:\Users\acer\Anaconda3\lib\site-packages\ipykernel_launcher.py:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#
  after removing the cwd from sys.path.
C:\Users\acer\Anaconda3\lib\site-packages\ipykernel_launcher.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#
  """
C:\Users\acer\Anaconda3\lib\site-packages\ipykernel_launcher.py:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#
```

```
In [1724]: tit_train.head()
```

Out[1724]:

	PassengerId	Survived	Pclass	Name	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	male
0	1	0	3	Braund, Mr. Owen Harris	22.0	1	0	A/5 21171	7.2500	NaN	2	1
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	38.0	1	0	PC 17599	71.2833	C85	1	0
2	3	1	3	Heikkinen, Miss. Laina	26.0	0	0	STON/O2. 3101282	7.9250	NaN	2	0
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	35.0	1	0	113803	53.1000	C123	2	0
4	5	0	3	Allen, Mr. William Henry	35.0	0	0	373450	8.0500	NaN	2	1

W dalszej części analizy wypełnimy wartości Null/NaN w kolumnie 'Cabin'. Zastepimy je wartością 'NO'

```
In [1725]: tit_train['Cabin'].fillna('NO', inplace=True)
           tit_test['Cabin'].fillna('NO', inplace=True)

In [1726]: cabin_letter_train = tit_train['Cabin'].apply(lambda x: x[0])
           cabin_letter_train.head()

Out[1726]: 0    N
           1    C
           2    N
           3    C
           4    N
           Name: Cabin, dtype: object

In [1727]: cabin_letter_test = tit_test['Cabin'].apply(lambda x: x[0])
           cabin_letter_test.head()

Out[1727]: 0    N
           1    N
           2    N
           3    N
           4    N
           Name: Cabin, dtype: object

In [1728]: tit_train.drop('Cabin', axis=1, inplace=True)
           tit_test.drop('Cabin', axis=1, inplace=True)

In [1729]: tit_train = pd.concat([tit_train, cabin_letter_train], axis=1)
           tit_test = pd.concat([tit_test, cabin_letter_test], axis=1)

In [1730]: tit_train.head()

Out[1730]:    PassengerId  Survived  Pclass  \
           0            1         0       3
           1            2         1       1
           2            3         1       3
           3            4         1       1
           4            5         0       3

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

                     Ticket     Fare Embarked  male Cabin
```

```
0         A/5 21171   7.2500        2     1     N
1          PC 17599  71.2833        1     0     C
2  STON/O2. 3101282   7.9250        2     0     N
3            113803  53.1000        2     0     C
4            373450   8.0500        2     1     N
```

```
In [1731]: print(tit_train.groupby('Cabin', as_index=False)['Survived'].mean().sort_values(by='S
```

```
  Cabin  Survived
3     D  0.757576
4     E  0.750000
1     B  0.744681
5     F  0.615385
2     C  0.593220
6     G  0.500000
0     A  0.466667
7     N  0.299854
8     T  0.000000
```

```
In [1732]: print(tit_train.groupby('Cabin', as_index=False)['Survived'].count().sort_values(by='
           print(tit_test.groupby('Cabin', as_index=False)['Fare'].count().sort_values(by='Fare'
```

```
  Cabin  Survived
7     N       687
2     C        59
1     B        47
3     D        33
4     E        32
0     A        15
5     F        13
6     G         4
8     T         1
  Cabin  Fare
7     N   326
2     C    35
1     B    18
3     D    13
4     E     9
5     F     8
0     A     7
6     G     1
```

Jak widzimy Pasażerowie z kabin D, E, B, F, C mieli większą szansę na przeżycie niż pozostali. Chociaż statystyka jest niepełna uwzględnimy tę kolumnę w naszym modelu. T zamienimy na N gdyż nie wystepuje ona w zbiorze testowym a pasażer i tak zginął

```
In [1733]: tit_train.Cabin[tit_train['Cabin']=='T']='N'
```

```
C:\Users\acer\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#
  """Entry point for launching an IPython kernel.
```

```
In [1734]: def cabin_to_number(cabin_letter):
               if cabin_letter == 'N':
                   return 0
               elif cabin_letter =='C':
                   return 1
               elif cabin_letter == 'B':
                   return 2
               elif cabin_letter == 'D':
                   return 3
               elif cabin_letter =='E':
                   return 4
               elif cabin_letter == 'A':
                   return 5
               elif cabin_letter == 'F':
                   return 6
               else:
                   return 7
```

```
In [1735]: tit_train['Cabin'] = tit_train['Cabin'].apply(cabin_to_number)
           tit_test['Cabin'] = tit_test['Cabin'].apply(cabin_to_number)
```

```
In [1736]: tit_train.head(10)
```

Out[1736]:

	PassengerId	Survived	Pclass	Name	Age	SibSp	Parch
0	1	0	3	Braund, Mr. Owen Harris	22.0	1	0
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	38.0	1	0
2	3	1	3	Heikkinen, Miss. Laina	26.0	0	0
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	35.0	1	0
4	5	0	3	Allen, Mr. William Henry	35.0	0	0
5	6	0	3	Moran, Mr. James	NaN	0	0
6	7	0	1				
7	8	0	3				
8	9	1	3				
9	10	1	2				

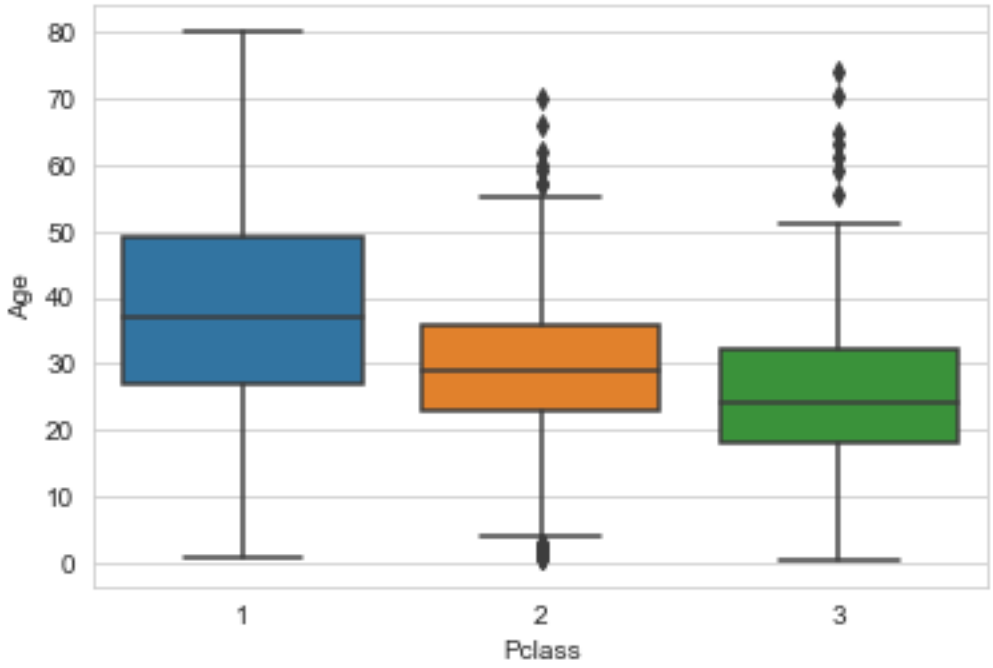
```
6                            McCarthy, Mr. Timothy J  54.0      0      0
7                     Palsson, Master. Gosta Leonard   2.0      3      1
8  Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)  27.0      0      2
9                Nasser, Mrs. Nicholas (Adele Achem)  14.0      1      0

             Ticket     Fare Embarked  male  Cabin
0         A/5 21171   7.2500        2     1      0
1          PC 17599  71.2833        1     0      1
2  STON/O2. 3101282   7.9250        2     0      0
3            113803  53.1000        2     0      1
4            373450   8.0500        2     1      0
5            330877   8.4583        3     1      0
6             17463  51.8625        2     1      4
7            349909  21.0750        2     1      0
8            347742  11.1333        2     0      0
9            237736  30.0708        1     0      0
```

Analizę imion odpuścimy, gdyż uważam, że imię nie ma wpływu na to czy ktoś przeżył katastrofę Nie będziemy dodawać oddzielnej kolumny na prefixy gdyż jest to ściśle powiązane z płcią, która już będzie częścią naszego modelu

```
In [1737]: tit_train.drop('Name', axis=1, inplace=True)
           tit_test.drop('Name', axis=1, inplace=True)
```

Teraz uzupełnimy brakujące wiersze w kolumnie 'Age', wartościami średnimi dla danej klasy

```
In [1738]: tit_train.groupby('Pclass', as_index=False)['Age'].mean()

Out[1738]:    Pclass        Age
           0       1  38.233441
           1       2  29.877630
           2       3  25.140620

In [1739]: sns.boxplot(data=tit_train, x="Pclass", y='Age')

Out[1739]: <matplotlib.axes._subplots.AxesSubplot at 0x39e7daaa90>
```



Dla pierwszej klasy wstawimy wartość 38, dla drugiej 30, a dla trzeciej 25

```
In [1740]: def fill_missing_age(cols):
               age = cols[0]
               pclass = cols[1]

               if pd.isnull(age):
                   if pclass == 1:
                       return 38
                   elif pclass == 2:
                       return 30
                   else:
                       return 25
               else:
                   return age
```

```
In [1741]: tit_train['Age'] = tit_train[['Age', 'Pclass']].apply(fill_missing_age, axis=1)
           tit_test['Age'] = tit_test[['Age', 'Pclass']].apply(fill_missing_age, axis=1)
```

```
In [1742]: tit_train.head(10)
```

```
Out[1742]:    PassengerId  Survived  Pclass   Age  SibSp  Parch            Ticket  \
           0            1         0       3  22.0      1      0         A/5 21171
           1            2         1       1  38.0      1      0          PC 17599
           2            3         1       3  26.0      0      0  STON/O2. 3101282
           3            4         1       1  35.0      1      0            113803
```

```
4            5         0       3  35.0      0      0            373450
5            6         0       3  25.0      0      0            330877
6            7         0       1  54.0      0      0             17463
7            8         0       3   2.0      3      1            349909
8            9         1       3  27.0      0      2            347742
9           10         1       2  14.0      1      0            237736
```

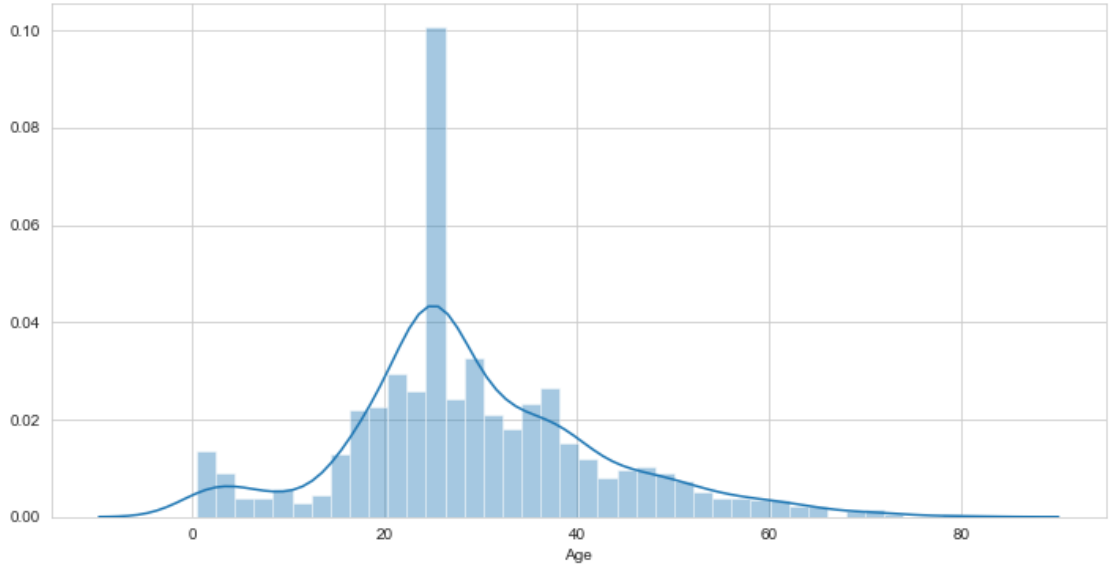
```
      Fare Embarked  male  Cabin
0   7.2500        2     1      0
1  71.2833        1     0      1
2   7.9250        2     0      0
3  53.1000        2     0      1
4   8.0500        2     1      0
5   8.4583        3     1      0
6  51.8625        2     1      4
7  21.0750        2     1      0
8  11.1333        2     0      0
9  30.0708        1     0      0
```

In [1743]:
```
tit_train.dropna(inplace=True)
```

Teraz przeanalizujemy wiek ocalonych, sprawdzimy ile osób z róznych grup wiekowych przeżyło wypadek

In [1744]:
```
plt.figure(figsize=(12,6))
sns.distplot(tit_train['Age'], bins=40)
```

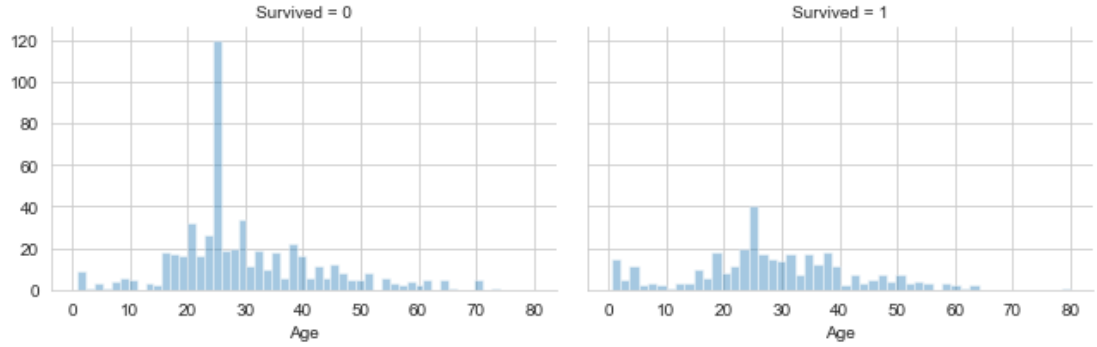
Out[1744]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x39e7e22668>
```



```
In [1745]: grid = sns.FacetGrid(data = tit_train, col='Survived', size=3, aspect=1.5)
           grid.map(sns.distplot, 'Age', kde=False, bins=50)
           grid.add_legend()
```

```
C:\Users\acer\Anaconda3\lib\site-packages\seaborn\axisgrid.py:230: UserWarning: The `size` param
  warnings.warn(msg, UserWarning)
```

```
Out[1745]: <seaborn.axisgrid.FacetGrid at 0x39e475c6d8>
```



Bazując na powyższych wynikach, podzielimy wiek pasażerów na nastepujące przedziały: 1. małe dzieci - (0,6) 2. dzieci - (7,12) 3. młodzież - (13,20) 4. młodzi dorośli - (21, 40) 5. starsi dorośli - (41,60) 6. emeryci - (60 - 80)

```
In [1746]: bins = [0, 6, 12, 20, 40, 60, 80]
           tit_train['Age_cut'] = pd.cut(tit_train['Age'], bins)
           tit_test['Age_cut'] = pd.cut(tit_test['Age'], bins)

           print(tit_train['Age_cut'].value_counts())
           print(tit_test['Age_cut'].value_counts())
           tit_train[['Age_cut', 'Survived']].groupby(['Age_cut'], as_index=False).mean().sort_v
```

```
(20, 40]    561
(40, 60]    128
(12, 20]    110
(0, 6]       47
(6, 12]      22
(60, 80]     21
Name: Age_cut, dtype: int64
(20, 40]    272
(40, 60]     66
(12, 20]     44
(0, 6]       15
(60, 80]     11
(6, 12]      10
Name: Age_cut, dtype: int64
```

```
Out[1746]:     Age_cut  Survived
           0    (0, 6]  0.702128
           4  (40, 60]  0.390625
           2  (12, 20]  0.381818
           3  (20, 40]  0.363636
           1   (6, 12]  0.318182
           5  (60, 80]  0.190476
```

Jak widzimy wyniki dla Mlodzieży, Młodych i starszych dorosłych są bardzo zblizone więc zamkniemy je w jednym przedziale

```
In [1747]: bins = [0, 6, 12, 60, 80]
           tit_train['Age_cut'] = pd.cut(tit_train['Age'], bins)
           tit_test['Age_cut'] = pd.cut(tit_test['Age'], bins)

           print(tit_train['Age_cut'].value_counts())
           print(tit_test['Age_cut'].value_counts())
           tit_train[['Age_cut', 'Survived']].groupby(['Age_cut'], as_index=False).mean().sort_v
```

```
(12, 60]    799
(0, 6]       47
(6, 12]      22
(60, 80]     21
Name: Age_cut, dtype: int64
(12, 60]    382
(0, 6]       15
(60, 80]     11
(6, 12]      10
Name: Age_cut, dtype: int64
```

```
Out[1747]:     Age_cut  Survived
           0    (0, 6]  0.702128
           2  (12, 60]  0.370463
           1   (6, 12]  0.318182
           3  (60, 80]  0.190476
```

```
In [1748]: def swap_intervals(value):
               if str(value) == '(0, 6]':
                   return 0
               elif str(value) == '(6, 12]':
                   return 1
               elif str(value) == '(12, 60]':
                   return 2
               else:
                   return 3
```

```
In [1749]: tit_train['Age_cut'] = tit_train['Age_cut'].apply(swap_intervals)
           tit_test['Age_cut'] = tit_test['Age_cut'].apply(swap_intervals)
```

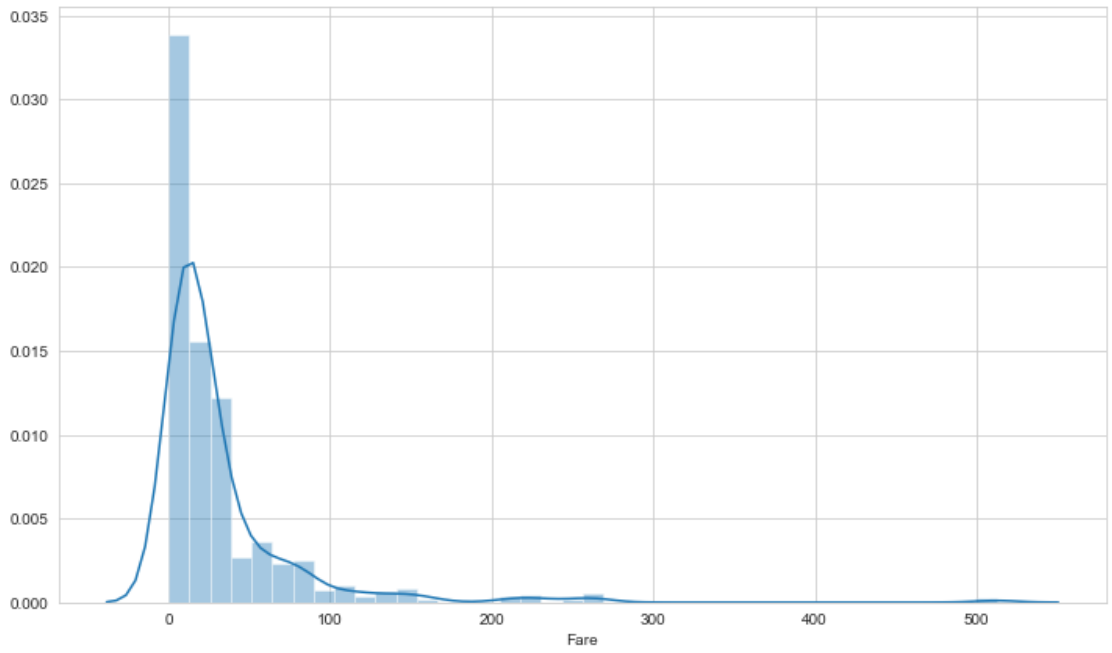
```
In [1750]: tit_train.head()
```

```
Out[1750]:    PassengerId  Survived  Pclass   Age  SibSp  Parch            Ticket  \
           0            1         0       3  22.0      1      0         A/5 21171
           1            2         1       1  38.0      1      0          PC 17599
           2            3         1       3  26.0      0      0  STON/O2. 3101282
           3            4         1       1  35.0      1      0            113803
           4            5         0       3  35.0      0      0            373450

                Fare Embarked  male  Cabin Age_cut
           0   7.2500        2     1      0       2
           1  71.2833        1     0      1       2
           2   7.9250        2     0      0       2
           3  53.1000        2     0      1       2
           4   8.0500        2     1      0       2
```

Teraz podobnie postąpimy z ceną biletu. Przypuszczamy, że osoby z droższymi biletami miały lepsze miejsca i dzięki temu np. łatwiejszą drogę ewakuacji.

```
In [1751]: plt.figure(figsize=(12,7))
           sns.distplot(tit_train['Fare'], bins=40)
```

```
Out[1751]: <matplotlib.axes._subplots.AxesSubplot at 0x39e94e0e10>
```



Spróbujemy podzielić ceny biletów na przedziały:

1. Bardzo Tani - (0,50)
2. Tani - (50,100)
3. Średni - (100,200)
4. Drogi - (200-300)
5. Ekskluzywny - (300,max)

```
In [1752]: tit_train['Fare'].max()

Out[1752]: 512.3292

In [1753]: bins = [-0.01, 50, 100, 200, 300, 600]

           tit_train['Fare_cut'] = pd.cut(tit_train['Fare'], bins)
           tit_test['Fare_cut'] = pd.cut(tit_test['Fare'], bins)

           print(tit_train['Fare_cut'].value_counts())
           print(tit_test['Fare_cut'].value_counts())
           tit_train[['Fare_cut', 'Survived']].groupby(['Fare_cut'], as_index=False).mean().sort
```

```
(-0.01, 50.0]     731
(50.0, 100.0]     105
(100.0, 200.0]     33
(200.0, 300.0]     17
(300.0, 600.0]      3
Name: Fare_cut, dtype: int64
(-0.01, 50.0]     337
(50.0, 100.0]      49
(200.0, 300.0]     17
(100.0, 200.0]     13
(300.0, 600.0]      1
Name: Fare_cut, dtype: int64


Out[1753]:          Fare_cut  Survived
           4  (300.0, 600.0]  1.000000
           2  (100.0, 200.0]  0.757576
           1   (50.0, 100.0]  0.647619
           3  (200.0, 300.0]  0.647059
           0   (-0.01, 50.0]  0.318741
```

Jak widzimy pasażerowie posiadający droższe bilety rzeczywiście mieli wieksze szanse na przezycie. Jakiekolwiek brakujące wartości uzupełniamy wartościami średnimi dla danej klasy.

Teraz dla każdego przedziału przypiszemy odpowiadający numer:

```
In [1754]: def fare_cut_to_number(value):
               if str(value) == '(-0.01, 50.0]':
                   return 0
```

```
        elif str(value) == '(50.0, 100.0]' or str(value) == '(200.0, 300.0]':
            return 1
        elif str(value) == '(100.0, 200.0]':
            return 2
        elif str(value) == '(300.0, 600.0]':
            return 3
```

In [1755]:
```
tit_train['Fare_cut'] = tit_train['Fare_cut'].apply(fare_cut_to_number).astype('int64
tit_test['Fare_cut'] = tit_test['Fare_cut'].apply(fare_cut_to_number).astype('int64')

tit_train.drop('Fare', axis=1, inplace=True)
tit_test.drop('Fare', axis=1, inplace=True)
```

In [1756]:
```
tit_train.head()
```

Out[1756]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Ticket	\
0	1	0	3	22.0	1	0	A/5 21171	
1	2	1	1	38.0	1	0	PC 17599	
2	3	1	3	26.0	0	0	STON/O2. 3101282	
3	4	1	1	35.0	1	0	113803	
4	5	0	3	35.0	0	0	373450	

	Embarked	male	Cabin	Age_cut	Fare_cut
0	2	1	0	2	0
1	1	0	1	2	1
2	2	0	0	2	0
3	2	0	1	2	1
4	2	1	0	2	0

W tej sekcji przejdziemy do analizy samych biletów. Nazwa biletu mogła determinować miejsce kabiny a więc, także drogę ewakuacji pasażera

In [1757]:
```
tickets_len_train = tit_train['Ticket'].apply(lambda x: len(x))
tickets_len_test = tit_test['Ticket'].apply(lambda x: len(x))
```

In [1758]:
```
tit_train['Ticket_len'] = tickets_len_train
tit_test['Ticket_len'] = tickets_len_test
```

In [1759]:
```
print(tit_train['Ticket_len'].value_counts())
print(tit_test['Ticket_len'].value_counts())
tit_train.groupby('Ticket_len', as_index=False)['Survived'].mean().sort_values(by='Su
```

```
6     417
5     131
4     101
8      76
10     41
7      27
9      26
```

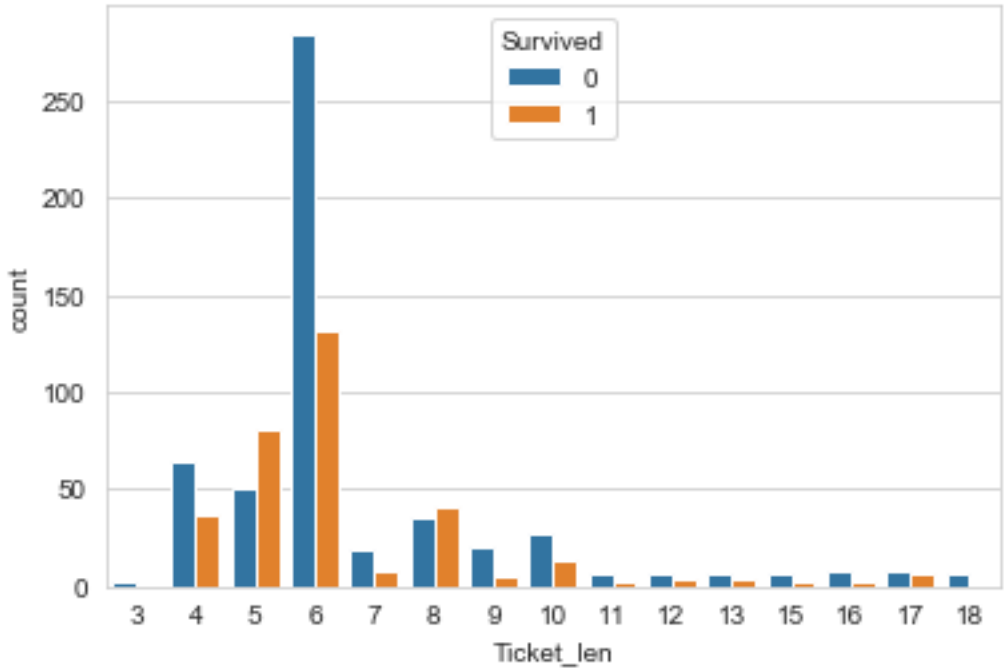
```
17     14
16     11
13     10
12     10
15      9
11      8
18      6
3       2
Name: Ticket_len, dtype: int64
6     183
5      62
4      47
8      42
10     22
9      11
11     10
13      9
7       9
18      8
12      6
15      3
17      2
16      2
3       2
Name: Ticket_len, dtype: int64
```

Out[1759]:

	Ticket_len	Survived
2	5	0.618321
5	8	0.539474
13	17	0.428571
9	12	0.400000
10	13	0.400000
1	4	0.366337
7	10	0.341463
11	15	0.333333
3	6	0.316547
4	7	0.296296
12	16	0.272727
8	11	0.250000
6	9	0.192308
0	3	0.000000
14	18	0.000000

In [1760]:
```
sns.countplot(data=tit_train, x='Ticket_len', hue='Survived')
```

Out[1760]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x39e98caef0>
```



Teraz sprawdzimy jaki wpływ miał pierwszy znak w nazwie biletu

```
In [1761]: ticket_start_train = tit_train['Ticket'].apply(lambda x: x[0])
           ticket_start_test = tit_test['Ticket'].apply(lambda x: x[0])

In [1762]: tit_train['Ticket_start'] = ticket_start_train
           tit_test['Ticket_start'] = ticket_start_test

In [1763]: print(tit_train['Ticket_start'].value_counts())
           print(tit_test['Ticket_start'].value_counts())
           tit_train.groupby('Ticket_start', as_index=False)['Survived'].mean().sort_values(by='
```

```
3    301
2    183
1    144
S     65
P     65
C     47
A     29
W     13
4     10
7      9
F      7
6      6
L      4
5      3
```

```
8      2
9      1
Name: Ticket_start, dtype: int64
3    128
2     95
1     64
S     33
P     33
C     30
A     13
W      6
F      6
7      4
6      3
4      1
L      1
9      1
Name: Ticket_start, dtype: int64
```

Out[1763]:

	Ticket_start	Survived
8	9	1.000000
13	P	0.646154
0	1	0.625000
11	F	0.571429
1	2	0.464481
10	C	0.340426
14	S	0.323077
12	L	0.250000
2	3	0.239203
3	4	0.200000
5	6	0.166667
15	W	0.153846
6	7	0.111111
9	A	0.068966
4	5	0.000000
7	8	0.000000

Jak widzimy wartości jest bardzo dużo, więc cały zbiór postaramy się trochę uszczuplić. Przede wszystkim prefixy o niskiej częstotliwości wystepowania podzielimy na takie które były związanie z dużą przeżywalnością i na takie które były związane z małą przezywalnością posiadających bilet z danym prefixem pasażerów

```
In [1764]: tit_train['Ticket_start'] = tit_train['Ticket_start'].replace(['W', '4', '7', '6', 'L
           tit_train['Ticket_start'] = tit_train['Ticket_start'].replace(['F', '9'], 'Rare_High_
           tit_test['Ticket_start'] = tit_test['Ticket_start'].replace(['W', '4', '7', '6', 'L',
           tit_test['Ticket_start'] = tit_test['Ticket_start'].replace(['F', '9'], 'Rare_High_Su
```

```
tit_train.tail()
tit_train.groupby('Ticket_start', as_index=False)['Survived'].mean().sort_values(by='
```

Out[1764]:

	Ticket_start	Survived
5	P	0.646154
0	1	0.625000
6	Rare_High_Surv	0.625000
1	2	0.464481
4	C	0.340426
8	S	0.323077
2	3	0.239203
7	Rare_Low_Surv	0.148936
3	A	0.068966

In [1765]:
```
ticket_dummies_train = pd.get_dummies(tit_train['Ticket_start'], prefix = 'Ticket_sta
tit_train = pd.concat([tit_train.drop(['Ticket', 'Ticket_start'], axis=1),
          ticket_dummies_train], axis=1)

ticket_dummies_test = pd.get_dummies(tit_test['Ticket_start'], prefix = 'Ticket_start
tit_test = pd.concat([tit_test.drop(['Ticket', 'Ticket_start'], axis=1),
          ticket_dummies_test], axis=1)

# tit_test.drop('Ticket', axis=1, inplace=True)
# tit_train.drop('Ticket', axis=1, inplace=True)

# tit_train['Ticket_start'] = tit_train['Ticket_start'].map( {'1': 1, '2': 2, '3': 3,
# tit_test['Ticket_start'] = tit_test['Ticket_start'].map( {'1': 1, '2': 2, '3': 3,'A
```

Jednym z ostatnich etapów przygotowywania i analizy naszych danych będzie sprawdzenie jaki wpływ na przeżywalność pasażerów miała ilość osób z nimi podróżująca

In [1766]:
```
tit_train['Family'] = tit_train['SibSp'] + tit_train['Parch']
tit_test['Family'] = tit_test['SibSp'] + tit_test['Parch']
```

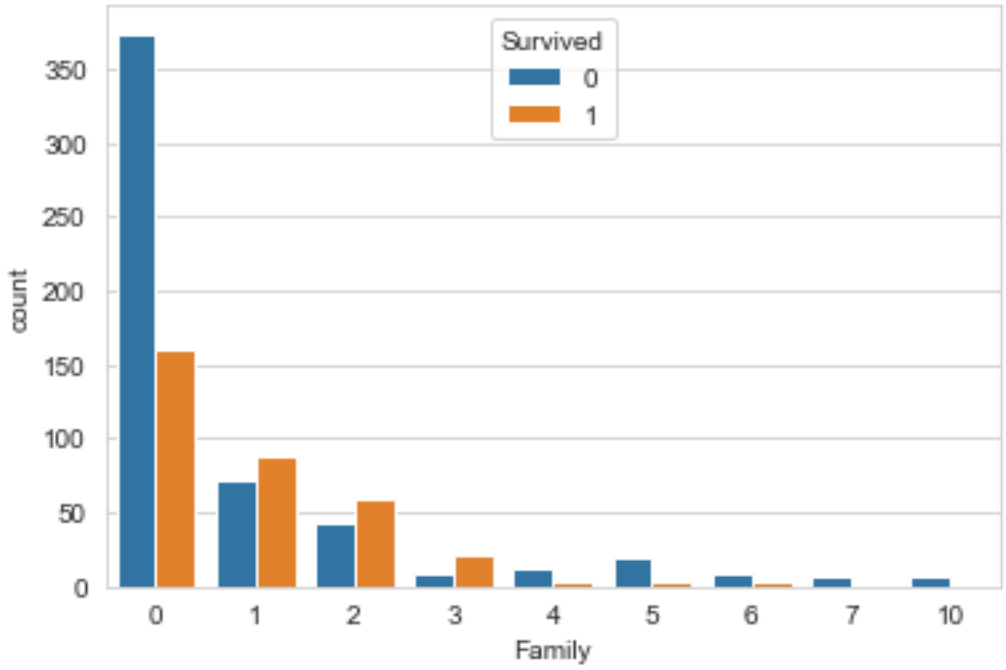
In [1767]:
```
tit_train.groupby('Family', as_index=False)['Survived'].mean().sort_values(by='Surviv
```

Out[1767]:

	Family	Survived
3	3	0.724138
2	2	0.578431
1	1	0.552795
6	6	0.333333
0	0	0.300935
4	4	0.200000
5	5	0.136364
7	7	0.000000
8	10	0.000000

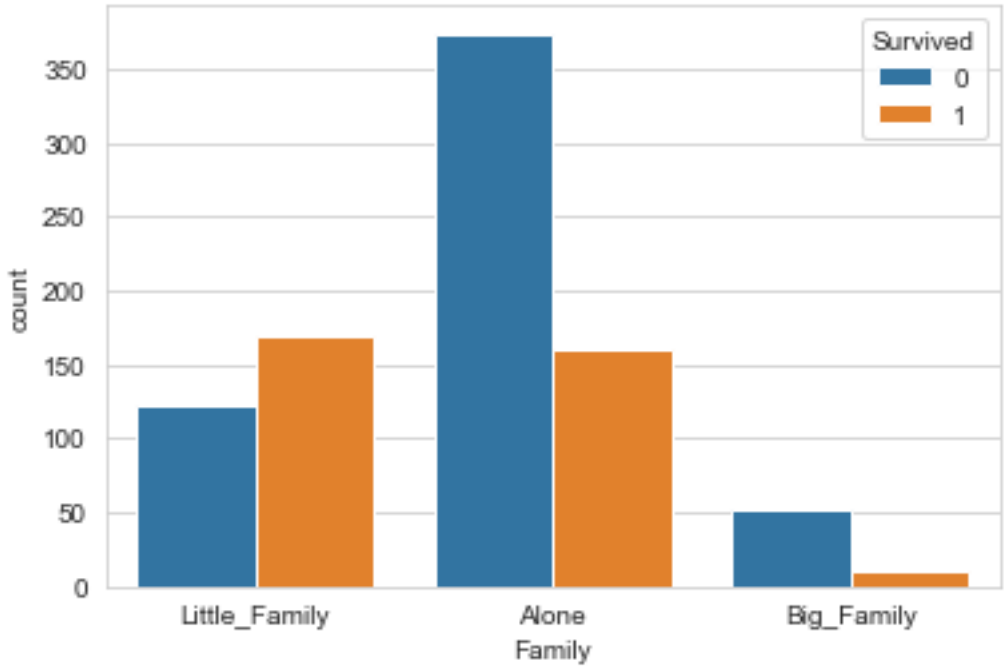
In [1768]:
```
sns.countplot(data=tit_train, x='Family', hue='Survived')
```

Out[1768]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x39e7e79fd0>
```



Kolejny raz, pierwszym krokiem będzie scalenie niektórych wartości. Tutaj podzielimy to w nastepujący sposób 1. Osoby podróżujące same('Alone') - wartość 0 2. Osoby podrużujące w małej rodzinie('Little_Family') - wartości 1,2,3 3. Osoby podruzujące w dużej rodzinie('Bif_Family') - wartości > 3

```
In [1769]: tit_train['Family'] = tit_train['Family'].replace([0], 'Alone')
           tit_train['Family'] = tit_train['Family'].replace([1, 2, 3], 'Little_Family')
           tit_train['Family'] = tit_train['Family'].replace([4, 5, 6, 7, 10], 'Big_Family')

           tit_test['Family'] = tit_test['Family'].replace([0], 'Alone')
           tit_test['Family'] = tit_test['Family'].replace([1, 2, 3], 'Little_Family')
           tit_test['Family'] = tit_test['Family'].replace([4, 5, 6, 7, 10], 'Big_Family')

In [1770]: tit_train[['Family', 'Survived']].groupby(['Family'], as_index=False).mean().sort_val

Out[1770]:          Family  Survived
           2  Little_Family  0.578767
           0          Alone  0.300935
           1     Big_Family  0.161290

In [1771]: sns.countplot(data=tit_train, x='Family', hue='Survived')

Out[1771]: <matplotlib.axes._subplots.AxesSubplot at 0x39e4770898>
```

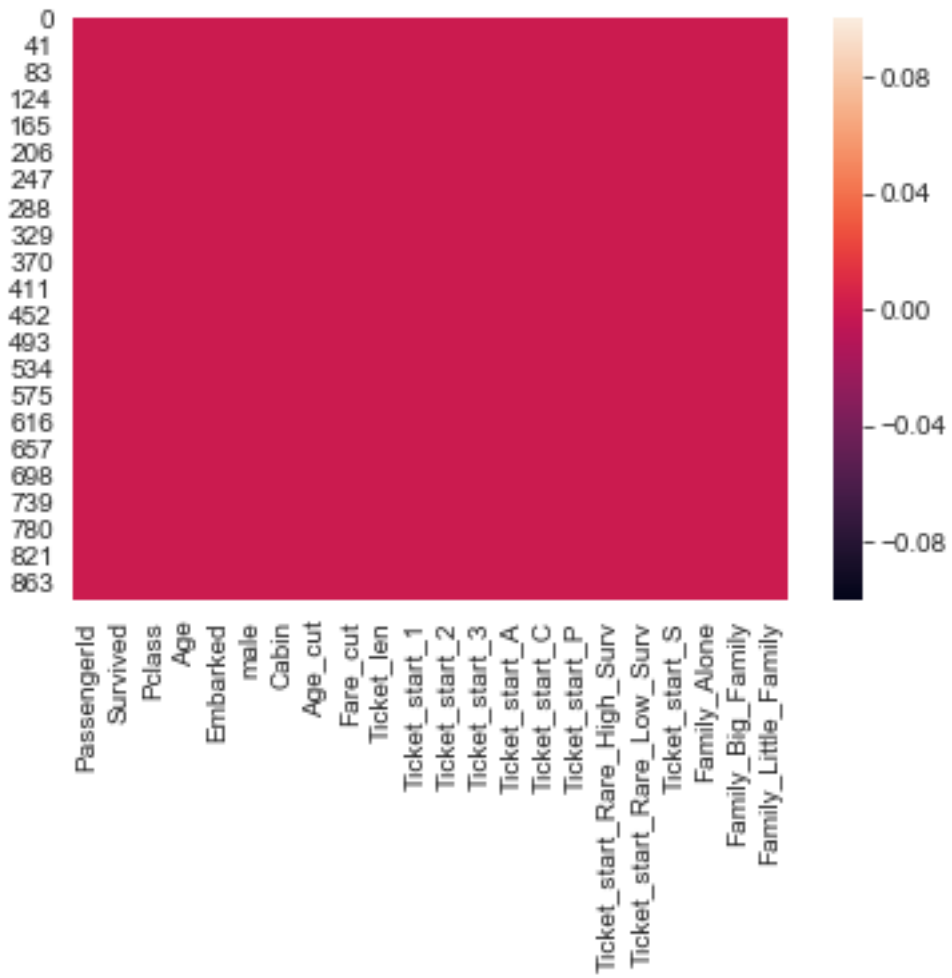


Jak widzimy osoby podrużujące w małej rodzinie miały dużo wieskze szanse na przezycie niż osoby podrużujące samotnie, oraz w duzej rodzinie

```
In [1772]: tit_train = pd.concat([tit_train.drop(['SibSp', 'Parch', 'Family'], axis=1),
                                  pd.get_dummies(tit_train['Family'], prefix = 'Fam

           tit_test = pd.concat([tit_test.drop(['SibSp', 'Parch', 'Family'], axis=1),
                                  pd.get_dummies(tit_test['Family'], prefix = 'Fami
           # tit_train.drop(['SibSp', 'Parch'], axis=1, inplace=True)
           # tit_test.drop(['SibSp', 'Parch'], axis=1, inplace=True)
           # tit_train['Family'] = tit_train['Family'].map( {'Little_Family': 0, 'Alone': 1, 'Bi
           # tit_test['Family'] = tit_test['Family'].map( {'Little_Family': 0, 'Alone': 1, 'Big_
```

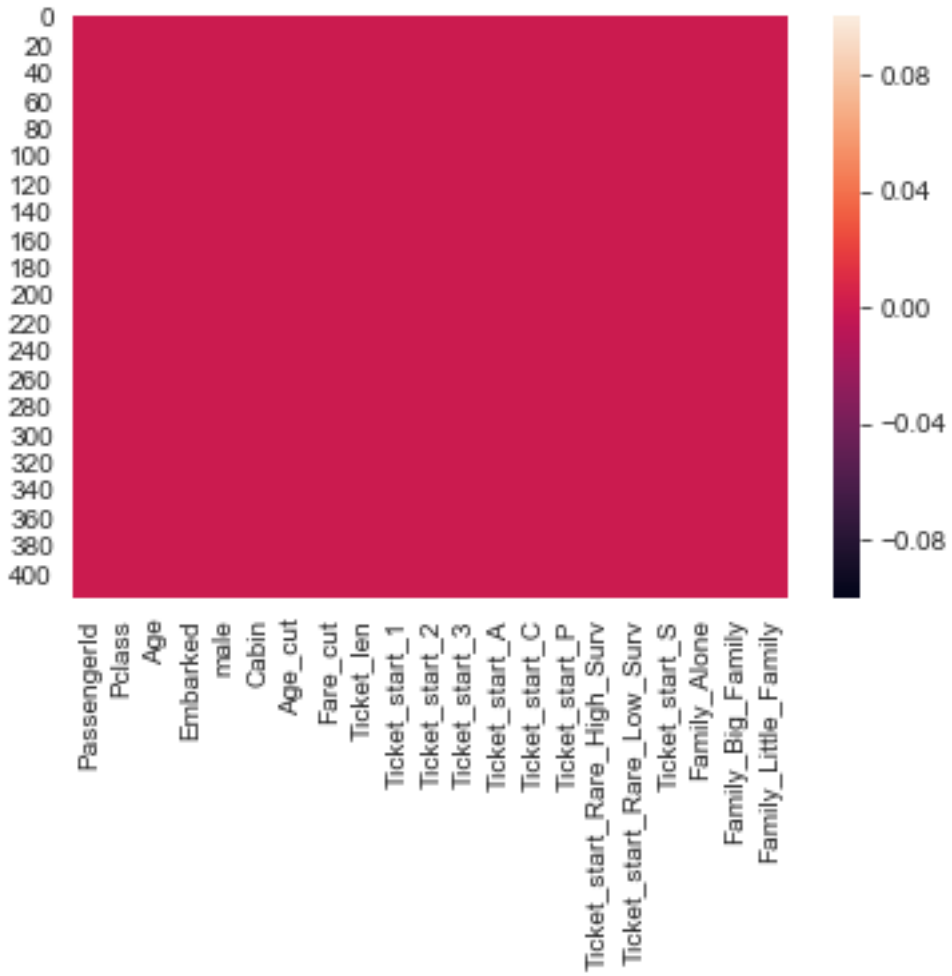
Na koniec sprawdzimy czy na pewno nie ma żadnych wartości Null/NaN

```
In [1773]: sns.heatmap(tit_train.isnull())

Out[1773]: <matplotlib.axes._subplots.AxesSubplot at 0x39e98a6898>
```




```
In [1774]: sns.heatmap(tit_test.isnull())
```

```
Out[1774]: <matplotlib.axes._subplots.AxesSubplot at 0x39e97212b0>
```



Usuwamy jedną wartość NaN

```
In [1775]: tit_test.fillna({'Fare_cut': tit_test['Fare_cut'].median()}, inplace=True)

In [1776]: tit_train.head()

Out[1776]:    PassengerId  Survived  Pclass   Age Embarked  male  Cabin Age_cut  \
           0            1         0       3  22.0        2     1      0       2
           1            2         1       1  38.0        1     0      1       2
           2            3         1       3  26.0        2     0      0       2
           3            4         1       1  35.0        2     0      1       2
           4            5         0       3  35.0        2     1      0       2

              Fare_cut  Ticket_len          ...           Ticket_start_3  Ticket_start_A  \
           0         0           9          ...                        0               1
           1         1           8          ...                        0               0
           2         0          16          ...                        0               0
           3         1           6          ...                        0               0
```

```
4         0           6          ...                          1                0

   Ticket_start_C  Ticket_start_P  Ticket_start_Rare_High_Surv  \
0               0               0                            0
1               0               1                            0
2               0               0                            0
3               0               0                            0
4               0               0                            0

   Ticket_start_Rare_Low_Surv  Ticket_start_S  Family_Alone  \
0                           0               0             0
1                           0               0             0
2                           0               1             1
3                           0               0             0
4                           0               0             1

   Family_Big_Family  Family_Little_Family
0                  0                     1
1                  0                     1
2                  0                     0
3                  0                     1
4                  0                     0

[5 rows x 22 columns]
```

Na koniec usuniemy niepotrzebną kolumnę PassengerId

```
In [1777]: y_train = tit_train['Survived']
           X_train = tit_train.drop(['Survived','PassengerId'], axis=1)

In [1778]: from sklearn.linear_model import LogisticRegression

In [1779]: lgmodel = LogisticRegression()

In [1780]: lgmodel.fit(X_train, y_train)

C:\Users\acer\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:433: FutureWarning: D
  FutureWarning)


Out[1780]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                     intercept_scaling=1, max_iter=100, multi_class='warn',
                     n_jobs=None, penalty='l2', random_state=None, solver='warn',
                     tol=0.0001, verbose=0, warm_start=False)

In [1781]: test_set = tit_test.drop('PassengerId', axis=1)

In [ ]:
```

Teraz kolejno sprawdzimy skuteczność różnych modeli

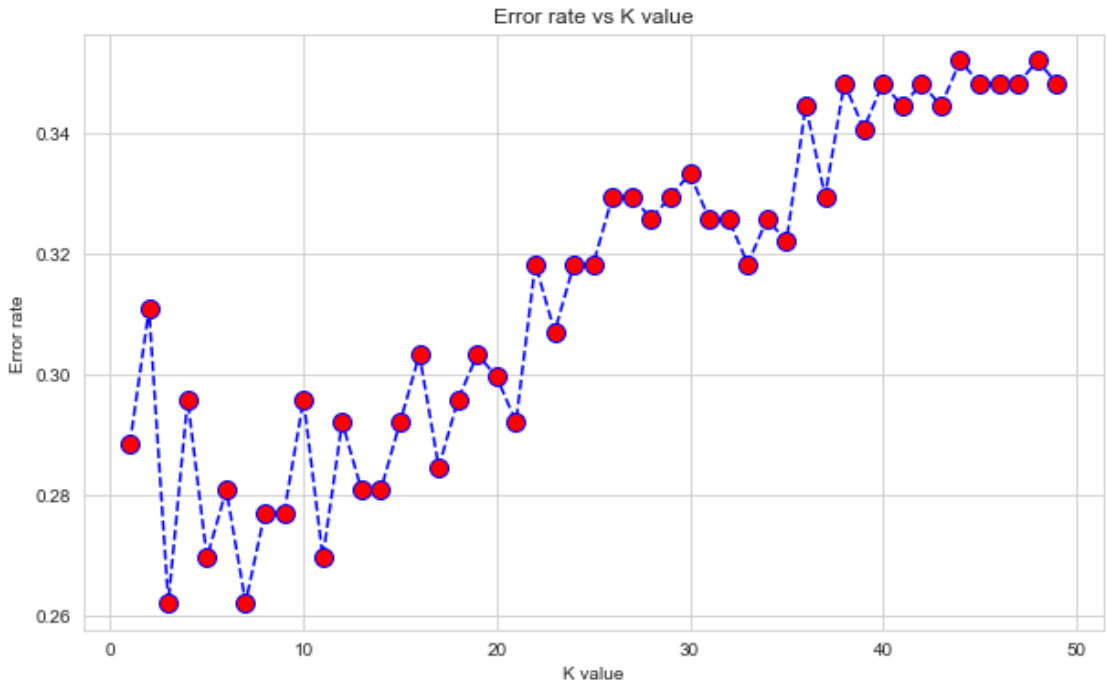
```
In [1782]: from sklearn.model_selection import train_test_split
           X_train, X_test, y_train, y_test = train_test_split(X_train, y_train, test_size=0.3,

In [1783]: lgmodel_check = LogisticRegression()
           lgmodel_check.fit(X_train, y_train)
           predictions_check = lgmodel_check.predict(X_test)
           from sklearn.metrics import classification_report
           from sklearn.metrics import confusion_matrix
           from sklearn.metrics import accuracy_score

           print(classification_report(y_test, predictions_check))
           print('\n')
           print(confusion_matrix(y_test, predictions_check))
           print('\n')
           print('Accuracy score: ' + str(accuracy_score(y_test, predictions_check)))
```

	precision	recall	f1-score	support
0	0.79	0.91	0.84	161
1	0.82	0.62	0.71	106
micro avg	0.80	0.80	0.80	267
macro avg	0.81	0.77	0.78	267
weighted avg	0.80	0.80	0.79	267

```
[[147  14]
 [ 40  66]]


Accuracy score: 0.797752808988764


C:\Users\acer\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:433: FutureWarning: D
  FutureWarning)


In [1784]: from sklearn.model_selection import GridSearchCV
           from sklearn.svm import SVC

In [1785]: svc = SVC()
           svc.fit(X_train, y_train)
           pred = svc.predict(X_test)

C:\Users\acer\Anaconda3\lib\site-packages\sklearn\svm\base.py:196: FutureWarning: The default va
  "avoid this warning.", FutureWarning)
```

```
In [1786]: param_grid = {'C': np.arange(150,190,1), 'gamma':np.arange(0.0005, 0.002, 0.0002) }
           grid = GridSearchCV(SVC(), param_grid, verbose=3)
           grid.fit(X_train, y_train)

C:\Users\acer\Anaconda3\lib\site-packages\sklearn\model_selection\_split.py:2053: FutureWarning:
  warnings.warn(CV_WARNING, FutureWarning)
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.0s remaining:    0.0s


Fitting 3 folds for each of 320 candidates, totalling 960 fits
[CV] C=150, gamma=0.0005 ...
[CV] ... C=150, gamma=0.0005, score=0.8028846153846154, total=   0.0s
[CV] C=150, gamma=0.0005 ...
[CV] ... C=150, gamma=0.0005, score=0.7922705314009661, total=   0.0s
[CV] C=150, gamma=0.0005 ...
[CV] ... C=150, gamma=0.0005, score=0.8502415458937198, total=   0.0s
[CV] C=150, gamma=0.0007 ...
...
...
...
[CV] C=189, gamma=0.0018999999999999998 ...
[CV]  C=189, gamma=0.0018999999999999998, score=0.7884615384615384, total=   0.0s
[CV] C=189, gamma=0.0018999999999999998 ...
[CV]  C=189, gamma=0.0018999999999999998, score=0.8164251207729468, total=   0.0s
[CV] C=189, gamma=0.0018999999999999998 ...
[CV]  C=189, gamma=0.0018999999999999998, score=0.8599033816425121, total=   0.0s


[Parallel(n_jobs=1)]: Done 960 out of 960 | elapsed:   27.1s finished


Out[1786]: GridSearchCV(cv='warn', error_score='raise-deprecating',
                estimator=SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
             decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
             kernel='rbf', max_iter=-1, probability=False, random_state=None,
             shrinking=True, tol=0.001, verbose=False),
                fit_params=None, iid='warn', n_jobs=None,
                param_grid={'C': array([150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160,
                163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175,
                176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188,
                189]), 'gamma': array([0.0005, 0.0007, 0.0009, 0.0011, 0.0013, 0.0015, 0.0017,
                pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
                scoring=None, verbose=3)

In [1787]: print('Best params: ' + str(grid.best_params_))
           print('\n')
           print('Best score' + str(grid.best_score_))
```

```
Best params: {'C': 150, 'gamma': 0.0013}


Best score0.8247588424437299


In [1788]: grid_pred = grid.predict(X_test)
           print(confusion_matrix(grid_pred, y_test))
           print('\n')
           print(classification_report(grid_pred, y_test))
           print('\n')
           print('Accuracy score: ' + str(accuracy_score(y_test, grid_pred)))

[[145  35]
 [ 16  71]]


              precision    recall  f1-score   support

           0       0.90      0.81      0.85       180
           1       0.67      0.82      0.74        87

   micro avg       0.81      0.81      0.81       267
   macro avg       0.79      0.81      0.79       267
weighted avg       0.83      0.81      0.81       267



Accuracy score: 0.8089887640449438


In [1789]: from sklearn.neighbors import KNeighborsClassifier

           error_rate = []

           for k in range(1,50):
               knn = KNeighborsClassifier(n_neighbors=k)
               knn.fit(X_train, y_train)
               pred_i = knn.predict(X_test)
               error_rate.append(np.mean(abs(pred_i - y_test)))

In [1790]: plt.figure(figsize=(10,6))
           plt.plot(range(1,50), error_rate, color='blue', linestyle='dashed', marker='o', marke
           plt.title('Error rate vs K value')
           plt.ylabel('Error rate')
           plt.xlabel('K value')

Out[1790]: Text(0.5, 0, 'K value')
```




```
In [1805]: k=3
           knn = KNeighborsClassifier(n_neighbors=k)

In [1806]: knn.fit(X_train,y_train)
           knn_pred = knn.predict(X_test)
           print(confusion_matrix(knn_pred, y_test))
           print('\n')
           print(classification_report(knn_pred, y_test))
           print(accuracy_score(y_test, knn_pred))

[[140  49]
 [ 21  57]]


              precision    recall  f1-score   support

           0       0.87      0.74      0.80       189
           1       0.54      0.73      0.62        78

   micro avg       0.74      0.74      0.74       267
   macro avg       0.70      0.74      0.71       267
weighted avg       0.77      0.74      0.75       267

0.7378277153558053
```

```
In [1807]: from sklearn.tree import DecisionTreeClassifier

In [1808]: dtree = DecisionTreeClassifier()

In [1809]: dtree.fit(X_train, y_train)

Out[1809]: DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
                       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, presort=False, random_state=None,
                       splitter='best')

In [1810]: predictions = dtree.predict(X_test)

In [1811]: print(confusion_matrix(predictions, y_test))
           print('\n')
           print(classification_report(predictions, y_test))
           print(accuracy_score(y_test, predictions))

[[131  31]
 [ 30  75]]


              precision    recall  f1-score   support

           0       0.81      0.81      0.81       162
           1       0.71      0.71      0.71       105

   micro avg       0.77      0.77      0.77       267
   macro avg       0.76      0.76      0.76       267
weighted avg       0.77      0.77      0.77       267

0.7715355805243446


In [1812]: from sklearn.ensemble import RandomForestClassifier

In [1817]: rfc = RandomForestClassifier(n_estimators=400)

In [1818]: rfc.fit(X_train, y_train)

Out[1818]: RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                       max_depth=None, max_features='auto', 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=400, n_jobs=None,
                       oob_score=False, random_state=None, verbose=0,
                       warm_start=False)
```

```
In [1819]: rfc_pred = rfc.predict(X_test)

In [1820]: print(confusion_matrix(rfc_pred, y_test))
           print('\n')
           print(classification_report(rfc_pred, y_test))
           print(accuracy_score(y_test, rfc_pred))
```

```
[[138  29]
 [ 23  77]]


              precision    recall  f1-score   support

           0       0.86      0.83      0.84       167
           1       0.73      0.77      0.75       100

   micro avg       0.81      0.81      0.81       267
   macro avg       0.79      0.80      0.79       267
weighted avg       0.81      0.81      0.81       267

0.8052434456928839
```

```
In [1803]: from sklearn import model_selection

In [1804]: kfold = model_selection.KFold(n_splits=10, random_state=12)
           cv_decision_tree = DecisionTreeClassifier(criterion="gini", max_depth=8, min_samples_

           results = model_selection.cross_val_score(cv_decision_tree, X_train, y_train, cv=kfol
           print("Decision Tree accuracy: Final mean:%.3f%%, Final standard deviation:(%.3f%%)"
           print('Decision Tree accuracies from each of the 10 folds using kfold:',results)
```

```
Decision Tree accuracy: Final mean:79.580%, Final standard deviation:(5.305%)
Decision Tree accuracies from each of the 10 folds using kfold: [0.77777778 0.82539683 0.8387096
 0.82258065 0.74193548 0.82258065 0.80645161]
```

1.3 Wnioski

Jak widzimy nasz najlepszy model model miał skuteczność ok. 81%. Wynik był zero-jedynkowy, pasażer przeżył albo nie przeżył. A więc minimalna skuteczność wynosiłaby 50%. Ale jeśli weźmiemy pod uwagę, że 67% pasażerów nie przeżyło wypadku, jeśli nasz model przewidywałby, że każdy pasażer umrze, jego skuteczność wynosiłaby właśnie około 67%. Biorąc pod uwagę powyższe informacje - skuteczność naszego modelu możemy ocenić na bardzo dobrą.

Lepszy wynik może można byłoby uzyskać przeprowadzając dokładniejszą analizę wieku, np. uzupełniając brakujące wartości inaczej niż po prostu średnim wiekiem dla klasy, jedną z opcji jest wziąć pod uwagę jeszcze płeć pasażera. Mozna także pozyskać dokładniejsze dane co do kabin pasażerów, gdyż tutaj połowa danych była pusta więc ta kolumna w naszym modelu mogła nie odgrywać znaczącej roli. Można także przeprowadzić dokładniejszą analizę statystyczną niektórych wartości np. Embarked, Cabin czy Ticket_len pod względem ich przydatności do dalszej analizy