Patryk Ostrowski

Mod_4, zad_2 – Titanic, EDA

1. Przedstawiam losową próbkę danych:

```
# sniff data
print('##################################################')
print('################ MEET THE DATA SET ################')
print('##################################################')
print(df.sample(5).to_string())
print()
```

```
##################################################
################ MEET THE DATA SET ################
##################################################
     pclass  survived                                     name     sex   age  sibsp  parch  ticket    fare cabin embarked boat  body             home.dest
270     1.0       1.0  Smith, Mrs. Lucien Philip (Mary Eloise Hughes)  female  18.0    1.0    0.0   13695  60.000   C31        S    6   NaN        Huntington, WV
34      1.0       0.0                      Borebank, Mr. John James    male  42.0    0.0    0.0  110489  26.550   D22        S  NaN   NaN  London / Winnipeg, MB
725     3.0       1.0                          Connolly, Miss. Kate  female  22.0    0.0    0.0  370373   7.750   NaN        Q   13   NaN               Ireland
381     2.0       0.0             Corbett, Mrs. Walter H (Irene Colvin)  female  30.0    0.0    0.0  237249  13.000   NaN        S  NaN   NaN             Provo, UT
149     1.0       1.0   Harris, Mrs. Henry Birkhardt (Irene Wallach)  female  35.0    1.0    0.0   36973  83.475   C83        S    D   NaN           New York, NY
```

2. Zmieniam nazwy kolumn na bardziej user-friendly i ponownie przedstawiam losową próbkę danych:

```
# rename columns and sniff data again
print('##################################################')
print('################ RENAMED COLUMNS ################')
print('##################################################')
df.columns = ['class', 'survived', 'full_name', 'sex', 'age',
'siblings/spouse', 'parents/children', 'ticket_no', 'fare_price',
'cabin_no', 'embarked', 'boat_no', 'body_no', 'destination']
print(df.sample(15).to_string())
print()
```

```
##################################################
################ RENAMED COLUMNS ################
##################################################
      class  survived                                      full_name     sex   age  siblings/spouse  parents/children          ticket_no  fare_price cabin_no embarked boat_no  body_no                      destination
511     2.0       0.0                        Myles, Mr. Thomas Francis    male  62.0              0.0               0.0             240276      9.6875      NaN        Q     NaN      NaN                    Cambridge, MA
1071    3.0       1.0   O'Brien, Mrs. Thomas (Johanna "Hannah" Godfrey)  female   NaN              1.0               0.0             370365     15.5000      NaN        Q     NaN      NaN                              NaN
709     3.0       1.0                        Carr, Miss. Helen "Ellen"  female  16.0              0.0               0.0             367231      7.7500      NaN        Q      16      NaN  Co Longford, Ireland New York, NY
319     1.0       1.0                        Wilson, Miss. Helen Alice  female  31.0              0.0               0.0              16966    134.5000  E39 E41        C       3      NaN                              NaN
986     3.0       0.0                     Maenpaa, Mr. Matti Alexanteri    male  22.0              0.0               0.0  STON/O 2. 3101275      7.1250      NaN        S     NaN      NaN                              NaN
1015    3.0       0.0                               Meo, Mr. Alfonzo    male  55.5              0.0               0.0        A.5. 11206      8.0500      NaN        S     NaN    201.0                              NaN
75      1.0       0.0                        Colley, Mr. Edward Pomeroy    male  47.0              0.0               0.0               5727     25.5875      E58        S     NaN      NaN                      Victoria, BC
962     3.0       0.0                               Lennon, Mr. Denis    male   NaN              1.0               0.0             370371     15.5000      NaN        Q     NaN      NaN                              NaN
1035    3.0       1.0  Moubarek, Master. Halim Gonios ("William George")    male   NaN              1.0               1.0               2661     15.2458      NaN        C       C      NaN                              NaN
241     1.0       0.0                           Rood, Mr. Hugh Roscoe    male   NaN              0.0               0.0             113767     50.0000      A32        S     NaN      NaN                      Seattle, WA
195     1.0       1.0                          Maioni, Miss. Roberta  female  16.0              0.0               0.0             110152     86.5000      B79        S       8      NaN                              NaN
123     1.0       1.0                  Frolicher-Stehli, Mr. Maxmillian    male  60.0              1.0               1.0              13567     79.2000      B41        C       5      NaN              Zurich, Switzerland
371     2.0       1.0                    Christy, Mrs. (Alice Frances)  female  45.0              0.0               2.0             237789     30.0000      NaN        S      12      NaN                           London
565     2.0       0.0                     Sobey, Mr. Samuel James Hayden    male  25.0              0.0               0.0        C.A. 29178     13.0000      NaN        S     NaN      NaN         Cornwall / Houghton, MI
1179    3.0       0.0                           Sage, Mr. John George    male   NaN              1.0               9.0           CA. 2343     69.5500      NaN        S     NaN      NaN                              NaN
```

3. Przedstawiam garść faktów w kontekście zbioru danych mającego zostać poddanym analizie eksploracyjnej:

```
# analyze facts
print('##################################################')
print('##### A FEW QUICK FACTS ON THE CIRCUMSTANCES #####')
print('##################################################')
total_passengers = 2200
passengers_in_this_set = len(df)
print(f'{total_passengers} traveled in total. This set analyses
{passengers_in_this_set} persons who have been found either alive or
dead.')
missing_passengers = total_passengers - passengers_in_this_set
print(f'What happened to {missing_passengers} is unknown.')
bodies_not_found = df['body_no'].isnull().sum()
survivors = (df['survived'] == 1).sum()
print(f'{bodies_not_found} bodies have never been found.')
print(f'{survivors} persons out of {passengers_in_this_set} survived.')
non_survivors = (df['survived'] == 0).sum()
print(f'{non_survivors} passengers {passengers_in_this_set} death has been
confirmed.')
print()
```

```
##################################################
##### A FEW QUICK FACTS ON THE CIRCUMSTANCES #####
##################################################
2200 traveled in total. This set analyses 1310 persons who have been found either alive or dead.
What happened to 890 is unknown.
1189 bodies have never been found.
500 persons out of 1310 survived.
809 passengers 1310 death has been confirmed.
```

4. Przybliżam charakterystykę zestawu – jego problemy dot. wybrakowanych informacji. Jak widać takowe występują w każdej kolumnie, gdzieniegdzie w ilościach znaczących:

```
# data set analysis
print('################################################')
print('############### DATA SET ANALYSIS ###############')
print('################################################')
print()
print('############## NULL VALUES COUNTER ##############')
print()
for column in df:
    column_sum_of_null = df[column].isnull().sum()
    print(f'{column_sum_of_null} times null in {column}.')
```

```
################################################
############### DATA SET ANALYSIS ###############
################################################

############## NULL VALUES COUNTER ##############

1 times null in class.
1 times null in survived.
1 times null in full_name.
1 times null in sex.
264 times null in age.
1 times null in siblings/spouse.
1 times null in parents/children.
1 times null in ticket_no.
2 times null in fare_price.
1015 times null in cabin_no.
3 times null in embarked.
824 times null in boat_no.
1189 times null in body_no.
565 times null in destination.
```

5. Bardzo pobieżna analiza poszczególnych kolumn – w niektórych przypadkach nieco głębsza – przeprowadzona w celu określenia co ciekawego będzie można z tych danych powyciągać.

```
print()
print('############### COLUMN: CLASS ###############')
print()
print('Column "class" type:', df['class'].dtype)
print('Unique values:', df['class'].unique())
print()
```

```
############### COLUMN: CLASS ###############

Column "class" type: float64
Unique values: [ 1.  2.  3. nan]
```

```
print('############### COLUMN: SURVIVED ###############')
print()
print('Column "survived" type:', df['survived'].dtype)
print('Unique values:', df['survived'].unique())
```

```
############### COLUMN: SURVIVED ###############

Column "survived" type: float64
Unique values: [ 1.  0. nan]
```

```
print('############### COLUMN: FULL NAME ###############')
print()
print('Column "full_name" type:', df['full_name'].dtype)
print('Unique values:', df['full_name'].unique())
```

```
############### COLUMN: FULL NAME ###############

Column "full_name" type: object
Unique values: ['Allen, Miss. Elisabeth Walton' 'Allison, Master. Hudson Trevor'
 'Allison, Miss. Helen Loraine' ... 'Zakarian, Mr. Ortin'
 'Zimmerman, Mr. Leo' nan]
```

```python
print('############### COLUMN: SEX ###############')
print()
print('Column "sex" type:', df['sex'].dtype)
print('Unique values:', df['sex'].unique())
```

```
############### COLUMN: SEX ###############

Column "sex" type: object
Unique values: ['female' 'male' nan]
```

```python
print('############### COLUMN: AGE ###############')
print()
print('Column "age" type:', df['age'].dtype)
print('Unique values:', np.sort(df['age'].unique()))
```

```
############### COLUMN: AGE ###############

Column "age" type: float64
Unique values: [ 0.1667  0.3333  0.4167  0.6667  0.75    0.8333  0.9167  1.      2.
  3.      4.      5.      6.      7.      8.      9.      10.     11.
 11.5    12.     13.     14.     14.5    15.     16.     17.     18.
 18.5    19.     20.     20.5    21.     22.     22.5    23.     23.5
 24.     24.5    25.     26.     26.5    27.     28.     28.5    29.
 30.     30.5    31.     32.     32.5    33.     34.     34.5    35.
 36.     36.5    37.     38.     38.5    39.     40.     40.5    41.
 42.     43.     44.     45.     45.5    46.     47.     48.     49.
 50.     51.     52.     53.     54.     55.     55.5    56.     57.
 58.     59.     60.     60.5    61.     62.     63.     64.     65.
 66.     67.     70.     70.5    71.     74.     76.     80.        nan]
```

```python
print('############### COLUMN: SIBLINGS/SPOUSE ###############')
print()
print('Column "siblings/spouse" type:', df['siblings/spouse'].dtype)
print('Unique values:', df['siblings/spouse'].unique())
```

```
############### COLUMN: SIBLINGS/SPOUSE ###############

Column "siblings/spouse" type: float64
Unique values: [ 0.  1.  2.  3.  4.  5.  8. nan]
```

```
print('############### COLUMN: PARENTS/CHILDREN ###############')
print()
print('Column "parents/children" type:', df['parents/children'].dtype)
print('Unique values:', df['parents/children'].unique())
```

```
############### COLUMN: PARENTS/CHILDREN ###############

Column "parents/children" type: float64
Unique values: [ 0.  2.  1.  4.  3.  5.  6.  9. nan]
```

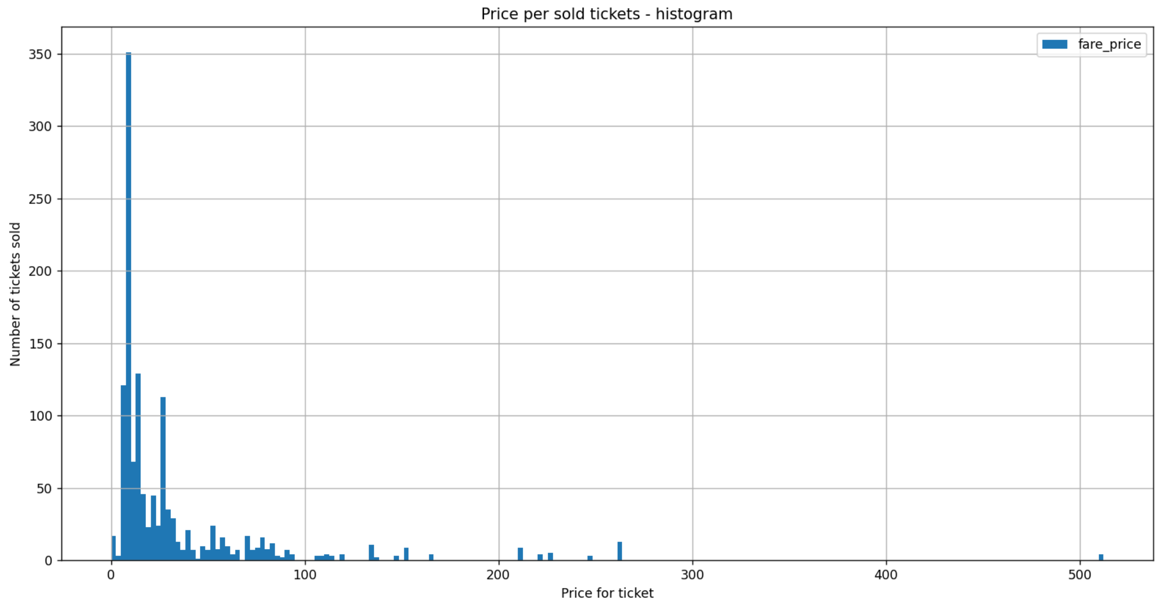
```
print('############### COLUMN: TICKET NO ###############')
print()
print('Column "ticket_no" type:', df['ticket_no'].dtype)
print('Unique values:', sorted(df['ticket_no'].astype(str).unique()))
```

```
############### COLUMN: TICKET NO ###############

Column "ticket_no" type: object
Unique values: ['110152', '110413', '110465', '110469', '110489', '110564', '110813', '111163', '111240', '111320', '111
```

```python
print('############### COLUMN: FARE PRICE ###############')
print()
print('Column "fare_price" type:', df['fare_price'].dtype)
print('Unique values:', np.sort(df['fare_price'].unique()))
total_cost_of_journey = df['fare_price'].sum()
print()
print(f'All passengers paid {total_cost_of_journey} for the journey.')
average_ticket_price = df['fare_price'].mean()
median_ticket_price = df['fare_price'].median()
print(f'Average ticket price was {round(average_ticket_price, 2)} while
the median was {round(median_ticket_price, 2)}.')
# histogram
df['fare_price'].hist(bins = 200, legend=True)
plt.title('Price per sold tickets - histogram')
plt.xlabel('Price for ticket')
plt.ylabel('Number of tickets sold')
plt.show()
```

Na następnej stronie outcome z konsoli a jeszcze niżej histogram.

```
############### COLUMN: FARE PRICE ###############

Column "fare_price" type: float64
Unique values: [  0.       3.1708   4.0125   5.       6.2375   6.4375   6.45     6.4958
   6.75     6.8583   6.95     6.975    7.       7.0458   7.05     7.0542
   7.125    7.1417   7.225    7.2292   7.25     7.2833   7.3125   7.4958
   7.5208   7.55     7.575    7.5792   7.6292   7.65     7.7208   7.725
   7.7292   7.7333   7.7375   7.7417   7.75     7.775    7.7792   7.7875
   7.7958   7.8      7.8208   7.8292   7.85     7.8542   7.875    7.8792
   7.8875   7.8958   7.925    8.0292   8.05     8.1125   8.1375   8.1583
   8.3      8.3625   8.4042   8.4333   8.4583   8.5167   8.6542   8.6625
   8.6833   8.7125   8.85     8.9625   9.       9.2167   9.225    9.325
   9.35     9.475    9.4833   9.5      9.5875   9.6875   9.825    9.8375
   9.8417   9.8458  10.1708  10.4625  10.5     10.5167  10.7083  11.1333
  11.2417  11.5     12.      12.1833  12.275   12.2875  12.35    12.475
  12.525   12.65    12.7375  12.875   13.      13.4167  13.5     13.775
  13.7917  13.8583  13.8625  13.9     14.      14.1083  14.4     14.4542
  14.4583  14.5     15.      15.0333  15.0458  15.05    15.1     15.2458
  15.5     15.55    15.5792  15.7417  15.75    15.85    15.9     16.
  16.1     16.7     17.4     17.8     18.      18.75    18.7875  19.2583
  19.5     19.9667  20.2125  20.25    20.525   20.575   21.      21.075
  21.6792  22.025   22.3583  22.525   23.      23.25    23.45    24.
  24.15    25.4667  25.5875  25.7     25.7417  25.925   25.9292  26.
  26.25    26.2833  26.2875  26.3875  26.55    27.      27.4458  27.7208
  27.75    27.9     28.5     28.5375  28.7125  29.      29.125   29.7
  30.      30.0708  30.5     30.6958  31.      31.275   31.3875  31.5
  31.6792  31.6833  32.3208  32.5     33.      33.5     34.0208  34.375
  34.6542  35.      35.5     36.75    37.0042  38.5     39.      39.4
  39.6     39.6875  40.125   41.5792  42.4     42.5     45.5     46.9
  47.1     49.5     49.5042  50.      50.4958  51.4792  51.8625  52.
  52.5542  53.1     55.      55.4417  55.9     56.4958  56.9292  57.
  57.75    57.9792  59.4     60.      61.175   61.3792  61.9792  63.3583
  65.      66.6     69.3     69.55    71.      71.2833  73.5     75.2417
  75.25    76.2917  76.7292  77.2875  77.9583  78.2667  78.85    79.2
  79.65    80.      81.8583  82.1708  82.2667  83.1583  83.475   86.5
  89.1042  90.      91.0792  93.5    106.425  108.9    110.8833 113.275
 120.     133.65   134.5    135.6333 136.7792 146.5208 151.55   153.4625
 164.8667 211.3375 211.5    221.7792 227.525  247.5208 262.375  263.
 512.3292      nan]

All passengers paid 43550.4869 for the journey.
Average ticket price was 33.3 while the median was 14.45.
```



Histogram pokazuje ile biletów w danym przedziale cenowym zostało sprzedanych.

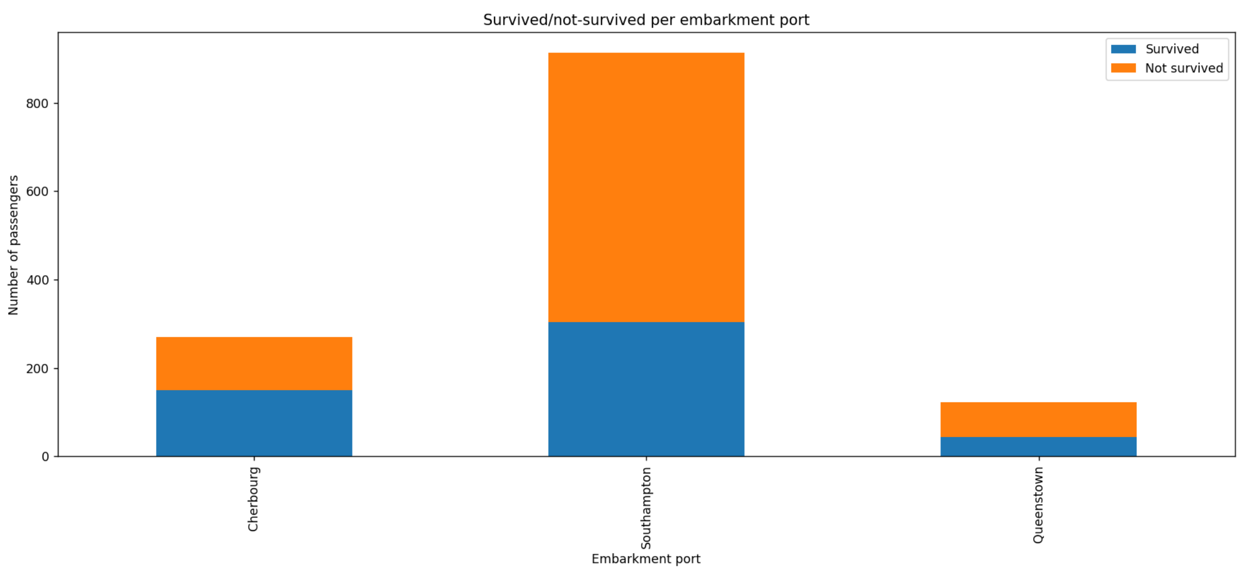
```python
print('############### COLUMN: CABIN NO. ###############')
print()
print('Column "cabin_no" type:', df['cabin_no'].dtype)
print('Unique values:', df['cabin_no'].unique())
print()
known_cabins_assignment_sum = df['cabin_no'].count()
unknown_cabins_assignment_sum = df['cabin_no'].isnull().sum()
print(f'{known_cabins_assignment_sum} allocations to cabins have been identified. Still allocation of {unknown_cabins_assignment_sum} cabins is unknown.')
```

```
############### COLUMN: CABIN NO. ###############

Column "cabin_no" type: object
Unique values: ['B5' 'C22 C26' 'E12' 'D7' 'A36' 'C101' nan 'C62 C64' 'B35' 'A23'
 'B58 B60' 'D15' 'C6' 'D35' 'C148' 'C97' 'B49' 'C99' 'C52' 'T' 'A31' 'C7'
 'C103' 'D22' 'E33' 'A21' 'B10' 'B4' 'E40' 'B38' 'E24' 'B51 B53 B55'
 'B96 B98' 'C46' 'E31' 'E8' 'B61' 'B77' 'A9' 'C89' 'A14' 'E58' 'E49' 'E52'
 'E45' 'B22' 'B26' 'C85' 'E17' 'B71' 'B20' 'A34' 'C86' 'A16' 'A20' 'A18'
 'C54' 'C45' 'D20' 'A29' 'C95' 'E25' 'C111' 'C23 C25 C27' 'E36' 'D34'
 'D40' 'B39' 'B41' 'B102' 'C123' 'E63' 'C130' 'B86' 'C92' 'A5' 'C51' 'B42'
 'C91' 'C125' 'D10 D12' 'B82 B84' 'E50' 'D33' 'C83' 'B94' 'D49' 'D45'
 'B69' 'B11' 'E46' 'C39' 'B18' 'D11' 'C93' 'B28' 'C49' 'B52 B54 B56' 'E60'
 'C132' 'B37' 'D21' 'D19' 'C124' 'D17' 'B101' 'D28' 'D6' 'D9' 'B80' 'C106'
 'B79' 'C47' 'D30' 'C90' 'E38' 'C78' 'C30' 'C118' 'D36' 'D48' 'D47' 'C105'
 'B36' 'B30' 'D43' 'B24' 'C2' 'C65' 'B73' 'C104' 'C110' 'C50' 'B3' 'A24'
 'A32' 'A11' 'A10' 'B57 B59 B63 B66' 'C28' 'E44' 'A26' 'A6' 'A7' 'C31'
 'A19' 'B45' 'E34' 'B78' 'B50' 'C87' 'C116' 'C55 C57' 'D50' 'E68' 'E67'
 'C126' 'C68' 'C70' 'C53' 'B19' 'D46' 'D37' 'D26' 'C32' 'C80' 'C82' 'C128'
 'E39 E41' 'D' 'F4' 'D56' 'F33' 'E101' 'E77' 'F2' 'D38' 'F' 'F G63'
 'F E57' 'F E46' 'F G73' 'E121' 'F E69' 'E10' 'G6' 'F38']

295 allocations to cabins have been identified. Still allocation of 1015 cabins is unknown.
```

```python
print('############### COLUMN: EMBARKED ###############')
print()
print('Column "embarked" type:', df['embarked'].dtype)
print('Unique values:', df['embarked'].unique())
embarked_from_cherbourg = (df['embarked'] == 'C').sum()
embarked_from_southampton = (df['embarked'] == 'S').sum()
embarked_from_queenstown = (df['embarked'] == 'Q').sum()
print(f'{embarked_from_southampton} persons onboarded in Southampton then
{embarked_from_cherbourg} onboarded in Cherbourg and finally
{embarked_from_queenstown} onbarded in Queenstown.')
print()
survivors_cherbourg = ((df['embarked'] == 'C') & (df['survived'] == 1)).sum()
non_survivors_cherbourg = ((df['embarked'] == 'C') & (df['survived'] ==
0)).sum()
print(f'From among of those who embarked in Cherbourg {survivors_cherbourg}
survived while {non_survivors_cherbourg} died.')

survivors_southampton = ((df['embarked'] == 'S') & (df['survived'] ==
1)).sum()
non_survivors_southampton = ((df['embarked'] == 'S') & (df['survived'] ==
0)).sum()
print(f'From among of those who embarked in Southampton
{survivors_southampton} survived while {non_survivors_southampton} died.')

survivors_queenstown = ((df['embarked'] == 'Q') & (df['survived'] ==
1)).sum()
non_survivors_queenstown = ((df['embarked'] == 'Q') & (df['survived'] ==
0)).sum()
print(f'From among of those who embarked in Queenstown {survivors_queenstown}
survived while {non_survivors_queenstown} died.')
```

```
############### COLUMN: EMBARKED ###############

Column "embarked" type: object
Unique values: ['S' 'C' nan 'Q']
914 persons onboarded in Southampton then 270 onboarded in Cherbourg and finally 123 onbarded in Queenstown.

From among of those who embarked in Cherbourg 150 survived while 120 died.
From among of those who embarked in Southampton 304 survived while 610 died.
From among of those who embarked in Queenstown 44 survived while 79 died.
```

```
#new data frame for chart purposes
embarkment_survived_df = pd.DataFrame({
    'embarked' : ['Cherbourg', 'Southampton', 'Queenstown'],
    'survived' : [survivors_cherbourg, survivors_southampton,
survivors_queenstown],
    'not-survived' : [non_survivors_cherbourg, non_survivors_southampton,
non_survivors_queenstown]
})

embarkment_survived_df.set_index('embarked', inplace=True)
embarkment_survived_df.plot(kind='bar', stacked=True)
plt.title('Survived/not-survived per embarkment port')
plt.xlabel('Embarkment port')
plt.ylabel('Number of passengers')
plt.legend(['Survived', 'Not survived'])
plt.tight_layout()
plt.show()
```



```
print('############### COLUMN: BOAT NO. ###############')
print()
print('Column "boat_no" type:', df['boat_no'].dtype)
print('Unique values:', sorted(df['boat_no'].astype(str).unique()))
boats_total = df['boat_no'].nunique()
print(f'There has been {boats_total} boats in total.')
in_boat = df['boat_no'].notnull().sum()
boat_passangers = in_boat / boats_total
print(f'{in_boat} persons got their boat which gives {int(boat_passangers)}
passengers in one boat.')
not_in_boat = df['boat_no'].isnull().sum()
print(f'{not_in_boat} persons did not get their boat.')
```

```
############### COLUMN: BOAT NO. ###############

Column "boat_no" type: object
Unique values: ['1', '10', '11', '12', '13', '13 15', '13 15 B', '14', '15', '15 16', '16', '2', '3',
There has been 27 boats in total.
486 persons got their boat which gives 18 passengers in one boat.
824 persons did not get their boat.
```

```
print('###############################################')
print('# Check if no shitty data regarding survivors #')
print('###############################################')
bodies_from_boats = ((df['boat_no'].notnull()) &
(df['body_no'].notnull())).sum()
print(f'Dead from boats: {bodies_from_boats}.')
bodies_despite_survived = ((df['survived'] == 1) &
(df['body_no'].notnull())).sum()
print(f'Survived despite body_no: {bodies_despite_survived}.')
```

```
###############################################
# Check if no shitty data regarding survivors #
###############################################
Dead from boats: 0.
Survived despite body_no: 0.
```

```python
print('############### COLUMN: BODY NO. ###############')
print()
print('Column "body_no" type:', df['body_no'].dtype)
print('Unique values:', np.sort(df['body_no'].unique()))
total_bodies = df['body_no'].count()
print(f'Bodies found in total: {total_bodies}')
```

```
############### COLUMN: BODY NO. ###############

Column "body_no" type: float64
Unique values: [  1.   4.   7.   9.  14.  15.  16.  17.  18.  19.  22.  32.  35.  37.
  38.  43.  45.  46.  47.  50.  51.  52.  53.  58.  61.  62.  67.  68.
  69.  70.  72.  75.  79.  80.  81.  89.  96.  97.  98. 101. 103. 108.
 109. 110. 119. 120. 121. 122. 124. 126. 130. 131. 133. 135. 142. 143.
 147. 148. 149. 153. 155. 156. 165. 166. 169. 171. 172. 173. 174. 175.
 176. 181. 187. 188. 189. 190. 196. 197. 201. 206. 207. 208. 209. 230.
 232. 234. 236. 245. 249. 255. 256. 258. 259. 260. 261. 263. 269. 271.
 275. 283. 284. 285. 286. 287. 292. 293. 294. 295. 297. 298. 299. 304.
 305. 306. 307. 309. 312. 314. 322. 327. 328.  nan]
Bodies found in total: 121
```

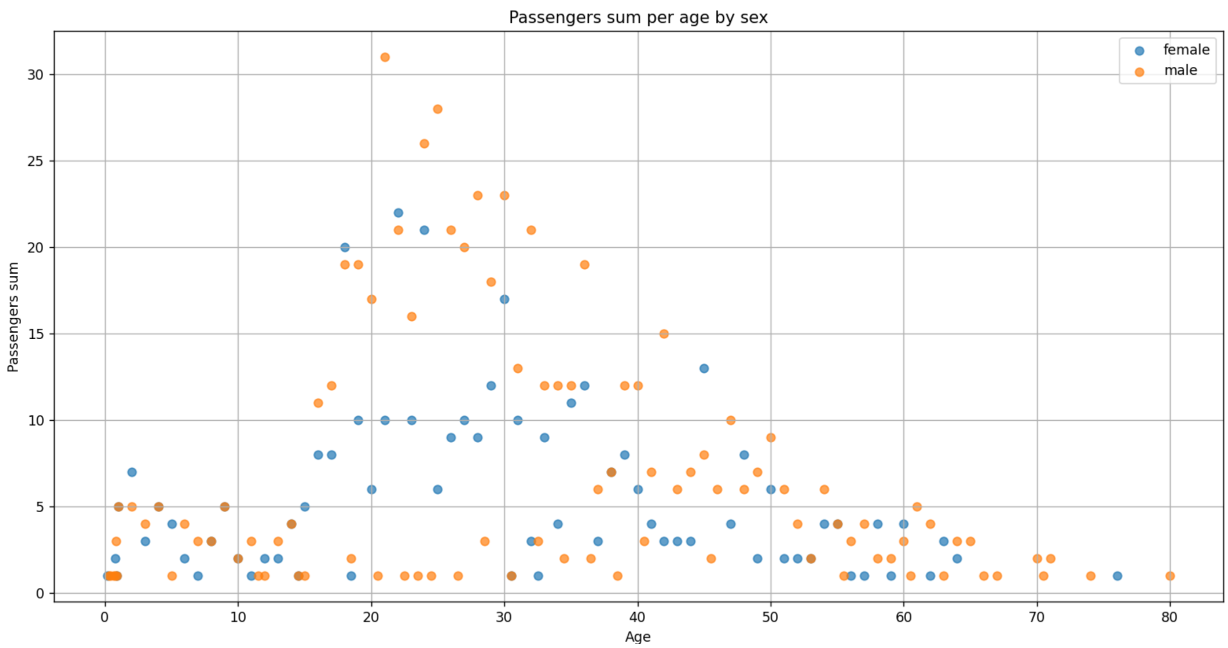
```python
age_by_sex = df.groupby(['sex'])['age'].mean()
print(f'Average age in each sex was:')
print(round(age_by_sex, 0))
print()
sex_by_age = df.groupby(['age', 'sex']).size()
print(f'Sum of passengers of each sex per age:')
print(sex_by_age)
```

```
Average age in each sex was:
sex
female    29.0
male      31.0
Name: age, dtype: float64

Sum of passengers of each sex per age:
age       sex
0.1667    female    1
0.3333    male      1
0.4167    male      1
0.6667    male      1
0.7500    female    2
          male      1
0.8333    male      3
0.9167    female    1
          male      1
1.0000    female    5
          male      5
2.0000    female    7
          male      5
3.0000    female    3
          male      4
4.0000    female    5
```

```
            male      1
6.0000      female    2
            male      4
7.0000      female    1
            male      3
8.0000      female    3
            male      3
9.0000      female    5
            male      5
10.0000     female    2
            male      2
11.0000     female    1
            male      3
11.5000     male      1
12.0000     female    2
            male      1
13.0000     female    2
            male      3
14.0000     female    4
            male      4
14.5000     female    1
            male      1
15.0000     female    5
            male      1
16.0000     female    8
            male     11
```

...itd.

```
# new data frame for scatter plot purpose
sex_by_age_df = df.groupby(['age', 'sex']).size().reset_index(name='count')
for gender in sex_by_age_df['sex'].unique():
    subset = sex_by_age_df[sex_by_age_df['sex'] == gender]
    plt.scatter(subset['age'], subset['count'], label=gender, alpha=0.7)

# scatter plot itself
plt.xlabel('Age')
plt.ylabel('Passengers sum')
plt.title('Passengers sum per age by sex')
plt.legend()
plt.grid(True)
plt.show()
```



6. Analiza innych czynników zbiorczych – nie wg eksplorowanych po kolei kolumn jak powyżej. Od teraz przeprowadzona analiza skupia się na wyciągnięciu różnorakich złożonych wniosków.

```
# the analysis starts here
print('##################################################')
print('############ PURE ANALYSIS STARTS HERE ############')
print('##################################################')
first_class_passengers = (df['class'] == 1).sum()
second_class_passengers = (df['class'] == 2).sum()
third_class_passengers = (df['class'] == 3).sum()
print(f'Passengers of 1st class: {first_class_passengers}, 2nd class:
{second_class_passengers}, 3rd class: {third_class_passengers}')
print()
```

```
##################################################
############ PURE ANALYSIS STARTS HERE ############
##################################################
Passengers of 1st class: 323, 2nd class: 277, 3rd class: 709
```

```
# 1st class surviving ratio analysis
first_class_survivors = ( (df['class'] == 1) & (df['survived'] == 1)
).sum()
print('1st class survivors', first_class_survivors)

first_class_non_survivors = ( (df['class'] == 1) & (df['survived'] == 0)
).sum()
print('1st class non-survivors', first_class_non_survivors)
surviving_ratio_first_class = round(first_class_survivors /
passengers_in_this_set * 100, 2)
print(f'Your chance to survive as the 1st class passenger was
{surviving_ratio_first_class}%')
print()
```

```
1st class survivors 200
1st class non-survivors 123
Your chance to survive as the 1st class passenger was 15.27%
```

```python
# 2nd class surviving ratio analysis
second_class_survivors = ( (df['class'] == 2) & (df['survived'] == 1)
).sum()
print('2nd class survivors', second_class_survivors)

second_class_non_survivors = ( (df['class'] == 2) & (df['survived'] == 0)
).sum()
print('2nd class non-survivors', second_class_non_survivors)
surviving_ratio_second_class = round(second_class_survivors /
passengers_in_this_set * 100, 2)
print(f'Your chance to survive as the 2nd class passenger was
{surviving_ratio_second_class}%')
print()
```

```
2nd class survivors 119
2nd class non-survivors 158
Your chance to survive as the 2nd class passenger was 9.08%
```

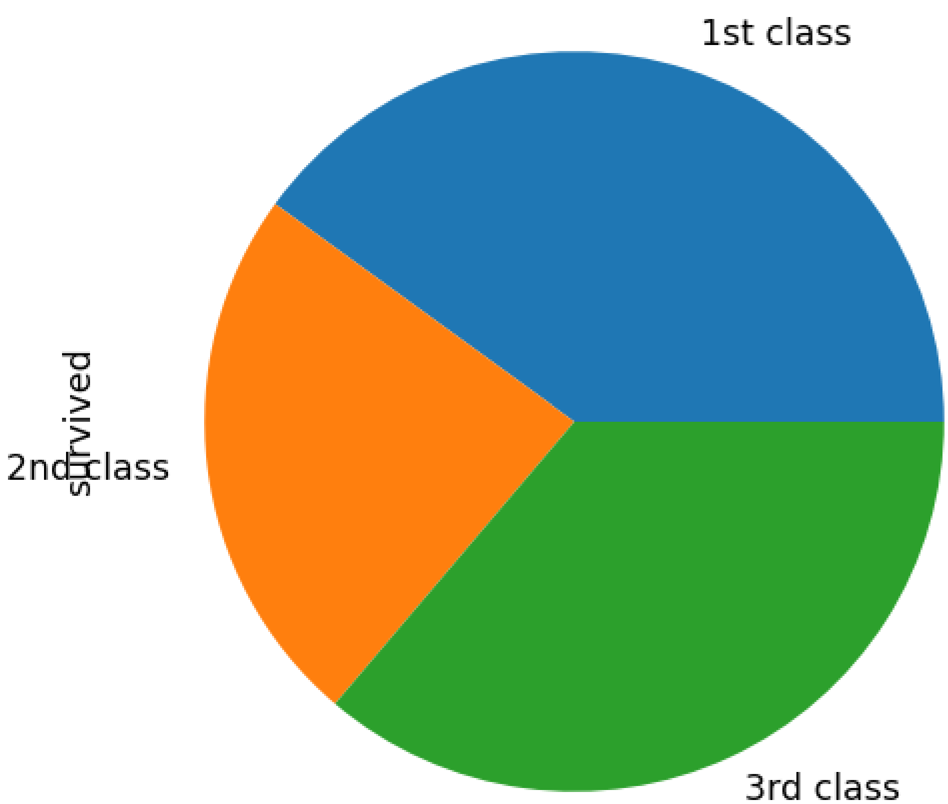
```python
# 3rd class surviving ratio analysis
third_class_survivors = ( (df['class'] == 3) & (df['survived'] == 1)
).sum()
print('3rd class survivors', third_class_survivors)

third_class_non_survivors = ( (df['class'] == 3) & (df['survived'] == 0)
).sum()
print('3rd class non-survivors', third_class_non_survivors)
surviving_ratio_third_class = round(third_class_survivors /
passengers_in_this_set * 100, 2)
print(f'Your chance to survive as the 3rd class passenger was
{surviving_ratio_third_class}%')
```

```
3rd class survivors 181
3rd class non-survivors 528
Your chance to survive as the 3rd class passenger was 13.82%
```

```python
# correlation
class_survived_corr = df[["class", "survived"]].corr()
print(f'Class vs. survived correlation is:')
print(class_survived_corr)
print()
df['sex_numeric'] = df['sex'].map({'male': 0, 'female': 1})
sex_survived_corr = df[["sex_numeric", "survived"]].corr()
print(f'Sex vs. survived correlation is:')
print(sex_survived_corr)
```

```
Class vs. survived correlation is:
             class  survived
class     1.000000 -0.312469
survived -0.312469  1.000000

Sex vs. survived correlation is:
             sex_numeric  survived
sex_numeric     1.000000  0.528693
survived        0.528693  1.000000
```

```
# new variables for new data frame for charts drawing purposes
class_labels = ['1st class', '2nd class', '3rd class']
survivors_by_class = [first_class_survivors, second_class_survivors,
third_class_survivors]
non_survivors_by_class = [first_class_non_survivors,
second_class_non_survivors, third_class_non_survivors]
chance_to_survive = [surviving_ratio_first_class,
surviving_ratio_second_class, surviving_ratio_third_class]

plot_df = pd.DataFrame({
    'class' : class_labels,
    'survived' : survivors_by_class,
    'not_survived' : non_survivors_by_class,
    'chance_to_survive' : chance_to_survive
})

plot_df.plot(kind="pie", y="survived", labels=plot_df["class"],
legend=False)
plt.show()

plot_df.plot(kind="bar", x="class", y=["chance_to_survive"])
plt.show()
```



```python
print('The cheapest tickets price (including zeros):')
print(
    df.groupby(['class'])['fare_price'].min()
)
print()
print('The most expensive tickets price:')
print(
    df.groupby(['class'])['fare_price'].max()
)
```

```
The cheapest tickets price (including zeros):
class
1.0    0.0
2.0    0.0
3.0    0.0
Name: fare_price, dtype: float64

The most expensive tickets price:
class
1.0    512.3292
2.0     73.5000
3.0     69.5500
Name: fare_price, dtype: float64
```

```python
# new data frame to replace 0 price with None
prices_not_zero_df = df.copy()
prices_not_zero_df.loc[(prices_not_zero_df['fare_price'] == 0)] = None
prices_not_zero_df = prices_not_zero_df.dropna(subset="fare_price")
print('New data frame with non-zeros for price:')
print(prices_not_zero_df.sample(15).to_string())
print()

cheapest_tickets_by_class =
prices_not_zero_df.groupby(['class'])['fare_price'].min()
print(
    'The cheapest tickets by class (excluding zeros): \n',
cheapest_tickets_by_class
)
print()

most_expensive_tickets_by_class =
prices_not_zero_df.groupby(['class'])['fare_price'].max()
print(
    'The most expensive tickets by class:\n',
most_expensive_tickets_by_class
)
```

```
The cheapest tickets by class (excluding zeros):
 class
1.0    5.0000
2.0    9.6875
3.0    3.1708
Name: fare_price, dtype: float64

The most expensive tickets by class:
 class
1.0    512.3292
2.0     73.5000
3.0     69.5500
Name: fare_price, dtype: float64
```

```python
# the cheapest vs. the most expensive tickets in 1st class
print(f'1st class cheapest ticket: {cheapest_tickets_by_class[1.0]}')
print(f'1st class most expensive ticket:
{most_expensive_tickets_by_class[1.0]}')
print()

# the cheapest vs. the most expensive tickets in 2n class
print(f'2nd class cheapest ticket: {cheapest_tickets_by_class[2.0]}')
print(f'2nd class most expensive ticket:
{most_expensive_tickets_by_class[2.0]}')
print()

# the cheapest vs. the most expensive tickets in 3rd class
print(f'3rd class cheapest ticket: {cheapest_tickets_by_class[3.0]}')
print(f'3rd class most expensive ticket:
{most_expensive_tickets_by_class[3.0]}')
```

```
1st class cheapest ticket: 5.0
1st class most expensive ticket: 512.3292

2nd class cheapest ticket: 9.6875
2nd class most expensive ticket: 73.5

3rd class cheapest ticket: 3.1708
3rd class most expensive ticket: 69.55
```

```python
# how many percent was the cheapest to the most expensive in the 1st class
cheapest_to_most_expensive_1st_class = (cheapest_tickets_by_class[1.0] /
most_expensive_tickets_by_class[1.0]) * 100
print(f'The 1st class cheapest ticket price was
{round(cheapest_to_most_expensive_1st_class, 2)}% of the most expensive
one.')

# how many percent was the cheapest to the most expensive in the 2nd class
cheapest_to_most_expensive_2nd_class = (cheapest_tickets_by_class[2.0] /
most_expensive_tickets_by_class[2.0]) * 100
print(f'The 2nd class cheapest ticket price was
{round(cheapest_to_most_expensive_2nd_class, 2)}% of the most expensive
one.')

# how many percent was the cheapest to the most expensive in the 3rd class
cheapest_to_most_expensive_3rd_class = (cheapest_tickets_by_class[3.0] /
most_expensive_tickets_by_class[3.0]) * 100
print(f'The 3rd class cheapest ticket price was
{round(cheapest_to_most_expensive_3rd_class, 2)}% of the most expensive
one.')
```

```
The 1st class cheapest ticket price was 0.98% of the most expensive one.
The 2nd class cheapest ticket price was 13.18% of the most expensive one.
The 3rd class cheapest ticket price was 4.56% of the most expensive one.

The 1st class most expensive ticket was 102.47 times more expensive than the cheapest one.
The 2nd class most expensive ticket was 7.59 times more expensive than the cheapest one.
The 3rd class most expensive ticket was 21.93 times more expensive than the cheapest one.
```

```python
# how many times was the most expensive ticket more expensive than the cheapest in the 1st class
most_expensive_to_cheapest_1st_class = most_expensive_tickets_by_class[1.0] / cheapest_tickets_by_class[1.0]
print(f'The 1st class most expensive ticket was {round(most_expensive_to_cheapest_1st_class, 2)} times more expensive than the cheapest one.')

# how many times was the most expensive ticket more expensive than the cheapest in the 2nd class
most_expensive_to_cheapest_2nd_class = most_expensive_tickets_by_class[2.0] / cheapest_tickets_by_class[2.0]
print(f'The 2nd class most expensive ticket was {round(most_expensive_to_cheapest_2nd_class, 2)} times more expensive than the cheapest one.')

# how many times was the most expensive ticket more expensive than the cheapest in the 3rd class
most_expensive_to_cheapest_3rd_class = most_expensive_tickets_by_class[3.0] / cheapest_tickets_by_class[3.0]
print(f'The 3rd class most expensive ticket was {round(most_expensive_to_cheapest_3rd_class, 2)} times more expensive than the cheapest one.')
```

```
The 1st class most expensive ticket was 102.47 times more expensive than the cheapest one.
The 2nd class most expensive ticket was 7.59 times more expensive than the cheapest one.
The 3rd class most expensive ticket was 21.93 times more expensive than the cheapest one.
```

```python
# each column from among of 'survived', 'siblings/spouse', and
'parents/children' has nulls so data is not consistent - a new data frame
is needed
print('Problematic columns:')
print(df['survived'].isnull().sum(), 'rows is not a number for column
"survived"')
print(df['siblings/spouse'].isnull().sum(), 'rows is not a number for
column "siblings/spouse"')
print(df['parents/children'].isnull().sum(), 'rows is not a number for
column "parents/children"')
print()

# new data frame created because of the above:
not_null_passengers_df = df.copy()
not_null_passengers_df =
not_null_passengers_df.dropna(subset=['siblings/spouse',
'parents/children'])

# confirm that the data is consistent around the columns of interest
print('Because of the above, the following data frame has been created to
exclude missing data:')
print(not_null_passengers_df['survived'].isnull().sum(), 'rows is not a
number for column "survived"')
print(not_null_passengers_df['siblings/spouse'].isnull().sum(), 'rows is
not a number for column "siblings/spouse"')
print(not_null_passengers_df['parents/children'].isnull().sum(), 'rows is
not a number for column "parents/children"')
```

```
Problematic columns:
1 rows is not a number for column "survived"
1 rows is not a number for column "siblings/spouse"
1 rows is not a number for column "parents/children"

Because of the above, the following data frame has been created to exclude missing data:
0 rows is not a number for column "survived"
0 rows is not a number for column "siblings/spouse"
0 rows is not a number for column "parents/children"
```

```python
# conduct data mining
print('############################################################')
print('# CHANCES TO SURVIVE - ALONE vs. WITH ADULTS vs. WITH KIDS #')
print('############################################################')
traveling_alone = ((not_null_passengers_df['siblings/spouse'] == 0) &
(not_null_passengers_df['parents/children'] == 0)).sum()
survivors_traveling_alone = ((not_null_passengers_df['survived'] == 1) &
(not_null_passengers_df['siblings/spouse'] == 0) &
(not_null_passengers_df['parents/children'] == 0)).sum()
non_survivors_traveling_alone = ((not_null_passengers_df['survived'] == 0)
& (not_null_passengers_df['siblings/spouse'] == 0) &
(not_null_passengers_df['parents/children'] == 0)).sum()
chance_to_survive_traveling_alone = survivors_traveling_alone /
traveling_alone * 100
print(f'{traveling_alone} passengers traveled alone -
{survivors_traveling_alone} survived while {non_survivors_traveling_alone}
did not survive so traveling alone your chance to survive was
{round(chance_to_survive_traveling_alone, 2)}%.')

traveling_with_siblings_spouse =
(not_null_passengers_df['siblings/spouse'] != 0).sum()
survivors_traveling_with_siblings_spouse =
((not_null_passengers_df['survived'] == 1) &
(not_null_passengers_df['siblings/spouse'] != 0)).sum()
non_survivors_traveling_with_siblings_spouse =
((not_null_passengers_df['survived'] == 0) &
(not_null_passengers_df['siblings/spouse'] != 0)).sum()
chance_to_survive_traveling_with_siblings_spouse =
survivors_traveling_with_siblings_spouse / traveling_with_siblings_spouse
* 100
print(f'{traveling_with_siblings_spouse} passengers traveled with siblings
or spouse - {survivors_traveling_with_siblings_spouse} survived while
{non_survivors_traveling_with_siblings_spouse} did not survive so
traveling with siblings or spouse your chance to survive was
{round(chance_to_survive_traveling_with_siblings_spouse, 2)}%')

traveling_with_parents_children =
(not_null_passengers_df['parents/children'] != 0).sum()
survivors_traveling_with_parents_children =
((not_null_passengers_df['survived'] == 1) &
(not_null_passengers_df['parents/children'] != 0) ).sum()
non_survivors_traveling_with_parents_children =
((not_null_passengers_df['survived'] == 0) &
(not_null_passengers_df['parents/children'] != 0)).sum()
chance_to_survive_traveling_with_parents_children =
survivors_traveling_with_parents_children /
traveling_with_parents_children * 100
print(f'{traveling_with_parents_children} passengers traveled with parents
or children - {survivors_traveling_with_parents_children} survived while
{non_survivors_traveling_with_parents_children} did not survive so
traveling with parents or children your chance to survive was
{round(chance_to_survive_traveling_with_parents_children, 2)}%')
```

```
############################################################
# CHANCES TO SURVIVE - ALONE vs. WITH ADULTS vs. WITH KIDS #
############################################################
790 passengers traveled alone - 239 survived while 551 did not survive so traveling alone your chance to survive was 30.25%.
418 passengers traveled with siblings or spouse - 191 survived while 227 did not survive so traveling with siblings or spouse your chance to survive was 45.69%
307 passengers traveled with parents or children - 164 survived while 143 did not survive so traveling with parents or children your chance to survive was 53.42%
```

```python
# the youngest and the oldest survivors and non-survivors
print('################################################')
print('# YOUNGEST AND OLDEST SURVIVORS AND NON-SURVIVORS #')
print('################################################')
youngest_survived = df[df['survived'] == 1]['age'].min()
print(f'The youngest survivor was {round(youngest_survived, 2)} years
old.')

oldest_survived = df[df['survived'] == 1]['age'].max()
print(f'The oldest survivor was {round(oldest_survived, 2)} years old.')

average_survivor_age = df[df['survived'] == 1]['age'].mean()
print(f'Survivor average age was {round(average_survivor_age, 2)} years
old.')

median_survivor_age = df[df['survived'] == 1]['age'].median()
print(f'Median of the survivor age was {round(median_survivor_age, 2)}.')

print()

youngest_non_survivor = df[df['survived'] == 0]['age'].min()
print(f'The youngest non-survivor was {round(youngest_non_survivor, 2)}
years old.')

oldest_non_survivor = df[df['survived'] == 0]['age'].max()
print(f'The oldest non-survivor was {round(oldest_non_survivor, 2)} years
old.')

average_non_survivor_age = df[df['survived'] == 0]['age'].mean()
print(f'Non-survivors average age was {round(average_non_survivor_age, 2)}
years old.')

median_non_survivor_age = df[df['survived'] == 0]['age'].median()
print(f'Median of non-survivors age was {round(median_non_survivor_age,
2)}.')
```

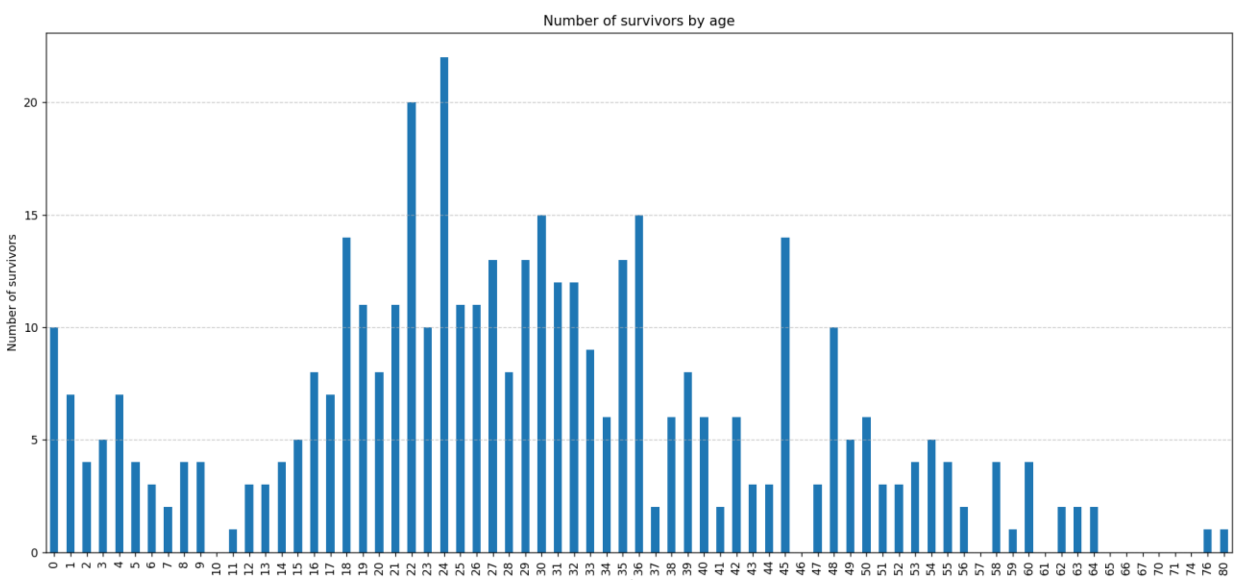
```
###################################################
# YOUNGEST AND OLDEST SURVIVORS AND NON-SURVIVORS #
###################################################
The youngest survivor was 0.17 years old.
The oldest survivor was 80.0 years old.
Survivor average age was 28.92 years old.
Median of the survivor age was 28.0.

The youngest non-survivor was 0.33 years old.
The oldest non-survivor was 74.0 years old.
Non-survivors average age was 30.55 years old.
Median of non-survivors age was 28.0.
```

```
print('##################################################')
print('####### AGE & SUM OF SURVIVORS AT EACH AGE #######')
print('##################################################')
df_clean = df[['age', 'survived']].dropna()
df_clean['age'] = df_clean['age'].astype(int)
survivors_by_age = df_clean.groupby('age')['survived'].sum().astype(int)
print(survivors_by_age)

plt.figure(figsize=(12, 6))
survivors_by_age.plot(kind='bar')

plt.title('Number of survivors by age')
plt.xlabel('Age')
plt.ylabel('Number of survivors')
plt.xticks(rotation=90)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```

```
##################################################
####### AGE & SUM OF SURVIVORS AT EACH AGE #######
##################################################
age
0     10
1      7
2      4
3      5
4      7
5      4
6      3
7      2
8      4
9      4
10     0
11     1
12     3
13     3
14     4
15     5
16     8
17     7
18    14
```

```
21    11
22    20
23    10
24    22
25    11
26    11
27    13
28     8
29    13
30    15
31    12
32    12
33     9
34     6
35    13
36    15
37     2
38     6
39     8
40     6
41     2
42     6
43     3
44     3
45    14
```

...itd.



```python
print('#################################################')
print('############# % OF SURVIVORS PER SEX #############')
print('#################################################')
women_traveled = df[df['sex'] == 'female'].shape[0]
print(f'{int(women_traveled)} women traveled.')
women_survied = df[df['sex'] == 'female']['survived'].sum()
print(f'{int(women_survied)} women survived.')
print()

men_traveled = df[df['sex'] == 'male'].shape[0]
print(f'{int(men_traveled)} men traveled.')
men_survied = df[df['sex'] == 'male']['survived'].sum()
print(f'{int(men_survied)} men survived.')
print()

women_chance_to_survive = women_survied / women_traveled * 100
print(f'Women had {round(women_chance_to_survive, 2)}% chance to survive.')
print()

men_chance_to_survive = men_survied / men_traveled * 100
print(f'Men had {round(men_chance_to_survive, 2)}% chance to survive.')
```

```
#################################################
############# % OF SURVIVORS PER SEX #############
#################################################
466 women traveled.
339 women survived.

843 men traveled.
161 men survived.

Women had 72.75% chance to survive.

Men had 19.1% chance to survive.
```

Wnioski:

1. Z dostępnych danych można wyliczyć, że obsługa statku wymagała poniesienia kosztów w wysokości co najmniej 43 550 dolarów (?) ówcześnie na sprzedaży samych tylko biletów wstępu na pokład. Nie są tu uwzględnione zyski z promocji materiałów merchandisingowych jak figurki-zabawki statków, proporczyki, naklejki itp.
2. Ilość łodzi ratunkowych była niewystarczająca.
3. Do brzegu nie dotarły osoby martwe co może sugerować, że jeśli ktokolwiek zmarł z wyziębienia pomimo uprzedniego wciągnięcia na łódź, został ostatecznie (niekoniecznie natychmiast) wrzucony z powrotem do wody.
4. Przypisania osób ocalałych do klas sprzedanych biletów a także do portów wejścia na pokład stanowią ciekawostkę analityczną – nie stanowiły raczej o fakcie przeżycia lub nieprzeżycia. Relacje osób ocalałych z innej katastrofy morskiej – zatonięcia promu Estonia – wskazują, że to gdzie akurat komu udało się znaleźć w konkretnych momentach zalania poszczególnych partii pokładu, miało nieporównanie większe znaczenie. Pasażerowie świętujący wyjątkową podróż, mogący znajdować się pod wpływem alkoholu, a dodatkowo niedowierzający temu co się dzieje, mogli mieć zupełnie inne postrzeganie powagi wydarzenia.
5. Największe znaczenie dla faktu ocalenia z katastrofy Titanica wydaje się mieć płeć – kobiety/mężczyźni zidentyfikowani wśród podróżujących: 466/843. Prawie 2x tyle mężczyzn co kobiet. Natomiast kobiety/mężczyźni zidentyfikowani wśród ocalałych: 339/161 – 72% kobiet przeżyło, podczas gdy mężczyzn przeżyło zaledwie 19%.
6. Drugim istotnym czynnikiem był fakt podróżowania z kimś – analiza pokazuje, że aż 53% szans na przeżycie miały osoby współpodróżujące z osobami w relacji rodzic-dziecko. Osoby współpodróżujące w relacji dorosły-dorosły, np. rodzeństwa lub małżeństwa, miały 45-procentową szansę na ujście z życiem. Osoby podróżujące samotnie miały 30% szansy na przeżycie. Może to wskazywać na siłę determinacji do opieki nad drugą bliską nam osobą, ale na pewno jest też wypadkową reguł pierwszeństwa dostępu do łodzi ratunkowych.

Zadanie bardzo pouczające, a wcale niewyczerpujące możliwości poznanych metod pracy na danych – zachęca do powrotu.

Całe repo dostępne tutaj: https://github.com/racibornio/Python-lessons/blob/master/akademia/zadania/mod_4/zad_2/mod_4_zad_2.py