Data Cleaning

In [1]:
```
# lecture imports / dependencies
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import pandas as pd
import seaborn as sns
sns.set(style="ticks")
from sklearn.feature_extraction.text import CountVectorizer
from skimage.io import imread, imshow
```

Typical steps of ML/DM

1. Identify question / task
2. Collect data
3. Clean and preprocess data
4. Exploratory data anlysis (EDA)
5. Feature and model selection
6. Train model
7. Evaluate and communicate results
8. Deploy working system

(but not necessarily in this order...)

Today we'll discuss steps (3) and (4)

What does data look like?

Often, it is tabular (but certainly not always!).

In [2]:
```
titanic = pd.read_csv("titantic_train.csv")
titanic.head()
```

Out[2]:

	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 [3]:
```
titanic.dtypes
```

Out[3]:
```
PassengerId      int64
Survived         int64
Pclass           int64
Name            object
Sex             object
Age            float64
SibSp            int64
Parch            int64
Ticket          object
Fare           float64
Cabin           object
Embarked        object
dtype: object
```

In [4]:
```
titanic.describe()
```

Out[4]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

In [5]:
```
titanic["Sex"].describe()
```

Out[5]:
```
count      891
unique       2
top       male
freq       577
Name: Sex, dtype: object
```

We can get a summary of the categorical variables by passing only those columns to describe()

In [6]:
```
categorical=titanic.dtypes[(titanic.dtypes == "object")].index
print(categorical)
```

```
Index(['Name', 'Sex', 'Ticket', 'Cabin', 'Embarked'], dtype='object')
```

In [8]:
```
titanic[categorical].describe()
```

Out[8]:

	Name	Sex	Ticket	Cabin	Embarked
count	891	891	891	204	889
unique	891	2	681	147	3
top	Becker, Miss. Marion Louise	male	347082	G6	S
freq	1	577	7	4	644

```
The categorical variable summary shows
1) the count of non-NaN records,
2) the number of unique categories,
3) the most frequently occurring value
4) the number of occurrences of the most frequent value.
```

After looking at the data for the first time, you should ask yourself a few questions:

```
1) Do I need all of the variables?
2) Should I transform any variables?
3) Are there NA values, outliers or other strange values?
4) Should I create new variables?
```

1) Do I Need All of the Variables?

In [56]:
```
Getting rid of unnecessary variables is a good first step when dealing with any data set,
since dropping variables reduces complexity and can make computation on the data faster.

it can still be helpful to drop variables that will only distract from your goal.

"PassengerId" is just a number assigned to each passenger. It is nothing more than an
arbitrary identifier; we could keep it for identification purposes, but let's remove it anyway:
```

In [9]:
```
del titanic["PassengerId"]
```

We have 3 more features to consider: Name, Ticket and Cabin.

In [10]:
```
sorted(titanic["Name"])[0:8]   # Check the first 15 sorted names
```

Out[10]:
```
['Abbing, Mr. Anthony',
 'Abbott, Mr. Rossmore Edward',
 'Abbott, Mrs. Stanton (Rosa Hunt)',
 'Abelson, Mr. Samuel',
 'Abelson, Mrs. Samuel (Hannah Wizosky)',
 'Adahl, Mr. Mauritz Nils Martin',
 'Adams, Mr. John',
 'Ahlin, Mrs. Johan (Johanna Persdotter Larsson)']
```

```
we see that the Name variable has 891 unique values,
In general, a categorical variable that is unique to each case isn't useful for prediction.
From the perspective of predictive modeling, the Name variable could be removed.
```

In [11]:
```
del titanic["Name"]
del titanic["Ticket"]
```

```
Data projects are iterative processes: you can start with a simple analysis or model using only
a few variables and then expand later by adding more and more of the other variables you initially
ignored or removed.
```

2) Should I Transform Any Variables?

In [12]:
```
titanic.groupby("Survived").size()
```

Out[12]:
```
Survived
0    549
1    342
dtype: int64
```

In [35]:
```
new_survived = pd.Categorical(titanic["Survived"])
new_survived = new_survived.rename_categories(["Died","Survived"])
titanic["newSurvived"]=new_survived
```

In [58]:
```
new_survived.describe()
```

Out[58]:

	counts	freqs
categories		
Died	549	0.616162
Survived	342	0.383838

In [38]:
```
titanic.head()
```

Out[38]:

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Cabin	Embarked	newSurvived
0	1	0	3	male	22.0	1	0	7.2500	NaN	S	Died
1	2	1	1	female	38.0	1	0	71.2833	C85	C	Survived
2	3	1	3	female	26.0	0	0	7.9250	NaN	S	Survived
3	4	1	1	female	35.0	1	0	53.1000	C123	S	Survived
4	5	0	3	male	35.0	0	0	8.0500	NaN	S	Died

```
Now it's time to revisit the Cabin variable. It appears that each Cabin is in a general section
```

of the ship indicated by the capital letter at the start of each factor level:

In [48]:
```
titanic["Cabin"].unique()
```

Out[48]:
```
array([nan, 'C85', 'C123', 'E46', 'G6', 'C103', 'D56', 'A6',
       'C23 C25 C27', 'B78', 'D33', 'B30', 'C52', 'B28', 'C83', 'F33',
       'F G73', 'E31', 'A5', 'D10 D12', 'D26', 'C110', 'B58 B60', 'E101',
       'F E69', 'D47', 'B86', 'F2', 'C2', 'E33', 'B19', 'A7', 'C49', 'F4',
       'A32', 'B4', 'B80', 'A31', 'D36', 'D15', 'C93', 'C78', 'D35',
       'C87', 'B77', 'E67', 'B94', 'C125', 'C99', 'C118', 'D7', 'A19',
       'B49', 'D', 'C22 C26', 'C106', 'C65', 'E36', 'C54',
       'B57 B59 B63 B66', 'C7', 'E34', 'C32', 'B18', 'C124', 'C91', 'E40',
       'T', 'C128', 'D37', 'B35', 'E50', 'C82', 'B96 B98', 'E10', 'E44',
       'A34', 'C104', 'C111', 'C92', 'E38', 'D21', 'E12', 'E63', 'A14',
       'B37', 'C30', 'D20', 'B79', 'E25', 'D46', 'B73', 'C95', 'B38',
       'B39', 'B22', 'C86', 'C70', 'A16', 'C101', 'C68', 'A10', 'E68',
       'B41', 'A20', 'D19', 'D50', 'D9', 'A23', 'B50', 'A26', 'D48',
       'E58', 'C126', 'B71', 'B51 B53 B55', 'D49', 'B5', 'B20', 'F G63',
       'C62 C64', 'E24', 'C90', 'C45', 'E8', 'B101', 'D45', 'C46', 'D30',
       'E121', 'D11', 'E77', 'F38', 'B3', 'D6', 'B82 B84', 'D17', 'A36',
       'B102', 'B69', 'E49', 'C47', 'D28', 'E17', 'A24', 'C50', 'B42',
       'C148'], dtype=object)
```

If we grouped cabin just by this letter, we could reduce the number of levels while potentially extracting some useful information.

In [51]:
```
char_cabin = titanic["Cabin"].astype(str) # Convert data to str
new_Cabin = np.array([cabin[0] for cabin in char_cabin]) # Take first letter
print(new_Cabin)
new_Cabin = pd.Categorical(new_Cabin)
new_Cabin .describe()
```

```
 'n' 'n' 'n' 'B' 'n' 'n' 'n' 'B' 'n' 'D' 'n' 'n' 'n' 'n' 'n' 'n' 'E' 'n'
 'n' 'n' 'F' 'n' 'n' 'B' 'n' 'B' 'D' 'n' 'n' 'n' 'n' 'n' 'n' 'B' 'n' 'n'
 'n' 'n' 'n' 'n' 'D' 'n' 'n' 'n' 'n' 'n' 'B' 'n' 'n' 'n' 'A' 'n' 'n' 'E'
 'n' 'n' 'n' 'n' 'n' 'B' 'n' 'n' 'n' 'n' 'B' 'n' 'n' 'E' 'n' 'n' 'n' 'n'
 'n' 'B' 'n' 'n' 'n' 'n' 'n' 'E' 'n' 'n' 'n' 'C' 'n' 'n' 'n' 'n' 'n' 'n'
 'n' 'n' 'n' 'C' 'n' 'n' 'n' 'D' 'n' 'n' 'n' 'E' 'n' 'n' 'n' 'n' 'D' 'n'
 'n' 'n' 'n' 'A' 'n' 'n' 'n' 'D' 'B' 'n' 'n' 'n' 'n' 'n' 'n' 'C' 'n' 'n'
 'n' 'n' 'n' 'n' 'n' 'B' 'n' 'C' 'n']
```

Out[51]:

categories	counts	freqs
A	15	0.016835
B	47	0.052750
C	59	0.066218
D	33	0.037037
E	32	0.035915

Since there are so many missing values, the Cabin variable might be devoid of useful information for prediction. On the other hand, a missing cabin variable could be an indication that a passenger died: after all, how would we know what cabin a passenger stayed in if they weren't around to tell the tale?

Let's keep the new cabin variable:

In [54]:
```
titanic["new_cabin"]=new_Cabin
titanic.head(7)
```

Out[54]:

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Cabin	Embarked	newSurvived	new_cabin
0	1	0	3	male	22.0	1	0	7.2500	NaN	S	Died	n
1	2	1	1	female	38.0	1	0	71.2833	C85	C	Survived	C
2	3	1	3	female	26.0	0	0	7.9250	NaN	S	Survived	n
3	4	1	1	female	35.0	1	0	53.1000	C123	S	Survived	C
4	5	0	3	male	35.0	0	0	8.0500	NaN	S	Died	n
5	6	0	3	male	NaN	0	0	8.4583	NaN	Q	Died	n
6	7	0	1	male	54.0	0	0	51.8625	E46	S	Died	E

3) Are there NA Values, Outliers or Other Strange Values?

Data sets are often littered with missing data, extreme data points called outliers and other strange values. Missing values, outliers and strange values can negatively affect statistical tests and models and may even cause certain functions to fail.

In Python, you can detect missing values with the pd.isnull() function:

In [ ]:
```
1) In cases where you have a lot of data and only a few missing values, it might make
sense to simply delete records with missing values present.

2) Missing values in categorical data are not particularly troubling because you can simply
treat NA as an additional category

3) Missing values in numeric variables are more troublesome, since you can't just treat
a missing value as number.
```

In [16]:
```
dummy_vector = pd.Series([1,None,3,None,7,8])
dummy_vector.isnull()
```

Out[16]:
```
0    False
1     True
2    False
3     True
4    False
5    False
dtype: bool
```

We can get the row indexes of the missing values with np.where():

In [18]:
```
missing_rows_age=np.where(titanic["Age"].isnull())
print(missing_rows_age[0])
print("\n Number_of_missing_values= ",len(missing_rows_age[0]))
```

```
[  5  17  19  26  28  29  31  32  36  42  45  46  47  48  55  64  65  76
  77  82  87  95 101 107 109 121 126 128 140 154 158 159 166 168 176 180
 181 185 186 196 198 201 214 223 229 235 240 241 250 256 260 264 270 274
 277 284 295 298 300 301 303 304 306 324 330 334 335 347 351 354 358 359
 364 367 368 375 384 388 409 410 411 413 415 420 425 428 431 444 451 454
 457 459 464 466 468 470 475 481 485 490 495 497 502 507 511 517 522 524
 527 531 533 538 547 552 557 560 563 564 568 573 578 584 589 593 596 598
 601 602 611 612 613 629 633 639 643 648 650 653 656 667 669 674 680 692
 697 709 711 718 727 732 738 739 740 760 766 768 773 776 778 783 790 792
 793 815 825 826 828 832 837 839 846 849 859 863 868 878 888]

 Number_of_missing_values=  177
```

In [22]:
```
titanic.iloc[[5,17,19,26,28]]
```

Out[22]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
5	6	0	3	Moran, Mr. James	male	NaN	0	0	330877	8.4583	NaN	Q
17	18	1	2	Williams, Mr. Charles Eugene	male	NaN	0	0	244373	13.0000	NaN	S
19	20	1	3	Masselmani, Mrs. Fatima	female	NaN	0	0	2649	7.2250	NaN	C
26	27	0	3	Emir, Mr. Farred Chehab	male	NaN	0	0	2631	7.2250	NaN	C
28	29	1	3	O'Dwyer, Miss. Ellen "Nellie"	female	NaN	0	0	330959	7.8792	NaN	Q

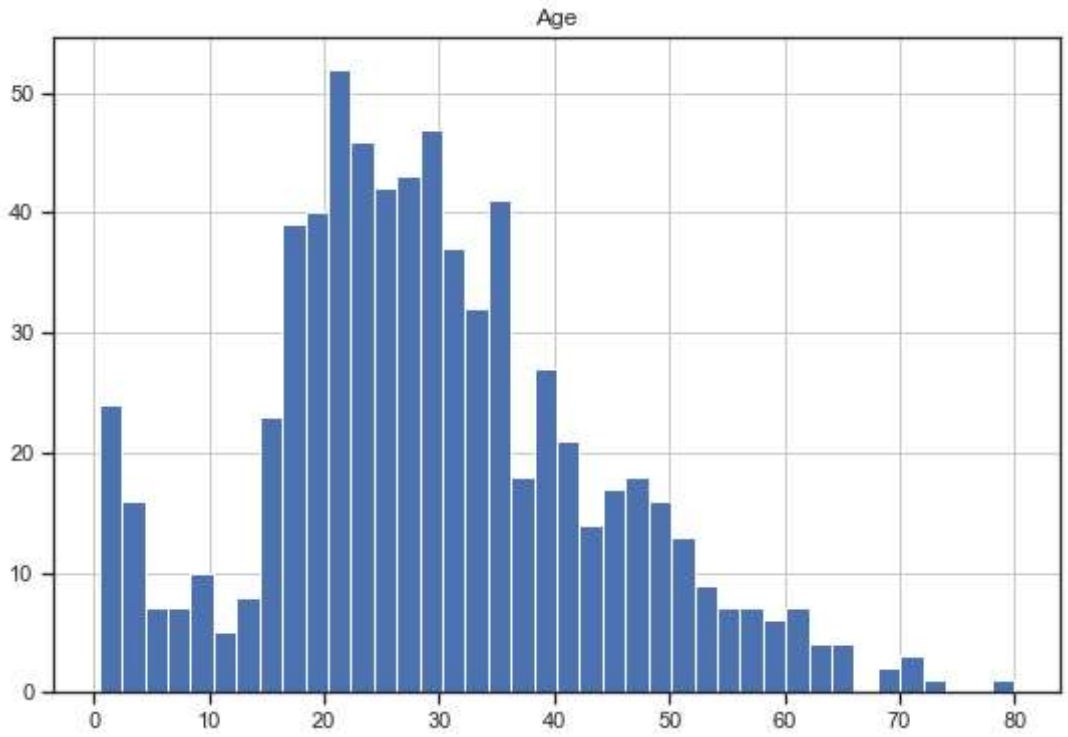
```
 With 177 missing values it's probably not a good idea to throw all those records away.
Here are a few ways we could deal with them:

1) Replace the null values with 0s
2) Replace the null values with some central value like the mean or median
3) Impute some other value
4) Split the data set into two parts: one set with where records have an Age value
   and another set where age is null.
```

Setting missing values in numeric data to zero makes sense in some cases, but it doesn't make any sense here because a person's age can't be zero. Setting all ages to some central number like the median is a simple fix but there's no telling whether such a central number is a reasonable estimate of age without looking at the distribution of ages. For all we know each age is equally common. We can quickly get a sense of the distribution of ages by creating a histogram of the age variable with df.hist():

In [29]:
```
titanic.hist(column='Age',     # Column to plot
                   figsize=(9,6),   # Plot size
                   bins=40)         # Number of histogram bins
```

Out[29]:
```
array([[<AxesSubplot:title={'center':'Age'}>]], dtype=object)
```



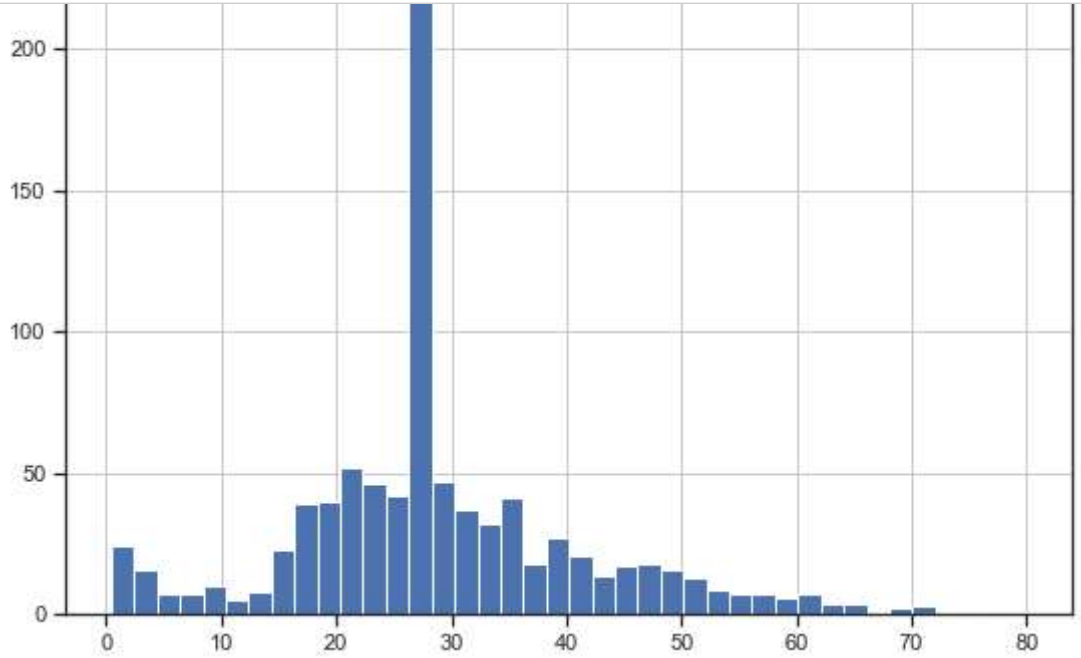
From the histogram, we see that ages between 20 and 30 are the most common,
so filling in missing values with a central number like the mean or median
wouldn't be entirely unreasonable. Let's fill in the missing values with the median value of 28

In [36]:
```
new_age_var = np.where(titanic["Age"].isnull(), # Logical check
                       28,                      # Value if check is true
                       titanic["Age"])     # Value if check is false
titanic["Age"] = new_age_var
titanic["Age"].describe()
```

Out[36]:
```
count    891.000000
mean      29.361582
std       13.019697
min        0.420000
25%       22.000000
50%       28.000000
75%       35.000000
max       80.000000
Name: Age, dtype: float64
```

Since we just added a bunch of 28s to age, let's look at the histogram again for a sanity check.
The bar representing 28 to be much taller this time.

In [37]:
```
titanic.hist(column='Age',     # Column to plot
                   figsize=(9,6),   # Plot size
                   bins=40)         # Number of histogram bins
```



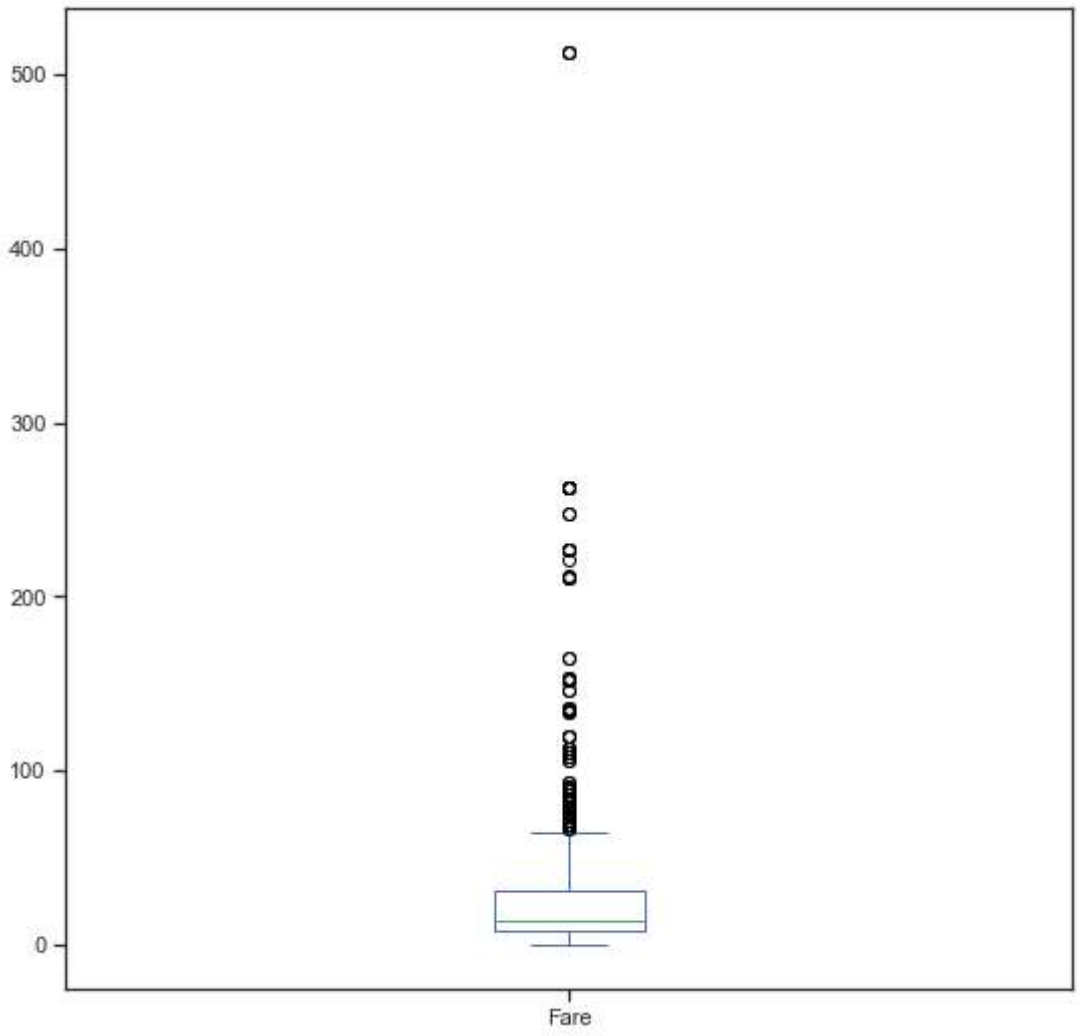
imputing the missing data (estimating age based on other variables) might have been a better option, but we'll stick with this for now.

OUtliers

Next, let's consider outliers. Outliers are extreme numerical values: values that lie far away from the typical values a variable takes on. Creating plots is one of the quickest ways to detect outliers.

In [38]:
```
titanic["Fare"].plot(kind="box",figsize=(9,9))
```

Out[38]: <AxesSubplot:>




```
In a boxplot, the central box represents 50% of the data and the central bar represents the median.
The dotted lines with bars on the ends are "whiskers" which encompass the great majority of the data
and points beyond the whiskers indicate uncommon values. In this case, we have some uncommon values
that are so far away from the typical value that the box appears squashed in the plot: this is
    a clear indication of outliers. Indeed, it looks like one passenger paid almost twice as
```

much as any other passenger. Even the passengers that paid between 200 and 300 are far higher than the vast majority of the other passengers.

In [ ]: