data_manipulation_eda

December 19, 2020

1 Workout Notebook for Data Manipulation and Exploratory Data Analysis (EDA)

2 NumPy (Numerical Python)

2.1 Why Numpy?

- Useful for scientific calculations.
- Work with high performance on arrays and matrices (fixed type array)

Python Lists vs. Numpy Arrays - What is the difference?

Numpy is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays. A numpy array is a grid of values, all of the same type, and is indexed by a tuple of nonnegative integers. The number of dimensions is the rank of the array; the shape of an array is a tuple of integers giving the size of the array along each dimension.

The Python core library provided Lists. A list is the Python equivalent of an array, but is resizeable and can contain elements of different types.

A common beginner question is what is the real difference here. The answer is performance. Numpy data structures perform better in:

- Size - Numpy data structures take up less space
- Performance - they have a need for speed and are faster than lists
- Functionality - SciPy and NumPy have optimized functions such as linear algebra operations built in.

check for more information: https://webcourses.ucf.edu/courses/1249560/pages/python-lists-vs-numpy-arrays-what-is-the-difference#:~:text=It%20provides%20a%20high-performance,a%20tuple%20of%20nonnegative%20integers.&text=A%20list%20is%20the%20Python,contain%20el

[1]:
```
a = [1, 2, 3, 4]
b = [2, 3, 4, 5]

ab = []

for i in range(0, len(a)):
    ab.append(a[i] * b[i])
```

```
ab
```

[1]: [2, 6, 12, 20]

[2]:
```python
import numpy as np

a = np.array([1, 2, 3, 4])
b = np.array([2, 3, 4, 5])

a * b
```

[2]: array([2, 6, 12, 20])

[3]:
```python
type(np.array([1, 2, 3, 4, 5]))
```

[3]: numpy.ndarray

[4]:
```python
np.zeros(10, dtype = int) # zero vector
```

[4]: array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0])

[5]:
```python
# Generate random integers with randint method
np.random.randint(0, 10, size=10) # generate 10 random integers between 0 and 10
```

[5]: array([9, 9, 1, 7, 3, 5, 2, 7, 6, 8])

[6]:
```python
np.random.randint(0, 10, (3,3)) # generate 3 by 3 matrix which comprises␣
↪integers between 0 and 10
```

[6]:
```
array([[7, 5, 3],
       [8, 2, 0],
       [3, 1, 2]])
```

[7]:
```python
# Generate random gaussian floats
a = np.random.normal(10, 4, (4, 4)) # 4 by 4 normally distributed matrix␣
↪generation with a mean value of 10 and a standard deviation of 4
a
```

[7]:
```
array([[ 6.09182113,  8.52890084,  7.43722147,  8.86820461],
       [12.18525151,  6.45558411,  5.08363664,  7.85293087],
       [ 7.61056207,  9.02932214, 19.01405942, 13.27836931],
       [ 2.79120363, 10.10023343,  7.11253523, 14.99329963]])
```

[8]:
```python
print(np.mean(a))
print(np.std(a))
```

```
9.152071003041979
3.9035375162817134
```

[9]:
```python
a.ndim # number of dimensions
```

[9]: 2

[10]:
```python
a.shape # dimension info
```

[10]: (4, 4)

[11]:
```python
a.size # number of elements
```

[11]: 16

[12]:
```python
b.dtype # array data type
```

[12]: dtype('int32')

2.2 Reshaping

[13]:
```python
b = np.arange(1,10)
b
```

[13]: array([1, 2, 3, 4, 5, 6, 7, 8, 9])

[14]:
```python
b.reshape((3, 3))
```

[14]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

2.3 Index Operations

[15]:
```python
# Index operations are more or less the same with lists with a few differences
m = np.random.randint(10, size = (3,5))
m
```

[15]:
```
array([[3, 5, 5, 0, 1],
       [8, 5, 8, 3, 1],
       [6, 1, 8, 4, 9]])
```

[16]:
```python
m[0, 0]
```

[16]: 3

[17]:
```python
m[1, 1]
```

[17]: 5

[18]:
```python
m[2,3] = 9999
```

[19]:
```
m
```

[19]:
```
array([[   3,    5,    5,    0,    1],
       [   8,    5,    8,    3,    1],
       [   6,    1,    8, 9999,    9]])
```

[20]:
```
m[:, 0] # Select all rows and the first column
```

[20]:
```
array([3, 8, 6])
```

[21]:
```
m[1, :] # select the first row and all columns
```

[21]:
```
array([8, 5, 8, 3, 1])
```

[22]:
```
m[:2, :3] # select first 2 rows and first 3 columns
```

[22]:
```
array([[3, 5, 5],
       [8, 5, 8]])
```

[23]:
```
m[1::2] # select from 2nd row to the end by two (2. satır dahil tüm satıları␣
↪2şer atlayarak seç yani 2 4 6. satırları örneğin)
```

[23]:
```
array([[8, 5, 8, 3, 1]])
```

2.3.1 Fancy Index

[24]:
```
v = np.arange(0, 30, 3) # create an array between 0 and 30 with three by three
v
```

[24]:
```
array([ 0,  3,  6,  9, 12, 15, 18, 21, 24, 27])
```

[25]:
```
v[[1, 2, 3]] # get the second, third and fourth values of v array
```

[25]:
```
array([3, 6, 9])
```

[26]:
```
m = np.arange(9).reshape((3, 3))
m
```

[26]:
```
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
```

[27]:
```
m[2, [1,2]] # get the third row and second and third columns of matrix m
```

[27]:
```
array([7, 8])
```

2.4 Conditinal Element Operations

[28]:
```
v = np.array([1, 2, 3, 4, 5])
v
```

[28]:
```
array([1, 2, 3, 4, 5])
```

[29]:
```
ab = []

for i in v:
    if i < 3:
        ab.append(i)
ab
```

[29]:
```
[1, 2]
```

[30]:
```
v < 3
```

[30]:
```
array([ True,  True, False, False, False])
```

[31]:
```
v[v < 3]
```

[31]:
```
array([1, 2])
```

[32]:
```
v[(v>4) | (v<2)] # select the elements that are greater than 4 or lower than 2
```

[32]:
```
array([1, 5])
```

[33]:
```
v * 5 / 10
```

[33]:
```
array([0.5, 1. , 1.5, 2. , 2.5])
```

[34]:
```
v - 1
```

[34]:
```
array([0, 1, 2, 3, 4])
```

[35]:
```
np.subtract(v, 1) # same with v - 1
```

[35]:
```
array([0, 1, 2, 3, 4])
```

[36]:
```
np.add(v, 1) # same with v + 1
```

[36]:
```
array([2, 3, 4, 5, 6])
```

[37]:
```
np.mean(v) # get the mean value
```

[37]:
```
3.0
```

[38]:
```
v.sum() # get the summation of values
```

[38]: 15

[39]:
```
a = np.random.randint(0, 10, size=20)
np.mean(a > 8) # basically it gives the percent of the array numbers that are␣
↪greater than 8
```

[39]: 0.1

[40]:
```
a
```

[40]: array([2, 5, 2, 9, 7, 8, 0, 7, 0, 0, 8, 9, 2, 6, 0, 5, 6, 7, 8, 2])

[41]:
```
np.sort(a) # sort array numbers
```

[41]: array([0, 0, 0, 0, 2, 2, 2, 2, 5, 5, 6, 6, 7, 7, 7, 8, 8, 8, 9, 9])

[42]:
```
np.sort(a)[::-1] # sort array number revers order
```

[42]: array([9, 9, 8, 8, 8, 7, 7, 7, 6, 6, 5, 5, 2, 2, 2, 2, 0, 0, 0, 0])

[43]:
```
# numpy reading a csv file
csv_array = np.genfromtxt('sample.csv', delimiter=',')
csv_array
```

[43]: array([34., 9., 12., 11., 7.])

2.5 Math with Python

[44]:
```
np.ones((3,3), dtype=int) # generate a matrix with ones according to given␣
↪dimensions
```

[44]:
```
array([[1, 1, 1],
       [1, 1, 1],
       [1, 1, 1]])
```

[45]:
```
np.linspace(1, 10, 6) # create an array of 6 elements equally distributed␣
↪between 1 and 10
```

[45]: array([1. , 2.8, 4.6, 6.4, 8.2, 10.])

2.5.1 Matrix Multiplication, Inverse

[53]:
```
a = np.array([[1, 2], [3, 4]])
a
```

[53]:
```
array([[1, 2],
       [3, 4]])
```

[54]:
```python
b = np.array([[2, 4], [8, 16]])
b
```

[54]:
```
array([[ 2,  4],
       [ 8, 16]])
```

[55]:
```python
c = a @ b # matrix multiplication
c
```

[55]:
```
array([[18, 36],
       [38, 76]])
```

[56]:
```python
np.linalg.pinv(c) # take the pseudo inverse of a matrix (np.linalg.inv() takes␣
↪normal inverse though it can give an error if matrix has no inverse)
```

[56]:
```
array([[0.0020362 , 0.00429864],
       [0.0040724 , 0.00859729]])
```

2.5.2 hstack, vstack

[49]:
```python
a = np.array([1, 1])
b = np.array([2, 2])
np.hstack((a, b)) # horizontal concatenation
```

[49]:
```
array([1, 1, 2, 2])
```

[50]:
```python
np.vstack((a, b)) # vertical concatenation
```

[50]:
```
array([[1, 1],
       [2, 2]])
```

2.5.3 Append, insert methods

[62]:
```python
a = np.array([1, 2, 3])
np.append(a, 4) # note that you cannot do a.append(4) like we do in list objects
```

[62]:
```
array([1, 2, 3, 4])
```

[64]:
```python
np.insert(a, 2, 7) # insert 7 to the 3rd index
```

[64]:
```
array([1, 2, 7, 3])
```

2.5.4 hsplit, vsplit methods

[76]:
```python
a = np.random.normal(5, 20, (3,3))
a
```

[76]:
```
array([[-11.90542731,  64.42670478,  35.85083511],
       [-11.95150277,  -7.54593028,  -1.95613269],
       [-11.06376621,  26.18424815,  26.30196163]])
```

[77]:
```python
np.hsplit(a, 3) # horizontally split the matrix
```

[77]:
```
[array([[-11.90542731],
        [-11.95150277],
        [-11.06376621]]),
 array([[64.42670478],
        [-7.54593028],
        [26.18424815]]),
 array([[35.85083511],
        [-1.95613269],
        [26.30196163]])]
```

[78]:
```python
np.vsplit(a, 3) # vertically split the matrix
```

[78]:
```
[array([[-11.90542731,  64.42670478,  35.85083511]]),
 array([[-11.95150277,  -7.54593028,  -1.95613269]]),
 array([[-11.06376621,  26.18424815,  26.30196163]])]
```

3 Pandas DataFrame

[79]:
```python
import pandas as pd

l = [1, 2, 39, 67, 90]
df = pd.DataFrame(l, columns = ["variable"])
df
```

[79]:

	variable
0	1
1	2
2	39
3	67
4	90

[80]:
```python
type(df)
```

[80]:
```
pandas.core.frame.DataFrame
```

[81]:
```python
m = np.arange(1, 10).reshape((3, 3))
m
```

[81]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

[82]:
```
df = pd.DataFrame(m, columns=["var1", "var2", "var3"])
df
```

[82]:

	var1	var2	var3
0	1	2	3
1	4	5	6
2	7	8	9

[83]:
```
df.columns
```

[83]:
```
Index(['var1', 'var2', 'var3'], dtype='object')
```

[84]:
```
df.columns = ("deg1", "deg2", "deg3")
df
```

[84]:

	deg1	deg2	deg3
0	1	2	3
1	4	5	6
2	7	8	9

[85]:
```
df.axes
```

[85]:
```
[RangeIndex(start=0, stop=3, step=1),
 Index(['deg1', 'deg2', 'deg3'], dtype='object')]
```

[86]:
```
df.shape # number of rows, number of columns
```

[86]:
```
(3, 3)
```

[87]:
```
df.ndim # number of dimensions
```

[87]:
```
2
```

[88]:
```
df.size # number of elements the dataframe has
```

[88]:
```
9
```

[89]:
```
df.values # get values of dataframe in the form of matrix
```

[89]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

[90]:
```
type(df.values)
```

[90]:
```
numpy.ndarray
```

3.1 Element Operations

[91]:
```
s1 = np.random.randint(10, size=5)
s2 = np.random.randint(10, size=5)
s3 = np.random.randint(10, size=5)
s1
```

[91]:
```
array([7, 5, 9, 3, 7])
```

[92]:
```
sozluk = {"var1": s1, "var2": s2, "var3": s3 }
sozluk
```

[92]:
```
{'var1': array([7, 5, 9, 3, 7]),
 'var2': array([0, 4, 4, 9, 5]),
 'var3': array([4, 1, 7, 8, 8])}
```

[93]:
```
df = pd.DataFrame(sozluk) # create dataframe using a dictionary object
df.head()
```

[93]:

	var1	var2	var3
0	7	0	4
1	5	4	1
2	9	4	7
3	3	9	8
4	7	5	8

[94]:
```
df.index = ["a", "b", "c", "d", "d"] # change indexes of a dataframe
df
```

[94]:

	var1	var2	var3
a	7	0	4
b	5	4	1
c	9	4	7
d	3	9	8
d	7	5	8

[97]:
```
# use .reset_index() method to fix non-consecutive indices
df.reset_index()
```

[97]:

	index	var1	var2	var3
0	a	7	0	4
1	b	5	4	1
2	c	9	4	7
3	d	3	9	8
4	d	7	5	8

[98]:
```
df.reset_index(drop=True) # drop the old index column. If you give inplace=True␣
↪argument it drops it permanently
```

[98]:

	var1	var2	var3
0	7	0	4
1	5	4	1
2	9	4	7
3	3	9	8
4	7	5	8

[99]:
```
df["c": "e"]
```

[99]:

	var1	var2	var3
c	9	4	7
d	3	9	8
d	7	5	8

[100]:
```
df.drop("a", axis=0) # drop an instance according to its index
```

[100]:

	var1	var2	var3
b	5	4	1
c	9	4	7
d	3	9	8
d	7	5	8

[101]:
```
df # drop didn't affect the dataframe because we didn't give inplace argument␣
↪as True (inplace=True)
```

[101]:

	var1	var2	var3
a	7	0	4
b	5	4	1
c	9	4	7
d	3	9	8
d	7	5	8

[102]:
```
df.drop("a", axis=0, inplace=True) # drop method will now affect the dataframe␣
↪permanently
df
```

[102]:

	var1	var2	var3
b	5	4	1
c	9	4	7
d	3	9	8
d	7	5	8

[103]:
```
l = ["c", "d"]
df.drop(l, axis = 0)
```

[103]:

	var1	var2	var3
b	5	4	1

[104]:
```python
"var1" in df
```

[104]: True

[105]:
```python
"var8" in df
```

[105]: False

[106]:
```python
df[["var1"]] # select the variable as a dataframe object
```

[106]:
```
   var1
b     5
c     9
d     3
d     7
```

[107]:
```python
# df.var1 --> same as df["var1"] if the column name doesn't start with a␣
↪number, contain spaces or special characters
df["var1"] # select the variable as a series object
```

[107]:
```
b    5
c    9
d    3
d    7
Name: var1, dtype: int32
```

[108]:
```python
type(df["var1"])
```

[108]: pandas.core.series.Series

[109]:
```python
df["var4"] = df["var1"] / df["var2"] # create new variables using other ones
df.head()
```

[109]:
```
   var1  var2  var3      var4
b     5     4     1  1.250000
c     9     4     7  2.250000
d     3     9     8  0.333333
d     7     5     8  1.400000
```

[110]:
```python
l = ["var1", "var2"]
df.drop(l, axis=1)
```

[110]:
```
   var3      var4
b     1  1.250000
c     7  2.250000
d     8  0.333333
d     8  1.400000
```

3.2 iloc & loc

[112]:
```
# iloc is an integer based selection
m = np.random.randint(1, 30, size=(10,3))
df = pd.DataFrame(m, columns=["var1","var2","var3"])
df
```

[112]:

	var1	var2	var3
0	23	8	2
1	16	10	29
2	27	7	8
3	19	7	14
4	25	28	6
5	11	24	26
6	6	11	7
7	18	18	15
8	13	22	28
9	27	17	5

[113]:
```
df.iloc[0:3] # select the first three rows
```

[113]:

	var1	var2	var3
0	23	8	2
1	16	10	29
2	27	7	8

[114]:
```
df.iloc[0, 0] # select the first row and first column which is of course the␣
↪value of the first cell
```

[114]: 23

[115]:
```
# loc is a label based selection
df.loc[0:3] # Select the first four rows of the dataframe.
# Note that, the last value of a given slice is included in the loc method
```

[115]:

	var1	var2	var3
0	23	8	2
1	16	10	29
2	27	7	8
3	19	7	14

[116]:
```
df.loc[0:3, "var3"] # select the first four rows of the variable var3
```

[116]:
```
0     2
1    29
2     8
3    14
Name: var3, dtype: int32
```

[117]:
```
df.iloc[0:3][["var3"]] # select the first there rows of the variable var3 in the form of dataframe
```

[117]:

	var3
0	2
1	29
2	8

[118]:
```
l = ["var1", "var3"]
df.loc[0:3, l]
```

[118]:

	var1	var3
0	23	2
1	16	29
2	27	8
3	19	14

3.3 Conditional Selection

[119]:
```
df[["var1","var2","var3"]] # select multiple variables from a dataframe
```

[119]:

	var1	var2	var3
0	23	8	2
1	16	10	29
2	27	7	8
3	19	7	14
4	25	28	6
5	11	24	26
6	6	11	7
7	18	18	15
8	13	22	28
9	27	17	5

[120]:
```
df[0:2][["var1", "var3"]] # Select the first two rows of var1 and var2 in the form dataframe
```

[120]:

	var1	var3
0	23	2
1	16	29

[121]:
```
df["var1"] > 5
```

[121]:
```
0    True
1    True
2    True
3    True
4    True
5    True
```

```
6    True
7    True
8    True
9    True
Name: var1, dtype: bool
```

[122]:
```
df[df["var1"] > 13] # select the dataframe values which var1 variable values␣
↪are greater than 13
```

[122]:
```
   var1  var2  var3
0    23     8     2
1    16    10    29
2    27     7     8
3    19     7    14
4    25    28     6
7    18    18    15
9    27    17     5
```

[123]:
```
df[df["var1"] > 13]["var2"]
```

[123]:
```
0     8
1    10
2     7
3     7
4    28
7    18
9    17
Name: var2, dtype: int32
```

[124]:
```
df.loc[df["var1"] > 6, "var2"]
```

[124]:
```
0     8
1    10
2     7
3     7
4    28
5    24
7    18
8    22
9    17
Name: var2, dtype: int32
```

[125]:
```
df[(df["var1"] > 7) & (df["var3"] > 3)]
```

[125]:
```
   var1  var2  var3
1    16    10    29
2    27     7     8
3    19     7    14
```

```
4    25    28    6
5    11    24   26
7    18    18   15
8    13    22   28
9    27    17    5
```

[126]:
```
df.loc[(df["var1"] > 7) & (df["var3"] > 3), ["var1","var2"]]
```

[126]:
```
   var1  var2
1    16    10
2    27     7
3    19     7
4    25    28
5    11    24
7    18    18
8    13    22
9    27    17
```

[127]:
```
df = pd.read_csv("train.csv") # load the titanic dataset from kaggle
df.head()
```

[127]:
```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

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

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

3.4 Rename Columns

[143]:
```
df.rename(columns={"PassengerId": "id", "Pclass": "class"}) # if you give␣
↪inplace=True argument it saves it permenantly
```

[143]:
```
      id  Survived  class                                               Name  \
0      1         0      3                            Braund, Mr. Owen Harris
1      2         1      1  Cumings, Mrs. John Bradley (Florence Briggs Th…
2      3         1      3                             Heikkinen, Miss. Laina
3      4         1      1       Futrelle, Mrs. Jacques Heath (Lily May Peel)
4      5         0      3                           Allen, Mr. William Henry
..   …         …      …                                                …
886  887         0      2                              Montvila, Rev. Juozas
887  888         1      1                       Graham, Miss. Margaret Edith
888  889         0      3           Johnston, Miss. Catherine Helen "Carrie"
889  890         1      1                              Behr, Mr. Karl Howell
890  891         0      3                                Dooley, Mr. Patrick

        Sex   Age  SibSp  Parch            Ticket     Fare Cabin Embarked  \
0      male  22.0      1      0         A/5 21171   7.2500   NaN        S
1    female  38.0      1      0          PC 17599  71.2833   C85        C
2    female  26.0      0      0  STON/O2. 3101282   7.9250   NaN        S
3    female  35.0      1      0            113803  53.1000  C123        S
4      male  35.0      0      0            373450   8.0500   NaN        S
..      …   …    …    …               …      …   …      …
886    male  27.0      0      0            211536  13.0000   NaN        S
887  female  19.0      0      0            112053  30.0000   B42        S
888  female   NaN      1      2        W./C. 6607  23.4500   NaN        S
889    male  26.0      0      0            111369  30.0000  C148        C
890    male  32.0      0      0            370376   7.7500   NaN        Q

    Elderliness
0         Young
1      Mid-Aged
2      Mid-Aged
3      Mid-Aged
4      Mid-Aged
..          …
886    Mid-Aged
887       Young
888         Old
889    Mid-Aged
890    Mid-Aged

[891 rows x 13 columns]
```

3.5 Aggregation & Grouping

Methods: - count() - first() - last() - mean() - median() - min() - max() - std() - var() - sum()

[128]:
```
df[["Age", "Fare"]].max()
```

[128]:
```
Age      80.0000
Fare    512.3292
dtype: float64
```

[129]:
```
df.max()
```

[129]:
```
PassengerId                         891
Survived                              1
Pclass                                3
Name           van Melkebeke, Mr. Philemon
Sex                                male
Age                                  80
SibSp                                 8
Parch                                 6
Ticket                        WE/P 5735
Fare                            512.329
dtype: object
```

[130]:
```
df.groupby("Sex").agg({"Age": "mean"}) # or df.groupby("Sex").agg({"Age": np.
↪mean})
```

[130]:
```
              Age
Sex
female  27.915709
male    30.726645
```

[131]:
```
df.groupby(["Sex", "Pclass"]).agg({"Age": "mean"})
```

[131]:
```
                     Age
Sex    Pclass
female 1       34.611765
       2       28.722973
       3       21.750000
male   1       41.281386
       2       30.740707
       3       26.507589
```

[132]:
```
df.groupby(["Sex", "Pclass", "Embarked"]).agg({"Survived":"mean", "Age":"max"})
```

[132]:
```
                        Survived   Age
Sex    Pclass Embarked
female 1      C         0.976744  60.0
              Q         1.000000  33.0
              S         0.958333  63.0
       2      C         1.000000  28.0
              Q         1.000000  30.0
              S         0.910448  57.0
       3      C         0.652174  45.0
```

```
                 Q          0.727273  39.0
                 S          0.375000  63.0
male    1        C          0.404762  71.0
                 Q          0.000000  44.0
                 S          0.354430  80.0
        2        C          0.200000  36.0
                 Q          0.000000  57.0
                 S          0.154639  70.0
        3        C          0.232558  45.5
                 Q          0.076923  70.5
                 S          0.128302  74.0
```

[133]:
```
df.groupby(["Sex", "Pclass", "Embarked"]).agg({"Survived": "mean", "Age":
 ↪["min", np.mean, "max"]})
```

[133]:
```
                          Survived    Age
                              mean    min       mean   max
Sex    Pclass Embarked
female 1      C           0.976744  16.00  36.052632  60.0
              Q           1.000000  33.00  33.000000  33.0
              S           0.958333   2.00  32.704545  63.0
       2      C           1.000000   3.00  19.142857  28.0
              Q           1.000000  30.00  30.000000  30.0
              S           0.910448   2.00  29.719697  57.0
       3      C           0.652174   0.75  14.062500  45.0
              Q           0.727273  15.00  22.850000  39.0
              S           0.375000   1.00  23.223684  63.0
male   1      C           0.404762  17.00  40.111111  71.0
              Q           0.000000  44.00  44.000000  44.0
              S           0.354430   0.92  41.897188  80.0
       2      C           0.200000   1.00  25.937500  36.0
              Q           0.000000  57.00  57.000000  57.0
              S           0.154639   0.67  30.875889  70.0
       3      C           0.232558   0.42  25.016800  45.5
              Q           0.076923   2.00  28.142857  70.5
              S           0.128302   1.00  26.574766  74.0
```

3.6 Apply Method

[134]:
```
df[["Age", "Parch"]].apply(np.sum, axis=0)
```

[134]:
```
Age      21205.17
Parch      340.00
dtype: float64
```

[135]:
```
df[["Age", "Parch"]].apply(lambda x: x**2).head()
```

[135]:
```
        Age  Parch
0   484.0      0
1  1444.0      0
2   676.0      0
3  1225.0      0
4  1225.0      0
```

[136]:
```
df[["Age","Parch"]].apply(lambda x: (x-x.mean())/x.std())
```

[136]:
```
          Age     Parch
0   -0.530005 -0.473408
1    0.571430 -0.473408
2   -0.254646 -0.473408
3    0.364911 -0.473408
4    0.364911 -0.473408
..        …         …
886 -0.185807 -0.473408
887 -0.736524 -0.473408
888       NaN  2.007806
889 -0.254646 -0.473408
890  0.158392 -0.473408

[891 rows x 2 columns]
```

[137]:
```
# lambda with multiple conditions:
# We created a new feature with apply and lambda. If age variable values are␣
↪greater than 25 it will be old else it will be young.
df["Elderliness"] = df.apply((lambda row: "Young" if row.Age <= 25 else "Old"),␣
↪axis=1)
df.head()
```

[137]:
```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

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

   Parch           Ticket     Fare Cabin Embarked Elderliness
0      0        A/5 21171   7.2500   NaN        S       Young
```

```
1       0           PC 17599  71.2833   C85         C          Old
2       0  STON/O2. 3101282   7.9250   NaN         S          Old
3       0            113803  53.1000  C123         S          Old
4       0            373450   8.0500   NaN         S          Old
```

[138]:
```
# Applying lambda with if, elif, else blocks
df["Elderliness"] = df.apply((lambda row: "Young" if row.Age <= 25 else␣
↪("Mid-Aged" if 25 < row.Age <= 50 else "Old")), axis=1)
df.head()
```

[138]:
```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

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

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

3.7 Pivoting with pandas

[92]:
```
df.pivot_table("Survived", index ="Sex", columns ="Age")
```

[92]:
```
Age     0.42   0.67   0.75   0.83   0.92   1.00      2.00   3.00   4.00   \
Sex
female    NaN    NaN    1.0    NaN    NaN    1.0  0.333333    0.5    1.0
male      1.0    1.0    NaN    1.0    1.0    0.6  0.250000    1.0    0.4

Age     5.00   …     62.00  63.00  64.00  65.00  66.00  70.00  70.50  71.00  \
Sex            …
female    1.0  …  1.000000    1.0    NaN    NaN    NaN    NaN    NaN    NaN
male      NaN  …  0.333333    NaN    0.0    0.0    0.0    0.0    0.0    0.0

Age     74.00  80.00
Sex
```

```
female     NaN    NaN
male       0.0    1.0

[2 rows x 88 columns]
```

[93]:
```
df.head()
```

[93]:
```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

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

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

[95]:
```
df["NewAge"] = pd.cut(df["Age"], [0, 10, 18, 25, 40, 90])
df.pivot_table("Survived", index="Sex", columns="NewAge")
```

[95]:
```
NewAge   (0, 10]  (10, 18]  (18, 25]  (25, 40]  (40, 90]
Sex
female  0.612903  0.729730  0.759259  0.802198  0.770833
male    0.575758  0.131579  0.120370  0.220930  0.176471
```

3.8 Merge Dataframes

Operations will not be done. Codes will be given only.

new_df = pd.merge(df1, df2)

new_df = df1.merge(df2)

big_df = df1.merge(df2).merge(df3) –> merge more than 2 dataframes

We use rename() and merge() methods together when the column names of dataframes are the same.
For example: **pd.merge(orders, customers.rename(columns = {'id': 'customer_id}))**

If we don't want to use rename() method when merging dataframes we use left_on, right_on:
For example: **pd.merge(orders, customers, left_on='customer_id', right_on='id')**

with left_on and right_on we specify which columns we want to perform merge on. We can add suffixes for the name of the columns;
For example: **pd.merge(orders, customers, left_on='customer_id', right_on='id', suffixes=['_order', '_customer'])**

When there are unmatched rows:
pd.merge(df1, df2, how='outer') how can take 'left' and 'right' arguments as well. 'left' means for example all rows from the first dataframe included but only matching row from the second. For the 'right' it is vice versa.

3.9 Concatenate Dataframes

The difference between concat and merge in dataframes: concat method only works if all of the columns are the same in all of the dataframes

pd.concat([df1, df2, df3, ...]) ignore_index=True(indeksler sıralıysa oluşan indeks kayması önlenir.) can be given as an argument.

4 Exploratory Data Analysis & Data Visualization

[1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
from matplotlib import pyplot as plt

pd.set_option('display.max_columns', None)
df = pd.read_csv("train.csv") # kernel has been restarted and titanic dataset␣
 ↪reread.
```

4.1 Outlook

[2]:
```
df.head()
```

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

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

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

[3]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object
 11  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

[4]:
```
df.describe().T
```

[4]:
```
             count       mean         std   min       25%       50%    75%  \
PassengerId  891.0  446.000000  257.353842  1.00  223.5000  446.0000  668.5
Survived     891.0    0.383838    0.486592  0.00    0.0000    0.0000    1.0
Pclass       891.0    2.308642    0.836071  1.00    2.0000    3.0000    3.0
Age          714.0   29.699118   14.526497  0.42   20.1250   28.0000   38.0
SibSp        891.0    0.523008    1.102743  0.00    0.0000    0.0000    1.0
Parch        891.0    0.381594    0.806057  0.00    0.0000    0.0000    0.0
Fare         891.0   32.204208   49.693429  0.00    7.9104   14.4542   31.0

                  max
PassengerId  891.0000
Survived       1.0000
Pclass         3.0000
Age           80.0000
SibSp          8.0000
Parch          6.0000
```

```
Fare          512.3292
```

[5]:
```python
df.isnull().values.any() # Is there any null value
```

[5]:
```
True
```

[6]:
```python
df.isnull().sum() # How many null values are there in each variable
```

[6]:
```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

4.2 Categorical Variable Analysis

- How many categorical variables are in the data set?
- What are the number of classes the categorical variables have?
- What are the classes of each categorical variable?
- Simple plots to get a lot of information about data set

[7]:
```python
df.head()
```

[7]:
```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

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

   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
1      0          PC 17599  71.2833   C85        C
```

```
2      0  STON/O2. 3101282   7.9250   NaN        S
3      0           113803  53.1000  C123        S
4      0           373450   8.0500   NaN        S
```

[8]:
```
# how many classes the cat. variable has?
len(df.Survived.unique()) # len(df["Survived"].unique())
```

[8]:
```
2
```

[9]:
```
df.Survived.nunique() # again this gives the same result with len(df.Survived.
 ↪unique())
```

[9]:
```
2
```

[10]:
```
# cat variable classes and frequencies
df.Survived.value_counts() # df["Survived"].value_counts()
```

[10]:
```
0    549
1    342
Name: Survived, dtype: int64
```

[12]:
```
cat_cols = [col for col in df.columns if df[col].dtype == "O"] # get the cat.␣
 ↪variables
print('Kategorik Değişken Sayısı:', len(cat_cols))
print(cat_cols)
```

```
Kategorik Değişken Sayısı: 5
['Name', 'Sex', 'Ticket', 'Cabin', 'Embarked']
```

[15]:
```
# number of cat. variables according to number of classes the variables have
cat_cols = [col for col in df.columns if df[col].nunique() < 10]
print('Kategorik Değişken Sayısı:', len(cat_cols))
cat_cols
# Usually, categorical variables do not have more than 10 classes. Categorical␣
 ↪variables are also not necesarly has to be values combined of strings. For␣
 ↪example, Sex variable might be encoded as 0 and 1 to represent woman and man.
 ↪ In that case we might have to check numeric types of categorical variables␣
 ↪as well in the data set to do a good categorical variable analysis.
```

```
Kategorik Değişken Sayısı: 6
```

[15]:
```
['Survived', 'Pclass', 'Sex', 'SibSp', 'Parch', 'Embarked']
```

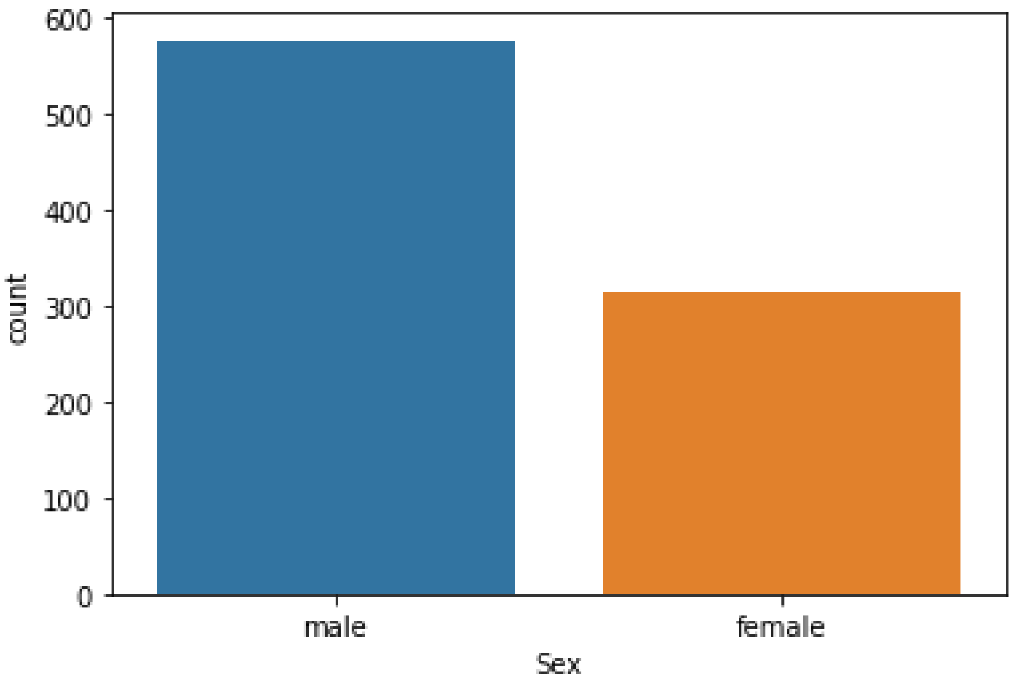
[16]:
```
df[cat_cols].nunique() # number of classes each cat variable has
```

[16]:
```
Survived    2
Pclass      3
Sex         2
SibSp       7
```

```
Parch       7
Embarked    3
dtype: int64
```

- Bar chart for cat variables
- Histogram, boxplot for numerical variables

[17]:
```
# bar chart with seaborn library
sns.countplot(x="Sex", data=df);
```



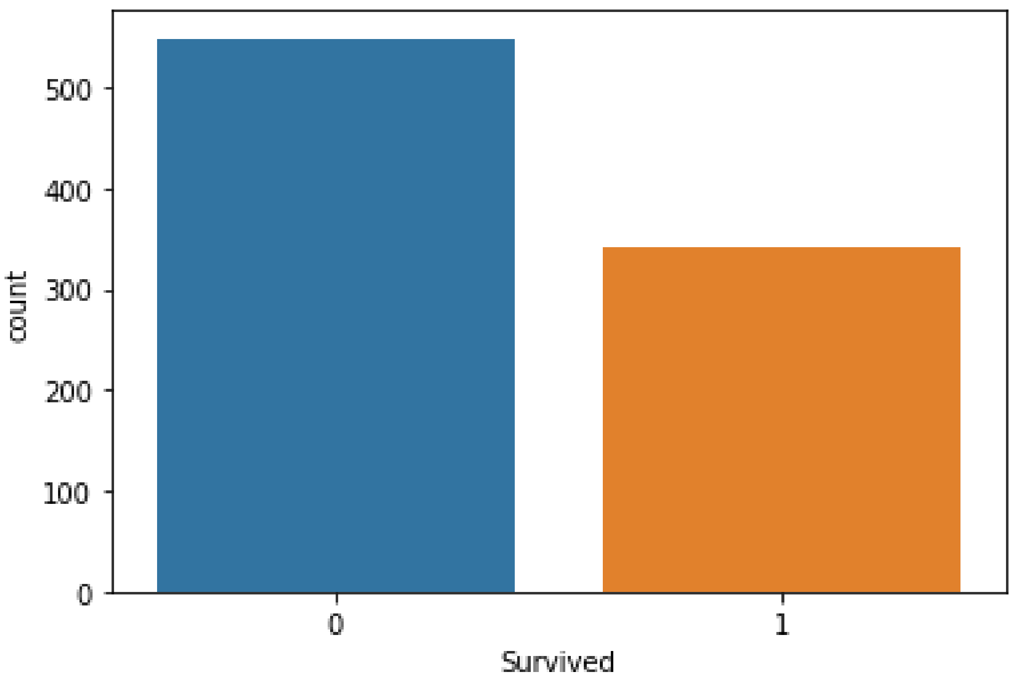
[18]:
```
100 * df["Sex"].value_counts() / len(df) # get the percantage of distribution␣
↪of classes in a variable
```

[18]:
```
male      64.758698
female    35.241302
Name: Sex, dtype: float64
```

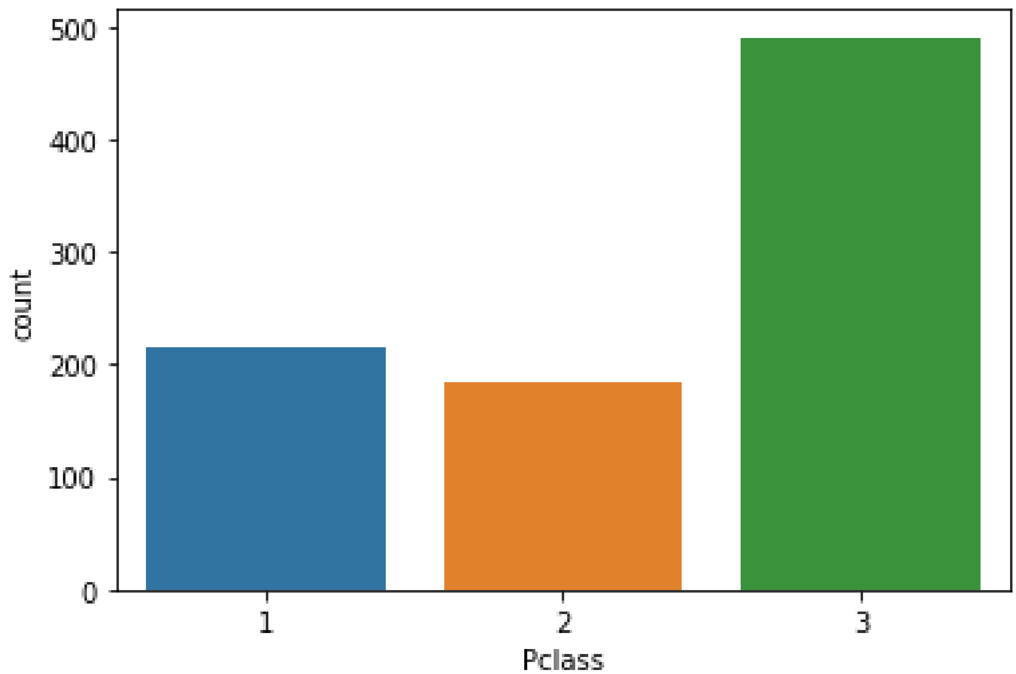
[25]:
```
# function of categorical variable summary
def cat_summary(data): # data should be given as dataframe argument
    cat_names = [col for col in data.columns if data[col].nunique() < 10]
    for col in cat_names:
        print(pd.DataFrame({col: data[col].value_counts(),
                            "Ratio (%)": round(100 * data[col].value_counts()/␣
↪len(data), 2)}), end="\n\n\n")
        sns.countplot(x=col, data=data)
```

```
        plt.show()

cat_summary(df)
```

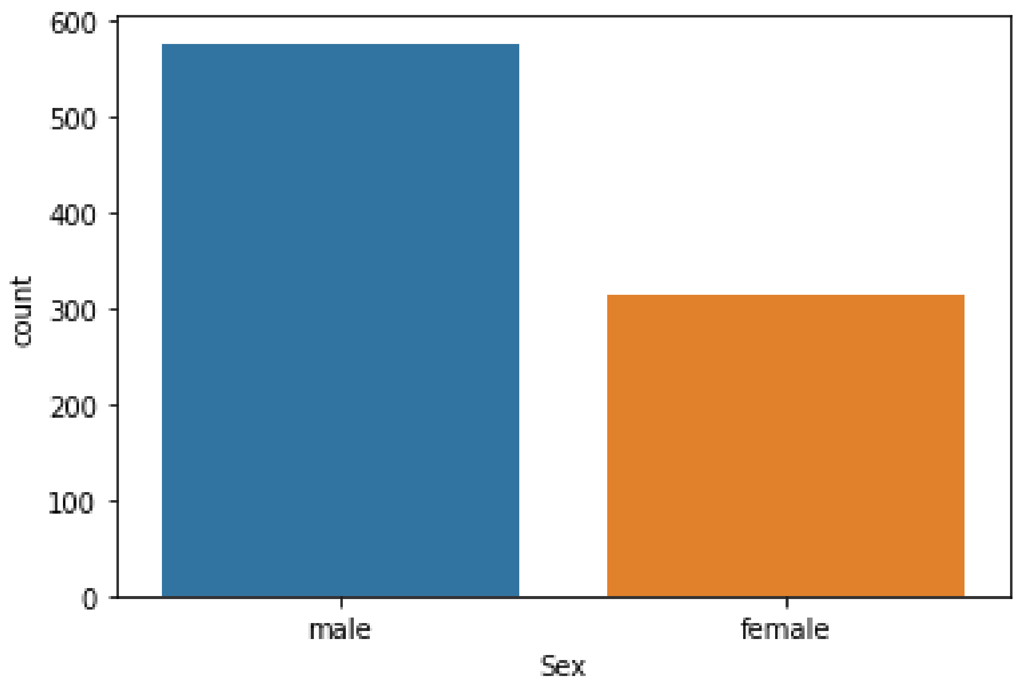
```
   Survived  Ratio (%)
0       549      61.62
1       342      38.38
```



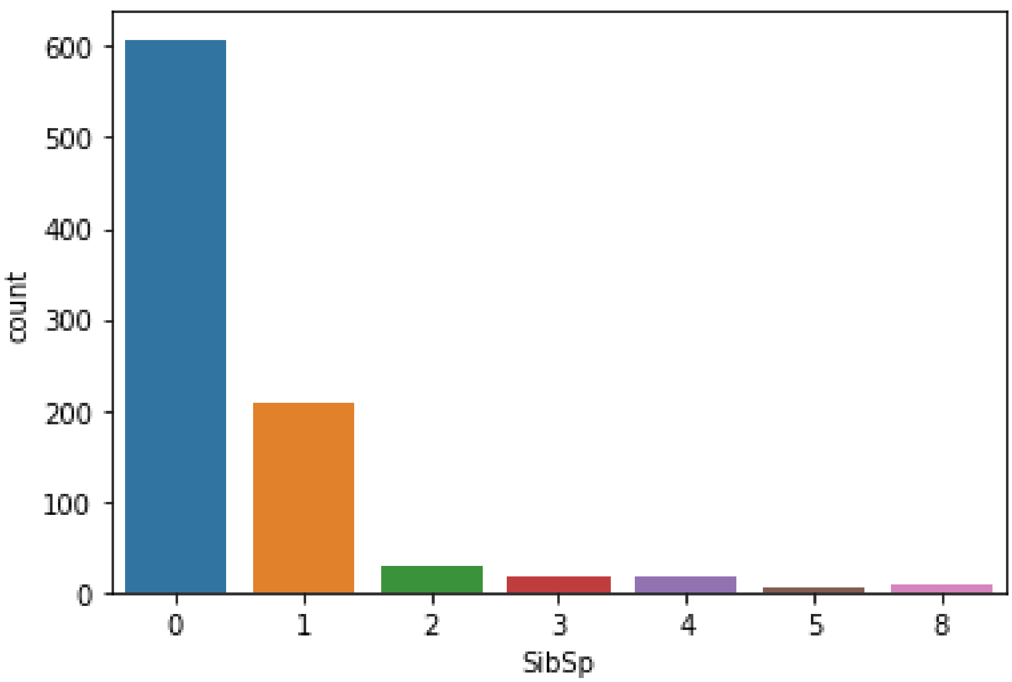

```
   Pclass  Ratio (%)
3     491      55.11
1     216      24.24
2     184      20.65
```



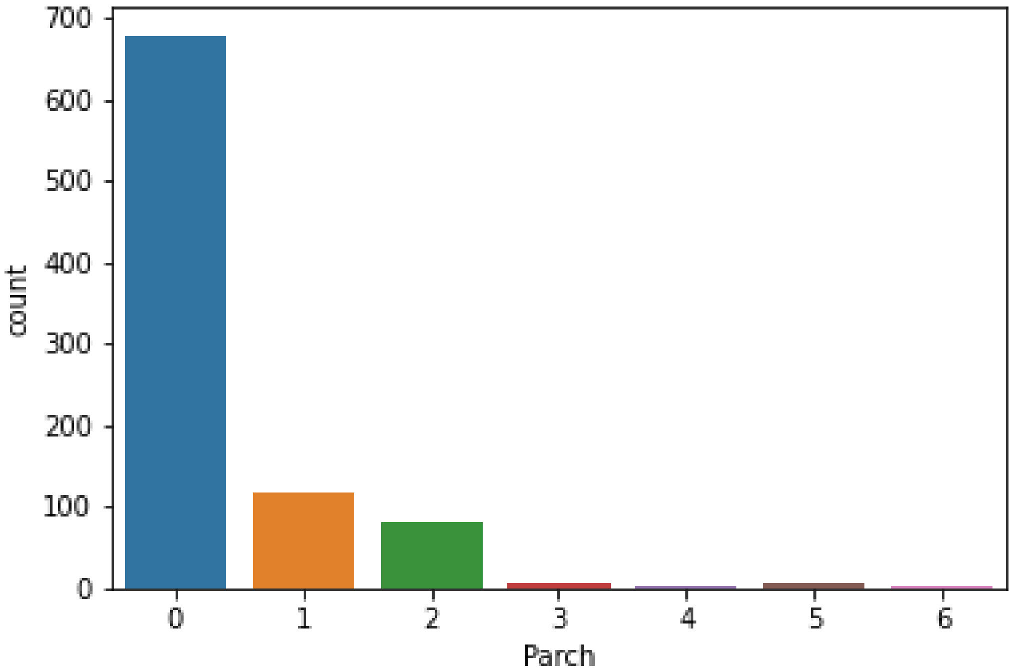
	Sex	Ratio (%)
male	577	64.76
female	314	35.24



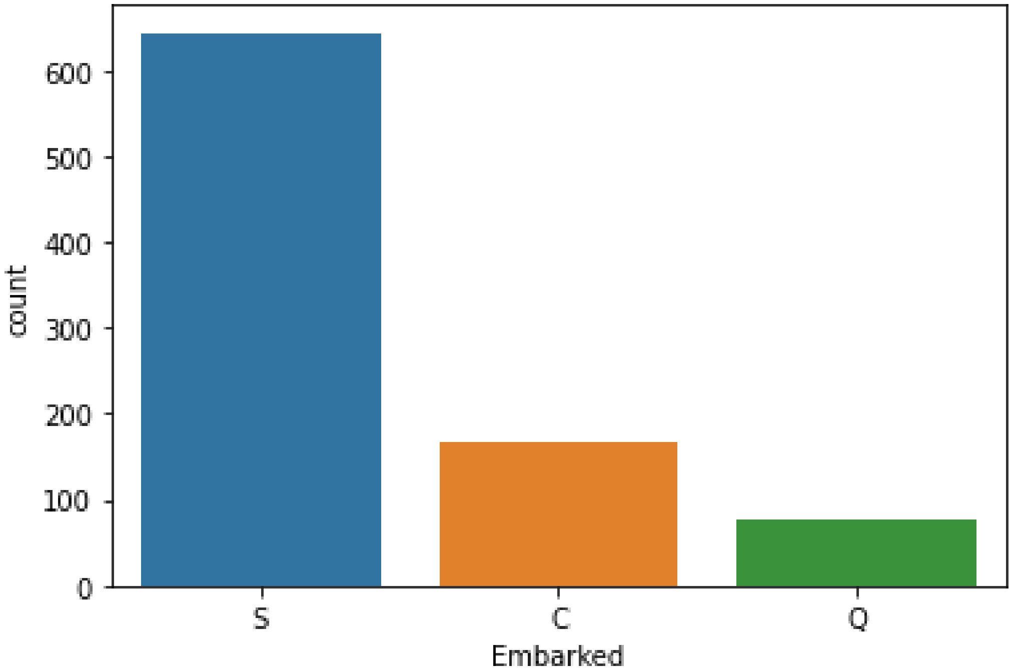
	SibSp	Ratio (%)
0	608	68.24
1	209	23.46
2	28	3.14
4	18	2.02
3	16	1.80
8	7	0.79
5	5	0.56



	Parch	Ratio (%)
0	678	76.09
1	118	13.24
2	80	8.98
5	5	0.56
3	5	0.56
4	4	0.45
6	1	0.11



	Embarked	Ratio (%)
S	644	72.28
C	168	18.86
Q	77	8.64



[26]:
```
cat_cols = [col for col in df.columns if df[col].dtype=="O"]
cat_cols
```

[26]: ['Name', 'Sex', 'Ticket', 'Cabin', 'Embarked']

[32]:
```
def cat_summary(data, categorical_cols, number_of_classes=10):

    var_count = 0  # Number of cat. variables will be printed.
    vars_more_classes = []  # Number of cat. variables that have more than␣
↪given argument number_of_classes will be returned.

    for var in categorical_cols:
        if data[var].nunique() <= number_of_classes:  # select according to its␣
↪number of classes
            print(pd.DataFrame({var: data[var].value_counts(),
                                "Ratio (%)": round(100 * data[var].value_counts()/␣
↪len(data), 2)}), end="\n\n\n")
            var_count += 1
        else:
            vars_more_classes.append(data[var].name)
    print(f"{var_count} categorical variables have been described.\n\n")
    if len(vars_more_classes) > 0:
        print(f"There are {len(vars_more_classes)} variables which have more␣
↪than {number_of_classes} classes.\n\n")
```

```
        print(f"Variable names that have more than {number_of_classes} classes.
↪\n\n")
        print(vars_more_classes)

cat_summary(df, cat_cols)
```

```
        Sex  Ratio (%)
male    577      64.76
female  314      35.24


   Embarked  Ratio (%)
S       644      72.28
C       168      18.86
Q        77       8.64


2 categorical variables have been described.


There are 3 variables which have more than 10 classes.


Variable names that have more than 10 classes.


['Name', 'Ticket', 'Cabin']
```

4.3 Numerical Variable Analysis

[33]:
```
df.describe().T
```

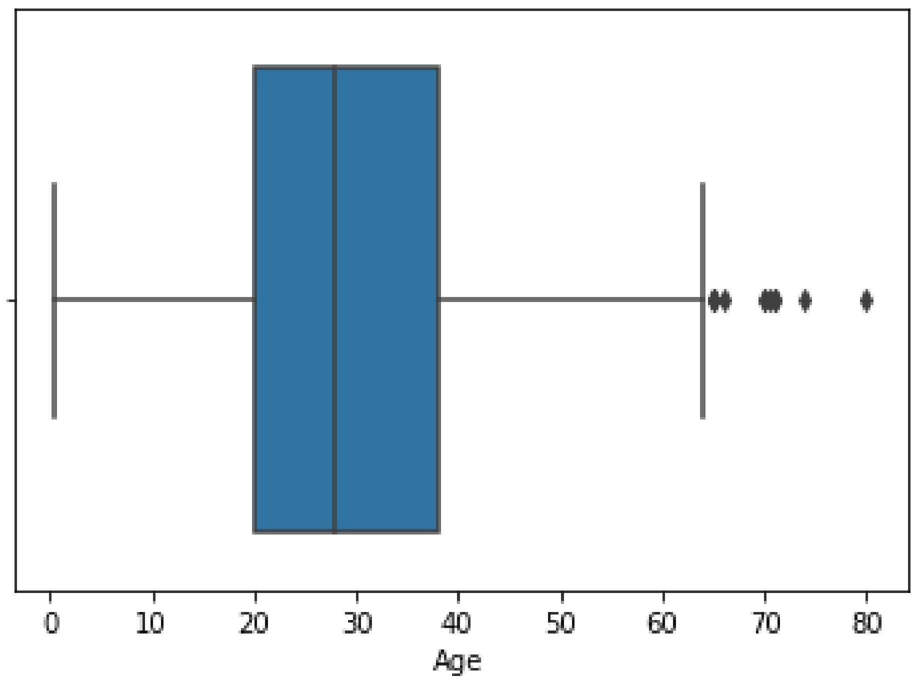
[33]:
```
             count        mean         std   min      25%       50%    75%  \
PassengerId  891.0  446.000000  257.353842  1.00  223.5000  446.0000  668.5
Survived     891.0    0.383838    0.486592  0.00    0.0000    0.0000    1.0
Pclass       891.0    2.308642    0.836071  1.00    2.0000    3.0000    3.0
Age          714.0   29.699118   14.526497  0.42   20.1250   28.0000   38.0
SibSp        891.0    0.523008    1.102743  0.00    0.0000    0.0000    1.0
Parch        891.0    0.381594    0.806057  0.00    0.0000    0.0000    0.0
Fare         891.0   32.204208   49.693429  0.00    7.9104   14.4542   31.0

                  max
PassengerId  891.0000
Survived       1.0000
Pclass         3.0000
Age           80.0000
SibSp          8.0000
```

```
Parch             6.0000
Fare            512.3292
```

[34]:
```python
df.describe([0.05, 0.10, 0.25, 0.50, 0.75, 0.80, 0.90, 0.95, 0.99]).T
```

[34]:
```
             count        mean         std   min      5%    10%       25%  \
PassengerId  891.0  446.000000  257.353842  1.00  45.500  90.00  223.5000
Survived     891.0    0.383838    0.486592  0.00   0.000   0.00    0.0000
Pclass       891.0    2.308642    0.836071  1.00   1.000   1.00    2.0000
Age          714.0   29.699118   14.526497  0.42   4.000  14.00   20.1250
SibSp        891.0    0.523008    1.102743  0.00   0.000   0.00    0.0000
Parch        891.0    0.381594    0.806057  0.00   0.000   0.00    0.0000
Fare         891.0   32.204208   49.693429  0.00   7.225   7.55    7.9104

                  50%    75%       80%       90%        95%        99%  \
PassengerId  446.0000  668.5  713.0000  802.0000  846.50000  882.10000
Survived       0.0000    1.0    1.0000    1.0000    1.00000    1.00000
Pclass         3.0000    3.0    3.0000    3.0000    3.00000    3.00000
Age           28.0000   38.0   41.0000   50.0000   56.00000   65.87000
SibSp          0.0000    1.0    1.0000    1.0000    3.00000    5.00000
Parch          0.0000    0.0    1.0000    2.0000    2.00000    4.00000
Fare          14.4542   31.0   39.6875   77.9583  112.07915  249.00622

                  max
PassengerId  891.0000
Survived       1.0000
Pclass         3.0000
Age           80.0000
SibSp          8.0000
Parch          6.0000
Fare         512.3292
```

[35]:
```python
sns.boxplot(x=df["Age"]); # box plot is very good at visualizing outliers
```



[36]:
```
# get the numerical variables
num_cols = [col for col in df.columns if df[col].dtypes!="O"]
print('Sayısal değişken sayısı:', len(num_cols))
print(num_cols)
```

```
Sayısal değişken sayısı: 7
['PassengerId', 'Survived', 'Pclass', 'Age', 'SibSp', 'Parch', 'Fare']
```

[37]:
```
df.drop("PassengerId", axis=1).columns # Although PassengerId is numerical␣
↪variable it is actually meaningless in terms machine learning modeling. So␣
↪we drop it
```

[37]:
```
Index(['Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp', 'Parch', 'Ticket',
       'Fare', 'Cabin', 'Embarked'],
      dtype='object')
```

[38]:
```
num_cols = [col for col in df.columns if df[col].dtypes != "O" # PassengerId is␣
↪just an Id, Survived is actually our target. That is
            and col not in "PassengerId"                        # why when we␣
↪get the list of numerical variables we disregard them
            and col not in "Survived"]
num_cols
```

[38]:
```
['Pclass', 'Age', 'SibSp', 'Parch', 'Fare']
```

What if we had ıd, ID, iD etc. names to represent id variable? We might want to use regex or a list of names not to select those in our num_cols list:

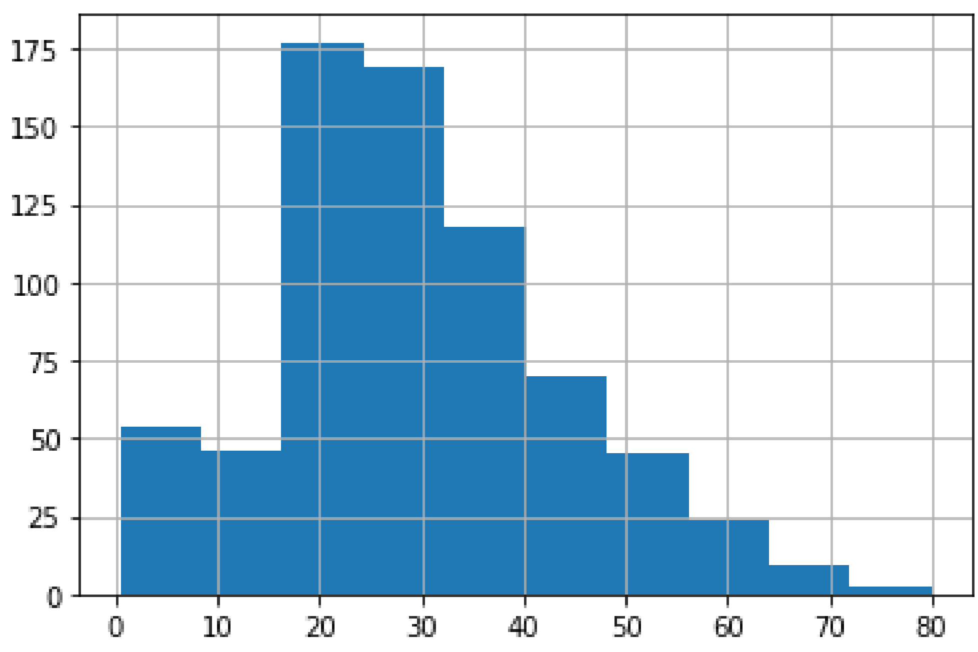
[39]:
```
# with a list:
liste = ["PassengerId", "Passengerid", "PassengerID", "PassengeriD"]
num_cols = [col for col in df.columns if df[col].dtypes != "O"
            and col not in liste
            and col not in "Survived"]
print(num_cols)
```

```
['Pclass', 'Age', 'SibSp', 'Parch', 'Fare']
```

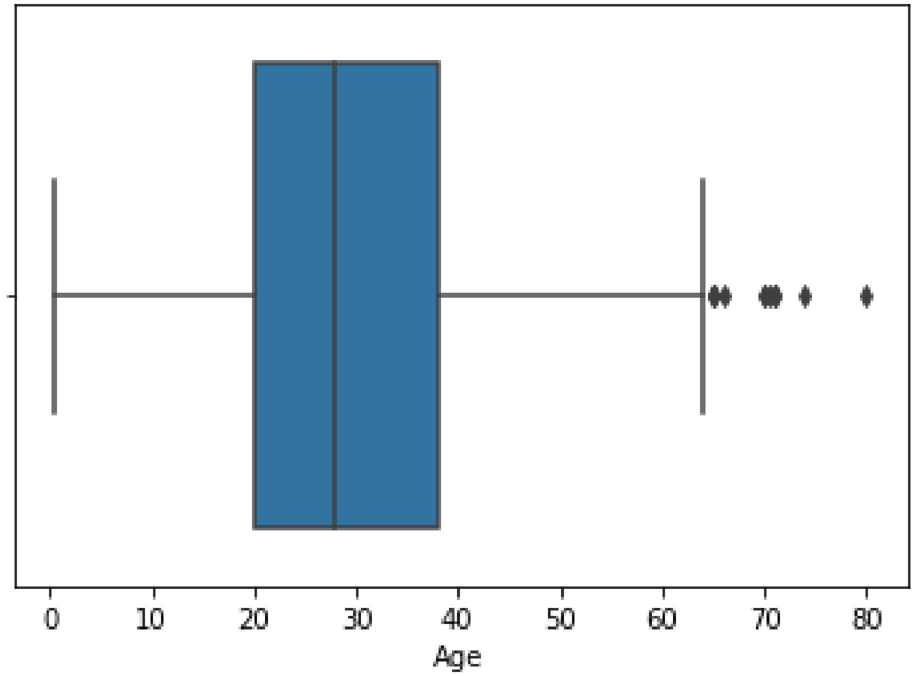
[40]:
```
# with Regex:
import re
passenger_regex = re.compile(r"Passengerid", re.I)
liste = [passenger_regex.search(var).group() for var in df.columns if␣
 ↪passenger_regex.findall(var) !=[]]
num_cols = [col for col in df.columns if df[col].dtypes != "O"
            and col not in liste
            and col not in "Survived"]
print(num_cols)
```

```
['Pclass', 'Age', 'SibSp', 'Parch', 'Fare']
```

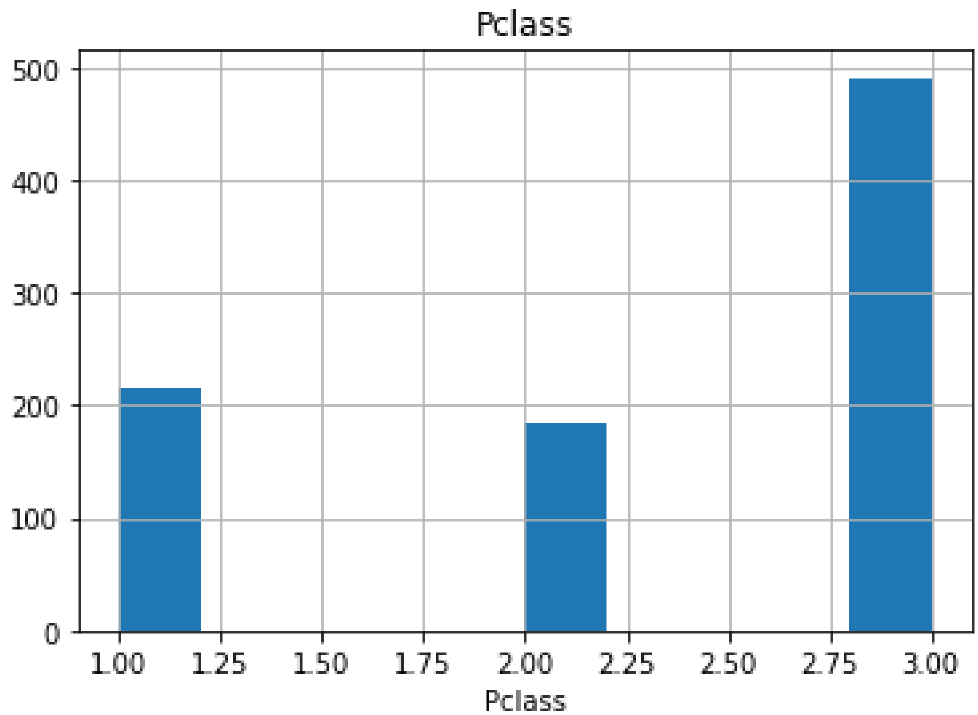
[41]:
```
df["Age"].hist();
```

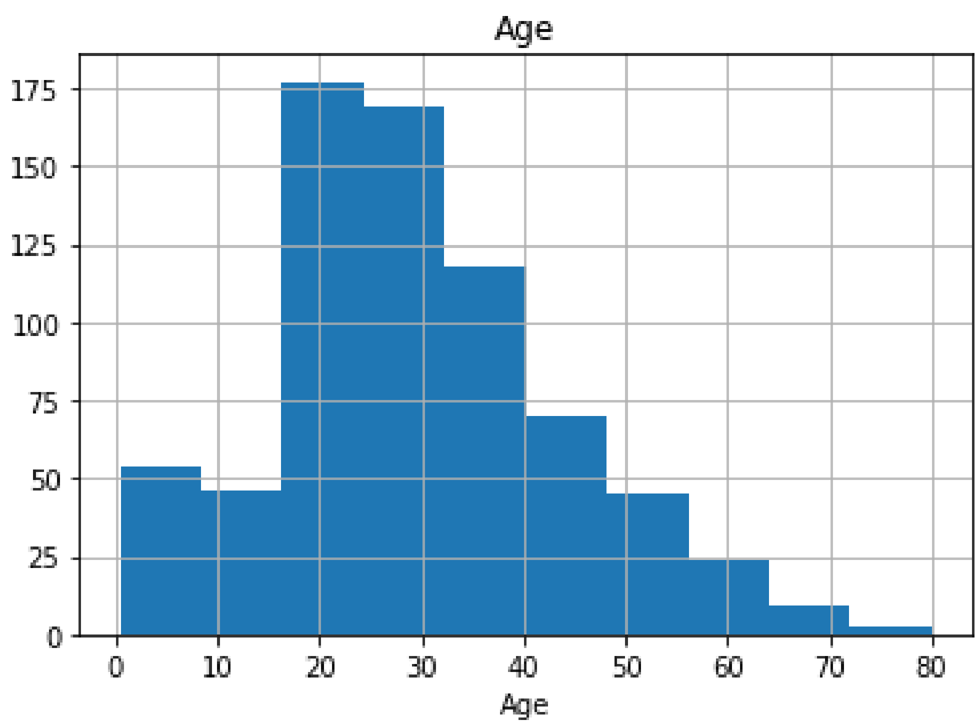


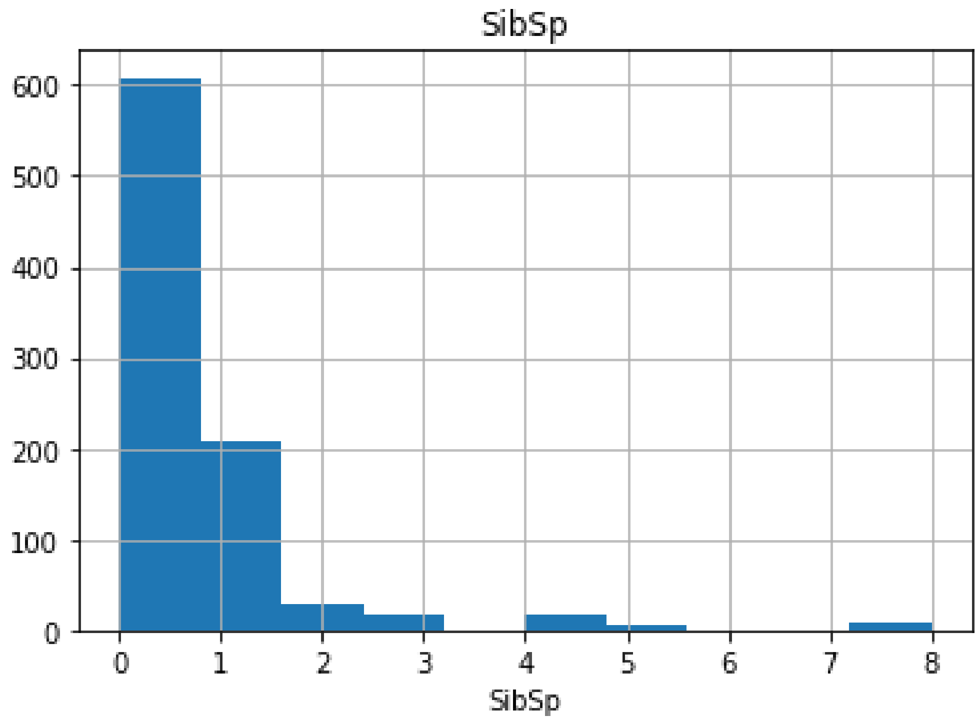
[42]:
```
sns.boxplot(x=df["Age"]);
```

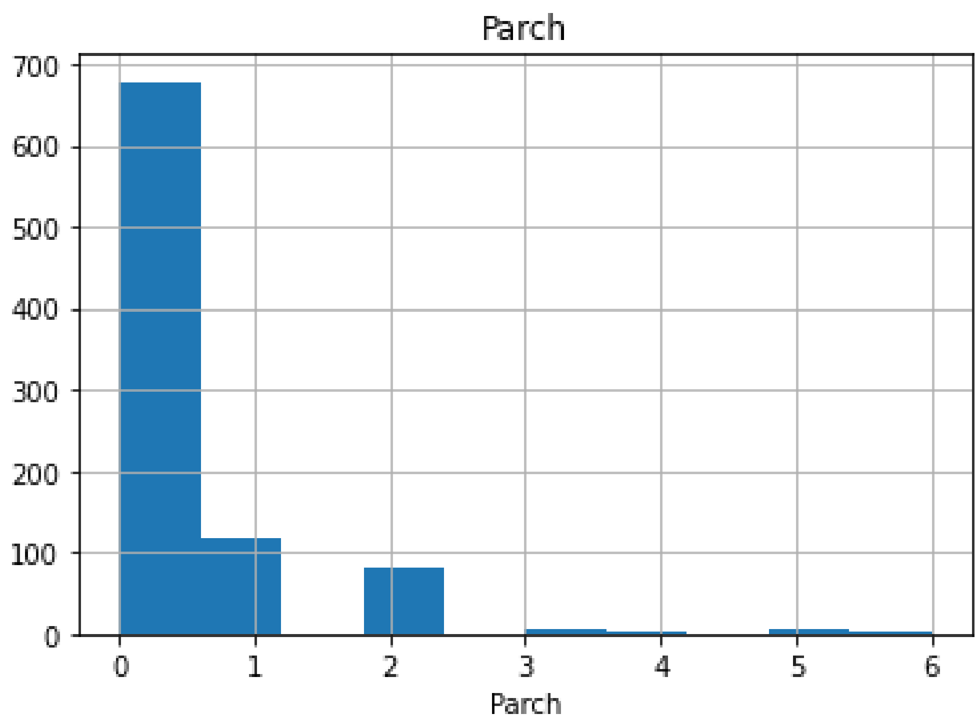


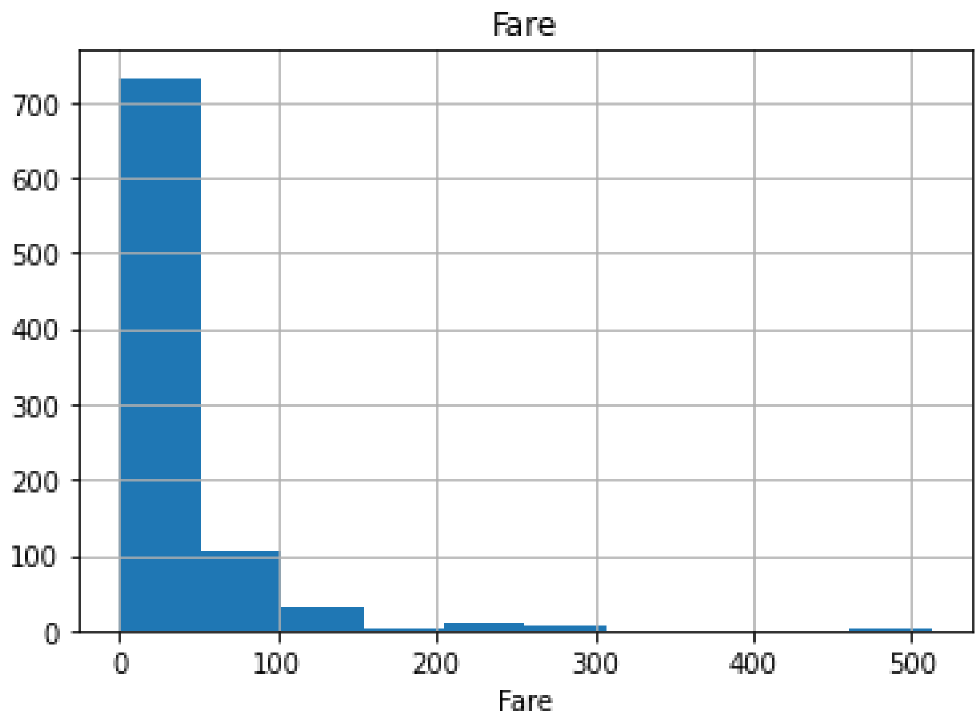
[43]:
```
def hist_for_nums(data, numeric_cols):
    col_counter = 0
    for col in numeric_cols:
        data[col].hist()
        plt.xlabel(col)
        plt.title(col)
        plt.show()
        col_counter += 1
    print(col_counter, "variables have been plotted.")

hist_for_nums(df, num_cols)
```


Pclass
500
400
300
200
100
0
1.00
1.25
1.50
1.75
2.00
2.25
2.50
2.75
3.00
Pclass


Age
175
150
125
100
75
50
25
0
0
10
20
30
40
50
60
70
80
Age


SibSp
600
500
400
300
200
100
0
0
1
2
3
4
5
6
7
8
SibSp


Parch
700
600
500
400
300
200
100
0
0
1
2
3
4
5
6
Parch




```
5 variables have been plotted.
```

4.4 Target Analysis

[45]:
```python
df.head()
```

[45]:
```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

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

   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
```

```
3      0             113803   53.1000   C123         S
4      0             373450    8.0500    NaN         S
```

4.4.1 Target Analysis According to Categorical Variables

[46]:
```
df["Survived"].value_counts()
```

[46]:
```
0    549
1    342
Name: Survived, dtype: int64
```

[47]:
```
df.groupby("Sex")["Survived"].mean()
```

[47]:
```
Sex
female    0.742038
male      0.188908
Name: Survived, dtype: float64
```

[49]:
```
def target_summary_with_cat(data, target):
    cat_names = [col for col in data.columns if len(data[col].unique()) < 10␣
 ↪and col not in target]
    for var in cat_names:
        print(pd.DataFrame({"TARGET_MEAN": data.groupby(var)[target].mean()}),␣
 ↪end="\n\n\n")

target_summary_with_cat(df, "Survived")
```

```
        TARGET_MEAN
Pclass
1          0.629630
2          0.472826
3          0.242363


        TARGET_MEAN
Sex
female     0.742038
male       0.188908


       TARGET_MEAN
SibSp
0         0.345395
1         0.535885
2         0.464286
3         0.250000
4         0.166667
5         0.000000
```

```
8          0.000000


       TARGET_MEAN
Parch
0          0.343658
1          0.550847
2          0.500000
3          0.600000
4          0.000000
5          0.200000
6          0.000000


          TARGET_MEAN
Embarked
C             0.553571
Q             0.389610
S             0.336957
```

4.4.2 Target Analysis According to Numerical Variables

[50]:
```python
df.groupby("Survived").agg({"Age": np.mean})
```

[50]:
```
                Age
Survived
0         30.626179
1         28.343690
```

[51]:
```python
def target_summary_with_nums(data, target):
    num_names = [col for col in data.columns if len(data[col].unique()) > 5
                 and df[col].dtypes != 'O'
                 and col not in target
                 and col not in "PassengerId"]

    for var in num_names:
        print(df.groupby(target).agg({var: np.mean}), end="\n\n\n")

target_summary_with_nums(df, "Survived")
```

```
                Age
Survived
0         30.626179
1         28.343690
```

```
             SibSp
Survived
0         0.553734
1         0.473684
```

```
             Parch
Survived
0         0.329690
1         0.464912
```

```
              Fare
Survived
0         22.117887
1         48.395408
```

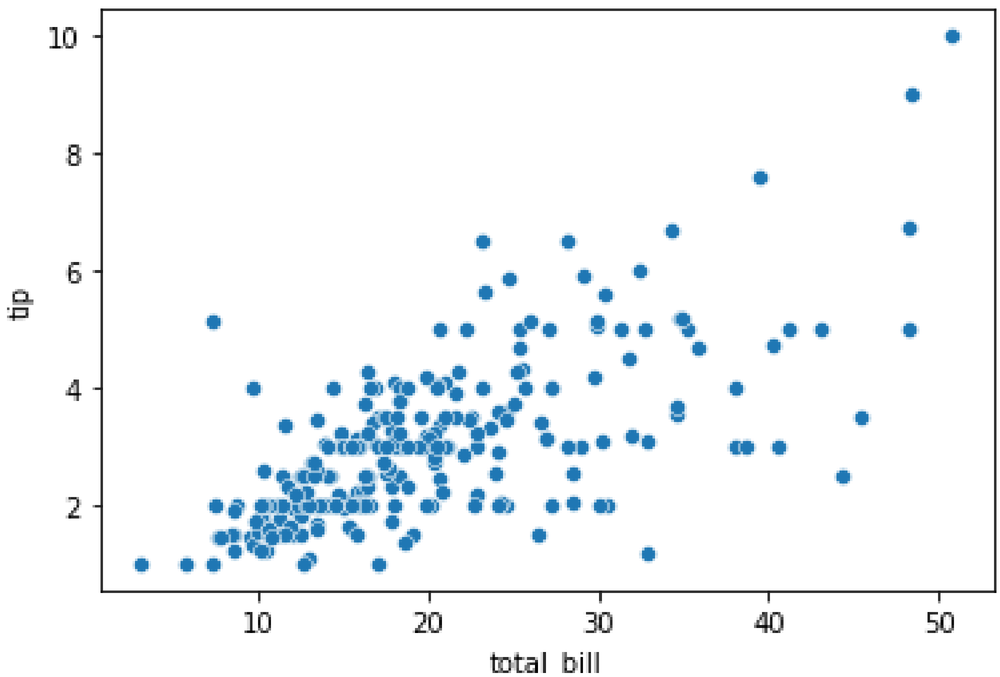
4.4.3 Numerical Variable Analysis with Respect to Each Other

[52]:
```python
df = sns.load_dataset("tips")
df.head()
```

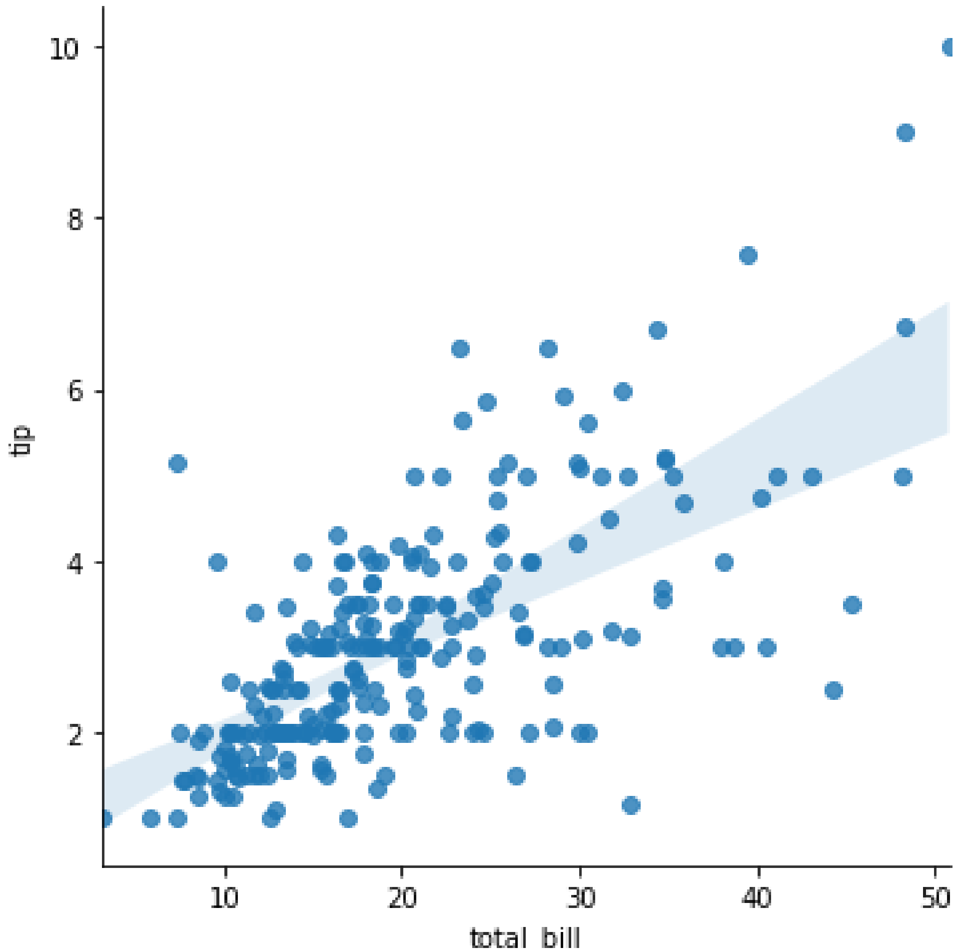
[52]:

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

[53]:
```python
sns.scatterplot(x="total_bill", y="tip", data=df)
plt.show()
```



[54]:
```
sns.lmplot(x="total_bill", y="tip", data=df)
plt.show()
```



[55]:
```
df.corr()
```

[55]:

	total_bill	tip	size
total_bill	1.000000	0.675734	0.598315
tip	0.675734	1.000000	0.489299
size	0.598315	0.489299	1.000000