Data Preparation- Pandas

Series

A Series is very similar to a NumPy array . What differentiates the NumPy array from a Series, is that a Series can have axis labels, meaning it can be indexed by a label, instead of just a number location. It also doesn't need to hold numeric data, it can hold any arbitrary Python Object.

In [124]:

```python
import numpy as np
import pandas as pd
#from pandas import Series, DataFrame
```

In [125]:

```python
Series_obj = pd.Series(np.arange(8), index=['row 1', 'row 2', 'row 3', 'row 4', 'row 5', 'row 6', 'row 7', 'row 8'])
Series_obj
```

Out[125]:

```
row 1    0
row 2    1
row 3    2
row 4    3
row 5    4
row 6    5
row 7    6
row 8    7
dtype: int32
```

Now we want to select an element with the label index of row 7:

In [67]:

```python
Series_obj['row 7']
```

Out[67]:

```
6
```

DataFrame

Create a DataFrame object:

Here is an example of 36 random number in a 6x6 matrices.

In [126]:

```python
np.random.seed(25)
DF_obj = pd.DataFrame(np.random.rand(36).reshape((6,6)),
                   index=['row 1', 'row 2', 'row 3', 'row 4', 'row 5', 'row 6'],
                   columns=['column 1', 'column 2', 'column 3', 'column 4', 'column 5', 'column 6'])
DF_obj
```

Out[126]:

	column 1	column 2	column 3	column 4	column 5	column 6
row 1	0.870124	0.582277	0.278839	0.185911	0.411100	0.117376
row 2	0.684969	0.437611	0.556229	0.367080	0.402366	0.113041
row 3	0.447031	0.585445	0.161985	0.520719	0.326051	0.699186
row 4	0.366395	0.836375	0.481343	0.516502	0.383048	0.997541
row 5	0.514244	0.559053	0.034450	0.719930	0.421004	0.436935
row 6	0.281701	0.900274	0.669612	0.456069	0.289804	0.525819

Reading csv file

In [ ]:

```python
#pd.read_csv
```

In [3]:

```python
import pandas as pd

#How we read in a pandas dataframe. The header=0 means column names are in the first row
df=pd.read_csv('Downloads/mtcars.csv', header=0)
df

```

Out[3]:

	name	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
0	Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
1	Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
2	Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
3	Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
4	Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2
5	Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
6	Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4
7	Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
8	Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2
9	Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4
10	Merc 280C	17.8	6	167.6	123	3.92	3.440	18.90	1	0	4	4
11	Merc 450SE	16.4	8	275.8	180	3.07	4.070	17.40	0	0	3	3
12	Merc 450SL	17.3	8	275.8	180	3.07	3.730	17.60	0	0	3	3
13	Merc 450SLC	15.2	8	275.8	180	3.07	3.780	18.00	0	0	3	3
14	Cadillac Fleetwood	10.4	8	472.0	205	2.93	5.250	17.98	0	0	3	4
15	Lincoln Continental	10.4	8	460.0	215	3.00	5.424	17.82	0	0	3	4
16	Chrysler Imperial	14.7	8	440.0	230	3.23	5.345	17.42	0	0	3	4
17	Fiat 128	32.4	4	78.7	66	4.08	2.200	19.47	1	1	4	1
18	Honda Civic	30.4	4	75.7	52	4.93	1.615	18.52	1	1	4	2
19	Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.90	1	1	4	1
20	Toyota Corona	21.5	4	120.1	97	3.70	2.465	20.01	1	0	3	1
21	Dodge Challenger	15.5	8	318.0	150	2.76	3.520	16.87	0	0	3	2
22	AMC Javelin	15.2	8	304.0	150	3.15	3.435	17.30	0	0	3	2
23	Camaro Z28	13.3	8	350.0	245	3.73	3.840	15.41	0	0	3	4
24	Pontiac Firebird	19.2	8	400.0	175	3.08	3.845	17.05	0	0	3	2
25	Fiat X1-9	27.3	4	79.0	66	4.08	1.935	18.90	1	1	4	1
26	Porsche 914-2	26.0	4	120.3	91	4.43	2.140	16.70	0	1	5	2
27	Lotus Europa	30.4	4	95.1	113	3.77	1.513	16.90	1	1	5	2
28	Ford Pantera L	15.8	8	351.0	264	4.22	3.170	14.50	0	1	5	4
29	Ferrari Dino	19.7	6	145.0	175	3.62	2.770	15.50	0	1	5	6
30	Maserati Bora	15.0	8	301.0	335	3.54	3.570	14.60	0	1	5	8
31	Volvo 142E	21.4	4	121.0	109	4.11	2.780	18.60	1	1	4	2

The head() Method

In [4]:

```python
#The head method returns the first five rows
df.head()
```

Out[4]:

	name	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
0	Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
1	Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
2	Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
3	Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
4	Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2

Basic Features

In [76]:

```
#There are column names
df.columns
```

Out[76]:

```
Index(['name', 'mpg', 'cyl', 'disp', 'hp', 'drat', 'wt', 'qsec', 'vs', 'am',
       'gear', 'carb'],
      dtype='object')
```

In [77]:

```
#And there are row names
list(df.index)
```

Out[77]:

```
[0,
 1,
 2,
 3,
 4,
 5,
 6,
 7,
 8,
 9,
 10,
 11,
 12,
 13,
 14,
 15,
 16,
 17,
 18,
 19,
 20,
 21,
 22,
 23,
 24,
 25,
 26,
 27,
 28,
 29,
 30,
 31]
```

In [79]:

```
#Get the dimensions of the data frame with shape
dimensions = df.shape
dimensions
```

Out[79]:

```
(32, 12)
```

In [80]:

```
#Get the data type of each column
df.dtypes
```

Out[80]:

```
name     object
mpg     float64
cyl       int64
disp    float64
hp        int64
drat    float64
wt      float64
qsec    float64
vs        int64
am        int64
gear      int64
carb      int64
dtype: object
```

In [81]:

```
#We can pick out a column by referencing its name. The result is a series or one dimensional data frame
df['mpg'].head()
```

Out[81]:

```
0    21.0
1    21.0
2    22.8
3    21.4
4    18.7
Name: mpg, dtype: float64
```

In [82]:

```
#You can similarly pick out columns as attributes with the '.'
df.mpg.head()
```

Out[82]:

```
0    21.0
1    21.0
2    22.8
3    21.4
4    18.7
Name: mpg, dtype: float64
```

Note that when we slice a series, the second entry in non-inclusive.

In [83]:

```
#You can pick out multiple columns by specifying a list of column names
name_grade = df[['mpg', 'cyl','hp']].head()
name_grade
```

Out[83]:

	mpg	cyl	hp
0	21.0	6	110
1	21.0	6	110
2	22.8	4	93
3	21.4	6	110
4	18.7	8	175

Slicing and Indexing

We will be using the .loc (just labels) approach. You can also slice with .iloc (just indicies) or .ix (indices and labels).

In [84]:

```
#Let's look at the data
df.head()
```

Out[84]:

	name	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
0	Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
1	Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
2	Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
3	Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
4	Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2

In [85]:

```
#Pick out a single entry
df.loc[3,"name"]
```

Out[85]:

```
'Hornet 4 Drive'
```

In [87]:

```
#Select contiguous rows and columns
df.loc[1:5, "disp":"wt"]
```

Out[87]:

	disp	hp	drat	wt
1	160.0	110	3.90	2.875
2	108.0	93	3.85	2.320
3	258.0	110	3.08	3.215
4	360.0	175	3.15	3.440
5	225.0	105	2.76	3.460

In [88]:

```
#Select none continuguous rows
df.loc[[0,2,4], ["wt","am"]]
```

Out[88]:

	wt	am
0	2.62	1
2	2.32	1
4	3.44	0

Built in Functions

- Useful built in column methods.
- Creating new columns and deleting existing ones.

In [89]:

```
#Read in the data frame
df=pd.read_csv("Downloads/mtcars.csv", header=0)

df.head()
```

Out[89]:

	name	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
0	Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
1	Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
2	Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
3	Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
4	Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2

In [90]:

```
#Compute mean of carb column
avg_final = df["carb"].mean()
avg_final
```

Out[90]:

```
2.8125
```

Creating New Columns

Next, we look at how to create new columns

In [91]:

```
#Create a New Column that is a function of other columns
df["carb_new"] = df["carb"]/2
df.head()
```

Out[91]:

	name	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb	carb_new
0	Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4	2.0
1	Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4	2.0
2	Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1	0.5
3	Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1	0.5
4	Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2	1.0

Deleting Columns (Drop Method)

In [92]:

```
#I can then delete it with the drop method
df.drop(["carb_new"], inplace = True, axis=1)
df.head()
```

Out[92]:

	name	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
0	Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
1	Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
2	Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
3	Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
4	Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2

The inplace argument works as follows:

- inplace = True : The dataframe itself will have the given column(s) deleted.
- inplace = False: Will return a dataframe with the column(s) deleted.

  The axis argument works as follows:
- axis = 1 : delete columns given
- axis = 0 : delete rows given.

  Let's look at an example where we delete rows

In [93]:

```
#Delete rows with index 0 and 2
drop_rows = df.drop([0,2], inplace = False, axis=0)
drop_rows.head()
```

Out[93]:

```
              name   mpg  cyl   disp   hp  drat     wt   qsec  vs  am  gear  carb
1    Mazda RX4 Wag  21.0    6  160.0  110  3.90  2.875  17.02   0   1     4     4
3   Hornet 4 Drive  21.4    6  258.0  110  3.08  3.215  19.44   1   0     3     1
4 Hornet Sportabout  18.7   8  360.0  175  3.15  3.440  17.02   0   0     3     2
5          Valiant  18.1    6  225.0  105  2.76  3.460  20.22   1   0     3     1
6       Duster 360  14.3    8  360.0  245  3.21  3.570  15.84   0   0     3     4
```

Let's have a look at df

In [94]:

```
df.head()
```

Out[94]:

```
              name   mpg  cyl   disp   hp  drat     wt   qsec  vs  am  gear  carb
0        Mazda RX4  21.0    6  160.0  110  3.90  2.620  16.46   0   1     4     4
1    Mazda RX4 Wag  21.0    6  160.0  110  3.90  2.875  17.02   0   1     4     4
2       Datsun 710  22.8    4  108.0   93  3.85  2.320  18.61   1   1     4     1
3   Hornet 4 Drive  21.4    6  258.0  110  3.08  3.215  19.44   1   0     3     1
4 Hornet Sportabout  18.7   8  360.0  175  3.15  3.440  17.02   0   0     3     2
```

Note that df was not changed! This is what happens when you set inplace.

Let's see how we can sort a data frame. The inplace argument has the same affect as the drop method.

In [95]:

```
#Sort the data frame according tothe mpg Column
#By setting inplace= False will just return the sorted dataframe and not chnage df
df.sort_values(by = ["mpg"], inplace =False, ascending=False).head()
```

Out[95]:

```
             name   mpg  cyl  disp   hp  drat     wt   qsec  vs  am  gear  carb
19  Toyota Corolla  33.9    4  71.1   65  4.22  1.835  19.90   1   1     4     1
17        Fiat 128  32.4    4  78.7   66  4.08  2.200  19.47   1   1     4     1
27    Lotus Europa  30.4    4  95.1  113  3.77  1.513  16.90   1   1     5     2
18     Honda Civic  30.4    4  75.7   52  4.93  1.615  18.52   1   1     4     2
25       Fiat X1-9  27.3    4  79.0   66  4.08  1.935  18.90   1   1     4     1
```

Now let's sort by multiple columns, specifying more than one column is essentially specifying a tie break

In [97]:

```
#Sort by Mini Exam 1 and tie breal with Previous Part

result_sorted = df.sort_values(by = ["mpg", "wt"], inplace =False, ascending=False)
result_sorted.head()
```

Out[97]:

```
             name   mpg  cyl  disp   hp  drat     wt   qsec  vs  am  gear  carb
19  Toyota Corolla  33.9    4  71.1   65  4.22  1.835  19.90   1   1     4     1
17        Fiat 128  32.4    4  78.7   66  4.08  2.200  19.47   1   1     4     1
18     Honda Civic  30.4    4  75.7   52  4.93  1.615  18.52   1   1     4     2
27    Lotus Europa  30.4    4  95.1  113  3.77  1.513  16.90   1   1     5     2
25       Fiat X1-9  27.3    4  79.0   66  4.08  1.935  18.90   1   1     4     1
```

In this part, we will a collection of important miscellaneous concepts that include:

- Changing columns names
- Combining dataframes
- Understanding the index
- Missing Data

In [99]:

```python
import pandas as pd

#Read in the data frame
df=pd.read_csv("Downloads/mtcars.csv", header=0)

df.head()
```

Out[99]:

	name	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
0	Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
1	Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
2	Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
3	Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
4	Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2

Recall that we can get the column names through the attribute column

In [100]:

```python
#Get the column names
df.columns
```

Out[100]:

```
Index(['name', 'mpg', 'cyl', 'disp', 'hp', 'drat', 'wt', 'qsec', 'vs', 'am',
       'gear', 'carb'],
      dtype='object')
```

Changing Column Names

We can change column names through the rename method

In [101]:

```python
#Change the column names
df.rename(columns={"mpg":"mpg_1", "cyl":"cyl_1"}, inplace=True)

df.head()
```

Out[101]:

	name	mpg_1	cyl_1	disp	hp	drat	wt	qsec	vs	am	gear	carb
0	Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
1	Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
2	Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
3	Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
4	Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2

Concatenation

In [ ]:

```python
#pd.concat
```

Next, we see how to combine or concatenate two (or more) data frames.

In [102]:

```python
#I can combine data frames with concat function
head = df.head()
tail = df.tail()


```

In [103]:

```python
#Have a look at the variable head
head
```

Out[103]:

	name	mpg_1	cyl_1	disp	hp	drat	wt	qsec	vs	am	gear	carb
0	Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
1	Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
2	Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
3	Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
4	Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2

In [104]:

```
#Have a look at the variable head
tail
```

Out[104]:

	name	mpg_1	cyl_1	disp	hp	drat	wt	qsec	vs	am	gear	carb
27	Lotus Europa	30.4	4	95.1	113	3.77	1.513	16.9	1	1	5	2
28	Ford Pantera L	15.8	8	351.0	264	4.22	3.170	14.5	0	1	5	4
29	Ferrari Dino	19.7	6	145.0	175	3.62	2.770	15.5	0	1	5	6
30	Maserati Bora	15.0	8	301.0	335	3.54	3.570	14.6	0	1	5	8
31	Volvo 142E	21.4	4	121.0	109	4.11	2.780	18.6	1	1	4	2

In [105]:

```
#axis=0 says stack them top to bottom. axis =1 stacks side to side
dfConcat = pd.concat([head,tail], axis =0)
dfConcat
```

Out[105]:

	name	mpg_1	cyl_1	disp	hp	drat	wt	qsec	vs	am	gear	carb
0	Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
1	Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
2	Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
3	Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
4	Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2
27	Lotus Europa	30.4	4	95.1	113	3.77	1.513	16.90	1	1	5	2
28	Ford Pantera L	15.8	8	351.0	264	4.22	3.170	14.50	0	1	5	4
29	Ferrari Dino	19.7	6	145.0	175	3.62	2.770	15.50	0	1	5	6
30	Maserati Bora	15.0	8	301.0	335	3.54	3.570	14.60	0	1	5	8
31	Volvo 142E	21.4	4	121.0	109	4.11	2.780	18.60	1	1	4	2

Handling Missing Data

Missing data is common in most data analysis applications. You have a number of options for filtering out missing data. One option is doing it by hand or you can use the *dropna* method.

With dataframes objects, things get a little more complex. You may want to drop rows or columns which are all NA or just those containing any NAs. *dropna* by default drops any row containing a missing value.

In [106]:

```
#Here we have two pieces of missing data
df_missing = pd.read_csv("Downloads/mtcars_missing.csv")
df_missing
```

Out[106]:

	name	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
0	Mazda RX4	21.0	6	160	110.0	3.90	2.620	16.46	0	1	4	4
1	Mazda RX4 Wag	NaN	6	160	110.0	3.90	2.875	17.02	0	1	4	4
2	Datsun 710	22.8	4	108	93.0	3.85	2.320	18.61	1	1	4	1
3	Hornet 4 Drive	21.4	6	258	NaN	3.08	3.215	19.44	1	0	3	1

The isnull() method returns a series or dataframe of booleans corresponding to whether the particular entries are null or not.

In [107]:

```
#isnull method for a data frame
df_missing.isnull()
```

Out[107]:

	name	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
0	False	False	False	False	False	False	False	False	False	False	False	False
1	False	True	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	False	False
3	False	False	False	False	True	False	False	False	False	False	False	False

We can make sure they are all read in as NA values using the na_values input when we read in the file

Now lets see how we can change/replace these NA values

In [109]:
```
#Get rid of all rows with an NA
df_missing.dropna(axis=0)
```

Out[109]:

	name	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
0	Mazda RX4	21.0	6	160	110.0	3.90	2.62	16.46	0	1	4	4
2	Datsun 710	22.8	4	108	93.0	3.85	2.32	18.61	1	1	4	1

Rather than filtering ou missing data, you may want to fill in the "holes" in any number of ways. For most purposes, the *fillna* method with a constant relplaces missing values with that value.

In [110]:
```
df_missing.fillna(0)
```

Out[110]:

	name	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
0	Mazda RX4	21.0	6	160	110.0	3.90	2.620	16.46	0	1	4	4
1	Mazda RX4 Wag	0.0	6	160	110.0	3.90	2.875	17.02	0	1	4	4
2	Datsun 710	22.8	4	108	93.0	3.85	2.320	18.61	1	1	4	1
3	Hornet 4 Drive	21.4	6	258	0.0	3.08	3.215	19.44	1	0	3	1

In [112]:
```
#You can pass fillna a dict which gives the replacement value for each column
df_missing.fillna({"mpg":20,"hp":100})
```

Out[112]:

	name	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
0	Mazda RX4	21.0	6	160	110.0	3.90	2.620	16.46	0	1	4	4
1	Mazda RX4 Wag	20.0	6	160	110.0	3.90	2.875	17.02	0	1	4	4
2	Datsun 710	22.8	4	108	93.0	3.85	2.320	18.61	1	1	4	1
3	Hornet 4 Drive	21.4	6	258	100.0	3.08	3.215	19.44	1	0	3	1

With *fillna* you can do lots of things with a little creativity. For example, you might pass the mean of median value of a series.

In [113]:
```
#Replace with mean
df_missing.fillna(df_missing.mean())
```

Out[113]:

	name	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
0	Mazda RX4	21.000000	6	160	110.000000	3.90	2.620	16.46	0	1	4	4
1	Mazda RX4 Wag	21.733333	6	160	110.000000	3.90	2.875	17.02	0	1	4	4
2	Datsun 710	22.800000	4	108	93.000000	3.85	2.320	18.61	1	1	4	1
3	Hornet 4 Drive	21.400000	6	258	104.333333	3.08	3.215	19.44	1	0	3	1

In [ ]:
```

```