**Lesson5 Using Pandas Data Frames** Last Update: Apr 24, 2021

Conventions used in these lessons:

**bold** - headings, keywords, code

*italics* - code syntax

underline - important words

A data frame stores data as rows and columns similar to a spread sheet. Each column has a heading. Each row contains the data for each column heading. We can make the following data frame to store information about cars having a make, model, number of doors , car type and selling price.

| Make | Model | Doors | Price |
|---|---|---|---|
| Ford | Mustang | 2 | 12000 |
| GM | Spitfire | 2 | 34000 |
| Toyota | Yarris | 4 | 26000 |
| Nissan | Sentra | 2 | 18000 |
| Honda | Accord | 2 | 16000 |

To use data frames you first need to import pandas into your python

**import pandas as pd**

You also may need to load in the pandas module into your python

**pip install pandas**

To make the data frame, we first make a python dictionary of the above rows and columns and then store in a data frame. The dictionary **keys** represent a column heading where as the dictionary **values** are represented by a list of column values for each row.

```
cars = {'make':['Ford','GM','Toyota','Nissan', 'Honda'],
        'model':['Mustang','Spitfire','Yarris','Sentra', 'Accord'],
        'doors':[2,2,4,2,2],
        'price':[12000,34000,26000,18000,16000]}
```

We print out the dictionary as follows:

```
print(cars)
```

```
{'make': ['Ford', 'GM', 'Toyota', 'Nissan', 'Honda'],
'model': ['Mustang', 'Spitfire', 'Yarris', 'Sentra', 'Accord'],
'doors': [2, 2, 4, 2, 2],
 'price': [12000, 34000, 26000, 18000, 16000]}
```

Next we load the dictionary of cars into a pandas data frame called **df**

```
df = pd.DataFrame(cars)
```

Then we print out the data frame

```
print(df)
```

```
     make     model  doors  price
0    Ford   Mustang      2  12000
1      GM  Spitfire      2  34000
2  Toyota    Yarris      4  26000
3  Nissan    Sentra      2  18000
4   Honda    Accord      2  16000
```

We now have a data frame based on our cars dictionary. Note each list values in the dictionary is represented by a column row value's in the data frame.

Note: We have automatic indexes 0 to 3 supplied to us. Indexes represent a lookup for each row. Row 0 is index 0. Later on we will specify a column to be an index.

You can make a empty data frame by not supplying a dictionary.

```
df= pd.DataFrame()
print(df)
```

```
Empty DataFrame
Columns: []
Index: []
```

We have empty column names and indexes.

We can then add rows to the data frame like this:

```
df['make']  = ['Ford','GM','Toyota','Nissan', 'Honda']
df['model'] = ['Mustang','Spitfire','Yarris','Sentra', 'Accord']
df['doors'] = [2,2,4,2,2]
df['price'] = [12000,34000,26000,18000,16000]
```

Some data frames can be quite large , you can use **df.head()** to print out the first 5 rows.

```
print(df.head())
```

```
     make     model  doors  price
0    Ford   Mustang      2  12000
1      GM  Spitfire      2  34000
2  Toyota    Yarris      4  26000
3  Nissan    Sentra      2  18000
4   Honda    Accord      2  16000
```

You can also specify the number of rows to print out. Here we specify to print out the first 2 rows.

**print(df.head(2))**

```
   make     model  doors  price
0  Ford   Mustang      2  12000
1    GM  Spitfire      2  34000
```

**Loading a data frame from a csv file**

We can also load a data frame from a csv file cars.csv

cars.csv

```
make,model,doors,price
Ford,Mustang,2,12000
GM,Spitfire,2,34000
Toyota,Yarris,4,26000
Nissan,Sentra,2,18000
Honda,Accord,2,16000
```

To load a pandas data frame from a csv file we use the pandas **read_csv** functions and specify a file name. We load the above csv file called "cars.csv" into our data frame. The csv file must be in the same folder as your py files in order to access it.

**df = pd.read_csv('cars.csv')**

We then print out the data frame.

**print(df)**

```
     make     model  doors  price
0    Ford   Mustang      2  12000
1      GM  Spitfire      2  34000
2  Toyota    Yarris      4  26000
3  Nissan    Sentra      2  18000
4   Honda    Accord      2  16000
```

Note: Again we have automatic indexes 0 to 3 supplied to us. Indexes represent a lookup index for each row. Row 0 is index 0.

**Removing NaN'S from a data frame.**

Many csv file have blank cell entries that become NaN in a data frame. NaN stands for **Not a Number**. To demonstrate NaN's we have loaded the data frame from a csv file with blank entries.

cars2.csv

```
make,model,doors,price
Ford,Mustang,2,12000
GM,Spitfire,,34000
Toyota,Yarris,4,
Nissan,Sentra,2,18000
Honda,Accord,2,16000
```

**df = pd.read_csv('cars2.csv')**

```
     make     model  doors    price
0    Ford   Mustang    2.0  12000.0
1      GM  Spitfire    NaN  34000.0
2  Toyota    Yarris    4.0      NaN
3  Nissan    Sentra    2.0   1800.0
4   Honda    Accord    2.0  16000.0
```

The best thing to do is set all NaN (**Not a Number**) to 0. To change all Nan's to 0 we use the pandas **fillna** function. We need to specify **inplace = True** or else the values will not changed in our data frame.

**df.fillna(0,inplace=True)**

Without specifying **inplace = True** a new data frame is returned instead, you would have to the returned data frame to itself or another one.

**df = df.fillna(0)**

We print out the updated data frame. Notice all NaN's have been changed to 0.

**print(df)**

```
     make     model  doors    price
0    Ford   Mustang    2.0  12000.0
1      GM  Spitfire    0.0  34000.0
2  Toyota    Yarris    4.0      0.0
3  Nissan    Sentra    2.0   1800.0
4   Honda    Accord    2.0  16000.0
```

Note: You can also put NaN's in a dictionary or data frame using numpy **np.NaN**

To continue we reload our data frame with the original data frame.

**df = pd.DataFrame(cars)**
**print(df)**

or you can read from the csv file again.

**df = pd.read_csv('cars.csv')**
**print(df)**

```
     make     model  doors  price
0    Ford   Mustang      2  12000
1      GM  Spitfire      2  34000
2  Toyota    Yarris      4  26000
3  Nissan    Sentra      2  18000
4   Honda    Accord      2  16000
```

**Printing out column names**

Once you got your data frame loaded you can print out the column names indexes by using the data frame **columns** property

**print(df.columns)**

```
Index(['make', 'model', 'doors', 'price'], dtype='object')
```

You can also use **tolist** to just print out the columns names as a list instead.

**print(df.columns.tolist())**

```
['make', 'model', 'doors', 'price']
```

**Printing out indexes**

Once you got your data frame loaded you can print out the row index details by using the data frame **index** property

**print(df.index)**

```
RangeIndex(start=0, stop=5, step=1)
```

You can use **tolist** to print out the row indexes as a list instead.

**print(df.index.tolist())**

```
[0, 1, 2, 3, 4]
```

**Getting number of rows and columns in a Data frame**

You can use the **len** function to get the number of rows in a data frame.

**print(len(df))**

```
5
```

You can also use the python **len** function on the columns list to get the number of columns in a data frame.

**print(len(df.columns))**

```
4
```

**Printing out individual rows**

To print out individual rows or group of rows we use index slicing

*df[start_index=0 : end_index=rows : step=1]*

The default start index is 0, the default end index is the number of rows and a the default step size is 1. Note: The end_index never gets selected.

To print out the first row you can use **print(df[0:1])** or default **print(df[:1])** where the **[:1]** means **[0:1]**

**print(df[0:1])**

```
     make     model  doors  price
0    Ford   Mustang      2  12000
```

To print out a group of rows like row 1 to 3 you can use **print(df[1:4])**

**print(df[1:4])**

```
     make     model  doors  price
1      GM  Spitfire      2  34000
2  Toyota    Yarris      4  26000
3  Nissan    Sentra      2   1800
```

To print out the last row we use -1 as the start index since you may not know how many rows you have.

**print(df[-1:])**

```
    make   model  doors  price
4  Honda  Accord      2  16000
```

To print all rows except the last one you can use **print(df[:-1])**

**print(df[:-1])**

```
     make     model  doors  price
0    Ford   Mustang      2  12000
1      GM  Spitfire      2  34000
2  Toyota    Yarris      4  26000
3  Nissan    Sentra      2  18000
```

You can even reverse a data frame using df[::-1])

**print(df[::-1])**

```
     make     model  doors  price
4   Honda    Accord      2  16000
3  Nissan    Sentra      2  18000
2  Toyota    Yarris      4  26000
1      GM  Spitfire      2  34000
0    Ford   Mustang      2  12000
```

**Accessing individual columns**

You can access column data by the column name. The column names are case sensitive. To access a column you specify the column name in quotes enclosed by square brackets as follows:

*df['column_name']* ( return data frame as a series)

The data is returned as a **series** of values rather than as a data frame. A pandas **series** contains the row index as the key and associated column row values for that column row. A series is similar to a 1 dimensional array, only having rows and 1 column, where as a data frame is similar to a 2 dimensional array having rows and many columns.

Here we return the **make** column as a series.

```
print(df['make'])
```

```
0      Ford
1        GM
2    Toyota
3    Nissan
4     Honda
Name: make, dtype: object
```

**dtype** refers to the data type of the column in our case it's a object representing string object.

You can cases individual values by specifying a row index

To access the make value first row in the column use row index 0 like:

```
value = df['make'][0]
print(value)
```

```
Ford
```

**to do:**

Access one of the other column values using a row index.

For convenience you can also convert the column values to a list using **tolist** function.

```
print(df['make'].tolist())
```

```
['Ford', 'GM', 'Toyota', 'Nissan', 'Honda']
```

To return a **data frame** instead of a **series** you specify the column name enclosed in **double** square brackets as follows:

*df[['column_name']]* (return column as a dataframe)

Here we return a data frame just with the **make** column

**print(df[['make']])**

```
     make
0    Ford
1      GM
2  Toyota
3  Nissan
4   Honda
```

You can also return a data frame with multiple columns like 'make' and 'model'.

**print(df[['make','model']])**

```
     make     model
0    Ford   Mustang
1      GM  Spitfire
2  Toyota    Yarris
3  Nissan    Sentra
4   Honda    Accord
```

**Locating rows using loc and iloc**

**loc** uses labels to locate rows and column data

**iloc** uses integer indexes to locate rows and columns data.

We will first work with **iloc**.

With **iloc** you can specify either row or columns or both. **iloc** has both a row selection and a column selection. You can use **row selection**, or **column selection** or both.

*df.iloc[<row selection>,<column selection>]*

Where row selection can be **start_index : end_index** and where column selection can also be **start_index: end_index**

You can visualize row selection and column selection like this:

*df.iloc[start_index : end_index, start_index : end_index]*

Data can be returned as a series of values or as a data frame.

When a single row is selected and enclosed by a single bracket than data is returned as a series for the column values in the specified row index.

**Returning a Series using iloc**

Here we return row 0 as a series of column values for the specified row 0.

**print(df.iloc[0])**

```
make        Ford
model    Mustang
doors          2
price      12000
Name: 0, dtype: object
```

**To do:**

Print the value of the doors using a column index. Convert the series to a list using tolist() like this:

**print(df.iloc[0].tolist())**

**Returning a DataFrame using iloc**

If double brackets are used then data is returned in a data frame for a specified row. Here we return row 0 as a data frame.

**print(df.iloc[[0]])**

```
   make    model  doors  price
0  Ford  Mustang      2  12000
```

**Selecting columns**

We can use the column selection to return a data frame for specified column(s)

*df.iloc[:,<column selection>]*

*df.iloc[:, start_index : end_index]*

where ' :' means all rows

We now select just the **make** and **model** columns

**print(df.iloc[:,0:2])**

```
     make     model
0    Ford   Mustang
1      GM  Spitfire
2  Toyota    Yarris
3  Nissan    Sentra
4   Honda    Accord
```

Note we use ':' to specify all rows.

**Specifying rows and columns**

We can use the **row selection** and **column selection** to specify desired rows and columns.

*df.iloc[<row selection>,<column selection>]*

*df.iloc[start_index : end_index, start_index : end_index]*

Here we return rows 1 and 2 of the **make** and **model and** columns 0 and 1 as a data frame. Remember you always get row index -1 and column index – 1 in the specified range.

**print(df.iloc[1:3,0:2])**

```
     make     model
1      GM  Spitfire
2  Toyota    Yarris
```

**using loc**

**specifying columns using loc**

We can use **loc** to specify **columns** by name.

*df.loc[<row selection>,<column selection>]*

*df.loc[start_index : end_index, [column_name_list]]*

Here we specify '**make**' and '**model**' columns using **loc**. We print out all rows of the data frame for columns 'model','make' specifying : for the rows.

**print(df.loc[:,['model','make']])**

```
      model    make
0   Mustang    Ford
1  Spitfire      GM
2    Yarris  Toyota
3    Sentra  Nissan
4    Accord   Honda
```

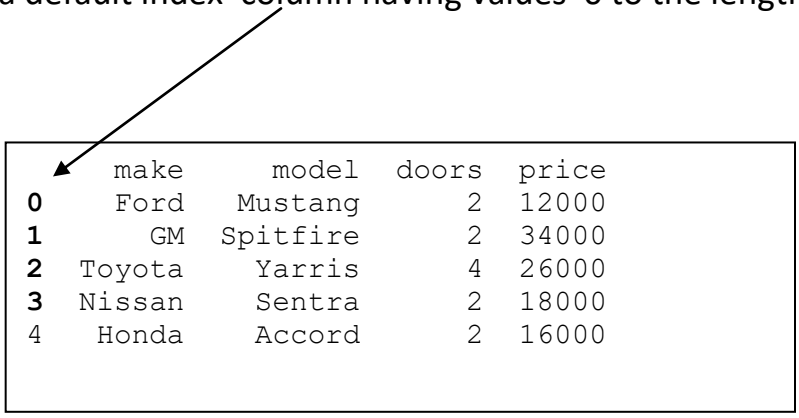
You can specifying rows using slices. Here we select rows 1 to 3 and columns

```
# specifying rows by index and columns by labels
print(df.loc[1:3,['model','make']])
```

```
      model    make
1  Spitfire      GM
2     Yarris  Toyota
3     Sentra  Nissan
```

## Making an index column

An index column lets use specify rows by a index column value. All pandas data frames have a default index column having values 0 to the length of the data frame-1 .

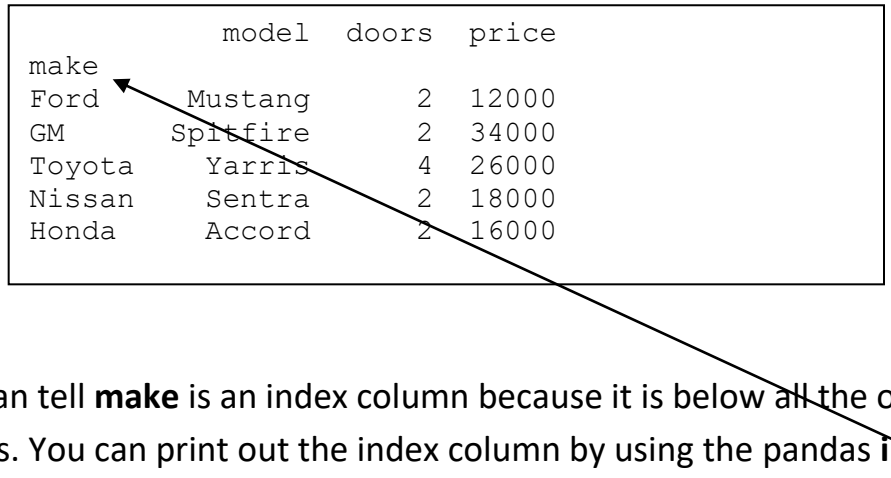


| | make | model | doors | price |
|---|---|---|---|---|
| **0** | Ford | Mustang | 2 | 12000 |
| **1** | GM | Spitfire | 2 | 34000 |
| **2** | Toyota | Yarris | 4 | 26000 |
| **3** | Nissan | Sentra | 2 | 18000 |
| 4 | Honda | Accord | 2 | 16000 |

## Making an index column inplace

You make a index column by using the pandas **set_index** function and the column name. We set the index to the **'make'** column using the **set_index** function. We use **inplace = True** to do this in place.

```
df.set_index("make", inplace=True)
print(df)
```



```
           model  doors  price
make
Ford      Mustang      2  12000
GM       Spitfire      2  34000
Toyota     Yarris      4  26000
Nissan     Sentra      2  18000
Honda      Accord      2  16000
```

You can tell **make** is an index column because it is below all the other column names. You can print out the index column by using the pandas **index** property

**print(df.index)**

```
Index(['Ford', 'GM', 'Toyota', 'Nissan', 'Honda'], dtype='object', name='make')
```

**Read in a csv file and set index column**

You can also read in a csv file and set the index column at the same time using the **index_col** parameter

**df = pd.read_csv('cars.csv', index_col=0)**
**print(df)**

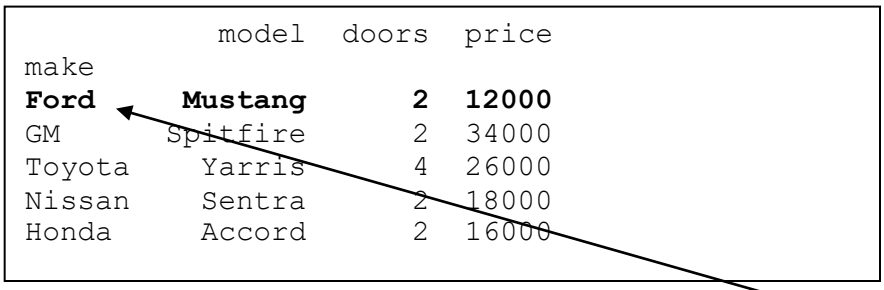
```
           model  doors  price
make
Ford      Mustang      2  12000
GM       Spitfire      2  34000
Toyota     Yarris      4  26000
Nissan     Sentra      2  18000
Honda      Accord      2  16000
```

**Specifying rows using loc**

To specify a row using **loc,** you need to use a **label index** column or to use **loc** as **a lookup condition** looking for a column data value. We first locate rows using **loc and** using the index column label value.

**using loc to locate rows using a index column label**

Using an index column label is convenient because you can locate rows by a certain label value.



```
           model  doors  price
make
Ford     Mustang      2  12000
GM       Spitfire     2  34000
Toyota    Yarris      4  26000
Nissan    Sentra      2  18000
Honda     Accord      2  16000
```

We locate the row having the make index column label value 'Ford', where the index column is **make** and the label value is **'Ford'**

**Lookup a row using index column and return as a Series**

*df.loc[row label]* (as a Series)

**print(df.loc['Ford'])**

```
model    Mustang
doors          2
price      12000
Name: Ford, dtype: object
```

Note: the row Ford is return as a series of values. Notice we no longer have the 'make' returned because the make column is now the index column.

**Lookup a row using index column and return as a DataFrame**

If we enclose in square brackets then data is returned as a data frame.

*df.loc[[row label]]* (as a Data Frame)

**print(df.loc[['Ford']])**

```
        model  doors  price
make
Ford  Mustang      2  12000
         model  doors  price
```

**Looking up more than 1 row using a index column**

*df.loc[start_row_label: end_row_label]* (as a Data Frame)

**print(df.loc['Ford':'GM'])**

```
         model  doors  price
make
Ford    Mustang      2  12000
GM     Spitfire      2  34000
```

We have located locate 'Ford' and 'GM' cars.

Note: 2 or more rows are returned as a data frame where a single row is returned as a series of values.

**Specifing columns when looking up rows by index**

*df.loc[start_row_label: end_row_label.,[column_list]]*

Columns are specified in a list like this:

*[column_name, column_name]*

You need to put the column names in a list.

Here we locate Ford and GM row with model and price

**print(df.loc['Ford':'GM',['model','price']])**

```
         model  price
make
Ford    Mustang  12000
GM     Spitfire  34000
```

Note: The inner square brackets [ ] is a list of column names.

**Specifying a range of columns**

You can also use slicing to specify a range of column names like this

*df.loc[start_row_label: end_row_label,start_column:end_column]*

*column_name : column_name*

Here we locate Ford and GM row with model and price

**print(df.loc['Ford':'GM','model':'price'])**

```
         model  doors  price
make
Ford   Mustang      2  12000
GM    Spitfire      2  34000
```

Note: We do not include inner square brackets because we are specifying a continuous range of column names mot individual specified column names.

If you just specify one column name you will get a series returned.

*df.loc[start_row_label: end_row_label,column]* (as a Series)

**print(df.loc['Ford':'GM','model'])**

```
make
Ford     Mustang
GM      Spitfire
Name: model, dtype: object
```

If you enclose the column name in square brackets **['model']** a DataFrame is returned because you a specifying a column name in a list.

*df.loc[start_row_label: end_row_label,[column]]* (as a Data Frame)

**print(df.loc['Ford':'GM',['model']])**

```
         model
make
Ford    Mustang
GM     Spitfire
```

**Using loc with conditions**

It maybe easier to just state a column and a value to look for rather than change data frame to use indexes instead. We can look up a value in a column like this:

*df.loc.df ['column_name'] == value*

We first need to reload our data frame to restore it back to default integer indexes rather than label index.

You can use you cars dictionary like this:

**df= pd.DataFrame(cars)**
**print(df)**

Or reload from the cars.csv file like this:

**df = pd.read_csv('cars.csv')**
**print(df)**

```
     make     model  doors  price
0    Ford   Mustang      2  12000
1      GM  Spitfire      2  34000
2  Toyota    Yarris      4  26000
3  Nissan    Sentra      2  18000
4   Honda    Accord      2  16000
```

Here we look up the row having the column make ‘Ford’

```
result = df.loc[df['make']=='Ford']
print(result)
```

```
   make    model  doors  price
0  Ford  Mustang      2  12000
```

Our result above is return as a data frame.

Conditions work as a two step process.

```
df['make']=='Ford'
print(df['make']=='Ford')
```

The condition returns a series of **True** and **False** values for the values you are looking for.

```
0     True
1    False
2    False
3    False
4    False
Name: make, dtype: bool
```

Then we feed the **True** and **False** value to **loc** to locate the rows in the specified column that has the **True** values.

```
result = df.loc[df['make']=='Ford']
print(result)
```

```
   make    model  doors  price
0  Ford  Mustang      2  12000
```

In this situation a data frame is returned where a True value is found in the **result** series**.**

You can get the individual values from the data frame column returned as a one item series like this:

```
print(result['make'])
```

```
0    Ford
Name: make, dtype: object
```

You would get the value using the index 0 of the series(we only have 1 value so only 1 index) like this:

```
print(result['make'][0])
```

```
Ford
```

A one step solution would be like this:

```
print(df.loc[df['make']=='Ford']['make'][0])
```

```
Ford
```

Here is another example using the 'model' column:

```
print(result['model'])
```

```
0    Mustang
Name: model, dtype: object
```

You would get the value using the index 0 of the sries (we only have 1 value so only 1 index) like this:

```
print(result['model'][0])
```

```
Mustang
```

A one step solution would be like this:

```
print(df.loc[df['model']=='Mustang']['model'][0])
```

```
Mustang
```

**Accessing a cell using iloc**

You can access a particular cell in the data frame using **iloc,** the row index and the column name. When 1 row is returned the row index is 0. A scalar value is returned. A scalar value is a single value.

```
x = result.iloc[0][ 'model']
print(x)
```

```
Mustang
```

If more than 1 column is specified a **Series** is returned rather than a scalar value.

```
x = result.iloc[0][['model','doors']]
print(x)
```

```
model    Mustang
doors          2
Name: 0, dtype: object
```

In this situation you can specify the cell you want by the series row index or label column name to get a scalar value.

**print(x[0])**
**print(x[1])**

```
Mustang
2
```

**print(x['model'])**
**print(x['doors'])**

```
Mustang
2
```

You can look up more than 1 condition by using the & (AND) logical operator. In this situation each condition must be enclosed in round brackets.

**print(df.loc[(df['make']=='Ford') & (df['model']=='Mustang')])**

```
   make    model  doors  price
0  Ford  Mustang      2  12000
```

Looking for multiple conditions comes in handy when you have many same values in the same column and you use a second condition to get specific rows

**When more than 1 row is returned for a lookup condition**

If more than 1 row is returned  for the condition lookup then you need to specify which row  and column you want to access the cell data. Here we get many rows are returned for cars with 2 doors.

```
result = df.loc[df['doors']==2]
print(result)
```

```
     make     model  doors  price
0    Ford   Mustang      2  12000
1      GM  Spitfire      2  34000
3  Nissan    Sentra      2  18000
4   Honda    Accord      2  16000
```

We have 4 rows returned with cars having doors of value 2

We can the model value from row index 0 using **iloc**:

```
x = result.iloc[0]['model']
print(x)
```

```
Mustang
```

We can the model value from row index 1 using **iloc**:

```
x = result.iloc[1]['model']
print(x)
```

```
Spitfire
```

We can get more than 1 column value using **iloc** returned as a **series**:

```
x = result.iloc[1]['make','model']
print(x)
```

```
make       GM
model   Spitfire
Name: 1, dtype: object
```

You can now get individual values of the series using indexes:

```
x = result.iloc[1][['make','model']][0]
print(x)
```

```
GM
```

```
x = result.iloc[1][['make','model']][1]
print(x)
```

```
Spitfire
```

**to do**

Print out the model for rows index 2 and 3.
Print out the price for row index 0 and 1.

**looking up more than 1 column**

Next we look up 'Ford' from the **make** column and specifying the columns **model** and **price** to be displayed

```
print(df.loc[df['make']=='Ford',['model','price']])
```

```
     model  price
0  Mustang  12000
```

Our result is returned as a data frame.

We can also lookup more than 1 car like ‘Ford’ and ‘GM’. In this case we use the **isin** function. Here we lookup  look up ‘Ford’ and ‘GM’ in the **make** column.

**print(df.loc[df['make'].isin(['Ford','GM']),['model','price']])**

```
      model  price
0   Mustang  12000
1  Spitfire  34000
```

**isin** is for Rows

If you want to see the **make** column  in the printout you also need to specify in the column selector as well.

**print(df.loc[df['make'].isin(['Ford','GM']),['make','model','price']])**

```
   make     model  price
0  Ford   Mustang  12000
1    GM  Spitfire  34000
```

**PANDAS HOMEWORK**

**Question 1**

Make a csv file or a dictionary of animals. Animals  have a type like cat, dog or lion, and they have a name’s like ‘fluffy’, ‘rover’ or ‘tony’. They make sounds like ‘purr’, ‘bark’ and ‘roar’. You should have data frame columns: ‘type’, ‘name’ and ‘sound’. Load your csv file or dictionary into a data  frame.

Print out the animal types on the screen.
Then ask the user to type in their favorite animal from the list.

Next print out the names of the animals.

Ask the user  what is the name of the animal.

Then look up the name of the animal selected.
If they are correct  print out the sounds of the animals.

Then look up the sound of the animal selected.
Ask the user what sound  the animal makes.

If they are correct tell them congratulations .
If they are wrong let them try again.

Hints:

Access a column as a series:
**df['column_name ']**

Access a column as a list:
**df['column_name '] .tolist()**

Get a individual  row value in the series column:
**df['column_name '][row_index]**

Access a column as a data frame:
**df['column_name ']**

Get a individual row value in the data frame column:
**df['column_name '].iloc[row_index**]

Access a row as a series:
**df.iloc[row_index]**

Get a individual column value in the  row as a series:
**df.iloc[row_index][column_index]**

Access a row as a data frame:
**df.iloc[[row_index]]**

Access a row and column of a data frame:
**df.iloc[row_index, column_index]**

Access a row where a column has a certain value as a DataFrame
**df.loc[df['column_name]==value]**

Access a column in a row where a column has a certain value as a series
**df.loc[df['column_name]==value][‘column_name2’]**

Access a column in a row where a column has a certain value as a series
**df.loc[df['column_name]==value][‘column_name2’].iloc[0]**

or:

Access a column in a row where a column has a certain value as a series
**df.loc[df['column_name]==value] .iloc[0] [‘column_name2’]**

You should get something like this:

['cat', 'dog', 'lion']
what is your favourite animal? lion
['fluffy', 'rover', 'tony']
what is the name of the lion? tony
you are correct!
['purr', 'bark', 'roar']
what sound does a lion make ? roar
you are correct

**Question 2**

Repeat the above but make the ‘type’ column to be a index column**.**
Use **tolist** to print out the column without the index column.

Hints:

Print out index names as a list:

**print(df.index.tolist())**

Get the value of a column by specifying a row index label

**value = df.loc[‘index_label]’[‘column_name’]**

or

**sound = df.loc[df.index == index_label]['column_name'].iloc[0]**

You should get something like this:

['cat', 'dog', 'lion']
what is your favourite animal? dog
['fluffy', 'rover', 'tony']
what is the name of the dog? rover
you are correct!
['purr', 'bark', 'roar']
what sound does a dog make ? bark
you are correct

Call your python file pandas_homework.py

End