Lesson 4 Using Pandas Series Last Update: Apr 9, 2021

A Pandas **Series** is a one dimensional labelled array. This means each value in the array has an associated label that can be used as an index to look up data values in the array.

Index	data
0	'cat'
1	'dog'
2	'Tiger'
3	'lion'
4	'zebra'

The Series can hold any data type, and the label axes is called an index. The label index can be numeric or text. The default label index is numeric. You can get and set values by using the index label.

To make a series you first must import pandas

```
import pandas as pd
```

You may have to install pandas using your python shell

```
pip install pandas
```

Using pandas you can use the **Series** function to make the series

We now make a series from a list of values.

```
import pandas as pd
values = [1,2,3,4,5,6,7,8,9,10]
series1 = pd.Series (data=values)
print(series1)
```

Our output is a series of values 1 to 10 with automatic default indexes of 0 to 9. Indexes in a pandas series start at index 0

```
0      1
1      2
2      3
3      4
4      5
5      6
6      7
7      8
8      9
9     10
dtype: int64
```

dtype refers to the data type of the series, in our case it is an int64 (64 bit integer number).

Retrieving a value from a series at a specified index

```
x = series1[0]
print(x)
```

```
1
```

Assigning a value to a series at a specified index

```
series1[0] = 42
print(series1)
```

```
0   42
1    2
2    3
3    4
4    5
5    6
6    7
7    8
8    9
9   10
dtype: int64
```

To do:

Make a pandas series from a list of your favorite numbers, then assign one of the values from the end index to the start index. Print out the series before and after you made the assignment.

Assigning labels to a series

Labels names allow you to look up a value by a label name, in our example we use the letters 'a' to 'j' as label name indexes

```
values = [1,2,3,4,5,6,7,8,9,10]
names = ['a','b','c','d','e','f','g','h','i','j']
series2 = pd.Series(data=values,index=names)
print(series2)
```

```
a      1
b      2
c      3
d      4
e      5
f      6
g      7
h      8
i      9
j     10
dtype: int64
```

We can now retrieve a value by a label name rather than an index number. This should be more convenient for you.

```
x = series2['a']
print(x)
```

```
1
```

We can now assigning a new value to a series using a label name index.

```
series2['a'] = 42
print(series2)
```

```
a    42
b     2
c     3
d     4
e     5
f     6
g     7
h     8
i     9
j    10
dtype: int64
```

To do

Make a pandas series from a list of your favorite numbers and use label names like a person's name, animals or fruits. Then assign one of the values from another label index. Print out the series before and after you made the assignment.

Printing out rows

Printing out first 5 rows of a series:

```
print(series2.head())
```

```
a    42
b     2
c     3
d     4
e     5
dtype: int64
```

Specifying number of rows to print out from a series:

print(series2.head(8))

```
a    42
b     2
c     3
d     4
e     5
f     6
g     7
h     8
dtype: int64
```

Slicing

Slicing allows you which rows to access.

Series_name[start_index : end_index : step_size]

The **start_index** defaults to 0, **end_index** defaults to length of series and **step_size** is defaults to 1

Here we print out row indexes 2 to 7

print(series2[2:7])

```
c    3
d    4
e    5
f    6
g    7
dtype: int64
```

Alternately you can use the **iloc** function instead iloc means index location

print(series2.iloc[2:7])

```
c  3
d  4
e  5
f  6
g  7
dtype: int64
```

Iloc works on index numbers

We can also slice by label names when you have assigned labels.

print(series2['c':'g'])

```
c    3
d    4
e    5
f    6
g    7
dtype: int64
```

Alternatively we can use the **loc** function loc means location on labels.

print(series2.loc['c':'g'])

```
c    3
d    4
e    5
f    6
g    7
dtype: int64
```

loc works on label names when you have assigned them where as **iloc** works on numeric indexes.

More Slicing examples:

Print first Rows

print(series2[:4])

Print last starting from index 4

print(series2[4:])

print all rows except last

print(series2[:-1])

The -1 means go back one index from last element index

reverse a series

```
print(series2[::-1])
```

-1 means step backwards from last element index to first index (reverses list)

To do

Try all the above slicing examples

get list of keys from series

```
print(series2.keys().tolist())
```

```
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
```

get list of values from series

```
print(series2.tolist())
```

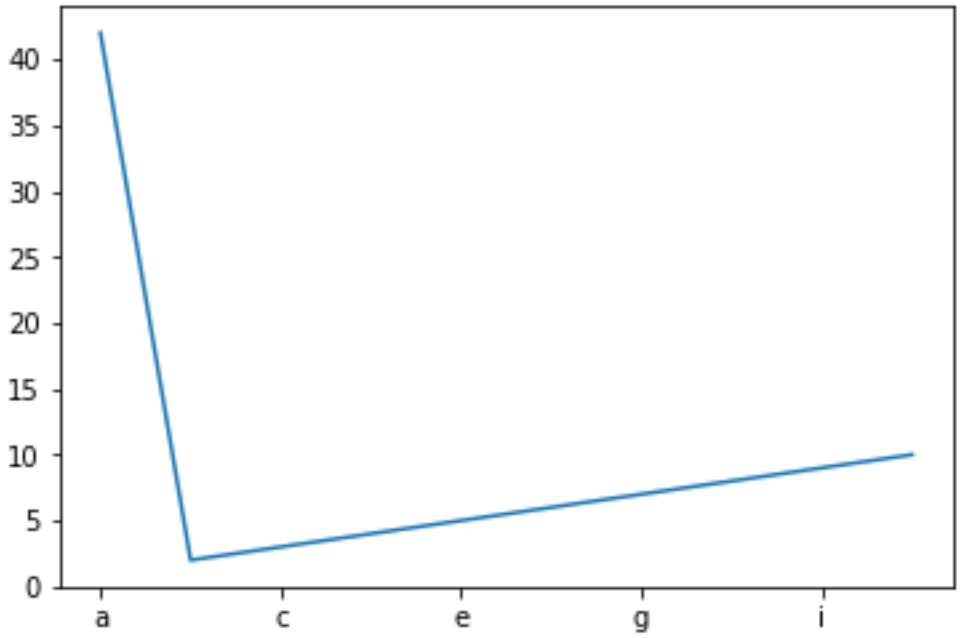
```
[42, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Plotting a series

We use **matplotlib.pyplot** for plotting plots.

```
import matplotlib.pyplot as plt

series2.plot()
plt.show()
```



Read series from a csv file:

We first read in a series from a csv file without a index column.

cars_series.csv

Counts
6
5
10
12
7
8
6

To read a series from a csv file we specify the csv file name and the column to be our values (counts)

```
series3 = pd.read_csv('cars_series.csv')['counts']
print(series3)
```

```
0     6
1     5
2    10
3    12
4     7
5     8
6     6
Name: counts, dtype:
int64
```

Print series as a list

print(series3.tolist())

```
[6, 5, 10, 12, 7, 8, 6]
```

We now read in a csv file that has a column index.

cars2_series.csv

```
make,counts
Honda,6
Ford,5
Toyota,10
Datsun,12
GM,7
Lexus,8
BMW,6
```

To read a series from a csv file with a column index, we specify the csv file name (cars2_series.csv), the column index (make) and the column to be our values (counts)

series3 = pd.read_csv('cars2_series.csv',index_col=0)['counts']
print(series3)

```
Make
Honda     6
Ford      5
Toyota    10
Datsun    12
GM        7
Lexus     8
BMW       6
Name: counts, dtype: int64
```

We now print out the labels as a list

print(series3.keys().tolist())

```
['Honda', 'Ford', 'Toyota', 'Datsun', 'GM', 'Lexus', 'BMW']
```

We now print out the individual data as a list

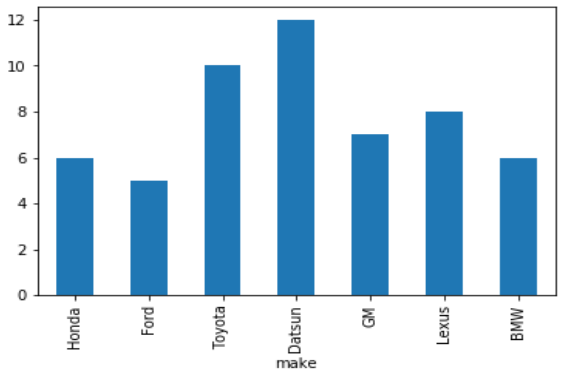
print(series3.tolist())

```
[6, 5, 10, 12, 7, 8, 6]
```

series3.plot
plt.show()

We now plot this series as a bar chart:

series3.plot.bar()
plt.show()



Time series

A time series uses index that are python time stamps using the python **datetime** module

```
from datetime import datetime
```

You make a python **datetime** object by specifying the year month and day.

```
date = datetime(year=2015, month=7, day=4)
print(date)
```

```
2020-07-04 00:00:00
```

You can also specify hours minute second and optional nanoseconds.

```
date = datetime(year=2020, month=7, day=4, hour=6, minute=34, second=26)
print(date)
```

```
2020-07-04 06:34:26
```

You do not need to worry too much about using the datetime module directly because pandas does all the work for you. But you should know all about it to understand how pandas uses it.

For our series we can make an array of date times index and some corresponding data values.

```
index = pd.DatetimeIndex(['2020-07-04', '2020-07-05',
                          '2020-07-06', '2020-07-07',
                          '2020-07-08', '2020-08-09'])

data = [10, 11, 22, 13, 44, 25]

series4 = pd.Series(data=data,index=index)

print(series4)
```

```
2020-07-04    10
2020-07-05    11
2020-07-06    22
2020-07-07    13
2020-07-08    44
2020-08-09    25
dtype: int64
```

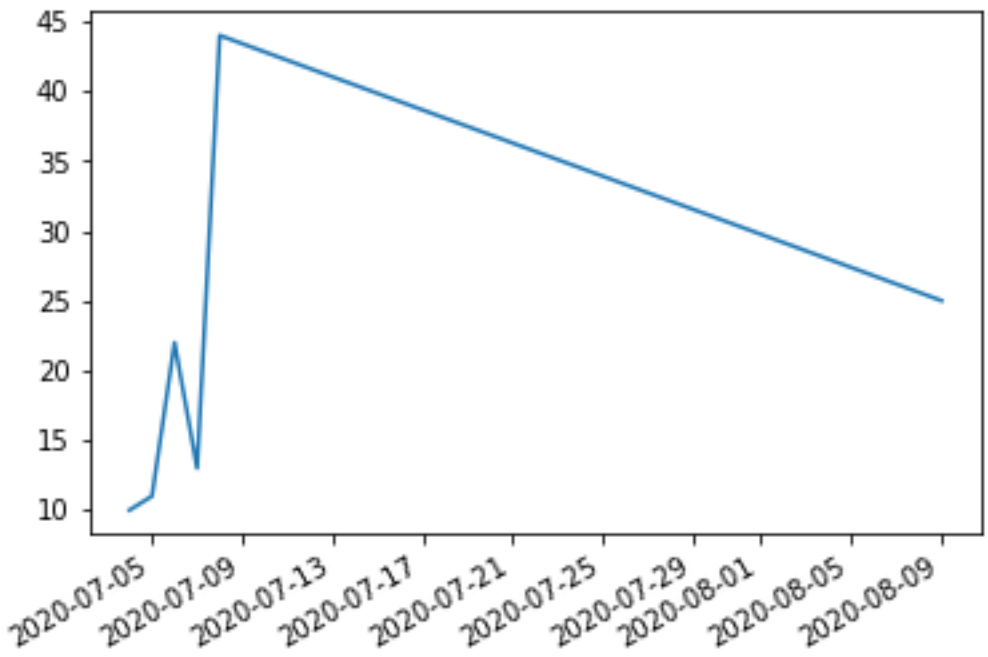
Note: The date time strings key indexes have already automatically been converted to **datetime** objects when you use the pandas **DatetimeIndex** function.

```
print(series4.keys())
```

```
DatetimeIndex(['2020-07-04', '2020-07-05', '2020-07-06',
'2020-07-07','2020-07-08', '2020-08-09'],
dtype='datetime64[ns]', freq=None)
```

Plotting the date time series:

```
series4.plot()
plt.show()
```



slicing time series

We can specify a range of dates to print out using data index slicing. Here we print out all the dates between **'2020-07-05' and '2020-07-08'.**

```
print(series4['2020-07-05':'2020-07-08'])
```

```
2020-07-05    11
2020-07-06    22
2020-07-07    13
2020-07-08    44
dtype: int64
```

Generating time stamp indexes

We generating time stamp indexes by specifying a **start date** time and an **end date** time and a **time span** frequency Here are the available time span frequencies:

Code	Description
D	Calendar day
W	Weekly
M	Month end
Q	Quarter end
A	Year end
H	Hours
T	Minutes
S	Seconds
L	Milliseconds
U	Microseconds
N	Nanoseconds

We now generate a date range from '2020-07-04' to '2020-07-09' with daily intervals 'D'.

```
timestamps=pd.date_range(start='2020-07-04',end= '2020-07-09', freq='D')
print(timestamps)
```

```
DatetimeIndex(['2020-07-04', '2020-07-05', '2020-
07-06', '2020-07-07', '2020-07-08', '2020-07-09'],
dtype='datetime64[ns]', freq='D')
```

We can also specify a start date and the number of periods.

```
timestamps=pd.date_range(start='2020-07-04', periods=6, freq='D')
print(timestamps)
```

```
DatetimeIndex(['2020-07-04', '2020-07-05', '2020-
07-06', '2020-07-07', '2020-07-08', '2020-07-09'],
dtype='datetime64[ns]', freq='D')
```

making a series with a time stamp index

We now make a series using our generated **timestamps** and previous **data** .

```
timestamps=pd.date_range(start='2020-07-04', periods=6, freq='D')
data = [10, 11, 22, 13, 44, 25]
series4 = pd.Series(data=data, index=timestamps)
print(series4)
```

```
2020-07-04    10
2020-07-05    11
2020-07-06    22
2020-07-07    13
2020-07-08    44
2020-07-09    25
Freq: D, dtype: int64
```

Print out items as a list of tuples

```
print(list(series4.items()))
```

```
[(Timestamp('2020-07-04 00:00:00', freq='D'), 10),
(Timestamp('2020-07-05 00:00:00', freq='D'), 11),
(Timestamp('2020-07-06 00:00:00', freq='D'), 22),
(Timestamp('2020-07-07 00:00:00', freq='D'), 13),
(Timestamp('2020-07-08 00:00:00', freq='D'), 44),
(Timestamp('2020-07-09 00:00:00', freq='D'), 25)]
```

print out list of keys

```
print(list(series4.keys()))
```

```
[Timestamp('2020-07-04 00:00:00', freq='D'),
Timestamp('2020-07-05 00:00:00', freq='D'),
Timestamp('2020-07-06 00:00:00', freq='D'),
Timestamp('2020-07-07 00:00:00', freq='D'),
Timestamp('2020-07-08 00:00:00', freq='D'),
Timestamp('2020-07-09 00:00:00', freq='D')]
```

Iterating through a pandas series using the **items** function to print out the list of tuples.ans a date index and a data value count.

```
for index, value in series4.items():

    print(f"Index : {index}, Value : {value}")
```

```
Index : 2020-07-04 00:00:00, Value : 10
Index : 2020-07-05 00:00:00, Value : 11
Index : 2020-07-06 00:00:00, Value : 22
Index : 2020-07-07 00:00:00, Value : 13
Index : 2020-07-08 00:00:00, Value : 44
Index : 2020-07-09 00:00:00, Value : 25
```

Iterating over the elements of a pandas series using **iteritems** to print out the list of **tuples**.

```
for items in series4.iteritems():
    print(items)
```

```
(Timestamp('2020-07-04 00:00:00', freq='D'), 10)
(Timestamp('2020-07-05 00:00:00', freq='D'), 11)
(Timestamp('2020-07-06 00:00:00', freq='D'), 22)
(Timestamp('2020-07-07 00:00:00', freq='D'), 13)
(Timestamp('2020-07-08 00:00:00', freq='D'), 44)
(Timestamp('2020-07-09 00:00:00', freq='D'), 25)
```

Iterating over the elements of a pandas series and changing the values.
We are incrementing the values by 1. We print out the data values before and after on each line.

```
for i in range(len(series4.values)):

    print(series4.keys()[i],end = " : ")
    print(series4[i], end=" ")
    series4[i]+=1
    print(series4[i])
```

```
2020-07-04 00:00:00 : 10 11
2020-07-05 00:00:00 : 11 12
2020-07-06 00:00:00 : 22 23
2020-07-07 00:00:00 : 13 14
2020-07-08 00:00:00 : 44 45
2020-07-09 00:00:00 : 25 26
```

Adding two series together

We use the Pandas series calculation **add** function to add two series together.

We first make two series to add together called series 4 and series 5. We will use the same time stamp and data values as before.

```
timestamps=pd.date_range(start='2020-07-04', periods=6, freq='D')
data = [10, 11, 22, 13, 44, 25]

series4 = pd.Series(data=data, index=timestamps)
series5 = pd.Series(data=data, index=timestamps)
```

We now add the 2 series together using the Pandas series **add** function.

```
series6=series5.add(series4)
print(series6)
```

```
2020-07-04    20
2020-07-05    22
2020-07-06    44
2020-07-07    26
2020-07-08    88
2020-07-09    50
Freq: D, dtype: int64
```

Pandas series calculation methods

Pandas Series has many other calculation methods you can use. Here are all the calculation methods on you can use on series.

Todo:

Try some of the other following Pandas calculation methods like **mul** and **mean** on the above Panda series.

Function	Description
add()	Method is used to add series or list like objects with same length to the caller series
sub()	Method is used to subtract series or list like objects with same length from the caller series
mul()	Method is used to multiply series or list like objects with same length with the caller series
div()	Method is used to divide series or list like objects with same length by the caller series
sum()	Returns the sum of the values for the requested axis
prod()	Returns the product of the values for the requested axis
mean()	Returns the mean of the values for the requested axis
pow()	Method is used to put each element of passed series as exponential power of caller series and returned the results
abs()	Method is used to get the absolute numeric value of each element in Series/DataFrame
cov()	Method is used to find covariance of two series

All pandas series methods

Here are all the Pandas Series methods for your reference:

Function	Description
Series()	A pandas Series can be created with the Series() constructor method. This constructor method accepts a variety of inputs
combine_first()	Method is used to combine two series into one
count()	Returns number of non-NA/null observations in the Series
size()	Returns the number of elements in the underlying data
name()	Method allows to give a name to a Series object, i.e. to the column
is_unique()	Method returns boolean if values in the object are unique
idxmax()	Method to extract the index positions of the highest values in a Series
idxmin()	Method to extract the index positions of the lowest values in a Series
sort_values()	Method is called on a Series to sort the values in ascending or descending order
sort_index()	Method is called on a pandas Series to sort it by the index instead of its values
head()	Method is used to return a specified number of rows from the beginning of a Series. The method returns a brand new Series
tail()	Method is used to return a specified number of rows from the end of a Series. The method returns a brand new Series
le()	Used to compare every element of Caller series with passed series. It returns True for every element which is Less than or Equal to the element in passed series
ne()	Used to compare every element of Caller series with passed series. It returns True for every element which is Not Equal to the element in passed series
ge()	Used to compare every element of Caller series with passed series. It returns True for every element which is Greater than or Equal to the element in passed series

eq()	Used to compare every element of Caller series with passed series. It returns True for every element which is Equal to the element in passed series
gt()	Used to compare two series and return Boolean value for every respective element
lt()	Used to compare two series and return Boolean value for every respective element
clip()	Used to clip value below and above to passed Least and Max value
clip_lower()	Used to clip values below a passed least value
clip_upper()	Used to clip values above a passed maximum value
astype()	Method is used to change data type of a series
tolist()	Method is used to convert a series to list
get()	Method is called on a Series to extract values from a Series. This is alternative syntax to the traditional bracket syntax
unique()	Pandas unique() is used to see the unique values in a particular column
nunique()	Pandas nunique() is used to get a count of unique values
value_counts()	Method to count the number of the times each unique value occurs in a Series
factorize()	Method helps to get the numeric representation of an array by identifying distinct values
map()	Method to tie together the values from one object to another
between()	Pandas between() method is used on series to check which values lie between first and second argument
apply()	Method is called and feeds a Python function as an argument to use the function on every Series value. This method is helpful for executing custom operations that are not included in pandas or numpy

PANDAS SERIES HOMEWORK

Question 1

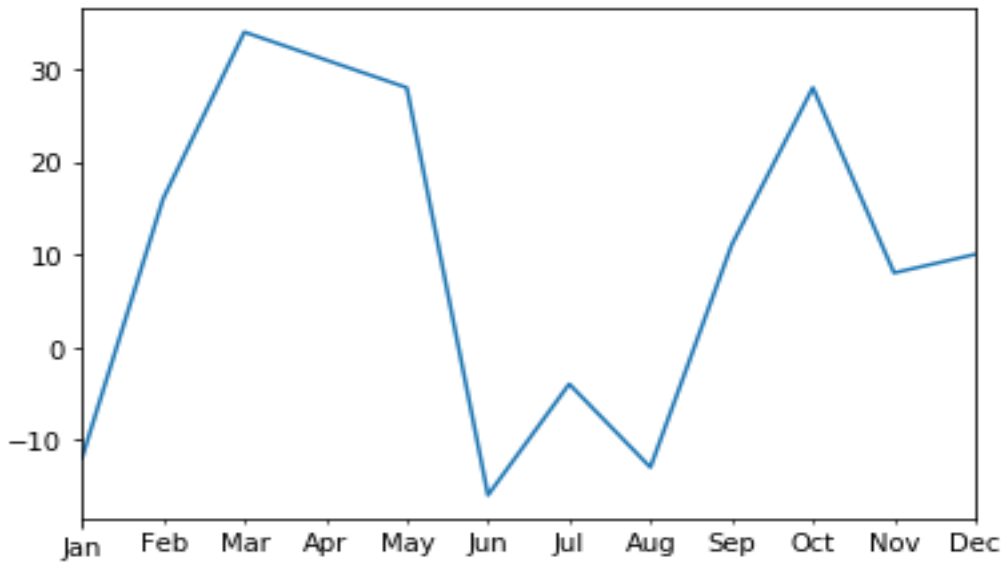
Make a series of random numbers to represent temperatures -30 to 120 degrees, for 12 months of a certain year. Generate a timestamp for the dates using a time span frequency of your choice. Put the temperate in a series and set the index to the generated time stamps dates. Print out the temperature series and plot the temp series as a line chart where the temperatures dates are the x index and the temperature values are the y axis.

Use the following code to format your dates:

```
import matplotlib.ticker as mticker

fig, ax = plt.subplots()
temp_series.plot()
ax.set_xticks([x for x in temp_series.index])
ticks_loc = ax.get_xticks().tolist()
ax.xaxis.set_major_locator(mticker.FixedLocator(ticks_loc))
ax.set_xticklabels([x.strftime("%b") for x in temp_series.index])
plt.show()
```

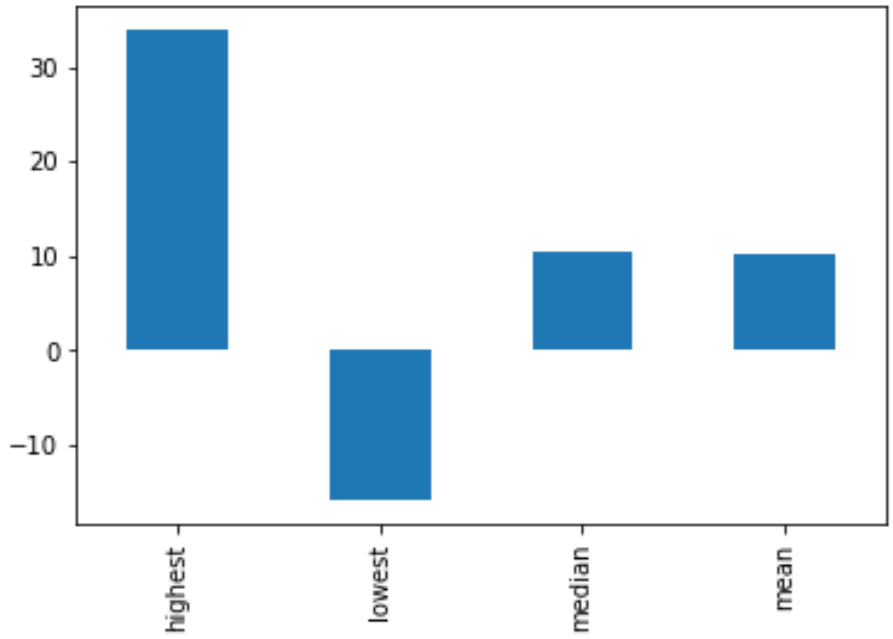
You should get something like this.



Question 2

Find the highest, Lowest, median and mean temperatures and put the values into another series. Set the index to a list of the statistic column names. Print out the statistic series and plot the highest, lowest, median and mean temperatures in a bar chart.

You should get something like this:



Call you homework file pandas_series_homework.py

END