DATA INGESTION AND INSPECTION

IMPORTING & EXPORTING DATA

```
1818,01,01,1818.004, -1,1
1818,01,02,1818.007, -1,1
1818,01,03,1818.010, -1,1
1818,01,04,1818.012, -1,1
1818,01,05,1818.015, -1,1
1818,01,06,1818.018, -1,1
```

filepath = "ISSN_D_tot.csv"
sunspots = pd.read_csv(filepath)
sunspots.**iloc**[10:20, :]

	1818	01	01.1	1818.004	-1	1
10	1818	1	12	1818.034	-1	1
11	1818	1	13	1818.037	22	1
12	1818	1	14	1818.040	-1	1
13	1818	1	15	1818.042	-1	1
14	1818	1	16	1818.045	-1	1
15	1818	1	17	1818.048	46	1
16	1818	1	18	1818.051	59	1
17	1818	1	19	1818.053	63	1
18	1818	1	20	1818.056	-1	1
19	1818	1	21	1818.059	-1	1

PB 1 : COLUMN HEADERS

sunspots = pd.read_csv(filepath, **header=None**)

col_names = ["year", "month", "day", "dec_date", "sunspots", "definite"**]**
sunspots = pd.read_csv(filepath, header=None, **names=col_names**)

	0	1	2	3	4	5
10	1818	1	11	1818.031	-1	1
11	1818	1	12	1818.034	-1	1
12	1818	1	13	1818.037	22	1
13	1818	1	14	1818.040	-1	1
14	1818	1	15	1818.042	-1	1
15	1818	1	16	1818.045	-1	1
16	1818	1	17	1818.048	46	1
17	1818	1	18	1818.051	59	1
18	1818	1	19	1818.053	63	1
19	1818	1	20	1818.056	-1	1

PB 2 : MISSING VALUES

sunspots = pd.read_csv(filepath, header=None, names=col_names, **na_values=" -1"**) *espace
ou
sunspots = pd.read_csv(filepath, header=None, names=col_names, **na_values={"sunspots":[" -1"]}**)

	year	month	day	dec_date	sunspots	definite
10	1818	1	11	1818.031	-1	1
11	1818	1	12	1818.034	-1	1
12	1818	1	13	1818.037	22	1
13	1818	1	14	1818.040	-1	1
14	1818	1	15	1818.042	-1	1
15	1818	1	16	1818.045	-1	1
16	1818	1	17	1818.048	46	1
17	1818	1	18	1818.051	59	1
18	1818	1	19	1818.053	63	1
19	1818	1	20	1818.056	-1	1

PB 3 : DATA REPRESENTATION

sunspots = pd.read_csv(filepath, header=None, names=col_names, na_values={"sunspots":[" -1"]}, **parse_dates=[[0, 1, 2]]**)

	year	month	day	dec_date	sunspots	definite
10	1818	1	11	1818.031	NaN	1
11	1818	1	12	1818.034	NaN	1
12	1818	1	13	1818.037	22.0	1
13	1818	1	14	1818.040	NaN	1
14	1818	1	15	1818.042	NaN	1
15	1818	1	16	1818.045	NaN	1
16	1818	1	17	1818.048	46.0	1
17	1818	1	18	1818.051	59.0	1
18	1818	1	19	1818.053	63.0	1
19	1818	1	20	1818.056	NaN	1

	year_month_day	dec_date	sunspots	definite
10	1818-01-11	1818.031	NaN	1
11	1818-01-12	1818.034	NaN	1
12	1818-01-13	1818.037	22.0	1
13	1818-01-14	1818.040	NaN	1
14	1818-01-15	1818.042	NaN	1
15	1818-01-16	1818.045	NaN	1
16	1818-01-17	1818.048	46.0	1
17	1818-01-18	1818.051	59.0	1
18	1818-01-19	1818.053	63.0	1
19	1818-01-20	1818.056	NaN	1

sunspots.**index** = sunspots["year_month_day"]
sunspots.**index.name** = "date"

cols = ["sunspots", "definite"**]**
sunspots = sunspots[cols]

date	sunspots	definite
1818-01-11	NaN	1
1818-01-12	NaN	1
1818-01-13	22.0	1
1818-01-14	NaN	1
1818-01-15	NaN	1
1818-01-16	NaN	1
1818-01-17	46.0	1
1818-01-18	59.0	1
1818-01-19	63.0	1
1818-01-20	NaN	1

WRITING FILES

out_csv = "sunspots.csv"
sunspots.to_csv(out_csv)

out_tsv = "sunspots.tsv"
sunspots.to_csv(out_csv, **sep = "\t")**

out_xlsx = "sunspots.xlsx"
sunspots.to_excel(out_xlsx)

PLOTTING WITH PANDAS

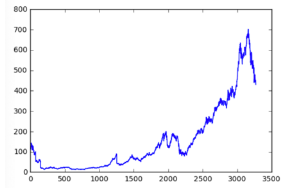
import pandas as pd
import matplotlib.pyplot as plt

aapl = pd.read_csv("aapl.csv", index_col="-date", parse_dates=True)

	adj_close	close	high	low	open	volume
date						
2000-03-01	31.68	130.31	132.06	118.50	118.56	38478000
2000-03-02	29.66	122.00	127.94	120.69	127.00	11136800
2000-03-03	31.12	128.00	128.23	120.00	124.87	11565200
2000-03-06	30.56	125.69	129.13	125.00	126.00	7520000
2000-03-07	29.87	122.87	127.44	121.12	126.44	9767600
2000-03-08	29.66	122.00	123.94	118.56	122.87	9690800

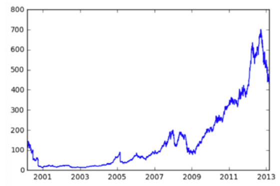
PLOTTING ARRAYS (MATPLOTLIB)

close_arr = aapl["close"]**.values**
plt.plot(close_arr)



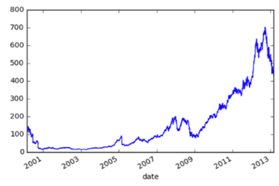
PLOTTING SERIES (MATPLOTLIB)

close_series = aapl["close"]
plt.plot(close_series**)**



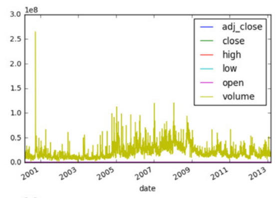
PLOTTING SERIES (PANDAS)

close_series**.plot()**



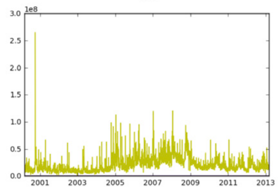
PLOTTING DATAFRAMES (PANDAS)

aapl**.plot()**



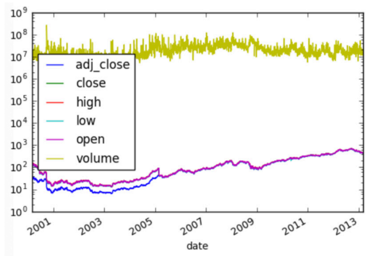
PLOTTING DATAFRAMES (MATPLOTLIB)

plt.plot(aapl)



FIXING SCALES

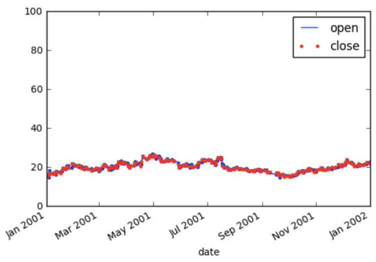
aapl.plot()
plt.yscale("log")



CUSTOMIZING PLOTS

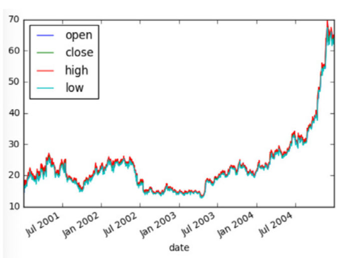
aapl["open"]**.plot(color="b", style= ".-", legend=True)**
aapl["close"].plot(color="r", style= ".", legend=True)

plt.axis(("2001", "2002", 0, 100))
("2001", "2002", 0, 100)



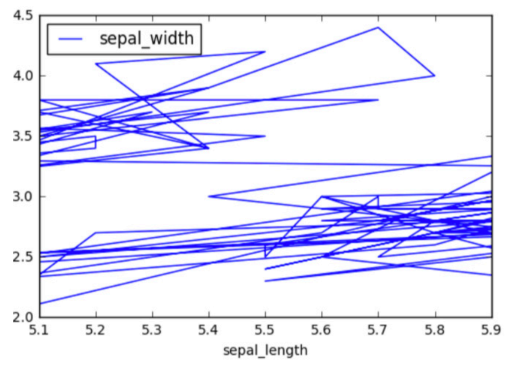
SAVING PLOTS

aapl.loc["2001":"2004", ["open", "close", "high", "low"]].plot()
plt.savefig("aapl.png")
plt.savefig("aapl.jpg")
plt.savefig("aapl.pdf")



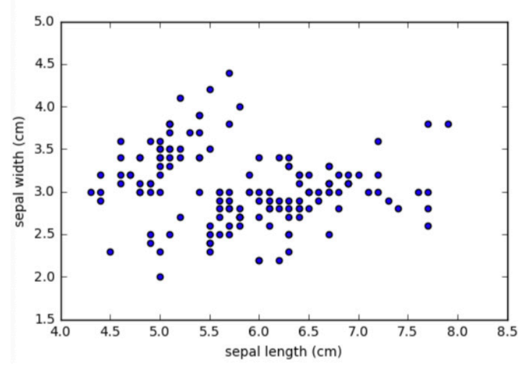
VISUAL EXPLORATORY DATA ANALYSIS

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa



LINE PLOT

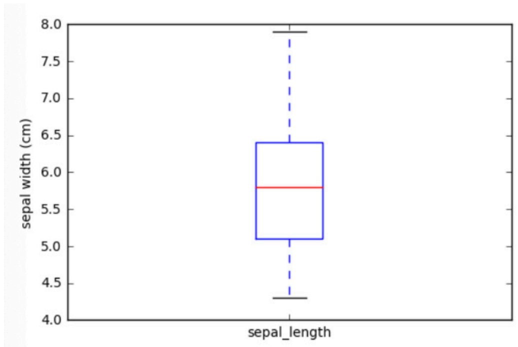
iris = pd.read_csv(“iris.csv”, index_col=0)
iris.plot(**x=”sepal_length”**, **y= “sepal_width”**)



SCATTER PLOT

iris.plot(x=”sepal_length”, y= “sepal_width”, **kind=”scatter”**)

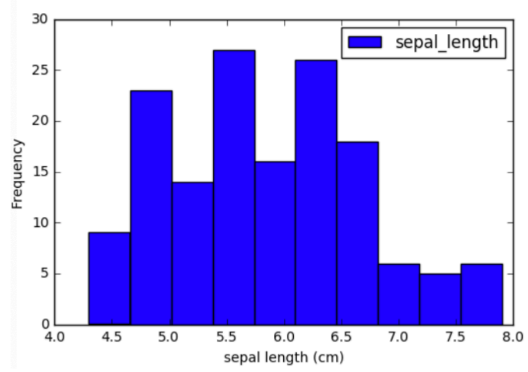
plt.**xlabel(“sepal length (cm)”)**
plt.**ylabel(“sepal width (cm)”)**



BOX PLOT

iris.plot(y=”sepal_length”, **kind=”box”**)

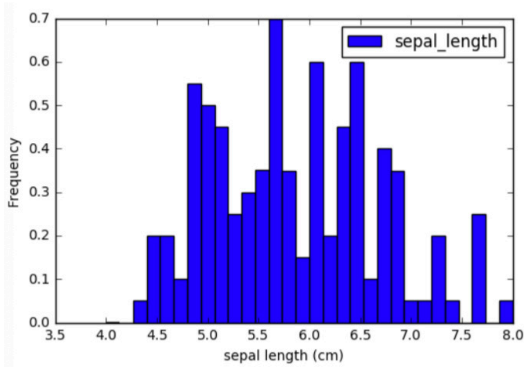
plt.**ylabel(“sepal width (cm)”)**



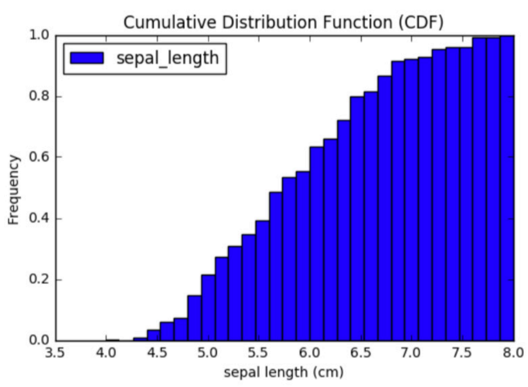
HISTOGRAM

iris.plot(y=”sepal_length”, **kind=”hist”**)
plt.**xlabel(“sepal length (cm)”)**

*Histogram options
bins (integer): number of intervals or bins
range (tuple): extrema of bins (min, max)
normed (boolean): whether to normalize to one
cumulative (boolean): compute Cumulative Distribution Function (CDF)



iris.plot(y=”sepal_length”, kind=”hist”, **bins=30, range=(4, 8), normed=True**)
plt.xlabel(“sepal length (cm)”)



CUMULATIVE DISTRIBUTION

iris.plot(y=”sepal_length”, kind=”hist”, bins=30, range=(4, 8), **cumulative=True**, normed=True)
plt.xlabel(“sepal length (cm)”)
plt.**title(“Cumulative distribution function (CDF)”)**

WARNING

3 different idioms
iris.**plot(kind=”hist”)**
iris.**plt.hist()**
iris.**hist()**
syntax/results differ
pandas API still evolving: check documentation

STATISTICAL EXPLORATORY DATA ANALYSIS

iris.describe()

	sepal_length	sepal_width	petal_length	petal_width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

TO SERIES

iris[**"sepal_length"].count()**

iris[**"sepal_length"].mean()**

TO DATAFRAME

iris[**["petal_length", "petal_width"]].count()**

iris.**mean()**

SEPARATING POPULATIONS

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

count	150
unique	3
top	setosa
freq	50

Name: species, dtype: object

iris[**"species"].describe()**

count : non null entries
unique : distinct values
top : most frequent category
freq : occurences of top

iris[**"species"].unique()**
array(["setosa", "versicolor", "virginica"], dtype=object)

FILTERING BY SPECIES
EXTRACT NEW DATAFRAME

indices = **iris[**"species"**] == "setosa"**
setosa = **iris.loc[**indices, :]

indices = iris["species"] **== "versicolor"**
versicolor = **iris.loc[**indices, :]

indices = **iris[**"species"**] == "virginica"**
virginica = **iris.loc[**indices, :]

CHECKING SPECIES

setosa["species"**].unique()**
array(["setosa"], dtype=object)

versicolor["species"**].unique()**
array(["versicolor"], dtype=object)

virginica["species"**].unique()**
array(["virginica"], dtype=object)

del setosa["species"], versicolor["species"], virginica["species"]

CHECKING INDEXES

setosa.head(2)

	sepal_length	sepal_width	petal_length	petal_width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2

versicolor.head(2)

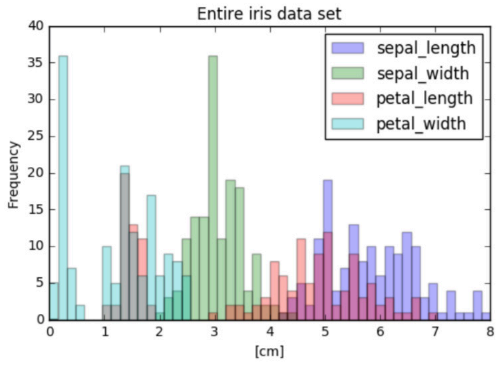
	sepal_length	sepal_width	petal_length	petal_width
50	7.0	3.2	4.7	1.4
51	6.4	3.2	4.5	1.5

virginica.head(2)

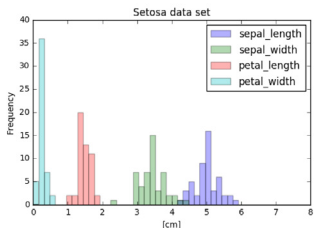
	sepal_length	sepal_width	petal_length	petal_width
100	6.3	3.3	6.0	2.5
101	5.8	2.7	5.1	1.9

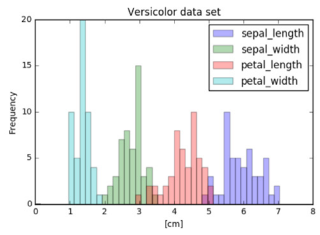
VISUAL EDA: ALL DATA

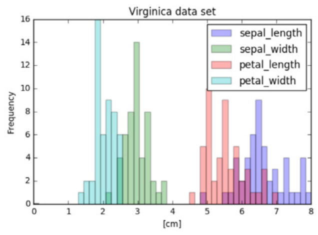
iris.plot(kind= "hist", bins=50, range=(0, 8), alpha=0.3**)**
plt.**title("Entire iris data set")**
plt.xlabel("[cm]")



VISUAL EDA: INDIVIDUAL FACTORS







setosa.plot(kind= "hist", bins=50, range=(0, 8), alpha=0.3**)**
plt.**title("Setosa data set")**
plt.xlabel("[cm]")

versicolor.plot(kind= "hist", bins=50, range=(0, 8), alpha=0.3**)**
plt.**title("Versicolor data set")**
plt.xlabel("[cm]")

virginica.plot(kind= "hist", bins=50, range=(0, 8), alpha=0.3**)**
plt.**title("Virginica data set")**
plt.xlabel("[cm]")

STATISTICAL EDA: DESCRIBE()

describe_all = iris.describe()
describe_setosa = setosa.describe()
describe_versicolor = versicolor.describe()
describe_virginica = virginica.describe()

COMPUTING AND VIEWING ERRORS]

error_setosa = 100 * np.abs(describe_setosa **- describe_all)**
error_setosa = error_setosa/describe_setosa

print(error_setosa)

	sepal_length	sepal_width	petal_length	petal_width
count	200.000000	200.000000	200.000000	200.000000
mean	16.726595	10.812913	157.045144	387.533875
std	134.919250	14.984768	916.502136	623.284534
min	0.000000	13.043478	0.000000	0.000000
25%	6.250000	12.500000	14.285714	50.000000
50%	16.000000	11.764706	190.000000	550.000000
75%	23.076923	10.204082	223.809524	500.000000
max	36.206897	0.000000	263.157895	316.666667

TIMES SERIES IN PANDAS

INDEXING TIME SERIES

USING PANDAS TO READ DATETIME OBJECTS

read_csv() function

can read strings into datetime objects: specify "parse_dates=True"

ISO 8601 format

yyyy-mm-dd hh:mm:ss

	Date	Company	Product	Units
0	2015-02-02 08:30:00	Hooli	Software	3
1	2015-02-02 21:00:00	Mediacore	Hardware	9
2	2015-02-03 14:00:00	Initech	Software	13
3	2015-02-04 15:30:00	Streeplex	Software	13
4	2015-02-04 22:00:00	Acme Coporation	Hardware	14

Product sales CSV

PARSE DATES

sales = pd.read_csv("sales-feb-2015.csv", **parse_dates=True, index_col = "Date"**)

```
                              Company   Product  Units
Date
2015-02-02 08:30:00             Hooli  Software      3
2015-02-02 21:00:00         Mediacore  Hardware      9
2015-02-03 14:00:00           Initech  Software     13
2015-02-04 15:30:00         Streeplex  Software     13
2015-02-04 22:00:00   Acme Coporation  Hardware     14
```

SELECTING SINGLE DATE TIME

sales.**loc["2015-02-19 11:00:00", "Company"]**

SELECTING WHOLE DAY

sales.**loc["2015-2-5"]**

ALTERNATIVE FORMATS:

sales.**loc["February 5, 2015"]**
sales.**loc["2015-Feb-5"]**

WHOLE MONTH:

sales.**loc["2015-2"]**

WHOLE YEAR:

sales.**loc["2015"]**

SLICING USING DATES/TIMES

sales.loc[**"2015-2-16":"2015-2-20"**]

```
                       Company   Product  Units
Date
2015-02-16 12:00:00      Hooli  Software     10
2015-02-19 11:00:00  Mediacore  Hardware     16
2015-02-19 16:00:00  Mediacore   Service     10
```

CONVERT STRINGS TO DATETIME

evening_2_11 = **pd.to_datetime([**"2015-2-11 20:00", "2015-2-11 21:00", "2015-2-11 22:00", "2015-2-11 23:00"**])**

```
DatetimeIndex(['2015-02-11 20:00:00', '2015-02-11 21:00:00',
               '2015-02-11 22:00:00', '2015-02-11 23:00:00'],
              dtype='datetime64[ns]', freq=None)
```

REINDEXING DATAFRAME

sales.**reindex**(evening_2_11)

```
                     Company   Product  Units
2015-02-11 20:00:00  Initech  Software    7.0
2015-02-11 21:00:00      NaN       NaN    NaN
2015-02-11 22:00:00      NaN       NaN    NaN
2015-02-11 23:00:00    Hooli  Software    4.0
```

FILLING MISSING VALUES

sales.reindex(evening_2_11, **method= "ffill"**)
sales.reindex(evening_2_11, **method="bfill"**)

RESAMPLING TIME SERIES DATA

```
                             Company   Product  Units
Date
2015-02-02 08:30:00            Hooli  Software      3
2015-02-02 21:00:00        Mediacore  Hardware      9
2015-02-03 14:00:00          Initech  Software     13
2015-02-04 15:30:00        Streeplex  Software     13
2015-02-04 22:00:00  Acme Coporation  Hardware     14
```

Statistical methods over different time intervals
mean(), count(), sum()...

Down-sampling
reduce datetime rows to slower frequency

Up-sampling
increase datetime rows to faster frequency

AGGREGATING MEANS

daily_mean = sales.**resample(**"D"**).mean()**

```
            Units
Date
2015-02-02    6.0
2015-02-03   13.0
2015-02-04   13.5
2015-02-05   14.5
2015-02-06    NaN
2015-02-07    1.0
2015-02-08    NaN
2015-02-09   13.0
2015-02-10    NaN
2015-02-11    5.5
2015-02-12    NaN
2015-02-13    NaN
2015-02-14    NaN
```

VERIFYING

print(daily_mean.loc["2015-2-2"])
print(sales.loc["2015-2-2", "Units"])
sales.loc["2015-2-2", "Units"].mean()

METHOD CHAINING

sales.**resample(**"D"**).sum().max()**

RESAMPLING STRINGS

sales.**resample(**"W"**).count()**

```
In [11]: sales.resample('W').count()
Out[11]:
            Company  Product  Units
Date
2015-02-08        8        8      8
2015-02-15        4        4      4
2015-02-22        5        5      5
2015-03-01        2        2      2
```

RESAMPLING FREQUENCIES

Input	Description
'min', 'T'	minute
'H'	hour
'D'	day
'B'	business day
'W'	week
'M'	month
'Q'	quarter
'A'	year

MULTIPLYING FREQUENCIES

sales.**loc[:,** "Units"**].resample(**"2W"**).sum()**

```
Date
2015-02-08    82
2015-02-22    79
2015-03-08    14
Freq: 2W-SUN, Name: Units, dtype: int64
```

UPSAMPLING

two_days = sales.loc[**"2015-2-4" : "2015-2-5"**, "Units"]

```
Date
2015-02-04 15:30:00    13
2015-02-04 22:00:00    14
2015-02-05 02:00:00    19
2015-02-05 22:00:00    10
Name: Units, dtype: int64
```

UPSAMPLING AND FILLING

two_days.**resample(**"4H"**).ffill()**

```
Date
2015-02-04 12:00:00     NaN
2015-02-04 16:00:00    13.0
2015-02-04 20:00:00    13.0
2015-02-05 00:00:00    14.0
2015-02-05 04:00:00    19.0
2015-02-05 08:00:00    19.0
2015-02-05 12:00:00    19.0
2015-02-05 16:00:00    19.0
2015-02-05 20:00:00    19.0
Freq: 4H, Name: Units, dtype: float64
```

TIMES SERIES IN PANDAS

MANIPULATING TIME SERIES DATA

sales = pd.read_csv("sales-feb-2015.csv", parse_dates=["Date"])

	Date	Company	Product	Units
0	2015-02-02 08:30:00	Hooli	Software	3
1	2015-02-02 21:00:00	Mediacore	Hardware	9
2	2015-02-03 14:00:00	Initech	Software	13
3	2015-02-04 15:30:00	Streeplex	Software	13
4	2015-02-04 22:00:00	Acme Coporation	Hardware	14

sales["Company"].**str.upper()**

```
0             HOOLI
1         MEDIACORE
2           INITECH
3         STREEPLEX
4   ACME COPORATION
5   ACME COPORATION
6             HOOLI
7   ACME COPORATION
8         STREEPLEX
9         MEDIACORE
10          INITECH
11            HOOLI
12            HOOLI
13        MEDIACORE
14        MEDIACORE
15        MEDIACORE
…
```

sales["Product"].**str.contains("ware")**

```
0      True
1      True
2      True
3      True
4      True
5      True
6     False
7      True
8     False
9      True
10     True
11     True
12     True
13     True
14    False
…
```

sales["Product"].**str.contains("ware").sum()**

14

DATETIME METHODS

sales["Date"].**dt.hour**

```
0      8
1     21
2     14
3     15
4     22
5      2
6     22
7     23
8      9
9     13
10    20
11    23
12    12
13    11
14    16
…
```

SET TIMEZONE

central = sales["Date"].**dt.tz_localize(**"US/Central"**)**

CONVERT TIMEZONE

central.**dt.tz(**"US/Eastern"**)**

METHOD CHAINING

sales["Date"].**dt.tz_localize(**"US/Central"**).dt.tz_convert(**"US/Eastern"**)**

WORLD POPULATION

population = pd.read_csv("world_population.csv", parse_dates=True, **index_col= "Date"**)

Date	Population
1960-12-31	2.087485e+10
1970-12-31	2.536513e+10
1980-12-31	3.057186e+10
1990-12-31	3.644928e+10
2000-12-31	4.228550e+10
2010-12-31	4.802217e+10

UNSAMPLE POPULATION

population.**resample(**"A"**).first()**

Date	Population
1960-12-31	2.087485e+10
1961-12-31	NaN
1962-12-31	NaN
1963-12-31	NaN
1964-12-31	NaN
1965-12-31	NaN
1966-12-31	NaN
1967-12-31	NaN
1968-12-31	NaN
1969-12-31	NaN
1970-12-31	2.536513e+10
1971-12-31	NaN
1972-12-31	NaN

INTERPOLATE MISSING DATA

population.resample("A").first()**.interpolate("linear")**

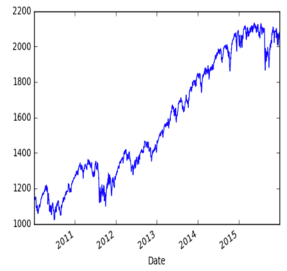
Date	Population
1960-12-31	2.087485e+10
1961-12-31	2.132388e+10
1962-12-31	2.177290e+10
1963-12-31	2.222193e+10
1964-12-31	2.267096e+10
1965-12-31	2.311999e+10
1966-12-31	2.356902e+10
1967-12-31	2.401805e+10
1968-12-31	2.446707e+10
1969-12-31	2.491610e+10
1970-12-31	2.536513e+10
1971-12-31	2.588580e+10
1972-12-31	2.640648e+10

TIME SERIES VISUALIZATION

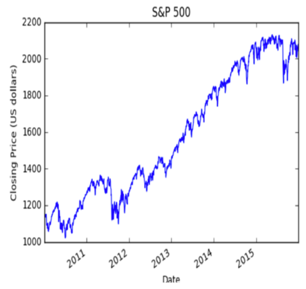
Date	Open	High	Low	Close	Volume	Adj Close
2010-01-04	1116.560059	1133.869995	1116.560059	1132.989990	3991400000	1132.989990
2010-01-05	1132.660034	1136.630005	1129.660034	1136.520020	2491020000	1136.520020
2010-01-06	1135.709961	1139.189941	1133.949951	1137.140015	4972660000	1137.140015
2010-01-07	1136.270020	1142.459961	1131.319946	1141.689941	5270680000	1141.689941
2010-01-08	1140.520020	1145.390015	1136.219971	1144.979980	4389590000	1144.979980

PANDAS PLOT

sp500["Close"]**.plot()**

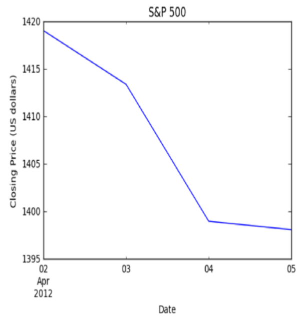


sp500["Close"].plot(**title="S&P 500"**)
plt.**ylabel("Closing price (US Dollars)")**



ONE WEEK

sp500.**loc["2012-4-1" : "2012-4-7"**, "Close"].plot(title="S&P 500")
plt.ylabel("Closing price (US Dollars)")

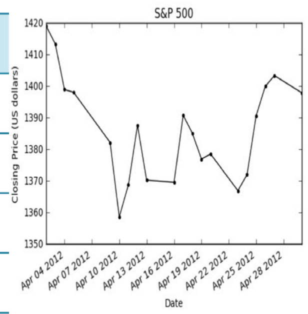


PLOT STYLES

sp500.loc["2012-4", "Close"].plot(**style="k.-"**, title="S&P500")
plt.ylabel("Closing price (US Dollars)")

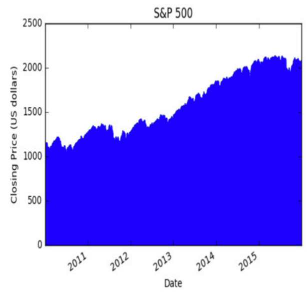
Style format string
color (**k** : black)
marker (**.** : dot)
line type (**-** : solid)

Color	Marker	Line
b: blue	o: circle	: dotted
g: green	*: star	--: dashed
r: red	s: square	
c: cyan	+: plus	



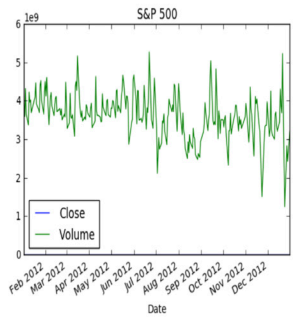
AREA PLOT

sp500["Close"].plot(**kind="area"**, title = "S&P 500")
plt.ylabel("Closing price (US Dollars)")



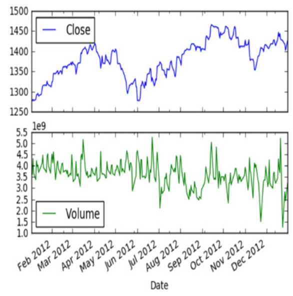
MULTIPLE COLUMNS

sp500.loc[**"2012", ["Close", "Volume"]**].plot(title="S&P 500")



SUBPLOTS

sp500.loc["2012, ["Close", "Volume"]].plot(**subplots=True**)



```
import pandas as pd
df = pd.read_csv(data_file)
print(df.head())
df_headers = pd.read_csv(data_file, header=None)
print(df_headers.head())
```

```
   13904  20110101  0053  12  OVC045  ...   .21   .22   .23 29.95.1   .24
0  13904  20110101   153  12  OVC049  ...                     30.02
1  13904  20110101   253  12  OVC060  ...                     30.02
2  13904  20110101   353  12  OVC065  ...                     30.04
3  13904  20110101   453  12  BKN070  ...                     30.04
4  13904  20110101   553  12  BKN065  ...                     30.06

[5 rows x 44 columns]
      0         1    2   3       4   ... 39 40 41     42 43
0  13904  20110101   53  12  OVC045  ...           29.95
1  13904  20110101  153  12  OVC049  ...           30.02
2  13904  20110101  253  12  OVC060  ...           30.02
3  13904  20110101  353  12  OVC065  ...           30.04
4  13904  20110101  453  12  BKN070  ...           30.04

[5 rows x 44 columns]
```

```
# Split on the comma to create a list: column_labels_list
column_labels_list = column_labels.split(",")
df.columns = column_labels_list
# Remove the appropriate columns: df_dropped
df_dropped = df.drop(list_to_drop, axis="columns")
print(df_dropped.head())
```

```
    Wban      date  Time  StationType sky_condition  ... relative_humidity wind_speed wind_direction station_pressure sea_level_pressure
0  13904  20110101    53           12        OVC045  ...                24         15            360            29.42              29.95
1  13904  20110101   153           12        OVC049  ...                23         10            340            29.49              30.01
2  13904  20110101   253           12        OVC060  ...                22         15            010            29.49              30.01
3  13904  20110101   353           12        OVC065  ...                27          7            350            29.51              30.03
4  13904  20110101   453           12        BKN070  ...                25         11            020            29.51              30.04

[5 rows x 17 columns]
```

```
# Convert the date column to string: df_dropped['date']
df_dropped['date'] = df_dropped["date"].astype(str)
# Pad leading zeros to the Time column: df_dropped['Time']
df_dropped['Time'] = df_dropped['Time'].apply(lambda x:'{:0>4}'.
format(x))
# Concatenate the new date and Time columns: date_string
date_string = df_dropped["date"] + df_dropped["Time"]
# Convert the date_string Series to datetime: date_times
date_times = pd.to_datetime(date_string, format='%Y%m%d%H%M')
# Set the index to be the new date_times container: df_clean
df_clean = df_dropped.set_index(date_times)
print(df_clean.head())
```

```
                     Wban date Time  StationType sky_condition  ... relative_humidity wind_speed wind_direction station_pressure
                        sea_level_pressure
2011-01-01 00:53:00   NaN  NaN  NaN          NaN           NaN  ...               NaN        NaN            NaN              NaN
           NaN
2011-01-01 01:53:00   NaN  NaN  NaN          NaN           NaN  ...               NaN        NaN            NaN              NaN
           NaN
2011-01-01 02:53:00   NaN  NaN  NaN          NaN           NaN  ...               NaN        NaN            NaN              NaN
           NaN
2011-01-01 03:53:00   NaN  NaN  NaN          NaN           NaN  ...               NaN        NaN            NaN              NaN
           NaN
2011-01-01 04:53:00   NaN  NaN  NaN          NaN           NaN  ...               NaN        NaN            NaN              NaN
           NaN

[5 rows x 17 columns]

ript.py> output:
                      Wban      date  Time  StationType sky_condition  ... relative_humidity wind_speed wind_direction station_pressure
                        sea_level_pressure
2011-01-01 00:53:00  13904  20110101  0053           12        OVC045  ...                24         15            360            29.42
              29.95
2011-01-01 01:53:00  13904  20110101  0153           12        OVC049  ...                23         10            340            29.49
              30.01
2011-01-01 02:53:00  13904  20110101  0253           12        OVC060  ...                22         15            010            29.49
              30.01
2011-01-01 03:53:00  13904  20110101  0353           12        OVC065  ...                27          7            350            29.51
              30.03
2011-01-01 04:53:00  13904  20110101  0453           12        BKN070  ...                25         11            020            29.51
              30.04
```

```
# Print the dry_bulb_faren temperature between 8 AM and 9 AM on
June 20, 2011
print(df_clean.loc['2011-6-20 8:00:00':'2011-6-20 9:00:00', 'dry_bulb_
faren'])

# Convert the dry_bulb_faren column to numeric values: df_clean['dry_
bulb_faren']
df_clean['dry_bulb_faren'] = pd.to_numeric(df_clean['dry_bulb_
faren'], errors='coerce')

# Print the transformed dry_bulb_faren temperature between 8 AM
and 9 AM on June 20, 2011
print(df_clean.loc['2011-6-20 8:00:00':'2011-6-20 9:00:00', 'dry_bulb_
faren'])

# Convert the wind_speed and dew_point_faren columns to numeric
values
df_clean['wind_speed'] = pd.to_numeric(df_clean['wind_speed'], er-
rors='coerce')
df_clean['dew_point_faren'] = pd.to_numeric(df_clean['dew_point_
faren'], errors='coerce')
```

```
2011-06-20 08:27:00     M
2011-06-20 08:28:00     M
2011-06-20 08:29:00     M
2011-06-20 08:30:00     M
2011-06-20 08:31:00     M
2011-06-20 08:32:00     M
2011-06-20 08:33:00     M
2011-06-20 08:34:00     M
```

EDA

```
# Print the median of the dry_bulb_faren column
print(df_clean["dry_bulb_faren"].median())
# Print the median of the dry_bulb_faren column for the time range
'2011-Apr':'2011-Jun'
print(df_clean.loc['2011-Apr':'2011-Jun', 'dry_bulb_faren'].median())
# Print the median of the dry_bulb_faren column for the month of
January
print(df_clean.loc['2011-Jan', 'dry_bulb_faren'].median())
```

```
72.0
78.0
48.0
```

```
# Downsample df_clean by day and aggregate by mean: daily_
mean_2011
daily_mean_2011 = df_clean.resample('D').mean()

# Extract the dry_bulb_faren column from daily_mean_2011 using
.values: daily_temp_2011
daily_temp_2011 = daily_mean_2011['dry_bulb_faren'].values

# Downsample df_climate by day and aggregate by mean: daily_cli-
mate
daily_climate = df_climate.resample('D').mean()

# Extract the Temperature column from daily_climate using .reset_in-
dex(): daily_temp_climate
daily_temp_climate = daily_climate.reset_index()['Temperature']

# Compute the difference between the two arrays and print the mean
difference
difference = daily_temp_2011 - daily_temp_climate
print(difference.mean())
```

```
1.3301831870056477
```

```
# Using df_clean, when is sky_condition 'CLR'?
is_sky_clear = df_clean['sky_condition']=='CLR'
# Filter df_clean using is_sky_clear
sunny = df_clean.loc[is_sky_clear]
# Resample sunny by day then calculate the max
sunny_daily_max = sunny.resample('D').max()
# See the result
sunny_daily_max.head()

# Using df_clean, when does sky_condition contain 'OVC'?
is_sky_overcast = df_clean['sky_condition'].str.contains('OVC')
# Filter df_clean using is_sky_overcast
overcast = df_clean.loc[is_sky_overcast]
# Resample overcast by day then calculate the max
overcast_daily_max = overcast.resample("D").max()
# See the result
overcast_daily_max.head()

# From previous steps
is_sky_clear = df_clean['sky_condition']=='CLR'
sunny = df_clean.loc[is_sky_clear]
sunny_daily_max = sunny.resample('D').max()
is_sky_overcast = df_clean['sky_condition'].str.contains('OVC')
overcast = df_clean.loc[is_sky_overcast]
overcast_daily_max = overcast.resample('D').max()
# Calculate the mean of sunny_daily_max
sunny_daily_max_mean = sunny_daily_max.mean()
# Calculate the mean of overcast_daily_max
overcast_daily_max_mean = overcast_daily_max.mean()
# Print the difference (sunny minus overcast)
print(sunny_daily_max_mean - overcast_daily_max_mean)
```