numpy.random

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
x = np.array(range(10))
x
```

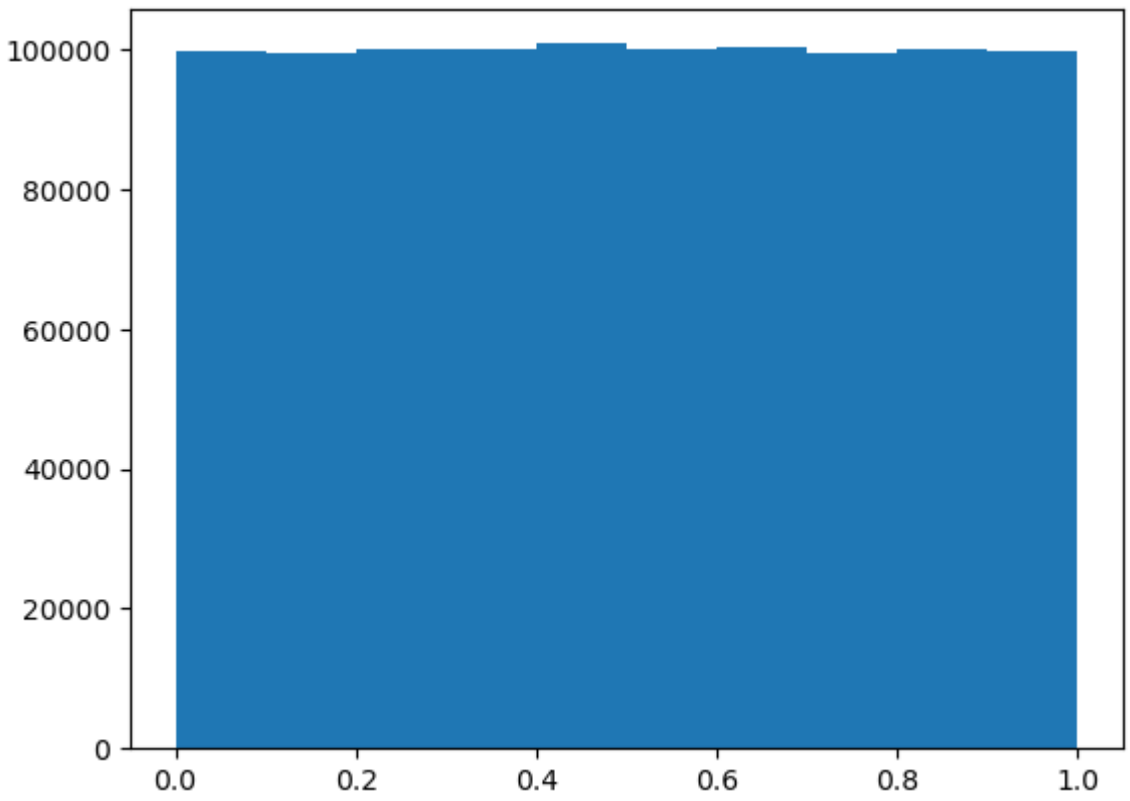
Out[2]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [3]:
```
np.random.permutation(x)
```

Out[3]:
```
array([6, 5, 7, 8, 0, 4, 2, 3, 9, 1])
```

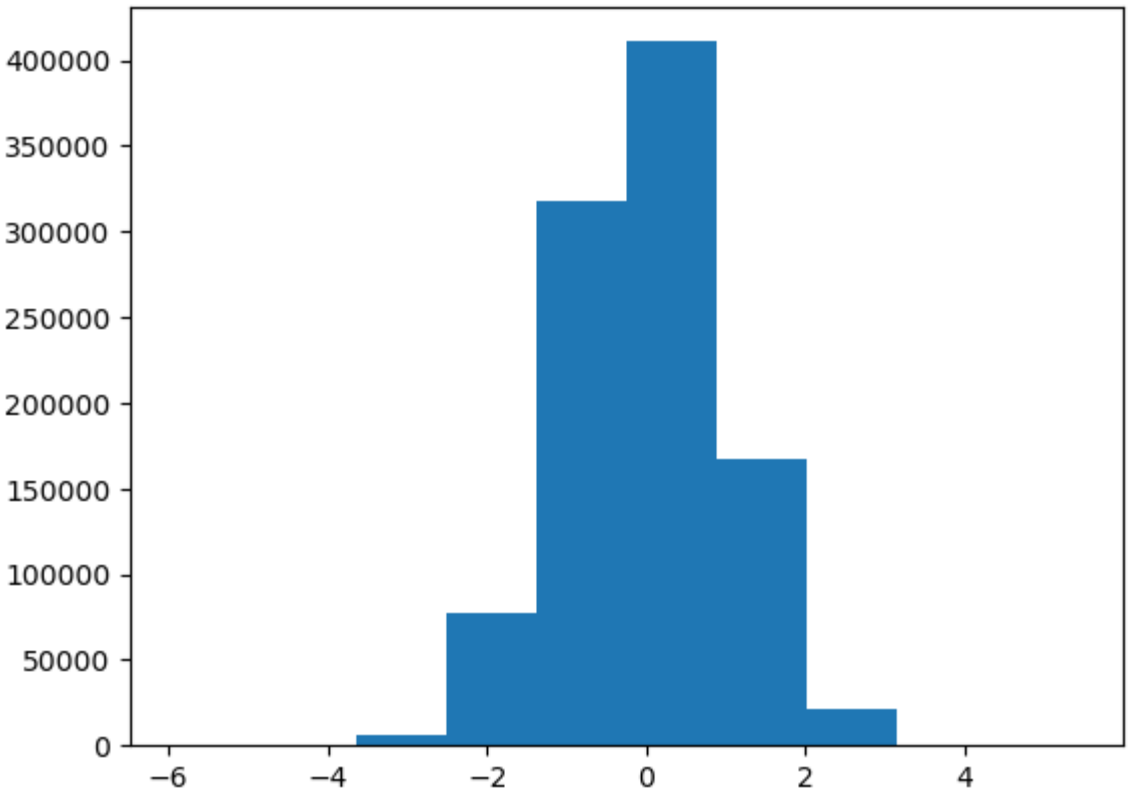
In [4]:
```
plt.hist(np.random.rand(1000000))
```

Out[4]:
```
(array([ 99698.,  99393.,  99986., 100149., 100856., 100199., 100273.,
         99541., 100017.,  99888.]),
 array([6.48776624e-08, 1.00000006e-01, 1.99999948e-01, 2.99999890e-01,
        3.99999831e-01, 4.99999773e-01, 5.99999714e-01, 6.99999656e-01,
        7.99999597e-01, 8.99999539e-01, 9.99999480e-01]),
 <BarContainer object of 10 artists>)
```



In [5]:
```
plt.hist(np.random.randn(1000000))
```

Out[5]:
```
(array([3.00000e+00, 1.25000e+02, 5.89200e+03, 7.73620e+04, 3.17250e+05,
        4.10833e+05, 1.66770e+05, 2.09390e+04, 8.17000e+02, 9.00000e+00]),
 array([-5.91762037, -4.78393005, -3.65023973, -2.51654941, -1.38285909,
        -0.24916877,  0.88452155,  2.01821187,  3.15190219,  4.28559252,
         5.41928284]),
 <BarContainer object of 10 artists>)
```



In [6]:
```
# random seed
```

In [7]:
```
np.random.rand(3,4)
```

Out[7]:
```
array([[0.21632343, 0.40975505, 0.44695627, 0.3678118 ],
       [0.6106304 , 0.67284978, 0.54694143, 0.20358173],
       [0.75214624, 0.616271  , 0.10784476, 0.82761636]])
```

In [8]:
```
np.random.rand(3,4)
```

Out[8]:
```
array([[0.23074304, 0.67697356, 0.82113735, 0.48783848],
       [0.58908519, 0.07553509, 0.18206368, 0.25898421],
       [0.22003754, 0.98557442, 0.05747807, 0.96157108]])
```

In [9]:
```
np.random.seed(44)
np.random.rand(3,4)
```

Out[9]:
```
array([[0.83484215, 0.1047961 , 0.74464048, 0.36050084],
       [0.35931084, 0.60923838, 0.39377955, 0.40907261],
       [0.50990241, 0.71014799, 0.96052623, 0.45662111]])
```

In [10]:
```
np.random.seed(44)
np.random.rand(3,4)
```

Out[10]:
```
array([[0.83484215, 0.1047961 , 0.74464048, 0.36050084],
       [0.35931084, 0.60923838, 0.39377955, 0.40907261],
       [0.50990241, 0.71014799, 0.96052623, 0.45662111]])
```

Lecture 6 Quiz 5

In [11]:
```
import numpy as np
```

In [12]:
```
x = np.array([ [3,2,1], [4,5,6] ])
x
```

Out[12]:
```
array([[3, 2, 1],
       [4, 5, 6]])
```

In [13]:
```
np.max(x[:, 0])
```

Out[13]:
```
4
```

In [14]:
```
np.max(x[-1,:])
```

Out[14]:
```
6
```

In [15]:
```
x + np.array([3,2,1])
```

Out[15]:
```
array([[6, 4, 2],
       [7, 7, 7]])
```

Lecture 7

In [16]:
```
import pandas as pd
```

1. From Lists, Dicts, ...to DataFrame

From a list

In [17]:
```
data = [['a', 1],
        ['b', 2]]
pd.DataFrame(data, columns = ['model', 'price'])
```

Out[17]:

	model	price
0	a	1
1	b	2

From a dictionary

In [18]:
```
data = {'model': ['a', 'b'], 'price' : [1,2]}
pd.DataFrame(data)
```

Out[18]:

	model	price
0	a	1
1	b	2

In [ ]:

2. From Excel to DataFrame

In [19]:
```
df = pd.read_csv('store1features.zip')
```

In [20]:
```
df.head()
```

Out[20]:

	Date	Store	Temperature	Fuel_Price	MarkDown1	MarkDown2	MarkDown3	Mark
0	2010-02-05	1	42.31	2.572	NaN	NaN	NaN	
1	2010-02-12	1	38.51	2.548	NaN	NaN	NaN	
2	2010-02-19	1	39.93	2.514	NaN	NaN	NaN	
3	2010-02-26	1	46.63	2.561	NaN	NaN	NaN	
4	2010-03-05	1	46.50	2.625	NaN	NaN	NaN	

In [21]:
```
type(df)
```

Out[21]:
```
pandas.core.frame.DataFrame
```

In [22]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 182 entries, 0 to 181
Data columns (total 12 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   Date          182 non-null    object 
 1   Store         182 non-null    int64  
 2   Temperature   182 non-null    float64
 3   Fuel_Price    182 non-null    float64
 4   MarkDown1     90 non-null     float64
 5   MarkDown2     73 non-null     float64
 6   MarkDown3     89 non-null     float64
 7   MarkDown4     90 non-null     float64
 8   MarkDown5     90 non-null     float64
 9   CPI           169 non-null    float64
 10  Unemployment  169 non-null    float64
 11  IsHoliday     182 non-null    bool   
dtypes: bool(1), float64(9), int64(1), object(1)
memory usage: 15.9+ KB
```

In [23]:
```
print(type(df))
print(type(df.iloc[:,0]))
print(type(df.iloc[0,:]))
print(df.shape)
print(df.iloc[:,0].shape)
```

```
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.series.Series'>
<class 'pandas.core.series.Series'>
(182, 12)
(182,)
```

3. The index

In [24]:
```
df = pd.read_csv('store1features.zip')
df
```

Out[24]:

	Date	Store	Temperature	Fuel_Price	MarkDown1	MarkDown2	MarkDown3	Ma
0	2010-02-05	1	42.31	2.572	NaN	NaN	NaN	
1	2010-02-12	1	38.51	2.548	NaN	NaN	NaN	
2	2010-02-19	1	39.93	2.514	NaN	NaN	NaN	
3	2010-02-26	1	46.63	2.561	NaN	NaN	NaN	
4	2010-03-05	1	46.50	2.625	NaN	NaN	NaN	
...	...	...	...	...	...	...	...	
177	2013-06-28	1	83.94	3.495	4205.98	796.70	6.84	
178	2013-07-05	1	79.85	3.422	7649.99	3503.29	1766.77	
179	2013-07-12	1	83.12	3.400	6089.94	1362.42	209.62	
180	2013-07-19	1	79.26	3.556	3117.04	1060.39	199.05	
181	2013-07-26	1	81.54	3.620	332.17	673.19	1.00	

182 rows × 12 columns

In [25]:
```
df['Date'] = pd.to_datetime(df['Date'])
```

In [26]:
```
df.index = df['Date']
df.head()
```

Out[26]:

	Date	Store	Temperature	Fuel_Price	MarkDown1	MarkDown2	MarkDo
Date							
2010-02-05	2010-02-05	1	42.31	2.572	NaN	NaN	
2010-02-12	2010-02-12	1	38.51	2.548	NaN	NaN	
2010-02-19	2010-02-19	1	39.93	2.514	NaN	NaN	
2010-02-26	2010-02-26	1	46.63	2.561	NaN	NaN	
2010-03-05	2010-03-05	1	46.50	2.625	NaN	NaN	

In [27]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 182 entries, 2010-02-05 to 2013-07-26
Data columns (total 12 columns):
 #   Column        Non-Null Count  Dtype         
---  ------        --------------  -----         
 0   Date          182 non-null    datetime64[ns]
 1   Store         182 non-null    int64         
 2   Temperature   182 non-null    float64       
 3   Fuel_Price    182 non-null    float64       
 4   MarkDown1     90 non-null     float64       
 5   MarkDown2     73 non-null     float64       
 6   MarkDown3     89 non-null     float64       
 7   MarkDown4     90 non-null     float64       
 8   MarkDown5     90 non-null     float64       
 9   CPI           169 non-null    float64       
 10  Unemployment  169 non-null    float64       
 11  IsHoliday     182 non-null    bool          
dtypes: bool(1), datetime64[ns](1), float64(9), int64(1)
memory usage: 17.2 KB
```

4. Indexing & Slicing DataFrames

In [28]:
```
df[2:3] # Access rows
```

Out[28]:

	Date	Store	Temperature	Fuel_Price	MarkDown1	MarkDown2	MarkDow
Date							
2010-02-19	2010-02-19	1	39.93	2.514	NaN	NaN	N

In [29]:
```
df[-5:]
# df.tail(5)
```

Out[29]:

Date	Date	Store	Temperature	Fuel_Price	MarkDown1	MarkDown2	MarkDo
2013-06-28	2013-06-28	1	83.94	3.495	4205.98	796.70	
2013-07-05	2013-07-05	1	79.85	3.422	7649.99	3503.29	176
2013-07-12	2013-07-12	1	83.12	3.400	6089.94	1362.42	20
2013-07-19	2013-07-19	1	79.26	3.556	3117.04	1060.39	19
2013-07-26	2013-07-26	1	81.54	3.620	332.17	673.19	

In [30]:
```
df['CPI'] # Access columns.
```

Out[30]:
```
Date
2010-02-05    211.096358
2010-02-12    211.242170
2010-02-19    211.289143
2010-02-26    211.319643
2010-03-05    211.350143
                  ...
2013-06-28           NaN
2013-07-05           NaN
2013-07-12           NaN
2013-07-19           NaN
2013-07-26           NaN
Name: CPI, Length: 182, dtype: float64
```

In [31]:
```
df['CPI']['2010-02-05']
```

Out[31]:
```
211.0963582
```

In [32]:
```
df[['CPI', 'Unemployment']] # Multiple columns
```

Out[32]:

Date	CPI	Unemployment
2010-02-05	211.096358	8.106
2010-02-12	211.242170	8.106
2010-02-19	211.289143	8.106
2010-02-26	211.319643	8.106
2010-03-05	211.350143	8.106
...	...	...
2013-06-28	NaN	NaN
2013-07-05	NaN	NaN
2013-07-12	NaN	NaN
2013-07-19	NaN	NaN
2013-07-26	NaN	NaN

182 rows × 2 columns

In [33]:
```
df.iloc[0:2,[1,2]]
```

Out[33]:

Date	Store	Temperature
2010-02-05	1	42.31
2010-02-12	1	38.51

In [34]:
```
df.iloc[0,0]
```

Out[34]:
```
Timestamp('2010-02-05 00:00:00')
```

In [35]:
```
df.iloc[0, :]
```

Out[35]:
```
Date            2010-02-05 00:00:00
Store                             1
Temperature                   42.31
Fuel_Price                    2.572
MarkDown1                       NaN
MarkDown2                       NaN
MarkDown3                       NaN
MarkDown4                       NaN
MarkDown5                       NaN
CPI                      211.096358
Unemployment                  8.106
IsHoliday                     False
Name: 2010-02-05 00:00:00, dtype: object
```

In [36]:
```
df.iloc[:, 0]
```

Out[36]:
```
Date
2010-02-05   2010-02-05
2010-02-12   2010-02-12
2010-02-19   2010-02-19
2010-02-26   2010-02-26
2010-03-05   2010-03-05
                ...
2013-06-28   2013-06-28
2013-07-05   2013-07-05
2013-07-12   2013-07-12
2013-07-19   2013-07-19
2013-07-26   2013-07-26
Name: Date, Length: 182, dtype: datetime64[ns]
```

In [37]:
```
df.loc['2010-02-05', ['Temperature', 'Fuel_Price']]
```

Out[37]:
```
Temperature    42.31
Fuel_Price     2.572
Name: 2010-02-05 00:00:00, dtype: object
```

In [38]:
```
df.loc['2010-02-05']
```

Out[38]:
```
Date            2010-02-05 00:00:00
Store                             1
Temperature                   42.31
Fuel_Price                    2.572
MarkDown1                       NaN
MarkDown2                       NaN
MarkDown3                       NaN
MarkDown4                       NaN
MarkDown5                       NaN
CPI                      211.096358
Unemployment                  8.106
IsHoliday                     False
Name: 2010-02-05 00:00:00, dtype: object
```

Quiz 1

In [39]:
```
cars = pd.DataFrame([['P',1.5,110],['F',2.5,245]],columns = ['model','price'
```

In [40]:
```
cars.index = cars['model']
```

In [41]:
```
cars
```

Out[41]:

	model	price	HP
model			
P	P	1.5	110
F	F	2.5	245

1. How to get the first row without using labels

In [42]:
```
cars[:1]
```

Out[42]:

	model	price	HP
model			
P	P	1.5	110

In [43]:
```
cars.iloc[0,:]
```

Out[43]:
```
model      P
price    1.5
HP       110
Name: P, dtype: object
```

In [44]:
```
cars.iloc[:1,:]
```

Out[44]:

	model	price	HP
model			
P	P	1.5	110

2. How to get the last row?

In [45]:
```
cars
```

Out[45]:

	model	price	HP
model			
P	P	1.5	110
F	F	2.5	245

In [46]:
```
cars[-1:]
```

Out[46]:

	model	price	HP
model			
F	F	2.5	245

In [47]:
```
cars.iloc[-1,:]
```

Out[47]:
```
model      F
price    2.5
HP       245
Name: F, dtype: object
```

In [48]:
```
cars.loc['F']
```

Out[48]:
```
model      F
price    2.5
HP       245
Name: F, dtype: object
```

3. How to get the second element of column "HP"

In [49]:
```
cars[1:2]['HP']
```

Out[49]:
```
model
F    245
Name: HP, dtype: int64
```

In [50]:
```
cars['HP'][1:2]
```

Out[50]:
```
model
F    245
Name: HP, dtype: int64
```

In [51]:
```
cars.iloc[1]['HP']
```

Out[51]:
```
245
```

In [52]:
```
# cars.loc[1,'HP']
```

4. How to get the value of "HP" for model "F"

In [53]:
```
cars['HP']['F']
```

Out[53]:
```
245
```

In [54]:
```
cars.loc['F','HP']
```

Out[54]:
```
245
```

5. Boolean Indexing

In [55]:
```
cars = pd.DataFrame([['P',1.5,110],['F',2.5,245]],columns = ['model','price'
cars
```

Out[55]:

	model	price	HP
0	P	1.5	110
1	F	2.5	245

In [56]:
```
cars[ (cars['HP'] > 200) ]
```

Out[56]:

	model	price	HP
1	F	2.5	245

In [57]:
```
cars[ (cars['HP'] > 100) & (cars['price'] < 2) ]
```

Out[57]:

	model	price	HP
0	P	1.5	110

In [58]:
```
cars[ (cars['model'] == 'P') | (cars['HP']> 200) ]
```

Out[58]:

	model	price	HP
0	P	1.5	110
1	F	2.5	245

In [ ]:
```

```

In [59]:
```
cars.loc[ (cars['HP'] > 200)]
```

Out[59]:

	model	price	HP
1	F	2.5	245

In [60]:
```
cars.loc[ (cars['HP'] > 100) & (cars['price'] < 2) ,'HP']
```

Out[60]:
```
0    110
Name: HP, dtype: int64
```

In [61]:
```
cars.loc[ (cars['model'] == 'P') | (cars['HP']> 200) ]
```

Out[61]:

	model	price	HP
0	P	1.5	110
1	F	2.5	245

6. .isin() Example

In [62]:
```
data = pd.DataFrame({'model': ['a', 'b', 'c'],
                     'price' : [1,2,3]})
data
```

Out[62]:

	model	price
0	a	1
1	b	2
2	c	3

In [63]:
```
# data[ (data['model'] == 'a') | (data['model'] == 'c')]
```

In [64]:
```
data['model'].isin(['a','c'])
```

Out[64]:
```
0     True
1    False
2     True
Name: model, dtype: bool
```

In [65]:
```
data[data['model'].isin(['a','c'])]['price']
```

Out[65]:
```
0    1
2    3
Name: price, dtype: int64
```

7. Modifying DataFrames

In [66]:
```
temperature = df.pop('Temperature')
df
```

Out[66]:

	Date	Store	Fuel_Price	MarkDown1	MarkDown2	MarkDown3	MarkDow
Date							
2010-02-05	2010-02-05	1	2.572	NaN	NaN	NaN	N
2010-02-12	2010-02-12	1	2.548	NaN	NaN	NaN	N
2010-02-19	2010-02-19	1	2.514	NaN	NaN	NaN	N
2010-02-26	2010-02-26	1	2.561	NaN	NaN	NaN	N
2010-03-05	2010-03-05	1	2.625	NaN	NaN	NaN	N
...	...	...	...	...	...	...	
2013-06-28	2013-06-28	1	3.495	4205.98	796.70	6.84	3816
2013-07-05	2013-07-05	1	3.422	7649.99	3503.29	1766.77	9454
2013-07-12	2013-07-12	1	3.400	6089.94	1362.42	209.62	2367
2013-07-19	2013-07-19	1	3.556	3117.04	1060.39	199.05	1012
2013-07-26	2013-07-26	1	3.620	332.17	673.19	1.00	38

182 rows × 11 columns

In [67]:
```
df.insert(0,'Temperature',temperature)
df
```

Out[67]:

Date	Temperature	Date	Store	Fuel_Price	MarkDown1	MarkDown2	MarkDo
2010-02-05	42.31	2010-02-05	1	2.572	NaN	NaN	
2010-02-12	38.51	2010-02-12	1	2.548	NaN	NaN	
2010-02-19	39.93	2010-02-19	1	2.514	NaN	NaN	
2010-02-26	46.63	2010-02-26	1	2.561	NaN	NaN	
2010-03-05	46.50	2010-03-05	1	2.625	NaN	NaN	
...	...	...	...	...	...	...	
2013-06-28	83.94	2013-06-28	1	3.495	4205.98	796.70	
2013-07-05	79.85	2013-07-05	1	3.422	7649.99	3503.29	176
2013-07-12	83.12	2013-07-12	1	3.400	6089.94	1362.42	20
2013-07-19	79.26	2013-07-19	1	3.556	3117.04	1060.39	19
2013-07-26	81.54	2013-07-26	1	3.620	332.17	673.19	

182 rows × 12 columns

In [68]:
```
df1 = pd.DataFrame([['a', 1], ['b', 2]], columns = ['model', 'price'])
df1
```

Out[68]:

	model	price
0	a	1
1	b	2

In [69]:
```
df2 = pd.DataFrame([[4, 'd'], [3, 'c']], columns = ['price', 'model'])
df2
```

Out[69]:

	price	model
0	4	d
1	3	c

In [70]:
```
# When ignore_index is set to True,
# it resets the index of the resulting DataFrame
# to have a continuous range of row labels starting from 0.
# df1.append(df2, ignore_index=True)
```

In [71]:
```
concatenated_df = pd.concat([df1,df2])
concatenated_df
```

Out[71]:

	model	price
0	a	1
1	b	2
0	d	4
1	c	3

In [72]:
```
# Reset the index
concatenated_df.reset_index(drop=True, inplace=True)
concatenated_df
```

Out[72]:

	model	price
0	a	1
1	b	2
2	d	4
3	c	3

8. Converting Data to the Right Type

In [73]:
```
df = pd.read_csv('store1features.csv')
df
```

Out[73]:

	Date	Store	Temperature	Fuel_Price	MarkDown1	MarkDown2	MarkDown3	Ma
0	2010-02-05	1	42.31	2.572	NaN	NaN	NaN	
1	2010-02-12	1	38.51	2.548	NaN	NaN	NaN	
2	2010-02-19	1	39.93	2.514	NaN	NaN	NaN	
3	2010-02-26	1	46.63	2.561	NaN	NaN	NaN	
4	2010-03-05	1	46.50	2.625	NaN	NaN	NaN	
...	...	...	...	...	...	...	...	
177	2013-06-28	1	83.94	3.495	4205.98	796.70	6.84	
178	2013-07-05	1	79.85	3.422	7649.99	3503.29	1766.77	
179	2013-07-12	1	83.12	3.400	6089.94	1362.42	209.62	
180	2013-07-19	1	79.26	3.556	3117.04	1060.39	199.05	
181	2013-07-26	1	81.54	3.620	332.17	673.19	1.00	

182 rows × 12 columns

In [74]:
```
# df.loc['2010-02', ['Temperature', 'Fuel_Price']]
```

In [75]:
```
# df.index = df['Date']
# df.loc['2010-02', ['Temperature', 'Fuel_Price']]
```

In [76]:
```
df.index = pd.to_datetime(df['Date'])
df.loc['2010-02', ['Temperature', 'Fuel_Price']]
```

Out[76]:

Date	Temperature	Fuel_Price
2010-02-05	42.31	2.572
2010-02-12	38.51	2.548
2010-02-19	39.93	2.514
2010-02-26	46.63	2.561

In [77]:
```
# only select February data
df.loc[df.index.month == 2]
```

Out[77]:

Date	Date	Store	Temperature	Fuel_Price	MarkDown1	MarkDown2	MarkDo
2010-02-05	2010-02-05	1	42.31	2.572	NaN	NaN	
2010-02-12	2010-02-12	1	38.51	2.548	NaN	NaN	
2010-02-19	2010-02-19	1	39.93	2.514	NaN	NaN	
2010-02-26	2010-02-26	1	46.63	2.561	NaN	NaN	
2011-02-04	2011-02-04	1	42.27	2.989	NaN	NaN	
2011-02-11	2011-02-11	1	36.39	3.022	NaN	NaN	
2011-02-18	2011-02-18	1	57.36	3.045	NaN	NaN	
2011-02-25	2011-02-25	1	62.90	3.065	NaN	NaN	
2012-02-03	2012-02-03	1	56.55	3.360	34577.06	3579.21	16
2012-02-10	2012-02-10	1	48.02	3.409	13925.06	6927.23	10
2012-02-17	2012-02-17	1	45.32	3.510	9873.33	11062.27	
2012-02-24	2012-02-24	1	57.25	3.555	9349.61	7556.01	
2013-02-01	2013-02-01	1	56.46	3.244	9290.91	1359.90	26
2013-02-08	2013-02-08	1	56.67	3.417	32355.16	729.80	28
2013-02-15	2013-02-15	1	49.66	3.475	72937.29	6665.52	4
2013-02-22	2013-02-22	1	50.25	3.597	20107.75	3163.89	4

9. Sorting

In [78]:
```
df = pd.read_csv('store1features.csv')
df.index = pd.to_datetime(df['Date'])
df = df[['Temperature', 'Fuel_Price', 'IsHoliday']]
df.head()
```

Out[78]:

Date	Temperature	Fuel_Price	IsHoliday
2010-02-05	42.31	2.572	False
2010-02-12	38.51	2.548	True
2010-02-19	39.93	2.514	False
2010-02-26	46.63	2.561	False
2010-03-05	46.50	2.625	False

In [79]:
```
df_h = df.head()
df_h
```

Out[79]:

Date	Temperature	Fuel_Price	IsHoliday
2010-02-05	42.31	2.572	False
2010-02-12	38.51	2.548	True
2010-02-19	39.93	2.514	False
2010-02-26	46.63	2.561	False
2010-03-05	46.50	2.625	False

compare sort_values() vs sort()

In [80]:
```
df_h.sort_values(by='Temperature')
```

Out[80]:

Date	Temperature	Fuel_Price	IsHoliday
2010-02-12	38.51	2.548	True
2010-02-19	39.93	2.514	False
2010-02-05	42.31	2.572	False
2010-03-05	46.50	2.625	False
2010-02-26	46.63	2.561	False

In [81]:
```
df_h_n = np.array(df_h.iloc[:,:2])
df_h_n.sort(axis = 0)
df_h_n
```

Out[81]:
```
array([[38.51 ,  2.514],
       [39.93 ,  2.548],
       [42.31 ,  2.561],
       [46.5  ,  2.572],
       [46.63 ,  2.625]])
```

In [82]:
```
df.sort_values(by='Temperature', ascending=[False]).head()
```

Out[82]:

Date	Temperature	Fuel_Price	IsHoliday
2011-08-05	91.65	3.684	False
2011-08-12	90.76	3.638	False
2011-08-19	89.94	3.554	False
2011-07-15	88.54	3.575	False
2011-08-26	87.96	3.523	False

In [83]:
```
df.sort_values(by=['IsHoliday', 'Temperature'], ascending=[False,False])
```

Out[83]:

Date	Temperature	Fuel_Price	IsHoliday
2012-09-07	83.96	3.730	True
2010-09-10	78.69	2.565	True
2011-09-09	76.00	3.546	True
2010-11-26	64.52	2.735	True
2011-11-25	60.14	3.236	True
...	...	...	...
2010-02-05	42.31	2.572	False
2011-02-04	42.27	2.989	False
2013-01-04	41.73	3.161	False
2010-02-19	39.93	2.514	False
2011-01-14	35.40	2.983	False

182 rows × 3 columns

10. Sorting the index

In [84]:
```
subdf = df.sample(n=5)
# This is random, so the result will change every time you run it
subdf
```

Out[84]:

Date	Temperature	Fuel_Price	IsHoliday
2011-01-07	48.27	2.976	False
2010-07-16	83.15	2.623	False
2011-03-18	62.76	3.488	False
2012-08-31	80.49	3.638	False
2011-10-14	71.74	3.274	False

In [85]:
```
subdf.sort_index()
```

Out[85]:

Date	Temperature	Fuel_Price	IsHoliday
2010-07-16	83.15	2.623	False
2011-01-07	48.27	2.976	False
2011-03-18	62.76	3.488	False
2011-10-14	71.74	3.274	False
2012-08-31	80.49	3.638	False

11. Numerical operations

In [86]:
```
df = pd.read_csv('store1features.zip')
```

In [87]:
```
(df['Temperature'] - 32)*5/9 # Farenheit to Celsius # panda series
```

Out[87]:
```
0       5.727778
1       3.616667
2       4.405556
3       8.127778
4       8.055556
          ...
177    28.855556
178    26.583333
179    28.400000
180    26.255556
181    27.522222
Name: Temperature, Length: 182, dtype: float64
```

In [88]:
```
~df['IsHoliday']
```

Out[88]:
```
0       True
1      False
2       True
3       True
4       True
       ...
177     True
178     True
179     True
180     True
181     True
Name: IsHoliday, Length: 182, dtype: bool
```

In [89]:
```
df['Temperature'] > 90
```

Out[89]:
```
0      False
1      False
2      False
3      False
4      False
       ...
177    False
178    False
179    False
180    False
181    False
Name: Temperature, Length: 182, dtype: bool
```

In [90]:
```
df['Temperature'] * df['CPI']
```

Out[90]:
```
0      8931.486915
1      8134.935959
2      8436.775476
3      9853.834948
4      9827.781645
          ...
177            NaN
178            NaN
179            NaN
180            NaN
181            NaN
Length: 182, dtype: float64
```

12. Automatic index alignment

In [91]:
```
sales = pd.DataFrame([['a', 1, 1, -1],
                      ['b', 1, 2, 2],
                      ['a', 2, 3, -1],
                      ['b', 2, 4, 2]], columns = ['model', 'shop', 'sales', '
sales
```

Out[91]:

	model	shop	sales	price
0	a	1	1	-1
1	b	1	2	2
2	a	2	3	-1
3	b	2	4	2

In [92]:
```
print(sales['sales'] * sales['price'])
```

```
0   -1
1    4
2   -3
3    8
dtype: int64
```

In [93]:
```
prices = pd.DataFrame([[-1],[2]], columns = ['price'], index = ['a','b'])
prices
```

Out[93]:

	price
a	-1
b	2

In [94]:
```
sales['sales'] * prices['price']
```

Out[94]:
```
0   NaN
1   NaN
2   NaN
3   NaN
a   NaN
b   NaN
dtype: float64
```

In [95]:
```
sales.index = sales['model'] # We index by model.
sales
```

Out[95]:

	model	shop	sales	price
model				
a	a	1	1	-1
b	b	1	2	2
a	a	2	3	-1
b	b	2	4	2

In [96]:
```
sales['sales'] * prices['price'] # Now it works!
```

Out[96]:
```
a   -1
a   -3
b    4
b    8
dtype: int64
```

In [ ]:
```
```

In [97]:
```
prices = pd.DataFrame([[-1],[2],[3]], columns = ['price'], index = ['a','b',
prices
```

Out[97]:

	price
a	-1
b	2
c	3

In [98]:
```
sales['sales'] * prices['price']
```

Out[98]:
```
a   -1.0
a   -3.0
b    4.0
b    8.0
c    NaN
dtype: float64
```

13. Applying methods to DataFrame

In [99]:
```
import numpy as np
```

In [100…
```
df3 = pd.DataFrame(dict(a=np.arange(1,5),
                        b=np.arange(-1,-5, -1),
                        c=np.arange(6, 2, -1)))
df3
```

Out[100]:

	a	b	c
0	1	-1	6
1	2	-2	5
2	3	-3	4
3	4	-4	3

In [101…
```
df3.mean()
```

Out[101]:
```
a    2.5
b   -2.5
c    4.5
dtype: float64
```

In [102…
```
df3.abs()
```

Out[102]:

	a	b	c
0	1	1	6
1	2	2	5
2	3	3	4
3	4	4	3

In [103…
```
def pow2(x):
    return(x**2)
# df3.pow2() # AttributeError: 'DataFrame' object has no attribute 'pow2'
```

In [104…
```
df3.apply(pow2)
```

Out[104]:

	a	b	c
0	1	1	36
1	4	4	25
2	9	9	16
3	16	16	9

14 Aggregations: DataFrames Axis

In [105…
```
df = pd.DataFrame(dict(a=np.arange(1,5),
                       b=np.arange(-1,-5, -1),
                       c=np.arange(6, 2, -1)))
df
```

Out[105]:

	a	b	c
0	1	-1	6
1	2	-2	5
2	3	-3	4
3	4	-4	3

In [106…
```
df.apply(np.mean)
```

Out[106]:
```
a    2.5
b   -2.5
c    4.5
dtype: float64
```

In [107…
```
df.apply(np.mean, axis='columns')
# df.apply(np.mean, axis=1)
```

Out[107]:
```
0    2.000000
1    1.666667
2    1.333333
3    1.000000
dtype: float64
```

In [108…
```
df.apply(np.mean, axis='index')
# df.apply(np.mean, axis=0)
```

Out[108]:
```
a    2.5
b   -2.5
c    4.5
dtype: float64
```

15. Pandas uses NumPy internally

In [109…
```
df
```

Out[109]:

	a	b	c
0	1	-1	6
1	2	-2	5
2	3	-3	4
3	4	-4	3

In [110…
```
df.values
```

Out[110]:
```
array([[ 1, -1,  6],
       [ 2, -2,  5],
       [ 3, -3,  4],
       [ 4, -4,  3]])
```

Data Visualization

In [111…
```
import matplotlib
import matplotlib.pyplot as plt
plt.style.use('ggplot')
```

16. Visualisation

In [112…
```
import pandas as pd
df = pd.read_csv('store1features.csv')
df.index = pd.to_datetime(df.pop('Date'))
df.head(100)
```

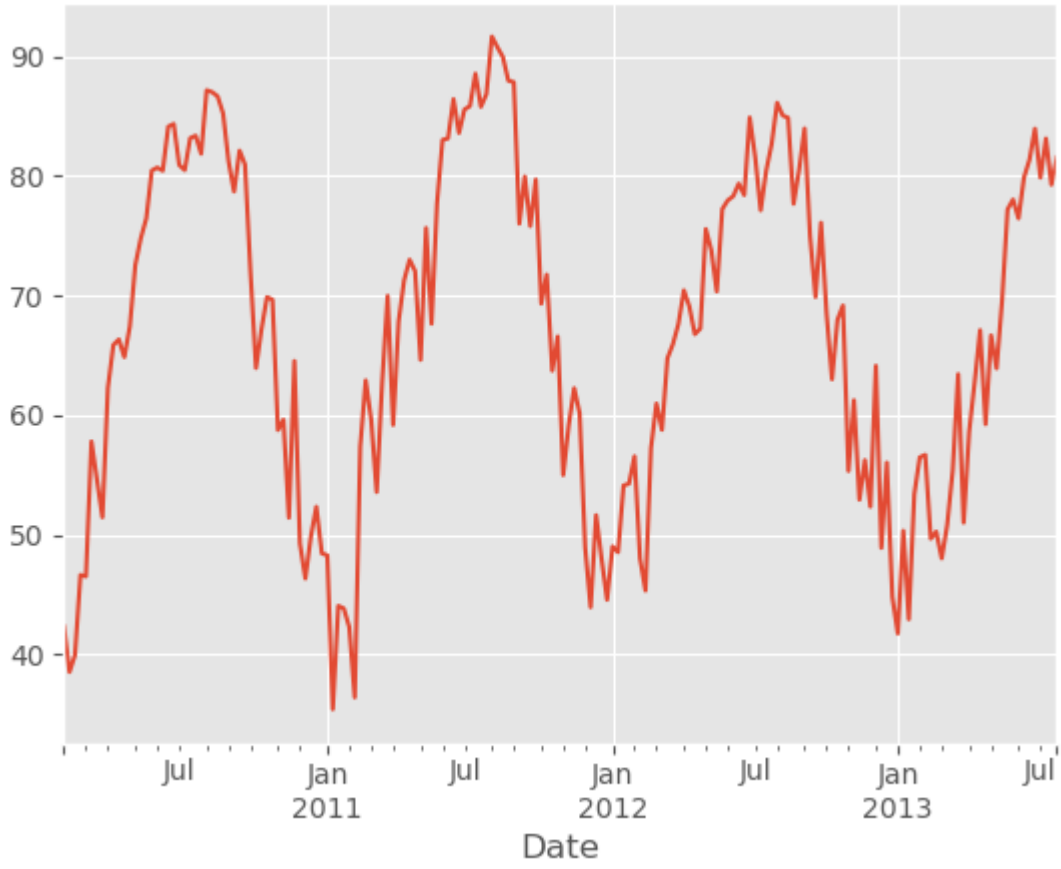
Out[112]:

Date	Store	Temperature	Fuel_Price	MarkDown1	MarkDown2	MarkDown3	MarkD
2010-02-05	1	42.31	2.572	NaN	NaN	NaN	
2010-02-12	1	38.51	2.548	NaN	NaN	NaN	
2010-02-19	1	39.93	2.514	NaN	NaN	NaN	
2010-02-26	1	46.63	2.561	NaN	NaN	NaN	
2010-03-05	1	46.50	2.625	NaN	NaN	NaN	
...	...	...	...	...	...	...	
2011-12-02	1	48.91	3.172	5629.51	68.00	1398.11	20
2011-12-09	1	43.93	3.158	4640.65	19.00	105.02	36
2011-12-16	1	51.63	3.159	5011.32	67.00	347.37	2
2011-12-23	1	47.96	3.112	2725.36	40.48	634.70	
2011-12-30	1	44.55	3.129	5762.10	46011.38	260.36	9

100 rows × 11 columns

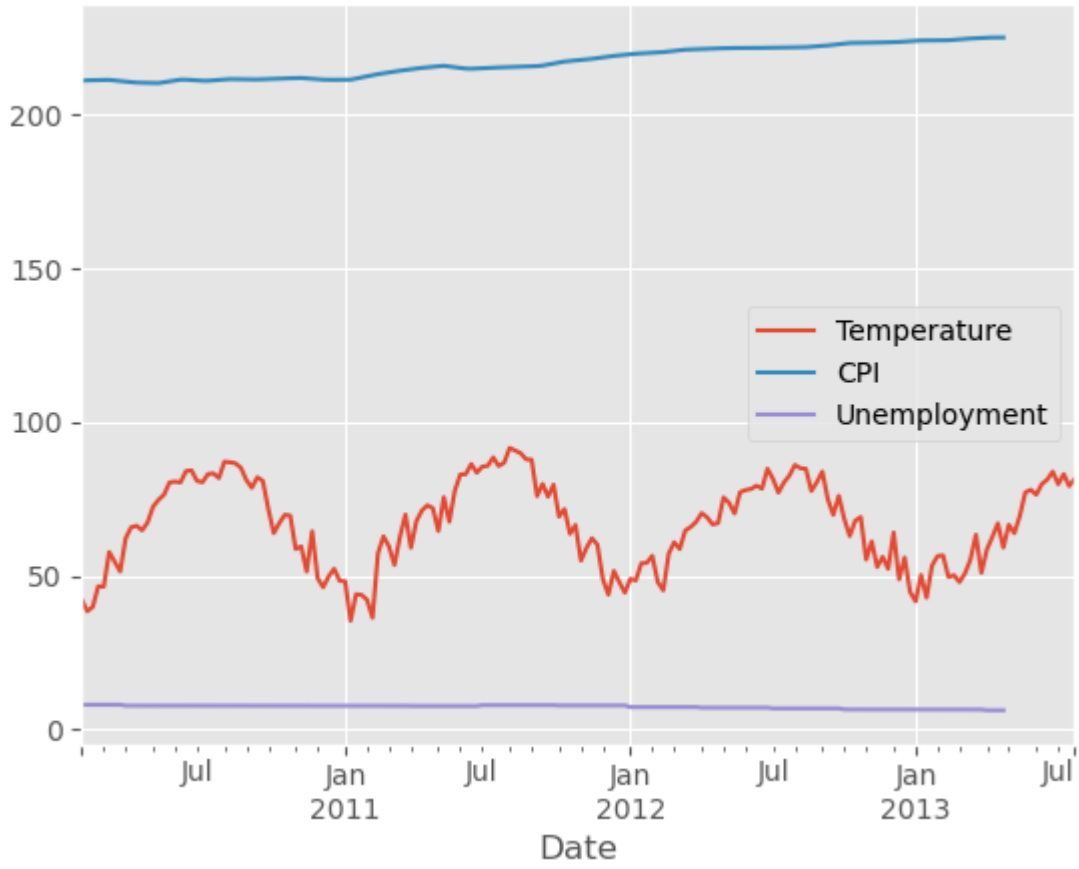
In [113…
```
df['Temperature'].plot()
```

Out[113]:
```
<Axes: xlabel='Date'>
```



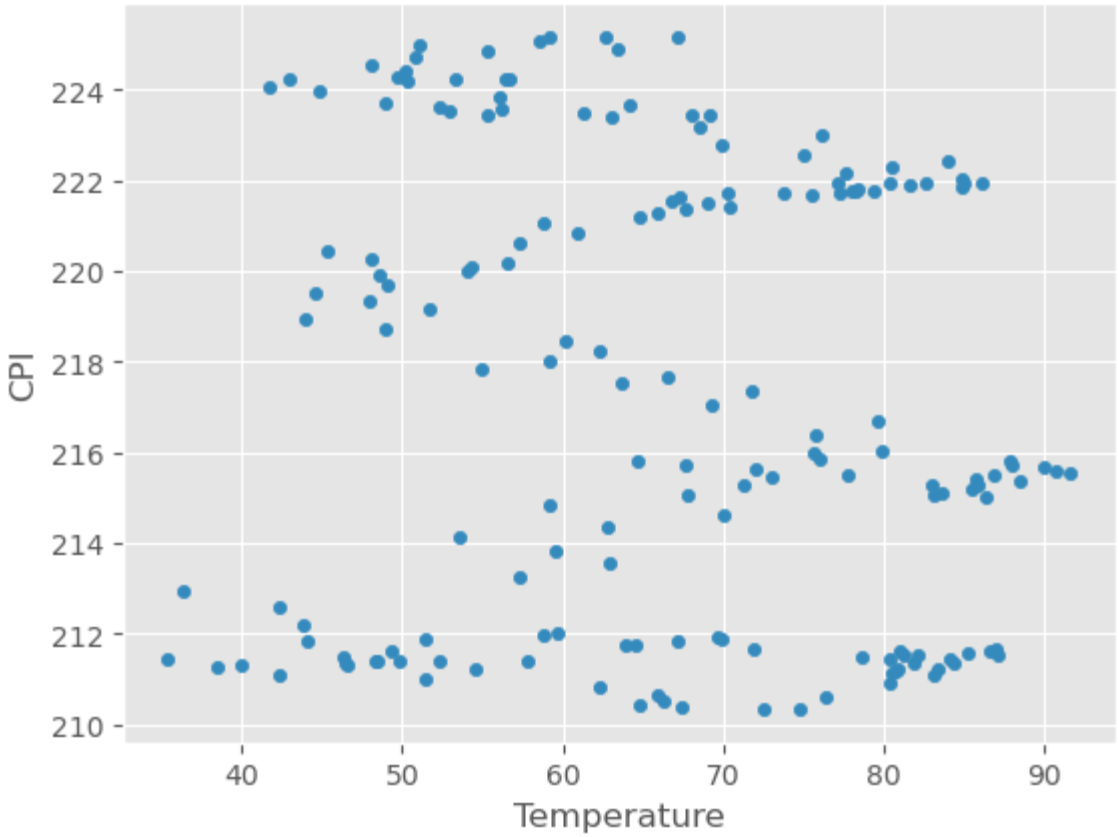
In [114…
```
df[ ['Temperature', 'CPI','Unemployment'] ].plot()
```

Out[114]: <Axes: xlabel='Date'>



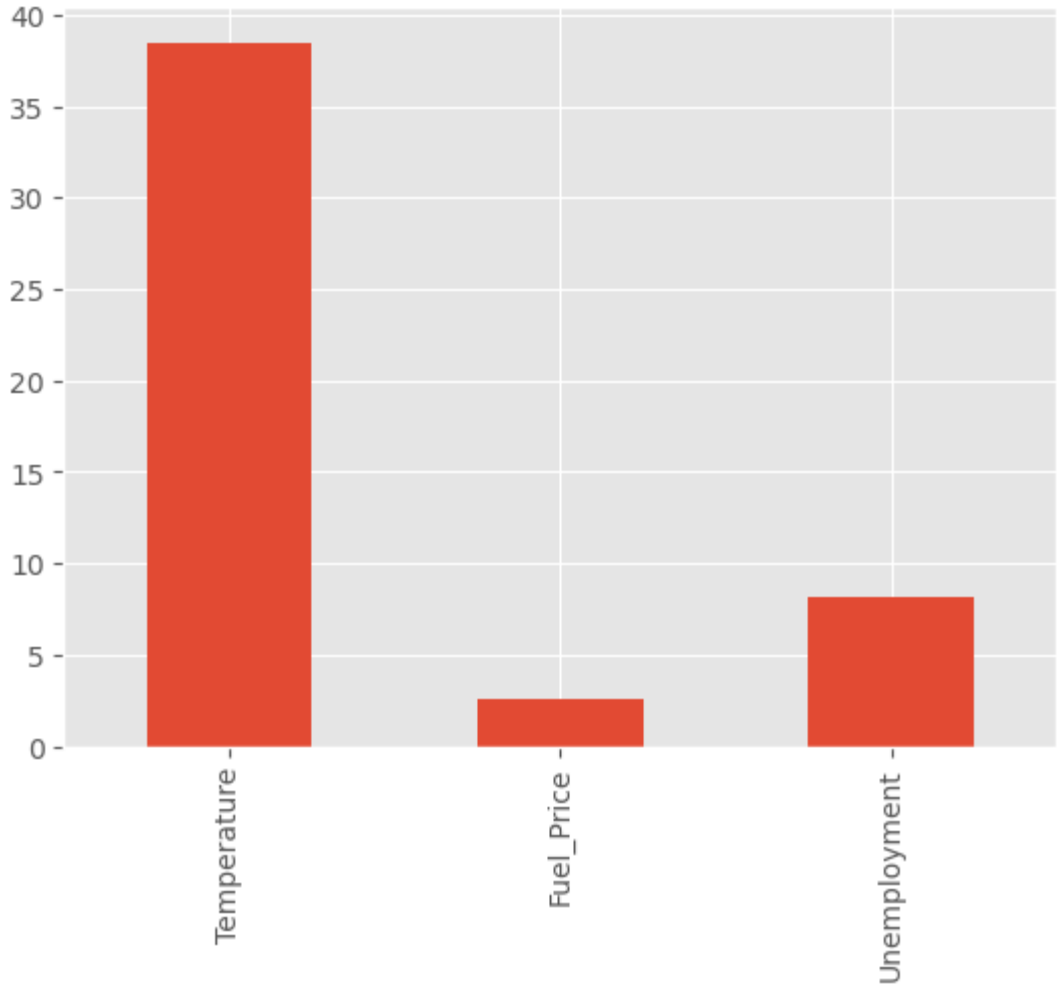
In [115…
```
df.plot.scatter(x = 'Temperature', y = 'CPI')
```

Out[115]: <Axes: xlabel='Temperature', ylabel='CPI'>



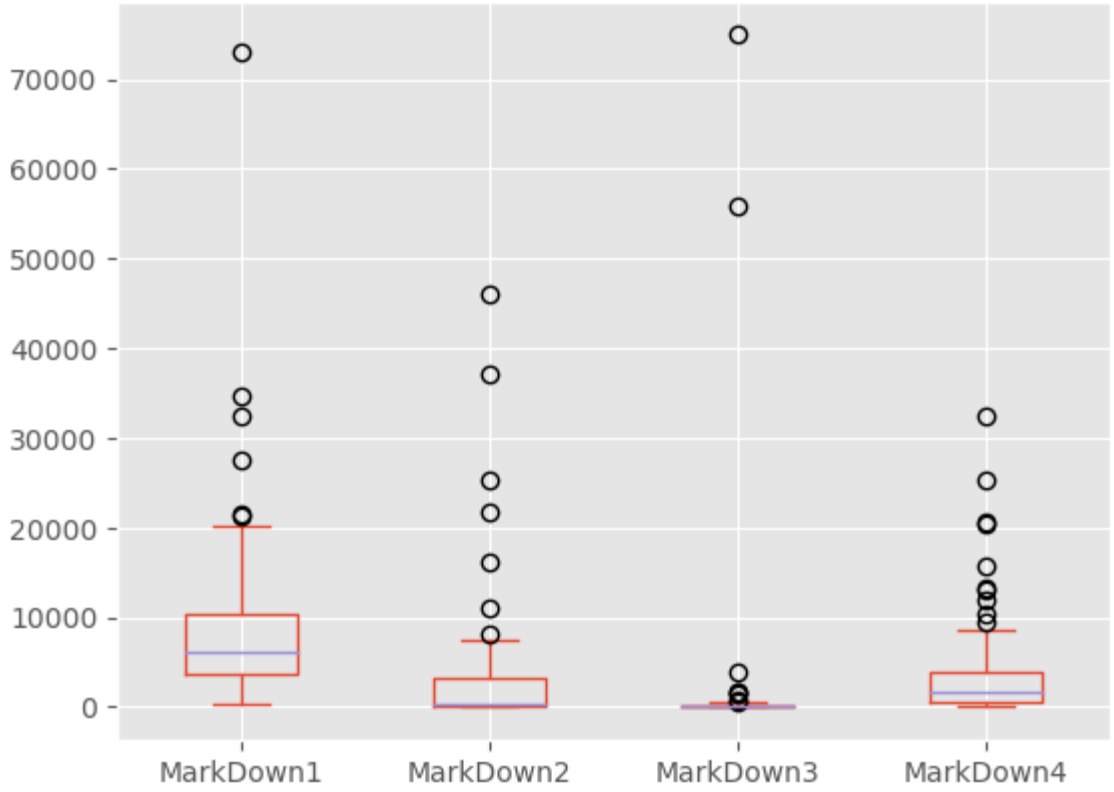
In [116…
```
df.iloc[1][['Temperature','Fuel_Price','Unemployment']].plot.bar()
```

Out[116]: <Axes: >



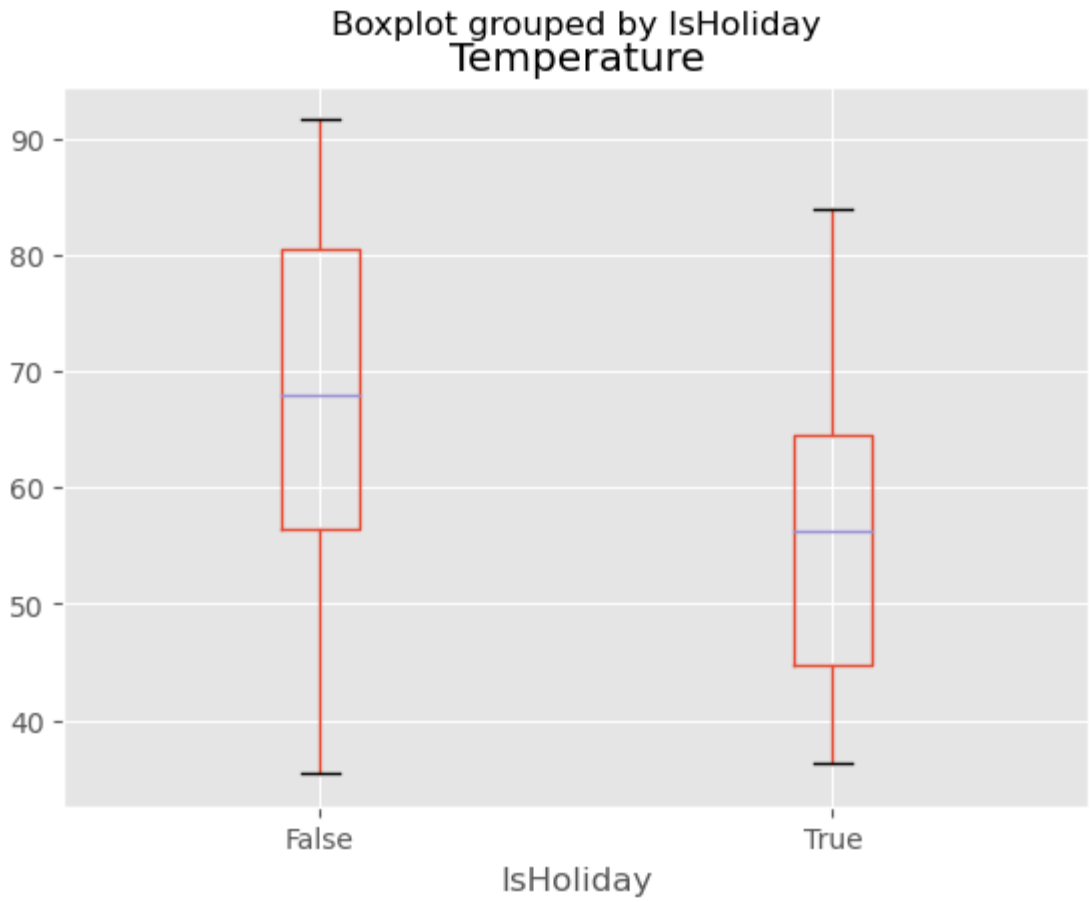
In [117…
```
df.loc[:,['MarkDown1','MarkDown2','MarkDown3','MarkDown4']].plot.box() # Som
```

Out[117]: <Axes: >



In [118…
```
df.boxplot(column = ['Temperature'], by = 'IsHoliday') # Some versions of Pa
# df.plot.box(column = ['Temperature'], by = 'IsHoliday') # Some versions of
```

Out[118]:
```
<Axes: title={'center': 'Temperature'}, xlabel='IsHoliday'>
```



17. Matplotlib step-by-step

In [119…

```
import numpy as np
df = pd.read_csv('store1features.csv')
df.index =  pd.to_datetime(df.pop('Date'))
df
```

Out[119]:

Date	Store	Temperature	Fuel_Price	MarkDown1	MarkDown2	MarkDown3	MarkD
2010-02-05	1	42.31	2.572	NaN	NaN	NaN	
2010-02-12	1	38.51	2.548	NaN	NaN	NaN	
2010-02-19	1	39.93	2.514	NaN	NaN	NaN	
2010-02-26	1	46.63	2.561	NaN	NaN	NaN	
2010-03-05	1	46.50	2.625	NaN	NaN	NaN	
...	...	...	...	...	...	...	
2013-06-28	1	83.94	3.495	4205.98	796.70	6.84	38
2013-07-05	1	79.85	3.422	7649.99	3503.29	1766.77	94
2013-07-12	1	83.12	3.400	6089.94	1362.42	209.62	23
2013-07-19	1	79.26	3.556	3117.04	1060.39	199.05	10
2013-07-26	1	81.54	3.620	332.17	673.19	1.00	

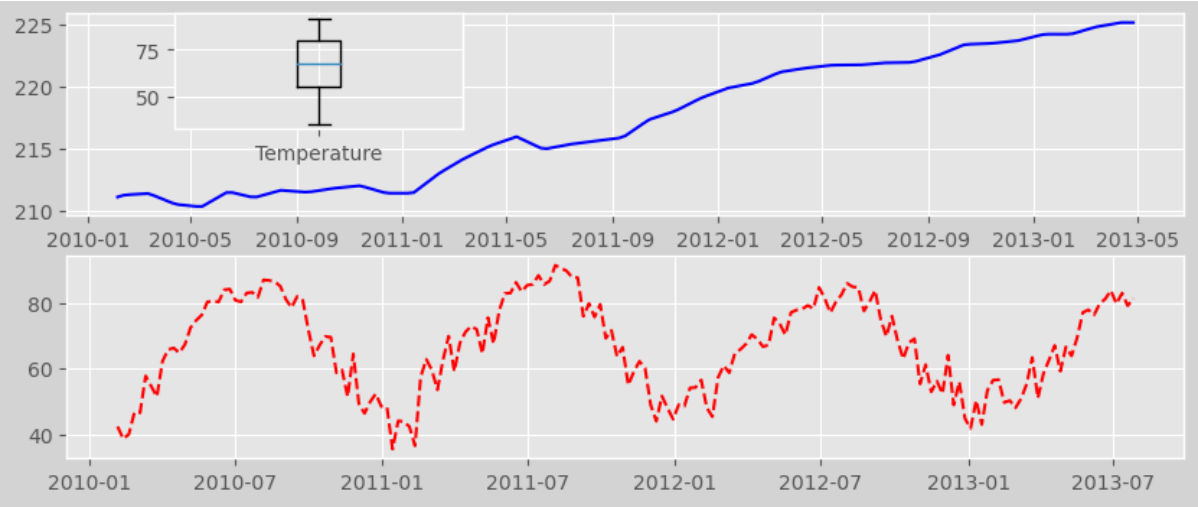
182 rows × 11 columns

In [120…

```
plt.figure(figsize=(10,4), facecolor = 'lightgray')
plt.subplot(2, 1, 1)
plt.plot_date(df.index, df['CPI'], 'b')
plt.subplot(2, 1, 2)
plt.plot_date(df.index,df['Temperature'],'r--')

# You're adding an axis that starts 20% from the left of the figure and 68%
# with a width and height both equal to 20% of the figure's dimensions.
plt.axes([.2, .68, .2, .2])
plt.boxplot(df["Temperature"], labels=["Temperature"])
```

Out[120]:

```
{'whiskers': [<matplotlib.lines.Line2D at 0x13246bdd0>,
  <matplotlib.lines.Line2D at 0x132474a90>],
 'caps': [<matplotlib.lines.Line2D at 0x132475710>,
  <matplotlib.lines.Line2D at 0x132476190>],
 'boxes': [<matplotlib.lines.Line2D at 0x1321a7490>],
 'medians': [<matplotlib.lines.Line2D at 0x132476c50>],
 'fliers': [<matplotlib.lines.Line2D at 0x132474810>],
 'means': []}
```



In [ ]:

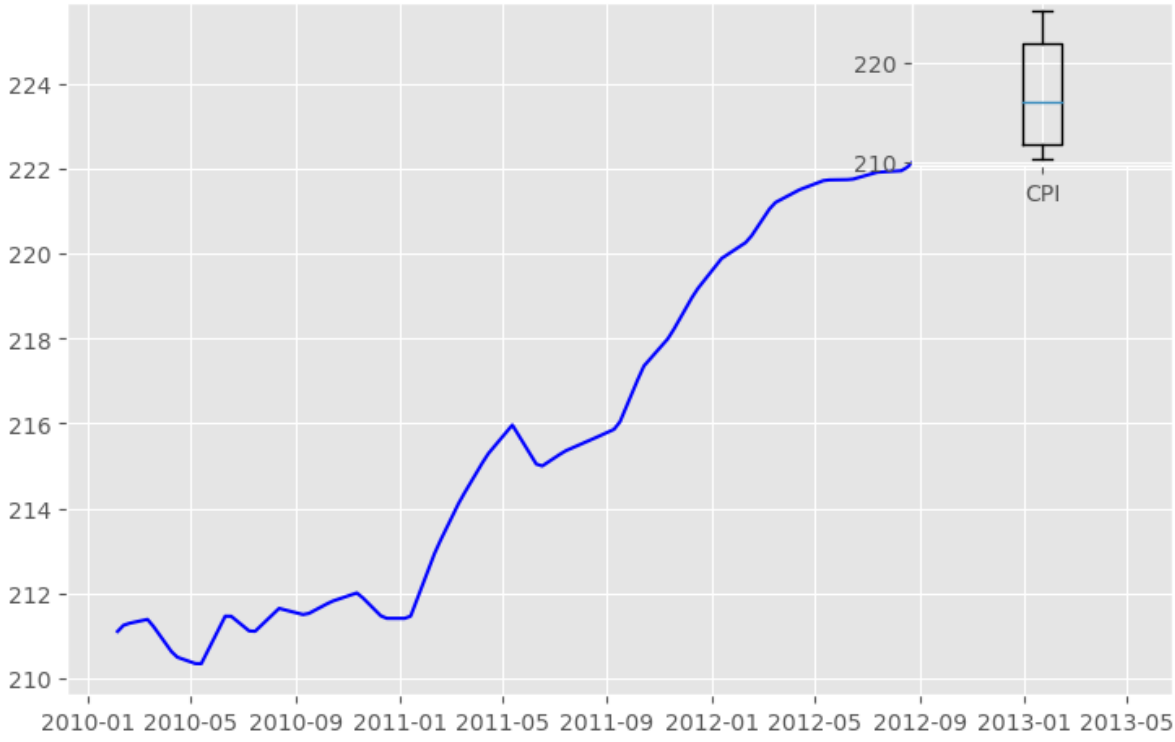
18. One-shot vs. Re-usability

In [121…

```
plt.figure(figsize=(8,5))
plt.axes([.1, .1, .85, .85])# [left, bottom, width, height]
plt.plot_date(df.index, df['CPI'], 'b')
plt.axes([.75, .75, .2, .2])
plt.boxplot(df['CPI'].dropna(), labels=["CPI"])
```

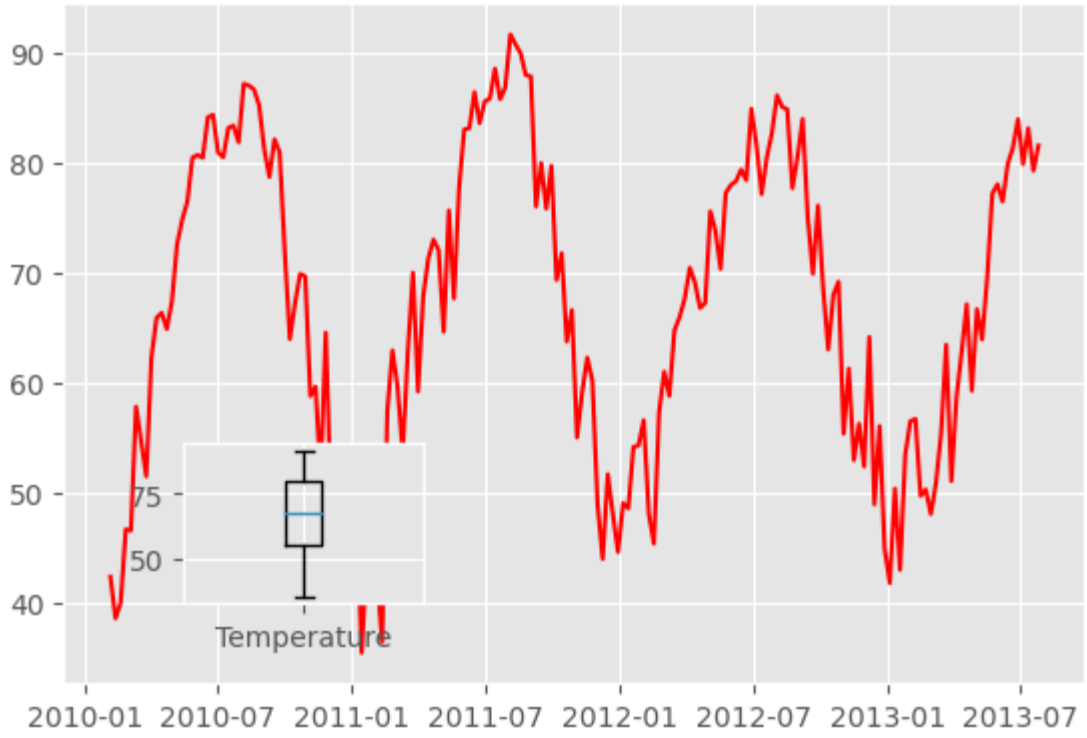
Out[121]:

```
{'whiskers': [<matplotlib.lines.Line2D at 0x132565310>,
  <matplotlib.lines.Line2D at 0x132565bd0>],
 'caps': [<matplotlib.lines.Line2D at 0x132566690>,
  <matplotlib.lines.Line2D at 0x132566e10>],
 'boxes': [<matplotlib.lines.Line2D at 0x1325649d0>],
 'medians': [<matplotlib.lines.Line2D at 0x132567690>],
 'fliers': [<matplotlib.lines.Line2D at 0x132566490>],
 'means': []}
```



In [122…
```
plt.figure(figsize=(6,4))
plt.axes([.1, .1, .85, .85])
plt.plot_date(df.index, df['Temperature'], 'r')
plt.axes([.2, .2, .2, .2])
plt.boxplot(df['Temperature'].dropna(), labels=['Temperature'])
```

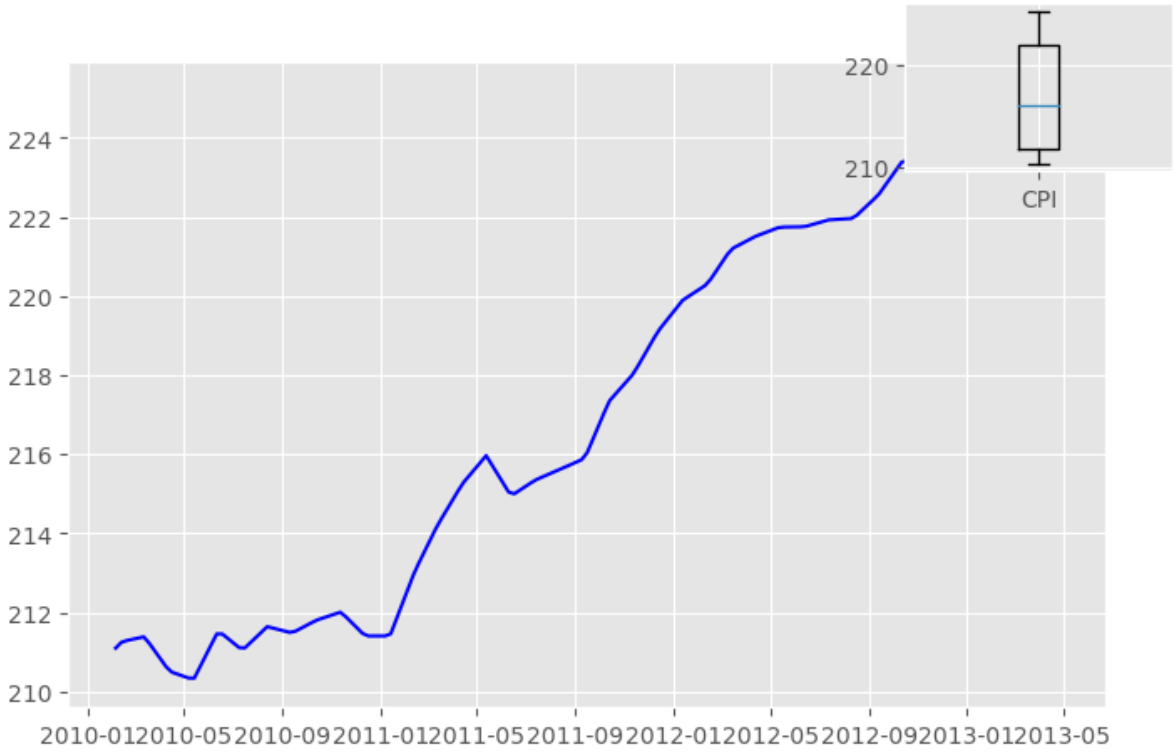
Out[122]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x13262f650>,
  <matplotlib.lines.Line2D at 0x13263c2d0>],
 'caps': [<matplotlib.lines.Line2D at 0x13263cf50>,
  <matplotlib.lines.Line2D at 0x13260f190>],
 'boxes': [<matplotlib.lines.Line2D at 0x13262e9d0>],
 'medians': [<matplotlib.lines.Line2D at 0x13263e210>],
 'fliers': [<matplotlib.lines.Line2D at 0x132583910>],
 'means': []}
```

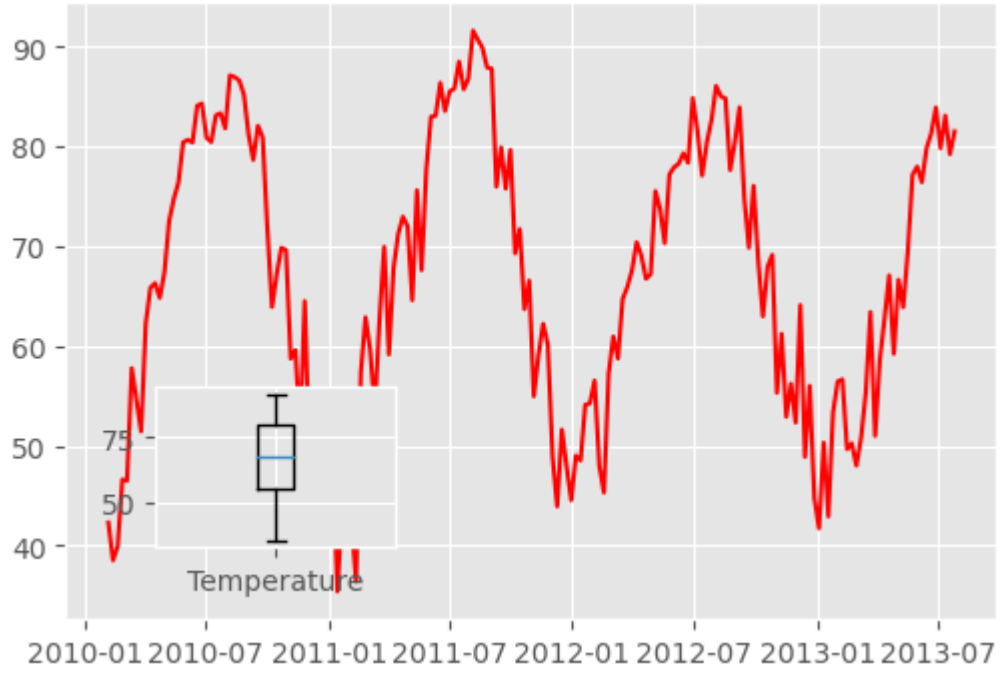


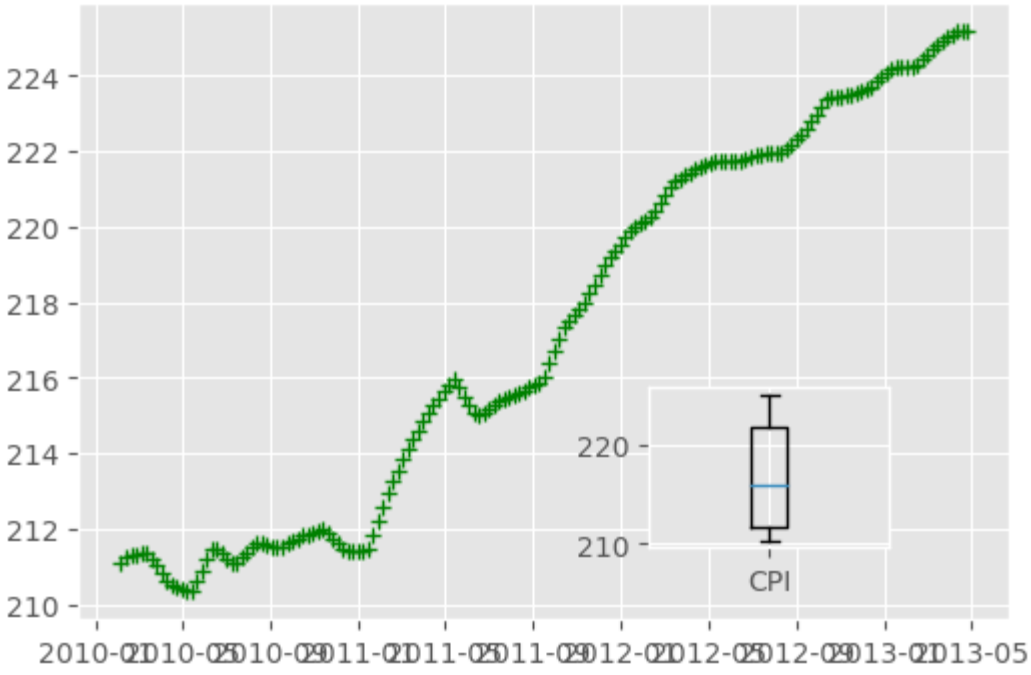
Re-usability

In [123…
```
def plot_date_boxplot(data, column, figsize = (6,4),
                      style = 'b', bxp_loc = [0.75,0.75]):
#     plt.style.use('classic')
    plt.style.use('ggplot')
    plt.figure(figsize = figsize)
    plt.plot_date(data.index, data[column], style)
    plt.axes(bxp_loc + [.2, .2])
    plt.boxplot(df[column].dropna(), labels = [column])
```

In [124…
```
plot_date_boxplot(df, 'CPI', figsize = (8,5))

plot_date_boxplot(df, 'Temperature', style = 'r', bxp_loc = [.2, .2])

plot_date_boxplot(df, 'CPI', style = 'g+', bxp_loc = [.6, .2])
```

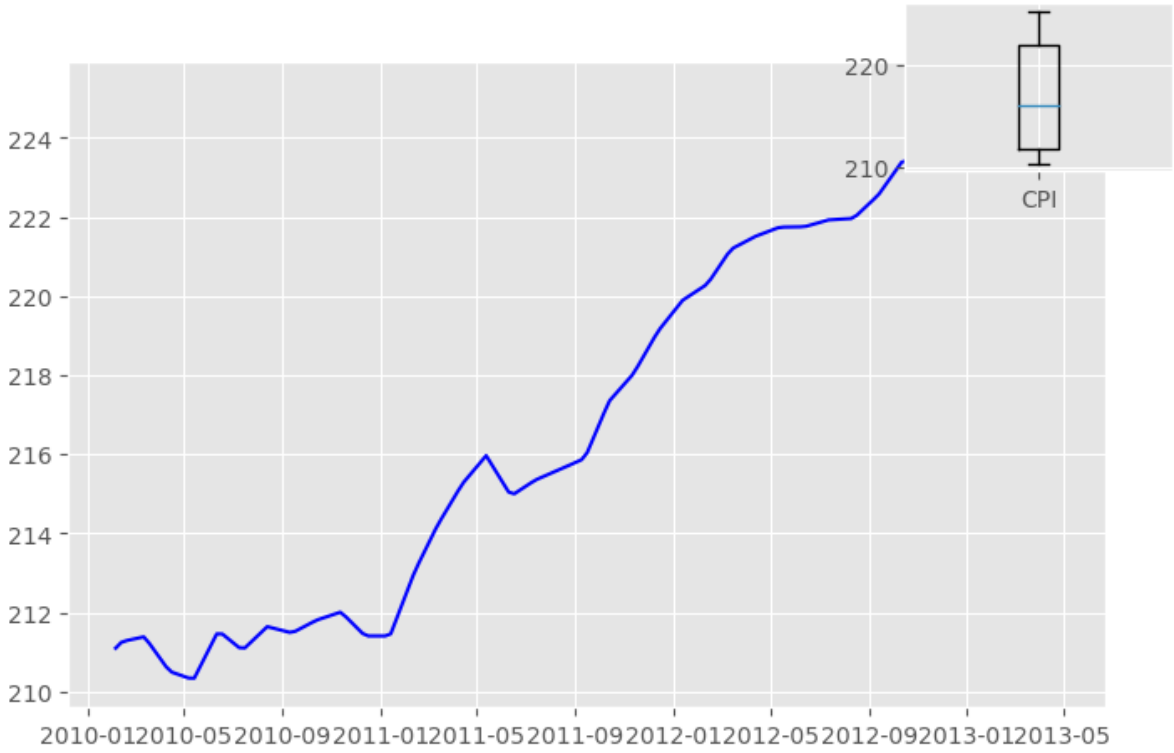

220
210
CPI
224
222
220
218
216
214
212
210
2010-01
2010-05
2010-09
2011-01
2011-05
2011-09
2012-01
2012-05
2012-09
2013-01
2013-05

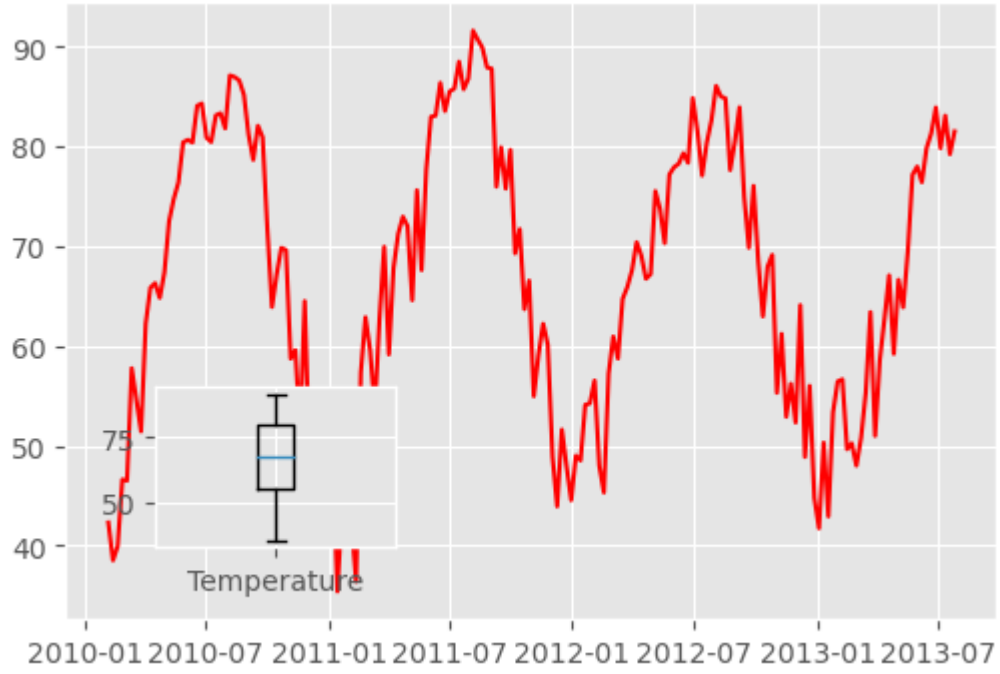

90
80
70
60
50
40
75
50
Temperature
2010-01
2010-07
2011-01
2011-07
2012-01
2012-07
2013-01
2013-07

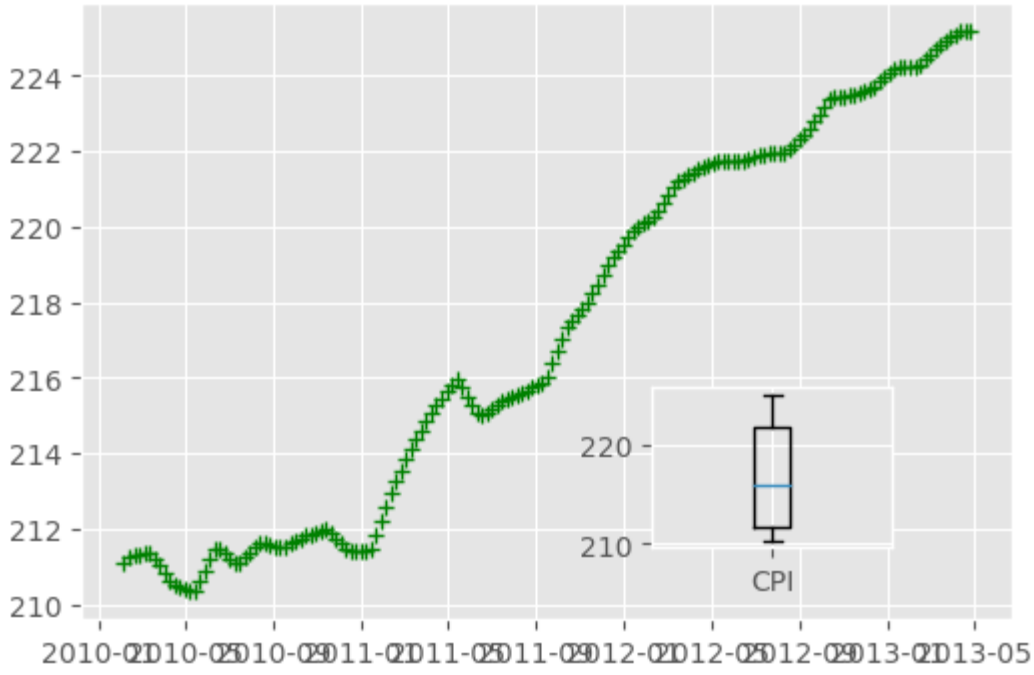


In [125…

```
import matplotlib.pyplot as plt

class CustomPlotter:
    def __init__(self, data):
        self.data = data

    def plot_date_boxplot(self, column, figsize=(6, 4), style='b', bxp_loc=[
        plt.style.use('ggplot')
        plt.figure(figsize=figsize)
        plt.plot_date(self.data.index, self.data[column], style)
        plt.axes(bxp_loc + [0.2, 0.2])
        plt.boxplot(self.data[column].dropna(), labels=[column])

custom_plotter = CustomPlotter(df)

custom_plotter.plot_date_boxplot('CPI', figsize = (8,5))
custom_plotter.plot_date_boxplot('Temperature', style = 'r', bxp_loc = [.2,
custom_plotter.plot_date_boxplot('CPI', style = 'g+', bxp_loc = [.6, .2])
```


220
210
CPI
224
222
220
218
216
214
212
210
2010-01
2010-05
2010-09
2011-01
2011-05
2011-09
2012-01
2012-05
2012-09
2013-01
2013-05


90
80
70
60
50
40
75
50
Temperature
2010-01
2010-07
2011-01
2011-07
2012-01
2012-07
2013-01
2013-07


224
222
220
218
216
214
212
210
220
210
CPI
2010-01
2010-05
2010-09
2011-01
2011-05
2011-09
2012-01
2012-05
2012-09
2013-01
2013-05

In [ ]: