Pandas

Pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language.

It will seamlessly bridge the gap between Python and Excel.

Jupyter Notebook

This is a web-based application (runs in the browser) that is used to interpret Python code.

- To add more code cells (or blocks) click on the **'+'** button in the top left corner
- There are 3 cell types in Jupyter:
  - Code: Used to write Python code
  - Markdown: Used to write texts (can be used to write explanations and other key information)
  - NBConvert: Used convert Jupyter (.ipynb) files to other formats (HTML, LaTex, etc.)
- To run Python code in a specific cell, you can click on the **'Run'** button at the top or press **Shift + Enter**
- The number sign (#) is used to insert comments when coding to leave messages for yourself or others. These comments will not be interpreted as code and are overlooked by the program

In [338]:
```
#Import pandas and assign it to a shorthand name pd
import pandas as pd
```

Reading CSV Files

- Function to use in Pandas: read_csv()
- Value passed to read_csv() must be string and the **exact** name of the file
- CSV Files must be in the same directory as the python file/notebook

In [339]:
```
#Read our data into a DataFrame names features_df
#read_excel does the same but for spreadsheet files
features_df = pd.read_csv('features.csv')

#print(df)
```

Basic DataFrame Functions

- head() will display the first 5 values of the DataFrame
- tail() will display the last 5 values of the DataFrame
- shape will display the dimensions of the DataFrame
- columns() will return the columns of the DataFrame as a list
- dtypes will display the types of each column of the DataFrame
- drop() will remove a column from the DataFrame

In [340]:
```
#Display top 5 rows
features_df.head()

#nan values are essentially empty entries
```

Out[340]:

	Store	Date	Temperature	Fuel_Price	MarkDown1	MarkDown2	MarkDown3	MarkDown4	MarkDown5
0	1	2010-02-05	42.31	2.572	NaN	NaN	NaN	NaN	NaN
1	1	2010-02-12	38.51	2.548	NaN	NaN	NaN	NaN	NaN
2	1	2010-02-19	39.93	2.514	NaN	NaN	NaN	NaN	NaN
3	1	2010-02-26	46.63	2.561	NaN	NaN	NaN	NaN	NaN
4	1	2010-03-05	46.50	2.625	NaN	NaN	NaN	NaN	NaN

In [341]:
```
#Display bottom 5 rows
features_df.tail()
```

Out[341]:

	Store	Date	Temperature	Fuel_Price	MarkDown1	MarkDown2	MarkDown3	MarkDown4	MarkDow
8185	45	2013-06-28	76.05	3.639	4842.29	975.03	3.00	2449.97	3169
8186	45	2013-07-05	77.50	3.614	9090.48	2268.58	582.74	5797.47	1514
8187	45	2013-07-12	79.37	3.614	3789.94	1827.31	85.72	744.84	2150
8188	45	2013-07-19	82.84	3.737	2961.49	1047.07	204.19	363.00	1059
8189	45	2013-07-26	76.06	3.804	212.02	851.73	2.06	10.88	1864

In [342]:
```
#Print dimensions of DataFrame as tuple
features_df.shape
```

Out[342]:
```
(8190, 12)
```

In [343]:
```
#Print list of column values
features_df.columns
```

Out[343]:
```
Index(['Store', 'Date', 'Temperature', 'Fuel_Price', 'MarkDown1', 'MarkDown2',
       'MarkDown3', 'MarkDown4', 'MarkDown5', 'CPI', 'Unemployment',
       'IsHoliday'],
      dtype='object')
```

In [344]:
```
#We can rename all columns at once by reassigning the .columns attribute
#Copy paste output from cell above and change column names accordingly
features_df.columns = ['Store', 'Date', 'Temperature', 'Fuel_Price', 'MD1',
                       'MD2', 'MD3', 'MD4','MD5', 'CPI', 'Unemployment', 'IsHolida

features_df.head()
```

Out[344]:

	Store	Date	Temperature	Fuel_Price	MD1	MD2	MD3	MD4	MD5	CPI	Unemployment	IsHol
0	1	2010-02-05	42.31	2.572	NaN	NaN	NaN	NaN	NaN	211.096358	8.106	F
1	1	2010-02-12	38.51	2.548	NaN	NaN	NaN	NaN	NaN	211.242170	8.106	
2	1	2010-02-19	39.93	2.514	NaN	NaN	NaN	NaN	NaN	211.289143	8.106	F
3	1	2010-02-26	46.63	2.561	NaN	NaN	NaN	NaN	NaN	211.319643	8.106	F
4	1	2010-03-05	46.50	2.625	NaN	NaN	NaN	NaN	NaN	211.350143	8.106	F

In [345]:
```
#To only rename specific columns
features_df.rename(columns={'Temperature': 'Temp'}, inplace=True)
```

In [346]:
```
#Print Pandas-specific data types of all columns
features_df.dtypes
```

Out[346]:
```
Store             int64
Date             object
Temp            float64
Fuel_Price      float64
MD1             float64
MD2             float64
MD3             float64
MD4             float64
MD5             float64
CPI             float64
Unemployment    float64
IsHoliday          bool
dtype: object
```

Indexing and Series Functions

- Columns of a DataFrame can be accessed through the following format: df_name["name_of_column"]
- Columns will be returned as a Series, which have different methods than DataFrames
- A couple useful Series functions: max(), median(), min(), value_counts(), sort_values()

In [347]:
```
#Extract CPI column of features_df
features_df["CPI"].head()
```

Out[347]:
```
0    211.096358
1    211.242170
2    211.289143
3    211.319643
4    211.350143
Name: CPI, dtype: float64
```

In [348]:
```
#We can use the in keyword as seen in Python 1

1 in features_df['Store']
```

Out[348]:
```
True
```

In [349]:
```
#Check the number of dimensions of our Data with 'ndim'
#Display the dimensions with 'shape'
#Display the total number of entries with 'shape'
# Example with our DataFrame
print(features_df.ndim)
print(features_df.shape)
print(features_df.size)
```

```
2
(8190, 12)
98280
```

In [350]:
```
# Example with our a column from our DataFrame

print(features_df['CPI'].ndim)
print(features_df['CPI'].shape)
print(features_df['CPI'].size)
```

```
1
(8190,)
8190
```

In [351]:
```
#Maximum value in Series
features_df["CPI"].max()
```

Out[351]:
```
228.9764563
```

In [352]:
```
#Median value in Series
features_df["CPI"].median()
```

Out[352]:
```
182.7640032
```

In [353]:
```
#Minimum value in Series
features_df["CPI"].min()
```

Out[353]:
```
126.064
```

In [354]:
```
#Basic Statistical Summary of a column
features_df['Temp'].describe()
```

Out[354]:
```
count    8190.000000
mean       59.356198
std        18.678607
min        -7.290000
25%        45.902500
50%        60.710000
75%        73.880000
max       101.950000
Name: Temp, dtype: float64
```

In [355]:
```
# To Check if the values in a column are unique

features_df['Store'].is_unique
```

Out[355]:
```
False
```

In [356]:
```
#Print list of unique values
features_df["Store"].unique()
```

Out[356]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
       18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34,
       35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45], dtype=int64)
```

In [357]:
```
#Print unique values and frequency
features_df["Date"].value_counts()
```

Out[357]:
```
2011-01-28    45
2011-02-04    45
2010-07-23    45
2012-03-02    45
2012-04-06    45
2012-12-07    45
2011-03-25    45
2010-10-22    45
2012-07-06    45
2011-07-22    45
2013-03-01    45
2011-09-30    45
2011-07-29    45
2012-04-13    45
2012-03-16    45
2011-08-12    45
2012-10-05    45
2010-05-14    45
2012-08-10    45
2010-03-05    45
2012-08-24    45
2011-10-28    45
2012-06-08    45
2010-08-20    45
2010-02-26    45
2012-07-13    45
2010-12-03    45
2013-05-17    45
2011-11-04    45
2010-07-02    45
              ..
2010-11-05    45
2012-03-09    45
2011-01-14    45
2011-03-11    45
2013-02-15    45
2010-11-26    45
2010-09-24    45
2013-06-21    45
2012-11-23    45
2013-05-31    45
2011-08-26    45
2013-01-25    45
2012-02-24    45
2011-12-23    45
2011-06-10    45
2012-05-04    45
2011-10-14    45
2013-03-15    45
2011-06-03    45
2012-10-12    45
2010-03-12    45
2012-02-03    45
2012-11-30    45
2011-08-19    45
2010-06-25    45
2011-11-11    45
2010-10-08    45
2010-02-05    45
2012-12-21    45
```

In [358]:
```
#Return a sorted DataFrame acording to specified column
features_df.sort_values(by = "Date", ascending = True)
features_df.head()
```

Out[358]:

	Store	Date	Temp	Fuel_Price	MD1	MD2	MD3	MD4	MD5	CPI	Unemployment	IsHoliday
0	1	2010-02-05	42.31	2.572	NaN	NaN	NaN	NaN	NaN	211.096358	8.106	False
1	1	2010-02-12	38.51	2.548	NaN	NaN	NaN	NaN	NaN	211.242170	8.106	True
2	1	2010-02-19	39.93	2.514	NaN	NaN	NaN	NaN	NaN	211.289143	8.106	False
3	1	2010-02-26	46.63	2.561	NaN	NaN	NaN	NaN	NaN	211.319643	8.106	False
4	1	2010-03-05	46.50	2.625	NaN	NaN	NaN	NaN	NaN	211.350143	8.106	False

In [359]:
```
features_df.head()
```

Out[359]:

	Store	Date	Temp	Fuel_Price	MD1	MD2	MD3	MD4	MD5	CPI	Unemployment	IsHoliday
0	1	2010-02-05	42.31	2.572	NaN	NaN	NaN	NaN	NaN	211.096358	8.106	False
1	1	2010-02-12	38.51	2.548	NaN	NaN	NaN	NaN	NaN	211.242170	8.106	True
2	1	2010-02-19	39.93	2.514	NaN	NaN	NaN	NaN	NaN	211.289143	8.106	False
3	1	2010-02-26	46.63	2.561	NaN	NaN	NaN	NaN	NaN	211.319643	8.106	False
4	1	2010-03-05	46.50	2.625	NaN	NaN	NaN	NaN	NaN	211.350143	8.106	False

In [360]:
```
# Drop duplicates for categorical data across a specific column

len(features_df.drop_duplicates(subset=['Store']))
```

Out[360]: 45

In [361]:
```
# delete one column
features_df.drop(columns = "MD1").tail()
```

Out[361]:

	Store	Date	Temp	Fuel_Price	MD2	MD3	MD4	MD5	CPI	Unemployment	IsHoliday
8185	45	2013-06-28	76.05	3.639	975.03	3.00	2449.97	3169.69	NaN	NaN	False
8186	45	2013-07-05	77.50	3.614	2268.58	582.74	5797.47	1514.93	NaN	NaN	False
8187	45	2013-07-12	79.37	3.614	1827.31	85.72	744.84	2150.36	NaN	NaN	False
8188	45	2013-07-19	82.84	3.737	1047.07	204.19	363.00	1059.46	NaN	NaN	False
8189	45	2013-07-26	76.06	3.804	851.73	2.06	10.88	1864.57	NaN	NaN	False

In [362]:
```
#Matrix of missing values
features_df.isnull().head()
```

Out[362]:

	Store	Date	Temp	Fuel_Price	MD1	MD2	MD3	MD4	MD5	CPI	Unemployment	IsHoliday
0	False	False	False	False	True	True	True	True	True	False	False	False
1	False	False	False	False	True	True	True	True	True	False	False	False
2	False	False	False	False	True	True	True	True	True	False	False	False
3	False	False	False	False	True	True	True	True	True	False	False	False
4	False	False	False	False	True	True	True	True	True	False	False	False

In [363]:
```
#Find the number of missing values per column
features_df.isnull().sum()
```

Out[363]:
```
Store              0
Date               0
Temp               0
Fuel_Price         0
MD1             4158
MD2             5269
MD3             4577
MD4             4726
MD5             4140
CPI              585
Unemployment     585
IsHoliday          0
dtype: int64
```

In [364]:
```
#Delete any row with missing data (NaN) in it

features_df.dropna().head()
```

Out[364]:

	Store	Date	Temp	Fuel_Price	MD1	MD2	MD3	MD4	MD5	CPI	Unemplo
92	1	2011-11-11	59.11	3.297	10382.90	6115.67	215.07	2406.62	6551.42	217.998085	
93	1	2011-11-18	62.25	3.308	6074.12	254.39	51.98	427.39	5988.57	218.220509	
94	1	2011-11-25	60.14	3.236	410.31	98.00	55805.51	8.00	554.92	218.467621	
95	1	2011-12-02	48.91	3.172	5629.51	68.00	1398.11	2084.64	20475.32	218.714733	
96	1	2011-12-09	43.93	3.158	4640.65	19.00	105.02	3639.42	14461.82	218.961846	

In [365]:
```
#Delete any column with missing data (NaN) in it
features_df.dropna(axis=1).head()
```

Out[365]:

	Store	Date	Temp	Fuel_Price	IsHoliday
0	1	2010-02-05	42.31	2.572	False
1	1	2010-02-12	38.51	2.548	True
2	1	2010-02-19	39.93	2.514	False
3	1	2010-02-26	46.63	2.561	False
4	1	2010-03-05	46.50	2.625	False

In [366]:
```
#Look along specific columns for NaN

features_df.dropna(subset=['Unemployment']).tail()
```

Out[366]:

	Store	Date	Temp	Fuel_Price	MD1	MD2	MD3	MD4	MD5	CPI	Unemploy
8172	45	2013-03-29	40.68	3.784	5444.00	NaN	350.84	53.90	1722.11	193.442790	
8173	45	2013-04-05	43.94	3.763	16427.83	5341.41	182.59	1523.83	1743.09	193.516047	
8174	45	2013-04-12	57.39	3.724	8760.15	1713.11	21.08	1302.31	1380.74	193.589304	
8175	45	2013-04-19	56.27	3.676	1399.81	39.89	44.38	60.83	1445.05	193.589304	
8176	45	2013-04-26	50.64	3.615	1260.65	NaN	57.52	40.51	2476.18	193.589304	

In [367]:
```
#Replace NaN (empty) values with 0's
features_df.fillna(0).head()
```

Out[367]:

	Store	Date	Temp	Fuel_Price	MD1	MD2	MD3	MD4	MD5	CPI	Unemployment	IsHoliday
0	1	2010-02-05	42.31	2.572	0.0	0.0	0.0	0.0	0.0	211.096358	8.106	False
1	1	2010-02-12	38.51	2.548	0.0	0.0	0.0	0.0	0.0	211.242170	8.106	True
2	1	2010-02-19	39.93	2.514	0.0	0.0	0.0	0.0	0.0	211.289143	8.106	False
3	1	2010-02-26	46.63	2.561	0.0	0.0	0.0	0.0	0.0	211.319643	8.106	False
4	1	2010-03-05	46.50	2.625	0.0	0.0	0.0	0.0	0.0	211.350143	8.106	False

In [368]:
```
# delete multiple columns
features_df.drop(columns = ['MD1', 'MD2', 'MD3', 'MD4', 'MD5'], inplace = True)
```

In [369]:
```
features_df.head()
```

Out[369]:

	Store	Date	Temp	Fuel_Price	CPI	Unemployment	IsHoliday
0	1	2010-02-05	42.31	2.572	211.096358	8.106	False
1	1	2010-02-12	38.51	2.548	211.242170	8.106	True
2	1	2010-02-19	39.93	2.514	211.289143	8.106	False
3	1	2010-02-26	46.63	2.561	211.319643	8.106	False
4	1	2010-03-05	46.50	2.625	211.350143	8.106	False

In [370]:
```
#Define a function to convert float values to our custom categorical ranges

def temp_categorical(temp):
    if temp < 50:
        return 'Mild'
    elif temp >= 50 and temp < 80:
        return 'Warm'
    else:
        return 'Hot'
```

In [371]:
```
#With the apply() function we can apply our custom function to each value of the S

features_df['Temp'] = features_df['Temp'].apply(temp_categorical)
```

In [372]:
```
features_df['Temp'].tail()
```

Out[372]:
```
8185    Warm
8186    Warm
8187    Warm
8188     Hot
8189    Warm
Name: Temp, dtype: object
```

In [373]:
```
#If we would like to define a 'one time use' anonymous function, we can use the 'l

features_df['Unemployment'].apply(lambda num: num + 1).head()
```

Out[373]:
```
0    9.106
1    9.106
2    9.106
3    9.106
4    9.106
Name: Unemployment, dtype: float64
```

In [374]:
```
features_df.head()
```

Out[374]:

	Store	Date	Temp	Fuel_Price	CPI	Unemployment	IsHoliday
0	1	2010-02-05	Mild	2.572	211.096358	8.106	False
1	1	2010-02-12	Mild	2.548	211.242170	8.106	True
2	1	2010-02-19	Mild	2.514	211.289143	8.106	False
3	1	2010-02-26	Mild	2.561	211.319643	8.106	False
4	1	2010-03-05	Mild	2.625	211.350143	8.106	False

Indexing

- Because Pandas will select entries based on column values by default, selecting data based on row values requires the use of the iloc method.
- Allowed inputs are:
  - An integer, e.g. 5.
  - A list or array of integers, e.g. [4, 3, 0].
  - A slice object with ints, e.g. 1:7.

In [375]:
```
#Return Fuel_Price to IsHoliday columns of 0-10th rows
#Note how LOC can reference columns by their names
features_df.loc[0:10,"Fuel_Price":"IsHoliday"]
```

Out[375]:

	Fuel_Price	CPI	Unemployment	IsHoliday
0	2.572	211.096358	8.106	False
1	2.548	211.242170	8.106	True
2	2.514	211.289143	8.106	False
3	2.561	211.319643	8.106	False
4	2.625	211.350143	8.106	False
5	2.667	211.380643	8.106	False
6	2.720	211.215635	8.106	False
7	2.732	211.018042	8.106	False
8	2.719	210.820450	7.808	False
9	2.770	210.622857	7.808	False
10	2.808	210.488700	7.808	False

In [376]:
```
features_df.loc[100:105]
```

Out[376]:

	Store	Date	Temp	Fuel_Price	CPI	Unemployment	IsHoliday
100	1	2012-01-06	Mild	3.157	219.714258	7.348	False
101	1	2012-01-13	Mild	3.261	219.892526	7.348	False
102	1	2012-01-20	Warm	3.268	219.985689	7.348	False
103	1	2012-01-27	Warm	3.290	220.078852	7.348	False
104	1	2012-02-03	Warm	3.360	220.172015	7.348	False
105	1	2012-02-10	Mild	3.409	220.265178	7.348	True

In [377]:
```
features_df.loc[:, ["Store", "IsHoliday"]].head()
```

Out[377]:

	Store	IsHoliday
0	1	False
1	1	True
2	1	False
3	1	False
4	1	False

In [378]:
```
#Retrieve a couple rows from their ROW index values
features_df.iloc[[0, 1]]
```

Out[378]:

	Store	Date	Temp	Fuel_Price	CPI	Unemployment	IsHoliday
0	1	2010-02-05	Mild	2.572	211.096358	8.106	False
1	1	2010-02-12	Mild	2.548	211.242170	8.106	True

In [379]:
```
#Similar to arrays, we can use splicing to access multiple rows
features_df.iloc[:5]
```

Out[379]:

	Store	Date	Temp	Fuel_Price	CPI	Unemployment	IsHoliday
0	1	2010-02-05	Mild	2.572	211.096358	8.106	False
1	1	2010-02-12	Mild	2.548	211.242170	8.106	True
2	1	2010-02-19	Mild	2.514	211.289143	8.106	False
3	1	2010-02-26	Mild	2.561	211.319643	8.106	False
4	1	2010-03-05	Mild	2.625	211.350143	8.106	False

In [380]:
```
#We may also provide specific row/column values to access specific values
features_df.iloc[0, 1]
```

Out[380]: '2010-02-05'

In [381]:
```
#Multiple rows and specific columns
features_df.iloc[[0, 2], [1, 3]]
```

Out[381]:

	Date	Fuel_Price
0	2010-02-05	2.572
2	2010-02-19	2.514

In [382]:
```
#We can also splice multiple rows / columns
features_df.iloc[1:3, 0:3]
```

Out[382]:

	Store	Date	Temp
1	1	2010-02-12	Mild
2	1	2010-02-19	Mild

Formatting Data

- To access and format the string values of a DataFrame, we can access methods within the "str" module of the DataFrame
- We may also format float values using options.display.float_format() in Pandas

In [400]:
```
# Split will return a list of substrings every time it finds a provided string

print("This is Python 2 - Pandas".split())
print("This is Python 2 - Pandas".split('-'))
```

```
['This', 'is', 'Python', '2', '-', 'Pandas']
['This is Python 2 ', ' Pandas']
```

In [401]:
```
features_df.head()
```

Out[401]:

	Store	Date	Temp	Fuel_Price	CPI	Unemployment	IsHoliday	Year	Month
0	1	2010-02-05	Mild	2.572	211.096358	8.106	False	2010	2010
1	1	2010-02-12	Mild	2.548	211.242170	8.106	True	2010	2010
2	1	2010-02-19	Mild	2.514	211.289143	8.106	False	2010	2010
3	1	2010-02-26	Mild	2.561	211.319643	8.106	False	2010	2010
4	1	2010-03-05	Mild	2.625	211.350143	8.106	False	2010	2010

In [383]:
```
#By accessing .str, we gain access to all the string methods we covered in Python
#new data frame with split value columns

new = features_df["Date"].str.split("-", expand = True)

new.head()
```

Out[383]:

	0	1	2
0	2010	02	05
1	2010	02	12
2	2010	02	19
3	2010	02	26
4	2010	03	05

In [384]:
```
#Declare new column named Year to be first column of new DataFrame
features_df["Year"]= new[0]

#Do the same for Month
features_df["Month"]= new[1]
```

In [385]:
```
features_df.head()
```

Out[385]:

	Store	Date	Temp	Fuel_Price	CPI	Unemployment	IsHoliday	Year	Month
0	1	2010-02-05	Mild	2.572	211.096358	8.106	False	2010	02
1	1	2010-02-12	Mild	2.548	211.242170	8.106	True	2010	02
2	1	2010-02-19	Mild	2.514	211.289143	8.106	False	2010	02
3	1	2010-02-26	Mild	2.561	211.319643	8.106	False	2010	02
4	1	2010-03-05	Mild	2.625	211.350143	8.106	False	2010	03

In [386]:
```
#Format float
features_df.round(2).head()
```

Out[386]:

	Store	Date	Temp	Fuel_Price	CPI	Unemployment	IsHoliday	Year	Month
0	1	2010-02-05	Mild	2.57	211.10	8.11	False	2010	02
1	1	2010-02-12	Mild	2.55	211.24	8.11	True	2010	02
2	1	2010-02-19	Mild	2.51	211.29	8.11	False	2010	02
3	1	2010-02-26	Mild	2.56	211.32	8.11	False	2010	02
4	1	2010-03-05	Mild	2.62	211.35	8.11	False	2010	03

Conditional Indexing

- Conditional Operators (>, ==, >=) can be used to return rows based on their values
- Bitwise Operators (|, &) can be used to combine conditonal statements

In [387]:
```
features_df.head()
```

Out[387]:

	Store	Date	Temp	Fuel_Price	CPI	Unemployment	IsHoliday	Year	Month
0	1	2010-02-05	Mild	2.572	211.096358	8.106	False	2010	02
1	1	2010-02-12	Mild	2.548	211.242170	8.106	True	2010	02
2	1	2010-02-19	Mild	2.514	211.289143	8.106	False	2010	02
3	1	2010-02-26	Mild	2.561	211.319643	8.106	False	2010	02
4	1	2010-03-05	Mild	2.625	211.350143	8.106	False	2010	03

In [388]:
```
#Check data types of new columns
features_df.dtypes
```

Out[388]:
```
Store             int64
Date             object
Temp             object
Fuel_Price      float64
CPI             float64
Unemployment    float64
IsHoliday          bool
Year             object
Month            object
dtype: object
```

In [389]:
```
#Convert Year and Month to integers from string
features_df['Year'] = features_df['Year'].astype('int64')
features_df['Month'] = features_df['Year'].astype('int64')
```

In [390]:
```
#Return rows with year value of 2011
year_filt = features_df["Year"] == 2011

feb_df = features_df[filt]
feb_df.head()
```

Out[390]:

	Store	Date	Temp	Fuel_Price	CPI	Unemployment	IsHoliday	Year	Month
48	1	2011-01-07	Mild	2.976	211.404742	7.742	False	2011	2011
49	1	2011-01-14	Mild	2.983	211.457411	7.742	False	2011	2011
50	1	2011-01-21	Mild	3.016	211.827234	7.742	False	2011	2011
51	1	2011-01-28	Mild	3.010	212.197058	7.742	False	2011	2011
52	1	2011-02-04	Mild	2.989	212.566881	7.742	False	2011	2011

In [391]:
```
#Return rows with CPI lower than 130
CPI_filt = features_df["CPI"] < 130
low_CPI = features_df[CPI_filt]
low_CPI.head()
```

Out[391]:

	Store	Date	Temp	Fuel_Price	CPI	Unemployment	IsHoliday	Year	Month
546	4	2010-02-05	Mild	2.598	126.442065	8.623	False	2010	2010
547	4	2010-02-12	Mild	2.573	126.496258	8.623	True	2010	2010
548	4	2010-02-19	Mild	2.540	126.526286	8.623	False	2010	2010
549	4	2010-02-26	Mild	2.590	126.552286	8.623	False	2010	2010
550	4	2010-03-05	Mild	2.654	126.578286	8.623	False	2010	2010

In [392]:
```
#Return rows with year equal to 2010 AND unemployment larger than 8
filt1 = features_df["Year"]  == 2010
filt2 = features_df["Unemployment"]  > 8.00

unemployment_2010 = features_df[ filt1 & filt2 ]
unemployment_2010.head()
```

Out[392]:

	Store	Date	Temp	Fuel_Price	CPI	Unemployment	IsHoliday	Year	Month
0	1	2010-02-05	Mild	2.572	211.096358	8.106	False	2010	2010
1	1	2010-02-12	Mild	2.548	211.242170	8.106	True	2010	2010
2	1	2010-02-19	Mild	2.514	211.289143	8.106	False	2010	2010
3	1	2010-02-26	Mild	2.561	211.319643	8.106	False	2010	2010
4	1	2010-03-05	Mild	2.625	211.350143	8.106	False	2010	2010

In [393]:
```
#Return rows with temp larger than 40 OR Store number equal to 4
filt1 = features_df["Temp"]  == 'Cold'
filt2 = features_df["Store"]  == 4

features_df[filt1 | filt2].head()
```

Out[393]:

	Store	Date	Temp	Fuel_Price	CPI	Unemployment	IsHoliday	Year	Month
546	4	2010-02-05	Mild	2.598	126.442065	8.623	False	2010	2010
547	4	2010-02-12	Mild	2.573	126.496258	8.623	True	2010	2010
548	4	2010-02-19	Mild	2.540	126.526286	8.623	False	2010	2010
549	4	2010-02-26	Mild	2.590	126.552286	8.623	False	2010	2010
550	4	2010-03-05	Mild	2.654	126.578286	8.623	False	2010	2010

In [394]:
```
##CLASS EXERCISE
# find the rows with Fuel_Price larger than 3.00 AND IsHoliday is True
filt1 = features_df["IsHoliday"]  == True
filt2 = features_df["Fuel_Price"]  > 3.00

holiday_Fuel = features_df[filt1 & filt2]
```

In [395]:
```
holiday_Fuel.head()
```

Out[395]:

	Store	Date	Temp	Fuel_Price	CPI	Unemployment	IsHoliday	Year	Month
53	1	2011-02-11	Mild	3.022	212.936705	7.742	True	2011	2011
83	1	2011-09-09	Warm	3.546	215.861056	7.962	True	2011	2011
94	1	2011-11-25	Warm	3.236	218.467621	7.866	True	2011	2011
99	1	2011-12-30	Mild	3.129	219.535990	7.866	True	2011	2011
105	1	2012-02-10	Mild	3.409	220.265178	7.348	True	2012	2012

In [396]:
```
# find the rows with CPI < 200  OR Unemployment < 5
filt1 = features_df["CPI"]  < 200
filt2 = features_df["Unemployment"]  < 5.00

CPI_unemployment = features_df[filt1 | filt2]
```

In [397]:
```
CPI_unemployment.head()
```

Out[397]:

	Store	Date	Temp	Fuel_Price	CPI	Unemployment	IsHoliday	Year	Month
546	4	2010-02-05	Mild	2.598	126.442065	8.623	False	2010	2010
547	4	2010-02-12	Mild	2.573	126.496258	8.623	True	2010	2010
548	4	2010-02-19	Mild	2.540	126.526286	8.623	False	2010	2010
549	4	2010-02-26	Mild	2.590	126.552286	8.623	False	2010	2010
550	4	2010-03-05	Mild	2.654	126.578286	8.623	False	2010	2010

In [398]:

```
#Export the current version of our DataFrame to a .csv file
features_df.to_csv("features_final.csv", index=False, header=True)

#to_excel also an option to export to Excel Spreadsheet
features_df.to_excel("features_final.xlsx", index=False, header=True)
```