Python 3

For this tutorial we'll be using the Iris dataset from sklearn.

In this notebook we will:

1. Import required modules and dataset
2. Define multiple Classification models
3. Fit the data to our models
4. Use our trained models to predict a class label
5. Evaluate our models and chose the best performing model

In [1]:
```
#Import Pandas to your workspace
import numpy as np
import pandas as pd
```

In [2]:
```
#Read the "features.csv" file and store it into a variable
df = pd.read_csv("data/features.csv")
```

In [3]:
```
df.head()
```

Out[3]:

	Store	Date	Temp	Fuel_Price	CPI	Unemployment	IsHoliday	Year	Month
0	1	2/5/2010	42.31	2.572	211.096358	8.106	False	2010	2
1	1	2/12/2010	38.51	2.548	211.242170	8.106	True	2010	2
2	1	2/19/2010	39.93	2.514	211.289143	8.106	False	2010	2
3	1	2/26/2010	46.63	2.561	211.319643	8.106	False	2010	2
4	1	3/5/2010	46.50	2.625	211.350143	8.106	False	2010	3

Index

The index of a DataFrame is used as the "address" for specific data points. As we saw in Python 2, by providing these indexes to .loc, we can access different ranges of data. Both the X and Y axes have an index. For rows, we can use the default integer index, or we can assign a column to act as the index. For columns, the column names are the index.

In [4]:
```
#Read the "features.csv" file and store it into a variable
features = pd.read_csv("data/features.csv", index_col = 'Date')
```

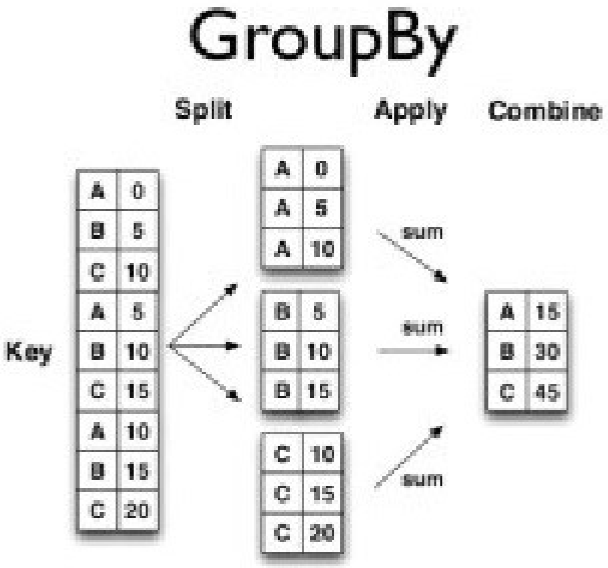
In [5]:
```
#Display the first few rows of the DataFrame
features.head()
```

Out[5]:

	Store	Temp	Fuel_Price	CPI	Unemployment	IsHoliday	Year	Month
Date								
2/5/2010	1	42.31	2.572	211.096358	8.106	False	2010	2
2/12/2010	1	38.51	2.548	211.242170	8.106	True	2010	2
2/19/2010	1	39.93	2.514	211.289143	8.106	False	2010	2
2/26/2010	1	46.63	2.561	211.319643	8.106	False	2010	2
3/5/2010	1	46.50	2.625	211.350143	8.106	False	2010	3

groupby()

- groupby combines 3 steps all in one function:
  1. Split a DataFrame
  2. Apply a function
  3. Combine the results
- groupby must be given the name of the column to group by as a string
- The column to apply the function onto must also be specified, as well as the function to apply



In [6]:

```
#Apply groupby to the Year and Month columns, calculating the mean of
year_CPI = features.groupby("Year")["CPI"].sum().reset_index()
year_CPI.head()
```

Out[6]:

	Year	CPI
0	2010	363099.848068
1	2011	401416.975385

	Year	CPI
2	2012	411176.892813

In [7]:

```
#Groupby returns a DataFrame, so we have access to all the same metho
year_CPI.sort_values(by = "Year", ascending = False, inplace = True)
year_CPI.head()
```

Out[7]:

	Year	CPI
3	2013	135870.737569
2	2012	411176.892813
1	2011	401416.975385
0	2010	363099.848068

In [8]:

```
# Exercise : Define a new variable that measures the average Temp by S
temp_store = features.groupby("Store")["Temp"].mean()
temp_store.head(50)
```

Out[8]:

```
Store
1      66.912033
2      66.728407
3      70.394176
4      61.416648
5      68.224505
6      68.504670
7      37.921264
8      61.180220
9      66.269505
10     71.329121
11     71.217308
```

In [9]:

```
#Exercise: Try out the next few cells on your own to test your underst
#1. Read the "stores.csv" file and store it into a variable called sto
stores = pd.read_csv("data/stores.csv")
```

In [10]:

```
#2. Display the first few rows of the stores DataFrame
stores.head()
```

Out[10]:

	Store	Type	Size
0	1	A : East	151315
1	2	A : East	202307
2	3	B : West	37392
3	4	A : East	205863
4	5	B : West	34875

In [11]:

```
#3. Redefine the Type column to lower case
stores["Type"] = stores["Type"].str.lower()
```

In [12]:

```
#4. Display the first few rows to verify changes
stores.head()
```

Out[12]:

	Store	Type	Size
0	1	a : east	151315
1	2	a : east	202307
2	3	b : west	37392
3	4	a : east	205863
4	5	b : west	34875

In [13]:

```
#5. Rename the 'Size' column to 'Area'
stores.rename(columns={'Size': 'Area'}, inplace=True)
```

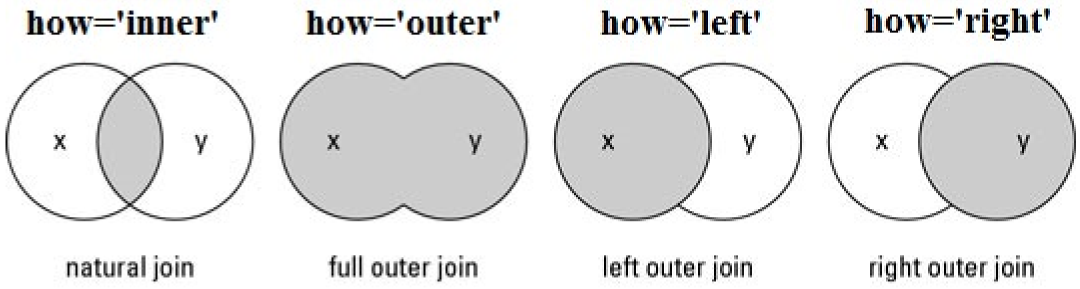
In [14]:
```
stores.head()
```

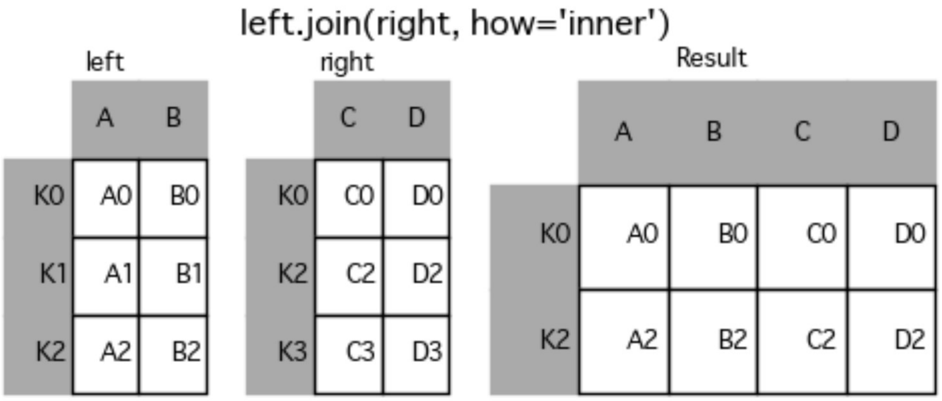
Out[14]:

	Store	Type	Area
0	1	a : east	151315
1	2	a : east	202307
2	3	b : west	37392
3	4	a : east	205863
4	5	b : west	34875

merge()

- Merge two DataFrames along common columns
- Must be provided the DataFrame to merge with, as well as the names of the common columns
- Will merge and map rows where the values in both DataFrames are equal




left.join(right, how='inner')

left

	A	B
K0	A0	B0
K1	A1	B1
K2	A2	B2

right

	C	D
K0	C0	D0
K2	C2	D2
K3	C3	D3

Result

	A	B	C	D
K0	A0	B0	C0	D0
K2	A2	B2	C2	D2

In [15]:
```
features.head()
```

Out[15]:

	Store	Temp	Fuel_Price	CPI	Unemployment	IsHoliday	Year	Month
Date								
2/5/2010	1	42.31	2.572	211.096358	8.106	False	2010	2

	Store	Temp	Fuel_Price	CPI	Unemployment	IsHoliday	Year	Month
Date								
2/12/2010	1	38.51	2.548	211.242170	8.106	True	2010	2
2/19/2010	1	39.93	2.514	211.289143	8.106	False	2010	2

In [16]:

```
stores.head()
```

Out[16]:

	Store	Type	Area
0	1	a : east	151315
1	2	a : east	202307
2	3	b : west	37392
3	4	a : east	205863
4	5	b : west	34875

In [17]:

```
#Merge the stores DataFrame into the features DataFrame on the Stores
df_merged = features.merge(stores, on = "Store")
```

In [18]:

```
#Display a few rows to verify changes
df_merged.head()
```

Out[18]:

	Store	Temp	Fuel_Price	CPI	Unemployment	IsHoliday	Year	Month	Type	Area
0	1	42.31	2.572	211.096358	8.106	False	2010	2	a : east	151315
1	1	38.51	2.548	211.242170	8.106	True	2010	2	a : east	151315
2	1	39.93	2.514	211.289143	8.106	False	2010	2	a : east	151315
3	1	46.63	2.561	211.319643	8.106	False	2010	2	a : east	151315
4	1	46.50	2.625	211.350143	8.106	False	2010	3	a : east	151315

apply()

- Allows us to apply a custom function along an axis of the DataFrame
- Can pull on logic from Python 1 to convert our numerical data to categorical

In [19]:

```
#Define a function to convert float values to our custom categorical i

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

In [20]:

```
#With the apply() function we can apply our custom function to each va

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

In [21]:

```
df_merged['Temp'].tail()
```

Out[21]:

```
8185    Warm
8186    Warm
8187    Warm
8188     Hot
8189    Warm
Name: Temp, dtype: object
```

In [22]:

```
df_merged.head()
```

Out[22]:

	Store	Temp	Fuel_Price	CPI	Unemployment	IsHoliday	Year	Month	Type	Area
0	1	Mild	2.572	211.096358	8.106	False	2010	2	a : east	151315
1	1	Mild	2.548	211.242170	8.106	True	2010	2	a : east	151315
2	1	Mild	2.514	211.289143	8.106	False	2010	2	a : east	151315
3	1	Mild	2.561	211.319643	8.106	False	2010	2	a : east	151315
4	1	Mild	2.625	211.350143	8.106	False	2010	3	a : east	151315

In [23]:

```
#lambda function
df_merged['Type'] = df_merged['Type'].apply(lambda x: x.split()[0])
```

In [24]:

```
df_merged.head()
```

Out[24]:

	Store	Temp	Fuel_Price	CPI	Unemployment	IsHoliday	Year	Month	Type	Area
0	1	Mild	2.572	211.096358	8.106	False	2010	2	a	151315
1	1	Mild	2.548	211.242170	8.106	True	2010	2	a	151315
2	1	Mild	2.514	211.289143	8.106	False	2010	2	a	151315
3	1	Mild	2.561	211.319643	8.106	False	2010	2	a	151315
4	1	Mild	2.625	211.350143	8.106	False	2010	3	a	151315

pivot_table()

- Create a spreadsheet-style pivot table as a DataFrame.
- Different from Groupby in shape of resulting DataFrame. Number of columns based on value passed and not combinations.

In [25]:
```
#Create a Pivot Table to display the fuel prices by store and temperat
fp_pivot = df_merged.pivot_table(values='Fuel_Price', index="Store",
                                 columns='Temp', aggfunc='mean')
```

In [27]:
```
fp_pivot.head()
```

Out[27]:

Temp Store	Hot	Mild	Warm
1	3.192864	3.032655	3.346321
2	3.206500	3.052281	3.348990
3	3.282842	2.963833	3.278708
4	3.384179	3.089566	3.305792
5	3.267300	3.023000	3.305394

In [28]:
```
df_merged.groupby(['Store', 'Temp'])['Fuel_Price'].mean()
```

Out[28]:
```
Store  Temp
1      Hot     3.192864
       Mild    3.032655
       Warm    3.346321
2      Hot     3.206500
       Mild    3.052281
                 ...
44     Mild    3.188244
       Warm    3.454271
45     Hot     3.402667
       Mild    3.438857
       Warm    3.505387
Name: Fuel_Price, Length: 124, dtype: float64
```

In [29]:
```
# Exercise: Create a Pivot table that displays the mean CPI by store t
cpi_pivot = df_merged.pivot_table(values='CPI', index="Type",
                                  columns = 'Year', aggfunc='mean')
```

In [30]:
```
cpi_pivot.head()
```

Out[30]:

Year Type	2010	2011	2012	2013
a	170.927868	174.427272	178.73595	180.679801
b	164.748262	168.113135	172.15070	173.936111

Year	2010	2011	2012	2013

In [ ]:

```
#Export the final version of our DataFrame to a .csv file named "final
```