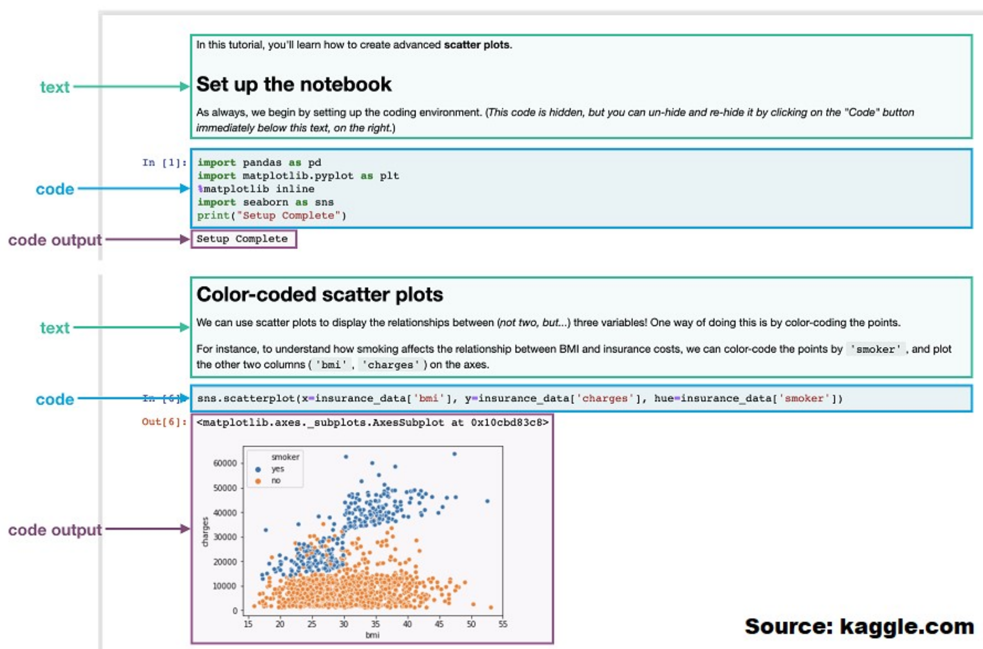
Jupyter Notebook

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



Classes

- Object-orientated programming approach popular and efficient
- Define classes of real-world things or situations (can be thought of as creating your own data type)
    - Attributes of various data types
    - Functions inside of a class are the same except called methods
    - Methods may be accessed using the dot operator
- Instanciate objects of your classes
- __init()__ method used to prefill attributes

- Capitalize class names

In [1]:
```
class Employee():
    """A simple attempt to represent am employee."""
    def __init__(self, name, employee_num, department ):
        self.name = name
        self.employee_num = employee_num
        self.department = department


    def description(self): # Creating a function (a.k.a method) that c
        print(f"{self.name} (employee number: {self.employee_num}) - D
```

In [2]:
```
employee1 = Employee("Mike", 12210, "Marketing")
employee2 = Employee("Peter", 31445, "IT")
employee1.description()
employee2.description()
```

```
Mike  (employee number: 12210) - Dept: Marketing
Peter (employee number: 31445) - Dept: IT
```

In [3]:
```
#Create a Payment class and assign it 3 attributes: payer, payee, amou
class Payment:
    def __init__(self, payer, payee, amount):
        self.payer = payer
        self.payee = payee
        self.amount = amount
```

In [4]:
```
pay1 = Payment("Peter", "Seamus", 100)
```

In [5]:
```
print(pay1.amount)
```

```
100
```

In [6]:
```
print(pay1.payee)
```

```
Seamus
```

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.

Built Around 2 Main Classes:

- DataFrames
- Series

In [7]:
```
#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 [8]:
```
#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 [9]:
```
#Display top 5 rows
features_df.head()

#nan values are essentially empty entries
```

Out[9]:

	Store	Date	Temperature	Fuel_Price	MarkDown1	CPI	Unemployment	IsHolida
0	1	2/5/2010	42.31	2.572	NaN	211.096358	8.106	Fals
1	1	2/12/2010	38.51	2.548	NaN	211.242170	8.106	Tru
2	1	2/19/2010	39.93	2.514	NaN	211.289143	8.106	Fals
3	1	2/26/2010	46.63	2.561	NaN	211.319643	8.106	Fals
4	1	3/5/2010	46.50	2.625	NaN	211.350143	8.106	Fals

In [10]:
```
#Display bottom 5 rows
features_df.tail()
```

Out[10]:

	Store	Date	Temperature	Fuel_Price	MarkDown1	CPI	Unemployment	IsHoliday
8185	45	6/28/2013	76.05	3.639	4842.29	NaN	NaN	False
8186	45	7/5/2013	77.50	3.614	9090.48	NaN	NaN	False
8187	45	7/12/2013	79.37	3.614	3789.94	NaN	NaN	False
8188	45	7/19/2013	82.84	3.737	2961.49	NaN	NaN	False
8189	45	7/26/2013	76.06	3.804	212.02	NaN	NaN	False

In [11]:

```
#Print dimensions of DataFrame as tuple
features_df.shape
```

Out[11]:

```
(8190, 9)
```

In [12]:

```
#Print list of column values
features_df.columns
```

Out[12]:

```
Index(['Store', 'Date', 'Temperature', 'Fuel_Price', 'MarkDown1', 'CP
I',
       'Unemployment', 'IsHoliday', 'Status'],
      dtype='object')
```

In [13]:

```
#To only rename specific columns
features_df.rename(columns={'Temperature': 'Temp', 'MarkDown1':'MD1'},
```

In [14]:

```
#Print Pandas-specific data types of all columns
features_df.dtypes
```

Out[14]:

```
Store             int64
Date             object
Temp            float64
Fuel_Price      float64
MD1             float64
CPI             float64
Unemployment    float64
IsHoliday          bool
Status           object
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 [15]:
```
#Extract CPI column of features_df
features_df["CPI"].head()
```

Out[15]:
```
0    211.096358
1    211.242170
2    211.289143
3    211.319643
4    211.350143
Name: CPI, dtype: float64
```

In [16]:
```
#Display the dimensions with 'shape'
#Display the total number of entries with 'size'
# Example with our DataFrame
print(features_df.shape)
print(features_df.size)
```

```
(8190, 9)
73710
```

In [17]:
```
#Maximum value in Series
features_df["CPI"].max()
```

Out[17]:
```
228.9764563
```

In [18]:
```
#Median value in Series
features_df["CPI"].median()
```

Out[18]:
```
182.7640032
```

In [19]:
```
#Minimum value in Series
features_df["CPI"].min()
```

Out[19]:
```
126.064
```

In [20]:
```
#Basic Statistical Summary of a column
features_df['Temp'].describe()
```

Out[20]:
```
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 [21]:
```
#Print list of unique values
```

```
features_df["Store"].unique()
```

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

In [22]:
```
#Print unique values and frequency
features_df["Date"].value_counts()
```

Out[22]:
```
10/22/2010    45
3/1/2013      45
11/30/2012    45
5/28/2010     45
7/2/2010      45
              ..
9/14/2012     45
12/9/2011     45
1/27/2012     45
6/22/2012     45
4/8/2011      45
Name: Date, Length: 182, dtype: int64
```

In [23]:
```
#Return a sorted DataFrame acording to specified column
features_df.sort_values(by = "Date", ascending = True)
features_df.head()
```

Out[23]:

	Store	Date	Temp	Fuel_Price	MD1	CPI	Unemployment	IsHoliday	Status
0	1	2/5/2010	42.31	2.572	NaN	211.096358	8.106	False	Fullfilled
1	1	2/12/2010	38.51	2.548	NaN	211.242170	8.106	True	Partial
2	1	2/19/2010	39.93	2.514	NaN	211.289143	8.106	False	Pending
3	1	2/26/2010	46.63	2.561	NaN	211.319643	8.106	False	Partial
4	1	3/5/2010	46.50	2.625	NaN	211.350143	8.106	False	Partial

In [24]:
```
features_df.head()
```

Out[24]:

	Store	Date	Temp	Fuel_Price	MD1	CPI	Unemployment	IsHoliday	Status
0	1	2/5/2010	42.31	2.572	NaN	211.096358	8.106	False	Fullfilled
1	1	2/12/2010	38.51	2.548	NaN	211.242170	8.106	True	Partial
2	1	2/19/2010	39.93	2.514	NaN	211.289143	8.106	False	Pending
3	1	2/26/2010	46.63	2.561	NaN	211.319643	8.106	False	Partial
4	1	3/5/2010	46.50	2.625	NaN	211.350143	8.106	False	Partial

In [25]:
```
# delete one column
features_df.drop(columns = "MD1").tail()
```

Out[25]:

	Store	Date	Temp	Fuel_Price	CPI	Unemployment	IsHoliday	Status
8185	45	6/28/2013	76.05	3.639	NaN	NaN	False	Partial
8186	45	7/5/2013	77.50	3.614	NaN	NaN	False	Partial
8187	45	7/12/2013	79.37	3.614	NaN	NaN	False	Partial
8188	45	7/19/2013	82.84	3.737	NaN	NaN	False	Fullfilled
8189	45	7/26/2013	76.06	3.804	NaN	NaN	False	Fullfilled

In [26]:
```
# Check for missing values and how many
features_df.isnull().sum()
```

Out[26]:
```
Store              0
Date               0
Temp               0
Fuel_Price         0
MD1             4158
CPI              585
Unemployment     585
IsHoliday          0
Status             0
dtype: int64
```

In [27]:
```
# delete multiple columns
features_df.drop(columns = 'MD1', inplace = True)
```

In [28]:
```
features_df.head()
```

Out[28]:

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

In [29]:
```
#Applying basic operations to columns
#Uses matrix manipulation instead of row by row increments
features_df['Unemployment'] += 1
```

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

Out[30]:

	Store	Date	Temp	Fuel_Price	CPI	Unemployment	IsHoliday	Status
0	1	2/5/2010	42.31	2.572	211.096358	9.106	False	Fullfilled
1	1	2/12/2010	38.51	2.548	211.242170	9.106	True	Partial
2	1	2/19/2010	39.93	2.514	211.289143	9.106	False	Pending
3	1	2/26/2010	46.63	2.561	211.319643	9.106	False	Partial
4	1	3/5/2010	46.50	2.625	211.350143	9.106	False	Partial

In [31]:

```
#Say a colleague of yours asks for a new metric called "customerCost"
#Add a column that is equal to Fuel_Price * CPI
```

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 [32]:

```
#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[32]:

	Fuel_Price	CPI	Unemployment	IsHoliday
0	2.572	211.096358	9.106	False
1	2.548	211.242170	9.106	True
2	2.514	211.289143	9.106	False
3	2.561	211.319643	9.106	False
4	2.625	211.350143	9.106	False
5	2.667	211.380643	9.106	False
6	2.720	211.215635	9.106	False
7	2.732	211.018042	9.106	False
8	2.719	210.820450	8.808	False
9	2.770	210.622857	8.808	False
10	2.808	210.488700	8.808	False

In [33]:

```
features_df.loc[[100,105]]
```

Out[33]:

	Store	Date	Temp	Fuel_Price	CPI	Unemployment	IsHoliday	Status	custor
100	1	1/6/2012	49.01	3.157	219.714258	8.348	False	Partial	693
105	1	2/10/2012	48.02	3.409	220.265178	8.348	True	Partial	750

In [34]:

```
#Retrieve the CPI and customerCost of rows 500 to 505
features_df.loc[500:505, ["CPI", "customerCost"]]
```

Out[34]:

	CPI	customerCost
500	226.112207	840.459072
501	226.315150	842.118672
502	226.518093	830.415327
503	226.721036	820.049986
504	226.923979	817.153247
505	226.968844	815.726026

In [35]:

```
#We can also retrieve rows with a condition
features_df.loc[features_df['Store'] == 2]
```

Out[35]:

	Store	Date	Temp	Fuel_Price	CPI	Unemployment	IsHoliday	Status	custo
182	2	2/5/2010	40.19	2.572	210.752605	9.324	False	Fullfilled	54
183	2	2/12/2010	38.49	2.548	210.897994	9.324	True	Fullfilled	53
184	2	2/19/2010	39.69	2.514	210.945160	9.324	False	Partial	53
185	2	2/26/2010	46.10	2.561	210.975957	9.324	False	Fullfilled	54
186	2	3/5/2010	47.17	2.625	211.006754	9.324	False	Pending	55
...	...	...	...	...	...	...	...	...	
359	2	6/28/2013	85.37	3.495	NaN	NaN	False	Pending	
360	2	7/5/2013	79.48	3.422	NaN	NaN	False	Fullfilled	
361	2	7/12/2013	85.41	3.400	NaN	NaN	False	Partial	
362	2	7/19/2013	79.16	3.556	NaN	NaN	False	Fullfilled	
363	2	7/26/2013	83.17	3.620	NaN	NaN	False	Partial	

182 rows × 9 columns

In [36]:

```
#We can layer conditions with &
filt1 = features_df['Store'] == 2
filt2 = features_df['CPI'] > 211
features_df.loc[filt1 & filt2]
```

Out[36]:

	Store	Date	Temp	Fuel_Price	CPI	Unemployment	IsHoliday	Status	custo
186	2	3/5/2010	47.17	2.625	211.006754	9.324	False	Pending	55
187	2	3/12/2010	57.56	2.667	211.037551	9.324	False	Pending	56
200	2	6/11/2010	83.40	2.668	211.112002	9.200	False	Pending	56
201	2	6/18/2010	85.81	2.637	211.109654	9.200	False	Fullfilled	55
207	2	7/30/2010	83.49	2.640	211.026468	9.099	False	Fullfilled	55
...	...	...	...	...	...	...	...	...	
346	2	3/29/2013	50.54	3.606	224.635985	7.237	False	Partial	81
347	2	4/5/2013	58.30	3.583	224.719258	7.112	False	Pending	80
348	2	4/12/2013	61.23	3.529	224.802531	7.112	False	Fullfilled	79
349	2	4/19/2013	67.05	3.451	224.802531	7.112	False	Fullfilled	77
350	2	4/26/2013	58.13	3.417	224.802531	7.112	False	Fullfilled	76

148 rows × 9 columns

In [37]:

```
#Retrieve all rows with a isHoliday of True and customerCost larger th
filt1 = features_df['IsHoliday'] == True
filt2 = features_df['customerCost'] > 550
features_df.loc[filt1 & filt2]
```

Out[37]:

	Store	Date	Temp	Fuel_Price	CPI	Unemployment	IsHoliday	Status	cu
42	1	11/26/2010	64.52	2.735	211.748433	8.838	True	Partial	
47	1	12/31/2010	48.43	2.943	211.404932	8.838	True	Pending	
53	1	2/11/2011	36.39	3.022	212.936705	8.742	True	Fullfilled	
83	1	9/9/2011	76.00	3.546	215.861056	8.962	True	Pending	
94	1	11/25/2011	60.14	3.236	218.467621	8.866	True	Partial	
...	...	...	...	...	...	...	...	...	
8113	45	2/10/2012	37.00	3.640	189.707605	9.424	True	Partial	
8143	45	9/7/2012	75.70	3.911	191.577676	9.684	True	Pending	
8154	45	11/23/2012	43.08	3.748	192.283032	9.667	True	Partial	
8159	45	12/28/2012	35.96	3.563	192.559264	9.667	True	Pending	
8165	45	2/8/2013	28.99	3.753	192.897089	9.625	True	Fullfilled	

265 rows × 9 columns

In [38]:

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

Out[38]:

	Store	Date	Temp	Fuel_Price	CPI	Unemployment	IsHoliday	Status	custom
0	1	2/5/2010	42.31	2.572	211.096358	9.106	False	Fullfilled	542.!

	Store	Date	Temp	Fuel_Price	CPI	Unemployment	IsHoliday	Status	custom

In [39]:
```
#We may also provide specific row/column values to access specific val
features_df.iloc[0, 1]
```

Out[39]: '2/5/2010'

In [40]:
```
#Multiple rows and specific columns
features_df.iloc[[0, 2], [1, 3]]
```

Out[40]:

	Date	Fuel_Price
0	2/5/2010	2.572
2	2/19/2010	2.514

In [41]:
```
#Access rows 1 to 3 for Store column to Fuel_Price
features_df.iloc[1:3, 0:3]
```

Out[41]:

	Store	Date	Temp
1	1	2/12/2010	38.51
2	1	2/19/2010	39.93

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 [42]:
```
# We can access all the same string methods from Python 1 using .str
features_df['Status'] = features_df['Status'].str.upper()
```

In [43]:
```
features_df.head()
```

Out[43]:

	Store	Date	Temp	Fuel_Price	CPI	Unemployment	IsHoliday	Status	cu:
0	1	2/5/2010	42.31	2.572	211.096358	9.106	False	FULLFILLED	
1	1	2/12/2010	38.51	2.548	211.242170	9.106	True	PARTIAL	
2	1	2/19/2010	39.93	2.514	211.289143	9.106	False	PENDING	
3	1	2/26/2010	46.63	2.561	211.319643	9.106	False	PARTIAL	
4	1	3/5/2010	46.50	2.625	211.350143	9.106	False	PARTIAL	

In [44]:

```
#Format float
features_df.round(2).head()
```

Out[44]:

	Store	Date	Temp	Fuel_Price	CPI	Unemployment	IsHoliday	Status	custom
0	1	2/5/2010	42.31	2.57	211.10	9.11	False	FULLFILLED	
1	1	2/12/2010	38.51	2.55	211.24	9.11	True	PARTIAL	
2	1	2/19/2010	39.93	2.51	211.29	9.11	False	PENDING	
3	1	2/26/2010	46.63	2.56	211.32	9.11	False	PARTIAL	
4	1	3/5/2010	46.50	2.62	211.35	9.11	False	PARTIAL	

In [45]:

```
#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)
```