Python 2 - Object Oriented Programming and Pandas

4 Pillars of OOP

- Encapsulation: Group related variables and functions together to reduce complexity and increase reusability
- Data Abstraction: Creating methods to interface with attributes of your class. Show only essentials to reduce complexity
- Inheritance
- Polymorphism

Inheritance

- New classes do not need to be declared from scratch. They may build on existing classes
- When one class inherits from another, it automatically takes on all the attributes and methods of the first class
- Goal: Eliminate redundant code by inheriting attributes and methods from a parent class

In [274]:
```
class Employee():
    """A simple attempt to represent am employee."""
    def __init__(self, employee_num, department, name):
        self.employee_num = employee_num
        self.department = department
        self.name = name
        self.days_worked = 0

    def get_descriptive_name(self):
        #Method to display basic information of employee
        long_name = f"{self.name} ({self.employee_num}) of {self.department} dept."
        return long_name.title()

    def num_days(self):
        #Method to display days_work attribute
        print(f"{self.name} has worked {self.days_worked} days")

    def increment_days(self):
        #Method to increment days_work attribute
        self.days_worked += 1
        print("days increased")
```

In [275]:
```
class Engineer(Employee):
    """Represent aspects of an employee, specific to engineers."""
    def __init__(self, employee_num, department, name, p_eng):
        """Initialize attributes of the parent emplyee class, adding new p_eng attrib
        super().__init__(employee_num, department, name)
        self.p_eng = p_eng
```

In [276]:
```
#Create instance of Engineer class to verify it inherited all the attributes and meth
new_hire = Engineer(1213, "Machine Learning", "Peter Ling", False)
print(new_hire.get_descriptive_name())
new_hire.num_days()
new_hire.increment_days()
```

```
Peter Ling (1213) Of Machine Learning Dept.
Peter Ling has worked 0 days
days increased
```

In [277]:
```
new_hire.num_days()
```

```
Peter Ling has worked 1 days
```

Polymorphism

- Because child classes inherit all attributes and methods from their parent class, we may wish to refactor and customize classes to specific use cases.
- Overiding involves the redefining of methods to better suit child classes

In [278]:
```
class Recruiter(Employee):
    """Represent aspects of an employee, specific to recruiters"""
    def __init__(self, employee_num, department, name):
        """Initialize attributes of the parent class, while adding hires attibute of
        super().__init__(employee_num, department, name)
        self.hires = []

    def get_descriptive_name(self):
        #Override get_descriptive_name method of parent class to better suit our Recr
        long_name = f"{self.name} ({self.employee_num}) hires for {self.department} d
        return long_name.title()

    def add_hire(self, emp_id):
        #Add new method add_hire to add to our new hires attribute
        self.hires.append(emp_id)
        print(self.hires)
```

In [279]:
```
'''Create new instance of Recruiter class to ensure it inherited from parent class Em
while modifying and adding our specified methods'''

new_recruiter = Recruiter(1211, "Finance", "Robert Goss")
print(new_recruiter.get_descriptive_name())
print(new_recruiter.hires)
new_recruiter.add_hire(1423)
print(new_recruiter.hires)
```

```
Robert Goss (1211) Hires For Finance Dept.
[]
[1423]
[1423]
```

Pandas

In [280]:
```
#Import pandas and assign it to a shorthand name pd
import pandas as pd
import numpy as np

%matplotlib inline
```

Reading CVS 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 [281]:
```
#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 [282]:
```
#Display top 5 rows
features_df.head()

#nan values are essentially empty entries
```

Out[282]:

	Store	Date	Temperature	Fuel_Price	MarkDown1	MarkDown2	MarkDown3	MarkDown4	MarkDown5	
0	1	2010-02-05	42.31	2.57	nan	nan	nan	nan	nan	2
1	1	2010-02-12	38.51	2.55	nan	nan	nan	nan	nan	2
2	1	2010-02-19	39.93	2.51	nan	nan	nan	nan	nan	2
3	1	2010-02-26	46.63	2.56	nan	nan	nan	nan	nan	2
4	1	2010-03-05	46.50	2.62	nan	nan	nan	nan	nan	2

In [283]:
```
#Display bottom 5 rows
features_df.tail()
```

Out[283]:

	Store	Date	Temperature	Fuel_Price	MarkDown1	MarkDown2	MarkDown3	MarkDown4	MarkDown5
8185	45	2013-06-28	76.05	3.64	4842.29	975.03	3.00	2449.97	3169.69
8186	45	2013-07-05	77.50	3.61	9090.48	2268.58	582.74	5797.47	1514.93
8187	45	2013-07-12	79.37	3.61	3789.94	1827.31	85.72	744.84	2150.36
8188	45	2013-07-19	82.84	3.74	2961.49	1047.07	204.19	363.00	1059.46
8189	45	2013-07-26	76.06	3.80	212.02	851.73	2.06	10.88	1864.57

In [284]:
```
#Print dimensions of DataFrame as tuple
features_df.shape
```

Out[284]:
```
(8190, 12)
```

In [285]:
```
#Print list of column values
features_df.columns
```

Out[285]:
```
Index(['Store', 'Date', 'Temperature', 'Fuel_Price', 'MarkDown1', 'MarkDown2',
       'MarkDown3', 'MarkDown4', 'MarkDown5', 'CPI', 'Unemployment',
       'IsHoliday'],
      dtype='object')
```

In [286]:
```
#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', 'IsHoliday']

features_df.head()
```

Out[286]:

	Store	Date	Temperature	Fuel_Price	MD1	MD2	MD3	MD4	MD5	CPI	Unemployment	IsHoliday
0	1	2010-02-05	42.31	2.57	nan	nan	nan	nan	nan	211.10	8.11	False
1	1	2010-02-12	38.51	2.55	nan	nan	nan	nan	nan	211.24	8.11	True
2	1	2010-02-19	39.93	2.51	nan	nan	nan	nan	nan	211.29	8.11	False
3	1	2010-02-26	46.63	2.56	nan	nan	nan	nan	nan	211.32	8.11	False
4	1	2010-03-05	46.50	2.62	nan	nan	nan	nan	nan	211.35	8.11	False

In [287]:
```
#To only rename specific columns
features_df.rename(columns={'Temperature': 'Temp'}, inplace=True)
```

In [288]:
```
#Print Pandas-specific data types of all columns
features_df.dtypes
```

Out[288]:
```
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 [289]:
```
#Extract CPI column of features_df
features_df["CPI"]
```

Out[289]:
```
0       211.10
1       211.24
2       211.29
3       211.32
4       211.35
         ...
8185       nan
8186       nan
8187       nan
8188       nan
8189       nan
Name: CPI, Length: 8190, dtype: float64
```

In [290]:
```
#Replace NaN (empty) values with 0's
features_df.fillna(0)
```

Out[290]:

	Store	Date	Temp	Fuel_Price	MD1	MD2	MD3	MD4	MD5	CPI	Unemployment	Is
0	1	2010-02-05	42.31	2.57	0.00	0.00	0.00	0.00	0.00	211.10	8.11	
1	1	2010-02-12	38.51	2.55	0.00	0.00	0.00	0.00	0.00	211.24	8.11	
2	1	2010-02-19	39.93	2.51	0.00	0.00	0.00	0.00	0.00	211.29	8.11	
3	1	2010-02-26	46.63	2.56	0.00	0.00	0.00	0.00	0.00	211.32	8.11	
4	1	2010-03-05	46.50	2.62	0.00	0.00	0.00	0.00	0.00	211.35	8.11	
...	...	...	...	...	...	...	...	...	...	...	...	...
8185	45	2013-06-28	76.05	3.64	4842.29	975.03	3.00	2449.97	3169.69	0.00	0.00	
8186	45	2013-07-05	77.50	3.61	9090.48	2268.58	582.74	5797.47	1514.93	0.00	0.00	
8187	45	2013-07-12	79.37	3.61	3789.94	1827.31	85.72	744.84	2150.36	0.00	0.00	
8188	45	2013-07-19	82.84	3.74	2961.49	1047.07	204.19	363.00	1059.46	0.00	0.00	
8189	45	2013-07-26	76.06	3.80	212.02	851.73	2.06	10.88	1864.57	0.00	0.00	

8190 rows × 12 columns

In [291]:
```
#Maximum value in Series
features_df["CPI"].max()
```

Out[291]:
```
228.9764563
```

In [292]:
```
#Median value in Series
features_df["CPI"].median()
```

Out[292]:
```
182.7640032
```

In [293]:
```
#Minimum value in Series
features_df["CPI"].min()
```

Out[293]:
```
126.064
```

In [294]:
```
#Print list of unique values
features_df["Store"].unique()
```

Out[294]:
```
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 [295]:
```
#Print unique values and frequency
features_df["Date"].value_counts()
```

Out[295]:
```
2012-02-10    45
2011-11-04    45
2011-05-06    45
2011-04-01    45
2013-02-22    45
              ..
2012-01-13    45
2013-01-11    45
2011-10-21    45
2013-05-03    45
2011-10-14    45
Name: Date, Length: 182, dtype: int64
```

In [296]:
```
#Return a sorted DataFrame acording to specified column
features_df.sort_values(by = "Date", ascending = True)
features_df.head()
```

Out[296]:

	Store	Date	Temp	Fuel_Price	MD1	MD2	MD3	MD4	MD5	CPI	Unemployment	IsHoliday
0	1	2010-02-05	42.31	2.57	nan	nan	nan	nan	nan	211.10	8.11	False
1	1	2010-02-12	38.51	2.55	nan	nan	nan	nan	nan	211.24	8.11	True
2	1	2010-02-19	39.93	2.51	nan	nan	nan	nan	nan	211.29	8.11	False
3	1	2010-02-26	46.63	2.56	nan	nan	nan	nan	nan	211.32	8.11	False
4	1	2010-03-05	46.50	2.62	nan	nan	nan	nan	nan	211.35	8.11	False

In [297]:
```
# delete one column
features_df.drop(columns = "MD1").head()
```

Out[297]:

	Store	Date	Temp	Fuel_Price	MD2	MD3	MD4	MD5	CPI	Unemployment	IsHoliday
0	1	2010-02-05	42.31	2.57	nan	nan	nan	nan	211.10	8.11	False
1	1	2010-02-12	38.51	2.55	nan	nan	nan	nan	211.24	8.11	True
2	1	2010-02-19	39.93	2.51	nan	nan	nan	nan	211.29	8.11	False
3	1	2010-02-26	46.63	2.56	nan	nan	nan	nan	211.32	8.11	False
4	1	2010-03-05	46.50	2.62	nan	nan	nan	nan	211.35	8.11	False

In [298]:
```
# delete multiple columns
features_df.drop(columns = ['MD1', 'MD2', 'MD3', 'MD4', 'MD5'], inplace = True)
```

In [299]:
```
features_df.head()
```

Out[299]:

	Store	Date	Temp	Fuel_Price	CPI	Unemployment	IsHoliday
0	1	2010-02-05	42.31	2.57	211.10	8.11	False
1	1	2010-02-12	38.51	2.55	211.24	8.11	True
2	1	2010-02-19	39.93	2.51	211.29	8.11	False
3	1	2010-02-26	46.63	2.56	211.32	8.11	False
4	1	2010-03-05	46.50	2.62	211.35	8.11	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 [300]:
```
#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[300]:

	Fuel_Price	CPI	Unemployment	IsHoliday
0	2.57	211.10	8.11	False
1	2.55	211.24	8.11	True
2	2.51	211.29	8.11	False
3	2.56	211.32	8.11	False
4	2.62	211.35	8.11	False
5	2.67	211.38	8.11	False
6	2.72	211.22	8.11	False
7	2.73	211.02	8.11	False
8	2.72	210.82	7.81	False
9	2.77	210.62	7.81	False
10	2.81	210.49	7.81	False

In [301]:
```
#Retrieve a couple rows from their ROW index values
features_df.iloc[[0, 1]]
```

Out[301]:

	Store	Date	Temp	Fuel_Price	CPI	Unemployment	IsHoliday
0	1	2010-02-05	42.31	2.57	211.10	8.11	False
1	1	2010-02-12	38.51	2.55	211.24	8.11	True

In [302]:
```
#Similar to arrays, we can use splicing to access multiple rows
features_df.iloc[:5]
```

Out[302]:

	Store	Date	Temp	Fuel_Price	CPI	Unemployment	IsHoliday
0	1	2010-02-05	42.31	2.57	211.10	8.11	False
1	1	2010-02-12	38.51	2.55	211.24	8.11	True
2	1	2010-02-19	39.93	2.51	211.29	8.11	False
3	1	2010-02-26	46.63	2.56	211.32	8.11	False
4	1	2010-03-05	46.50	2.62	211.35	8.11	False

In [303]:
```
#We may also provide specific row/column values to access specific values
features_df.iloc[0, 1]
```

Out[303]: '2010-02-05'

In [304]:
```
#Multiple rows and specific columns
features_df.iloc[[0, 2], [1, 3]]
```

Out[304]:

	Date	Fuel_Price
0	2010-02-05	2.57
2	2010-02-19	2.51

In [305]:
```
#We can also splice multiple rows / columns
features_df.iloc[1:3, 0:3]
```

Out[305]:

	Store	Date	Temp
1	1	2010-02-12	38.51
2	1	2010-02-19	39.93

In [306]:
```
#How to iterate over rows
for index, row in features_df.iterrows():
    print(f'CPI of :{row["CPI"]} at Store: {row["Store"]}')
```

```
CPI of :211.0963582 at Store: 1
CPI of :211.24216980000003 at Store: 1
CPI of :211.2891429 at Store: 1
CPI of :211.3196429 at Store: 1
CPI of :211.3501429 at Store: 1
CPI of :211.3806429 at Store: 1
CPI of :211.21563500000002 at Store: 1
CPI of :211.0180424 at Store: 1
CPI of :210.82044989999997 at Store: 1
CPI of :210.62285740000002 at Store: 1
CPI of :210.4887 at Store: 1
CPI of :210.4391228 at Store: 1
CPI of :210.3895456 at Store: 1
CPI of :210.33996839999998 at Store: 1
CPI of :210.3374261 at Store: 1
CPI of :210.6170934 at Store: 1
CPI of :210.89676060000002 at Store: 1
CPI of :211.1764278 at Store: 1
CPI of :211.4560951 at Store: 1
```

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 [307]:
```
#By accessing .str, we gain access to all the string methods we covered in Python 1!
#4.
# new data frame with split value columns

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

new.head()
```

Out[307]:

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

In [308]:
```
#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 [309]:
```
features_df.head()
```

Out[309]:

	Store	Date	Temp	Fuel_Price	CPI	Unemployment	IsHoliday	Year	Month
0	1	2010-02-05	42.31	2.57	211.10	8.11	False	2010	02
1	1	2010-02-12	38.51	2.55	211.24	8.11	True	2010	02
2	1	2010-02-19	39.93	2.51	211.29	8.11	False	2010	02
3	1	2010-02-26	46.63	2.56	211.32	8.11	False	2010	02
4	1	2010-03-05	46.50	2.62	211.35	8.11	False	2010	03

In [310]:
```
#Format float
pd.options.display.float_format = "{:.2f}".format
features_df.head()
```

Out[310]:

	Store	Date	Temp	Fuel_Price	CPI	Unemployment	IsHoliday	Year	Month
0	1	2010-02-05	42.31	2.57	211.10	8.11	False	2010	02
1	1	2010-02-12	38.51	2.55	211.24	8.11	True	2010	02
2	1	2010-02-19	39.93	2.51	211.29	8.11	False	2010	02
3	1	2010-02-26	46.63	2.56	211.32	8.11	False	2010	02
4	1	2010-03-05	46.50	2.62	211.35	8.11	False	2010	03

In [311]:
```
#Export the current version of our DataFrame to a .csv file
features_df.to_csv("features.csv")

#to_excel also an option to export to Excel Spreadsheet
```

Conditional Indexing

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

In [312]:
```
features_df.head()
```

Out[312]:

	Store	Date	Temp	Fuel_Price	CPI	Unemployment	IsHoliday	Year	Month
0	1	2010-02-05	42.31	2.57	211.10	8.11	False	2010	02
1	1	2010-02-12	38.51	2.55	211.24	8.11	True	2010	02
2	1	2010-02-19	39.93	2.51	211.29	8.11	False	2010	02
3	1	2010-02-26	46.63	2.56	211.32	8.11	False	2010	02
4	1	2010-03-05	46.50	2.62	211.35	8.11	False	2010	03

In [313]:
```
feb_df = features_df[features_df["Year"] == "2011"]
feb_df.head()
```

Out[313]:

	Store	Date	Temp	Fuel_Price	CPI	Unemployment	IsHoliday	Year	Month
48	1	2011-01-07	48.27	2.98	211.40	7.74	False	2011	01
49	1	2011-01-14	35.40	2.98	211.46	7.74	False	2011	01
50	1	2011-01-21	44.04	3.02	211.83	7.74	False	2011	01
51	1	2011-01-28	43.83	3.01	212.20	7.74	False	2011	01
52	1	2011-02-04	42.27	2.99	212.57	7.74	False	2011	02

In [314]:
```
#Return rows with CPI lower than 130
low_CPI = features_df[features_df["CPI"] < 130]
low_CPI.head()
```

Out[314]:

	Store	Date	Temp	Fuel_Price	CPI	Unemployment	IsHoliday	Year	Month
546	4	2010-02-05	43.76	2.60	126.44	8.62	False	2010	02
547	4	2010-02-12	28.84	2.57	126.50	8.62	True	2010	02
548	4	2010-02-19	36.45	2.54	126.53	8.62	False	2010	02
549	4	2010-02-26	41.36	2.59	126.55	8.62	False	2010	02
550	4	2010-03-05	43.49	2.65	126.58	8.62	False	2010	03

In [315]:
```
#Return rows with year equal to 2010 AND unemployment larger than 8
unemployment_2010 = features_df[(features_df["Year"]  == "2010") & (features_df["Unem
unemployment_2010.head()
```

Out[315]:

	Store	Date	Temp	Fuel_Price	CPI	Unemployment	IsHoliday	Year	Month
0	1	2010-02-05	42.31	2.57	211.10	8.11	False	2010	02
1	1	2010-02-12	38.51	2.55	211.24	8.11	True	2010	02
2	1	2010-02-19	39.93	2.51	211.29	8.11	False	2010	02
3	1	2010-02-26	46.63	2.56	211.32	8.11	False	2010	02
4	1	2010-03-05	46.50	2.62	211.35	8.11	False	2010	03

In [316]:
```
#Return rows with temp larger than 40 OR Store number equal to 4
features_df[(features_df["Temp"]  > 40) | (features_df["Store"]  == 4)].head()
```

Out[316]:

	Store	Date	Temp	Fuel_Price	CPI	Unemployment	IsHoliday	Year	Month
0	1	2010-02-05	42.31	2.57	211.10	8.11	False	2010	02
3	1	2010-02-26	46.63	2.56	211.32	8.11	False	2010	02
4	1	2010-03-05	46.50	2.62	211.35	8.11	False	2010	03
5	1	2010-03-12	57.79	2.67	211.38	8.11	False	2010	03
6	1	2010-03-19	54.58	2.72	211.22	8.11	False	2010	03

In [317]:
```
##CLASS EXERCISE
# find the rows with Fuel_Price larger than 3.00 AND IsHoliday
holiday_Fuel = features_df[(features_df["IsHoliday"]  == True) & (features_df["Fuel_P
```

In [318]:
```
holiday_Fuel.head()
```

Out[318]:

	Store	Date	Temp	Fuel_Price	CPI	Unemployment	IsHoliday	Year	Month
53	1	2011-02-11	36.39	3.02	212.94	7.74	True	2011	02
83	1	2011-09-09	76.00	3.55	215.86	7.96	True	2011	09
94	1	2011-11-25	60.14	3.24	218.47	7.87	True	2011	11
99	1	2011-12-30	44.55	3.13	219.54	7.87	True	2011	12
105	1	2012-02-10	48.02	3.41	220.27	7.35	True	2012	02

In [319]:
```
# find the rows with CPI < 200  OR Unemployment < 5
CPI_unemployment = features_df[(features_df["CPI"]  < 200) | (features_df["Unemployme
```

In [320]:
```
CPI_unemployment.head()
```

Out[320]:

	Store	Date	Temp	Fuel_Price	CPI	Unemployment	IsHoliday	Year	Month
546	4	2010-02-05	43.76	2.60	126.44	8.62	False	2010	02
547	4	2010-02-12	28.84	2.57	126.50	8.62	True	2010	02
548	4	2010-02-19	36.45	2.54	126.53	8.62	False	2010	02
549	4	2010-02-26	41.36	2.59	126.55	8.62	False	2010	02
550	4	2010-03-05	43.49	2.65	126.58	8.62	False	2010	03

In [ ]:
```
```