Financial Modeling with Python

Part 2 – Staff Salary Calculation and Charts

In this section, we are going to learn how to use advance feature of **Pandas** package to calculate staff salaries and use **Matplotlib** package to create different types of charts.

By finished this section, you will be able to:

- Retrieve the content of the CSV files and create data frames.
- Perform calculation with DataFrame.
- Create summaries using Pivot Table.
- Plot graphs.

Downloading the Data Files and Creating a Code File

Download the files **door_access_records.csv** and **hourly_rates.csv** from the BU eLearning web site and save them at your Spyder working directory.

Importing Required Library

Add the following lines to import the libraries:

```
import pandas as pd
import numpy as np
```

Creating a Data Frame with the CSV File Content

Firstly, we have to load the data from the data file and create a data frame.

1. Load the door access records from the file named **door_access_records.csv** and create a data frame named **records**. The **pd.read_csv()** function accepts a file name and returns a data frame. Add the following line to create a data frame.

   ```
   records = pd.read_csv('door_access_records.csv')
   ```

   The data is stored in the **records** data frame as follows:

	Date	**Staff Name**	**Department**	**Working Hours (In & Out)**
0	6/15/2018	Jacky Lee	IT	6:12-20:44
1	6/15/2018	Joyce Ho	Human Resources	8:26-19:51
2	6/15/2018	Ivy Chu	Sales	8:29-18:44
3	6/15/2018	David Chan	Purchasing	8:33-19:50
4	6/15/2018	Gary Yuen	Logistics	8:34-19:45
…	…	…	…	…

2. Load the hourly salary rates from another file named **hourly_rates.csv** and create a data frame named **hourly_rates**.

   ```
   hourly_rate = pd.read_csv('hourly_rate.csv', index_col = 'Staff Name')
   ```

   With **the index_col** parameter, we set the column 'Staff Name' as an index. The data is stored in the **hourly_rates** data frame as follows:

	Salary Rate (per hour)
Staff Name	
Jacky Lee	60
Joyce Ho	70
Ivy Chu	55
David Chan	65
Gary Yuen	75
…	…

Computing the Working Hours

In the records data frame, the "Working Hours (In & Out)" column does not really store the working hours. It stores the IN time and OUT time of the staffs on individual days. The IN time and OUT time are stored in the same column with a minus sign as a separator. Now, we split the values and use them to compute the working hours. The following formula are used to compute the working hours:

$$Working\ Hours = \frac{Minute(OUT\ time) - Minute(IN\ time)}{60}$$

3. Split the "Working Hours (In & Out)" column by using the **str.split()** function. We temporarily store the values in two sets of Series – **in_time** and **out_time**.

   ```
   in_time, out_time = records['Working Hours (In & Out)'].str.split('-').str
   ```

   The **str.split()** function accepts a string for splitting the values. In this example, we use a minus sign. And, we use **.str** to convert the results to string format. Now, **in_time** and **out_time** store time strings respectively.

4. Store the string versions of **IN** time and **OUT** time in the records data frame.

   ```
   records['In'] = in_time
   records['Out'] = out_time
   ```

   The records data frame has two more columns now.

5. Convert the values stored in **in_time** and **out_time** to the **datetime** format, so as to compute the time differences.

   ```
   in_time = pd.to_datetime(in_time)
   out_time = pd.to_datetime(out_time)
   ```

The **to_datetime()** function returns a datetime object of the time string.

6. Then, we subtract the OUT time by the IN time to get the working hours. The results of the subtraction are in **timedelta** object format.

7. Use the **astype()** function with the option **timedelta64[m]** to retrieve the <u>number of minutes</u> representing in float format.

```
working_hours = out_time - in_time
working_hours = working_hours.astype('timedelta64[m]')
```

8. Convert the working hours from minutes to hours and round the results down to two decimal places.
   Add the working hours to the records data frame as a new column named "Working Hours".

```
working_hours = round(working_hours/60, 2)
records['Working Hours'] = working_hours
```

9. Delete the "Working Hours (In & Out)" column because it is no longer necessary.

```
records.drop('Working Hours (In & Out)', axis = 1, inplace = True)
```

We now have the records data frame as follows:

	Date	**Staff Name**	**Department**	**In**	**Out**	**Working Hours**
0	6/15/2018	Jacky Lee	IT	6:12	20:44	14.53
1	6/15/2018	Joyce Ho	Human Resources	8:26	19:51	11.42
2	6/15/2018	Ivy Chu	Sales	8:29	18:44	10.25
3	6/15/2018	David Chan	Purchasing	8:33	19:50	11.28
4	6/15/2018	Gary Yuen	Logistics	8:34	19:45	11.18
…	…	…	…	…	…	…

Computing the Daily Wages

To compute the daily wages, we need the hourly salary rate of each staff. After that we use the following formula to compute the daily wages.

$$Daily\ Wages = Working\ Hours * Hourly\ Salary\ Rate$$

10. Look up the hourly salary rates for individual staffs by merging two data frames.

```
records = pd.merge(records, hourly_rate, how='left', left_on='Staff Name', right_index=True)
```

In the statement above, we provide the **merge()** function many things including:

- **records, hourly_rate** – two source data frames. The **records** data frame is on left-hand side, the **hourly_rate** data frame is on right-hand side.

- **how='left'** – use *left join* method to merge the data. The merged result includes all rows of the left-hand side but some rows of right-hand side may not be included.
- **left_on='Staff Name'** – the left-hand side, use the 'Staff Name' column for merging.
- **right_index=True** – the right-hand side, use the index column for merging.

After the merging, the records data frame stores the data as follows:

	Date	**Staff Name**	**Department**	**In**	**Out**	**Working Hours**	**Salary Rate (per hour)**
0	6/15/2018	Jacky Lee	IT	6:12	20:44	14.53	60
1	6/15/2018	Joyce Ho	Human Resources	8:26	19:51	11.42	70
2	6/15/2018	Ivy Chu	Sales	8:29	18:44	10.25	55
3	6/15/2018	David Chan	Purchasing	8:33	19:50	11.28	65
4	6/15/2018	Gary Yuen	Logistics	8:34	19:45	11.18	75
…	…	…	…	…	…	…	…

11. Compute the daily wages using the columns "Working Hours" and "Salary Rate (per hour)".

```
records['Daily Wage'] = records['Salary Rate (per hour)'] * records['Working Hours']
```

12. Sort the rows by "Working Hours" and "Staff Name" in **descending** order. The result is stored back in the **records** data frame.

```
records.sort_values(by = ['Working Hours', 'Staff Name'], ascending = False, inplace = True)
```

Creating Summaries using Pivot Table

Pivot Table is a very useful and powerful feature which can be used to summarize, analyze, explore and present our data. We could use a Pivot Table to produce meaningful information from a large table of information, such as:

- The total and daily working hours for each staff
- The total and average working hours for each staff in each department
- The total and average working hours of all staff in each department
- The total and average working hours for each staff

13. Create a pivot table named table1 that contains the total working hours for each staff.

```
table1 = pd.pivot_table(records, values = 'Working Hours', index = 'Staff Name', aggfunc =
np.sum)
```

14. Create a pivot table named table2 that contains the daily working hours grouped by the departments.

```
table2 = pd.pivot_table(records, values = 'Working Hours', index = 'Department',
columns='Date', aggfunc = np.sum)
```

15. Create a pivot table named table3 that contains the total working hours and the total wages, average wages, minimum and maximum daily wages for each staff.

```
table3 = pd.pivot_table(records, values = ['Working Hours', 'Daily Wage'], index = 'Staff Name',
   aggfunc = {'Working Hours':[min, max, np.mean, np.sum], 'Daily Wage':np.sum})
```

16. Create a pivot table named table4 that contains the total wages and working hours grouped by the departments. In addition, we rename the columns by using the **rename()** function.

```
table4 = pd.pivot_table(records, values = ['Daily Wage', 'Working Hours'], index = 'Department',
   aggfunc = np.sum)

table4.rename(columns={'Daily Wage':'Total salary', 'Working Hours':'Total Working Hours'},
inplace = True)
```

Creating Charts

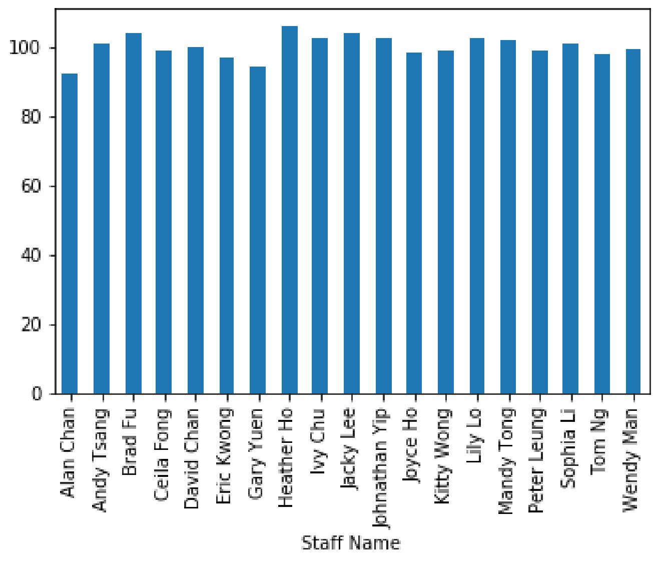
After generating the pivot tables, we could create charts to show the distribution of the working hours among all departments, total working hours of each staff in the period, etc. **Pandas' DataFrame** provides the **plot()** function for creating charts.
The **plot()** function provides many options for us to create different charts. The commonly used options (parameters) are:

- x: label of x axis. If not specify, the index column will be used.
- y: label of y axis.
- kind: the kind of the chart including:
    - 'line' : line plot (default)
    - 'bar' : vertical bar plot
    - 'barh' : horizontal bar plot
    - 'hist' : histogram
    - 'box' : boxplot
    - 'kde' : Kernel Density Estimation plot
    - 'density' : same as 'kde'
    - 'area' : area plot
    - 'pie' : pie plot
    - 'scatter' : scatter plot
    - 'hexbin' : hexbin plot
- ax: matplotlib axes object.

- title: add the title to the chart.
- legend: hide, show or reverse the legend.

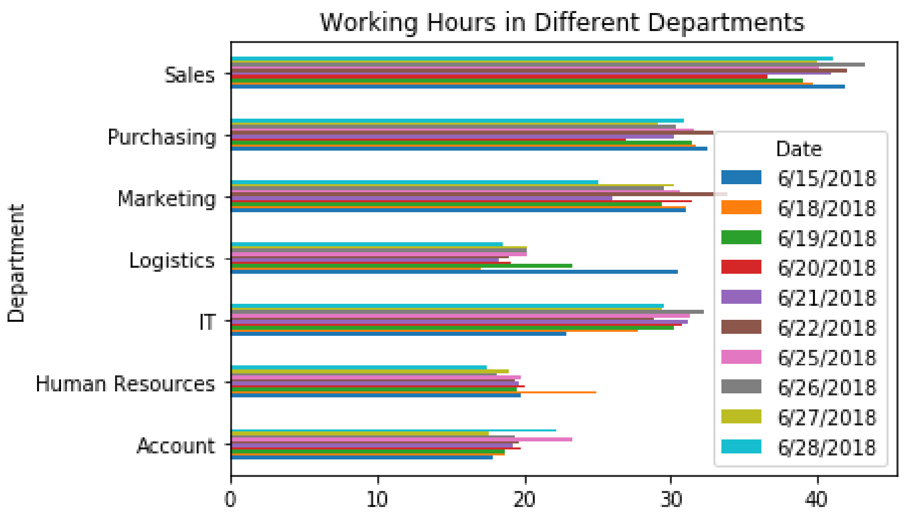
17. Create a vertical bar chart using the data of **table1** with parameters – *kind = 'bar' and legend = False*:

```
table1.plot(kind='bar', legend = False)
```



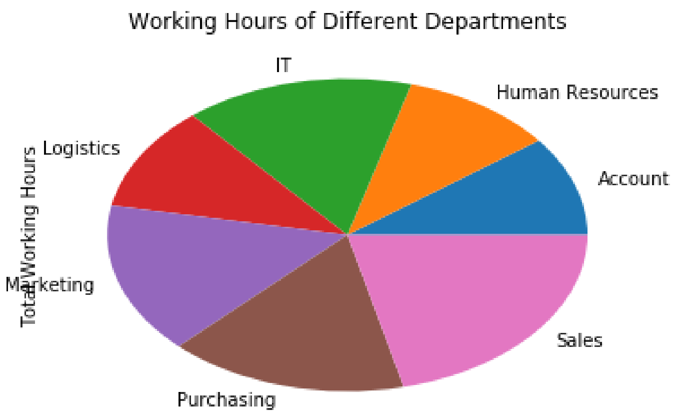
18. Create a horizontal bar chart using the data of **table2** with parameters – title = *'Working Hours in Different Departments' and kind = 'barh'*.

```
table2.plot(title = 'Working Hours in Different Departments', kind = 'barh')
```



19. Create a pie chart using the data in the column “Total Working Hours” of **table4** with parameters – *kind = ‘pie’, y = ‘Total Working Hours’ and legend = False*:

```
table4.plot(title = 'Working Hours of Different Departments', kind = 'pie', y = 'Total Working Hours', legend = False)
```



Exercise

Exercise 1 – Calculating Discounted Amount

Write a program to load the data from the data file named **payment_records.csv** to a data frame. Then, add two additional columns – discount rate and discounted amount. The discount rate can be found in another data file named **vip.csv**. The discounted amount can be calculated using the following formula:

$$Discounted_{Amount} = Amount * Discount\_rate$$

You need to use the “VIP Level” to match the discount rate for each record. And, save the data to another file named result.csv.

Exercise 2 – Pivot Table about Payment Method

Write a program to create a pivot table to show the sum of amounts in different payment methods (Mastercard, VISA, and Cash).

More Visualization Practice using Matplotlib

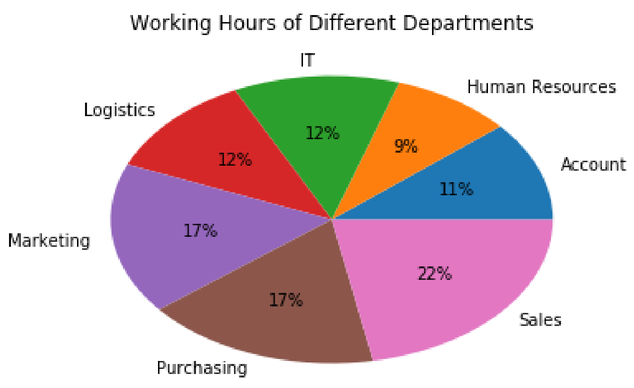
You may find that there may be some problems on the charts, such as the bars go behind the legend, text are overlapped, etc. To solve the problems and create some advanced charts, we use the library **Matplotlib** too. Let's explore the common chart functions include **pie()**, **bar()** and **barh()**.

Pie Chart

The following example creates a pie chart with percentage labels using **Mathplotlib's pie()**. The parameters of the **pie()** function used in the example means:

- data: the chart data.
- labels: the text label shown for each slice.
- autopct: the value label with a formatted string "%.0f%%". In the formatted string, the string "%.0f" represents a float with no decimal place. The double-percent signs are used to add a percent sign to the label. With the string "%.0f%%", we can show the float numbers as a percentage with no decimal place.

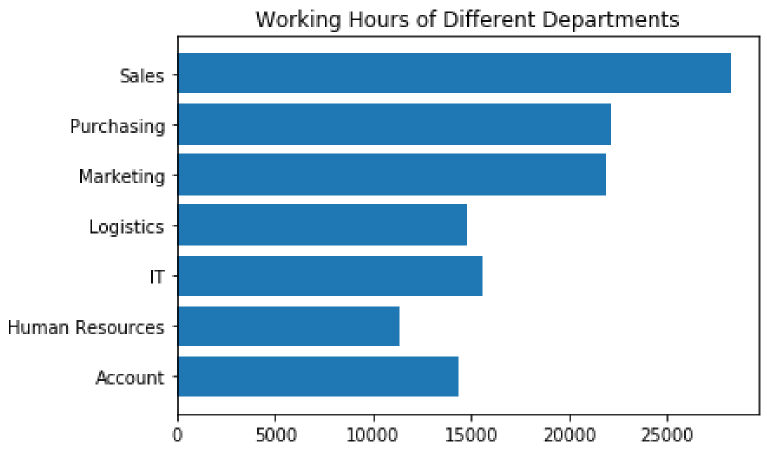
```
import matplotlib.pyplot as plt

plt.pie(table4['Total salary'], labels=table4.index, autopct='%.0f%%')
plt.title('Working Hours of Different Departments')
plt.show()
```



Horizontal Bar Chart

The following example creates a horizontal bar chart using **Matplotlib's barh()** function. We pass two parameters to the **barh()** function as chart data and **table4.index** is used as Y axis labels, **table4['Total salary']** is used as X axis labels.

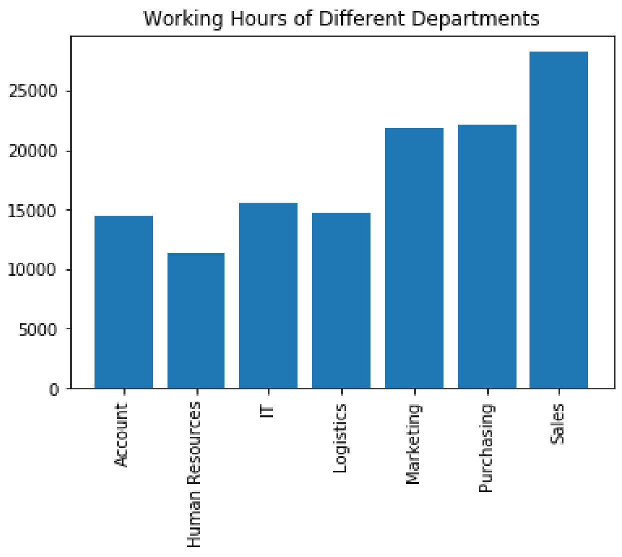
```
plt.barh(table4.index, table4['Total salary'])
plt.title('Working Hours of Different Departments')
plt.show()
```



Vertical Bar Chart

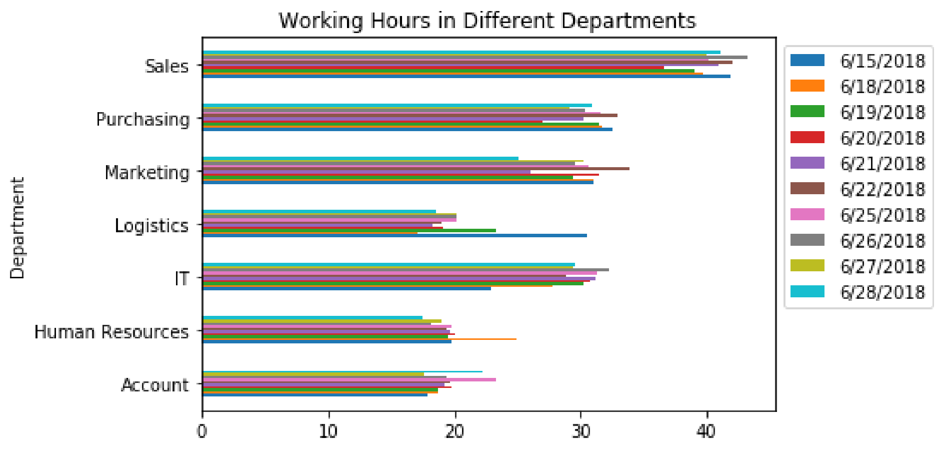
The following example creates a vertical bar chart using **Matplotlib's bar()** function. We pass two parameters to the **bar()** function as chart data and **table4.index** is used as X axis labels, **table4['Total salary']** is used as Y axis labels.

```
plt.bar(table4.index, table4['Total salary'])
plt.xticks(rotation='vertical')
plt.title('Working Hours of Different Departments')
plt.show()
```



MatPlotlib Together with Pandas' Plot

Creating a chart below using **MatPlotlib** along is not easy. But, the legend overlaps with the bars if we create a chart using the **Pandas' plot()** function. Actually, the problem can be solved by using both **MatPlotlib** and **Pandas' plot()** function.



The procedure is as follows:

1. Create a figure using Matplotlib.

```
fig = plt.figure()
```

2. Create a bar chart using Pandas' plot() function with the axes of the figure. The **fig.gca()** function returns the current axes of the figure.

```
table2.plot(title = 'Working Hours in Different Departments', kind = 'barh', ax=fig.gca())
```

3. Move the legend outside the chart area.

```
plt.legend(bbox_to_anchor=(1,1))
```

4. Show the chart.

```
plt.show()
```