Experiment No.: 8

Title: Write a Python program to perform table pivoting and Data/time functionality.

Objectives:

1. Learn how to perform table pivoting.
2. Implement Date/time functionality

Theory:

A pivot table is a way of summarizing data in a Dataframe for a particular purpose. It makes heavy use of the aggregation function. A pivot table itself a Dataframe, where the row represent one variable that interested in, the columns another and the cell's some aggregate value.

Using a panda's pivot table can be a good alternative because it is:

- Quicker (once it is set up)
- Self documenting (look at the code and you know what it does)
- Easy to use to generate a report or email
- More flexible because you can define custom aggregation functions

Exploring the Titanic Dataset using Pandas in Python

Let's import the relevant libraries:

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

plt.style.use('ggplot')
```

```
df = pd.read_csv('file path.csv')

df.head()
```

dropping a few features to make it easier to analyze the data and demonstrate the capabilities of the *pivot_table* function:

```
df.drop(['PassengerId','Ticket','Name'],inplace=True,axis=1)
```

- ***pivot_table*** requires a ***data*** and an ***index*** parameter
- ***data*** is the Pandas dataframe you pass to the function

- ***index*** is the feature that allows you to group your data. The index feature will appear as an index in the resultant table

```
#a single index
table = pd.pivot_table(data=df,index=['Sex'])
table
```

let's visualize the finding:

```
table.plot(kind='bar');
```

Run a pivot with a multi-index?

You can even use more than one feature as an index to group your data. This increases the level of granularity in the resultant table and you can get more specific with your findings:

```
#multiple indexes
table = pd.pivot_table(df,index=['Sex','Pclass'])
table
```

Using multiple indexes on the dataset enables us to concur that the disparity in ticket fare for ***female*** and ***male*** passengers was valid across every ***Pclass*** on Titanic.

Different aggregation function for different features:

The values shown in the table are the result of the summarization that ***aggfunc*** applies to the feature data. ***aggfunc*** is an aggregate function that ***pivot_table*** applies to your grouped data.

By default, it is ***np.mean()***, but you can use different aggregate functions for different features too! Just provide a dictionary as an input to the ***aggfunc*** parameter with the feature name as the key and the corresponding aggregate function as the value.

using ***np.mean()*** for the ***'Age'*** feature and ***np.sum()*** for the ***'Survived'*** feature:

```
#different aggregate functions
table = pd.pivot_table(df,index=['Sex','Pclass'],aggfunc={'Age':np.mean,'Survived':np.sum})
table
```

Aggregate on specific features with values parameter

value parameter is where you tell the function which features to aggregate on. It is an optional field and if you don't specify this value, then the function will aggregate on all the numerical features of the dataset:

```
table = pd.pivot_table(df,index=['Sex','Pclass'],values=['Survived'], aggfunc=np.mean)
table
table.plot(kind='bar');
```

The survival rate of passengers aboard the Titanic decreased with a degrading Pclass among both the genders. Moreover, the survival rate of male passengers was lower than the female passengers in any given Pclass.

Find the relationship between features with columns parameter

Using multiple features as indexes is fine, but using some features as columns will help you to intuitively understand the relationship between them. Also, the resultant table can always be better viewed by incorporating the ***columns*** parameter of the ***pivot_table***.

This ***columns*** parameter is optional and displays the values horizontally on the top of the resultant table.

Both ***columns*** and the ***index*** parameters are optional, but using them effectively will help you to intuitively understand the relationship between the features.

```
#columns
table = pd.pivot_table(df,index=['Sex'],columns=['Pclass'],values=['Survived'],aggfunc=np.sum)
table

table.plot(kind='bar');
```

Date and time

Date and time features find importance in data science problems spanning industries from sales, marketing, and finance to HR, e-commerce, retail, and many more. Predicting how the stock markets will behave tomorrow, how many products will be sold in the upcoming week, when is the best time to launch a new product, how long before a position at the company gets filled, etc. are some of the problems that we can find answers to using date and time data.

The **date** class in the **DateTime** module of Python deals with dates in the Gregorian calendar. It accepts three integer arguments: year, month, and day.

```
from datetime import date
d1 = date(2020,4,23)
print(d1)
print(type(d1))
```

You can see how easy it was to create a date object of **datetime** class. And it's even easier to extract features like day, month, and year from the date. This can be done using the **day**, **month**, and **year** attributes.

```
# present day date
d1 = date.today()
print(d1)
# day
print('Day :',d1.day)
# month
print('Month :',d1.month)
# year
print('Year :',d1.year)
```

time is another class of the DateTime module that accepts integer arguments for time up to

microseconds and returns a DateTime object

```
from datetime import time
t1 = time(13,20,13,40)
print(t1)
print(type(t1))
```

Extract features like **hour**, **minute**, **second**, and **microsecond** from the time object using the respective attributes

```
# hour
print('Hour :',t1.hour)
# minute
print('Minute :',t1.minute)
# second
print('Second :',t1.second)
# microsecond
print('Microsecond :',t1.microsecond)
```

datetime is a class and an object in Python's *DateTime* module, just like date and time. The arguments are a combination of date and time attributes, starting from the year and ending in microseconds.

```
from datetime import datetime
d1 = datetime(2020,4,23,11,20,30,40)
print(d1)
print(type(d1))
```