Chapter 2

Working with Dates and Times

2.1 Overview

In Python, there are core modules that contain base functionality to deal with dates and time, as well as the **pandas** and **NumPy** packages containing other methods. We will be focusing on the **pandas** methods for manipulating dates and times, using the **pandas** `Timestamp` object that is built upon the `NumPy` `datetime64` object.

2.2 Working with the `Timestamp` Object

An object of this type can be created using the `pd.to_datetime()` function on a string containing the date and time. The format of how the date time is constructed in the string can be done in a number of ways, where the function makes an appropriate guess as to what represents what in your date/time. The below strings all create the same `Timestamp` object.

```
>>> import pandas as pd
>>> pd.to_datetime("1999/01/12 22:01:00")

Timestamp('1999-01-12 22:01:00')

>>> pd.to_datetime("22:01 19990112")

Timestamp('1999-01-12 22:01:00')

>>> pd.to_datetime("199901122201")

Timestamp('1999-01-12 22:01:00')

>>> pd.to_datetime("12th Jan 1999 10:01pm")

Timestamp('1999-01-12 22:01:00')
```

Often our dates can be in more complex formats than those above. For example, the following should make the same time stamp as above, but doesn't:

```
>>> pd.to_datetime("12/01/1999 22:01")

Timestamp('1999-12-01 22:01:00')
```

We wish to have the 12th January date here, but have the 1st December returned. To amend this, we must use the `format` argument, which takes a string of strftime as follows:

```
>>> pd.to_datetime("12/01/1999 22:01", format="%d/%m/%Y %H:%M")

Timestamp('1999-01-12 22:01:00')
```

The format argument introduced above takes a strftime string, see http://strftime.org/ for the code values of each directive.

2.2.1 Extracting Information from `Timestamp` Objects

`Timestamp` objects have an easy way of extracting characteristics in the form of attributes. These attributes are accessed using the dot "." and the attribute name; e.g. `Timestamp.hours`

```
>>> my_day = pd.to_datetime("12th Jan 1999 10:01pm")
>>> my_day.year

1999

>>> my_day.weekday_name

'Tuesday'
```

There are a large number of useful attributes for Timestamp objects, a full list of these attributes are detailed in the objects documentation http://pandas.pydata.org/pandas-docs/stable/generated/pandas.Timestamp.html.

When our data is stored in `Series` objects, we first need to use the `.dt` accessor to extract information.

2.3 The `pd.date_range` Function

Often, it is useful to create a sequence of dates, and this can be done via the `date_range` function from the `pandas` package. This creates an object of `DatetimeIndex` that contains a sequence of `Timestamp` objects.

```
>>> my_range = pd.date_range(start="1999-12-01", end="1999-12-09")
>>> my_range

DatetimeIndex(['1999-12-01', '1999-12-02', '1999-12-03', '1999-12-
04',
               '1999-12-05', '1999-12-06', '1999-12-07', '1999-12-
08',
               '1999-12-09'],
              dtype='datetime64[ns]', freq='D')
```

The default is to create a daily sequence but we can change that with either the `periods` argument or the `freq` argument or both.

```
>>> res = pd.date_range("19991201", "20000101", freq="4D")
>>> res

DatetimeIndex(['1999-12-01', '1999-12-05', '1999-12-09', '1999-12-
13',
               '1999-12-17', '1999-12-21', '1999-12-25', '1999-12-
29'],
              dtype='datetime64[ns]', freq='4D')

>>> pd.date_range(start="20000101", periods=4)

DatetimeIndex(['2000-01-01', '2000-01-02', '2000-01-03', '2000-01-
04'], dtype='datetime64[ns]', freq='D')
```

2.4 Date Arithmetic

Dates and times are stored as numbers which means you can perform some arithmetic operations on them. For example, we can subtract two dates from each other.

```
>>> time_difference = pd.to_datetime("2018-01-31") -
pd.to_datetime("2018-01-01")
>>> time_difference

Timedelta('30 days 00:00:00')
```

This returns a `Timedelta` object which represents a period of time and is a subclass of `np.timedelta64` and `datetime.timedelta`. It shares some properties with the `Timestamp` which we can access in much the same way.

```
>>> time_difference.days

30

>>> time_difference.seconds

0
```

We can also create our own `Timedelta` object and add that to a `Timestamp` to do more complicated date arithmetic.

```
>>> newYearEve = pd.to_datetime("2017-12-31 23:59:59")
>>> newYearEve + pd.Timedelta("2 seconds")

Timestamp('2018-01-01 00:00:01')

>>> newYearEve + pd.Timedelta(3, unit='M')

Timestamp('2018-04-02 07:27:17')

>>> from pandas.tseries.offsets import *
>>> newYearEve - YearEnd(1)

Timestamp('2016-12-31 23:59:59')
```

See http://pandas.pydata.org/pandas-docs/stable/timeseries.html#timeseries-offsets for a full list off short hand versions of time spans.

1. Load in `dji.csv`, reformat the Date column, and store the weekday of that date in an additional column.
2. How many times does Monday occur in the data?
3. Load in `air_passengers.csv`. The Time column is in the format Month-Year. Reformat the column to a Day-Month-Year format.