Visualising Time Series Data

In the section, we will explore ways to visualise data gathered over time. We will:

- Plot simple time series plots
- Derive variables such as month and year and use them for richer visualisations

In [21]:
```
# loading libraries and reading the data

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# set seaborn theme if you prefer
sns.set(style="white")

# read data
market_df = pd.read_csv("./global_sales_data/market_fact.csv")
customer_df = pd.read_csv("./global_sales_data/cust_dimen.csv")
product_df = pd.read_csv("./global_sales_data/prod_dimen.csv")
shipping_df = pd.read_csv("./global_sales_data/shipping_dimen.csv")
orders_df = pd.read_csv("./global_sales_data/orders_dimen.csv")
```

Visualising Simple Time Series Data

Let's say you want to visualise numeric variables such as `Sales`, `Profit`, `Shipping_Cost` etc. over time.

In [22]:
```
market_df.head()
```

Out[22]:

	Ord_id	Prod_id	Ship_id	Cust_id	Sales	Discount	Order_Quantity	Profit
0	Ord_5446	Prod_16	SHP_7609	Cust_1818	136.81	0.01	23	-30.51
1	Ord_5406	Prod_13	SHP_7549	Cust_1818	42.27	0.01	13	4.56
2	Ord_5446	Prod_4	SHP_7610	Cust_1818	4701.69	0.00	26	1148.90
3	Ord_5456	Prod_6	SHP_7625	Cust_1818	2337.89	0.09	43	729.34
4	Ord_5485	Prod_17	SHP_7664	Cust_1818	4233.15	0.08	35	1219.87

Since the `Order_Date` variable is in the orders dataframe, let's merge it.

In [23]:
```
# merging with the Orders data to get the Date column
df = pd.merge(market_df, orders_df, how='inner', on='Ord_id')
df.head()
```

Out[23]:

	Ord_id	Prod_id	Ship_id	Cust_id	Sales	Discount	Order_Quantity	Profit
0	Ord_5446	Prod_16	SHP_7609	Cust_1818	136.81	0.01	23	-30.51
1	Ord_5446	Prod_4	SHP_7610	Cust_1818	4701.69	0.00	26	1148.90
2	Ord_5446	Prod_6	SHP_7608	Cust_1818	164.02	0.03	23	-47.64
3	Ord_5406	Prod_13	SHP_7549	Cust_1818	42.27	0.01	13	4.56
4	Ord_5456	Prod_6	SHP_7625	Cust_1818	2337.89	0.09	43	729.34

In [24]:
```
# Now we have the Order_Date in the df
# It is stored as a string (object) currently
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 8399 entries, 0 to 8398
Data columns (total 13 columns):
Ord_id                 8399 non-null object
Prod_id                8399 non-null object
Ship_id                8399 non-null object
Cust_id                8399 non-null object
Sales                  8399 non-null float64
Discount               8399 non-null float64
Order_Quantity         8399 non-null int64
Profit                 8399 non-null float64
Shipping_Cost          8399 non-null float64
Product_Base_Margin    8336 non-null float64
Order_ID               8399 non-null int64
Order_Date             8399 non-null object
Order_Priority         8399 non-null object
dtypes: float64(5), int64(2), object(6)
memory usage: 721.8+ KB
```

Since `Order_Date` is a string, we need to convert it into a `datetime` object. You can do that using `pd.to_datetime()`.

In [25]:
```
# Convert Order_Date to datetime type
df['Order_Date'] = pd.to_datetime(df['Order_Date'])

# Order_Date is now datetime type
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 8399 entries, 0 to 8398
Data columns (total 13 columns):
Ord_id                 8399 non-null object
Prod_id                8399 non-null object
Ship_id                8399 non-null object
Cust_id                8399 non-null object
Sales                  8399 non-null float64
Discount               8399 non-null float64
Order_Quantity         8399 non-null int64
Profit                 8399 non-null float64
Shipping_Cost          8399 non-null float64
Product_Base_Margin    8336 non-null float64
Order_ID               8399 non-null int64
Order_Date             8399 non-null datetime64[ns]
Order_Priority         8399 non-null object
dtypes: datetime64[ns](1), float64(5), int64(2), object(5)
memory usage: 754.6+ KB
```

Now, since on each day, multiple orders were placed, we need to aggregate `Sales` using a metric such as mean, median etc., and then create a time series plot.

We will group by `Order_Date` and compute the sum of `Sales` on each day.

In [26]:
```
# aggregating total sales on each day
time_df = df.groupby('Order_Date')['Sales'].sum()
print(time_df.head())

print(type(time_df))
```

```
Order_Date
2009-01-01    1052.8400
2009-01-02    5031.9000
2009-01-03    7288.1375
2009-01-04    6188.4245
2009-01-05    2583.3300
Name: Sales, dtype: float64
<class 'pandas.core.series.Series'>
```

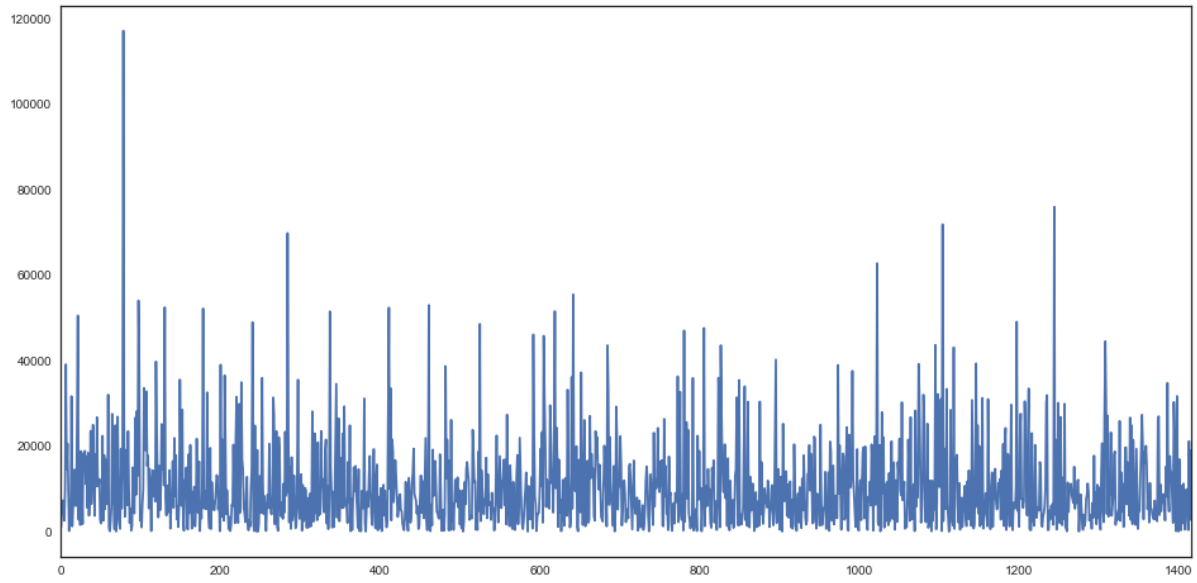
We can now create a time-series plot using `sns.tsplot()`.

In [27]:
```
# time series plot

# figure size
plt.figure(figsize=(16, 8))

# tsplot
sns.tsplot(data=time_df)
plt.show()
```

```
c:\users\pratika\appdata\local\programs\python\python35-32\lib\site-packages
\seaborn\timeseries.py:183: UserWarning: The tsplot function is deprecated an
d will be removed or replaced (in a substantially altered version) in a futur
e release.
  warnings.warn(msg, UserWarning)
```



Using Derived Date Metrics for Visualisation

It is often helpful to use derived variables from date such as month and year and using them to identify hidden patterns.

In [28]:
```
# extracting month and year from date

# extract month
df['month'] = df['Order_Date'].dt.month

# extract year
df['year'] = df['Order_Date'].dt.year

df.head()
```

Out[28]:

	Ord_id	Prod_id	Ship_id	Cust_id	Sales	Discount	Order_Quantity	Profit
0	Ord_5446	Prod_16	SHP_7609	Cust_1818	136.81	0.01	23	-30.51
1	Ord_5446	Prod_4	SHP_7610	Cust_1818	4701.69	0.00	26	1148.90
2	Ord_5446	Prod_6	SHP_7608	Cust_1818	164.02	0.03	23	-47.64
3	Ord_5406	Prod_13	SHP_7549	Cust_1818	42.27	0.01	13	4.56
4	Ord_5456	Prod_6	SHP_7625	Cust_1818	2337.89	0.09	43	729.34

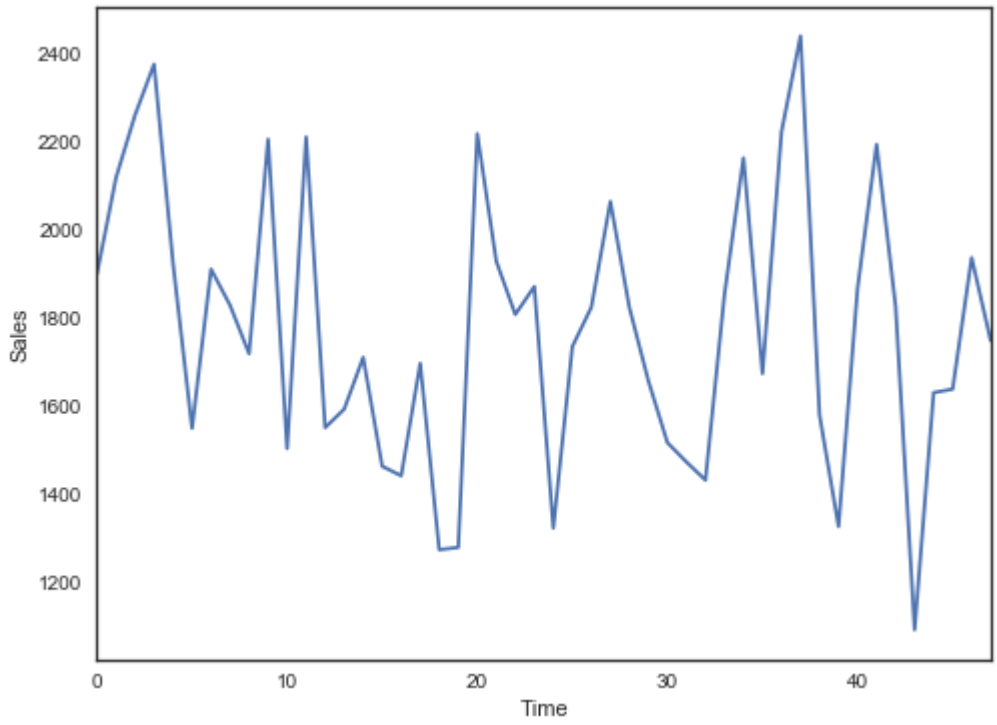
Now you can plot the average sales across years and months.

In [29]:
```
# grouping by year and month
df_time = df.groupby(["year", "month"]).Sales.mean()
df_time.head()
```

Out[29]:
```
year  month
2009  1        1898.475090
      2        2116.510723
      3        2258.661599
      4        2374.155868
      5        1922.317055
Name: Sales, dtype: float64
```

In [30]:
```
plt.figure(figsize=(8, 6))
# time series plot
sns.tsplot(df_time)
plt.xlabel("Time")
plt.ylabel("Sales")
plt.show()
```

```
c:\users\pratika\appdata\local\programs\python\python35-32\lib\site-packages
\seaborn\timeseries.py:183: UserWarning: The tsplot function is deprecated an
d will be removed or replaced (in a substantially altered version) in a futur
e release.
  warnings.warn(msg, UserWarning)
```



There is another way to visualise numeric variables, such as `Sales`, across the year and month. We can pivot the `month` column to create a wide-format dataframe, and then plot a heatmap.

In [31]:
```
# Pivoting the data using 'month'
year_month = pd.pivot_table(df, values='Sales', index='year', columns='month',
aggfunc='mean')
year_month.head()
```

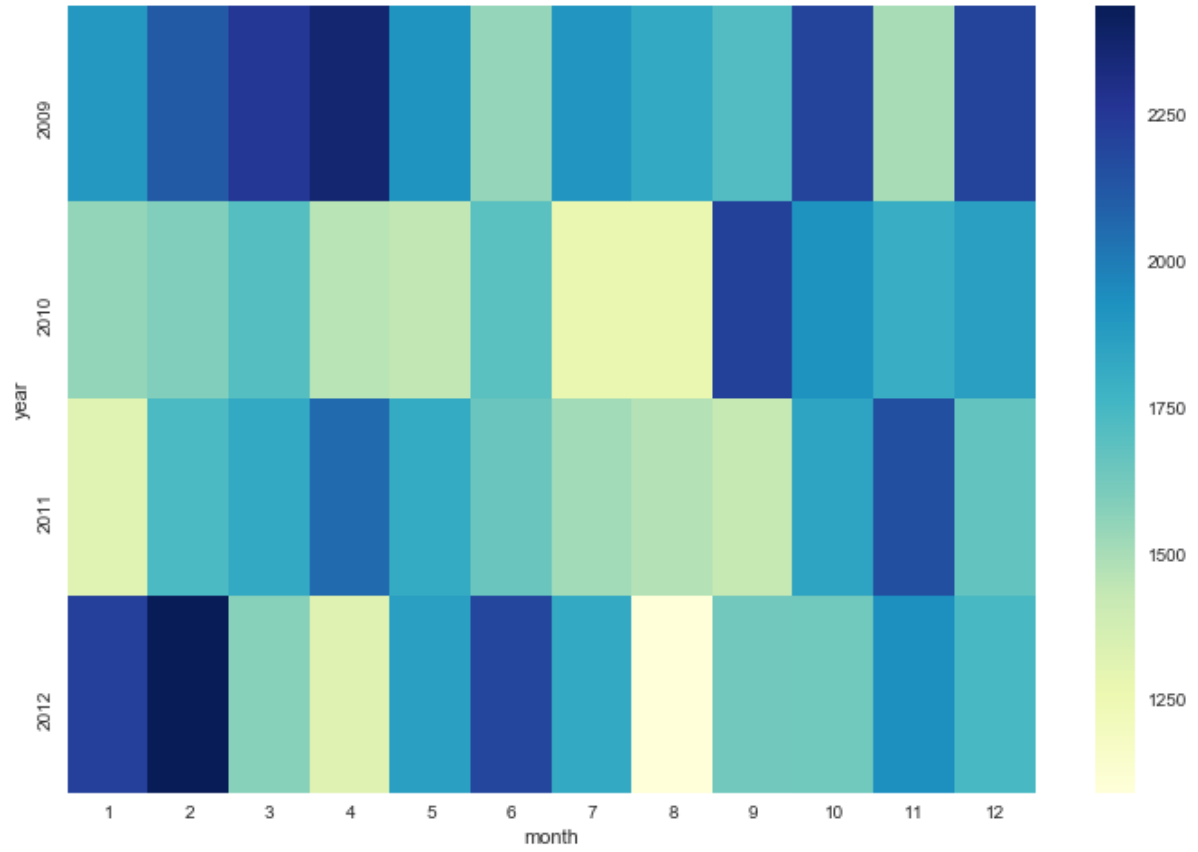
Out[31]:

month	1	2	3	4	5	6
year						
2009	1898.475090	2116.510723	2258.661599	2374.155868	1922.317055	1548.093259
2010	1549.664361	1591.532297	1708.934944	1461.935539	1440.393540	1695.397085
2011	1321.671562	1733.378070	1822.860614	2062.716921	1822.033936	1655.599644
2012	2220.831551	2438.166961	1578.284028	1325.253694	1865.744629	2192.228263

You can now create a heatmap using `sns.heatmap()`.

In [33]:
```
# figure size
plt.figure(figsize=(12, 8))

# heatmap with a color map of choice
sns.heatmap(year_month, cmap="YlGnBu")
plt.show()
```



Addtional Reading on Time Series Plots and Heatmaps

1. Seaborn heatmaps (documentation) (https://seaborn.pydata.org/generated/seaborn.heatmap.html) (https://seaborn.pydata.org/generated/seaborn.heatmap.html)

(https://seaborn.pydata.org/generated/seaborn.heatmap.html)

(https://seaborn.pydata.org/generated/seaborn.heatmap.html)
(https://seaborn.pydata.org/generated/seaborn.heatmap.html)

(https://seaborn.pydata.org/generated/seaborn.heatmap.html)
(https://seaborn.pydata.org/generated/seaborn.heatmap.html)
(https://seaborn.pydata.org/generated/seaborn.heatmap.html)