Lab 5 Seminar Activity 2

Sample warehouse Operations

Merging data in different Excel sheets into a into a dataframe. Open the 'berlin_temp.xlsx' dataset in excel sheet first to understand about the sheets.

In [13]:
```
import numpy as np
import pandas as pd
```

Use berlin_temp.xlsx workbook which can be downloaded from moodle

Choice 1: Consolidating Excel sheets to a dataframe without including the sheet name

In [14]:
```
data = pd.concat(pd.read_excel('berlin_temp.xlsx', sheet_name= [0, 1, 2, 3, 4, 5]).values())
data.head()
```

Out[14]:

	date	ta	yr
0	2000-01-01	1.2	2000
1	2000-01-02	3.6	2000
2	2000-01-03	5.7	2000
3	2000-01-04	5.1	2000
4	2000-01-05	2.2	2000

Choice 2: Consolidating Excel sheets to a dataframe by including sheet names

In [15]:
```
df_dict = pd.read_excel("berlin_temp.xlsx", sheet_name=[0, 1, 2, 3, 4, 5])
data = pd.concat(df.assign(sheet_name=name) for name, df in df_dict.items())
data.head()
```

Out[15]:

	date	ta	yr	sheet_name
0	2000-01-01	1.2	2000	0
1	2000-01-02	3.6	2000	0
2	2000-01-03	5.7	2000	0
3	2000-01-04	5.1	2000	0
4	2000-01-05	2.2	2000	0

In [16]:
```
data.tail()
```

Out[16]:

	date	ta	yr	sheet_name
360	2005-12-27	-2.4	2005	5
361	2005-12-28	-4.2	2005	5
362	2005-12-29	-3.4	2005	5
363	2005-12-30	-5.9	2005	5
364	2005-12-31	-4.3	2005	5

In [17]:
```
data.to_csv('combined.csv',header=True)
```

We now have a consolidated dataframe.Next we are going to join multiple csv files

Step 1 : Use glob() to list all files that match a pattern and sort the results

In [18]:
```
from glob import glob
# sorted function is used to sort the files in order.
sales_files = sorted(glob('Sales_Data*.csv'))
sales_files
```

Out[18]:
```
['Sales_Data_1.csv', 'Sales_Data_2.csv']
```

In [19]:
```
%%time
sales = pd.concat((pd.read_csv(file).assign(filename = file) for file in sales_files), ignore_index = True)
```

```
Wall time: 2.55 s
```

We now have a consolidated dataframe. We can see the first five of the dataset by using head() function.

In [20]:
```
sales.head()
```

Out[20]:

	Region	Product	Date	Sales	filename
0	South	Prod F	2015-03-01	99496.651674	Sales_Data_1.csv
1	West	Prod L	2015-01-10	96849.893584	Sales_Data_1.csv
2	West	Prod G	2014-04-24	83845.468432	Sales_Data_1.csv
3	South	Prod H	2014-06-05	86444.985833	Sales_Data_1.csv
4	North	Prod J	2010-03-31	9502.540135	Sales_Data_1.csv

In [21]:
```
sales['Region'].value_counts()
```

Out[21]:
```
North    2499622
South    1250683
West      749194
East      500501
Name: Region, dtype: int64
```

In [22]:
```
sales['Product'].value_counts()
```

Out[22]:
```
Prod L    251432
Prod A    250807
Prod P    250608
Prod F    250507
Prod H    250358
Prod Q    250278
Prod C    250241
Prod K    250166
Prod D    250019
Prod B    250000
Prod I    249968
Prod S    249916
Prod O    249914
Prod J    249747
Prod E    249712
Prod T    249547
Prod R    249546
Prod G    249143
Prod N    249066
Prod M    249025
Name: Product, dtype: int64
```

In [23]:
```
sales_small=sales.iloc[0:1000, 0:4]
sales_small
```

Out[23]:

	Region	Product	Date	Sales
0	South	Prod F	2015-03-01	99496.651674
1	West	Prod L	2015-01-10	96849.893584
2	West	Prod G	2014-04-24	83845.468432
3	South	Prod H	2014-06-05	86444.985833
4	North	Prod J	2010-03-31	9502.540135
...	...	...	...	...
995	South	Prod J	2014-03-10	81630.250672
996	South	Prod P	2014-02-19	80833.480007
997	South	Prod T	2012-06-09	49496.168039
998	North	Prod K	2013-05-03	66174.176464
999	South	Prod D	2013-05-30	67793.802405

1000 rows × 4 columns

```
<div class="alert alert-danger">

<b>NOTE</b>:

 <ul>
  <li>**Pivot** is purely restructuring: a single value for each index/column combination is required.</li>
</ul>
</div>
```

In [24]:
```
# sales_small.sort_values(by=['Date'])
# sales_small.drop_duplicates(subset='Date', keep='first')
```

In [25]:
```
#pivot table using aggfunc = max

sales_small.pivot_table(index='Product',columns='Region', values='Sales', aggfunc='max')
```

Out[25]:

Region	East	North	South	West
Product				
Prod A	86952.800108	106129.481536	121805.785760	122281.729952
Prod B	114194.938476	115361.662006	121688.755993	116081.653456
Prod C	109726.779534	122076.622070	105032.190276	98282.976044
Prod D	111367.773799	119568.686731	122485.976601	49734.280627
Prod E	107684.804700	116020.638238	118193.510442	111757.390585
Prod F	122164.866904	116575.626121	116618.079022	121822.256854
Prod G	93562.244514	117677.004408	113488.980763	118534.995503
Prod H	117754.839582	121213.630752	102278.929389	112973.781370
Prod I	110801.854736	122901.266295	118799.590835	107993.171459
Prod J	120627.665333	122367.462351	122451.905269	108061.868618
Prod K	22788.206668	122467.186131	112491.781408	121325.584869
Prod L	109956.278823	121798.926876	119586.599262	96849.893584
Prod M	95795.089060	110103.543410	122764.142583	106632.721985
Prod N	120062.948284	113228.575958	94938.962980	75736.436058
Prod O	107255.092867	118438.784092	100238.365221	121011.113373
Prod P	122305.877340	118279.272820	119757.220200	118632.590933
Prod Q	119464.279155	118861.741051	118612.594013	100034.697109
Prod R	97076.722021	109527.822510	109045.696857	102238.195325
Prod S	105008.380843	116930.747801	108351.926550	105554.511416
Prod T	121020.369338	119753.829634	119507.562306	61275.852570

In [26]:
```
#pivot table using aggfunc = count

sales_small.pivot_table(index='Product',columns='Region', values='Sales', aggfunc='count')
```

Out[26]:

Region	East	North	South	West
Product				
Prod A	5	14	12	3
Prod B	3	21	17	7
Prod C	8	26	14	11
Prod D	8	26	14	7
Prod E	3	19	15	3
Prod F	3	24	10	11
Prod G	5	27	9	9
Prod H	4	36	11	7
Prod I	7	24	16	10
Prod J	6	32	14	5
Prod K	1	27	16	8
Prod L	2	29	9	9
Prod M	3	28	8	8
Prod N	5	23	6	5
Prod O	7	19	9	13
Prod P	9	30	12	4
Prod Q	3	22	17	6
Prod R	5	25	9	9
Prod S	6	28	14	6
Prod T	7	27	16	4

REMEMBER:

- There is a shortcut function for a `pivot_table` with a `aggfunc=count` as aggregation: `crosstab`

In [27]:
```
pd.crosstab(index=sales_small['Product'], columns=sales_small['Region'])
```

Out[27]:

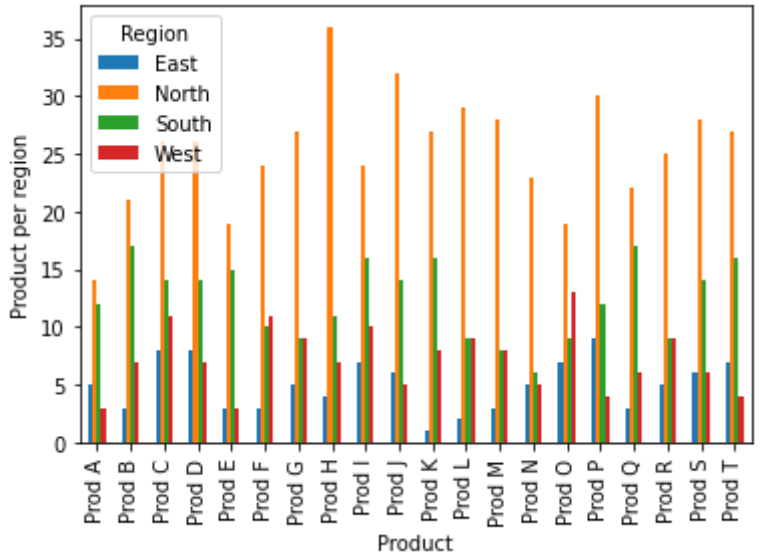
Region	East	North	South	West
Product				
Prod A	5	14	12	3
Prod B	3	21	17	7
Prod C	8	26	14	11
Prod D	8	26	14	7
Prod E	3	19	15	3
Prod F	3	24	10	11
Prod G	5	27	9	9
Prod H	4	36	11	7
Prod I	7	24	16	10
Prod J	6	32	14	5
Prod K	1	27	16	8
Prod L	2	29	9	9
Prod M	3	28	8	8
Prod N	5	23	6	5
Prod O	7	19	9	13
Prod P	9	30	12	4
Prod Q	3	22	17	6
Prod R	5	25	9	9
Prod S	6	28	14	6
Prod T	7	27	16	4

In [29]:
```
#Plotting figures

import matplotlib.pyplot as plt
plt.figure(figsize=(10,4))
```

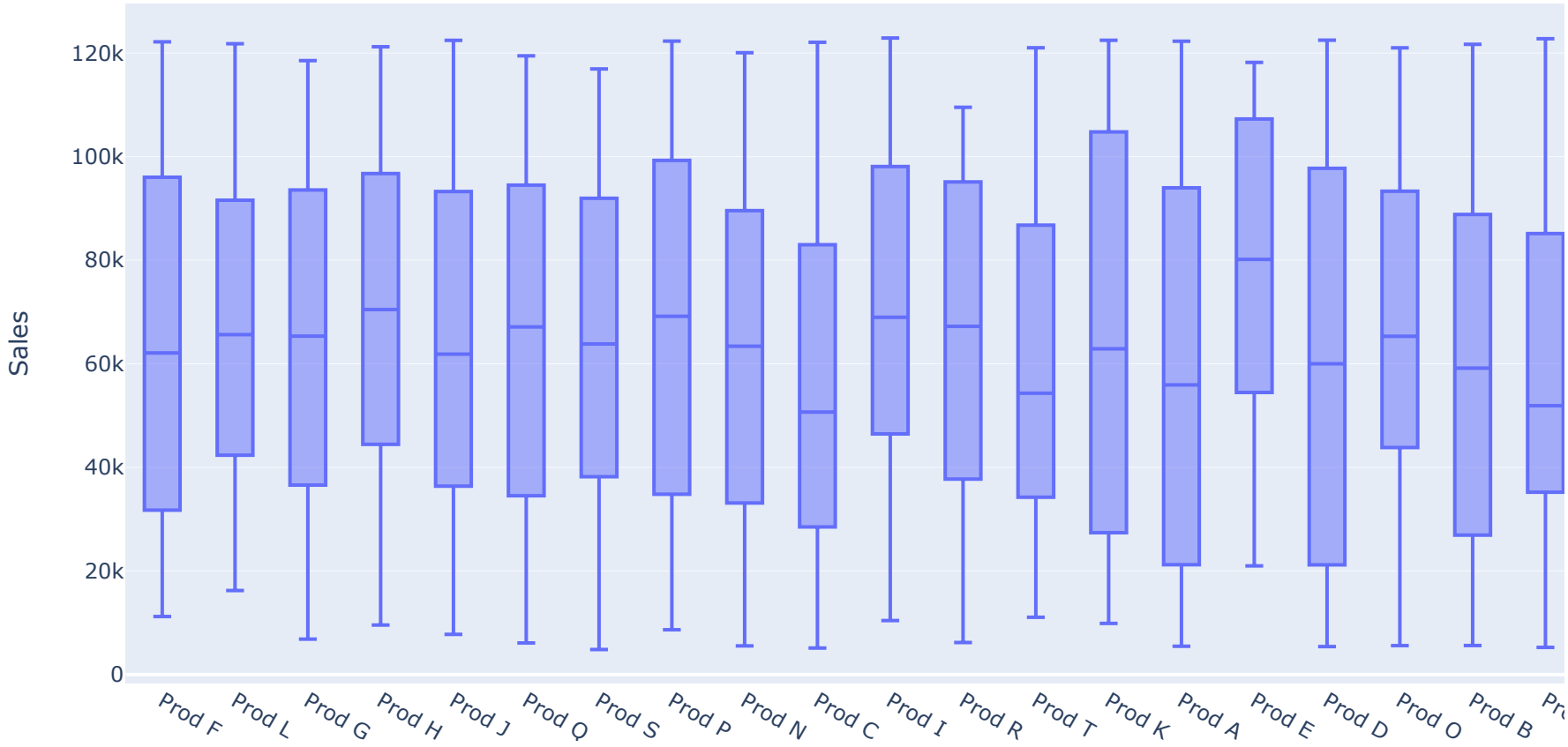
Out[29]:
```
<Figure size 720x288 with 0 Axes>

<Figure size 720x288 with 0 Axes>
```

In [30]:
```
fig, ax1 = plt.subplots()
sales_small.pivot_table(index='Product',columns='Region', values='Sales', aggfunc='count').plot(kind='bar',
                                                        rot=90,ax=ax1)
ax1.set_ylabel('Product per region')
```

Out[30]:
```
Text(0, 0.5, 'Product per region')
```



In [31]:
```
import plotly.express as px
fig = px.box(sales_small,x='Product', y='Sales')
fig.show()
```



Melt

The `melt` function performs the inverse operation of a `pivot`. This can be used to make your frame longer, i.e. to make a *tidy* version of your data.

In [32]:
```
pivoted = sales_small.pivot_table(index='Product',columns='Region', values='Sales').reset_index()
pivoted.columns.name = None
```

In [33]:
```
pivoted
```

Out[33]:

	Product	East	North	South	West
0	Prod A	52761.145704	51509.139511	57636.860711	90203.337385
1	Prod B	69405.811314	54316.913516	58840.082069	78894.881516
2	Prod C	61876.423399	57438.141523	46844.316091	48087.376731
3	Prod D	61027.580766	61228.642700	72231.169976	29572.024537
4	Prod E	74236.973599	82476.438688	71471.063597	97870.755477
5	Prod F	83296.744687	53432.845894	79342.432712	70698.866983
6	Prod G	47415.675811	67354.643355	58081.654116	75297.093644
7	Prod H	63594.920158	72941.985248	66173.331664	63654.470996
8	Prod I	86951.733186	67472.619402	64411.693286	68573.602620
9	Prod J	71512.648398	55865.795627	80996.944354	72971.333466
10	Prod K	22788.206668	61273.050509	73063.587234	72329.664946
11	Prod L	100907.058332	67445.550380	70319.025845	60594.816119
12	Prod M	76039.763689	49724.817827	77863.347855	59120.625348
13	Prod N	77086.565425	62163.660595	61182.209041	42444.408390
14	Prod O	64263.603987	71319.795277	60092.802727	66824.024241
15	Prod P	73733.582405	61967.901835	72091.737859	71886.498838
16	Prod Q	76069.577554	60014.237267	65402.149635	72536.273737
17	Prod R	60161.885754	62064.255500	81120.951875	61911.714350
18	Prod S	61850.053823	59757.177313	69668.562796	60082.692768
19	Prod T	73499.296741	61324.488745	56638.728898	37693.098727

In [34]:
```
pd.melt(pivoted)
```

Out[34]:

	variable	value
0	Product	Prod A
1	Product	Prod B
2	Product	Prod C
3	Product	Prod D
4	Product	Prod E
...	...	...
95	West	71886.498838
96	West	72536.273737
97	West	61911.71435
98	West	60082.692768
99	West	37693.098727

100 rows × 2 columns

As you can see above, the `melt` function puts all column labels in one column, and all values in a second column.

In this case, this is not fully what we want. We would like to keep the 'Sex' column separately:

In [35]:
```
pd.melt(pivoted, id_vars=['Product'],var_name='Region') #, var_name='Pclass', value_name='Fare')
```

Out[35]:

	Product	Region	value
0	Prod A	East	52761.145704
1	Prod B	East	69405.811314
2	Prod C	East	61876.423399
3	Prod D	East	61027.580766
4	Prod E	East	74236.973599
...	...	...	...
75	Prod P	West	71886.498838
76	Prod Q	West	72536.273737
77	Prod R	West	61911.714350
78	Prod S	West	60082.692768
79	Prod T	West	37693.098727

80 rows × 3 columns

You can try using other datasets for your practice. Once you finish both the activities please download a dataset from google, kaggle or any other sources and start working on it which will give you an idea of various datasets.