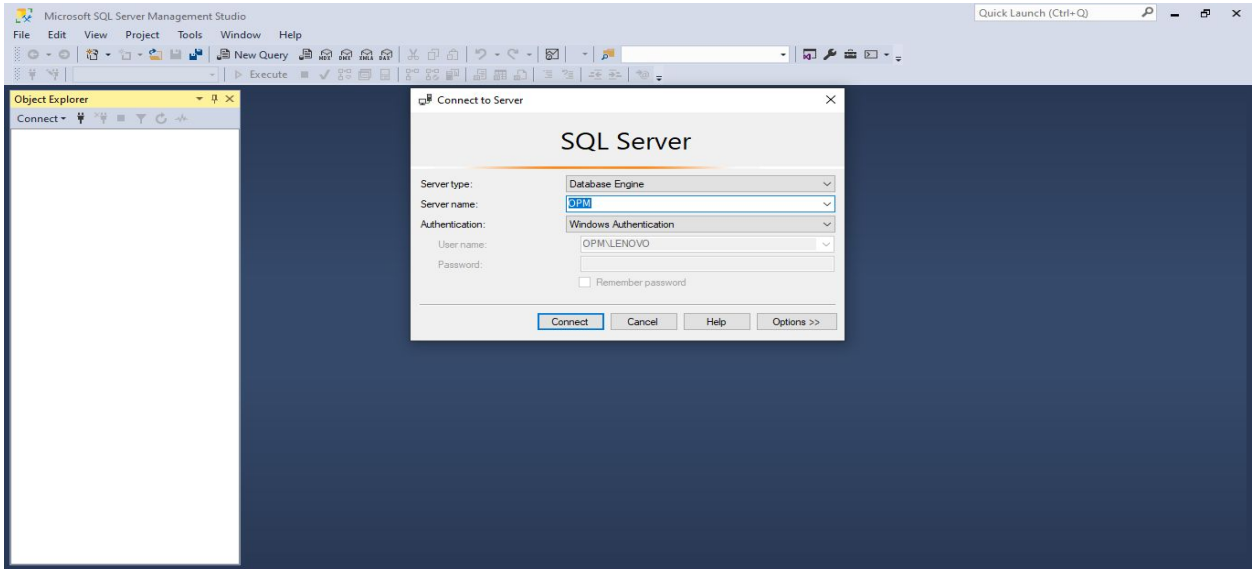
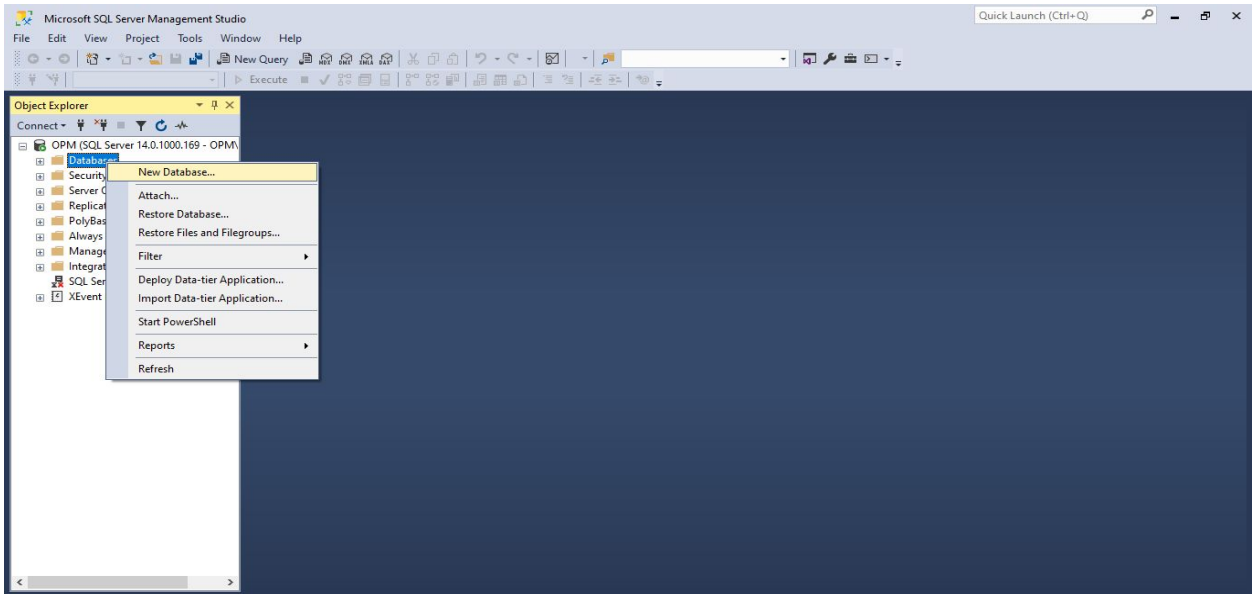
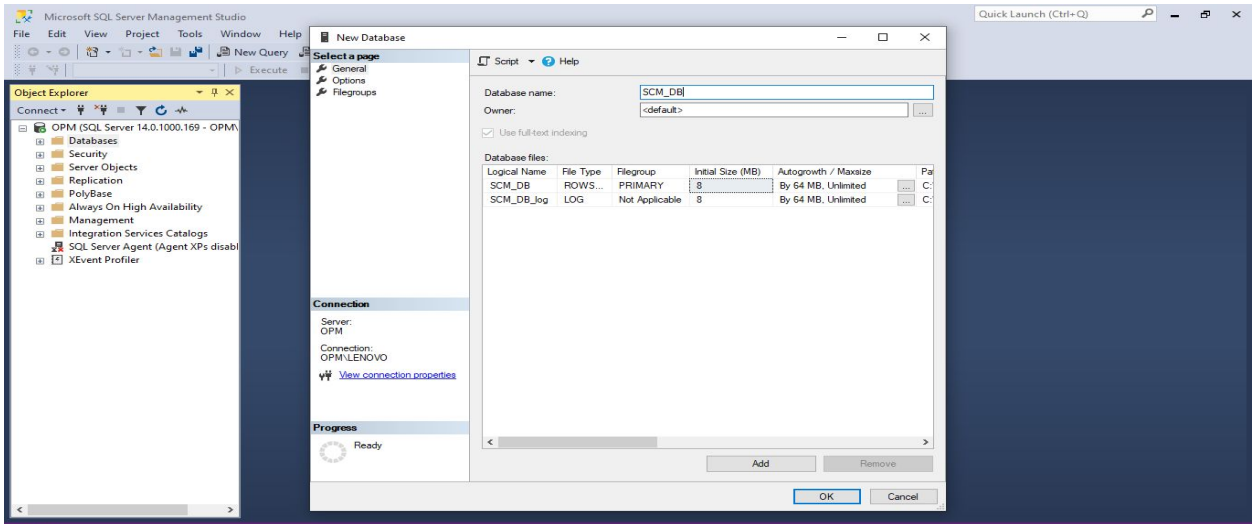
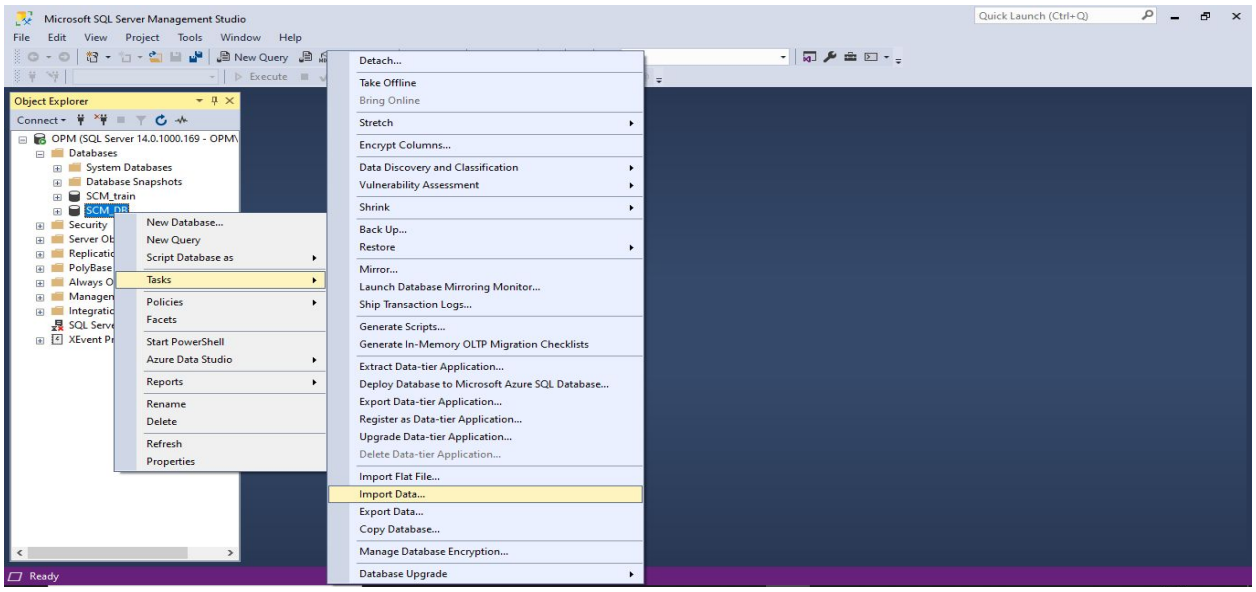
Starting with SQL server management studio.

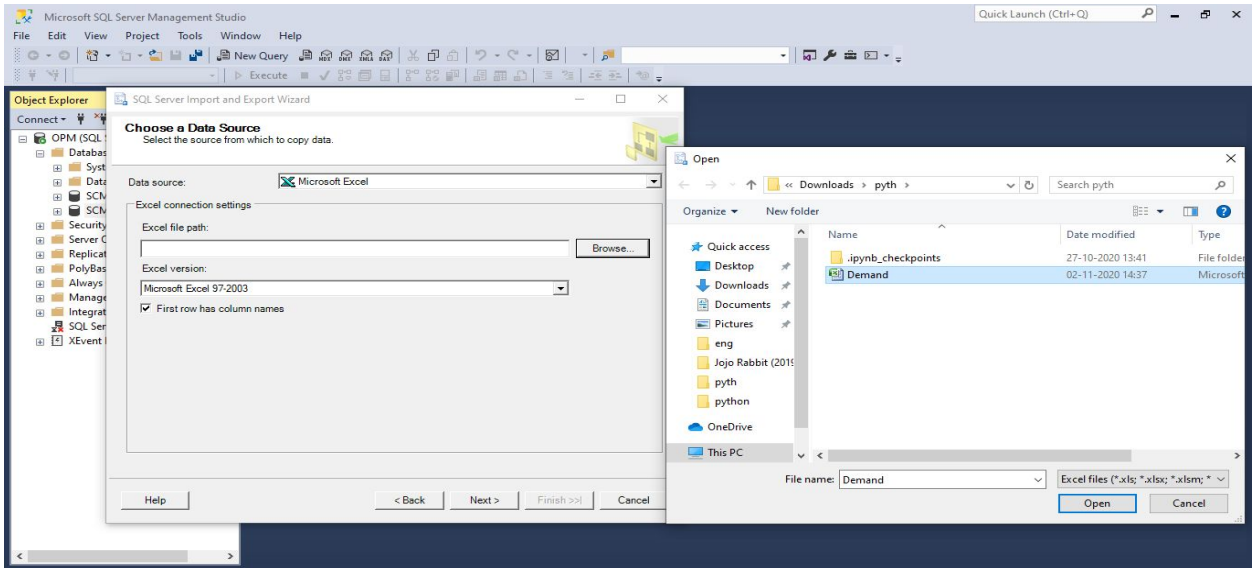
In the opening window, Choose server type as Database Engine, authentication as Windows authentication.

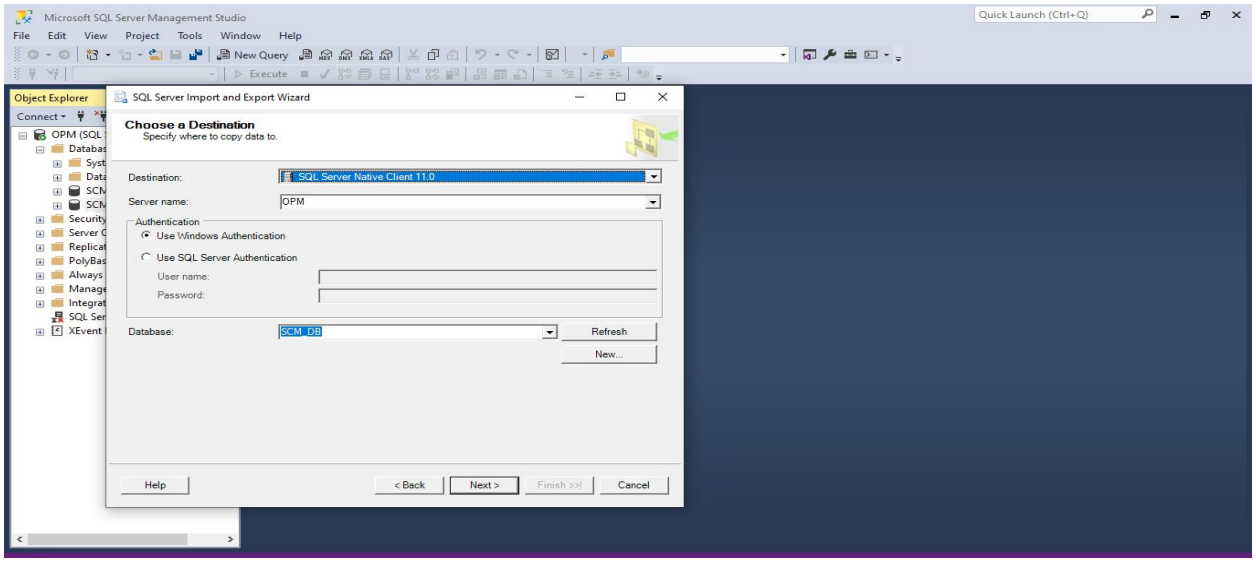


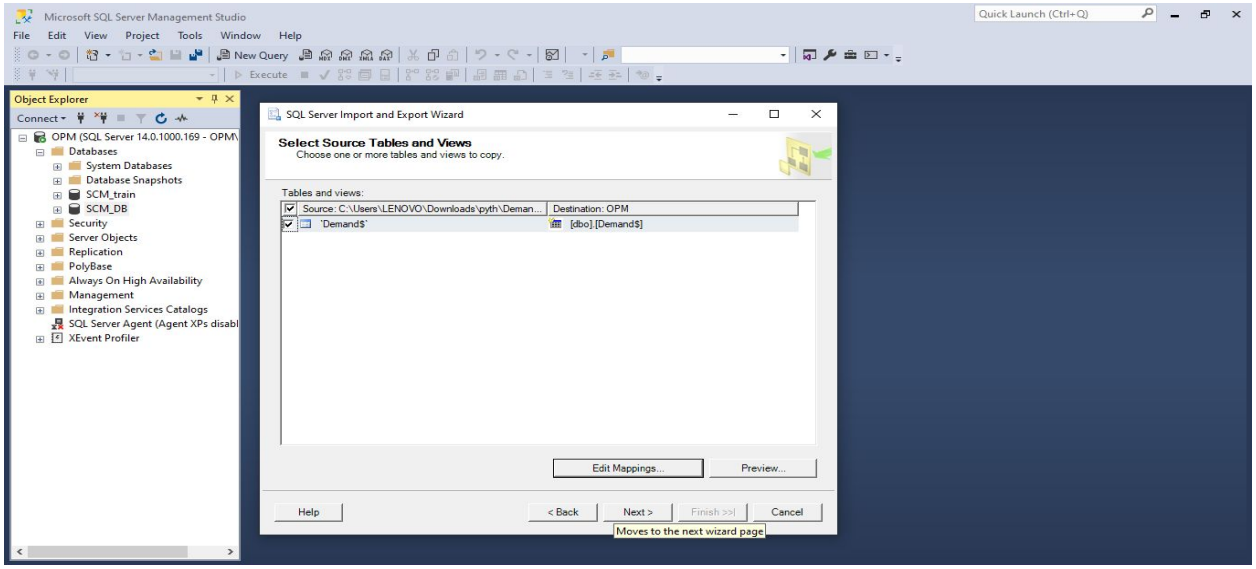


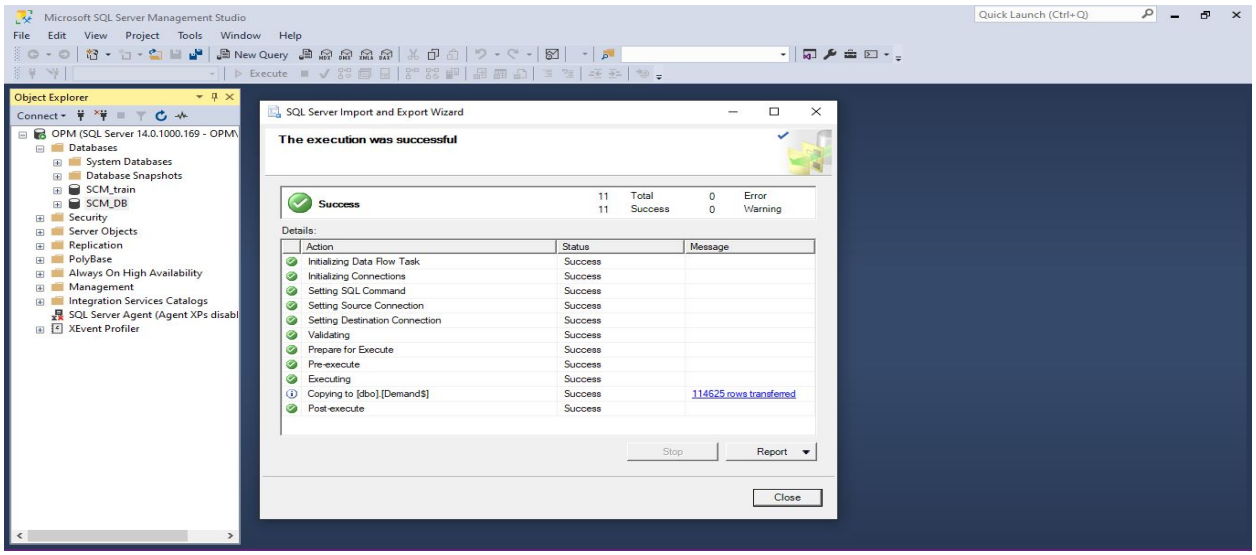


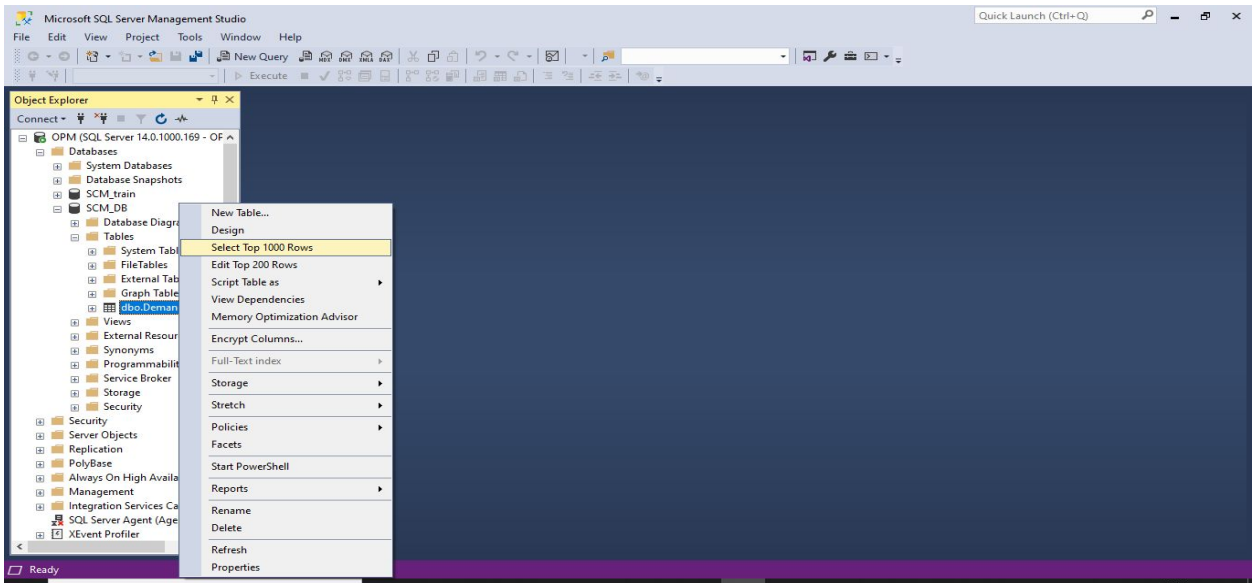

Microsoft SQL Server Management Studio
Quick Launch (Ctrl+Q)
File Edit View Project Tools Window Help
New Query
Execute
Object Explorer
Connect
OPM (SQL Server 14.0.1000.169 - OPM\
Databases
System Databases
Database Snapshots
SCM_train
SCM_DB
Security
Server Ob
Replicatio
PolyBase
Always O
Managem
Integratio
SQL Serv
XEvent Pr
New Database...
New Query
Script Database as
Tasks
Policies
Facets
Start PowerShell
Azure Data Studio
Reports
Rename
Delete
Refresh
Properties
Detach...
Take Offline
Bring Online
Stretch
Encrypt Columns...
Data Discovery and Classification
Vulnerability Assessment
Shrink
Back Up...
Restore
Mirror...
Launch Database Mirroring Monitor...
Ship Transaction Logs...
Generate Scripts...
Generate In-Memory OLTP Migration Checklists
Extract Data-tier Application...
Deploy Database to Microsoft Azure SQL Database...
Export Data-tier Application...
Register as Data-tier Application...
Upgrade Data-tier Application...
Delete Data-tier Application...
Import Flat File...
Import Data...
Export Data...
Copy Database...
Manage Database Encryption...
Database Upgrade
Ready


Microsoft SQL Server Management Studio
Quick Launch (Ctrl+Q)
File Edit View Project Tools Window Help
New Query
Execute
Object Explorer
Connect
SQL Server Import and Export Wizard
Choose a Data Source
Select the source from which to copy data.
Data source:
Microsoft Excel
Excel connection settings
Excel file path:
Browse...
Excel version:
Microsoft Excel 97-2003
First row has column names
Help
< Back
Next >
Finish >>|
Cancel
Open
Downloads › pyth ›
Search pyth
Organize
New folder
Quick access
Desktop
Downloads
Documents
Pictures
eng
Jojo Rabbit (2019
pyth
python
OneDrive
This PC
Name
Date modified
Type
.ipynb_checkpoints
27-10-2020 13:41
File folder
Demand
02-11-2020 14:37
Microsoft
File name:
Demand
Excel files (*.xls; *.xlsx; *.xlsm; *
Open
Cancel


Microsoft SQL Server Management Studio
Quick Launch (Ctrl+Q)
File Edit View Project Tools Window Help
New Query
Execute
Object Explorer
Connect
SQL Server Import and Export Wizard
Choose a Destination
Specify where to copy data to.
Destination:
SQL Server Native Client 11.0
Server name:
OPM
Authentication
Use Windows Authentication
Use SQL Server Authentication
User name:
Password:
Database:
SCM_DB
Refresh
New...
Help
< Back
Next >
Finish >>|
Cancel

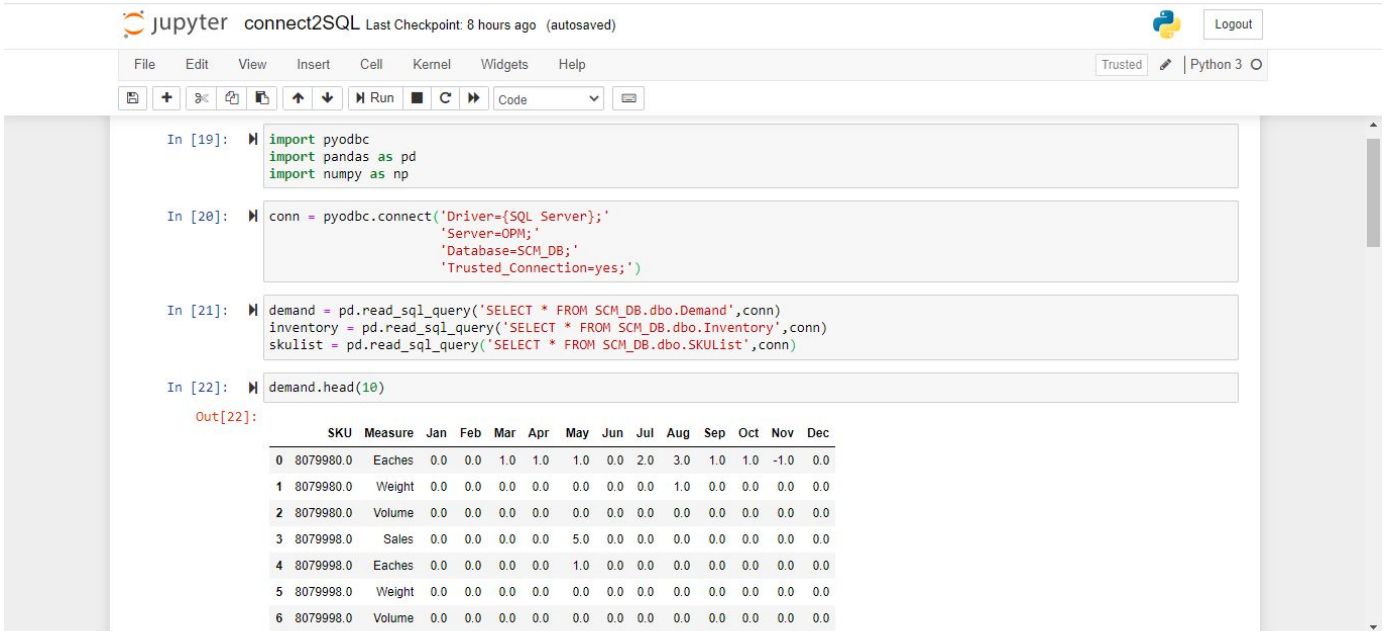




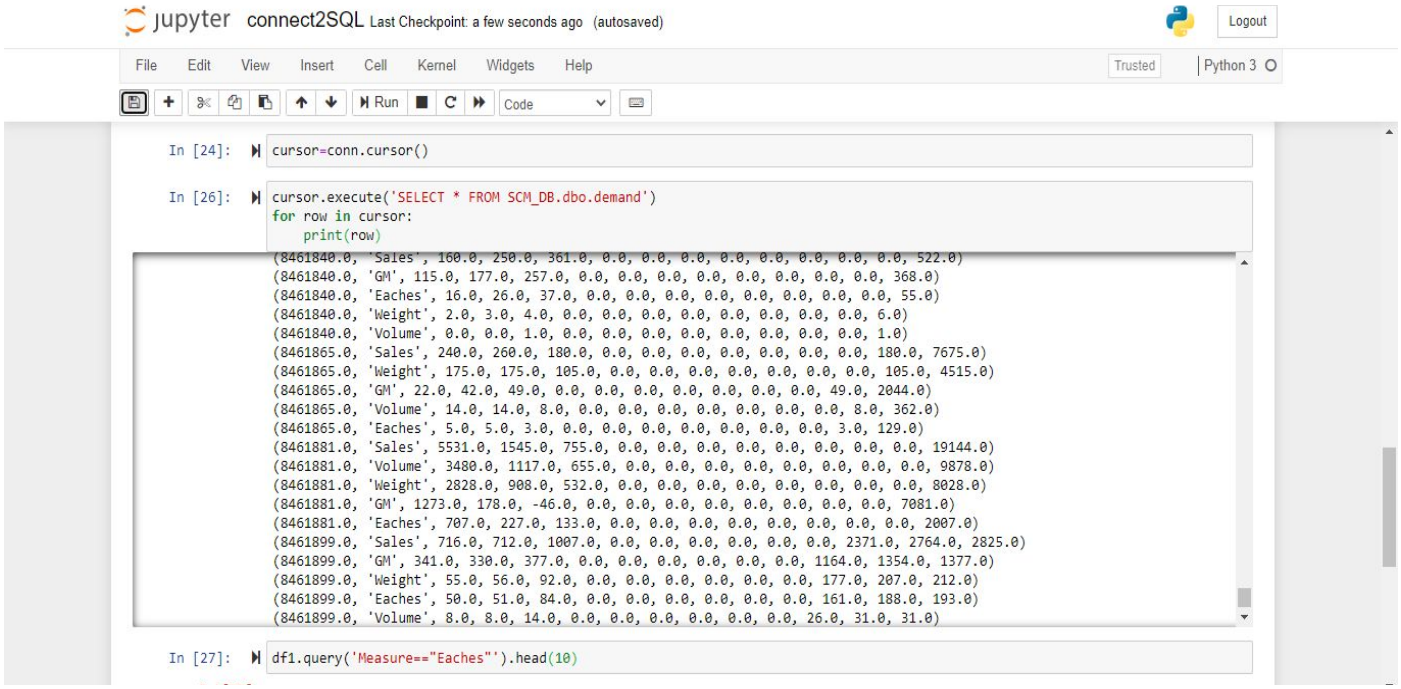


We can perform queries on the database by selecting ‘New Query’ from the toolbar.

We can connect Python to our sql server for performing Data Cleaning and analysis.



	SKU	Measure	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
0	8079980.0	Eaches	0.0	0.0	1.0	1.0	1.0	0.0	2.0	3.0	1.0	1.0	-1.0	0.0
1	8079980.0	Weight	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
2	8079980.0	Volume	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	8079998.0	Sales	0.0	0.0	0.0	0.0	5.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	8079998.0	Eaches	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	8079998.0	Weight	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6	8079998.0	Volume	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0



We can also add data frames created in Python to our SQL server.

jupyter connect2SQL Last Checkpoint: 11/08/2020 (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3

Run Code

```
In [27]: df2 = demand.query('Measure=="Eaches"').head(10)
```

```
In [28]: df2
```

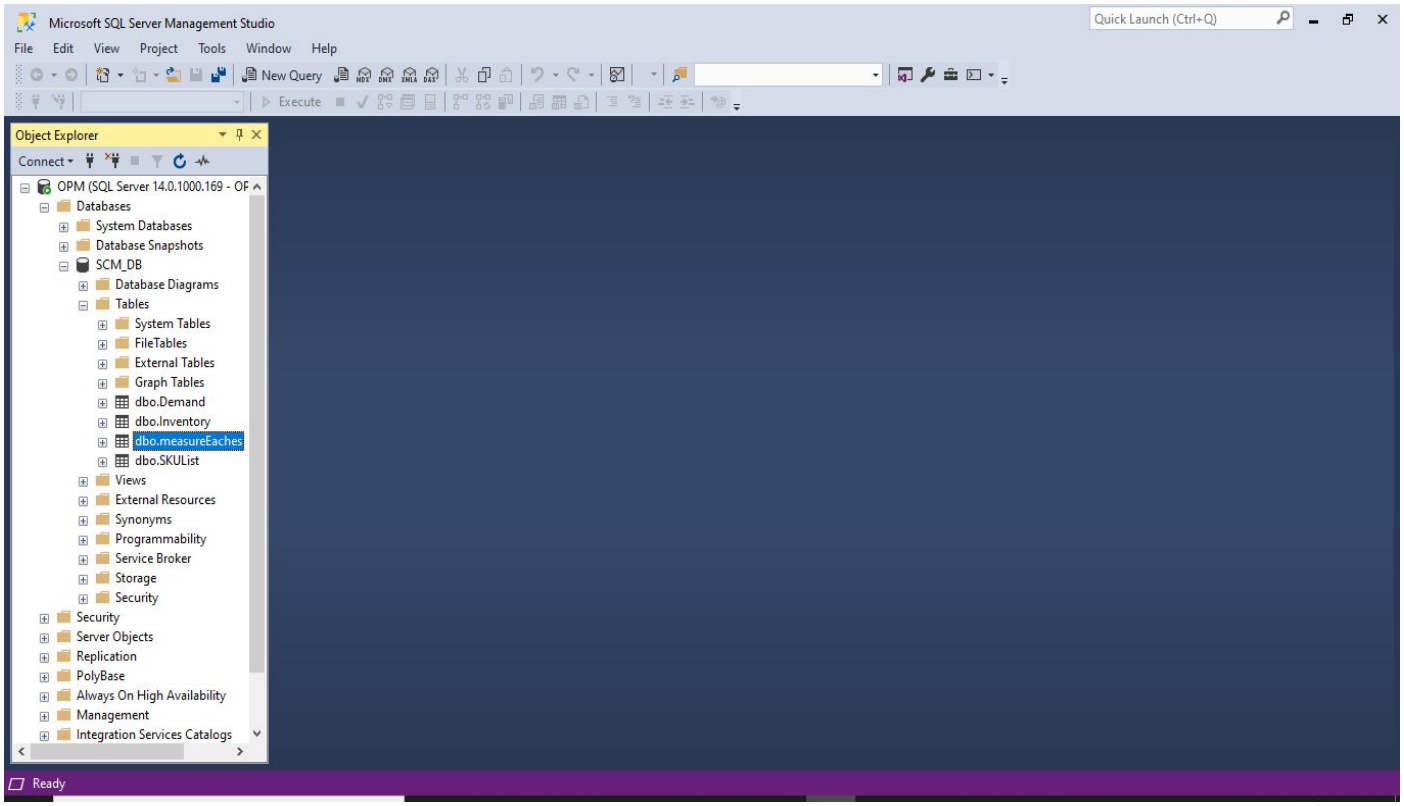
Out[28]:

	SKU	Measure	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
0	8079980.0	Eaches	0.0	0.0	1.0	1.0	1.0	0.0	2.0	3.0	1.0	1.0	-1.0	0.0
4	8079998.0	Eaches	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
9	8080004.0	Eaches	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
15	8080012.0	Eaches	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0
20	8080020.0	Eaches	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
25	8080194.0	Eaches	204.0	235.0	255.0	247.0	285.0	360.0	225.0	249.0	311.0	235.0	219.0	378.0
30	8080376.0	Eaches	197.0	232.0	246.0	9.0	3.0	1.0	1.0	0.0	-1.0	1.0	76.0	269.0
35	8080913.0	Eaches	36.0	53.0	89.0	55.0	84.0	89.0	75.0	81.0	89.0	41.0	47.0	60.0
41	8081457.0	Eaches	47.0	45.0	64.0	121.0	146.0	203.0	50.0	4.0	62.0	66.0	50.0	50.0
46	8081739.0	Eaches	7.0	14.0	16.0	18.0	9.0	9.0	9.0	10.0	13.0	11.0	10.0	12.0

```
In [29]: create_statement = fts.fast_to_sql(df2, "measureEaches", conn, if_exists="replace")
```

```
In [30]: conn.commit()
```

We can see that ‘measureEaches’ is added to the sql server.



Now we will perform data analysis on a given sample data using python.

Q1. Normalize the inventory, and demand dataset for these columns (i.e make it flat dataset. each data item listed below should a column header)

Demand Data: SKU, Month, Sales, GM, Eaches, Weight, Volume

Inventory Data: SKU, Month, Units

In [18]:
```
df1 = demand
```

In [19]:
```
df1.head(10)
```

Out[19]:

	SKU	Measure	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
0	110053	Sales	559	1738	755	769	519	919	519	709	779	589	519	699
1	110053	GM	333	790	452	457	309	547	309	422	459	350	309	416
2	110053	Eaches	56	234	75	77	52	92	52	71	79	59	52	70
3	110053	Weight	15	61	20	20	14	24	14	18	21	15	14	18
4	110053	Volume	1	5	1	1	1	2	1	1	2	1	1	1
5	110054	Sales	300	1434	777	450	380	557	300	380	450	350	330	526
6	110054	GM	178	648	457	267	226	331	178	226	267	208	196	312
7	110054	Eaches	30	194	79	45	38	56	30	38	45	35	33	53
8	110054	Weight	7	43	17	10	8	12	7	8	10	8	7	12
9	110054	Volume	1	4	2	1	1	1	1	1	1	1	1	1

In [20]:
```
# Making new dataframes with only the selected attributes
df_Sales = df1[df1.Measure.str.contains('Sales',case=False)]
df_GM = df1[df1.Measure.str.contains('GM',case=False)]
df_Eaches = df1[df1.Measure.str.contains('Eaches',case=False)]
df_Weight = df1[df1.Measure.str.contains('Weight',case=False)]
df_Volume = df1[df1.Measure.str.contains('Volume',case=False)]
```

In [21]:
```
# Making new dataframes using prev dataframe using melt
```

In [22]:
```
df_Eaches1 = df_Eaches.melt(id_vars=['SKU'], value_vars=['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul',
       'Aug', 'Sep', 'Oct', 'Nov', 'Dec'],var_name='Month', value_name='Eaches')
```

In [23]:
```
df_Sales1 = df_Sales.melt(id_vars=['SKU'], value_vars=['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul',
       'Aug', 'Sep', 'Oct', 'Nov', 'Dec'],var_name='Month', value_name='Sales')
```

In [24]:
```
df_GM1 = df_GM.melt(id_vars=['SKU'], value_vars=['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul',
       'Aug', 'Sep', 'Oct', 'Nov', 'Dec'],var_name='Month', value_name='GM')
```

In [25]:
```
df_Weight1 = df_Weight.melt(id_vars=['SKU'], value_vars=['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul',
       'Aug', 'Sep', 'Oct', 'Nov', 'Dec'],var_name='Month', value_name='Weight')
```

In [26]:
```
df_Volume1 = df_Volume.melt(id_vars=['SKU'], value_vars=['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul',
       'Aug', 'Sep', 'Oct', 'Nov', 'Dec'],var_name='Month', value_name='Volume')
```

In [33]:
```
# merging all the dataframes
df1_flat = pd.concat([df_Sales1,df_GM1,df_Eaches1,df_Weight1,df_Volume1],axis=1,join='inner')
df1_flat.head()
```

Out[33]:

	SKU	Month	Sales	SKU	Month	GM	SKU	Month	Eaches	SKU	Month	Weight	SKU	Month	Volume
0	110053	Jan	559	110053	Jan	333	110053	Jan	56	110053	Jan	15	110053	Jan	1
1	110054	Jan	300	110054	Jan	178	110054	Jan	30	110054	Jan	7	110054	Jan	1
2	110056	Jan	350	110056	Jan	197	110056	Jan	35	110056	Jan	10	110056	Jan	1
3	110124	Jan	881	110124	Jan	501	110124	Jan	98	110124	Jan	55	110124	Jan	1
4	110125	Jan	656	110125	Jan	377	110125	Jan	73	110125	Jan	42	110125	Jan	1

In [34]:
```
# Removing duplicate columns
df1_flat = df1_flat.loc[:,~df1_flat.columns.duplicated()]
df1_flat.head()
```

Out[34]:

	SKU	Month	Sales	GM	Eaches	Weight	Volume
0	110053	Jan	559	333	56	15	1
1	110054	Jan	300	178	30	7	1
2	110056	Jan	350	197	35	10	1
3	110124	Jan	881	501	98	55	1
4	110125	Jan	656	377	73	42	1

In [35]:
```
# Sorting values in terms of SKU
df1_flat.sort_values(by=['SKU'],inplace=True)
df1_flat.head()
```

Out[35]:

	SKU	Month	Sales	GM	Eaches	Weight	Volume
0	110053	Jan	559	333	56	15	1
45850	110053	Mar	755	452	75	20	1
68775	110053	Apr	769	457	77	20	1
91700	110053	May	519	309	52	14	1
114625	110053	Jun	919	547	92	24	2

In [36]:
```
df2 = inventory
df2.head()
```

Out[36]:

	SKU	UOM	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
0	110053	Eaches	489	186	278	279	506	440	293	303	130	537	518	100
1	110054	Eaches	247	61	53	263	23	244	131	16	96	76	163	234
2	110056	Eaches	302	44	282	340	121	311	168	286	300	93	208	255
3	110124	Eaches	209	367	520	336	475	375	173	205	396	286	310	481
4	110125	Eaches	352	310	402	155	342	453	380	472	327	414	381	144

Here, we selected the rows with attributes needed to be transposed in the final table. We used the melt function to transform the table to a flat dataset and then combined all such datasets. Then we removed duplicate columns to form a final flat dataset. We will repeat the same with other dataset.

In [37]:
```
# Doing the same with the inventory data
df2_flat = df2.melt(id_vars=['SKU'], value_vars=['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul',
       'Aug', 'Sep', 'Oct', 'Nov', 'Dec'],var_name='Month', value_name='Units')
df2_flat.sort_values(by=['SKU'],inplace=True)
df2_flat.head()
```

Out[37]:

	SKU	Month	Units
0	110053	Jan	489
45846	110053	Mar	278
68769	110053	Apr	279
91692	110053	May	506
114615	110053	Jun	440

In [38]:
```
# Naming the datasets
demand_flat = df1_flat
inventory_flat = df2_flat
```

In [39]:
```
demand_flat.to_csv('demandFlat.csv',index=False)
```

In [40]:
```
inventory_flat.to_csv('inventoryFlat.csv',index=False)
```

Now, we will rank categoryIDs and SupplierIDs in order of any of the 5 measures.
To do so we will take the flat dataset and group by SKU according to their mean. Then we will join the master table to it and group across all CategoryID and SupplierID by their sum and then sort in decreasing order.

Q2. Rank CategoryIDs and SupplierIDs in decreasing order of any of the five measures (Sales, GM, Eaches, Weight and Volume)

In [38]:
```
skulist.head()
```

Out[38]:

	SKU	CategoryID	SKU.1	SupplierID
0	110053	460	Active	3452
1	110054	460	Active	3452
2	110056	460	Active	3452
3	110124	460	Active	3452
4	110125	460	Active	3452

In [39]:
```
df1_flat.head()
```

Out[39]:

	SKU	Month	Sales	GM	Eaches	Weight	Volume
0	110053	Jan	559	333	56	15	1
45850	110053	Mar	755	452	75	20	1
68775	110053	Apr	769	457	77	20	1
91700	110053	May	519	309	52	14	1
114625	110053	Jun	919	547	92	24	2

In [40]:
```
# Combining/Grouping all the SKUs using their mean
df1_avg = df1_flat.groupby(by=['SKU']).agg('mean')
df1_avg.head()
```

Out[40]:

SKU	Sales	GM	Eaches	Weight	Volume
110053	756.083333	429.416667	80.750000	21.166667	1.500000
110054	519.500000	291.166667	56.333333	12.416667	1.333333
110056	471.166667	254.916667	49.666667	14.000000	1.000000
110124	1285.416667	745.000000	144.916667	81.083333	1.000000
110125	905.666667	535.416667	102.166667	59.250000	0.916667

In [41]:
```
print(skulist.shape)
print(df1_avg.shape)
```

```
(22923, 4)
(22923, 5)
```

In [42]:
```
# Combining the two datasets ie SKU_master(master) on df1_avg(transaction)
demand_SKU = pd.merge(df1_avg,skulist, how='left', on=['SKU'])
demand_SKU.head()
```

Out[42]:

	SKU	Sales	GM	Eaches	Weight	Volume	CategoryID	SKU.1	SupplierID
0	110053	756.083333	429.416667	80.750000	21.166667	1.500000	460	Active	3452
1	110054	519.500000	291.166667	56.333333	12.416667	1.333333	460	Active	3452
2	110056	471.166667	254.916667	49.666667	14.000000	1.000000	460	Active	3452
3	110124	1285.416667	745.000000	144.916667	81.083333	1.000000	460	Active	3452
4	110125	905.666667	535.416667	102.166667	59.250000	0.916667	460	Active	3452

In [45]:
```
# Using only 'active' status for further calculations
active_demandSKU = demand_SKU[demand_SKU['SKU.1']=='Active']
active_demandSKU.shape
```

Out[45]: (9685, 9)

In [46]:
```
rank = active_demandSKU
rank.head()
```

Out[46]:

	SKU	Sales	GM	Eaches	Weight	Volume	CategoryID	SKU.1	SupplierID
0	110053	756.083333	429.416667	80.750000	21.166667	1.500000	460	Active	3452
1	110054	519.500000	291.166667	56.333333	12.416667	1.333333	460	Active	3452
2	110056	471.166667	254.916667	49.666667	14.000000	1.000000	460	Active	3452
3	110124	1285.416667	745.000000	144.916667	81.083333	1.000000	460	Active	3452
4	110125	905.666667	535.416667	102.166667	59.250000	0.916667	460	Active	3452

In [47]:
```
# Grouping different categoryID and SupplierID by taking their mean
cat = rank[['CategoryID','Sales']].groupby('CategoryID').agg('mean')
sup = rank[['SupplierID','Sales']].groupby('SupplierID').agg('mean')
```

In [48]:
```
# Ranking different CategoryID and SupplierID
cat['rank_CategoryID_Sales'] = cat['Sales'].rank(method='min',ascending=False)
sup['rank_SupplierID_Sales'] = sup['Sales'].rank(method='min',ascending=False)
```

```
In [51]: # Converting these into dictionaries
cat_dic = cat[['rank_CategoryID_Sales']].to_dict()
sup_dic = sup[['rank_SupplierID_Sales']].to_dict()

In [52]: # Using the dictionaries to apply on the rank dataframe
rank['rank_CategoryID_Sales'] = rank['CategoryID'].apply(lambda x:int(cat_dic['rank_CategoryID_Sales'][x]))
rank['rank_SupplierID_Sales'] = rank['SupplierID'].apply(lambda x:int(sup_dic['rank_SupplierID_Sales'][x]))

<ipython-input-52-93b91f955ce3>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  rank['rank_CategoryID_Sales'] = rank['CategoryID'].apply(lambda x:int(cat_dic['rank_CategoryID_Sales'][x]))
<ipython-input-52-93b91f955ce3>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  rank['rank_SupplierID_Sales'] = rank['SupplierID'].apply(lambda x:int(sup_dic['rank_SupplierID_Sales'][x]))

In [53]: rank.head()
```

Out[53]:

	SKU	Sales	GM	Eaches	Weight	Volume	CategoryID	SKU.1	SupplierID	rank_CategoryID_Sales	rank_SupplierID_Sales
0	110053	756.083333	429.416667	80.750000	21.166667	1.500000	460	Active	3452	8	201
1	110054	519.500000	291.166667	56.333333	12.416667	1.333333	460	Active	3452	8	201
2	110056	471.166667	254.916667	49.666667	14.000000	1.000000	460	Active	3452	8	201
3	110124	1285.416667	745.000000	144.916667	81.083333	1.000000	460	Active	3452	8	201
4	110125	905.666667	535.416667	102.166667	59.250000	0.916667	460	Active	3452	8	201

```
In [57]: rank.to_csv('rank.csv',index=False)
```

Now, we will find the total average of inventory by category code and supplier number in terms of GM,Eaches etc. Again we will start by taking mean across various SKU so that we get avg of units for the year. Then we will combine the master table to it. Then we will select only the active SKUs and group by categoryID and supplierID by their mean which gives the total avg of inventory by category and supplier code across all SKU and months.

Q3. Calculate total average inventory by Category code and SupplierNumber in terms of GM, Eaches, Weight and Volume

```
In [59]: inventory_flat.head()
```

Out[59]:

	SKU	Month	Units
0	110053	Jan	489
45846	110053	Mar	278
68769	110053	Apr	279
91692	110053	May	506
114615	110053	Jun	440

```
In [61]: # Taking avg/mean of SKU's across all the months
df2_avg = inventory_flat.groupby(by=['SKU']).agg('mean')
df2_avg.head()
```

Out[61]:

	Units
SKU	
110053	338.250000
110054	133.916667
110056	225.833333
110124	344.416667
110125	344.333333

```
In [62]: print(df2_avg.shape)
print(skulist.shape)

(22923, 1)
(22923, 4)
```

```
In [63]: # combining the Sku_master(master) and df2_avg(transaction)
inv_SKU = pd.merge(df2_avg,skulist, how='left', on=['SKU'])
inv_SKU.head()
```

Out[63]:

	SKU	Units	CategoryID	SKU.1	SupplierID
0	110053	338.250000	460	Active	3452
1	110054	133.916667	460	Active	3452
2	110056	225.833333	460	Active	3452
3	110124	344.416667	460	Active	3452
4	110125	344.333333	460	Active	3452

```
In [64]: # Using only 'active' status for further calculations
active_invSKU = inv_SKU[inv_SKU['SKU.1']=='Active']
active_invSKU.shape
```

Out[64]: (9685, 5)

```
In [65]: # Grouping different categoryID and SupplierID by taking their mean
cat_inv = active_invSKU[['CategoryID','Units']].groupby('CategoryID').agg('sum')
sup_inv = active_invSKU[['SupplierID','Units']].groupby('SupplierID').agg('sum')
```

```
In [70]: # Combing categoryID, supplierID across their mean
cat_code = rank[['CategoryID','GM','Eaches','Weight','Volume']].groupby('CategoryID').agg('mean')
sup_num = rank[['SupplierID','GM','Eaches','Weight','Volume']].groupby('SupplierID').agg('mean')

In [71]: cat_code.head(10)
```

Out[71]:

CategoryID	GM	Eaches	Weight	Volume
200	610.228070	147.307261	75.762671	4.515107
310	2390.615843	295.291287	889.844603	49.125596
320	1825.168038	187.238169	901.318244	36.650377
330	2329.447833	315.921653	1559.780974	80.633870
340	1350.319793	273.134309	387.831641	38.402230
350	1006.599108	119.277435	303.866255	20.071331
360	1201.273158	175.252559	234.231576	10.146963
370	630.744175	88.673346	155.716449	3.235011
380	1657.354881	604.805420	508.767385	44.459272
390	1621.021667	204.414167	1017.075833	72.632500

```
In [72]: sup_num.head(10)
```

Out[72]:

SupplierID	GM	Eaches	Weight	Volume
46	449.235026	2536.455078	98.678385	2.810547
74	1600.949074	165.032407	151.407407	3.402778
95	997.949805	219.083333	183.299951	12.721735

Next, we need to calculate the weighted average of Sales,GM, etc per unit from demand data and weights are taken as Eaches. Also weighted average for CategoryID and SupplierID.

We took products of Eaches with Sales,GM,Weight,Volume for every SKU. Then we added those columns grouped by SKUs and took a weighted average by dividing them by the number of Eaches for each SKU. Now we can combine the master table to it and again calculate the weighted average by taking Eaches as weights but grouping them by CategoryID and SupplierID sum.

Q4. Calculate the weighted average of Sales, GM, Eaches, Weight and Volume per Unit from Demand Data. Apply these values as new columns in SKUMaster data. Calculate same values by CategoryID and SupplierID and report them out.

In [73]:
```
demand_flat.head(10)
```

Out[73]:

	SKU	Month	Sales	GM	Eaches	Weight	Volume
0	110053	Jan	559	333	56	15	1
45850	110053	Mar	755	452	75	20	1
68775	110053	Apr	769	457	77	20	1
91700	110053	May	519	309	52	14	1
114625	110053	Jun	919	547	92	24	2
137550	110053	Jul	519	309	52	14	1
160475	110053	Aug	709	422	71	18	1
183400	110053	Sep	779	459	79	21	2
206325	110053	Oct	589	350	59	15	1
229250	110053	Nov	519	309	52	14	1

In [74]:
```
# Using eaches as weights
for i in demand_flat:
    if i not in ['SKU','Month','Eaches']:
        demand_flat[i+'eaches'] = demand_flat[i]*demand_flat['Eaches']
```

In [77]:
```
# Groupby across SKU using sum
df1 = demand_flat[['SKU','Eaches','Saleseaches', 'GMeaches', 'Weighteaches', 'Volumeeaches']]
df1 = df1.groupby(by='SKU').agg('sum')
```

In [78]:
```
# Dividing the sum with corresponding Eaches
for i in df1:
    if i not in ['Eaches']:
        df1[i] = df1[i]/df1['Eaches']
df1.head(10)
```

Out[78]:

SKU	Eaches	Saleseaches	GMeaches	Weighteaches	Volumeeaches
110053	969	944.176471	502.701754	28.511868	2.142415
110054	676	753.208580	388.344675	19.869822	1.977811
110056	596	569.600671	292.765101	17.904362	1.179530
110124	1739	1332.460035	771.633698	83.979298	1.000000
110125	1226	958.303426	567.075856	62.690865	0.948613
110127	10	33.800000	20.800000	0.800000	0.000000
110132	947	1003.214361	1003.214361	0.000000	0.000000
110239	1693	53142.528057	18617.109864	5375.851742	203.130538
110742	1490	37037.136913	11814.757047	4140.879195	156.090604
111120	2	1.000000	1.000000	1.000000	0.000000

In [79]:
```
# Using only SKU,Sales,GM,Eaches,Weight,Volume
dem_flat = demand_flat.drop(columns=['Saleseaches', 'GMeaches', 'Weighteaches', 'Volumeeaches'])
```

In [80]:
```
# Grouping by SKU by taking mean across SKU
dem_flat.groupby(by='SKU').agg('mean').head()
```

Out[80]:

SKU	Sales	GM	Eaches	Weight	Volume
110053	756.083333	429.416667	80.750000	21.166667	1.500000
110054	519.500000	291.166667	56.333333	12.416667	1.333333
110056	471.166667	254.916667	49.666667	14.000000	1.000000
110124	1285.416667	745.000000	144.916667	81.083333	1.000000
110125	905.666667	535.416667	102.166667	59.250000	0.916667

In [81]:
```
# Merging the two datasets
weighted_SKU = pd.merge(skulist,df1, how='left', on=['SKU'])
weighted_SKU.head(10)
```

Out[81]:

	SKU	CategoryID	SKU.1	SupplierID	Eaches	Saleseaches	GMeaches	Weighteaches	Volumeeaches
0	110053	460	Active	3452	969	944.176471	502.701754	28.511868	2.142415
1	110054	460	Active	3452	676	753.208580	388.344675	19.869822	1.977811
2	110056	460	Active	3452	596	569.600671	292.765101	17.904362	1.179530
3	110124	460	Active	3452	1739	1332.460035	771.633698	83.979298	1.000000
4	110125	460	Active	3452	1226	958.303426	567.075856	62.690865	0.948613
5	110127	700	Pre Active	2075	10	33.800000	20.800000	0.800000	0.000000
6	110132	460	Pre Active	2044	947	1003.214361	1003.214361	0.000000	0.000000
7	110239	460	Active	52070	1693	53142.528057	18617.109864	5375.851742	203.130538

In [82]:
```
# Using only the active SKU's
df2 = weighted_SKU[weighted_SKU['SKU.1']=='Active']
df2.head()
```

Out[82]:

	SKU	CategoryID	SKU.1	SupplierID	Eaches	Saleseaches	GMeaches	Weighteaches	Volumeeaches
0	110053	460	Active	3452	969	944.176471	502.701754	28.511868	2.142415
1	110054	460	Active	3452	676	753.208580	388.344675	19.869822	1.977811
2	110056	460	Active	3452	596	569.600671	292.765101	17.904362	1.179530
3	110124	460	Active	3452	1739	1332.460035	771.633698	83.979298	1.000000
4	110125	460	Active	3452	1226	958.303426	567.075856	62.690865	0.948613

In [83]:
```
# Using eaches as weights
for i in df2:
    if i in ['Saleseaches', 'GMeaches', 'Weighteaches', 'Volumeeaches']:
        df2[i] = df2[i]*df2['Eaches']
df2.head()
```

```
<ipython-input-83-593538016ab2>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df2[i] = df2[i]*df2['Eaches']
```

Out[83]:

	SKU	CategoryID	SKU.1	SupplierID	Eaches	Saleseaches	GMeaches	Weighteaches	Volumeeaches
0	110053	460	Active	3452	969	914907.0	487118.0	27628.0	2076.0
1	110054	460	Active	3452	676	509169.0	262521.0	13432.0	1337.0
2	110056	460	Active	3452	596	339482.0	174488.0	10671.0	703.0

1	110054	460	Active	3452	676	509169.0	262521.0	13432.0	1337.0
2	110056	460	Active	3452	596	339482.0	174488.0	10671.0	703.0
3	110124	460	Active	3452	1739	2317148.0	1341871.0	146040.0	1739.0
4	110125	460	Active	3452	1226	1174880.0	695235.0	76859.0	1163.0

In [84]:
```
df2_cat = df2[['CategoryID','Eaches','Saleseaches', 'GMeaches', 'Weighteaches', 'Volumeeaches']]
df2_cat = df2_cat.groupby(by='CategoryID').agg('sum')
```

In [85]:
```
# Getting weighted average for categoryID
for i in df2_cat:
    if i not in ['Eaches']:
        df2_cat[i] = df2_cat[i]/df2_cat['Eaches']
df2_cat.head(10)
```

Out[85]:

CategoryID	Eaches	Saleseaches	GMeaches	Weighteaches	Volumeeaches
200	604549	2540.414405	1533.072150	216.784090	11.567780
310	2724948	8981.086723	5290.272293	1837.811168	104.002417
320	1091973	7884.876273	4130.908543	4537.238315	102.977353
330	3294431	11475.119542	5160.502449	4616.106654	206.421838
340	2743361	7852.370186	3862.378811	1383.145552	86.611005
350	347813	11729.349096	5169.041097	3042.279932	118.897192
360	4109322	5039.225849	2757.426775	848.774732	48.983352
370	1141758	8759.666744	4724.023743	1688.268678	20.156258
380	7410076	3467.920206	2103.438732	762.839146	43.215776
390	490594	5158.023798	3400.148773	1008.237072	53.458940

In [87]:
```
df2_sup = df2[['SupplierID','Eaches','Saleseaches', 'GMeaches', 'Weighteaches', 'Volumeeaches']]
df2_sup = df2_sup.groupby(by='SupplierID').agg('sum')
```

In [88]:
```
# Getting weighted average for supplierID
for i in df2_sup:
    if i not in ['Eaches']:
        df2_sup[i] = df2_sup[i]/df2_sup['Eaches']
df2_sup.head(10)
```

Out[88]:

SupplierID	Eaches	Saleseaches	GMeaches	Weighteaches	Volumeeaches
46	3895995	831.755790	676.701794	114.234040	1.310853
74	35647	4767.806043	2051.387438	208.849272	4.708166
95	899118	5904.296707	4557.009704	659.199541	31.374431
113	1351	6547.615100	2604.538860	816.903775	23.472243
122	1097	1238.844120	696.031905	48.919781	1.011851
198	26410	6266.605831	2766.937486	957.955547	26.118402
231	86270	5723.697890	3270.925014	337.833812	18.788976
263	328570	12659.382180	6509.238680	1214.146888	109.314822
321	3568	11034.986547	4408.860987	3701.721413	162.214966
327	4119	16182.703812	7882.421947	1223.297888	48.277980

Here, we will find SKUs making the top 80perc of sales in each categoryID. We will sort the CategoryID in terms of Sales in descending order. Then we will calculate cumulative sum and at last find the cumulative percentage. Then we can filter the rows having cumulative percentage less than 80 thus giving us the desired output.

Q5. In each CategoryID, find the SKUs which make-up top 80% of volume in terms of RetailPrice. E.g. If there are 10 CategoryID, in each of the categoryID, find the SKUs which make up top 80% in terms of Sales price.

In [89]:
```
rank.head(10)
```

Out[89]:

	SKU	Sales	GM	Eaches	Weight	Volume	CategoryID	SKU.1	SupplierID	rank_CategoryID_Sales	rank_SupplierID_Sales
0	110053	756.083333	429.416667	80.750000	21.166667	1.500000	460	Active	3452	8	201
1	110054	519.500000	291.166667	56.333333	12.416667	1.333333	460	Active	3452	8	201
2	110056	471.166667	254.916667	49.666667	14.000000	1.000000	460	Active	3452	8	201
3	110124	1285.416667	745.000000	144.916667	81.083333	1.000000	460	Active	3452	8	201
4	110125	905.666667	535.416667	102.166667	59.250000	0.916667	460	Active	3452	8	201
7	110239	23502.250000	9117.083333	141.083333	2237.750000	84.500000	460	Active	52070	8	181
8	110742	19176.166667	6986.083333	124.166667	1999.333333	75.250000	460	Active	52070	8	181
10	140005	2741.916667	1716.250000	102.583333	139.416667	85.583333	460	Active	3452	8	201
12	140013	2025.416667	1207.500000	217.833333	43.583333	1.500000	460	Active	52070	8	181
13	140014	1115.916667	630.500000	88.000000	63.250000	1.333333	460	Active	52070	8	181

In [90]:
```
# Using Sku,categoryID,Sales column and creating a column containing cumulative sum of sales
q = rank[['SKU','CategoryID','Sales']]
q = q.sort_values(['CategoryID','Sales'],ascending=False)
q['cum'] = q.groupby(['CategoryID'])['Sales'].apply(lambda x: x.cumsum())
```

In [91]:
```
cat_sum_dic = dict(q.groupby(['CategoryID'])['Sales'].sum())
```

In [92]:
```
# Converting cumulative sum into cumulative percentage
q['cat_sum'] = q['CategoryID'].apply(lambda x: cat_sum_dic[x])
q = q.assign(cum_perc=lambda x: 100*x['cum']/x['cat_sum'])
```

In [93]:
```
q.head()
```

Out[93]:

	SKU	CategoryID	Sales	cum	cat_sum	cum_perc
7120	8164899	700	1499.666667	1499.666667	12253.0	12.239180
9332	8244386	700	1337.833333	2837.500000	12253.0	23.157594
5840	8099053	700	1131.000000	3968.500000	12253.0	32.387987
5842	8099079	700	923.416667	4891.916667	12253.0	39.924236
5843	8099087	700	758.750000	5650.666667	12253.0	46.116597

In [94]:
```
# Selecting the categoryID which make upto 80%
SKU_80 = q.query('cum_perc<=80')[['SKU','CategoryID','Sales']]
```

In [95]:
```
SKU_80.head(30)
```

Out[95]:

	SKU	CategoryID	Sales
7120	8164899	700	1499.666667
9332	8244386	700	1337.833333
5840	8099053	700	1131.000000
5842	8099079	700	923.416667
5843	8099087	700	758.750000
407	1202357	700	585.000000
12706	8323818	700	356.250000

Next, we need to analyse if there is seasonality among categoryID. Here we will take the initial demand dataset and choose only the Eaches rows. We can then find count,mean,std,min,quantile etc for each month all over and we can plot box-plot for different months sales and bar-graph for statistics calculated for all the months .

Q6. Analyse if there is any seasonality in overall volumes over the year; and also seasonality among CategoryID.

In [96]:
```
demand_flat.head(12)
```

Out[96]:

	SKU	Month	Sales	GM	Eaches	Weight	Volume	Saleseaches	GMeaches	Weighteaches	Volumeeaches
0	110053	Jan	559	333	56	15	1	31304	18648	840	56
45850	110053	Mar	755	452	75	20	1	56625	33900	1500	75
68775	110053	Apr	769	457	77	20	1	59213	35189	1540	77
91700	110053	May	519	309	52	14	1	26988	16068	728	52
114625	110053	Jun	919	547	92	24	2	84548	50324	2208	184
137550	110053	Jul	519	309	52	14	1	26988	16068	728	52
160475	110053	Aug	709	422	71	18	1	50339	29962	1278	71
183400	110053	Sep	779	459	79	21	2	61541	36261	1659	158
206325	110053	Oct	589	350	59	15	1	34751	20650	885	59
229250	110053	Nov	519	309	52	14	1	26988	16068	728	52
252175	110053	Dec	699	416	70	18	1	48930	29120	1260	70
22925	110053	Feb	1738	790	234	61	5	406692	184860	14274	1170

In [97]:
```
df_seas = demand[demand['Measure']=='Eaches']
df_seas.head(10)
```

Out[97]:

	SKU	Measure	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
2	110053	Eaches	56	234	75	77	52	92	52	71	79	59	52	70
7	110054	Eaches	30	194	79	45	38	56	30	38	45	35	33	53
12	110056	Eaches	35	132	37	39	49	55	25	42	59	39	46	38
17	110124	Eaches	98	111	173	137	141	149	134	135	179	147	133	202
22	110125	Eaches	73	63	142	107	110	136	81	106	119	81	83	125
27	110127	Eaches	0	0	0	0	0	0	1	2	4	1	0	2
32	110132	Eaches	110	83	46	90	34	45	70	71	94	152	65	87
38	110239	Eaches	22	345	228	29	50	524	43	30	25	29	22	346
43	110742	Eaches	27	37	120	27	15	35	17	275	49	209	417	262
45	111120	Eaches	0	0	0	0	0	0	1	0	0	1	0	0

In [98]:
```
# Analysing only the active SKU's
seas_SKU = pd.merge(skulist,df_seas, how='left', on=['SKU'])
seas_SKU = seas_SKU[seas_SKU['SKU.1']=='Active']
seas_SKU.head(10)
```

Out[98]:

	SKU	CategoryID	SKU.1	SupplierID	Measure	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
0	110053	460	Active	3452	Eaches	56	234	75	77	52	92	52	71	79	59	52	70
1	110054	460	Active	3452	Eaches	30	194	79	45	38	56	30	38	45	35	33	53
2	110056	460	Active	3452	Eaches	35	132	37	39	49	55	25	42	59	39	46	38
3	110124	460	Active	3452	Eaches	98	111	173	137	141	149	134	135	179	147	133	202
4	110125	460	Active	3452	Eaches	73	63	142	107	110	136	81	106	119	81	83	125
7	110239	460	Active	52070	Eaches	22	345	228	29	50	524	43	30	25	29	22	346
8	110742	460	Active	52070	Eaches	27	37	120	27	15	35	17	275	49	209	417	262
10	140005	460	Active	3452	Eaches	81	47	90	92	96	84	242	87	83	68	64	197
12	140013	460	Active	52070	Eaches	290	387	197	54	144	212	133	183	544	164	129	177
13	140014	460	Active	52070	Eaches	34	179	145	116	72	106	78	60	66	60	56	84

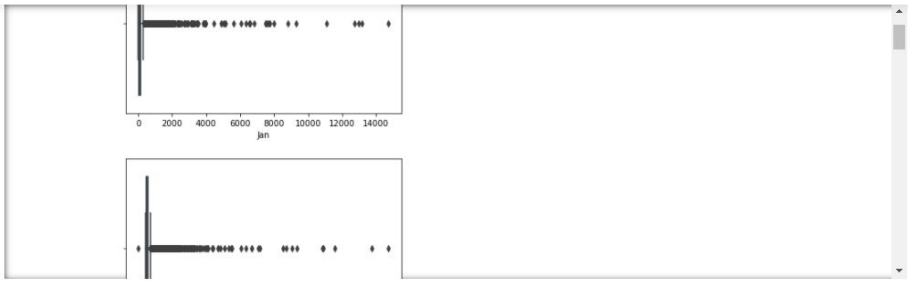
In [100]:
```
# Statistics of various months
seas_SKU[['Jan', 'Feb',
       'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']].describe()
```

Out[100]:

	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	
count	9685.000000	9685.000000	9685.000000	9685.000000	9685.000000	9685.000000	9685.000000	9685.000000	9685.000000	9685.000000	96
mean	162.217243	189.428807	286.579350	229.853898	248.207434	315.852039	224.089623	219.044708	276.417037	226.743934	2
std	512.122939	585.004986	964.889562	795.255023	898.320261	1218.560409	823.782540	764.925582	913.192902	737.142858	6
min	-9.000000	-562.000000	-11.000000	-11.000000	-15.000000	-47.000000	-28.000000	-3.000000	-4.000000	-9.000000	
25%	24.000000	27.000000	40.000000	26.000000	28.000000	34.000000	25.000000	26.000000	36.000000	29.000000	
50%	58.000000	66.000000	95.000000	75.000000	81.000000	99.000000	71.000000	72.000000	94.000000	75.000000	
75%	138.000000	158.000000	232.000000	190.000000	204.000000	255.000000	182.000000	182.000000	235.000000	192.000000	1
max	14737.000000	17907.000000	33101.000000	22458.000000	27152.000000	55957.000000	31289.000000	22886.000000	28001.000000	22046.000000	206

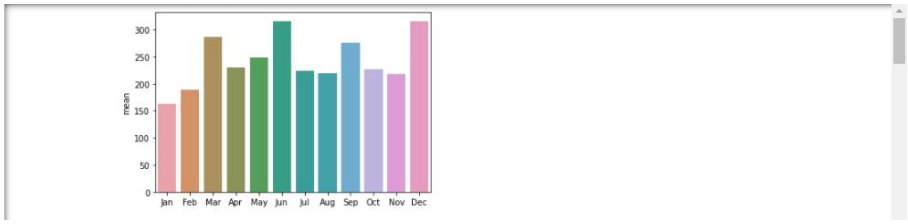
In [101]:
```
stat_seas = seas_SKU[['Jan', 'Feb',
       'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']].describe()
```

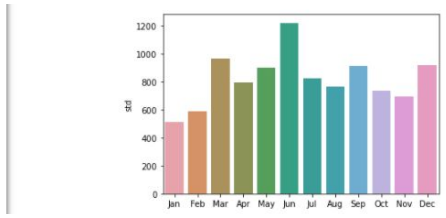
In [102]:
```
# Plottingbox plot of eaches for every month
import seaborn as sns
import matplotlib.pyplot as plt
for i in seas_SKU:
    if i not in ['SKU','CategoryID','SKU.1','SupplierID','Measure']:
        sns.boxplot(x=seas_SKU[i])
        plt.show()
```

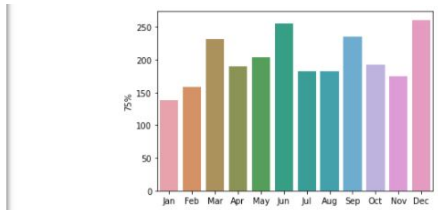


In [103]:
```
stat_seas = stat_seas.T
```

In [104]:
```
# Plotting mean,std,min,max etc across various months
for i in stat_seas:
    if i not in ['count']:
        x = sns.barplot(x=stat_seas.index, y=stat_seas[i], data=stat_seas, saturation=.5)
        plt.show()
```







Finally, we will calculate COV of demand for each product. For this we will take only SKU,Eaches columns in our table. We will calculate standard deviation and mean for each SKU and we will divide them to get COV. Then we will sort COV and divide the data into 3 categories Low, Med, High with 50%,30%,20% distribution respectively.

Q7. Calculate COV of Demand for each product

COV (in units)=(Std.Dev of Units)/(Avg of Units).

Create three clusters (Low, Med, High) in increasing order of COV with top 50% of units in Low, Next 30% in medium and bottom 20% in high.

In [41]:
```
demand_flat.head(10)
```

Out[41]:

	SKU	Month	Sales	GM	Eaches	Weight	Volume
0	110053	Jan	559	333	56	15	1
45850	110053	Mar	755	452	75	20	1
68775	110053	Apr	769	457	77	20	1
91700	110053	May	519	309	52	14	1
114625	110053	Jun	919	547	92	24	2
137550	110053	Jul	519	309	52	14	1
160475	110053	Aug	709	422	71	18	1
183400	110053	Sep	779	459	79	21	2
206325	110053	Oct	589	350	59	15	1
229250	110053	Nov	519	309	52	14	1

In [42]:
```
df3 = demand_flat[['SKU','Eaches']]
```

In [43]:
```
# Taking standard deviation across each SKU
df4 = df3.groupby(['SKU']).std()
```

In [44]:
```
# Taking mean across each SKU
df4['avg'] = df3.groupby(['SKU']).mean()
```

In [45]:
```
# Calculating COV
df4['COV'] = df4['Eaches']/df4['avg']
df4.head()
```

Out[45]:

SKU	Eaches	avg	COV
110053	49.943832	80.750000	0.618499
110054	45.519892	56.333333	0.808045
110056	27.493250	49.666667	0.553555
110124	28.633922	144.916667	0.197589
110125	25.732305	102.166667	0.251866

In [46]:
```
COV_sorted = df4.sort_values(['COV'])
COV_sorted.shape
```

Out[46]: (22923, 3)

In [49]:
```
COV_sorted.to_csv('COV.csv')
```

In [52]:
```
# Top 50% ie Low
COV_low = COV_sorted[:COV_sorted.shape[0]//2]
COV_low.shape
```

Out[52]: (11461, 3)

In [53]:
```
# Next 30% ie Medium
COV_med = COV_sorted[COV_sorted.shape[0]//2:COV_sorted.shape[0]//2+COV_sorted.shape[0]*3//10]
COV_med.shape
```

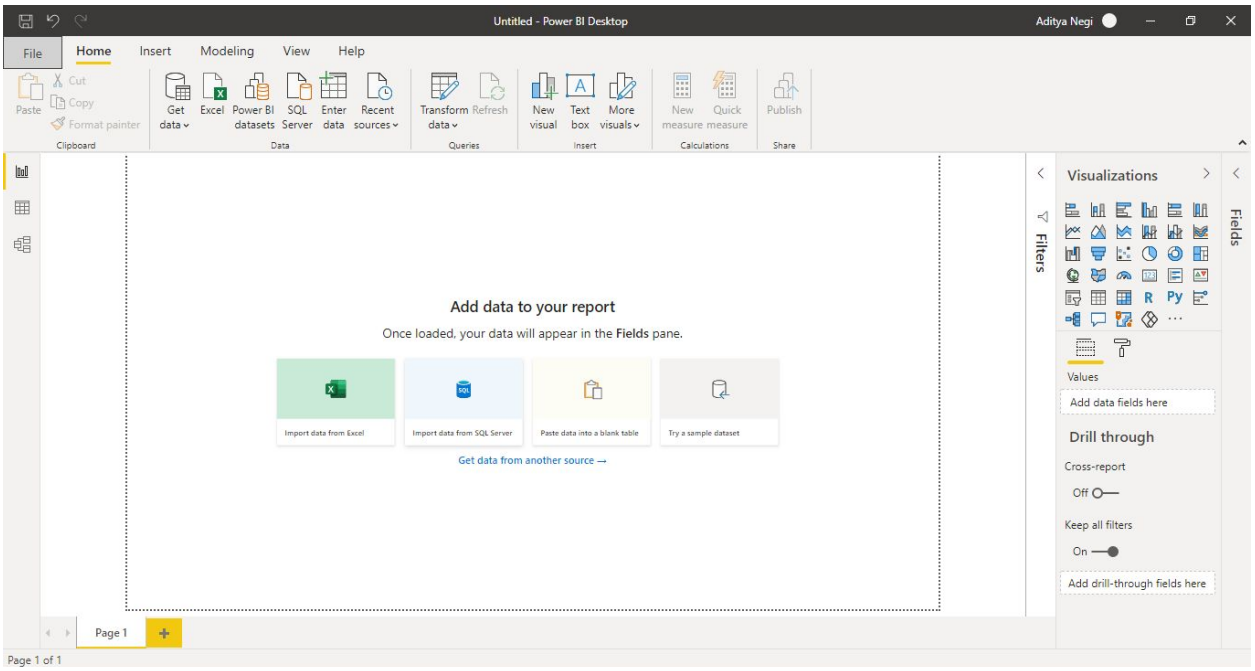
Out[53]: (6876, 3)

In [54]:
```
# Next 20% ie High
COV_high = COV_sorted[COV_sorted.shape[0]//2+COV_sorted.shape[0]*3//10:]
COV_high.shape
```

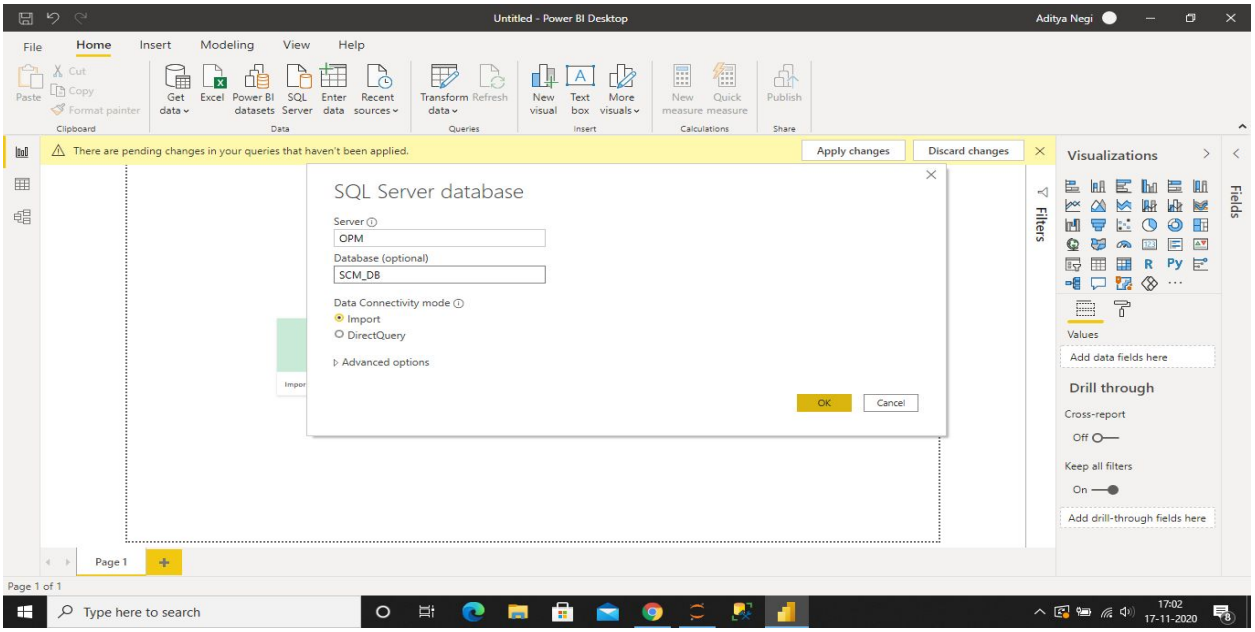
Out[54]: (4586, 3)

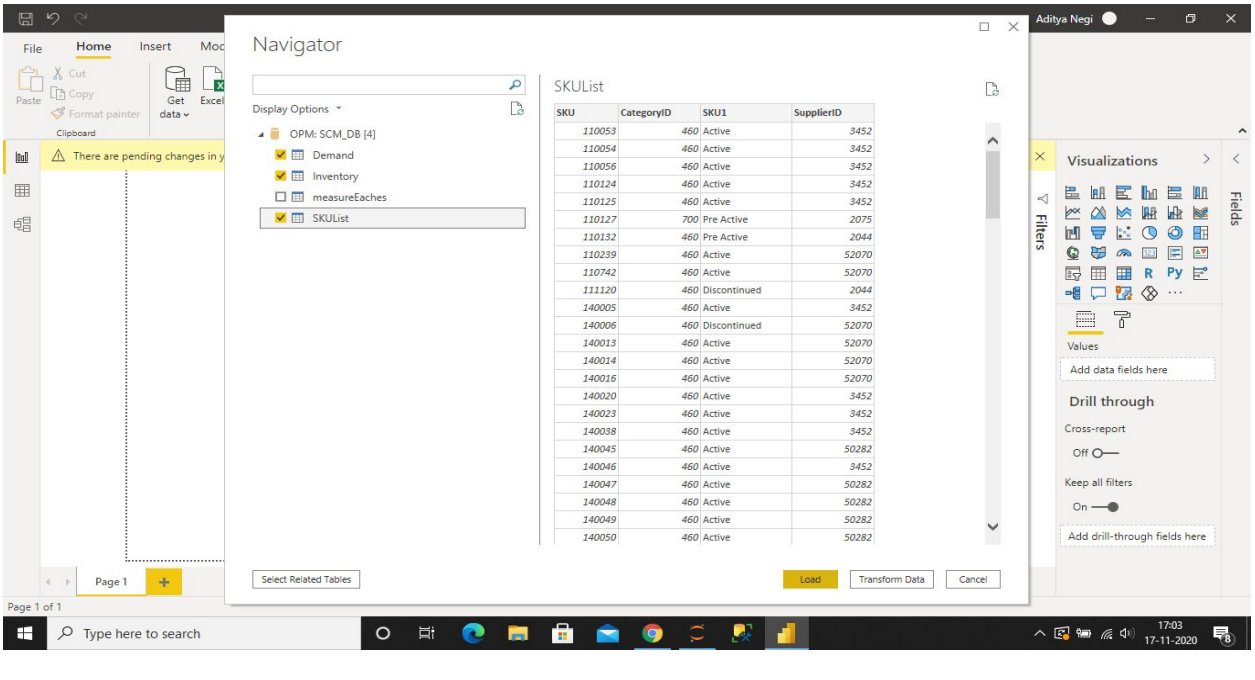
We can also connect our saved files (.csv,.xls) or sql server to Power BI used for creating powerful dashboards.

The powerBI window looks like this and we can import data from multiple sources. Let's connect with our SQL server and also add some flat files. Then we can perform visualization on the data.

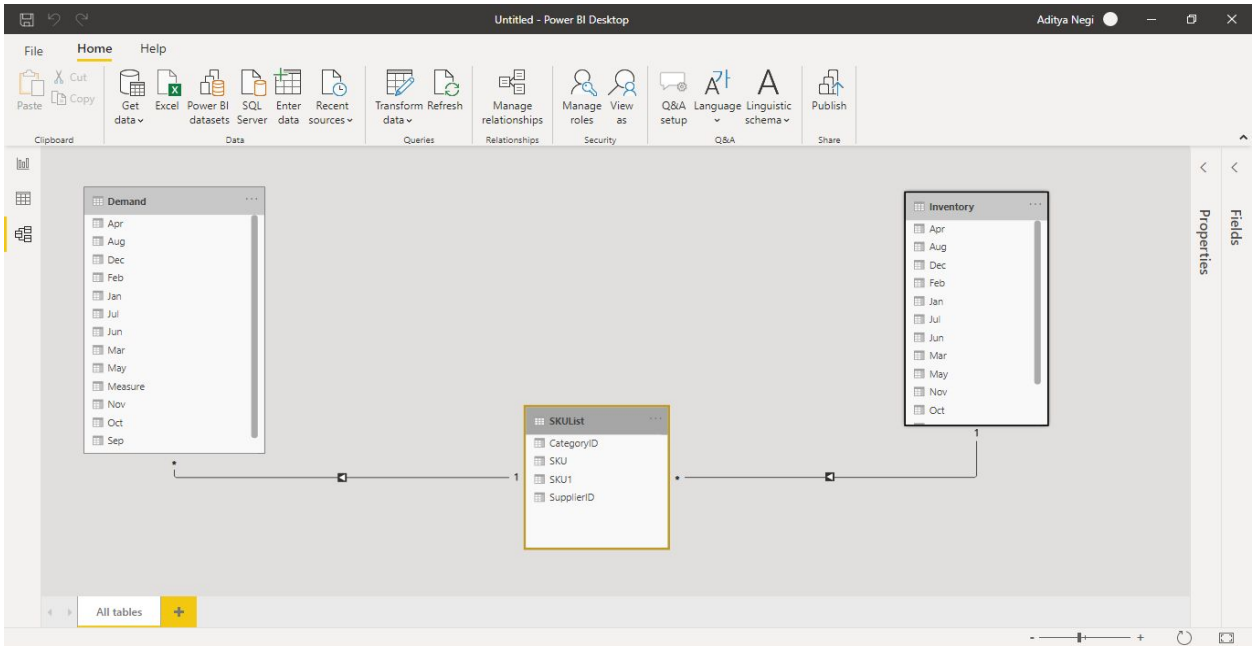


To connect to SQL we can select SQL Server from the toolbar and connect to the database.

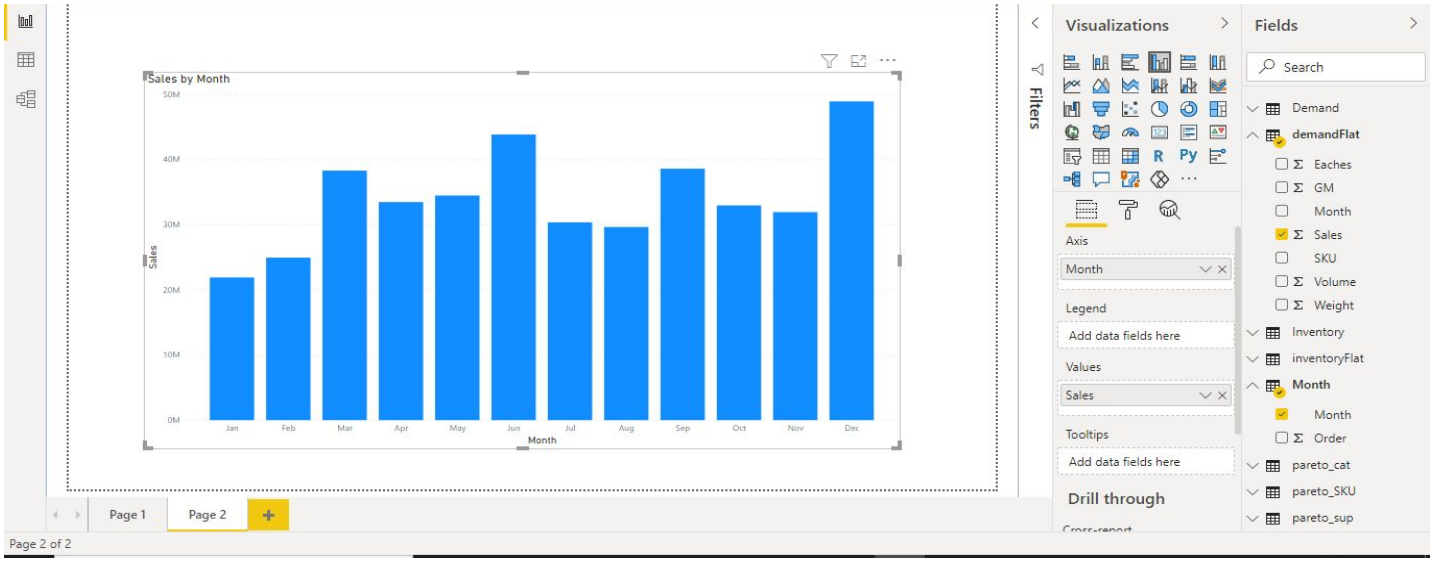


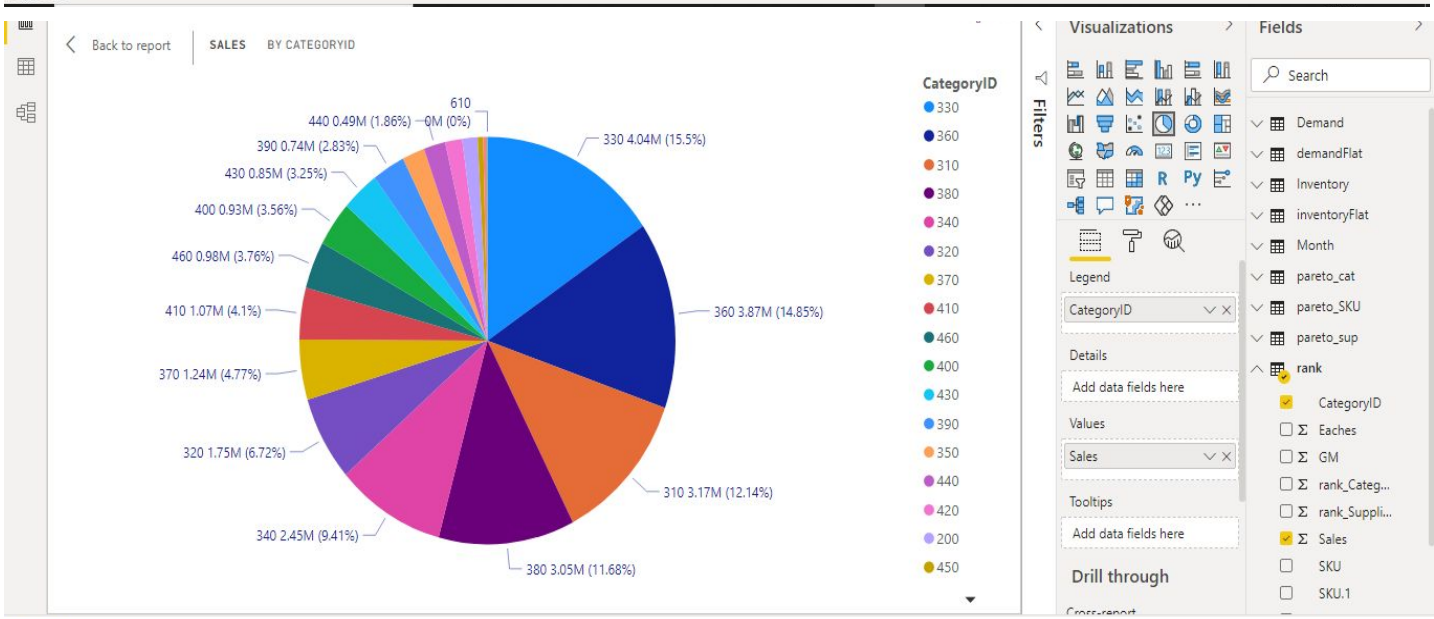


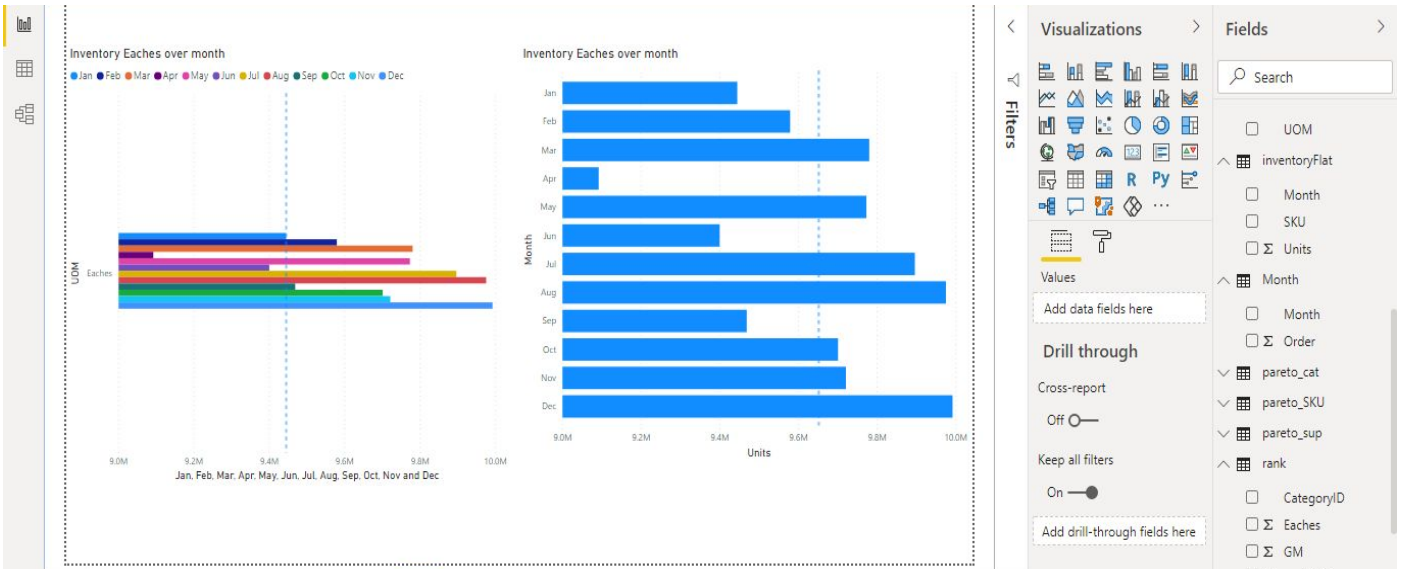
Data modelling can also be done using PowerBI. Relationships of different tables within a SQL server can be defined.

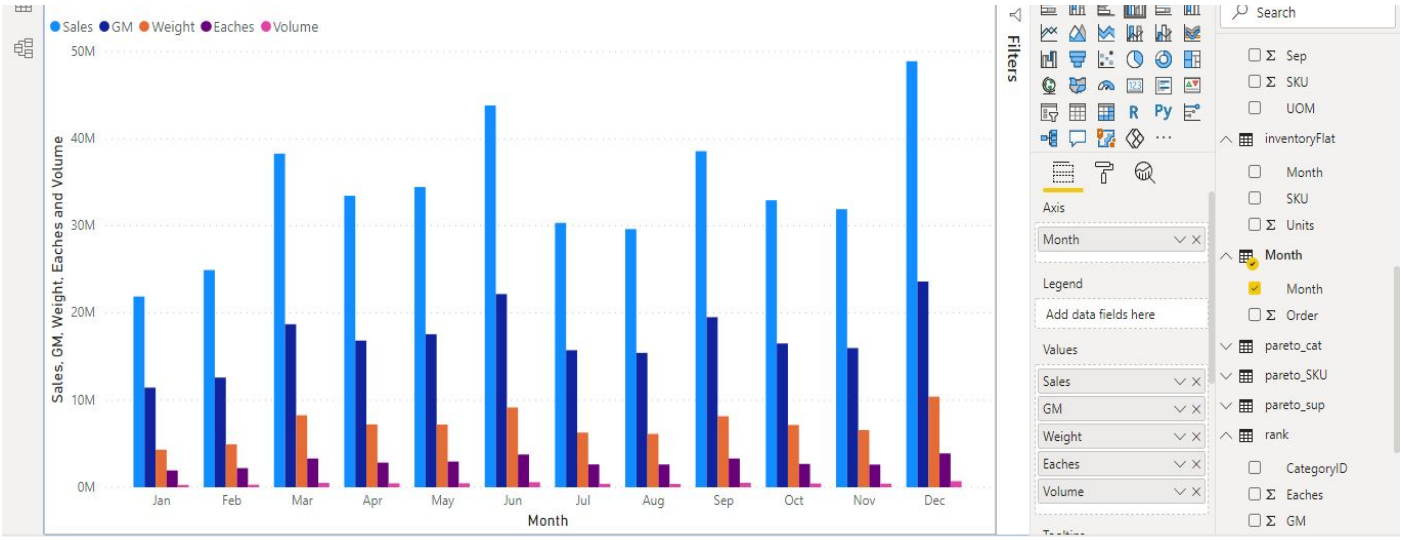


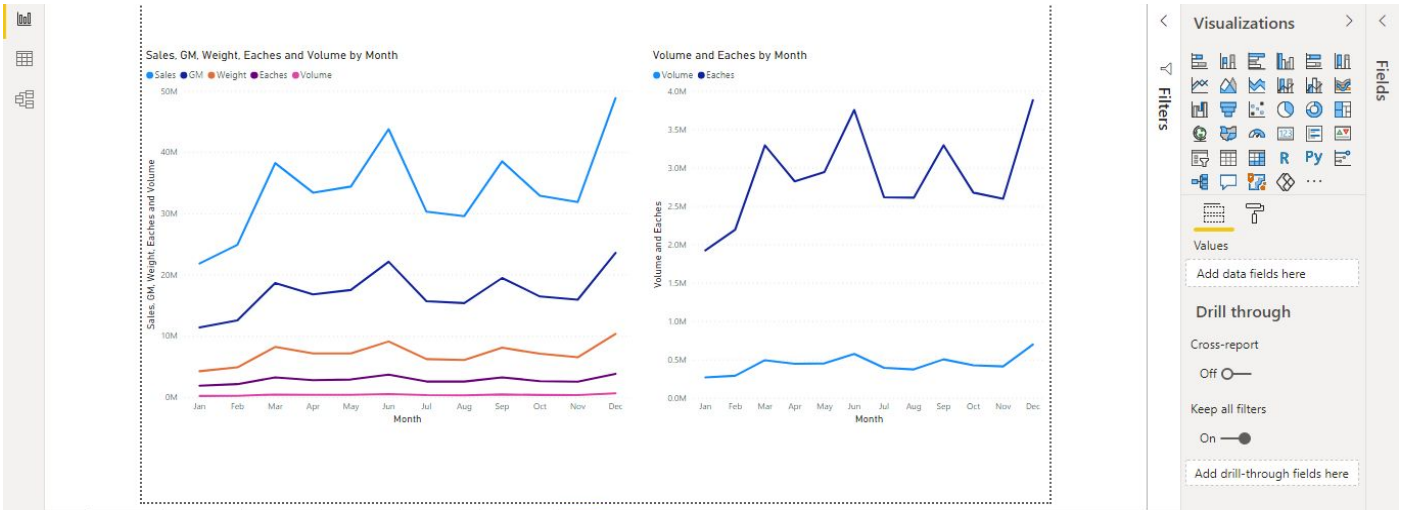
Once loaded we can create any visualization from the Visualizations panel and select the columns to be selected. We can load .csv/.xls files.






Inventory Eaches over month
Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
UOM
Eaches
9.0M 9.2M 9.4M 9.6M 9.8M 10.0M
Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov and Dec
Inventory Eaches over month
Month
Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
9.0M 9.2M 9.4M 9.6M 9.8M 10.0M
Units
Filters
Visualizations
Values
Add data fields here
Drill through
Cross-report
Off
Keep all filters
On
Add drill-through fields here
Fields
Search
UOM
inventoryFlat
Month
SKU
Units
Month
Month
Order
pareto_cat
pareto_SKU
pareto_sup
rank
CategoryID
Eaches
GM


Sales GM Weight Eaches Volume
50M
40M
30M
20M
10M
0M
Sales, GM, Weight, Eaches and Volume
Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
Month
Filters
Axis
Month
Legend
Add data fields here
Values
Sales
GM
Weight
Eaches
Volume
Search
Sep
SKU
UOM
inventoryFlat
Month
SKU
Units
Month
Month
Order
pareto_cat
pareto_SKU
pareto_sup
rank
CategoryID
Eaches
GM


Sales, GM, Weight, Eaches and Volume by Month
Sales GM Weight Eaches Volume
Sales, GM, Weight, Eaches and Volume
Month
Volume and Eaches by Month
Volume Eaches
Volume and Eaches
Month
Filters
Visualizations
Values
Add data fields here
Drill through
Cross-report
Off
Keep all filters
On
Add drill-through fields here
Fields

We can also form pareto using calculating cumulative percentage in python and using PowerBI for visualisation.

```
In [10]: pareto_ID = rank[['Sales','CategoryID','SupplierID']]
```

```
In [13]: pareto_ID.head()
```

Out[13]:

	Sales	CategoryID	SupplierID
0	756.083333	460	3452
1	519.500000	460	3452
2	471.166667	460	3452
3	1285.416667	460	3452
4	905.666667	460	3452

```
In [15]: cat_par = pareto_ID[['CategoryID','Sales']].groupby('CategoryID').agg('sum')
         sup_par = pareto_ID[['SupplierID','Sales']].groupby('SupplierID').agg('sum')
```

```
In [18]: cat_par.head()
```

Out[18]:

CategoryID	Sales
200	3.449941e+05
310	3.165176e+06
320	1.751735e+06
330	4.040084e+06
340	2.453166e+06

```
In [20]: cat_par = cat_par.sort_values(by='Sales',ascending=False)
         cat_par["cumpercentage"] = cat_par["Sales"].cumsum()/cat_par["Sales"].sum()*100
```

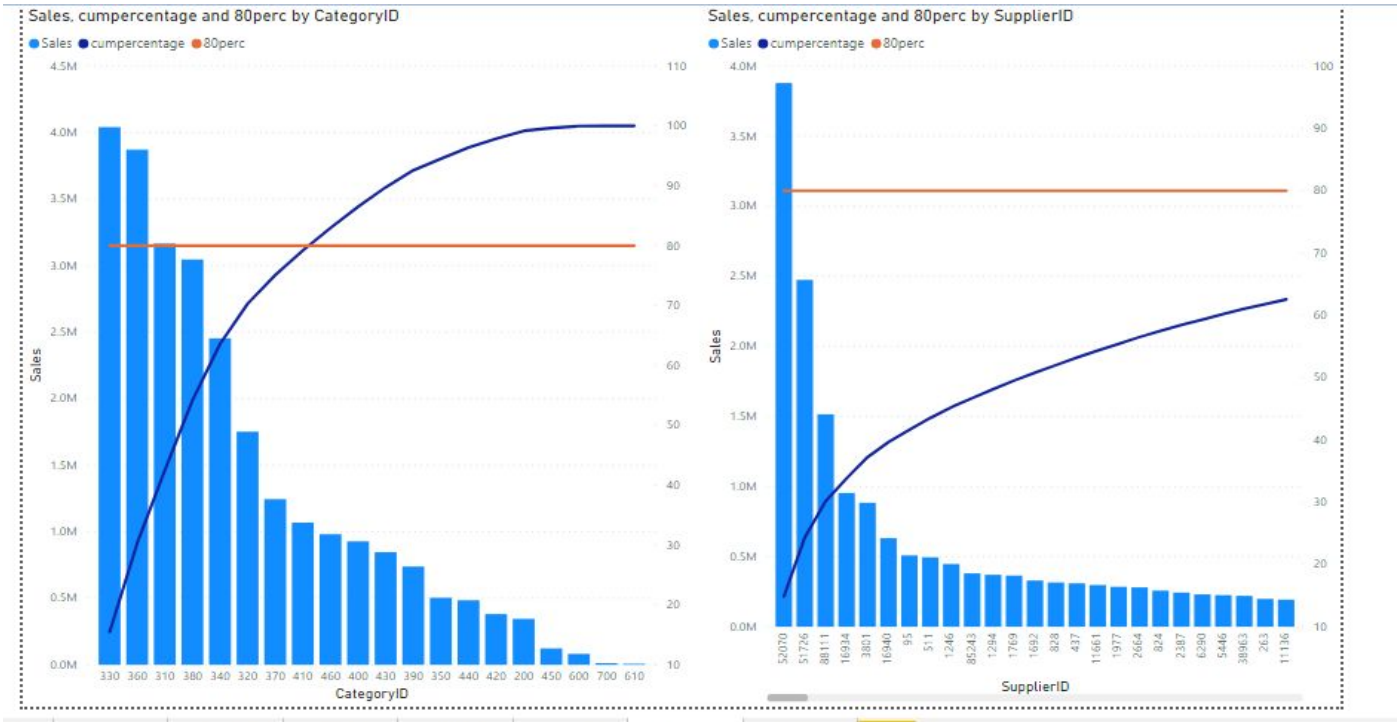
```
In [22]: sup_par = sup_par.sort_values(by='Sales',ascending=False)
         sup_par["cumpercentage"] = sup_par["Sales"].cumsum()/sup_par["Sales"].sum()*100
         sup_par.head()
```

Out[22]:

SupplierID	Sales	cumpercentage
52070	3.878076e+06	14.878850
51726	2.474093e+06	24.371097
88111	1.515727e+06	30.186424
16934	9.543797e+05	33.848053
3801	8.845327e+05	37.241702

```
In [24]: cat_par.to_csv('pareto_cat.csv')
         sup_par.to_csv('pareto_sup.csv')
```

Loading these files in PowerBI and creating a Pareto Chart by using Line and Column chart. Make sure to change categoryID and supplierID into ‘object’/’text’ datatype in powerBI.



We can combine all the import visualizations and create a dashboard all together. This gives a single view representation.