In [1]:

```
# This Python 3 environment comes with many helpful analytics libraries insta
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will li

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that g
# You can also write temporary files to /kaggle/temp/, but they won't be save
```

In [2]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import plotly.express as px
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
```

In [3]:

```
df=pd.read_csv('../input/sample-sales-data/sales_data_sample.csv',encoding='u
df.head()
```

Out[3]:

	ORDERNUMBER	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	ORDERD
0	10107	30	95.70	2	2871.00	2/24/
1	10121	34	81.35	5	2765.90	5/7/2003
2	10134	41	94.74	2	3884.34	7/1/2003
3	10145	45	83.26	6	3746.70	8/25/
4	10159	49	100.00	14	5205.27	10/10/

5 rows × 25 columns

In [4]:
```
to_drop = ['PHONE','ADDRESSLINE1','ADDRESSLINE2','CITY','STATE','POSTALCODE',
df=df.drop(to_drop, 1)
df.head()
```

```
/opt/conda/lib/python3.7/site-packages/ipykernel_launcher.py:2: FutureWarni
ng: In a future version of pandas all arguments of DataFrame.drop except fo
r the argument 'labels' will be keyword-only
```

Out[4]:

	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	ORDERDATE	STATUS	Q
0	30	95.70	2	2871.00	2/24/2003 0:00	Shipped	
1	34	81.35	5	2765.90	5/7/2003 0:00	Shipped	
2	41	94.74	2	3884.34	7/1/2003 0:00	Shipped	
3	45	83.26	6	3746.70	8/25/2003 0:00	Shipped	
4	49	100.00	14	5205.27	10/10/2003 0:00	Shipped	

In [5]:
```
df.nunique()
```

Out[5]:
```
QUANTITYORDERED      58
PRICEEACH          1016
ORDERLINENUMBER      18
SALES              2763
ORDERDATE           252
STATUS                6
QTR_ID                4
MONTH_ID             12
YEAR_ID               3
PRODUCTLINE           7
MSRP                 80
PRODUCTCODE         109
COUNTRY              19
DEALSIZE              3
dtype: int64
```

In [6]:
```
df.isnull().sum()
```

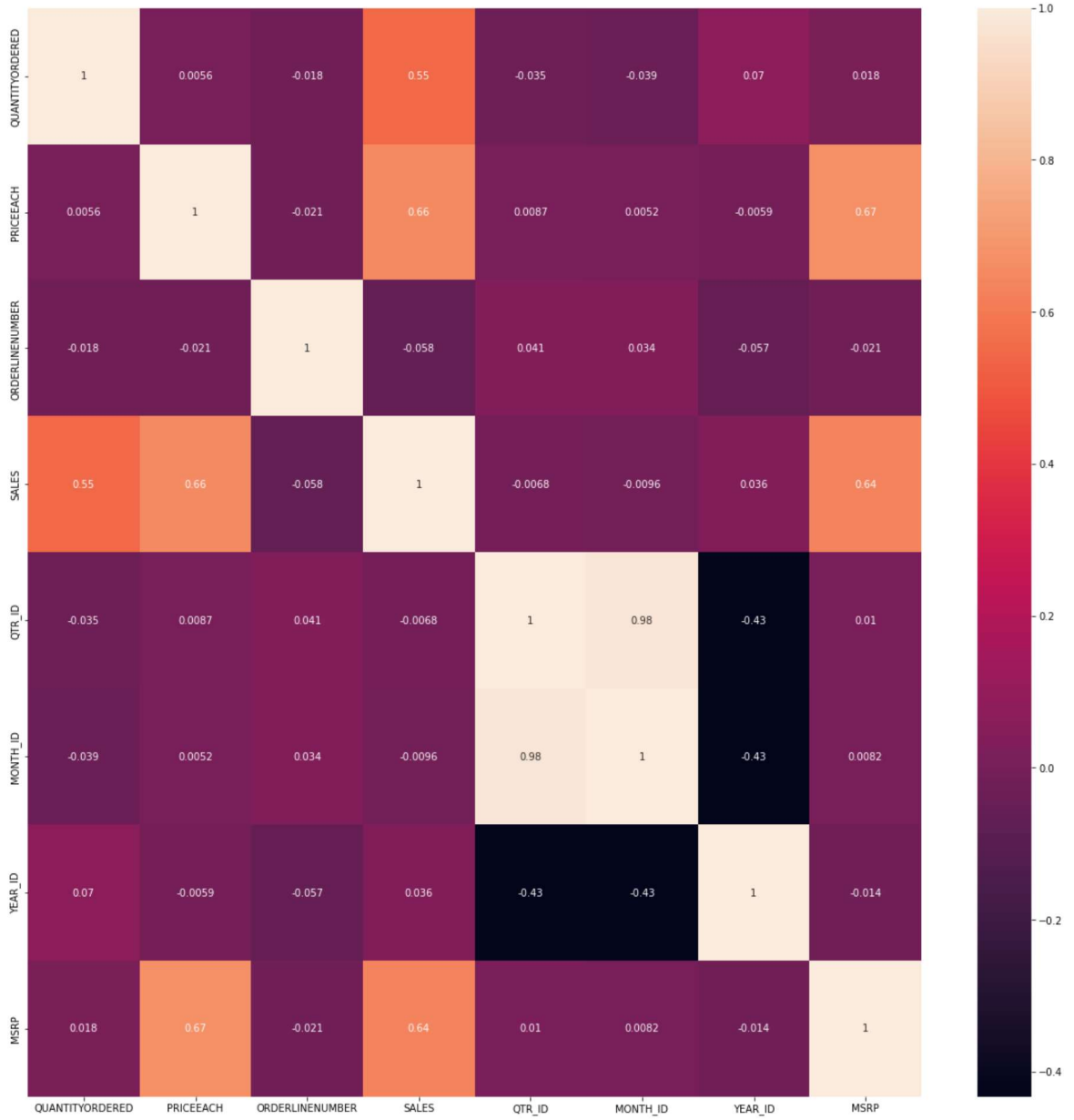
Out[6]:
```
QUANTITYORDERED    0
PRICEEACH          0
ORDERLINENUMBER    0
SALES              0
ORDERDATE          0
STATUS             0
QTR_ID             0
MONTH_ID           0
YEAR_ID            0
PRODUCTLINE        0
MSRP               0
PRODUCTCODE        0
COUNTRY            0
DEALSIZE           0
dtype: int64
```

In [7]:
```
status_dict = {'Snipped' :1, 'Cancelled':2, 'On Hold':2, 'Disputed' :2, 'In P
df['STATUS'].replace(status_dict, inplace=True)
df['PRODUCTCODE'] = pd.Categorical(df['PRODUCTCODE']).codes
df = pd.get_dummies(data=df, columns=['PRODUCTLINE','DEALSIZE', 'COUNTRY'])
df.dtypes
```

Out[7]:
```
QUANTITYORDERED                  int64
PRICEEACH                      float64
ORDERLINENUMBER                  int64
SALES                          float64
ORDERDATE                       object
STATUS                          object
QTR_ID                           int64
MONTH_ID                         int64
YEAR_ID                          int64
MSRP                             int64
PRODUCTCODE                       int8
PRODUCTLINE_Classic Cars         uint8
PRODUCTLINE_Motorcycles          uint8
PRODUCTLINE_Planes               uint8
PRODUCTLINE_Ships                uint8
PRODUCTLINE_Trains               uint8
PRODUCTLINE_Trucks and Buses     uint8
PRODUCTLINE_Vintage Cars         uint8
DEALSIZE_Large                   uint8
DEALSIZE_Medium                  uint8
DEALSIZE_Small                   uint8
COUNTRY_Australia                uint8
COUNTRY_Austria                  uint8
COUNTRY_Belgium                  uint8
COUNTRY_Canada                   uint8
COUNTRY_Denmark                  uint8
COUNTRY_Finland                  uint8
COUNTRY_France                   uint8
COUNTRY_Germany                  uint8
COUNTRY_Ireland                  uint8
COUNTRY_Italy                    uint8
COUNTRY_Japan                    uint8
COUNTRY_Norway                   uint8
COUNTRY_Philippines              uint8
COUNTRY_Singapore                uint8
COUNTRY_Spain                    uint8
COUNTRY_Sweden                   uint8
COUNTRY_Switzerland              uint8
COUNTRY_UK                       uint8
COUNTRY_USA                      uint8
dtype: object
```

In [8]:

```
plt.figure(figsize = (20, 20))
corr_matrix = df.iloc[:, :10].corr()
sns.heatmap(corr_matrix, annot=True);
```



In [12]:

```
fig = px.scatter_matrix(df, dimensions=df.columns[:8], color= 'MONTH_ID')
fig.update_layout(title_text='Sales Data', width=1100, height=1100)
fig.show()
```

In [16]:

```
std=StandardScaler()
sdf = std.fit_transform(df)
wcss=[]
for i in range(1,15):
    km = KMeans(n_clusters=i)
    km.fit(sdf)
    wcss.append(km.inertia_) # intertio is the Sum of squared distance

plt.plot(wcss, marker='o', Linestyle='--')
plt.title('The Elbow Method (Finding right number of clusters)')
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS')
plt.show()
```

```
---------------------------------------------------------------------------
---
ValueError                                Traceback (most recent call la
st)
/tmp/ipykernel_27/3612628103.py in <module>
      1 std=StandardScaler()
----> 2 sdf = std.fit_transform(df)
      3 wcss=[]
      4 for i in range(1,15):
      5     km = KMeans(n_clusters=i)

/opt/conda/lib/python3.7/site-packages/sklearn/base.py in fit_transform
(self, X, y, **fit_params)
    850         if y is None:
    851             # fit method of arity 1 (unsupervised transformatio
n)
--> 852             return self.fit(X, **fit_params).transform(X)
    853         else:
    854             # fit method of arity 2 (supervised transformation)

/opt/conda/lib/python3.7/site-packages/sklearn/preprocessing/_data.py in
fit(self, X, y, sample_weight)
    804         # Reset internal state before fitting
    805         self._reset()
--> 806         return self.partial_fit(X, y, sample_weight)
    807
    808     def partial_fit(self, X, y=None, sample_weight=None):

/opt/conda/lib/python3.7/site-packages/sklearn/preprocessing/_data.py in
partial_fit(self, X, y, sample_weight)
    845             dtype=FLOAT_DTYPES,
    846             force_all_finite="allow-nan",
--> 847             reset=first_call,
    848         )
    849         n_features = X.shape[1]

/opt/conda/lib/python3.7/site-packages/sklearn/base.py in _validate_data
(self, X, y, reset, validate_separately, **check_params)
    564             raise ValueError("Validation should be done on X, y
 or both.")
    565         elif not no_val_X and no_val_y:
--> 566             X = check_array(X, **check_params)
```

```
    567             out = X
    568         elif no_val_X and not no_val_y:

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py in ch
eck_array(array, accept_sparse, accept_large_sparse, dtype, order, copy,
force_all_finite, ensure_2d, allow_nd, ensure_min_samples, ensure_min_fe
atures, estimator)
    744                     array = array.astype(dtype, casting="unsafe"
, copy=False)
    745                 else:
--> 746                     array = np.asarray(array, order=order, dtype
=dtype)
    747             except ComplexWarning as complex_warning:
    748                 raise ValueError(

/opt/conda/lib/python3.7/site-packages/pandas/core/generic.py in __array
__(self, dtype)
   1991
   1992     def __array__(self, dtype: NpDtype | None = None) -> np.ndar
ray:
-> 1993         return np.asarray(self._values, dtype=dtype)
   1994
   1995     def __array_wrap__(

ValueError: could not convert string to float: '2/24/2003 0:00'
```