# Swivelt IT Sales Performance Dataset

January 9, 2021

## 0.1 Fatimah binti Mohd Nizam

**Data science Assessment**

[ ]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression,Ridge,Lasso
import numpy as np
from sklearn.metrics import r2_score, mean_squared_error
```

## 0.2 Objective

1. To analyze the IT sales Performance throughout the year.
2. To predict the CustomerOrder feature by using supervised machine learning algorithms.
3. To analyze which Companies has the highest purchase of product from the IT department.
4. To analyze which products are among the highest purchased product from the IT department.
5. To observe the rate of increment for both NetLineDollarPrice and NetOrderDollar Price features.

### 0.2.1 1. Data Understanding

[ ]:
```
#Read the csv file
df = pd.read_csv('Sales Data.csv')
```

```
/usr/local/lib/python3.6/dist-packages/IPython/core/interactiveshell.py:2718:
DtypeWarning:

Columns (38,46) have mixed types.Specify dtype option on import or set
low_memory=False.
```

[ ]:
```
#Observe the dataset
df
```

[ ]:
```
      PurchaseOrderDate PurchaseOrderNo  … NetOptionDollarPrice ShipDate
0             15/5/2017        PO 94464  …               179.06      NaN
1             15/5/2017        PO 94464  …               -35.52      NaN
2              5/1/2018   JV2017/000058  …                 0.00      NaN
3              5/1/2018   JV2017/000058  …                 5.98      NaN
4              5/1/2018   JV2017/000058  …                -1.19      NaN
…                   …             …  …                  …      …
59373         22/8/2017         SG 1084  …                26.04      NaN
59374         22/8/2017         SG 1084  …                -8.31      NaN
59375         22/8/2017         SG 1084  …                19.36      NaN
59376         22/8/2017         SG 1084  …                 0.00      NaN
59377         22/8/2017         SG 1084  …                -6.18      NaN

[59378 rows x 47 columns]
```

[ ]:
```
#to see on how many features and observations in the dataset
df.shape
```

[ ]:
```
(59378, 47)
```

[ ]:
```
# count how many data available
df.count()
```

[ ]:
```
PurchaseOrderDate          59378
PurchaseOrderNo            59378
PurchasingAgent            58560
ProductLineCode            58560
ProductLineDescription     58520
ProductLines               59378
ProductNumber              59378
NetLineDollarPrice         59378
NetOrderDollarPrice        59378
OrderedQuantity            59378
TeleWebAgentID             52180
SalesRep                   59378
WebOrderNo                 59378
CatalogID                  51908
CatalogName                  390
InvoiceDate                48420
InvoiceNo                  48420
InvoiceStatus              59378
InvoiceToAddr              59378
Status                     59378
CustomerName               59378
```

```
TeleWebAgentName              51796
BusinessUnit                  58520
SAPSalesOrderNoMfgSO          58988
ProductDescription            59368
EAD                           47064
SoldToAttentionEmail            250
SoldToAttentionPhone          58798
SoldToAddr                    59136
NetInvoiceLineDollarPrice         0
xypOrderNo                    59378
ShipToAddr1                   59348
ShipToAddr2                   59378
ShipToAddr3                   57908
ShipToAddr4                   51060
ShipToAddr5                   59378
PaymentMethod                 52180
PaymentMethodDescription      52180
PaymentReceiveDate              384
PaymentTerms                  58864
PaymentTermsDescription       58988
CustomerOrder                 59378
xypCustomerNumber             58960
VendorName                    53154
VendorPOMfgSO                 24860
NetOptionDollarPrice          59368
ShipDate                      19682
dtype: int64
```

[ ]:
```python
# To get more information regarding the dataset.
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 59378 entries, 0 to 59377
Data columns (total 47 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   PurchaseOrderDate          59378 non-null  object
 1   PurchaseOrderNo            59378 non-null  object
 2   PurchasingAgent            58560 non-null  object
 3   ProductLineCode            58560 non-null  object
 4   ProductLineDescription     58520 non-null  object
 5   ProductLines               59378 non-null  object
 6   ProductNumber              59378 non-null  object
 7   NetLineDollarPrice         59378 non-null  float64
 8   NetOrderDollarPrice        59378 non-null  float64
 9   OrderedQuantity            59378 non-null  int64
 10  TeleWebAgentID             52180 non-null  object
```

```
 11  SalesRep                    59378 non-null  object
 12  WebOrderNo                  59378 non-null  object
 13  CatalogID                   51908 non-null  object
 14  CatalogName                 390 non-null    object
 15  InvoiceDate                 48420 non-null  object
 16  InvoiceNo                   48420 non-null  object
 17  InvoiceStatus               59378 non-null  object
 18  InvoiceToAddr               59378 non-null  object
 19  Status                      59378 non-null  object
 20  CustomerName                59378 non-null  object
 21  TeleWebAgentName            51796 non-null  object
 22  BusinessUnit                58520 non-null  object
 23  SAPSalesOrderNoMfgSO        58988 non-null  object
 24  ProductDescription          59368 non-null  object
 25  EAD                         47064 non-null  object
 26  SoldToAttentionEmail        250 non-null    object
 27  SoldToAttentionPhone        58798 non-null  object
 28  SoldToAddr                  59136 non-null  object
 29  NetInvoiceLineDollarPrice   0 non-null      float64
 30  xypOrderNo                  59378 non-null  object
 31  ShipToAddr1                 59348 non-null  object
 32  ShipToAddr2                 59378 non-null  object
 33  ShipToAddr3                 57908 non-null  object
 34  ShipToAddr4                 51060 non-null  object
 35  ShipToAddr5                 59378 non-null  object
 36  PaymentMethod               52180 non-null  object
 37  PaymentMethodDescription    52180 non-null  object
 38  PaymentReceiveDate          384 non-null    object
 39  PaymentTerms                58864 non-null  object
 40  PaymentTermsDescription     58988 non-null  object
 41  CustomerOrder               59378 non-null  object
 42  xypCustomerNumber           58960 non-null  float64
 43  VendorName                  53154 non-null  object
 44  VendorPOMfgSO               24860 non-null  object
 45  NetOptionDollarPrice        59368 non-null  float64
 46  ShipDate                    19682 non-null  object
dtypes: float64(5), int64(1), object(41)
memory usage: 21.3+ MB
```

## 0.3 2. Data Cleansing

[ ]:
```python
# to check how many missing values in the feature
df.isnull().sum()
```

[ ]:
```
PurchaseOrderDate                 0
PurchaseOrderNo                   0
```

```
PurchasingAgent                  818
ProductLineCode                  818
ProductLineDescription           858
ProductLines                       0
ProductNumber                      0
NetLineDollarPrice                 0
NetOrderDollarPrice                0
OrderedQuantity                    0
TeleWebAgentID                  7198
SalesRep                           0
WebOrderNo                         0
CatalogID                       7470
CatalogName                    58988
InvoiceDate                    10958
InvoiceNo                      10958
InvoiceStatus                      0
InvoiceToAddr                      0
Status                             0
CustomerName                       0
TeleWebAgentName                7582
BusinessUnit                     858
SAPSalesOrderNoMfgSO             390
ProductDescription                10
EAD                            12314
SoldToAttentionEmail           59128
SoldToAttentionPhone             580
SoldToAddr                       242
NetInvoiceLineDollarPrice      59378
xypOrderNo                         0
ShipToAddr1                       30
ShipToAddr2                        0
ShipToAddr3                     1470
ShipToAddr4                     8318
ShipToAddr5                        0
PaymentMethod                   7198
PaymentMethodDescription        7198
PaymentReceiveDate             58994
PaymentTerms                     514
PaymentTermsDescription          390
CustomerOrder                      0
xypCustomerNumber                418
VendorName                      6224
VendorPOMfgSO                  34518
NetOptionDollarPrice              10
ShipDate                       39696
dtype: int64
```

[ ]:
```
## Percentage of NAN values
df.isnull().mean()
```

[ ]:
```
PurchaseOrderDate             0.000000
PurchaseOrderNo               0.000000
PurchasingAgent               0.013776
ProductLineCode               0.013776
ProductLineDescription        0.014450
ProductLines                  0.000000
ProductNumber                 0.000000
NetLineDollarPrice            0.000000
NetOrderDollarPrice           0.000000
OrderedQuantity               0.000000
TeleWebAgentID                0.121223
SalesRep                      0.000000
WebOrderNo                    0.000000
CatalogID                     0.125804
CatalogName                   0.993432
InvoiceDate                   0.184546
InvoiceNo                     0.184546
InvoiceStatus                 0.000000
InvoiceToAddr                 0.000000
Status                        0.000000
CustomerName                  0.000000
TeleWebAgentName              0.127690
BusinessUnit                  0.014450
SAPSalesOrderNoMfgSO          0.006568
ProductDescription            0.000168
EAD                           0.207383
SoldToAttentionEmail          0.995790
SoldToAttentionPhone          0.009768
SoldToAddr                    0.004076
NetInvoiceLineDollarPrice     1.000000
xypOrderNo                    0.000000
ShipToAddr1                   0.000505
ShipToAddr2                   0.000000
ShipToAddr3                   0.024757
ShipToAddr4                   0.140086
ShipToAddr5                   0.000000
PaymentMethod                 0.121223
PaymentMethodDescription      0.121223
PaymentReceiveDate            0.993533
PaymentTerms                  0.008656
PaymentTermsDescription       0.006568
CustomerOrder                 0.000000
xypCustomerNumber             0.007040
VendorName                    0.104820
```

```
VendorPOMfgSO              0.581326
NetOptionDollarPrice       0.000168
ShipDate                   0.668530
dtype: float64
```

It is possible to delete the missing values columns in the dataset because:

1. it is not affecting the result due to the large number of difference between the missing values and the available values.
2. From the data above the percentage of the missing values are not high except for PaymentReceivedDate, SoldToAttentionEmail, CatalogName, VendorPOMfgSO, NetInvoiceLineDollarPrice and ShipDate .
3. However, all of the columns mentioned in 2 are not neccesary for the objectives of this analysis.

[ ]:
```python
#Dropping the columns that have missing values
new_df = df.dropna(axis='columns')
```

[ ]:
```python
# Observe the new dataset
new_df
```

[ ]:
```
      PurchaseOrderDate PurchaseOrderNo  …          ShipToAddr5 CustomerOrder
0             15/5/2017        PO 94464  …  75350 BATU BERENDAM             N
1             15/5/2017        PO 94464  …  75350 BATU BERENDAM             N
2              5/1/2018    JV2017/000058  …  75350 BATU BERENDAM             N
3              5/1/2018    JV2017/000058  …  75350 BATU BERENDAM             N
4              5/1/2018    JV2017/000058  …  75350 BATU BERENDAM             N
…                   …             …  …                  …           …
59373         22/8/2017         SG 1084  …     629824 SINGAPORE             Y
59374         22/8/2017         SG 1084  …     629824 SINGAPORE             Y
59375         22/8/2017         SG 1084  …     629824 SINGAPORE             Y
59376         22/8/2017         SG 1084  …     629824 SINGAPORE             Y
59377         22/8/2017         SG 1084  …     629824 SINGAPORE             Y

[59378 rows x 17 columns]
```

[ ]:
```python
# to check how many missing values in the feature
new_df.isnull().sum()
```

[ ]:
```
PurchaseOrderDate       0
PurchaseOrderNo         0
ProductLines            0
ProductNumber           0
NetLineDollarPrice      0
NetOrderDollarPrice     0
OrderedQuantity         0
SalesRep                0
WebOrderNo              0
```

```
InvoiceStatus          0
InvoiceToAddr          0
Status                 0
CustomerName           0
xypOrderNo             0
ShipToAddr2            0
ShipToAddr5            0
CustomerOrder          0
dtype: int64
```

[ ]:
```
# To have a clearer observation of the dataset
new_df.head(10)
```

[ ]:
```
  PurchaseOrderDate PurchaseOrderNo  …          ShipToAddr5 CustomerOrder
0         15/5/2017        PO 94464  …  75350 BATU BERENDAM             N
1         15/5/2017        PO 94464  …  75350 BATU BERENDAM             N
2          5/1/2018   JV2017/000058  …  75350 BATU BERENDAM             N
3          5/1/2018   JV2017/000058  …  75350 BATU BERENDAM             N
4          5/1/2018   JV2017/000058  …  75350 BATU BERENDAM             N
5          5/1/2018   JV2017/000058  …  75350 BATU BERENDAM             N
6          5/1/2018   JV2017/000058  …  75350 BATU BERENDAM             N
7          5/1/2018   JV2017/000058  …  75350 BATU BERENDAM             N
8         10/4/2017      SCP0748376  …  75350 BATU BERENDAM             N
9         10/4/2017      SCP0748376  …  75350 BATU BERENDAM             N

[10 rows x 17 columns]
```

[ ]:
```
# To view the rows and columns of the dataset
new_df.shape
```

[ ]:
```
(59378, 17)
```

From 44 features, only 17 features are selected after the data cleasing.

## 0.4 3. Data Aggregation and Data Deduplication

- Data aggregation is the process of gathering the data and represent it in a summarized way.
- Data deduplication is a technique for eliminating duplicate copies of repeating data.

To evaluate whether that the ProductLines, ProductNumber and PurchaseOrderNo features are having the same information

[ ]:
```
# To see on how much unique values allocated in ProductLines feature
productline = new_df['ProductLines'].unique()
productline
```

[ ]:
```
array(['7F 8W 9F 9R AN BO MP', '8J 8N 8W MG', '8J 8N MG', '8J 8W FF',
       '8W MP TA', '8W 9F', '8W AN FF MG MP', '5X 8W AN FF MG MP',
       '5U 7F 8W BO', '5U 7F 8W', '8W 9G DG', '5U 7F 8W MG', '{}',
       '9G GA', '8W AN MG MP TA', '1M 2G 8W', '1M 2Q 8W',
       '1M 2G 8W 9G MP', '1M 5U BO', '1M', '1M 8W', '1M 8W MP',
       '1M 8W 9G KV', '1M 2N 8W 9G AU', '1M 2N 8W KV', '2G 8W',
       '16 5U 7F 8W BO', '5U 7F 8W BO MG', '5X 8W 9H', '8W 9G KV',
       '8W DG MG', '2G 8W DG', '8W BO DG MG', '5U 7F 8W 9F BO',
       '5U 7F 9F BO', '7F 8W 9R', '16 7F 9G BO MG', '7F 8W AN MG MP',
       '7F AN MG MP', '7F 8W', '7F 8W BO GA MP', '7F 8W 9G', '7F 8W BO',
       '7F 8W 9G BO', '7F 8W 9G BO MG', '7F 8W BO MG', '7F', '7F 8W MG',
       '7F 8W TB', '6U 8W MG MP', '5U 8W MP', '8W AN MG MP', 'AN',
       '16 7F 8W 9R', '16 7F 9R', '8W 9G KV MN MP', '2N 8W',
       '2N 4H 9G DU FF KV', '1M 2N 4H 9G DU FF KV', '2N AU',
       '2N 8W 9G AU KV MP', '2N 8W AU', '2N 5M 8W AU', '5X 8W',
       '8W GA MG', '5X 8W 9F 9G 9H MG TB', '7F 8W 9G MP', 'AN MP',
       '8J 8W MG MP', '8W DU MP TA', '8W BO MG MP TA', '8W 9G AN MP',
       '8W AN MG', '8W 9G AN MG MP', '8W AN MG MP TB',
       '8W 9G AN BO MG MP', '8W AN MP', 'AN MG MP', '8W 9F BO',
       '6U 8W BO MG MP', '6U 8W 9G MG MP', '6U MG MP', '6U 8W MP',
       '8W MP', '6U 8W 9G KV MP', '5X 8W TB', '5X 8W MP TB', '6U 8W',
       '1M 8W 9G MP', '1M 8W 9G', '1M 8W 9G MN', '1M 2G 8W 9G',
       '8W 9G KV MP', '8W KV', '8W AN BO MG MP', '6U 8W AN BO MG MP',
       '8N 8W', '8W 9G MN MP', '8W 9G', '5X 8W 9G BO', '5X 8W BO',
       '8W BO', 'BO', '8W BO DG', '8W 9G BO', '8W 9F BO MP',
       '8W 9G BO DG', '6U 8W MG', '6U 8W MG MP TB', '6U 8W 9G BO MG',
       '1N 4H 8W', '1N 8W', '1N 8W 9G', '5X 8W 9G', '8W 9F AN BO MG MP',
       '2G 8W 9G MP', '8W UD', '8W AU', '8W 9G MP', '5U 9G BO',
       '8J 8W 9G MG MP', '9G AN MG MP', '9G MG', '8W 9G MG', '9G',
       '5U 9G', '5U 6U 8W 9G MG MP', '5U 8W 9G BO MG',
       '6U 8W 9G BO MG MP', '5X 8W 9F 9G', '6U 9G', '4H 8W 9G KV MN MP',
       '5U 8W 9G BO', '8W TA', '5X 8W BO MP', '5X 8W MP', '5X 8W 9F MP',
       '8W BO MG', '5U 8W BO', '5U 8W BO MG', '7F 8W MP',
       '2G 8W 9G KV MP', '8W MN', '8W 9G KV MN', 'MG', '2Q 8W GP R6',
       '8W R6', '7T R6', '7T 8W R6', '8W MG', '8W MG MP TA', '8W MN MP',
       '8W MN MP TA', '1N 7T 8W', '2G 8W MP', '2B 8W', '2B 5X 8W TB',
       '8W BO GP', '2G 8W 9G', '1D 8W', '8W 9F MP', '8W GP', 'MP',
       '7F 8W 9F MP', '2C 8W MP', '5X 8W 9F BO', '8W PQ', '9G KV',
       '2B 3Y 8W 9G KV MP', '7T 8W', '2Q 8W GP', '2Q 8W', '8W DU',
       '8W MA', '8W GJ', '8W UK', '8W GN', '5U 8W', '8W DG',
       '5X 8W 9H TB', '8W GJ GP', '8W GK', '8A 8W', '5M 8W', '8W TB',
       '8W AN', '1N 8W GN', '8W GN GP', '1N', '5X TB', 'TB', 'KV'],
      dtype=object)
```

[ ]:
```
productline.shape
```

[ ]:
```
(192,)
```

[ ]:
```
# To see on how much unique values allocated in ProductNumber feature
ProductNumber = new_df['ProductNumber'].unique()
ProductNumber
```

[ ]:
```
array(['ZG229AV', 'Z9Y75AV', 'Z9R42AA#UUF', …, '1AB35AV', '1AB34AV',
       '1AB33AV'], dtype=object)
```

[ ]:
```
ProductNumber.shape
```

[ ]:
```
(1900,)
```

[ ]:
```
# To see on how much unique values allocated in PurchaseOrderNo feature
PurchaseOrderNo = new_df['PurchaseOrderNo'].unique()
PurchaseOrderNo
```

[ ]:
```
array(['PO 94464', 'JV2017/000058', 'SCP0748376', 'SCP0820594',
       'SCP0876647', '2017-11-114126', 'SCP0786091', 'SCP0755717',
       'SCP0760227', 'SCP0755719', 'FRONTPOS/SIMPANGEMPAT',
       'MY20180416-Muar', 'TGFADMGPCxyp2190318', 'MY20180416-BtGajah',
       'BACKEND/SIMPANGEMPAT', 'PO5009844', 'PO500983', '1803-020',
       'SCP0808462', 'PA58496ATM', 'SCP0876621', 'LF35108ATM',
       'SCP0847081', 'SCP0874275', 'SCP0874264', '4500834183',
       'SCP0834216', 'SCP0855537', 'SCP0757551', 'SCP0851835',
       'SCP0854230', 'SCP0854263', 'SCP0854259', 'SCP0854226',
       'SCP0854258', 'SCP0768509', 'SCP0756975', 'SCP0761712',
       'SCP0764661', 'PO1707-0050', 'SCP0766012', 'SCP0827436',
       'SCP0820959', 'SCP0835682', 'SCP0818061', 'SCP0837307',
       'SAP018063049', 'SCP0870225', 'SCP0757024', 'P098393',
       'SCP0857586', 'SCP0870974', 'SCP0867246', 'SCP0808929', 'POC18814',
       'SCP0870973', 'SCP0809615', 'SCP0871288', 'FMM/FA/1718/014',
       'C18842', 'SRO/FMM/1718/01', 'MY20171206-FrontPOS',
       'MY20171206-FRONTPOS', 'SCP0764554', 'SCP0768186', 'SCP0769217',
       'PO42115', '17001832 XN', 'TCMB/171212', 'SCP0836704', 'SG 1084',
       'RRI/01/10/17/MANDY', 'HQS/03171', 'POEI21711101', 'POEI21802125',
       'POEI21712093', 'OPSPO-37784', 'PORRI/05/12/17', 'PO26496',
       'PO11917', 'PO11181', 'PO11685', 'PO11744', 'SCP0826427',
       '5203/2018', '3803/2018', 'PO12594', 'PO12610', 'PA59272ATM',
       'SCP0806807', 'SCP0849395', 'SCP0842713', 'PO00210035',
       'SCP0859920', 'LF33703ATM', 'PA57576ATM', 'LF33569ATM',
       'PA58446ATM', 'PA58334ATM', 'PO28653', 'PA60752ATM', 'SCP0866068',
       '004/0618MLKIT', 'POML-201804-007', 'POML-201804-042',
       'MY20180416', 'MY20180308', 'SCP0873185', 'SCP0844347',
       'SCP0851065', 'SCP0859043', 'SCP0811227', 'PO0007560', 'PO180502',
       'PO180401', 'LF33629ATM', 'PA58511ATM', 'PO24028', '2017-11114126',
       'Sonoco-20171011-Elitebook 745', 'Sonoco-20170721-xyp EB 745 G4',
       'Sonoco-20170706-xyp EB 745 G4', 'SONOCO-20170721-xyp EB 745 G4',
       'SONOCO-20170706-xyp EB 745 G4', 'SCP0752563', 'SCP0758830',
```

'SCP0766957', 'SCP0759221', 'SCP0767095', 'SCP0767202',
'SCP0767128', 'SCP0773558', 'SCP0759902', 'SCP0760826',
'SCP0774848', 'SCP0762561', 'SCP0761751', 'SCP0771451',
'SCP0758860', 'SCP0776671', 'SCP0773509', 'SCP0758799',
'SCP0761107', 'SCP0764936', 'SCP0806346', 'SCP0854778',
'PO20180406-004', 'SCP0873078', 'SCP0869840', 'SCP0768857',
'SCP0752186', '5438598332116666MYS1', 'SCP0861165', 'SCP0861176',
'SAP018073141', 'SCP0853301', 'SCP0851245', '46186', '48561',
'PO2018346', 'SCP0806312', 'SCP0820086', 'SCP0810144',
'SCP0871137', 'PO2018-05-1188851', 'SCP0765146', 'SCP0825221',
'SCP0815746', 'PO3423ITEC', 'PO3441', 'PSB3412', 'PO1100052732',
'ISM2018023', 'ISM2018/008', 'PO1100055395', 'MY20180302',
'SCP0828148', 'SCP0819281', 'SCP0871931', 'YxypO2018/007',
'SCP0830695', 'SCP0811203', 'SCP0824199', '1016/2018',
'SCP0811217', 'SCP0811207', '46185', 'SCP0811226',
'2017-11-113314', 'SCP0840852', 'YxypO2018/053', '3703/2018',
'D/MY/PTS/SHISEIDO/02052018-1', '4700012435', 'PO-1503', 'PO-1504',
'MY20180427', 'SCP0766940', 'SCP0773219', 'POCSI1895',
'RRI/05/11/17/mandy', 'SCP0810799', 'PO2305', 'PO0007143',
'P17120015', 'POR0000403', 'SCP0871762', 'P18060000', '5.51E+11',
'P18060014', 'P18060005', 'SCP0871280', '8100059493',
'PO0124-SDIST', 'SCP0875688', 'P18040032', 'SCP0854779',
'P18040030', 'PO0910/AXS', 'CSI-2014-1', 'CSI-2015-2', 'P18040017',
'CSI-2015-1', '7100031938', '541524564896050MYS1', 'CSI-2032-1',
'7100031325', '2018-03-117702', 'P18040019', '2018-03-117753',
'P18040018', 'SCP0864394', 'PO00000986', 'POEI21806006',
'SCP0845978', 'PO3240', 'PO2970', 'PO2968', 'PO2463', 'PO00000239',
'POKM/2696/17/961', 'KL5/8289622', 'kl5/8289581', 'KL5/8287823',
'KL5', 'KL5/8287491', 'KL5/8287463', 'POML/201802/011', 'PO327880',
'2017-10-113134/1', 'MY016418', 'PO11750', 'PO26471', 'SCP0812769',
'SCP0805497', 'SCP0824632', 'SCP0858257', 'SCP0841130',
'SCP0758308', 'SCP0775367', 'PO1711/01', 'SCP0762119',
'SCP0815241', 'BGP017100131', 'BGP017100130', 'SCP0799058',
'SCP0762620', 'SCP0825879', 'SCP0779035', 'SCP0774525',
'SCP0768252', 'SCP0760249', 'SCP0763907', 'SCP0767793',
'SCP0768284', 'SCP0766881', 'SCP0767711', 'SCP0759140',
'SCP0767787', 'SCP0767078', 'SCP0767912', 'SCP0766964',
'SCP0765678', 'SCP0759719', 'SCP0765684', 'SCP0767941',
'SCP0767210', 'SCP0770809', 'SCP0769223', 'SCP0767727',
'SCP0771715', 'SCP0776661', 'SCP0767881', 'SCP0756750',
'SCP0768727', 'SCP0768263', 'SCP0767276', 'SCP0757641',
'SCP0767724', 'SCP0768085', 'SCP0768347', 'SCP0767211',
'SCP0767771', 'SCP0752604', 'SCP0752601', 'SCP0871759', 'PO28919',
'SCP0877866', 'PO29389', 'PO1100057473', 'SCP0849390',
'PO1100058422', 'MY20180619', 'PO00001485', 'SCP0859216',
'PO28976', 'TGFADMGPCxyp2180318', 'SCP0794092', 'SCP0755744',
'SCP0752565', 'SCP0770567', 'SCP0870979', 'SCP0825931', 'PO18820',

'PO11439', 'PO4501900021135', 'PO4501900021382', 'SCP0864348',
'SCP0864345', 'PO0011973', 'PO003000-1', 'TGFADMGPCxyp2210318',
'LF33715ATM', 'SCP0869375', 'SCP0874277', 'OPSPO-39963',
'3002/2018', 'OPSPO-40176', 'SCP0858261', 'SCP0820954',
'SCP0830727', '1006/2018', 'SCP0856490', 'SCP0856473',
'SCP0874259', 'SCP0876680', 'TGFADMGPCxyp2200318',
'SAL/D/MY/UNS/EFMB/210417-1', 'SCP0863878', 'SCP0820054',
'SCP0834748', '5383729253123253MYS4', '20180601-29637869',
'5383729253101907MYS3', '15611/2017', 'YxypO2018/072',
'SCP0816490', '10206/2018', '1611/2017', 'SCP0866745',
'SCP0878559', 'SCP0877833', 'SCP0858236', 'PO12466', 'PO00001046',
'POEI180002', 'PO0012922', 'PO0012427', 'PO00001131', 'PO00000126',
'PO00001159', 'PO00000107', 'PO00000981', 'PO0012797',
'SCP0857853', 'SCP0843649', 'SCP0843625', '61309', 'SCP0857860',
'SCP0760435', 'PO4501900021068', 'SCP0794644', 'SCP0870988',
'SCP0873628', 'SCP0870105', 'SCP0833285', 'SCP0844000',
'SCP0821068', 'SCP0843811', 'PO27905', 'SCP0749280', 'SCP0749132',
'SCP0747233', 'SCP0748245', 'SCP0748271', 'SCP0768734',
'SCP0768737', 'SCP0756015', 'SCP0753014', 'SCP0770791',
'SCP0777334', 'SCP0769222', '4500848753', 'SCP0749418',
'SCP0758520', 'SCP0760655', 'SCP0758895', 'SCP0841399', 'PO11914',
'SCP0819924', 'SCP0828421', 'PO11745', 'PO26402', 'SCP0870252',
'SCP0860355', 'SCP0815249', 'SCP0834758', 'SKTSB180029',
'SCP0871278', 'SCP0877254', 'SCP0847452', 'SCP0861195',
'YxypO2018/054', 'SCP0874708', 'SCP0861180', 'SCP0876668',
'SCP0824653', 'SCP0873775', 'MPO1806012', 'SCP0852027', 'JB12644',
'SCP0825171', 'SCP0854803', 'SCP0805307', 'SCP0816714',
'YxypO2017/138', 'SCP0853298', 'SCP0843617', 'SCP0841173',
'PA59298ATM', 'SCP0763857', 'SCP0751030', 'SCP0749373',
'SCP0749311', 'SCP0778163', 'SCP0750224', 'ISM2017/082',
'SCP0769236', 'SCP0769257', 'SCP0761240', 'PO171201', 'PO00209993',
'PO00209989', 'Ongmy20180119', 'ONGMY20180119',
'D/MY/PS/MNA/24012018/1', 'SCP0864343', 'SCP0778206', 'SCP0777333',
'PO001231', 'PO00210025', 'D/MY/PS/MNA/20032018/1', 'MY20180318',
'PO001187', 'PO27745', 'SCP0873778', 'SCP0824714', 'SCP0825890',
'SKTB170076', 'SCP0856640', 'SCP0827421', 'SCP0764782',
'SCP0821218', 'SCP0794647', 'SCP0767110', 'SCP0777382',
'SCP0763926', 'SCP0773315', 'SCP0761232', 'SCP0765263',
'SCP0756579', 'SCP0875698', 'SCP0828168', 'SCP0849305',
'SCP0851843', 'SCP0765133', 'SCP0756076', 'SCP0828354',
'SCP0806446', 'KL5/42035274', 'POKL5/42034759', 'PO/2303160',
'SCP0811387', 'SCP0767799', 'SONOCO-20170425-xyp EB 745 G3',
'SCP0748909', 'PO2303326', 'SCP0801184', 'SCP0830029',
'500/PO-01565', 'SCP0821093', 'SCP0866415', 'SCP0772305',
'SCP0760633', 'SCP0774268', 'SCP0786303', 'SCP0758754',
'PO50001118', 'PO7862', 'SCP0871899', 'SCP0871134', 'SCP0861171',
'MY20180321', 'SCP0849414', '1712/4863/2', '1805/5078',

'SCP0806823', 'P0001103', 'SCP0774220', 'SCP0865926', 'SCP0796381', '127545', 'SCP0786275', 'SCP0753012', 'SCP0756597', 'SCP0794651', 'SCP0860352', 'SCP0778208', 'SCP0772249', 'POEM180017', 'SCP0796512', 'SG1075', 'Hong Thai Travel-20170607', 'HONG THAI TRAVEL-20170607', 'SCP0869490', '17002032XN', 'PON006862', 'POC000173', 'POC000756', 'SCP0755662', 'SCP0766592', 'SCP0764960', 'SCP0758921', 'SCP0835976', 'SCP0812039', 'SCP0754676', 'SCP0774779', 'SCP0771406', 'SCP0755727', 'SCP0755755', 'SCP0760407', 'SCP0829179', 'PO180201', 'P180203', 'SCP0851636', 'PO00803', '1802/4963', '1804/5031', '1802/4975', '1803/4983', 'SCP0867863', 'PO51542', '4500825080', '4500851671', 'SCP0755127', 'PO-7704', 'SCP0754257', 'PA58268ATM', 'SCP0766323', 'PA58227ATM', 'P000001057', 'PA61158ATM', 'PA61515ATM', 'P000000127', 'PA61849ATM', 'PO60302ATM', 'PA58445ATM', 'P000210027', 'SCP0845957', 'SCP0773284', 'SCP0764002', 'PO51668', 'SCP0827438', 'SCP0854909', 'SCP0818045', 'SCP0807650', 'SCP0807423', 'SCP0808514', 'SCP0821045', 'SCP0847272', 'SCP0860504', 'SAPO17112148', 'EI170005', 'SCP0805313', 'SCP0853663', 'SCP0856928', 'SCP0850029', 'SCP0871489', 'SCP0823955', 'SCP0807748', 'SCP0835101', 'SCP0871062', 'SCP0870535', 'MY20180320', '8205-2018', 'SCP0833211', 'SCP0794090', 'SCP0795274', 'SCP0764858', 'SCP0803021', 'SCP0770724', 'SCP0772584', 'SCP0773241', 'SCP0773257', 'SCP0768659', 'SCP0768846', 'SCP0773254', 'SCP0773255', 'SCP0776311', 'SCP0767926', 'SCP0772241', 'SCP0775536', 'SCP0770778', 'SCP0771637', 'SCP0768645', 'SCP0777300', 'SCP0768881', 'SCP0768539', 'SCP0773297', 'SCP0768568', 'SCP0768522', 'SCP0768268', 'SCP0768574', 'SCP0773236', 'SCP0769293', 'SCP0776303', 'SCP0776314', 'SCP0762495', 'SCP0772165', 'SCP0770753', 'SCP0773229', 'SCP0770577', 'SCP0764803', 'SCP0775410', 'SCP0777952', 'SCP0778172', 'SCP0772451', 'SCP0773232', 'SCP0758840', 'SCP0770343', 'SCP0766550', 'SCP0768526', 'SCP0776298', 'SCP0753224', 'SCP0772665', 'SCP0753481', 'SCP0776307', 'SCP0773247', 'SCP0773281', 'SCP0769249', 'SCP0768259', 'SCP0757475', 'SCP0764013', 'SCP0802672', 'SCP0768650', 'SCP0773529', 'SCP0768511', 'SCP0829124', 'SCP0778158', 'SCP0759277', 'PO53689', 'KL5/8289500', 'PO1100057852', 'PO2302836', 'MY20170420-Petro', 'PO9018800069', 'SCP0861036', '71 33276', 'SCP0858738', 'SCP0848001', 'SCP0808466', 'SCP0805642', 'BGPO18010005', 'SCP0863650', 'SCP0863637', 'SCP0863653', 'SCP0863647', 'SCP0863651', 'SCP0863649', 'SCP0750640', 'SCP0863642', 'SCP0863646', 'SCP0863648', 'SCP0863636', 'SCP0863644', 'SCP0863645', 'SCP0872748', 'SCP0750856', 'PO50001254', 'PO1100053864', 'SCP0807760', 'SCP0859218', 'SCP0759096', 'SCP0861635', 'SCP0821397', 'SCP0808652', 'SCP0829937', 'SCP0873811', 'SCP0818003', 'SCP0843137', 'SCP0852395',

```
'SCP0834305', '2018/03/117014', 'SCP0759083', 'P00000399',
'SCP0826708', 'P0020/1117MLKIT', 'SCP0805468', 'SCP0805490',
'SCP0805477', 'P050001281', 'SCP0805513', 'SCP0816344',
'SCP0808588', 'P0000006/17', 'SCP0805487', 'SCP0808590',
'SCP0805478', 'SCP0805515', 'SCP0805517', 'P04501900021391',
'P076785', 'SCP0805469', 'SCP0805501', 'SCP0805514', 'SCP0805475',
'SCP0826309', 'SCP0805516', 'SCP0805509', 'SCP0805488',
'P0007 /1217MLKIT', 'SCP0805505', 'SCP0826711', 'SCP0807651',
'PO/000012/17', 'P076968', 'SCP0827370', 'P04501900021789',
'SCP0827459', 'SCP0805492', 'SCP0806445', 'SCP0805502',
'SCP0805498', 'P01709-0042', 'SCP0793720', 'SCP0793753',
'SCP0873978', 'SCP0848247', 'SCP0864715', 'SCP0848201',
'SCP0861202', 'SCP0819070', 'SCP0820298', 'SCP0875759', 'P0180603',
'4500818964', '4500839251', 'P0000024-17', 'SCP0817911',
'P0180601', '4500835342', '4500829176', 'POML-201806-',
'SCP0869846', 'SCP0809600', '4500843435', 'GS1/1718/020',
'SCP0828820', 'SCP0869842', 'GS1/1718/022', 'P077442',
'4500850071', 'PO-7732', '4500824367', '4500815070', 'PO-7758',
'4500848427', 'SCP0875687', 'SCP0871923', 'GS1/1718/030',
'PO-000024-17', 'PO/000015/17', 'SCP0842774', 'YxypO',
'SCP0849806', 'P077246', 'SCP0832710', 'SCP0849402', 'SCP0841351',
'FMM/1718/069', 'SCP0800047', '2DC-4507796493', 'SCP0856424',
'SCP0870359', 'Sonoco-20170425-xyp EB 745', 'SCP0758884',
'SCP0754678', 'SCP0754421', 'SCP0781510', 'SCP0754423',
'P01707-0020', 'SCP0754850', 'SCP0774078', 'SCP0754290',
'SCP0776319', 'SCP0754840', 'SCP0869488', 'SCP0846630',
'MY20180207', 'MY20180606', 'SCP0870957', 'MY20180413',
'MY20180508', 'P029071', 'PA59894ATM', 'P07915', 'PO-7877',
'SCP0751073', 'SCP0752851', 'SCP0758125', 'P01705-0065',
'SCP0754847', 'SCP0760058', 'SCP0844014', 'SCP0849394',
'SCP0829273', 'SCP0875758', 'SCP0864403', 'SCP0867390',
'SCP0878562', 'SCP0867296', 'SCP0877823', 'SCP0874977',
'SCP0876685', 'SCP0860817'], dtype=object)
```

[ ]:
```
PurchaseOrderNo.shape
```

[ ]:
```
(845,)
```

From the result above,we could not conclude that ProductLine, ProductNumber and PurchaseOrderNo are carrying the same information.

We will like to verify that PurchaseOrderNo has the same information as CustomerOrder and CustomerName.

[ ]:
```
customerOrder = new_df['CustomerOrder'].unique()
customerOrder
```

[ ]:
```
array(['N', 'Y'], dtype=object)
```

[ ]:
```
CustomerName = new_df['CustomerName'].unique()
CustomerName.shape
```

[ ]:
```
(409,)
```

They are not carrying the same information based on the unique result that we obtained.

We can start by eliminating the features that represent the unnecessary features in the dataset according to the objectives.

[ ]:
```
# remove the features that are not related to the objectives
sales = new_df.drop(['InvoiceStatus', 'InvoiceToAddr',␣
 ↪'ShipToAddr5','SalesRep', 'Status', 'xypOrderNo'], axis = 1)
sales
```

[ ]:
```
      PurchaseOrderDate  … CustomerOrder
0             15/5/2017  …             N
1             15/5/2017  …             N
2              5/1/2018  …             N
3              5/1/2018  …             N
4              5/1/2018  …             N
…                   …  …             …
59373         22/8/2017  …             Y
59374         22/8/2017  …             Y
59375         22/8/2017  …             Y
59376         22/8/2017  …             Y
59377         22/8/2017  …             Y

[59378 rows x 11 columns]
```

[ ]:
```
sales.shape
```

[ ]:
```
(59378, 11)
```

## 0.5 4. Data Transformation

Data Transformation is the process of converting data from one format to another.

[ ]:
```
#Converting the PurchaseOrderDate to DataTime format

sales["PurchaseOrderDate"] = pd.to_datetime(sales["PurchaseOrderDate"])
sales
```

[ ]:
```
      PurchaseOrderDate  … CustomerOrder
0            2017-05-15  …             N
1            2017-05-15  …             N
2            2018-05-01  …             N
```

```
3           2018-05-01  …          N
4           2018-05-01  …          N
…               …  …        …
59373       2017-08-22  …          Y
59374       2017-08-22  …          Y
59375       2017-08-22  …          Y
59376       2017-08-22  …          Y
59377       2017-08-22  …          Y

[59378 rows x 11 columns]
```

[ ]:
```
sales.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 59378 entries, 0 to 59377
Data columns (total 11 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   PurchaseOrderDate   59378 non-null  datetime64[ns]
 1   PurchaseOrderNo     59378 non-null  object
 2   ProductLines        59378 non-null  object
 3   ProductNumber       59378 non-null  object
 4   NetLineDollarPrice  59378 non-null  float64
 5   NetOrderDollarPrice 59378 non-null  float64
 6   OrderedQuantity     59378 non-null  int64
 7   WebOrderNo          59378 non-null  object
 8   CustomerName        59378 non-null  object
 9   ShipToAddr2         59378 non-null  object
 10  CustomerOrder       59378 non-null  object
dtypes: datetime64[ns](1), float64(2), int64(1), object(7)
memory usage: 5.0+ MB
```

## 0.6 5. Exploratory Data Analysis (EDA)

EDA involves with the process of getting insights from the dataset and perform descriptive statistics.

[ ]:
```
sales.describe()
```

[ ]:

| | NetLineDollarPrice | NetOrderDollarPrice | OrderedQuantity |
|---|---|---|---|
| count | 59378.000000 | 59378.000000 | 59378.000000 |
| mean | 167.960680 | 2601.289978 | 4.425107 |
| std | 1778.386717 | 10090.008458 | 10.974984 |
| min | -78138.000000 | -78926.950000 | 1.000000 |
| 25% | 0.000000 | 213.270000 | 1.000000 |
| 50% | 0.000000 | 1022.590000 | 1.000000 |
| 75% | 0.000000 | 2512.480000 | 3.000000 |
| max | 78138.000000 | 78926.950000 | 600.000000 |

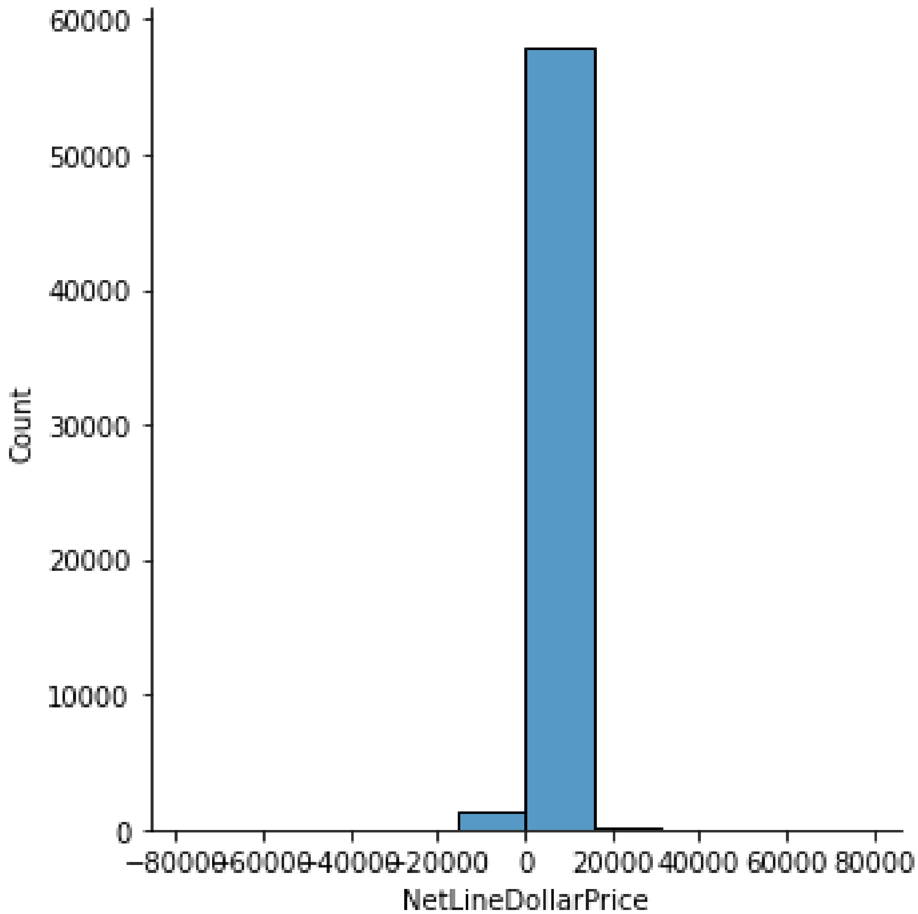
We can observe from the above table that:

- The maximum dollar price from the order net profit is around 78,927 dollars and the maximum dollar price from the product net profit is around 78, 138 dollars. To conclude, order net profit has a higher amount than the net profit obtained from the Line.
- The highest amount of ordered quantity would be 600 items.

**i. Continuous Variable Distribution**

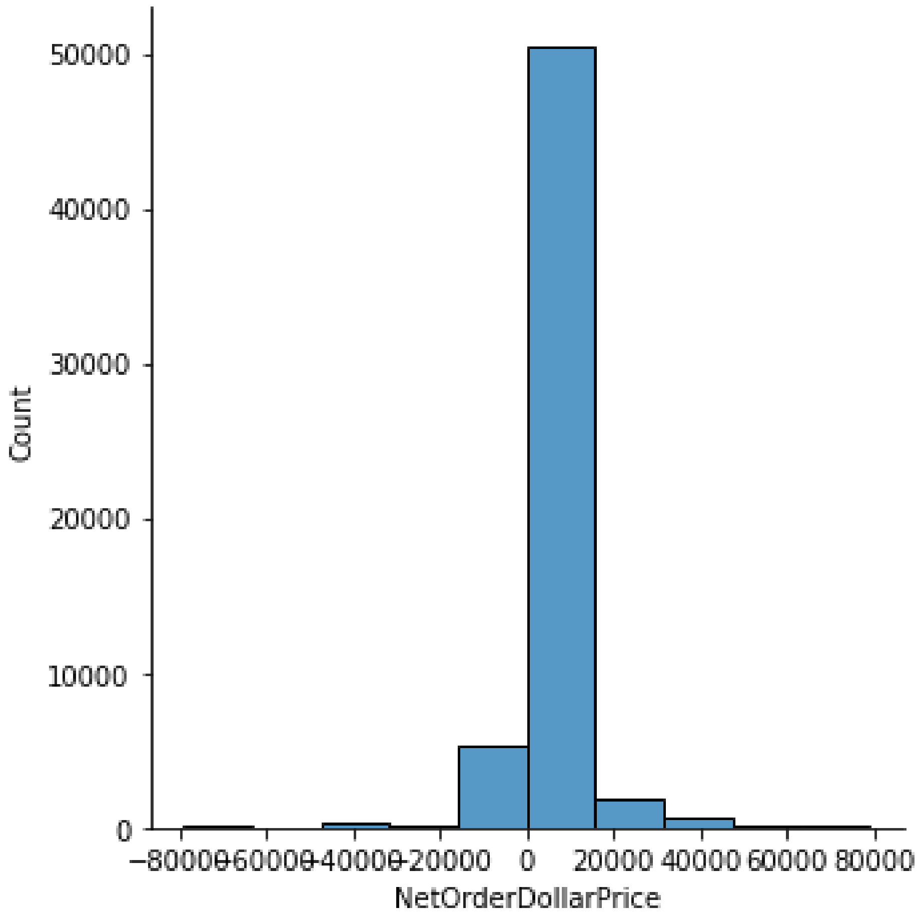
[ ]:
```python
sns.displot(sales, x="NetLineDollarPrice", bins=10)
```

[ ]: <seaborn.axisgrid.FacetGrid at 0x7f006b9936a0>



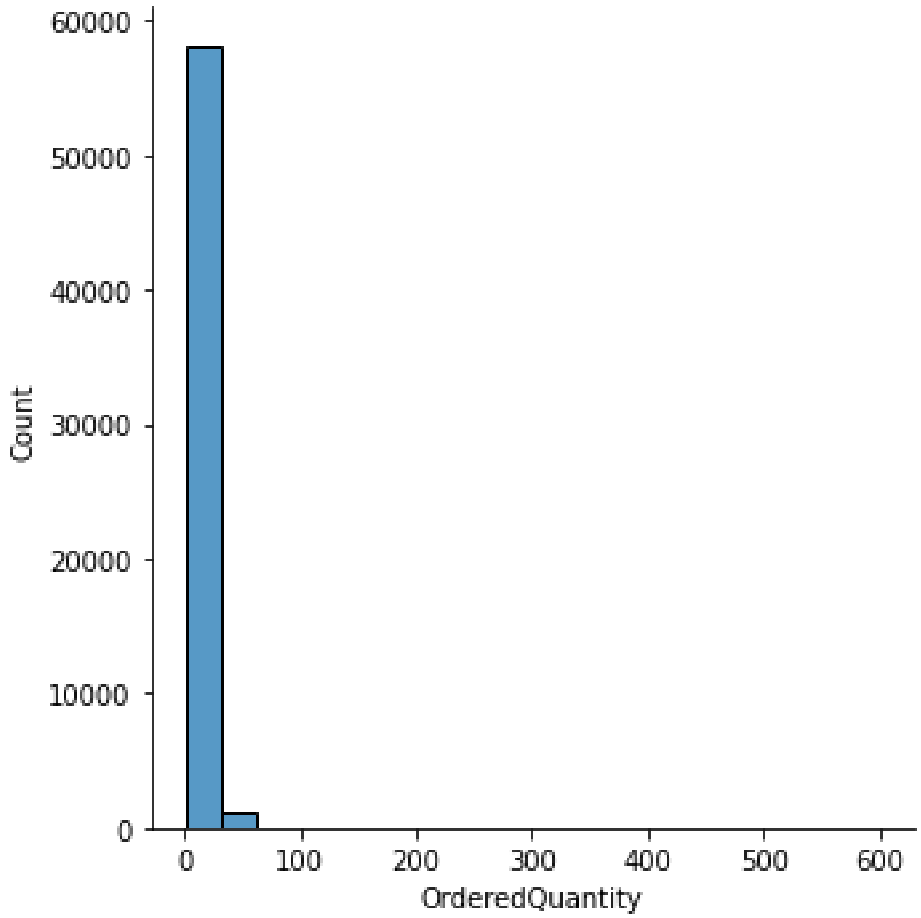

[ ]:
```python
sns.displot(sales, x="NetOrderDollarPrice", bins=10)
```

[ ]: <seaborn.axisgrid.FacetGrid at 0x7f006b988198>




[ ]:
```
sns.displot(sales, x="OrderedQuantity", bins=20)
```

[ ]: <seaborn.axisgrid.FacetGrid at 0x7f006b9885c0>




**ii. Datewise Analysis** Datewise analysis is used to analyze the trends of NetLineDollarPrice, NetOrderDollarPrice and OrderedQuantity according to time.

[ ]:
```
# Summation of the continuous variables inside the latest dataset
# A new variable called datawise stores the data regarding the continous␣
↪variables based on the purchase date
datewise=sales.groupby(["PurchaseOrderDate"]).agg({"NetLineDollarPrice":'sum',␣
↪"NetOrderDollarPrice":'sum', "OrderedQuantity":'sum'})
datewise
```

[ ]:

| | NetLineDollarPrice | NetOrderDollarPrice | OrderedQuantity |
|---|---|---|---|
| PurchaseOrderDate | | | |
| 2017-01-11 | 17269.02 | 246944.44 | 300 |
| 2017-02-05 | 399351.66 | 599379.22 | 1228 |
| 2017-02-06 | 20275.82 | 46776.18 | 68 |
| 2017-02-11 | 59290.18 | 941287.16 | 1418 |
| 2017-03-05 | 4526.70 | 16491.62 | 26 |

```
…                                 …             …             …
2018-12-01                 86156.30    1427557.68          5414
2018-12-02                  8819.16     313080.18           304
2018-12-03                 -5180.30    -122037.14           124
2018-12-04                 16148.92     428181.72           334
2018-12-06                 27661.16     293714.86           436

[232 rows x 3 columns]
```

**Trend of NetLineDollarPrice**

[ ]:
```
import plotly.graph_objs as go
fig = go.Figure()
fig.add_trace(go.Scatter(x=datewise.index, y=datewise["NetLineDollarPrice"],
                         mode='lines+markers', name='NetLineDollarPrice'))
fig.update_layout(title="The trend of NetLineDollarPrice",
                  xaxis_title="Date", yaxis_title="NetLineDollarPrice")
fig.show()
```

From the graph plotted above, we can see that:

- the highest net profit obtained from Line (productLine) are 486.73K on 14th of December 2017.
- The NetLineDollarPrice does not have a clear trend.
- The trend also does not prove that the profits obtained are based on seasons.
- The profit might only spikes due to the a sudden demand from the customers when it is neccesary for their companies.

**Trend of NetOrderDollarPrice**

[ ]:
```
fig1 = go.Figure()
fig1.add_trace(go.Scatter(x=datewise.index, y=datewise["NetOrderDollarPrice"],
                         mode='lines+markers', name='NetOrderDollarPrice'))
fig1.update_layout(title="The trend of NetOrderDollarPrice",
                  xaxis_title="Date", yaxis_title="NetOrderDollarPrice")
fig1.show()
```

From the graph we can observe that:

- The highest net profit obtained by orders are 8,303K dollars on 21st June 2018.
- The NetOrderDollarPrice also does not have a clear trend.
- However, the IT sales department almost make a constant net profit from 6th December 2017 to 21st December 2017 and also from 5th April 2018 to 27th April 2018.

**Trend of OrderedQuantity**

```
fig1 = go.Figure()
fig1.add_trace(go.Scatter(x=datewise.index, y=datewise["OrderedQuantity"],
                         mode='lines+markers', name='OrderedQuantity'))
fig1.update_layout(title="The trend of OrderedQuantity",
                 xaxis_title="Date", yaxis_title="OrderedQuantity")
fig1.show()
```

From the graph above, we can observe that:

- The highest order quantity received by the IT Sales department will be 28,682 orders on 12th December 2017. That was 2 days before the the second highest net profit obtained on 14th December 2017. That period will be a success of cash deposits from the IT Sales department.
- However, 21st Jun 2018 was the highest date receiving net profit from the order. We can assume that there might be some late payments occurred.

**To make a comparison between the NetLineDollarPrice and NetOrderDollarPrice**

```
fig2 = go.Figure()
fig2.add_trace(go.Scatter(x=datewise.index, y=datewise["NetLineDollarPrice"],
                         mode='lines+markers', name='NetLineDollarPrice'))
fig2.add_trace(go.Scatter(x=datewise.index, y=datewise["NetOrderDollarPrice"],
                         mode='lines+markers', name='NetOrderDollarPrice'))
fig2.update_layout(title="The trend of NetlineOrderDollarPrice and␣
 ↪NetOrderDollarPrice",
                 xaxis_title="Date", yaxis_title="NetLineOrderPrice and␣
 ↪NetOrderDollarPrice")
fig2.show()
```

From the graph above we definitely can conclude that the IT Sales department has gained for profit from the orders rather than from the product line services.

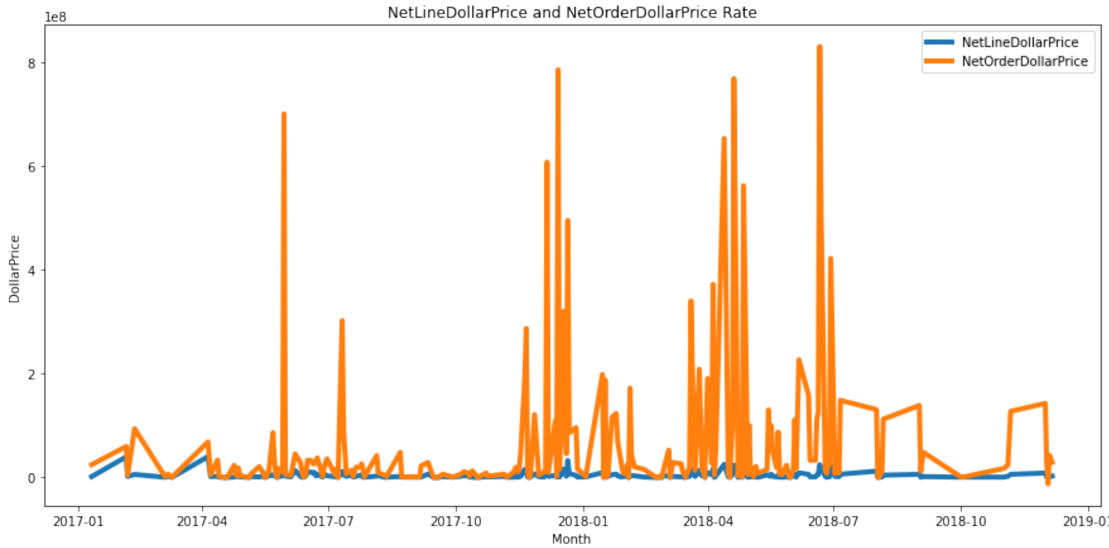
The rate of increment for both NetLineDollarPrice and NetOrderDollarPrice.

```
datewise["NetLineDollarPrice Rate"] =(datewise["NetLineDollarPrice"])*100
datewise["NetOrderDollarPrice Rate"] = (datewise["NetOrderDollarPrice"])*100

print("The average NetLineDollarPrice Rate: ", datewise["NetLineDollarPrice␣
 ↪Rate"].mean())
print("The average NetOrderDollarPrice Rate: ", datewise["NetOrderDollarPrice␣
 ↪Rate"].mean())
```

```
The average NetLineDollarPrice Rate:  4298779.862068965
The average NetOrderDollarPrice Rate:  66577325.99137922
```

```
plt.figure(figsize=(15,7))
plt.plot(datewise.index,datewise["NetLineDollarPrice Rate"], linewidth=4)
plt.plot(datewise.index,datewise["NetOrderDollarPrice Rate"], linewidth=4)
plt.legend(['NetLineDollarPrice', 'NetOrderDollarPrice'])
```

```python
plt.title('NetLineDollarPrice and NetOrderDollarPrice Rate')
plt.xlabel('Month')
plt.ylabel('DollarPrice')
plt.show()
```



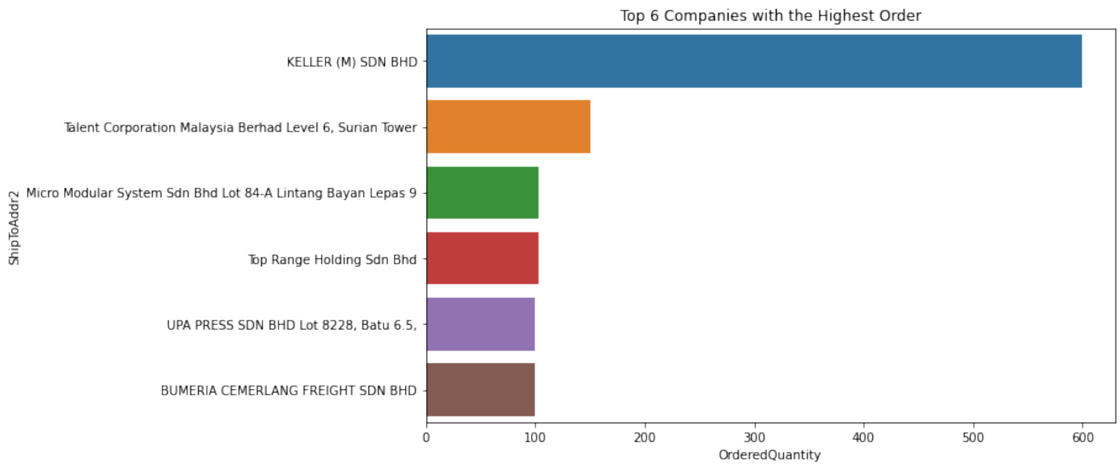

From the graph above we can see that the rate of increment for NetOrderDollarPrice was on July 2018.

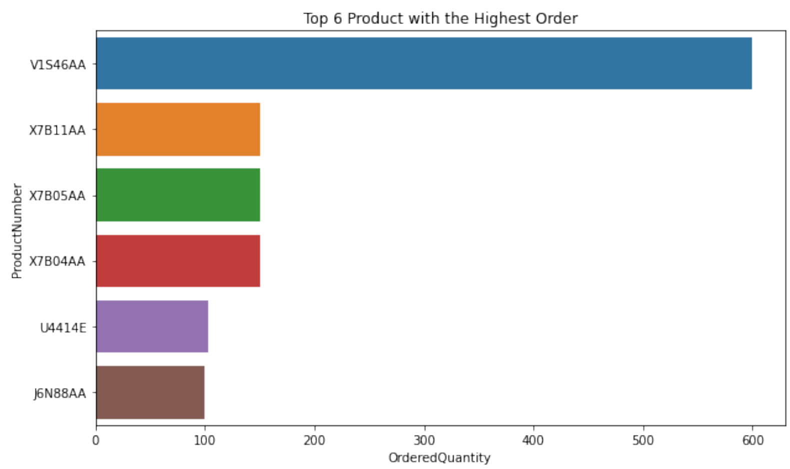
### iii. Categorical Variable Distribution

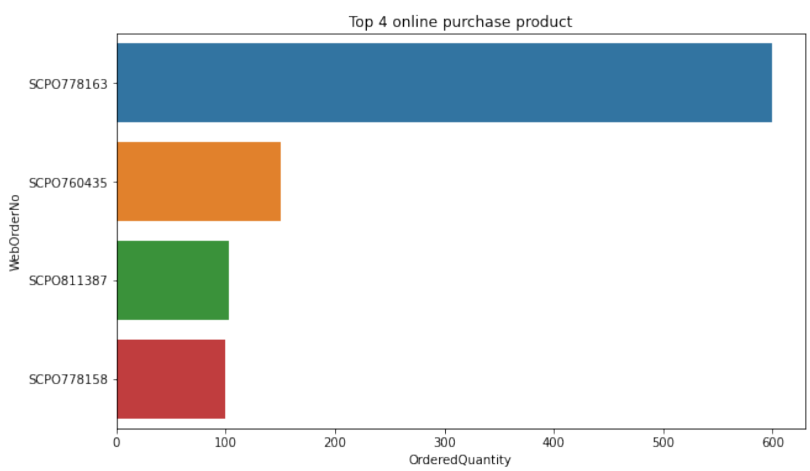
[ ]:
```python
Top_6Confirmed = sales.sort_values(["OrderedQuantity"], ascending=False).
    ↪head(30)
CompaniesNames = sales["ShipToAddr2"]
```

[ ]:
```python
#Horizontal bar
fig, (ax1, ax2, ax3) = plt.subplots(3, 1,figsize=(10,20))
sns.
    ↪barplot(x=Top_6Confirmed["OrderedQuantity"],y=Top_6Confirmed["ShipToAddr2"],ax=ax1)
ax1.set_title("Top 6 Companies with the Highest Order")
sns.
    ↪barplot(x=Top_6Confirmed["OrderedQuantity"],y=Top_6Confirmed["ProductNumber"],ax=ax2)
ax2.set_title("Top 6 Product with the Highest Order")
sns.
    ↪barplot(x=Top_6Confirmed["OrderedQuantity"],y=Top_6Confirmed["WebOrderNo"],ax=ax3)
ax3.set_title("Top 4 online purchase product")
```

[ ]:
```
Text(0.5, 1.0, 'Top 4 online purchase product')
```




From the horizontal Bar Chart above, we can observe that:

- The top 6 Companies with the highest order throughout the record are Keller (M) Sdn Bhd, Talent Corporation Malaysia Berhad, Micro Modular System Sdn Bhd, Top Range Holding Sdn bhd, UPA PRESS Sdn Bhd and Bumeria Cemerlang Freight Sdn Bhd.

- The top 6 products with the highest number of purchases are V1546AA, X7B11AA, X7B05AA, X7B04AA, U3314E and J6N88AA.
- Top 4 Online Purchase Product made are SCP0778163, SCPo760760435, SCP0811387 and SCPO778158.

**iv. Hypothesis Testing** From the dataset obtained, can we say that the net profit from orders are decreased in 2019?

We will create a null hypothesis and alternative hypothesis and determine whether to reject or accept the null hypothesis according to the p-value.

---

- Null Hypothesis, Ho = The net profit from the orders are decreased in 2019.
- Alternative Hypothesis, Ha = The net profit from the orders are increased in 2019.

[ ]:
```
NetOrder  = sales.groupby(["PurchaseOrderDate"]).agg({"NetLineDollarPrice":
 ↪'sum'})
NetOrder
```

[ ]:
```
                   NetLineDollarPrice
PurchaseOrderDate
2017-01-11                   17269.02
2017-02-05                  399351.66
2017-02-06                   20275.82
2017-02-11                   59290.18
2017-03-05                    4526.70
…                                   …
2018-12-01                   86156.30
2018-12-02                    8819.16
2018-12-03                   -5180.30
2018-12-04                   16148.92
2018-12-06                   27661.16

[232 rows x 1 columns]
```

[127]:
```
NetOrder['NetLineDollarPrice'].mean()
```

[127]:
```
42987.79862068966
```

In this case the mean is known.

[131]:
```
import statistics

statistics.stdev(NetOrder['NetLineDollarPrice'])
```

[131]:
```
67565.38247864606
```

[ ]:
```
sample_data = NetOrder.NetLineDollarPrice
sample_data[:5]
```

[ ]:
```
PurchaseOrderDate
2017-01-11     17269.02
2017-02-05    399351.66
2017-02-06     20275.82
2017-02-11     59290.18
2017-03-05      4526.70
Name: NetLineDollarPrice, dtype: float64
```

[130]:
```
sample_data.shape
```

[130]:
```
(232,)
```

[137]:
```
from scipy.stats import norm
# H0: mu = 42987
# Ha: mu < 42987

mu_0 = 42987
x_bar = 59378
n = 232
sigma = 67565
p_value =  norm.cdf(x_bar, mu_0, sigma)

alpha = .05


mu_0=42987
x_bar = np.mean(sample_data)
print('x_bar = ', x_bar)
s = np.std(sample_data, ddof=1)
print('s = ', s)
n = len(sample_data)
print('n = ', n)
t_score = (x_bar - mu_0)/(s/(n**.5))
print('t_score = ', t_score)
p_value = t.cdf(t_score, df = n-1)
print('p_value = ', p_value)
if p_value < alpha:
    print("\np_value = {}, Reject the null hypothesis in favour of the␣
↪alternative that the mean\
    of net profit for orders increased in 2019 comparing that in 2018 and 2017.".
↪format(round(p_value, 3)))
else:
```

```
    print("\np_value = {}, CANNOT Reject the null hypothesis. Therefore, there␣
↪is not strong enough evidence that the net of profit for orders is higher in␣
↪2019 comparing that in 2017 and 2018.".format(round(p_value, 3)))
```

```
x_bar =  42987.79862068966
s =  67565.38247864608
n =  232
t_score =  0.00018003639576890496
p_value =  0.5000717464404413

p_value = 0.5, CANNOT Reject the null hypothesis. Therefore, there is not strong
enough evidence that the net of profit for orders is higher in 2019 comparing
that in 2017 and 2018.
```

- When the p-value is <0.05, we can reject the null hypothesis.
- When the p-value is >0.05, we cannot reject the null hypothesis.

Thus, there is a probability that the net of profits for Order could decrease in 2019.

## 0.7 6. Modelling : Supervised Learning

**6.1 Time Series: ARIMA Model** ARIMA is an acronym that stands for AutoRegressive Integrated Moving Average. It is a class of model that captures a suite of different standard temporal structures in time series data.

[ ]:
```
# Regression for NetLineDollarPrice
datewise[['NetOrderDollarPrice']].shape
```

[ ]:
```
(232, 1)
```

[ ]:
```
# split the dataset into train and test data
train=datewise.iloc[:int(datewise.shape[0]*0.95)]
valid=datewise.iloc[int(datewise.shape[0]*0.95):]
y_pred=valid.copy()
```

[ ]:
```
!pip3 install pyramid-arima
```

```
Requirement already satisfied: pyramid-arima in /usr/local/lib/python3.6/dist-
packages (0.9.0)
Requirement already satisfied: numpy>=1.10 in /usr/local/lib/python3.6/dist-
packages (from pyramid-arima) (1.19.4)
Requirement already satisfied: Cython>=0.23 in /usr/local/lib/python3.6/dist-
packages (from pyramid-arima) (0.29.21)
Requirement already satisfied: scikit-learn>=0.17 in
/usr/local/lib/python3.6/dist-packages (from pyramid-arima) (0.22.2.post1)
Requirement already satisfied: scipy>=0.9 in /usr/local/lib/python3.6/dist-
packages (from pyramid-arima) (1.4.1)
Requirement already satisfied: statsmodels>=0.9.0 in
```

```
/usr/local/lib/python3.6/dist-packages (from pyramid-arima) (0.11.1)
Requirement already satisfied: pandas>=0.19 in /usr/local/lib/python3.6/dist-
packages (from pyramid-arima) (1.1.5)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.6/dist-
packages (from scikit-learn>=0.17->pyramid-arima) (1.0.0)
Requirement already satisfied: patsy>=0.5 in /usr/local/lib/python3.6/dist-
packages (from statsmodels>=0.9.0->pyramid-arima) (0.5.1)
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.6/dist-
packages (from pandas>=0.19->pyramid-arima) (2018.9)
Requirement already satisfied: python-dateutil>=2.7.3 in
/usr/local/lib/python3.6/dist-packages (from pandas>=0.19->pyramid-arima)
(2.8.1)
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-packages
(from patsy>=0.5->statsmodels>=0.9.0->pyramid-arima) (1.15.0)
```

[ ]:
```
from pyramid.arima import auto_arima
model_arima= auto_arima(train["NetLineDollarPrice"],trace=True,␣
 ↪error_action='ignore', start_p=1,start_q=1,max_p=3,max_q=3,
                    suppress_warnings=True,stepwise=False,seasonal=False)
model_arima.fit(train["NetLineDollarPrice"])
```

```
Fit ARIMA: order=(1, 0, 1); AIC=5531.574, BIC=5545.148, Fit time=0.087 seconds
Fit ARIMA: order=(1, 0, 2); AIC=5531.741, BIC=5548.709, Fit time=0.163 seconds
Fit ARIMA: order=(1, 0, 3); AIC=5537.281, BIC=5557.642, Fit time=0.553 seconds
Fit ARIMA: order=(2, 0, 1); AIC=5531.997, BIC=5548.966, Fit time=0.152 seconds
Fit ARIMA: order=(2, 0, 2); AIC=5535.124, BIC=5555.486, Fit time=0.481 seconds
Fit ARIMA: order=(2, 0, 3); AIC=5530.519, BIC=5554.275, Fit time=0.632 seconds
Fit ARIMA: order=(3, 0, 1); AIC=5533.495, BIC=5553.857, Fit time=0.201 seconds
Fit ARIMA: order=(3, 0, 2); AIC=5530.324, BIC=5554.080, Fit time=0.328 seconds
Fit ARIMA: order=(3, 0, 3); AIC=5531.940, BIC=5559.089, Fit time=0.665 seconds
Total fit time: 3.268 seconds
```

[ ]:
```
ARIMA(callback=None, disp=0, maxiter=50, method=None, order=(3, 0, 2),
      out_of_sample_size=0, scoring='mse', scoring_args={}, seasonal_order=None,
      solver='lbfgs', start_params=None, suppress_warnings=True,
      transparams=True, trend='c')
```

[ ]:
```
prediction_arima=model_arima.predict(len(valid))
y_pred["ARIMA Model Prediction"]=prediction_arima
```

[ ]:
```
# visualization on the Train Data, validation Data and Prediction.

fig=go.Figure()
fig.add_trace(go.Scatter(x=train.index, y=train["NetLineDollarPrice"],
                    mode='lines+markers',name="Train Data for NetLineDollarPrice"))
fig.add_trace(go.Scatter(x=valid.index, y=valid["NetLineDollarPrice"],
```

```
                    mode='lines+markers',name="Validation Data for␣
 ↪NetLineDollarPrice",))
fig.add_trace(go.Scatter(x=valid.index, y=y_pred["ARIMA Model Prediction"],
                    mode='lines+markers',name="Prediction for␣
 ↪NetLineDollarPrice",))
fig.update_layout(title="NetLineDollarPrice ARIMA Model Prediction",
                ␣
 ↪xaxis_title="Date",yaxis_title="NetLineDollarPrice",legend=dict(x=0,y=1,traceorder="normal"
fig.show()
```

[ ]:
```
r2_score(valid["NetLineDollarPrice"], prediction_arima)
```

[ ]:
```
-0.4524048442917401
```

*Note: $R^2$ can be negative in some situations. In particular, if no intercept was included in the fit, $R^2$ can become negative. In these cases, the learned model is worse than the null model.*

**6.2 Classification by using KNN** K-Nearest Neighbour is a supervised learning algorithm that stores all available cases and classfies new cased based on similarity measures (e.g. distance functions).

We are using the KNN classification to classify whether the purchase was Customer Order (Y) or no (N).

[ ]:
```
X = sales[['NetOrderDollarPrice', 'NetLineDollarPrice', 'OrderedQuantity']]
y = sales['CustomerOrder']
```

[ ]:
```
X.head()
```

[ ]:
```
   NetOrderDollarPrice  NetLineDollarPrice  OrderedQuantity
0                 0.00                 0.0                1
1                 0.00                 0.0                1
2                 0.00                 0.0                6
3                 0.00                 0.0                6
4              6585.37                 0.0                6
```

[ ]:
```
X.shape
```

[ ]:
```
(59378, 3)
```

[ ]:
```
y.head()
```

[ ]:
```
0    N
1    N
2    N
3    N
4    N
```

```
Name: CustomerOrder, dtype: object
```

[ ]:
```
y.unique()
```

[ ]:
```
array(['N', 'Y'], dtype=object)
```

[ ]:
```
#split the dataset into train data and test data
```

[ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y,stratify=y, test_size␣
↪= 0.2, random_state=42)
```

[ ]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier

scaler = StandardScaler()

scaler.fit(X_train)

X_train_s = scaler.transform(X_train)
X_test_s = scaler.transform(X_test)

E_knn = KNeighborsClassifier(n_neighbors=3, p=2)
E_knn.fit(X_train_s, y_train)
print('Accuracy for k = 3 & Euclidian on train set: {} \n'.format(E_knn.
↪score(X_train_s, y_train)))
print('Accuracy for k = 3 & Euclidian on test data: {} \n'.format(E_knn.
↪score(X_test_s, y_test)))

M_knn = KNeighborsClassifier(n_neighbors=3, p=1)
M_knn.fit(X_train_s, y_train)
print('Accuracy for k = 3 & Manhatan on train set: {} \n'.format(M_knn.
↪score(X_train_s, y_train)))
print('Accuracy for k = 3 & Manhatan on test data: {} \n'.format(M_knn.
↪score(X_test_s, y_test)))
```

```
Accuracy for k = 3 & Euclidian on train set: 0.7430634499600017

Accuracy for k = 3 & Euclidian on test data: 0.7321488716739643

Accuracy for k = 3 & Manhatan on train set: 0.7430845017051914

Accuracy for k = 3 & Manhatan on test data: 0.7322330751094644
```

Now we will try to test the prediction for using KNN with Euclidean distance.

[ ]:
```
CustomerOrder_prediction = E_knn.predict([[2, 3.2, 5]])
print('Customer Order is '+CustomerOrder_prediction[0])
```

```
Customer Order is Y
```

Now we will try to test the prediction for using KNN with Manhattan distance.

```
[ ]: CustomerOrder_prediction = M_knn.predict([[2, 3.2, 5]])
     print('Customer Order is '+CustomerOrder_prediction[0])
```

```
Customer Order is Y
```

Check the accuracy score for both KNN(Euclidean Distance) and KNN(Manhattan Distance)

```
[ ]: #accuracy
     E_knn.score(X_test, y_test)
```

```
[ ]: 0.504546985517009
```

```
[ ]: knnmodel1 = E_knn.fit(X_train_s, y_train)
     Euclidean = knnmodel1.predict(X_test)
     Knn_acc= accuracy_score(y_test, Euclidean)
     Knn_acc
```

```
[ ]: 0.504546985517009
```

```
[ ]: M_knn.score(X_test, y_test)
```

```
[ ]: 0.5067362748400135
```

```
[ ]: knnmodel1 = M_knn.fit(X_train_s, y_train)
     Manhattan = knnmodel1.predict(X_test)
     Knn_acc= accuracy_score(y_test, Manhattan)
     Knn_acc
```

```
[ ]: 0.5067362748400135
```

**6.3 Classification using Decision trees** Classification tree is a method of splitting the dataset into multiple sets until a decision is made. We will also used classification tree to determine whether CustomerOrder is a Yes (Y) or No(N).

```
[ ]: from sklearn.tree import DecisionTreeClassifier
     tree = DecisionTreeClassifier(max_depth=2)
     tree.fit(X_train, y_train)

     print("Accuracy on training set: {:.3f}".format(tree.score(X_train, y_train)))
     print("Accuracy on test set: {:.3f}".format(tree.score(X_test, y_test)))
```

```
Accuracy on training set: 0.806
Accuracy on test set: 0.806
```

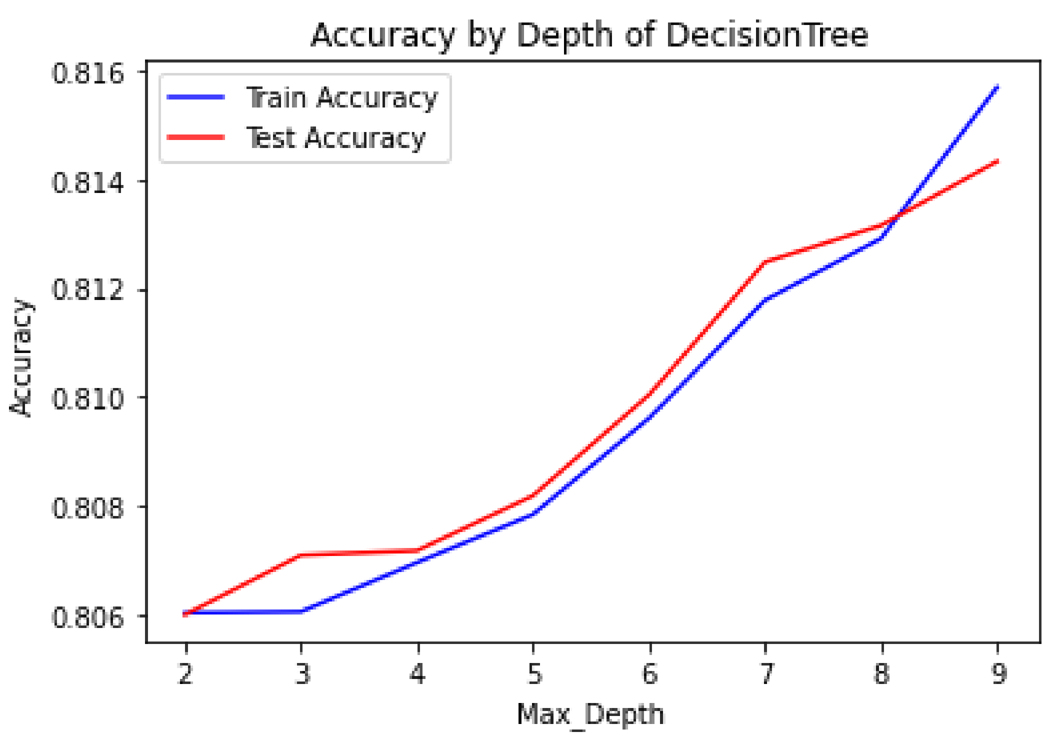
[ ]:
```
tree.score(X_test,y_test)
```

[ ]: 0.8059952846076119

[ ]:
```
rfc_Count = tree.predict(X_test)
accuracy_score(y_test, rfc_Count)
```

[ ]: 0.8059952846076119

[ ]:
```
train_acc = []
test_acc = []

for i in range(2,10):
    tree1 = DecisionTreeClassifier(max_depth=i, random_state=0)
    tree1.fit(X_train, y_train)

    train_acc.append(tree1.score(X_train, y_train))
    test_acc.append(tree1.score(X_test, y_test))

plt.plot(range (2,10),train_acc,'b-', label='Train Accuracy')
plt.plot(range (2,10),test_acc,'r-', label='Test Accuracy')
plt.xlabel('Max_Depth')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Accuracy by Depth of DecisionTree')
plt.show()
```



**6.4 Logistic Regression** Logistic Regression is also known as a binary classification. It impliments the sigmoid function to push the continuous values to its nearest binary value, either 1 or 0. The gradient descent is used for optimization by decreasing the cost function value.

Logistic Regression is also used to classify the CustomerOrder.

[ ]:
```
from sklearn.linear_model import LogisticRegression
logr = LogisticRegression()
logr.fit(X_train, y_train)
logr_y_pred = logr.predict(X_test)
print('Accuracy: ',logr.score(X_test, y_test))
```

```
Accuracy:  0.8058268777366117
```

[ ]:
```
# For test data
ytrain_pred = logr.predict(X_train)
```

[ ]:
```
# To see the accuracy of the train set and test set
from sklearn.metrics import accuracy_score

print('Accuracy the test set is: ', accuracy_score( y_test, logr_y_pred))
print('Accuracy the train set is: ', accuracy_score(y_train, ytrain_pred))
```

```
Accuracy the test set is:  0.8058268777366117
Accuracy the train set is:  0.8059871163319439
```

**6.5 Support Vector Machine (SVM)** SVM uses kernel trick technique to transform the data and then based on the transformations it finds an optimal boundary between the possible outputs.

[ ]:
```
from sklearn.svm import SVC # "Support vector classifier"
svcmodel = SVC(kernel='rbf')
svcmodel1= svcmodel.fit(X_train, y_train)
svm_Count = svcmodel1.predict(X_test)
```

[ ]:
```
SVM_acc= accuracy_score(y_test, svm_Count)
SVM_acc
```

[ ]:
```
0.8059952846076119
```

## 0.8 7. Evaluation Matric using Confusion Matrix

Confusion matrix is used to evaluate the performance of classification models.

Precision: It is all about what proportion of positive identifications was actually correct.

Recall: Sensitivity, is the proportion of the total amount of relevant instances that are actually retrieved.

F1-Score: Measures the test's accuracy.

[ ]:
```
from sklearn.metrics import classification_report
from sklearn import preprocessing
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import plot_confusion_matrix
import seaborn as sns
```

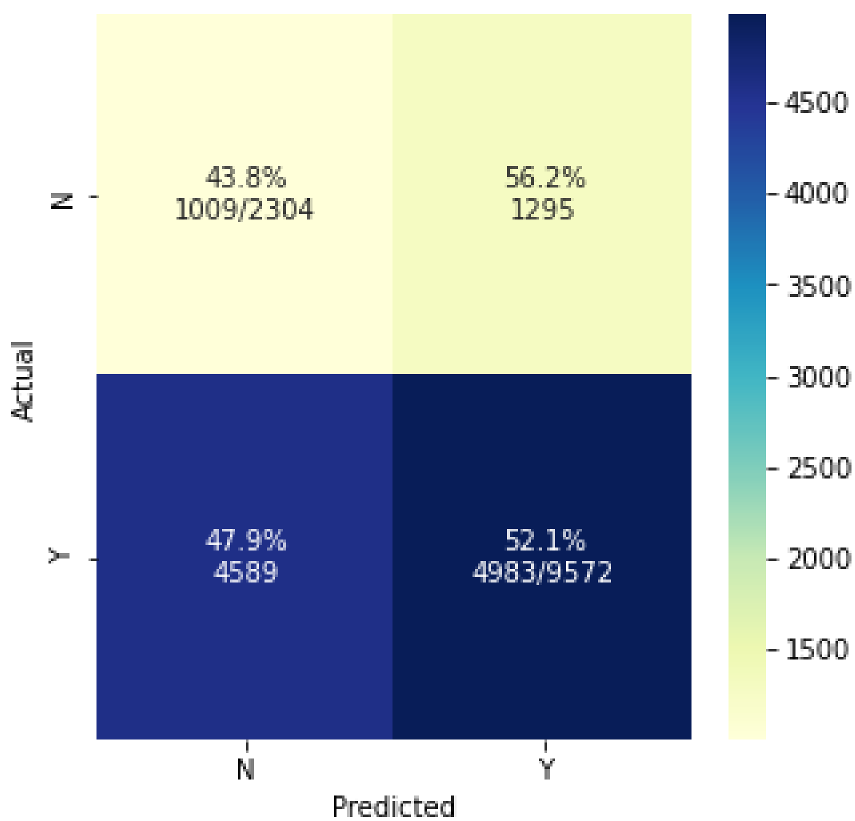
[ ]:
```
# define confusion matrix tables
def classi_report(prediction):
    lb = preprocessing.LabelBinarizer()
    lb.fit(df["CustomerOrder"])
     #### implemebt confusion matrix
    con_mat = pd.DataFrame(confusion_matrix(y_test, prediction), index=lb.
 ↪classes_, columns=lb.classes_)
    print('Confusion Matrix'.center(50) + '\n'*2 + str(con_mat) + '\n'*4)
    #### add classification_report
    print(classification_report(y_test, prediction, target_names=lb.classes_))
```

[ ]:
```
#define Heatmap the confustion matrix
def plot_cm(y_true, y_pred, figsize=(5,5)):
    cm = confusion_matrix(y_true, y_pred, labels=np.unique(y_test))
    cm_sum = np.sum(cm, axis=1, keepdims=True)
    cm_perc = cm / cm_sum.astype(float) * 100
    annot = np.empty_like(cm).astype(str)
    nrows, ncols = cm.shape
    for i in range(nrows):
        for j in range(ncols):
            c = cm[i, j]
            p = cm_perc[i, j]
            if i == j:
                s = cm_sum[i]
                annot[i, j] = '%.1f%%\n%d/%d' % (p, c, s)
            elif c == 0:
                annot[i, j] = ''
            else:
                annot[i, j] = '%.1f%%\n%d' % (p, c)
    cm = pd.DataFrame(cm, index=np.unique(y_true), columns=np.unique(y_true))
    cm.index.name = 'Actual'
    cm.columns.name = 'Predicted'
    fig, ax = plt.subplots(figsize=figsize)
    sns.heatmap(cm, cmap= "YlGnBu", annot=annot, fmt='', ax=ax)
```

### 0.8.1 Confusion Matrix for KNN Algorithm

[ ]:
```
classi_report(Euclidean)
plot_cm(y_test, Euclidean)
```

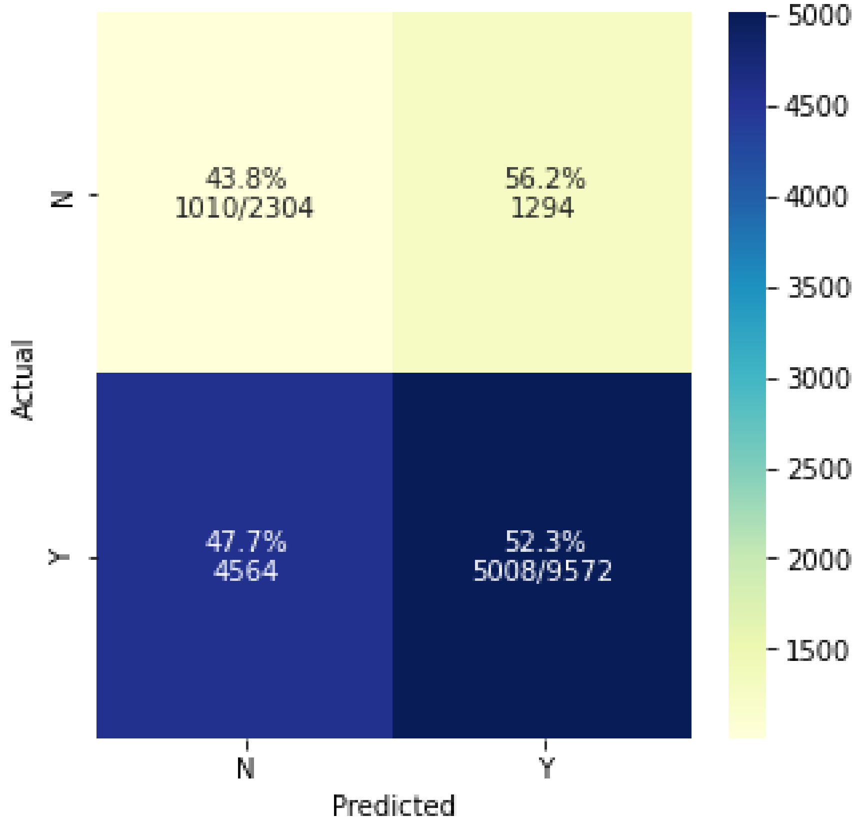
```
                  Confusion Matrix

      N     Y
N  1009  1295
Y  4589  4983




              precision    recall  f1-score   support

           N       0.18      0.44      0.26      2304
           Y       0.79      0.52      0.63      9572

    accuracy                           0.50     11876
   macro avg       0.49      0.48      0.44     11876
weighted avg       0.67      0.50      0.56     11876
```



F1-score shows an accuracy of 0.50.

```
[ ]: classi_report(Manhattan)
plot_cm(y_test, Manhattan)
```

```
                 Confusion Matrix

      N     Y
N  1010  1294
Y  4564  5008



              precision    recall  f1-score   support

           N       0.18      0.44      0.26      2304
           Y       0.79      0.52      0.63      9572

    accuracy                           0.51     11876
   macro avg       0.49      0.48      0.44     11876
weighted avg       0.68      0.51      0.56     11876
```




F1-score shows an accuracy of 0.51

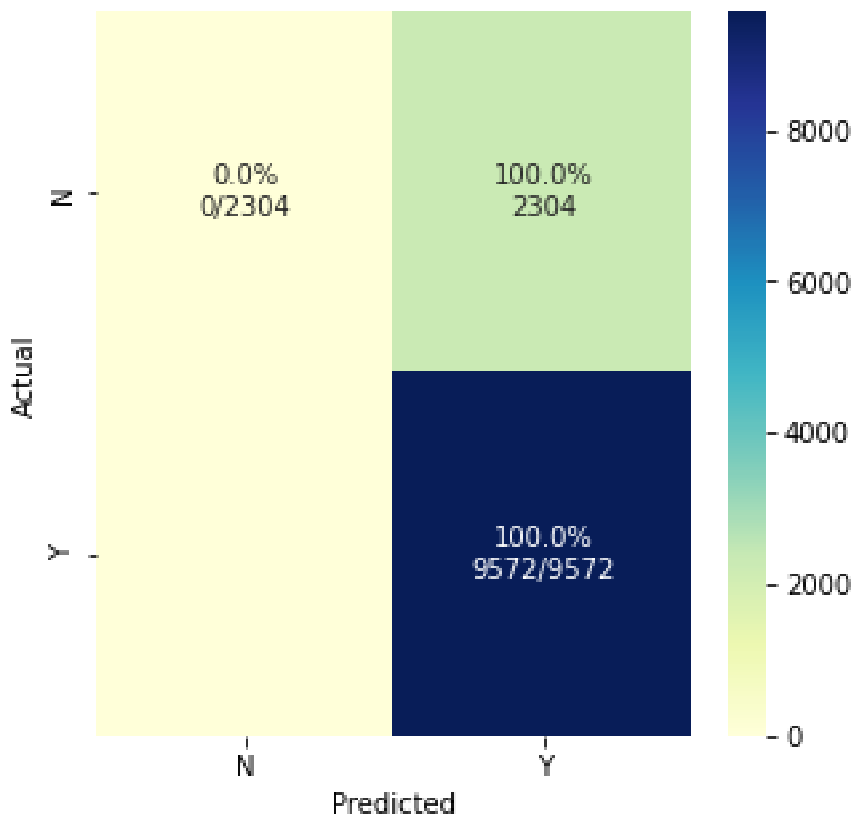
### 0.8.2 Confusion Matrix for Decision Tree

[ ]:
```
classi_report(rfc_Count)
plot_cm(y_test, rfc_Count)
```

```
                 Confusion Matrix

   N     Y
N  0  2304
Y  0  9572
```

```
/usr/local/lib/python3.7/site-packages/sklearn/metrics/_classification.py:1221:
UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no
```

```
predicted samples. Use `zero_division` parameter to control this behavior.

              precision    recall  f1-score   support

           N       0.00      0.00      0.00      2304
           Y       0.81      1.00      0.89      9572

    accuracy                           0.81     11876
   macro avg       0.40      0.50      0.45     11876
weighted avg       0.65      0.81      0.72     11876
```




This means that there is no F-score to calculate for this label, and thus the F-score for this case is considered to be 0.0.

When true positive + false positive == 0, precision returns 0 and raises UndefinedMetricWarning. This behavior can be modified with zero_division.

[ ]:
```
from sklearn.metrics import precision_score
precision_score(y_test, rfc_Count, average=None, zero_division=1)
```

[ ]: array([1.        , 0.80599528])

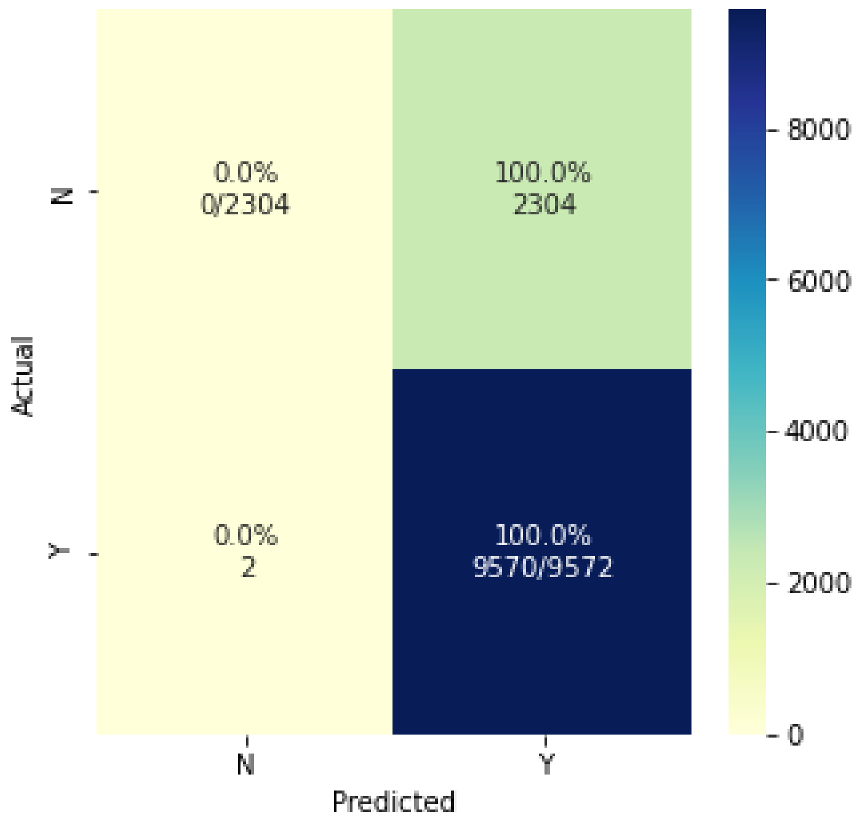
### 0.8.3 Confusion Matrix for Logistic Regression

```
[ ]: classi_report(logr_y_pred)
     plot_cm(y_test, logr_y_pred)
```

Confusion Matrix

| | N | Y |
|---|---|---|
| N | 0 | 2304 |
| Y | 2 | 9570 |

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| N | 0.00 | 0.00 | 0.00 | 2304 |
| Y | 0.81 | 1.00 | 0.89 | 9572 |
| accuracy | | | 0.81 | 11876 |
| macro avg | 0.40 | 0.50 | 0.45 | 11876 |
| weighted avg | 0.65 | 0.81 | 0.72 | 11876 |




This means that there is no F-score to calculate for this label, and thus the F-score for this case is considered to be 0.0.

[ ]:
```
precision_score(y_test, logr_y_pred, average=None, zero_division=1)
```

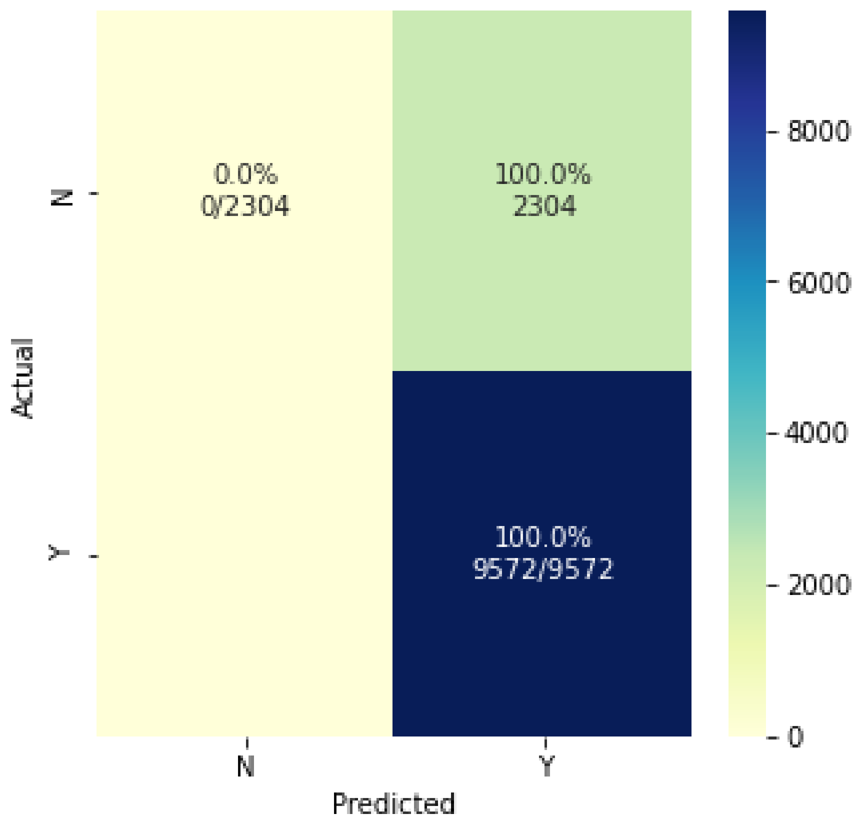
[ ]:
```
array([0.        , 0.80596261])
```

#### 0.8.4 Confusion Matrix for SVM

[ ]:
```
classi_report(svm_Count)
plot_cm(y_test, svm_Count)
```

```
                 Confusion Matrix

   N     Y
N  0  2304
Y  0  9572
```

```
/usr/local/lib/python3.7/site-packages/sklearn/metrics/_classification.py:1221:
UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.

              precision    recall  f1-score   support

           N       0.00      0.00      0.00      2304
           Y       0.81      1.00      0.89      9572

    accuracy                           0.81     11876
   macro avg       0.40      0.50      0.45     11876
weighted avg       0.65      0.81      0.72     11876
```




This means that there is no F-score to calculate for this label, and thus the F-score for this case is considered to be 0.0.

[ ]:
```
precision_score(y_test, svm_Count, average=None, zero_division=1)
```

[ ]:
```
array([1.        , 0.80599528])
```