# Load Dependencies and Configuration Settings

We started with the installation of the orange3 package through the command line, since it is not possible to include it through the usual procedure of adding custom packages in the Kernel.

In [ ]:

```
import os
import warnings
warnings.simplefilter(action = 'ignore', category=FutureWarning)
warnings.filterwarnings('ignore')
def ignore_warn(*args, **kwargs):
    pass

warnings.warn = ignore_warn #ignore annoying warning (from sklearn and seaborn)

import pandas as pd
import datetime
import math
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.mlab as mlab
import matplotlib.cm as cm

%matplotlib inline

from pandasql import sqldf
pysqldf = lambda q: sqldf(q, globals())

import seaborn as sns
sns.set(style="ticks", color_codes=True, font_scale=1.5)
color = sns.color_palette()
sns.set_style('darkgrid')

from mpl_toolkits.mplot3d import Axes3D

import plotly as py
import plotly.graph_objs as go
py.offline.init_notebook_mode()

from scipy import stats
from scipy.stats import skew, norm, probplot, boxcox
from sklearn import preprocessing
import math

from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_samples, silhouette_score

# import Orange
# from Orange.data import Domain, DiscreteVariable, ContinuousVariable
# from orangecontrib.associate.fpgrowth import *
```

# Load Dataset

In [ ]:

```
cs_df = pd.read_excel(io=r'../input/Online Retail.xlsx')
```

```
In [ ]: def rstr(df, pred=None):
            obs = df.shape[0]
            types = df.dtypes
            counts = df.apply(lambda x: x.count())
            uniques = df.apply(lambda x: [x.unique()])
            nulls = df.apply(lambda x: x.isnull().sum())
            distincts = df.apply(lambda x: x.unique().shape[0])
            missing_ration = (df.isnull().sum()/ obs) * 100
            skewness = df.skew()
            kurtosis = df.kurt()
            print('Data shape:', df.shape)

            if pred is None:
                cols = ['types', 'counts', 'distincts', 'nulls', 'missing ration', 'uniques
                str = pd.concat([types, counts, distincts, nulls, missing_ration, uniques,

            else:
                corr = df.corr()[pred]
                str = pd.concat([types, counts, distincts, nulls, missing_ration, uniques,
                corr_col = 'corr '  + pred
                cols = ['types', 'counts', 'distincts', 'nulls', 'missing ration', 'uniques

            str.columns = cols
            dtypes = str.types.value_counts()
            print('___________________________\nData types:\n',str.types.value_counts())
            print('___________________________')
            return str

        details = rstr(cs_df)
        display(details.sort_values(by='missing ration', ascending=False))
```

```
Data shape: (541909, 8)
___________________________
Data types:
 object            4
float64           2
int64             1
datetime64[ns]    1
Name: types, dtype: int64
___________________________
```

| | types | counts | distincts | nulls | missing ration | uniques | skewness | ku |
|---|---|---|---|---|---|---|---|---|
| **CustomerID** | float64 | 406829 | 4373 | 135080 | 24.926694 | [[17850.0, 13047.0, 12583.0, 13748.0, 15100.0,... | 0.029835 | -1.1 |
| **Description** | object | 540455 | 4224 | 1454 | 0.268311 | [[WHITE HANGING HEART T-LIGHT HOLDER, WHITE ME... | NaN | |
| **Country** | object | 541909 | 38 | 0 | 0.000000 | [[United Kingdom, France, Australia, Netherlan... | NaN | |
| **InvoiceDate** | datetime64[ns] | 541909 | 23260 | 0 | 0.000000 | [[2010-12-01 08:26:00, 2010-12-01 08:28:00, 20... | NaN | |
| **InvoiceNo** | object | 541909 | 25900 | 0 | 0.000000 | [[536365, 536366, 536367, 536368, 536369, 5363... | NaN | |
| **Quantity** | int64 | 541909 | 722 | 0 | 0.000000 | [[6, 8, 2, 32, 3, 4, 24, 12, 48, 18, 20, 36, 8... | -0.264076 | 119769.1 |
| **StockCode** | object | 541909 | 4070 | 0 | 0.000000 | [[85123A, 71053, 84406B, 84029G, 84029E, 22752... | NaN | |
| **UnitPrice** | float64 | 541909 | 1630 | 0 | 0.000000 | [[2.55, 3.39, 2.75, 7.65, 4.25, 1.85, 1.69, 2.... | 186.506972 | 59005.7 |

In [ ]:
```
cs_df.describe()
```

Out[ ]:

| | Quantity | UnitPrice | CustomerID |
|---|---|---|---|
| **count** | 541909.000000 | 541909.000000 | 406829.000000 |
| **mean** | 9.552250 | 4.611114 | 15287.690570 |
| **std** | 218.081158 | 96.759853 | 1713.600303 |
| **min** | -80995.000000 | -11062.060000 | 12346.000000 |
| **25%** | 1.000000 | 1.250000 | 13953.000000 |
| **50%** | 3.000000 | 2.080000 | 15152.000000 |
| **75%** | 10.000000 | 4.130000 | 16791.000000 |
| **max** | 80995.000000 | 38970.000000 | 18287.000000 |

In [ ]:

```
print('Check if we had negative quantity and prices at same register:',
     'No' if cs_df[(cs_df.Quantity<0) & (cs_df.UnitPrice<0)].shape[0] == 0 else 'Ye
print('Check how many register we have where quantity is negative',
      'and prices is 0 or vice-versa:',
      cs_df[(cs_df.Quantity<=0) & (cs_df.UnitPrice<=0)].shape[0])
print('\nWhat is the customer ID of the registers above:',
      cs_df.loc[(cs_df.Quantity<=0) & (cs_df.UnitPrice<=0),
                ['CustomerID']].CustomerID.unique())
print('\n% Negative Quantity: {:3.2%}'.format(cs_df[(cs_df.Quantity<0)].shape[0]/cs
print('\nAll register with negative quantity has Invoice start with:',
      cs_df.loc[(cs_df.Quantity<0) & ~(cs_df.CustomerID.isnull()), 'InvoiceNo'].app
print('\nSee an example of negative quantity and others related records:')
display(cs_df[(cs_df.CustomerID==12472) & (cs_df.StockCode==22244)])
```

```
Check if we had negative quantity and prices at same register: No

Check how many register we have where quantity is negative and prices is 0 or vice
-versa: 1336

What is the customer ID of the registers above: [nan]

% Negative Quantity: 1.96%

All register with negative quantity has Invoice start with: ['C']

See an example of negative quantity and others related records:
```

| | InvoiceNo | StockCode | Description | Quantity | InvoiceDate | UnitPrice | CustomerID | Count |
|---|---|---|---|---|---|---|---|---|
| **1973** | C536548 | 22244 | 3 HOOK HANGER MAGIC GARDEN | -4 | 2010-12-01 14:33:00 | 1.95 | 12472.0 | Germa |
| **9438** | 537201 | 22244 | 3 HOOK HANGER MAGIC GARDEN | 12 | 2010-12-05 14:19:00 | 1.95 | 12472.0 | Germa |
| **121980** | 546843 | 22244 | 3 HOOK HANGER MAGIC GARDEN | 12 | 2011-03-17 12:40:00 | 1.95 | 12472.0 | Germa |

In [ ]:

```
print('Check register with UnitPrice negative:')
display(cs_df[(cs_df.UnitPrice<0)])
```

```
print("Sales records with Customer ID and zero in Unit Price:",cs_df[(cs_df.UnitPri
cs_df[(cs_df.UnitPrice==0)  & ~(cs_df.CustomerID.isnull())]
```

```
Check register with UnitPrice negative:
```

| | InvoiceNo | StockCode | Description | Quantity | InvoiceDate | UnitPrice | CustomerID | Count |
|---|---|---|---|---|---|---|---|---|
| **299983** | A563186 | B | Adjust bad debt | 1 | 2011-08-12 14:51:00 | -11062.06 | NaN | Unit Kingdo |
| **299984** | A563187 | B | Adjust bad debt | 1 | 2011-08-12 14:52:00 | -11062.06 | NaN | Unit Kingdo |

```
Sales records with Customer ID and zero in Unit Price: 40
```

Out[ ]:

| | InvoiceNo | StockCode | Description | Quantity | InvoiceDate | UnitPrice | CustomerID | C |
|---|---|---|---|---|---|---|---|---|
| **9302** | 537197 | 22841 | ROUND CAKE TIN VINTAGE GREEN | 1 | 2010-12-05 14:02:00 | 0.0 | 12647.0 | G |
| **33576** | 539263 | 22580 | ADVENT CALENDAR GINGHAM SACK | 4 | 2010-12-16 14:36:00 | 0.0 | 16560.0 | Ki |
| **40089** | 539722 | 22423 | REGENCY CAKESTAND 3 TIER | 10 | 2010-12-21 13:45:00 | 0.0 | 14911.0 | |
| **47068** | 540372 | 22090 | PAPER BUNTING RETROSPOT | 24 | 2011-01-06 16:41:00 | 0.0 | 13081.0 | Ki |
| **47070** | 540372 | 22553 | PLASTERS IN TIN SKULLS | 24 | 2011-01-06 16:41:00 | 0.0 | 13081.0 | Ki |
| **56674** | 541109 | 22168 | ORGANISER WOOD ANTIQUE WHITE | 1 | 2011-01-13 15:10:00 | 0.0 | 15107.0 | Ki |
| **86789** | 543599 | 84535B | FAIRY CAKES NOTEBOOK A6 SIZE | 16 | 2011-02-10 13:08:00 | 0.0 | 17560.0 | Ki |
| **130188** | 547417 | 22062 | CERAMIC BOWL WITH LOVE HEART DESIGN | 36 | 2011-03-23 10:25:00 | 0.0 | 13239.0 | Ki |
| **139453** | 548318 | 22055 | MINI CAKE STAND HANGING STRAWBERY | 5 | 2011-03-30 12:45:00 | 0.0 | 13113.0 | Ki |
| **145208** | 548871 | 22162 | HEART GARLAND RUSTIC PADDED | 2 | 2011-04-04 14:42:00 | 0.0 | 14410.0 | Ki |
| **157042** | 550188 | 22636 | CHILDS BREAKFAST SET CIRCUS PARADE | 1 | 2011-04-14 18:57:00 | 0.0 | 12457.0 | Swit |
| **187613** | 553000 | 47566 | PARTY BUNTING | 4 | 2011-05-12 15:21:00 | 0.0 | 17667.0 | Ki |
| **198383** | 554037 | 22619 | SET OF 6 SOLDIER SKITTLES | 80 | 2011-05-20 14:13:00 | 0.0 | 12415.0 | A |
| **279324** | 561284 | 22167 | OVAL WALL MIRROR DIAMANTE | 1 | 2011-07-26 12:24:00 | 0.0 | 16818.0 | Ki |
| **282912** | 561669 | 22960 | JAM MAKING SET WITH JARS | 11 | 2011-07-28 17:09:00 | 0.0 | 12507.0 | |
| **285657** | 561916 | M | Manual | 1 | 2011-08-01 11:44:00 | 0.0 | 15581.0 | Ki |

| | InvoiceNo | StockCode | Description | Quantity | InvoiceDate | UnitPrice | CustomerID | C |
|---|---|---|---|---|---|---|---|---|
| **298054** | 562973 | 23157 | SET OF 6 NATIVITY MAGNETS | 240 | 2011-08-11 11:42:00 | 0.0 | 14911.0 | |
| **314745** | 564651 | 23270 | SET OF 2 CERAMIC PAINTED HEARTS | 96 | 2011-08-26 14:19:00 | 0.0 | 14646.0 | Neth |
| **314746** | 564651 | 23268 | SET OF 2 CERAMIC CHRISTMAS REINDEER | 192 | 2011-08-26 14:19:00 | 0.0 | 14646.0 | Neth |
| **314747** | 564651 | 22955 | 36 FOIL STAR CAKE CASES | 144 | 2011-08-26 14:19:00 | 0.0 | 14646.0 | Neth |
| **314748** | 564651 | 21786 | POLKADOT RAIN HAT | 144 | 2011-08-26 14:19:00 | 0.0 | 14646.0 | Neth |
| **358655** | 568158 | PADS | PADS TO MATCH ALL CUSHIONS | 1 | 2011-09-25 12:22:00 | 0.0 | 16133.0 | Ki |
| **361825** | 568384 | M | Manual | 1 | 2011-09-27 09:46:00 | 0.0 | 12748.0 | Ki |
| **379913** | 569716 | 22778 | GLASS CLOCHE SMALL | 2 | 2011-10-06 08:17:00 | 0.0 | 15804.0 | Ki |
| **395529** | 571035 | M | Manual | 1 | 2011-10-13 12:50:00 | 0.0 | 12446.0 | |
| **420404** | 572893 | 21208 | PASTEL COLOUR HONEYCOMB FAN | 5 | 2011-10-26 14:36:00 | 0.0 | 18059.0 | Ki |
| **436428** | 574138 | 23234 | BISCUIT TIN VINTAGE CHRISTMAS | 216 | 2011-11-03 11:26:00 | 0.0 | 12415.0 | A |
| **436597** | 574175 | 22065 | CHRISTMAS PUDDING TRINKET POT | 12 | 2011-11-03 11:47:00 | 0.0 | 14110.0 | Ki |
| **436961** | 574252 | M | Manual | 1 | 2011-11-03 13:24:00 | 0.0 | 12437.0 | |
| **439361** | 574469 | 22385 | JUMBO BAG SPACEBOY DESIGN | 12 | 2011-11-04 11:55:00 | 0.0 | 12431.0 | A |
| **446125** | 574879 | 22625 | RED KITCHEN SCALES | 2 | 2011-11-07 13:22:00 | 0.0 | 13014.0 | Ki |
| **446793** | 574920 | 22899 | CHILDREN'S APRON DOLLY GIRL | 1 | 2011-11-07 16:34:00 | 0.0 | 13985.0 | Ki |
| **446794** | 574920 | 23480 | MINI LIGHTS WOODLAND MUSHROOMS | 1 | 2011-11-07 16:34:00 | 0.0 | 13985.0 | Ki |
| **454463** | 575579 | 22437 | SET OF 9 BLACK SKULL | 20 | 2011-11-10 11:49:00 | 0.0 | 13081.0 | Ki |

| | InvoiceNo | StockCode | Description | Quantity | InvoiceDate | UnitPrice | CustomerID | C |
|---|---|---|---|---|---|---|---|---|
| | | | BALLOONS | | | | | |
| **454464** | 575579 | 22089 | PAPER BUNTING VINTAGE PAISLEY | 24 | 2011-11-10 11:49:00 | 0.0 | 13081.0 | Ki |
| **479079** | 577129 | 22464 | HANGING METAL HEART LANTERN | 4 | 2011-11-17 19:52:00 | 0.0 | 15602.0 | Ki |
| **479546** | 577168 | M | Manual | 1 | 2011-11-18 10:42:00 | 0.0 | 12603.0 | G |
| **480649** | 577314 | 23407 | SET OF 2 TRAYS HOME SWEET HOME | 2 | 2011-11-18 13:23:00 | 0.0 | 12444.0 | |
| **485985** | 577696 | M | Manual | 1 | 2011-11-21 11:57:00 | 0.0 | 16406.0 | Ki |
| **502122** | 578841 | 84826 | ASSTD DESIGN 3D PAPER STICKERS | 12540 | 2011-11-25 15:57:00 | 0.0 | 13256.0 | Ki |

In [ ]:

```
# Remove register withou CustomerID
cs_df = cs_df[~(cs_df.CustomerID.isnull())]

# Remove negative or return transactions
cs_df = cs_df[~(cs_df.Quantity<0)]
cs_df = cs_df[cs_df.UnitPrice>0]

details = rstr(cs_df)
display(details.sort_values(by='distincts', ascending=False))
```

```
Data shape: (397884, 8)
___________________________
Data types:
 object           4
float64          2
int64            1
datetime64[ns]   1
Name: types, dtype: int64
___________________________
```

| | types | counts | distincts | nulls | missing ration | uniques | skewness | kurtos |
|---|---|---|---|---|---|---|---|---|
| **InvoiceNo** | object | 397884 | 18532 | 0 | 0.0 | [[536365, 536366, 536367, 536368, 536369, 5363... | -0.178524 | -1.20074 |
| **InvoiceDate** | datetime64[ns] | 397884 | 17282 | 0 | 0.0 | [[2010-12-01 08:26:00, 2010-12-01 08:28:00, 20... | NaN | Na |
| **CustomerID** | float64 | 397884 | 4338 | 0 | 0.0 | [[17850.0, 13047.0, 12583.0, 13748.0, 15100.0,... | 0.025729 | -1.18082 |
| **Description** | object | 397884 | 3877 | 0 | 0.0 | [[WHITE HANGING HEART T-LIGHT HOLDER, WHITE ME... | NaN | Na |
| **StockCode** | object | 397884 | 3665 | 0 | 0.0 | [[85123A, 71053, 84406B, 84029G, 84029E, 22752... | NaN | Na |
| **UnitPrice** | float64 | 397884 | 440 | 0 | 0.0 | [[2.55, 3.39, 2.75, 7.65, 4.25, 1.85, 1.69, 2.... | 204.032727 | 58140.39667 |
| **Quantity** | int64 | 397884 | 301 | 0 | 0.0 | [[6, 8, 2, 32, 3, 4, 24, 12, 48, 18, 20, 36, 8... | 409.892972 | 178186.24325 |
| **Country** | object | 397884 | 37 | 0 | 0.0 | [[United Kingdom, France, Australia, Netherlan... | NaN | Na |

After this first cleanup, note that we still have more description than inventory codes, so we still have some inconsistency on the basis that requires further investigation. Let´s see it:

In [ ]:
```
cat_des_df = cs_df.groupby(["StockCode","Description"]).count().reset_index()
display(cat_des_df.StockCode.value_counts()[cat_des_df.StockCode.value_counts()>1].
cs_df[cs_df['StockCode'] == cat_des_df.StockCode.value_counts()[cat_des_df.StockCod
      .reset_index()['index'][4]]['Description'].unique()
```

| | index | StockCode |
|---|---|---|
| 0 | 23236 | 4 |
| 1 | 23196 | 4 |
| 2 | 23203 | 3 |
| 3 | 17107D | 3 |
| 4 | 23370 | 3 |

Out[ ]:

```
array(['SET 36 COLOUR PENCILS DOILEY', 'SET 36 COLOURING PENCILS DOILY',
       'SET 36 COLOURING PENCILS DOILEY'], dtype=object)
```

This gives the multiple descriptions for one of those items and we witness the simple ways in which data quality can be corrupted in any dataset. A simple spelling mistake can end up in reducing data quality and an erroneous analysis.

In [ ]:

```
unique_desc = cs_df[["StockCode", "Description"]].groupby(by=["StockCode"]).\
                apply(pd.DataFrame.mode).reset_index(drop=True)
q = '''
select df.InvoiceNo, df.StockCode, un.Description, df.Quantity, df.InvoiceDate,
       df.UnitPrice, df.CustomerID, df.Country
from cs_df as df INNER JOIN
     unique_desc as un on df.StockCode = un.StockCode
'''

cs_df = pysqldf(q)
```

In [ ]:

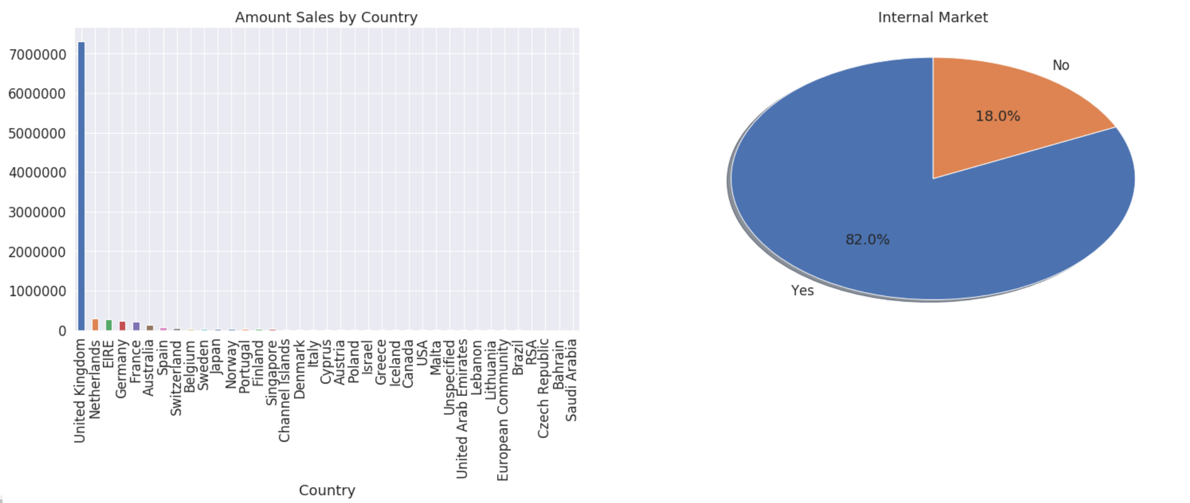
```
cs_df.InvoiceDate = pd.to_datetime(cs_df.InvoiceDate)
cs_df['amount'] = cs_df.Quantity*cs_df.UnitPrice
cs_df.CustomerID = cs_df.CustomerID.astype('Int64')

details = rstr(cs_df)
display(details.sort_values(by='distincts', ascending=False))
```

```
Data shape: (397884, 9)
___________________________
Data types:
 object            3
int64             3
float64           2
datetime64[ns]    1
Name: types, dtype: int64
___________________________
```

| | types | counts | distincts | nulls | missing ration | uniques | skewness | |
|---|---|---|---|---|---|---|---|---|
| **InvoiceNo** | int64 | 397884 | 18532 | 0 | 0.0 | [[536365, 536366, 536367, 536368, 536369, 5363... | -0.178524 | |
| **InvoiceDate** | datetime64[ns] | 397884 | 17282 | 0 | 0.0 | [[2010-12-01 08:26:00, 2010-12-01 08:28:00, 20... | NaN | |
| **CustomerID** | int64 | 397884 | 4338 | 0 | 0.0 | [[17850, 13047, 12583, 13748, 15100, 15291, 14... | 0.025729 | |
| **StockCode** | object | 397884 | 3665 | 0 | 0.0 | [[85123A, 71053, 84406B, 84029G, 84029E, 22752... | NaN | |
| **Description** | object | 397884 | 3647 | 0 | 0.0 | [[WHITE HANGING HEART T-LIGHT HOLDER, WHITE ME... | NaN | |
| **amount** | float64 | 397884 | 2939 | 0 | 0.0 | [[15.299999999999999, 20.34, 22.0, 15.3, 25.5,... | 451.443182 | 23 |
| **UnitPrice** | float64 | 397884 | 440 | 0 | 0.0 | [[2.55, 3.39, 2.75, 7.65, 4.25, 1.85, 1.69, 2.... | 204.032727 | 5 |
| **Quantity** | int64 | 397884 | 301 | 0 | 0.0 | [[6, 8, 2, 32, 3, 4, 24, 12, 48, 18, 20, 36, 8... | 409.892972 | 17 |
| **Country** | object | 397884 | 37 | 0 | 0.0 | [[United Kingdom, France, Australia, Netherlan... | NaN | |

In [ ]:

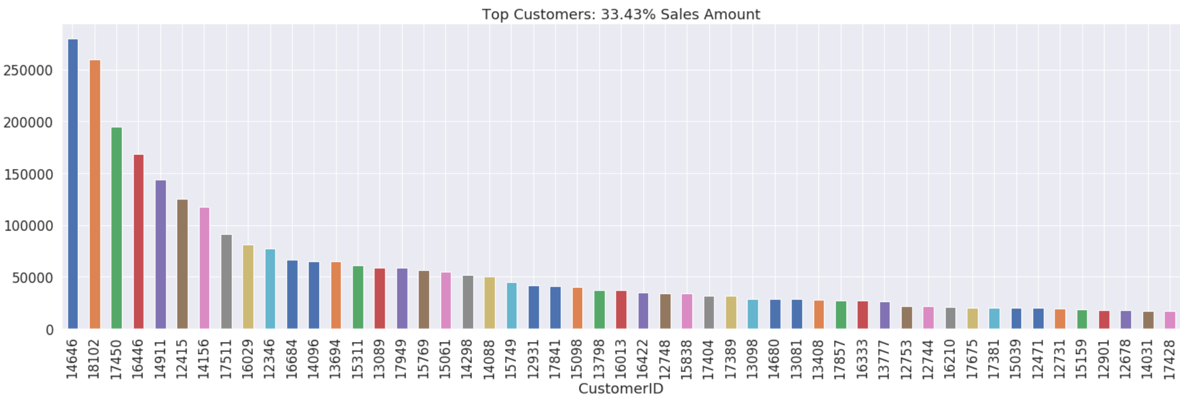
```
fig = plt.figure(figsize=(25, 7))
f1 = fig.add_subplot(121)
g = cs_df.groupby(["Country"]).amount.sum().sort_values(ascending = False).plot(kin
cs_df['Internal'] = cs_df.Country.apply(lambda x: 'Yes' if x=='United Kingdom' else
f2 = fig.add_subplot(122)
market = cs_df.groupby(["Internal"]).amount.sum().sort_values(ascending = False)
g = plt.pie(market, labels=market.index, autopct='%1.1f%%', shadow=True, startangle
plt.title('Internal Market')
plt.show()
```

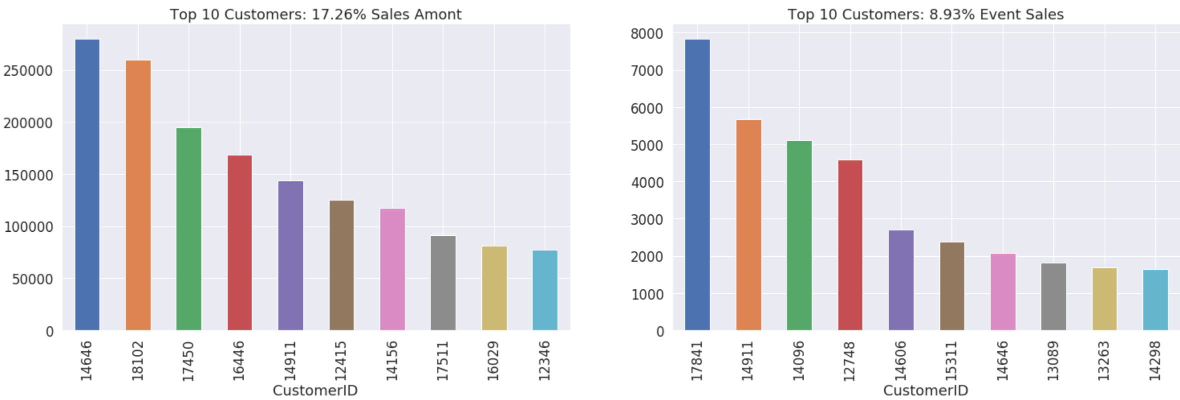



In [ ]:

```
fig = plt.figure(figsize=(25, 7))
PercentSales =  np.round((cs_df.groupby(["CustomerID"]).amount.sum().\
```

```
                          sort_values(ascending = False)[:51].sum()/cs_df.groupby([
                          amount.sum().sort_values(ascending = False).sum()) * 100,
g = cs_df.groupby(["CustomerID"]).amount.sum().sort_values(ascending = False)[:51].
    plot(kind='bar', title='Top Customers: {:3.2f}% Sales Amount'.format(PercentSal

fig = plt.figure(figsize=(25, 7))
f1 = fig.add_subplot(121)
PercentSales =  np.round((cs_df.groupby(["CustomerID"]).amount.sum().\
                          sort_values(ascending = False)[:10].sum()/cs_df.groupby([
                          amount.sum().sort_values(ascending = False).sum()) * 100,
g = cs_df.groupby(["CustomerID"]).amount.sum().sort_values(ascending = False)[:10]\
    .plot(kind='bar', title='Top 10 Customers: {:3.2f}% Sales Amont'.format(Percent
f1 = fig.add_subplot(122)
PercentSales =  np.round((cs_df.groupby(["CustomerID"]).amount.count().\
                          sort_values(ascending = False)[:10].sum()/cs_df.groupby([
                          amount.count().sort_values(ascending = False).sum()) * 10
g = cs_df.groupby(["CustomerID"]).amount.count().sort_values(ascending = False)[:10
    plot(kind='bar', title='Top 10 Customers: {:3.2f}% Event Sales'.format(PercentS
```

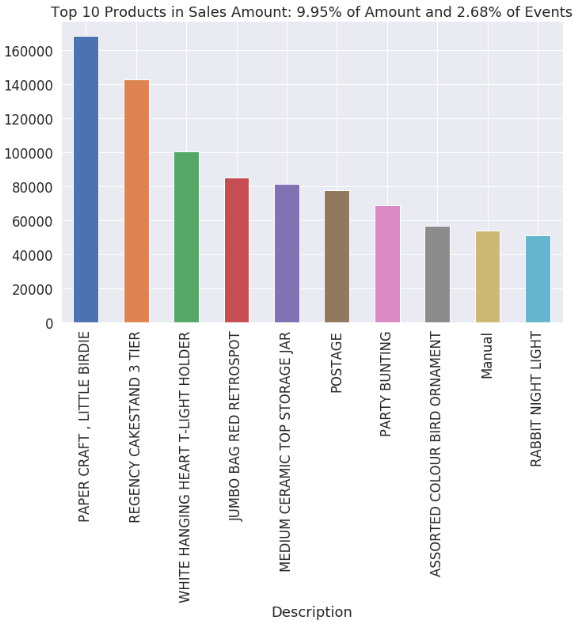


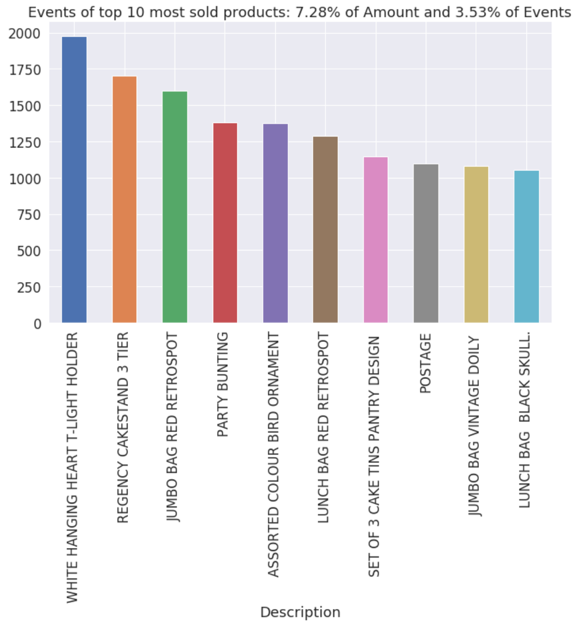



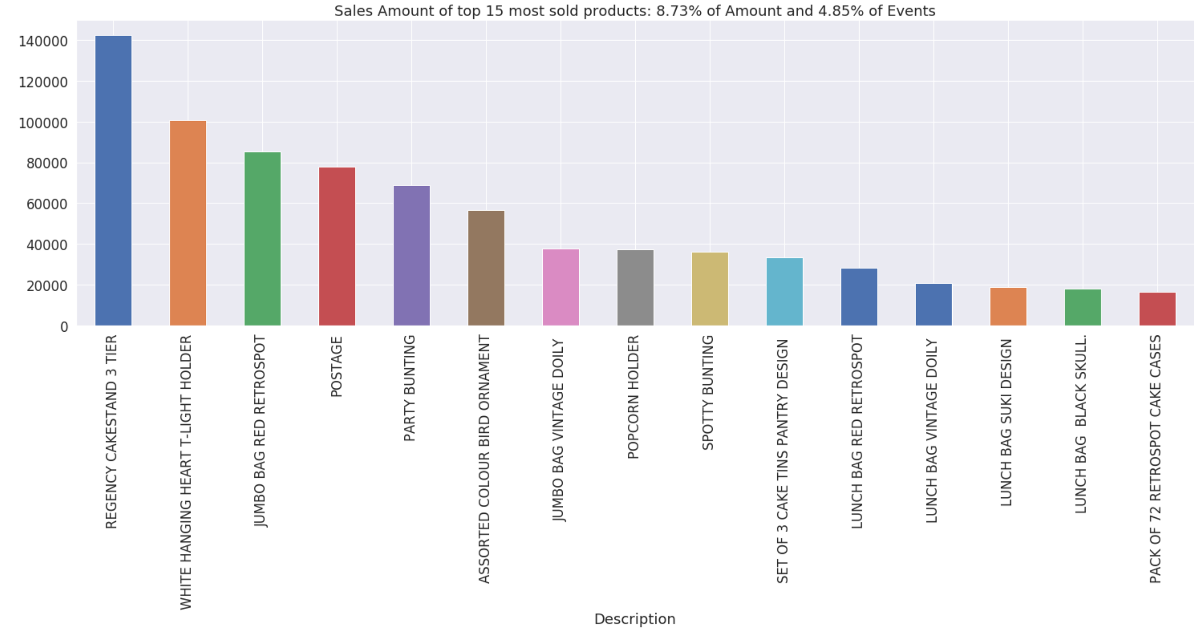

In [ ]:

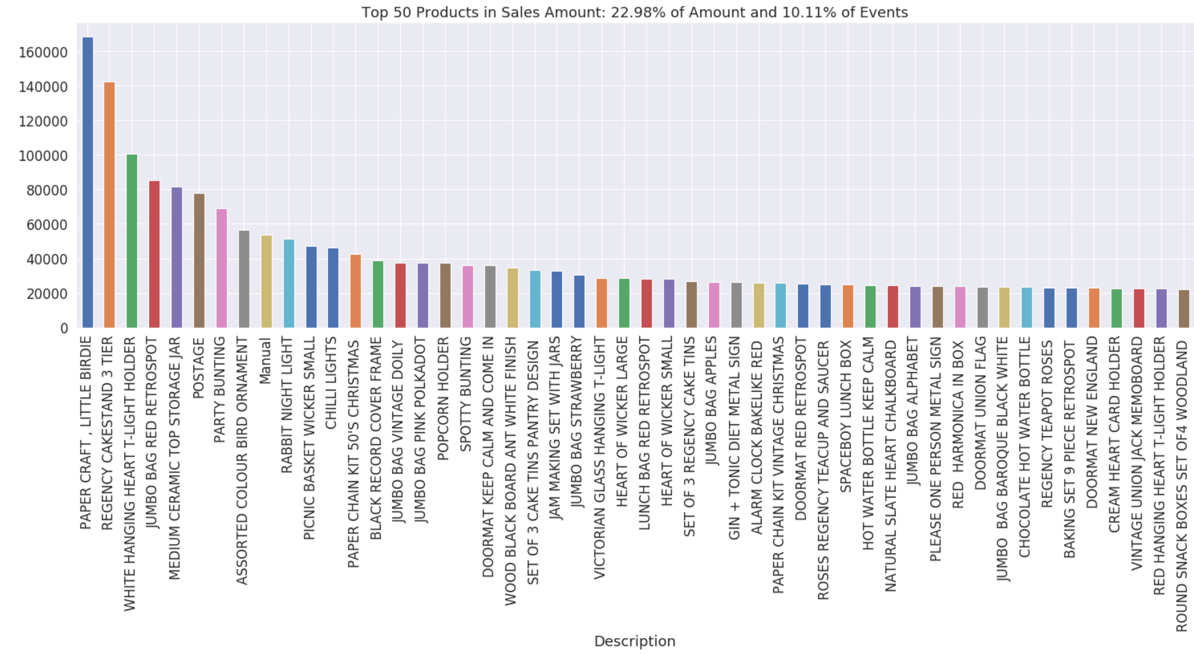
```
AmoutSum = cs_df.groupby(["Description"]).amount.sum().sort_values(ascending = Fals
inv = cs_df[["Description", "InvoiceNo"]].groupby(["Description"]).InvoiceNo.unique
      agg(np.size).sort_values(ascending = False)

fig = plt.figure(figsize=(25, 7))
f1 = fig.add_subplot(121)
Top10 = list(AmoutSum[:10].index)
PercentSales =  np.round((AmoutSum[Top10].sum()/AmoutSum.sum()) * 100, 2)
PercentEvents = np.round((inv[Top10].sum()/inv.sum()) * 100, 2)
g = AmoutSum[Top10].\
    plot(kind='bar', title='Top 10 Products in Sales Amount: {:3.2f}% of Amount and
                        format(PercentSales, PercentEvents))

f1 = fig.add_subplot(122)
Top10Ev = list(inv[:10].index)
PercentSales =  np.round((AmoutSum[Top10Ev].sum()/AmoutSum.sum()) * 100, 2)
PercentEvents = np.round((inv[Top10Ev].sum()/inv.sum()) * 100, 2)
g = inv[Top10Ev].\
    plot(kind='bar', title='Events of top 10 most sold products: {:3.2f}% of Amount
```

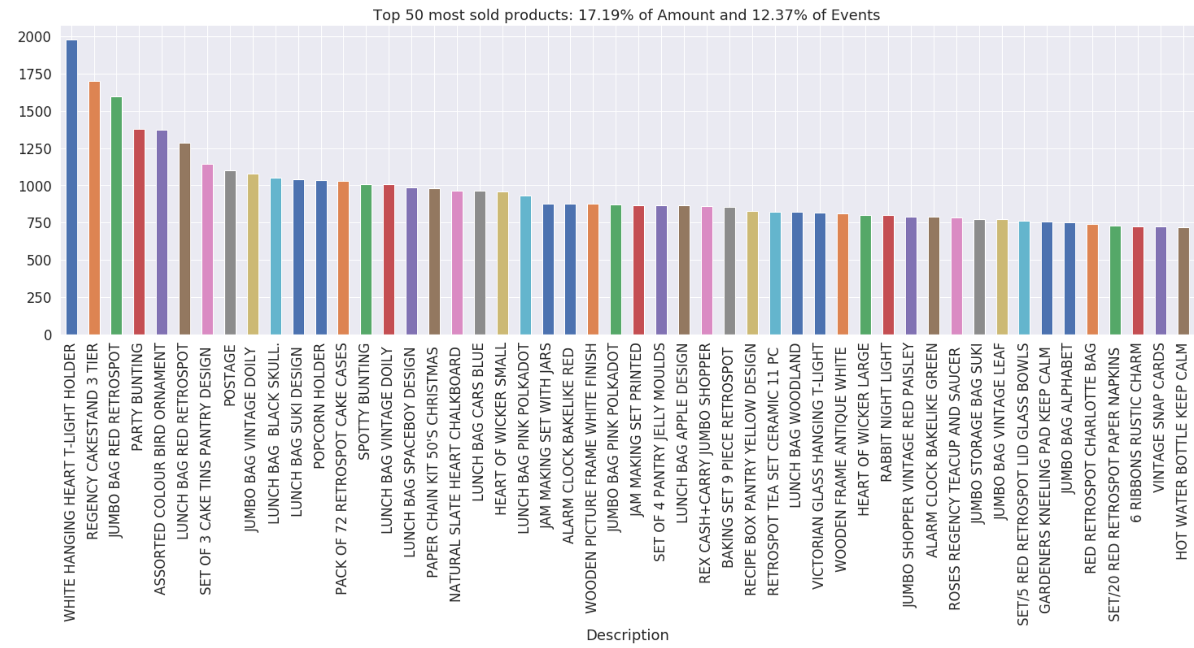
```
                        format(PercentSales, PercentEvents))


fig = plt.figure(figsize=(25, 7))
Top15ev = list(inv[:15].index)
PercentSales =  np.round((AmoutSum[Top15ev].sum()/AmoutSum.sum()) * 100, 2)
PercentEvents = np.round((inv[Top15ev].sum()/inv.sum()) * 100, 2)
g = AmoutSum[Top15ev].sort_values(ascending = False).\
    plot(kind='bar',
         title='Sales Amount of top 15 most sold products: {:3.2f}% of Amount and
         format(PercentSales, PercentEvents))

fig = plt.figure(figsize=(25, 7))
Top50 = list(AmoutSum[:50].index)
PercentSales =  np.round((AmoutSum[Top50].sum()/AmoutSum.sum()) * 100, 2)
PercentEvents = np.round((inv[Top50].sum()/inv.sum()) * 100, 2)
g = AmoutSum[Top50].\
    plot(kind='bar',
         title='Top 50 Products in Sales Amount: {:3.2f}% of Amount and {:3.2f}% o
         format(PercentSales, PercentEvents))

fig = plt.figure(figsize=(25, 7))
Top50Ev = list(inv[:50].index)
PercentSales =  np.round((AmoutSum[Top50Ev].sum()/AmoutSum.sum()) * 100, 2)
PercentEvents = np.round((inv[Top50Ev].sum()/inv.sum()) * 100, 2)
g = inv[Top50Ev].\
    plot(kind='bar', title='Top 50 most sold products: {:3.2f}% of Amount and {:3.
                        format(PercentSales, PercentEvents))
```






In [ ]:

```
refrence_date = cs_df.InvoiceDate.max() + datetime.timedelta(days = 1)
print('Reference Date:', refrence_date)
cs_df['days_since_last_purchase'] = (refrence_date - cs_df.InvoiceDate).astype('ti
```

```
customer_history_df =  cs_df[['CustomerID', 'days_since_last_purchase']].groupby("C
customer_history_df.rename(columns={'days_since_last_purchase':'recency'}, inplace=
customer_history_df.describe().transpose()
```

```
Reference Date: 2011-12-10 12:50:00
```

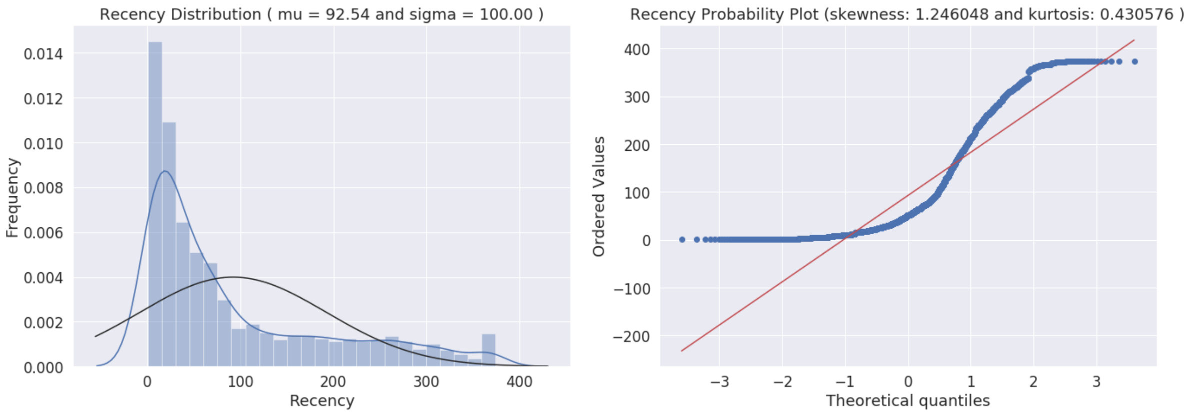
Out[ ]:

| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| **CustomerID** | 4338.0 | 15300.408022 | 1721.808492 | 12346.0 | 13813.25 | 15299.5 | 16778.75 | 18287.0 |
| **recency** | 4338.0 | 92.536422 | 100.014169 | 1.0 | 18.00 | 51.0 | 142.00 | 374.0 |

We will plot the Recency Distribution and QQ-plot to identify substantive departures from normality, likes outliers, skewness and kurtosis.

In [ ]:

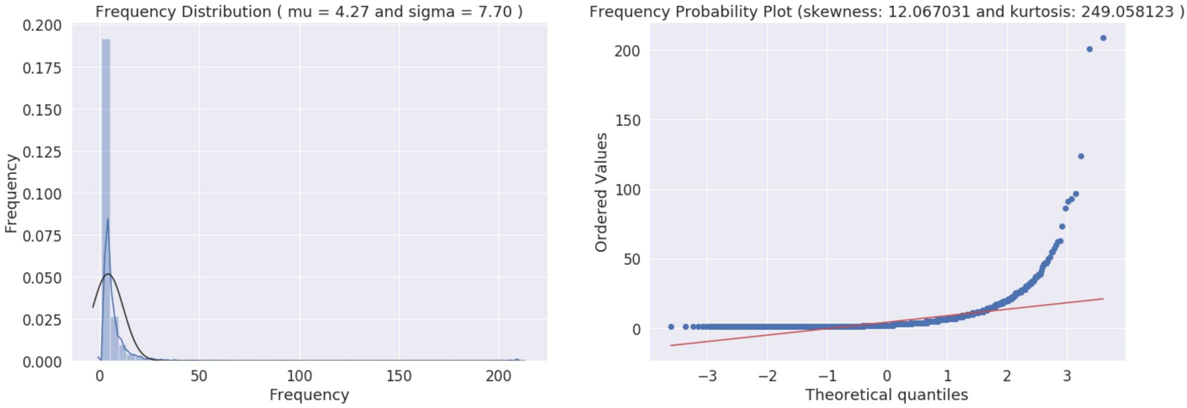
```
def QQ_plot(data, measure):
    fig = plt.figure(figsize=(20,7))

    #Get the fitted parameters used by the function
    (mu, sigma) = norm.fit(data)

    #Kernel Density plot
    fig1 = fig.add_subplot(121)
    sns.distplot(data, fit=norm)
    fig1.set_title(measure + ' Distribution ( mu = {:.2f} and sigma = {:.2f} )'.for
    fig1.set_xlabel(measure)
    fig1.set_ylabel('Frequency')

    #QQ plot
    fig2 = fig.add_subplot(122)
    res = probplot(data, plot=fig2)
    fig2.set_title(measure + ' Probability Plot (skewness: {:.6f} and kurtosis: {:.

    plt.tight_layout()
    plt.show()

QQ_plot(customer_history_df.recency, 'Recency')
```



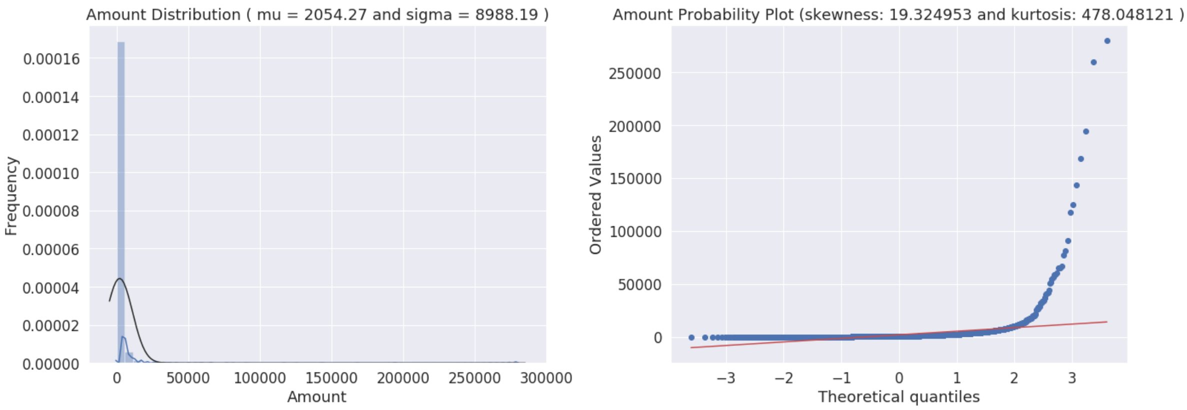

## Frequency

In [ ]:

```
customer_freq = (cs_df[['CustomerID', 'InvoiceNo']].groupby(["CustomerID", 'Invoice
                groupby(["CustomerID"]).count().reset_index()
customer_freq.rename(columns={'InvoiceNo':'frequency'},inplace=True)
customer_history_df = customer_history_df.merge(customer_freq)
QQ_plot(customer_history_df.frequency, 'Frequency')
```




```
In [ ]: customer_monetary_val = cs_df[['CustomerID', 'amount']].groupby("CustomerID").sum()
        customer_history_df = customer_history_df.merge(customer_monetary_val)
        QQ_plot(customer_history_df.amount, 'Amount')
```




```
In [ ]: customer_history_df.describe()
```

Out[ ]:

| | CustomerID | recency | frequency | amount |
|---|---|---|---|---|
| **count** | 4338.000000 | 4338.000000 | 4338.000000 | 4338.000000 |
| **mean** | 15300.408022 | 92.536422 | 4.272015 | 2054.266460 |
| **std** | 1721.808492 | 100.014169 | 7.697998 | 8989.230441 |
| **min** | 12346.000000 | 1.000000 | 1.000000 | 3.750000 |
| **25%** | 13813.250000 | 18.000000 | 1.000000 | 307.415000 |
| **50%** | 15299.500000 | 51.000000 | 2.000000 | 674.485000 |
| **75%** | 16778.750000 | 142.000000 | 5.000000 | 1661.740000 |
| **max** | 18287.000000 | 374.000000 | 209.000000 | 280206.020000 |

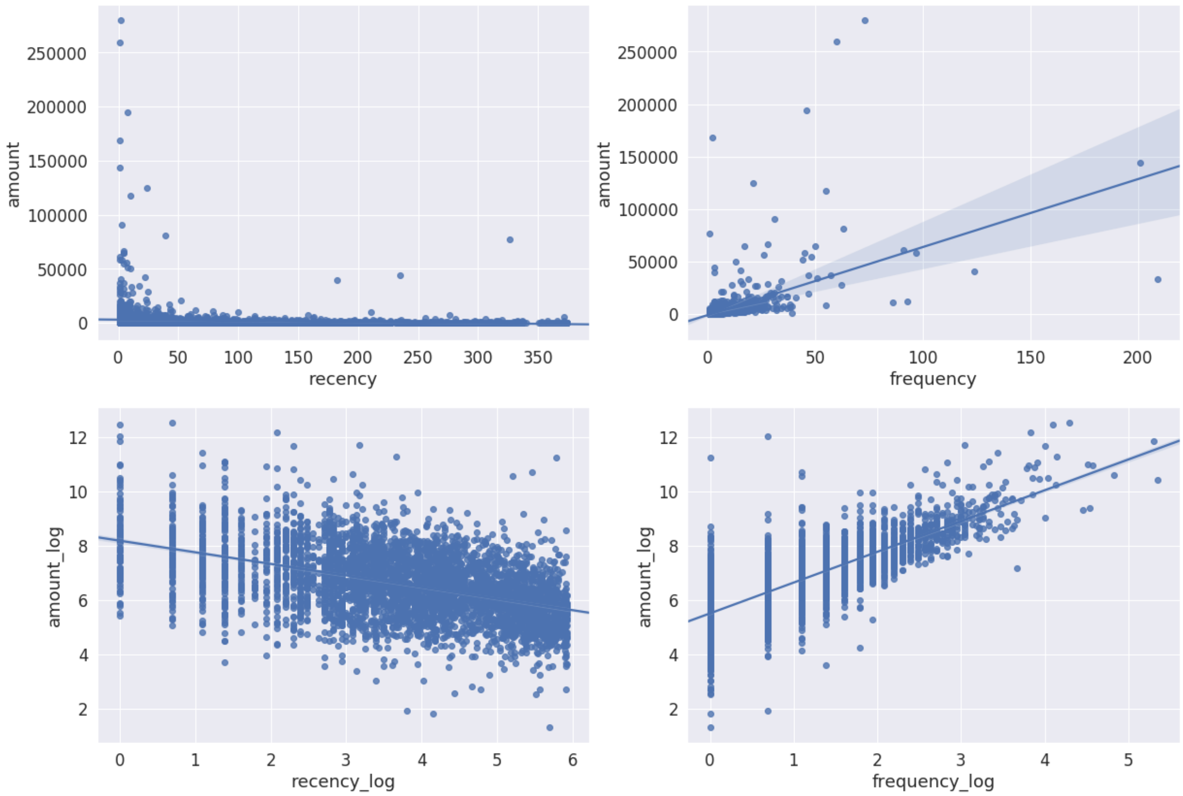
```
In [ ]: customer_history_df['recency_log'] = customer_history_df['recency'].apply(math.log)
        customer_history_df['frequency_log'] = customer_history_df['frequency'].apply(math.
        customer_history_df['amount_log'] = customer_history_df['amount'].apply(math.log)
        feature_vector = ['amount_log', 'recency_log','frequency_log']
        X_subset = customer_history_df[feature_vector] #.as_matrix()
        scaler = preprocessing.StandardScaler().fit(X_subset)
        X_scaled = scaler.transform(X_subset)
        pd.DataFrame(X_scaled, columns=X_subset.columns).describe().T
```

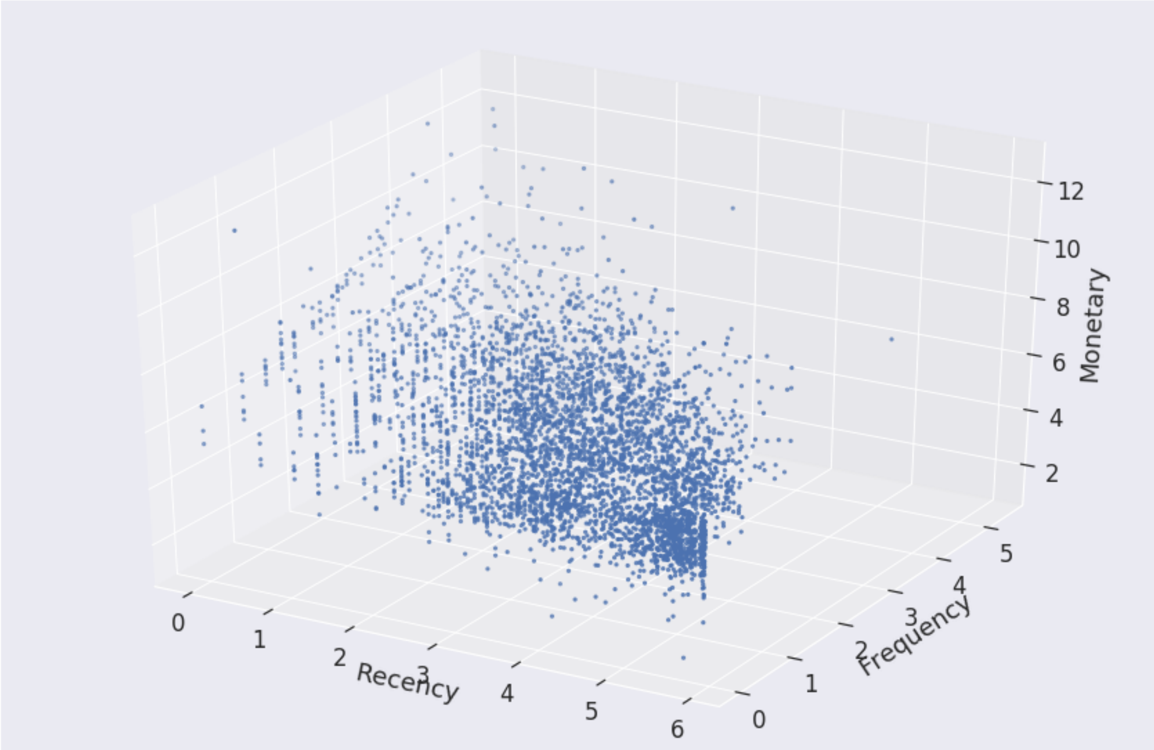
Out[ ]:

| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| **amount_log** | 4338.0 | -1.202102e-16 | 1.000115 | -4.179280 | -0.684183 | -0.060942 | 0.654244 | 4.721395 |
| **recency_log** | 4338.0 | -1.027980e-16 | 1.000115 | -2.630445 | -0.612424 | 0.114707 | 0.829652 | 1.505796 |
| **frequency_log** | 4338.0 | -2.355833e-16 | 1.000115 | -1.048610 | -1.048610 | -0.279044 | 0.738267 | 4.882714 |

In [ ]:

```
fig = plt.figure(figsize=(20,14))
f1 = fig.add_subplot(221); sns.regplot(x='recency', y='amount', data=customer_histo
f1 = fig.add_subplot(222); sns.regplot(x='frequency', y='amount', data=customer_his
f1 = fig.add_subplot(223); sns.regplot(x='recency_log', y='amount_log', data=custor
f1 = fig.add_subplot(224); sns.regplot(x='frequency_log', y='amount_log', data=cust

fig = plt.figure(figsize=(15, 10))
ax = fig.add_subplot(111, projection='3d')

xs =customer_history_df.recency_log
ys = customer_history_df.frequency_log
zs = customer_history_df.amount_log
ax.scatter(xs, ys, zs, s=5)

ax.set_xlabel('Recency')
ax.set_ylabel('Frequency')
ax.set_zlabel('Monetary')

plt.show()
```

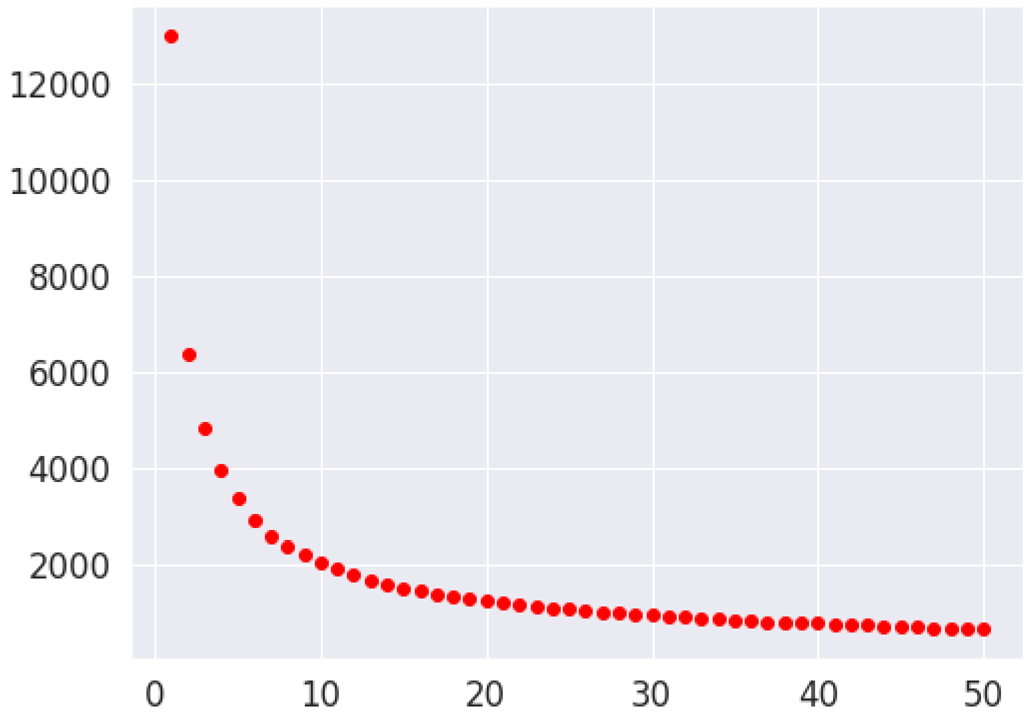




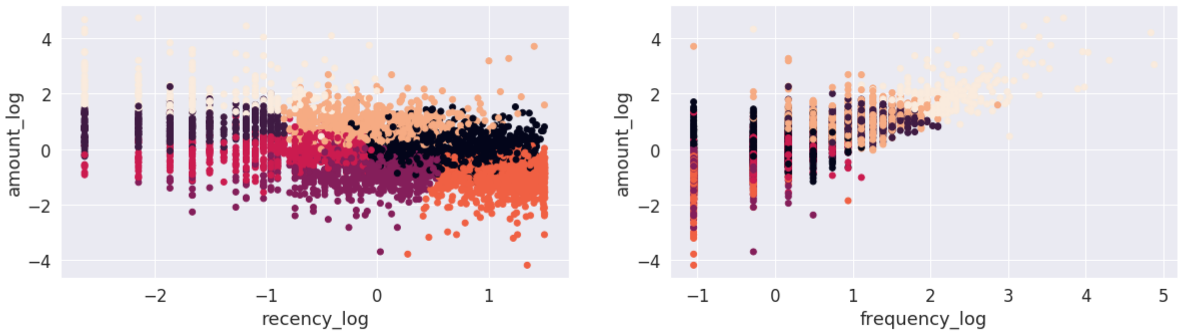

The obvious patterns we can see from the plots above is that costumers who buy with a higher frequency and more recency tend to spend more based on the increasing trend in Monetary (amount value) with a corresponding increasing and decreasing trend for Frequency and Recency, respectively.

In [ ]:
```
cl = 50
corte = 0.1

anterior = 100000000000000
cost = []
K_best = cl

for k in range (1, cl+1):
    # Create a kmeans model on our data, using k clusters.  random_state helps ensu
    model = KMeans(
        n_clusters=k,
        init='k-means++', #'random',
        n_init=10,
        max_iter=300,
        tol=1e-04,
        random_state=101)

    model = model.fit(X_scaled)

    # These are our fitted labels for clusters -- the first cluster has label 0, ar
    labels = model.labels_

    # Sum of distances of samples to their closest cluster center
    interia = model.inertia_
    if (K_best == cl) and (((anterior - interia)/anterior) < corte): K_best = k - 1
    cost.append(interia)
    anterior = interia

plt.figure(figsize=(8, 6))
plt.scatter(range (1, cl+1), cost, c='red')
plt.show()
```

```
# Create a kmeans model with the best K.
print('The best K sugest: ',K_best)
model = KMeans(n_clusters=K_best, init='k-means++', n_init=10,max_iter=300, tol=1e-

# Note I'm scaling the data to normalize it! Important for good results.
model = model.fit(X_scaled)

# These are our fitted labels for clusters -- the first cluster has label 0, and th
labels = model.labels_

# And we'll visualize it:
#plt.scatter(X_scaled[:,0], X_scaled[:,1], c=model.labels_.astype(float))
fig = plt.figure(figsize=(20,5))
ax = fig.add_subplot(121)
plt.scatter(x = X_scaled[:,1], y = X_scaled[:,0], c=model.labels_.astype(float))
ax.set_xlabel(feature_vector[1])
ax.set_ylabel(feature_vector[0])
ax = fig.add_subplot(122)
plt.scatter(x = X_scaled[:,2], y = X_scaled[:,0], c=model.labels_.astype(float))
ax.set_xlabel(feature_vector[2])
ax.set_ylabel(feature_vector[0])

plt.show()
```




```
The best K sugest:  7
```




```
In [ ]:
```