```
# IMPORTANT: RUN THIS CELL IN ORDER TO IMPORT YOUR KAGGLE DATA SOURCES
# TO THE CORRECT LOCATION (/kaggle/input) IN YOUR NOTEBOOK,
# THEN FEEL FREE TO DELETE THIS CELL.
# NOTE: THIS NOTEBOOK ENVIRONMENT DIFFERS FROM KAGGLE'S PYTHON
# ENVIRONMENT SO THERE MAY BE MISSING LIBRARIES USED BY YOUR
# NOTEBOOK.

import os
import sys
from tempfile import NamedTemporaryFile
from urllib.request import urlopen
from urllib.parse import unquote, urlparse
from urllib.error import HTTPError
from zipfile import ZipFile
import tarfile
import shutil

CHUNK_SIZE = 40960
DATA_SOURCE_MAPPING = 'customer_segmentation:https%3A%2F%2Fstorage.googleapis.com%2Fkaggle-data-sets%2F302641%2F618141%2Fbundle%2Farchive

KAGGLE_INPUT_PATH='/kaggle/input'
KAGGLE_WORKING_PATH='/kaggle/working'
KAGGLE_SYMLINK='kaggle'

!umount /kaggle/input/ 2> /dev/null
shutil.rmtree('/kaggle/input', ignore_errors=True)
os.makedirs(KAGGLE_INPUT_PATH, 0o777, exist_ok=True)
os.makedirs(KAGGLE_WORKING_PATH, 0o777, exist_ok=True)

try:
  os.symlink(KAGGLE_INPUT_PATH, os.path.join("..", 'input'), target_is_directory=True)
except FileExistsError:
  pass
try:
  os.symlink(KAGGLE_WORKING_PATH, os.path.join("..", 'working'), target_is_directory=True)
except FileExistsError:
  pass

for data_source_mapping in DATA_SOURCE_MAPPING.split(','):
    directory, download_url_encoded = data_source_mapping.split(':')
    download_url = unquote(download_url_encoded)
    filename = urlparse(download_url).path
    destination_path = os.path.join(KAGGLE_INPUT_PATH, directory)
    try:
        with urlopen(download_url) as fileres, NamedTemporaryFile() as tfile:
            total_length = fileres.headers['content-length']
            print(f'Downloading {directory}, {total_length} bytes compressed')
            dl = 0
            data = fileres.read(CHUNK_SIZE)
            while len(data) > 0:
                dl += len(data)
                tfile.write(data)
                done = int(50 * dl / int(total_length))
                sys.stdout.write(f"\r[{'=' * done}{' ' * (50-done)}] {dl} bytes downloaded")
                sys.stdout.flush()
                data = fileres.read(CHUNK_SIZE)
            if filename.endswith('.zip'):
              with ZipFile(tfile) as zfile:
                zfile.extractall(destination_path)
            else:
              with tarfile.open(tfile.name) as tarfile:
                tarfile.extractall(destination_path)
            print(f'\nDownloaded and uncompressed: {directory}')
    except HTTPError as e:
        print(f'Failed to load (likely expired) {download_url} to path {destination_path}')
        continue
    except OSError as e:
        print(f'Failed to load {download_url} to path {destination_path}')
        continue

print('Data source import complete.')
```

```
Downloading customer_segmentation, 7548720 bytes compressed
[==================================================] 7548720 bytes downloaded
Downloaded and uncompressed: customer_segmentation
Data source import complete.
```

```
#import libraries
from __future__ import division

from datetime import datetime, timedelta,date
import pandas as pd
%matplotlib inline
from sklearn.metrics import classification_report,confusion_matrix
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from sklearn.cluster import KMeans


import plotly as py
import plotly.offline as pyoff
import plotly.graph_objs as go

import xgboost as xgb
from sklearn.model_selection import KFold, cross_val_score, train_test_split

import xgboost as xgb


#Read data
tx_data = pd.read_csv('../input/customer_segmentation/customer_segmentation.csv', encoding='cp1252')


#initate plotly
pyoff.init_notebook_mode()

#read data from csv and redo the data work we done before
tx_data.head()
```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	C
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	12/1/2010 8:26	2.55	17850.0	K
1	536365	71053	WHITE METAL	6	12/1/2010 8:26	3.39	17850.0	K

Feature Engineering

```
#converting the type of Invoice Date Field from string to datetime.
tx_data['InvoiceDate'] = pd.to_datetime(tx_data['InvoiceDate'])


#creating YearMonth field for the ease of reporting and visualization
tx_data['InvoiceYearMonth'] = tx_data['InvoiceDate'].map(lambda date: 100*date.year + date.month)


tx_data.describe()
```

	Quantity	UnitPrice	CustomerID	InvoiceYearMonth
count	541909.000000	541909.000000	406829.000000	541909.000000
mean	9.552250	4.611114	15287.690570	201099.713989
std	218.081158	96.759853	1713.600303	25.788703
min	-80995.000000	-11062.060000	12346.000000	201012.000000
25%	1.000000	1.250000	13953.000000	201103.000000
50%	3.000000	2.080000	15152.000000	201107.000000
75%	10.000000	4.130000	16791.000000	201110.000000
max	80995.000000	38970.000000	18287.000000	201112.000000

```
tx_data['Country'].value_counts()
```

```
United Kingdom          495478
Germany                   9495
France                    8557
EIRE                      8196
Spain                     2533
Netherlands               2371
Belgium                   2069
Switzerland               2002
Portugal                  1519
Australia                 1259
Norway                    1086
Italy                      803
Channel Islands            758
Finland                    695
Cyprus                     622
Sweden                     462
Unspecified                446
Austria                    401
Denmark                    389
Japan                      358
Poland                     341
Israel                     297
USA                        291
Hong Kong                  288
Singapore                  229
Iceland                    182
Canada                     151
Greece                     146
Malta                      127
United Arab Emirates        68
European Community          61
RSA                         58
Lebanon                     45
Lithuania                   35
Brazil                      32
Czech Republic              30
Bahrain                     19
Saudi Arabia                10
Name: Country, dtype: int64
```

.

2. Recency

```
#create a generic user dataframe to keep CustomerID and new segmentation scores
tx_user = pd.DataFrame(tx_data['CustomerID'].unique())
tx_user.columns = ['CustomerID']
tx_user.head()
```

	CustomerID
0	17850.0
1	13047.0
2	12583.0
3	13748.0
4	15100.0

```
tx_uk.head()
```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	C
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	K
1	536365	71053	WHITE METAL	6	2010-12-01 08:26:00	3.39	17850.0	K

Since we are calculating recency, we need to know when last the person bought something. Let us calculate the last date of transaction for aperson.

```
#get the max purchase date for each customer and create a dataframe with it
tx_max_purchase = tx_uk.groupby('CustomerID').InvoiceDate.max().reset_index()
tx_max_purchase.columns = ['CustomerID','MaxPurchaseDate']
tx_max_purchase.head()
```

	CustomerID	MaxPurchaseDate
0	12346.0	2011-01-18 10:17:00
1	12747.0	2011-12-07 14:34:00
2	12748.0	2011-12-09 12:20:00
3	12749.0	2011-12-06 09:56:00
4	12820.0	2011-12-06 15:12:00

```
# Compare the last transaction of the dataset with last transaction dates of the individual customer IDs.
tx_max_purchase['Recency'] = (tx_max_purchase['MaxPurchaseDate'].max() - tx_max_purchase['MaxPurchaseDate']).dt.days
tx_max_purchase.head()
```

	CustomerID	MaxPurchaseDate	Recency
0	12346.0	2011-01-18 10:17:00	325
1	12747.0	2011-12-07 14:34:00	1
2	12748.0	2011-12-09 12:20:00	0
3	12749.0	2011-12-06 09:56:00	3
4	12820.0	2011-12-06 15:12:00	2

```
#merge this dataframe to our new user dataframe
tx_user = pd.merge(tx_user, tx_max_purchase[['CustomerID','Recency']], on='CustomerID')
tx_user.head()
```

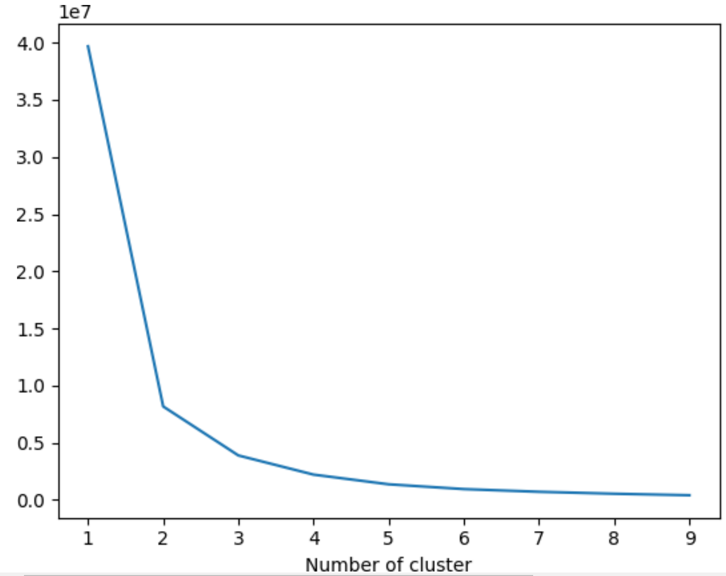
	CustomerID	Recency
0	17850.0	301
1	13047.0	31
2	13748.0	95
3	15100.0	329
4	15291.0	25

Assigning a recency score

```
from sklearn.cluster import KMeans

sse={} # error
tx_recency = tx_user[['Recency']]
for k in range(1, 10):
    kmeans = KMeans(n_clusters=k, max_iter=1000).fit(tx_recency)
    tx_recency["clusters"] = kmeans.labels_ #cluster names corresponding to recency values
    sse[k] = kmeans.inertia_ #sse corresponding to clusters
plt.figure()
plt.plot(list(sse.keys()), list(sse.values()))
plt.xlabel("Number of cluster")
plt.show()
```

```
The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarn
The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarn
The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarn
The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarn
The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarn
The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarn
The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarn
The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value
```




```
#build 4 clusters for recency and add it to dataframe
kmeans = KMeans(n_clusters=4)
tx_user['RecencyCluster'] = kmeans.fit_predict(tx_user[['Recency']])
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning:
The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
```

```
tx_user.head()
```

	CustomerID	Recency	RecencyCluster
0	17850.0	301	2
1	13047.0	31	3
2	13748.0	95	0
3	15100.0	329	2
4	15291.0	25	3

```
tx_user.groupby('RecencyCluster')['Recency'].describe()
```

RecencyCluster	count	mean	std	min	25%	50%	75%	max
0	954.0	77.679245	22.850898	48.0	59.00	72.5	93.00	131.0
1	568.0	184.625000	31.753602	132.0	156.75	184.0	211.25	244.0
2	478.0	304.393305	41.183489	245.0	266.25	300.0	336.00	373.0
3	1950.0	17.488205	13.237058	0.0	6.00	16.0	28.00	47.0

Ordering clusters

```
#function for ordering cluster numbers
def order_cluster(cluster_field_name, target_field_name,df,ascending):
    new_cluster_field_name = 'new_' + cluster_field_name
    df_new = df.groupby(cluster_field_name)[target_field_name].mean().reset_index()
    df_new = df_new.sort_values(by=target_field_name,ascending=ascending).reset_index(drop=True)
    df_new['index'] = df_new.index
    df_final = pd.merge(df,df_new[[cluster_field_name,'index']], on=cluster_field_name)
    df_final = df_final.drop([cluster_field_name],axis=1)
    df_final = df_final.rename(columns={"index":cluster_field_name})
    return df_final

tx_user = order_cluster('RecencyCluster', 'Recency',tx_user,False)


tx_user.head()
```

	CustomerID	Recency	RecencyCluster
0	17850.0	301	0
1	15100.0	329	0
2	18074.0	373	0
3	16250.0	260	0
4	13747.0	373	0

Next steps: Generate code with `tx_user` View recommended plots

```
tx_user.groupby('RecencyCluster')['Recency'].describe()
```

RecencyCluster	count	mean	std	min	25%	50%	75%	max
0	478.0	304.393305	41.183489	245.0	266.25	300.0	336.00	373.0
1	568.0	184.625000	31.753602	132.0	156.75	184.0	211.25	244.0
2	954.0	77.679245	22.850898	48.0	59.00	72.5	93.00	131.0
3	1950.0	17.488205	13.237058	0.0	6.00	16.0	28.00	47.0

3. Frequency

```
#get order counts for each user and create a dataframe with it
tx_frequency = tx_uk.groupby('CustomerID').InvoiceDate.count().reset_index()
tx_frequency.columns = ['CustomerID','Frequency']


tx_frequency.head() #how many orders does a customer have
```

	CustomerID	Frequency
0	12346.0	2
1	12747.0	103
2	12748.0	4642
3	12749.0	231
4	12820.0	59

```
#add this data to our main dataframe
tx_user = pd.merge(tx_user, tx_frequency, on='CustomerID')

tx_user.head()
```

	CustomerID	Recency	RecencyCluster	Frequency
0	17850.0	301	0	312
1	15100.0	329	0	6
2	18074.0	373	0	13
3	16250.0	260	0	24
4	13747.0	373	0	1

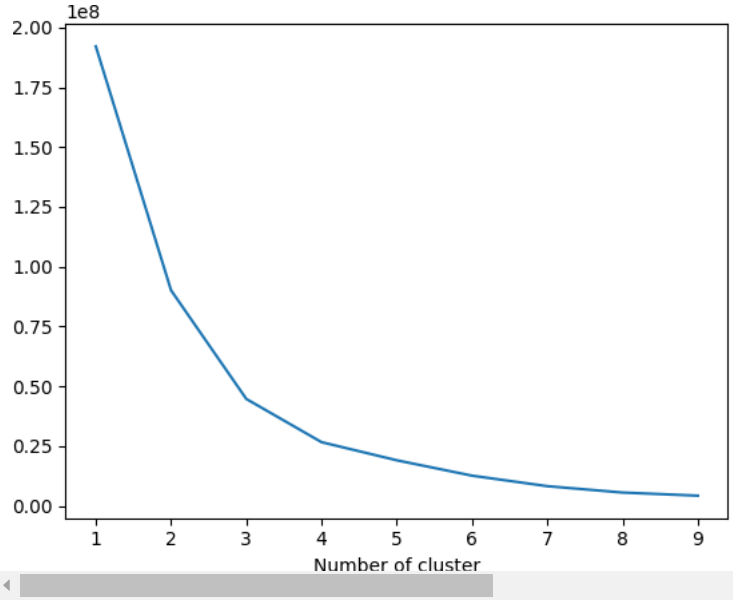
Frequency clusters

```
from sklearn.cluster import KMeans

sse={} # error
tx_recency = tx_user[['Frequency']]
for k in range(1, 10):
    kmeans = KMeans(n_clusters=k, max_iter=1000).fit(tx_recency)
    tx_recency["clusters"] = kmeans.labels_ #cluster names corresponding to recency values
    sse[k] = kmeans.inertia_ #sse corresponding to clusters
plt.figure()
plt.plot(list(sse.keys()), list(sse.values()))
plt.xlabel("Number of cluster")
plt.show()
```

```
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable
```




```
# Applying k-Means
kmeans=KMeans(n_clusters=4)
tx_user['FrequencyCluster']=kmeans.fit_predict(tx_user[['Frequency']])

#order the frequency cluster
tx_user = order_cluster('FrequencyCluster', 'Frequency', tx_user, True )
tx_user.groupby('FrequencyCluster')['Frequency'].describe()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning

The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of
```

FrequencyCluster	count	mean	std	min	25%	50%	75%	ma
0	3496.0	49.525744	44.954212	1.0	15.0	33.0	73.0	190
1	429.0	331.221445	133.856510	191.0	228.0	287.0	399.0	803
2	22.0	1313.136364	505.934524	872.0	988.5	1140.0	1452.0	2782
3	3.0	5917.666667	1805.062418	4642.0	4885.0	5128.0	6555.5	7983

Clsuter with max frequency is cluster 3, least frequency cluster is cluster 0.

4. Revenue

```
#calculate revenue for each customer
tx_uk['Revenue'] = tx_uk['UnitPrice'] * tx_uk['Quantity']
tx_revenue = tx_uk.groupby('CustomerID').Revenue.sum().reset_index()
```

```
tx_revenue.head()
```

	CustomerID	Revenue
0	12346.0	0.00
1	12747.0	4196.01
2	12748.0	29072.10
3	12749.0	3868.20
4	12820.0	942.34

```
#merge it with our main dataframe
tx_user = pd.merge(tx_user, tx_revenue, on='CustomerID')
tx_user.head()
```

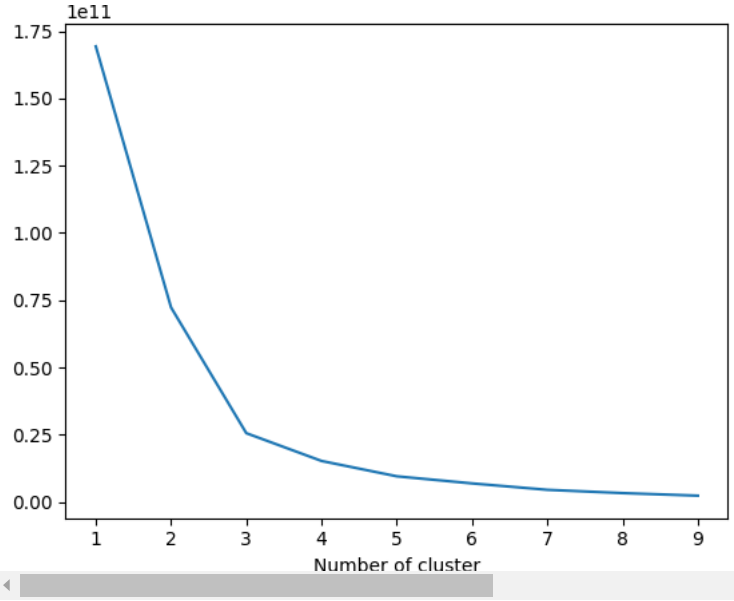
	CustomerID	Recency	RecencyCluster	Frequency	FrequencyCluster	Revenue
0	17850.0	301	0	312	1	5288.63
1	15808.0	305	0	210	1	3724.77
2	13047.0	31	3	196	1	3079.10
3	14688.0	7	3	359	1	5107.38
4	16029.0	38	3	274	1	50992.61

```
from sklearn.cluster import KMeans

sse={} # error
tx_recency = tx_user[['Revenue']]
for k in range(1, 10):
    kmeans = KMeans(n_clusters=k, max_iter=1000).fit(tx_recency)
    tx_recency["clusters"] = kmeans.labels_ #cluster names corresponding to recency values
    sse[k] = kmeans.inertia_ #sse corresponding to clusters
plt.figure()
plt.plot(list(sse.keys()), list(sse.values()))
plt.xlabel("Number of cluster")
plt.show()
```

```
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable
```



5.1. Revenue clusters

```
#apply clustering
kmeans = KMeans(n_clusters=4)
tx_user['RevenueCluster'] = kmeans.fit_predict(tx_user[['Revenue']])

#order the cluster numbers
tx_user = order_cluster('RevenueCluster', 'Revenue',tx_user,True)

#show details of the dataframe
tx_user.groupby('RevenueCluster')['Revenue'].describe()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning

The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of
```

RevenueCluster	count	mean	std	min	25%	50%
0	3687.0	907.254414	921.910820	-4287.63	263.115	572.56
1	234.0	7760.699530	3637.173671	4330.67	5161.485	6549.38
2	27.0	43070.445185	15939.249588	25748.35	28865.490	36351.42
3	2.0	221960.330000	48759.481478	187482.17	204721.250	221960.33

Cluster 3 has max revenue, cluster 0 has lowest revenue

5. Overall Score based on RFM Clsutering

```
#calculate overall score and use mean() to see details
tx_user['OverallScore'] = tx_user['RecencyCluster'] + tx_user['FrequencyCluster'] + tx_user['RevenueCluster']
tx_user.groupby('OverallScore')['Recency','Frequency','Revenue'].mean()
```

```
<ipython-input-33-ad2f8ed87503>:3: FutureWarning:

Indexing with multiple keys (implicitly converted to a tuple of keys) will be depreca
```

OverallScore	Recency	Frequency	Revenue
0	304.584388	21.995781	303.339705
1	185.362989	32.596085	498.087546
2	78.991304	46.963043	868.082991
3	20.689610	68.419590	1091.416414
4	14.892617	271.755034	3607.097114
5	9.662162	373.290541	9136.946014
6	7.740741	876.037037	22777.914815
7	1.857143	1272.714286	103954.025714
8	1.333333	5917.666667	42177.930000

Score 8 is our best customer, score 0 is our worst customer.

```
tx_user['Segment'] = 'Low-Value'
tx_user.loc[tx_user['OverallScore']>2,'Segment'] = 'Mid-Value'
tx_user.loc[tx_user['OverallScore']>4,'Segment'] = 'High-Value'
```

```
tx_user
```

	CustomerID	Recency	RecencyCluster	Frequency	FrequencyCluster	Revenue	Re
0	17850.0	301	0	312	1	5288.63	
1	14688.0	7	3	359	1	5107.38	
2	13767.0	1	3	399	1	16945.71	
3	15513.0	30	3	314	1	14520.08	
4	14849.0	21	3	392	1	7904.28	
...	...	...	...	...	...	...	...
3945	12748.0	0	3	4642	3	29072.10	
3946	17841.0	1	3	7983	3	40340.78	

---

6. Customer Lifetime Value

```
tx_uk.head()
```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	C
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	K
1	536365	71053	WHITE METAL	6	2010-12-01 08:26:00	3.39	17850.0	K

```
tx_uk['InvoiceDate'].describe()
```

```
<ipython-input-37-a12594270851>:1: FutureWarning:

Treating datetime data as categorical rather than numeric in `.describe` is deprecated and will be removed in a future version of p

count                  495478
unique                  21220
top       2011-10-31 14:41:00
freq                     1114
first     2010-12-01 08:26:00
last      2011-12-09 12:49:00
Name: InvoiceDate, dtype: object
```

We see that customers are active from 1 December 2010. Let us consider customers from March onwards (so that they are not new customers). We shall divide them into 2 subgroups. One will be where timeframe of analysing is 3 months, another will be timeframe of 6months.

```
tx_3m = tx_uk[(tx_uk.InvoiceDate < pd.to_datetime("2011-06-01")) & (tx_uk.InvoiceDate >= pd.to_datetime("2011-03-01"))].reset_index(dro
tx_6m = tx_uk[(tx_uk.InvoiceDate >= pd.to_datetime("2011-06-01")) & (tx_uk.InvoiceDate < pd.to_datetime("2011-12-01"))].reset_index(dro
```

```
#calculate revenue and create a new dataframe for it
tx_6m['Revenue'] = tx_6m['UnitPrice'] * tx_6m['Quantity']
tx_user_6m = tx_6m.groupby('CustomerID')['Revenue'].sum().reset_index()
tx_user_6m.columns = ['CustomerID','m6_Revenue']
```

```
tx_user_6m.head()
```

	CustomerID	m6_Revenue
0	12747.0	1666.11
1	12748.0	18679.01
2	12749.0	2323.04
3	12820.0	561.53
4	12822.0	918.98

:

```
#plot LTV histogram
plot_data = [
    go.Histogram(
        x=tx_user_6m['m6_Revenue']
    )
]

plot_layout = go.Layout(
        title='6m Revenue'
    )
fig = go.Figure(data=plot_data, layout=plot_layout)
pyoff.iplot(fig)
```

```
tx_user.head()
```

	CustomerID	Recency	RecencyCluster	Frequency	FrequencyCluster	Revenue	Revenu
0	17850.0	301	0	312	1	5288.63	
1	14688.0	7	3	359	1	5107.38	
2	13767.0	1	3	399	1	16945.71	

```
tx_uk.head()
```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	C
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	K
1	536365	71053	WHITE METAL	6	2010-12-01 08:26:00	3.39	17850.0	K

```
tx_merge = pd.merge(tx_user, tx_user_6m, on='CustomerID', how='left') #Only people who are in the timeline of tx_user_6m


tx_merge = tx_merge.fillna(0)



tx_graph = tx_merge.query("m6_Revenue < 50000") #because max values are ending at 50,000 as seen in graph above

plot_data = [
    go.Scatter(
        x=tx_graph.query("Segment == 'Low-Value'")['OverallScore'],
        y=tx_graph.query("Segment == 'Low-Value'")['m6_Revenue'],
        mode='markers',
        name='Low',
```

```
        marker= dict(size= 7,
            line= dict(width=1),
            color= 'blue',
            opacity= 0.8
           )
    ),
        go.Scatter(
        x=tx_graph.query("Segment == 'Mid-Value'")['OverallScore'],
        y=tx_graph.query("Segment == 'Mid-Value'")['m6_Revenue'],
        mode='markers',
        name='Mid',
        marker= dict(size= 9,
            line= dict(width=1),
            color= 'green',
            opacity= 0.5
           )
    ),
        go.Scatter(
        x=tx_graph.query("Segment == 'High-Value'")['OverallScore'],
        y=tx_graph.query("Segment == 'High-Value'")['m6_Revenue'],
        mode='markers',
        name='High',
        marker= dict(size= 11,
            line= dict(width=1),
            color= 'red',
            opacity= 0.9
           )
    ),
]

plot_layout = go.Layout(
        yaxis= {'title': "6m LTV"},
        xaxis= {'title': "RFM Score"},
        title='LTV'
    )
fig = go.Figure(data=plot_data, layout=plot_layout)
pyoff.iplot(fig)
```