











Load Dependencies and Configuration Settings

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 \*

## Exploratory Data Analysis (EDA)

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', 'skewness', 'kurtosis']

str = pd.concat([types, counts, distincts, nulls, missing\_ration, uniques, skewness, kurtosis], axis = 1, sort=True)

else:

corr = df.corr()[pred]

str = pd.concat([types, counts, distincts, nulls, missing\_ration, uniques, skewness, kurtosis, corr], axis = 1, sort=True)

corr\_col = 'corr ' + pred

cols = ['types', 'counts', 'distincts', 'nulls', 'missing ration', 'uniques', 'skewness', 'kurtosis', corr\_col ]

str.columns = cols

dtypes = str.types.value\_counts()

print('\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**\n**Data 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

datetime64[ns] 1

int64 1

Name: types, dtype: int64

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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 'Yes', '**\n**')

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('**\n**What 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\_df.shape[0]))

print('**\n**All register with negative quantity has Invoice start with:',

cs\_df.loc[(cs\_df.Quantity<0) & ~(cs\_df.CustomerID.isnull()), 'InvoiceNo'].apply(lambda x: x[0]).unique())

print('**\n**See 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']

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.UnitPrice==0) & ~(cs\_df.CustomerID.isnull())].shape[0])

cs\_df[(cs\_df.UnitPrice==0) & ~(cs\_df.CustomerID.isnull())]

*# 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

datetime64[ns] 1

int64 1

Name: types, dtype: int64

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].reset\_index().head())

cs\_df[cs\_df['StockCode'] == cat\_des\_df.StockCode.value\_counts()[cat\_des\_df.StockCode.value\_counts()>1]

.reset\_index()['index'][4]]['Description'].unique()

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 [11]:

linkcode

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

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

fig = plt.figure(figsize=(25, 7))

f1 = fig.add\_subplot(121)

g = cs\_df.groupby(["Country"]).amount.sum().sort\_values(ascending = False).plot(kind='bar', title='Amount Sales by Country')

cs\_df['Internal'] = cs\_df.Country.apply(lambda x: 'Yes' if x=='United Kingdom' else 'No' )

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=90)

plt.title('Internal Market')

plt.show()

fig = plt.figure(figsize=(25, 7))

PercentSales = np.round((cs\_df.groupby(["CustomerID"]).amount.sum().\

sort\_values(ascending = False)[:51].sum()/cs\_df.groupby(["CustomerID"]).\

amount.sum().sort\_values(ascending = False).sum()) \* 100, 2)

g = cs\_df.groupby(["CustomerID"]).amount.sum().sort\_values(ascending = False)[:51].\

plot(kind='bar', title='Top Customers: **{:3.2f}**% Sales Amount'.format(PercentSales))

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(["CustomerID"]).\

amount.sum().sort\_values(ascending = False).sum()) \* 100, 2)

g = cs\_df.groupby(["CustomerID"]).amount.sum().sort\_values(ascending = Fals

e)[:10]\

.plot(kind='bar', title='Top 10 Customers: **{:3.2f}**% Sales Amont'.format(PercentSales))

f1 = fig.add\_subplot(122)

PercentSales = np.round((cs\_df.groupby(["CustomerID"]).amount.count().\

sort\_values(ascending = False)[:10].sum()/cs\_df.groupby(["CustomerID"]).\

amount.count().sort\_values(ascending = False).sum()) \* 100, 2)

g = cs\_df.groupby(["CustomerID"]).amount.count().sort\_values(ascending = False)[:10].\

plot(kind='bar', title='Top 10 Customers: **{:3.2f}% E**vent Sales'.format(PercentSales))

AmoutSum = cs\_df.groupby(["Description"]).amount.sum().sort\_values(ascending = False)

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}% o**f Amount and **{:3.2f}% o**f Events'.\

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}% o**f Amount and **{:3.2f}% o**f Events'.\

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}% o**f Amount and **{:3.2f}% o**f Events'.\

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}% o**f Amount and **{:3.2f}% o**f Events'.\

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}% o**f Amount and **{:3.2f}% o**f Events'.\

format(PercentSales, PercentEvents))

## Customer Segmentation:

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('timedelta64[D]')

customer\_history\_df = cs\_df[['CustomerID', 'days\_since\_last\_purchase']].groupby("CustomerID").min().reset\_index()

customer\_history\_df.rename(columns={'days\_since\_last\_purchase':'recency'}, inplace=True)

customer\_history\_df.describe().transpose()

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}** )'.format(mu, sigma), loc='center')

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: **{:.6f}** )'.format(data.skew(), data.kurt()), loc='center')

plt.tight\_layout()

plt.show()

QQ\_plot(customer\_history\_df.recency, 'Recency')

#### Frequency:-

customer\_freq = (cs\_df[['CustomerID', 'InvoiceNo']].groupby(["CustomerID", 'InvoiceNo']).count().reset\_index()).\

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')

#### Monetary Value:-

customer\_monetary\_val = cs\_df[['CustomerID', 'amount']].groupby("CustomerID").sum().reset\_index()

customer\_history\_df = customer\_history\_df.merge(customer\_monetary\_val)

QQ\_plot(customer\_history\_df.amount, 'Amount')

### Data Preprocessing:-

customer\_history\_df['recency\_log'] = customer\_history\_df['recency'].apply(math.log)

customer\_history\_df['frequency\_log'] = customer\_history\_df['frequency'].apply(math.log)

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

fig = plt.figure(figsize=(20,14))

f1 = fig.add\_subplot(221); sns.regplot(x='recency', y='amount', data=customer\_history\_df)

f1 = fig.add\_subplot(222); sns.regplot(x='frequency', y='amount', data=customer\_history\_df)

f1 = fig.add\_subplot(223); sns.regplot(x='recency\_log', y='amount\_log', data=customer\_history\_df)

f1 = fig.add\_subplot(224); sns.regplot(x='frequency\_log', y='amount\_log', data=customer\_history\_df)

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 Elbow Method

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 ensure that the algorithm returns the same results each time.*

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)

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-04, random\_state=101)

model = model.fit(X\_scaled)

*# These are our fitted labels for clusters -- the first cluster has label 0, and the second has label 1.*

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()

#### Silhouette analysis on K-Means clustering

cluster\_centers = dict()

for n\_clusters **in** range(3,K\_best+1,2):

fig, (ax1, ax2, ax3) = plt.subplots(1, 3)

fig.set\_size\_inches(25, 7)

ax1.set\_xlim([-0.1, 1])

ax1.set\_ylim([0, len(X\_scaled) + (n\_clusters + 1) \* 10])

clusterer = KMeans(n\_clusters=n\_clusters, init='k-means++', n\_init=10,max\_iter=300, tol=1e-04, random\_state=101)

cluster\_labels = clusterer.fit\_predict(X\_scaled)

silhouette\_avg = silhouette\_score(X = X\_scaled, labels = cluster\_labels)

cluster\_centers.update({n\_clusters :{'cluster\_center':clusterer.cluster\_centers\_,

'silhouette\_score':silhouette\_avg,

labels':cluster\_labels}

})

sample\_silhouette\_values = silhouette\_samples(X = X\_scaled, labels = cluster\_labels)

y\_lower = 10

for i **in** range(n\_clusters):

ith\_cluster\_silhouette\_values = sample\_silhouette\_values[cluster\_labels == i]

ith\_cluster\_silhouette\_values.sort()

size\_cluster\_i = ith\_cluster\_silhouette\_values.shape[0]

y\_upper = y\_lower + size\_cluster\_i

color = cm.Spectral(float(i) / n\_clusters)

ax1.fill\_betweenx(np.arange(y\_lower, y\_upper),

0, ith\_cluster\_silhouette\_values,

facecolor=color, edgecolor=color, alpha=0.7)

ax1.text(-0.05, y\_lower + 0.5 \* size\_cluster\_i, str(i))

y\_lower = y\_upper + 10 *# 10 for the 0 samples*

ax1.set\_title("The silhouette plot for the various clusters")

ax1.set\_xlabel("The silhouette coefficient values")

ax1.set\_ylabel("Cluster label")

ax1.axvline(x=silhouette\_avg, color="red", linestyle="--")

ax1.set\_yticks([])

ax1.set\_xticks([-0.1, 0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1])

colors = cm.Spectral(cluster\_labels.astype(float) / n\_clusters)

centers = clusterer.cluster\_centers\_

y = 0

x = 1

ax2.scatter(X\_scaled[:, x], X\_scaled[:, y], marker='.', s=30, lw=0, alpha=0.7, c=colors, edgecolor='k')

ax2.scatter(centers[:, x], centers[:, y], marker='o', c="white", alpha=

1, s=200, edgecolor='k')

for i, c **in** enumerate(centers):

ax2.scatter(c[x], c[y], marker='$**%d**$' % i, alpha=1, s=50, edgecolor='k')

ax2.set\_title("**{}** Clustered data".format(n\_clusters))

ax2.set\_xlabel(feature\_vector[x])

ax2.set\_ylabel(feature\_vector[y])

x = 2

ax3.scatter(X\_scaled[:, x], X\_scaled[:, y], marker='.', s=30, lw=0, alpha=0.7, c=colors, edgecolor='k')

ax3.scatter(centers[:, x], centers[:, y], marker='o', c="white", alpha=1, s=200, edgecolor='k')

for i, c **in** enumerate(centers):

ax3.scatter(c[x], c[y], marker='$**%d**$' % i, alpha=1, s=50, edgecolor='k')

ax3.set\_title("Silhouette score: **{:1.2f}**".format(cluster\_centers[n\_clus

ters]['silhouette\_score']))

ax3.set\_xlabel(feature\_vector[x])

ax3.set\_ylabel(feature\_vector[y])

plt.suptitle(("Silhouette analysis for KMeans clustering on sample data with n\_clusters = **%d**" % n\_clusters),

fontsize=14, fontweight='bold')

plt.show()

#### Clusters Center:

features = ['amount', 'recency', 'frequency']

for i **in** range(3,K\_best+1,2):

print("for **{}** clusters the silhouette score is **{:1.2f}**".format(i, cluster\_centers[i]['silhouette\_score']))

print("Centers of each cluster:")

cent\_transformed = scaler.inverse\_transform(cluster\_centers[i]['cluster\_center'])

print(pd.DataFrame(np.exp(cent\_transformed),columns=features))

print('-'\*50)

for 3 clusters the silhouette score is 0.34

Centers of each cluster:

amount recency frequency

0 261.952265 116.604917 1.190876

1 3967.994380 7.236580 10.044493

2 1006.914317 33.819966 3.152227

--------------------------------------------------

for 5 clusters the silhouette score is 0.31

Centers of each cluster:

amount recency frequency

0 213.876290 159.060239 1.088129

1 5708.668108 4.285608 13.677542

2 1929.872406 22.442129 5.413014

3 372.314665 14.590855 1.665686

4 863.093356 100.092666 2.395562

--------------------------------------------------

for 7 clusters the silhouette score is 0.31

Centers of each cluster:

amount recency frequency

0 809.713152 107.590047 2.277095

1 2115.751105 4.436558 6.395614

2 239.805507 36.372861 1.132543

3 667.345658 13.698858 2.663541

4 205.016462 225.462781 1.082459

5 2414.804796 38.026754 6.003854

6 10182.351681 4.961015 20.687947

--------------------------------------------------

customer\_history\_df['clusters\_3'] = cluster\_centers[3]['labels']

customer\_history\_df['clusters\_5'] = cluster\_centers[5]['labels']

customer\_history\_df['clusters\_7'] = cluster\_centers[7]['labels']

display(customer\_history\_df.head())

fig = plt.figure(figsize=(20,7))

f1 = fig.add\_subplot(131)

market = customer\_history\_df.clusters\_3.value\_counts()

g = plt.pie(market, labels=market.index, autopct='**%1.1f%%**', shadow=True, startangle=90)

plt.title('3 Clusters')

f1 = fig.add\_subplot(132)

market = customer\_history\_df.clusters\_5.value\_counts()

g = plt.pie(market, labels=market.index, autopct='**%1.1f%%**', shadow=True, startangle=90)

plt.title('5 Clusters')

f1 = fig.add\_subplot(133)

market = customer\_history\_df.clusters\_7.value\_counts()

g = plt.pie(market, labels=market.index, autopct='**%1.1f%%**', shad

g = plt.pie(market, labels=market.index, autopct='**%1.1f%%**', shadow=True, startangle=90)

plt.title('7 Clusters')

plt.show()

x\_data = ['Cluster 0', 'Cluster 1','Cluster 2','Cluster 3','Cluster 4', 'Cluster 5', 'Cluster 6']

colors = ['rgba(93, 164, 214, 0.5)', 'rgba(255, 144, 14, 0.5)', 'rgba(44, 160, 101, 0.5)', 'rgba(255, 65, 54, 0.5)',

'rgba(22, 80, 57, 0.5)', 'rgba(127, 65, 14, 0.5)', 'rgba(207, 114, 255, 0.5)', 'rgba(127, 96, 0, 0.5)']

cutoff\_quantile = 95

for n\_clusters **in** range(3,K\_best+1,2):

cl = 'clusters\_' + str(n\_clusters)

for fild **in** range(0, 3):

field\_to\_plot = features[fild]

y\_data = list()

ymax = 0

for i **in** np.arange(0,n\_clusters):

y0 = customer\_history\_df[customer\_history\_df[cl]==i][field\_to\_p

lot].values

y0 = y0[y0<np.percentile(y0, cutoff\_quantile)]

if ymax < max(y0): ymax = max(y0)

y\_data.insert(i, y0)

traces = []

for xd, yd, cls **in** zip(x\_data[:n\_clusters], y\_data, colors[:n\_clusters]):

traces.append(go.Box(y=yd, name=xd, boxpoints=False, jitter=0.5, whiskerwidth=0.2, fillcolor=cls,

marker=dict( size=1, ),

line=dict(width=1),

))

layout = go.Layout(

title='Difference in **{}** with **{}** Clusters and **{:1.2f}** Score'.\

format(field\_to\_plot, n\_clusters, cluster\_centers[n\_clusters]['silhouette\_score']),

yaxis=dict( autorange=True, showgrid=True, zeroline=True,

dtick = int(ymax/10),

gridcolor='black', gridwidth=0.1, zerolinecolor='rgb(255, 255, 255)', zerolinewidth=2, ),

margin=dict(l=40, r=30, b=50, t=50, ),

paper\_bgcolor='white',

plot\_bgcolor='white',

showlegend=False

)

fig = go.Figure(data=traces, layout=layout)

py.offline.iplot(fig)

