In [1]:

```
# initialize Spark

import pyspark
from pyspark.sql import SparkSession, Row
from pyspark.sql.functions import col, asc,desc
import os
from pyspark.sql.functions import *
import time
conf = pyspark.SparkConf().setAll([('spark.master', 'local[4]'),
                                   ('spark.app.name', 'PySpark DataFrame Demo
spark = SparkSession.builder.config(conf=conf).getOrCreate()
spark.conf.set("spark.sql.repl.eagerEval.enabled", True)
print (spark.version, pyspark.version.__version__)
```

```
3.1.1 3.1.1
```

In [2]:

```
import pprint
import math
import numpy as np
import pandas as pd
from sklearn.cluster import KMeans as sklKMeans
from scipy.interpolate import make_interp_spline
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
from pyspark.sql import DataFrame
from pyspark.sql.types import StructType
from pyspark.sql.types import IntegerType,FloatType
from pyspark.sql.functions import unix_timestamp, from_unixtime
from pyspark.sql.functions import  asin, acos, sin, sqrt, cos
from pyspark.sql.functions import pow, col
from datetime import datetime
import pyspark.sql.functions as F
from pyspark.sql.types import DateType
from pyspark.sql.functions import radians
from pyspark.ml.feature import BucketedRandomProjectionLSH
from pyspark.ml.clustering import KMeans
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.feature import StandardScaler

from pyspark.ml.clustering import KMeans
from pyspark.ml.evaluation import ClusteringEvaluator

pp = pprint.PrettyPrinter(indent=4)
```

In [3]:

```python
def get_columns(df_list, counts=False):
    df_dict = {}
    for df in df_list:
        df_dict[namestr(df)] = {}
        if(counts):
            df_dict[namestr(df)]['count'] = df.count()
        else:
            df_dict[namestr(df)]['columns'] = df.schema.names
            df_dict[namestr(df)]['count'] = df.count()

    return df_dict

def namestr(obj, namespace=globals()):
    return [name for name in namespace if namespace[name] is obj][0]
```

In [4]:

```python
DATA_PATH="hdfs:///data/"
df_customers = spark.read.csv(DATA_PATH+"customers_dataset.csv", header=True)
df_customer_reviews = spark.read.csv(DATA_PATH+"customer_reviews_dataset.csv"
df_geolocation = spark.read.csv(DATA_PATH+"geolocation_dataset.csv", header=T
df_orders = spark.read.csv(DATA_PATH+"orders_dataset.csv", header=True)
df_order_items = spark.read.csv(DATA_PATH+"order_items_dataset.csv", header=T
df_order_payments = spark.read.csv(DATA_PATH+"order_payments_dataset.csv", he
df_sellers = spark.read.csv(DATA_PATH+"sellers_dataset.csv", header=True)
df_products = spark.read.csv(DATA_PATH+"products_dataset.csv", header=True)
df_product_category_name_translation = spark.read.csv(DATA_PATH+"product_cate

# This will restrict the datasets and prevent kernel crashes while running
N_TRAIN_DATA = 30000
N_TEST_DATA = 20000
```

In [5]:

```python
df_list = [df_customers,
           df_customer_reviews,
           df_geolocation,
           df_orders,
           df_order_items,
           df_order_payments,
           df_sellers,
           df_products,
           df_product_category_name_translation]

pp.pprint(get_columns(df_list))
```

```
{   'df_customer_reviews': {   'columns': [   'review_id',
                                              'order_id',
                                              'survey_score',
                                              'survey_review_title',
                                              'survey_review_content',
                                              'survey_send_date',
                                              'survey_completion_date'],
                               'count': 105189},
    'df_customers': {   'columns': [   'customer_id',
                                       'customer_unique_id',
                                       'customer_zip_code_prefix',
                                       'customer_city',
                                       'customer_state'],
```

```
                        'count': 99441},
    'df_geolocation': {   'columns': [   'geo_zip_code_prefix',
                                         'geo_lat',
                                         'geo_lng',
                                         'geo_city',
                                         'geo_state'],
                          'count': 1000163},
    'df_order_items': {   'columns': [   'order_id',
                                         'order_item_id',
                                         'product_id',
                                         'seller_id',
                                         'shipping_limit_date',
                                         'price',
                                         'freight_value'],
                          'count': 112650},
    'df_order_payments': {   'columns': [   'order_id',
                                            'payment_sequential',
                                            'payment_type',
                                            'payment_installments',
                                            'payment_value'],
                             'count': 103886},
    'df_orders': {   'columns': [   'order_id',
                                    'customer_id',
                                    'order_status',
                                    'order_purchase_timestamp',
                                    'order_approved_at',
                                    'order_carrier_delivery_date',
                                    'order_customer_delivery_date',
                                    'order_estimated_delivery_date'],
                     'count': 99441},
    'df_product_category_name_translation': {   'columns': [   'product_categ
ory_name',
                                                               'product_categ
ory_name_english'],
                                                'count': 71},
    'df_products': {   'columns': [   'product_id',
                                      'product_category_name',
                                      'product_name_lenght',
                                      'product_description_lenght',
                                      'product_photos_qty',
                                      'product_weight_g',
                                      'product_length_cm',
                                      'product_height_cm',
                                      'product_width_cm'],
                       'count': 32951},
    'df_sellers': {   'columns': [   'seller_id',
                                     'seller_zip_code_prefix',
                                     'seller_city',
                                     'seller_state'],
                      'count': 3095}}
```

# Data cleaning

- ### Drop NA

In [6]:

```
for df in df_list:
    df = df.dropna()

pp.pprint(get_columns(df_list, True))
```

```
{   'df_customer_reviews': {'count': 105189},
    'df_customers': {'count': 99441},
    'df_geolocation': {'count': 1000163},
    'df_order_items': {'count': 112650},
    'df_order_payments': {'count': 103886},
    'df_orders': {'count': 99441},
    'df_product_category_name_translation': {'count': 71},
    'df_products': {'count': 32951},
    'df_sellers': {'count': 3095}}
```

- ### Filter out only intergers for the customer reviews

In [7]:

```
df_customer_reviews = df_customer_reviews[
                        (df_customer_reviews['survey_score']=='0')|
                        (df_customer_reviews['survey_score']=='1')|
                        (df_customer_reviews['survey_score']=='2')|
                        (df_customer_reviews['survey_score']=='3')|
                        (df_customer_reviews['survey_score']=='4')|
                        (df_customer_reviews['survey_score']=='5')
                        ]
# do the type conversion of the text score to integer
df_customer_reviews = df_customer_reviews.withColumn('survey_score', df_custo
```

- ### Remove duplicate geolocations

In [8]:

```
print("Raw data count = {}".format(df_geolocation.count()))
df_geolocation = df_geolocation.dropDuplicates(['geo_zip_code_prefix'])
print("Data count after dropping duplicates = {}".format(df_geolocation.count
```

```
Raw data count = 1000163
Data count after dropping duplicates = 19015
```

# Merge order_items, product_category into products

In [9]:

```
df_grp_product_cat = df_order_items.join(df_products, on=['product_id'], how=
df_grp_product_cat = df_grp_product_cat.dropna(subset=["product_category_name
pp.pprint(get_columns([df_grp_product_cat]))
df_grp_product_cat = df_grp_product_cat.dropna()
df_order_merged = df_grp_product_cat.join(df_orders, on=['order_id'], how='in
df_order_merged = df_order_merged.dropna()
pp.pprint(get_columns([df_order_merged]))

# df_grp_product_cat.dropna().count()
```

```
{   'df_grp_product_cat': {   'columns': [   'product_id',
                                             'order_id',
                                             'order_item_id',
                                             'seller_id',
```

```
                                                    'shipping_limit_date',
                                                    'price',
                                                    'freight_value',
                                                    'product_category_name',
                                                    'product_name_lenght',
                                                    'product_description_lenght',
                                                    'product_photos_qty',
                                                    'product_weight_g',
                                                    'product_length_cm',
                                                    'product_height_cm',
                                                    'product_width_cm'],
                                     'count': 111047}}
{   'df_order_merged': {   'columns': [   'order_id',
                                          'product_id',
                                          'order_item_id',
                                          'seller_id',
                                          'shipping_limit_date',
                                          'price',
                                          'freight_value',
                                          'product_category_name',
                                          'product_name_lenght',
                                          'product_description_lenght',
                                          'product_photos_qty',
                                          'product_weight_g',
                                          'product_length_cm',
                                          'product_height_cm',
                                          'product_width_cm',
                                          'customer_id',
                                          'order_status',
                                          'order_purchase_timestamp',
                                          'order_approved_at',
                                          'order_carrier_delivery_date',
                                          'order_customer_delivery_date',
                                          'order_estimated_delivery_date'],
                           'count': 108643}}
```

## Data Conversion

In [10]:

```
# Convert to integer Type
name_type = ['shipping_limit_date']
int_col = ['product_photos_qty',
           'product_height_cm',
           'product_length_cm',
           'product_weight_g',
           'product_width_cm',
           'product_name_lenght',
           'product_description_lenght']
float_col = ['price',
             'freight_value']


for k in range(len(name_type)):
    df_order_merged = df_order_merged.drop(name_type[k])

for k in range(len(int_col)):
    df_order_merged = df_order_merged.withColumn(int_col[k], df_order_merged[

for k in range(len(float_col)):
    df_order_merged = df_order_merged.withColumn(float_col[k], df_order_merge

display(df_order_merged.schema)
```

```
StructType(List(StructField(order_id,StringType,true),StructField(product_id,
StringType,true),StructField(order_item_id,StringType,true),StructField(selle
r_id,StringType,true),StructField(price,FloatType,true),StructField(freight_v
alue,FloatType,true),StructField(product_category_name,StringType,true),Struc
tField(product_name_lenght,IntegerType,true),StructField(product_description_
lenght,IntegerType,true),StructField(product_photos_qty,IntegerType,true),Str
uctField(product_weight_g,IntegerType,true),StructField(product_length_cm,Int
egerType,true),StructField(product_height_cm,IntegerType,true),StructField(pr
oduct_width_cm,IntegerType,true),StructField(customer_id,StringType,true),Str
uctField(order_status,StringType,true),StructField(order_purchase_timestamp,S
tringType,true),StructField(order_approved_at,StringType,true),StructField(or
der_carrier_delivery_date,StringType,true),StructField(order_customer_deliver
y_date,StringType,true),StructField(order_estimated_delivery_date,StringType,
true)))
```

# Product dimesnsions

- ### multiply l x w x h and get rid of the l,w,h columns, name the resulting volume calculation as _dim

In [11]:

```
df_order_merged = df_order_merged.withColumn('product_dim_cm', df_order_merge

df_order_merged  = df_order_merged.drop('product_length_cm', 'product_height_

df_order_merged.schema.names
```

Out[11]:

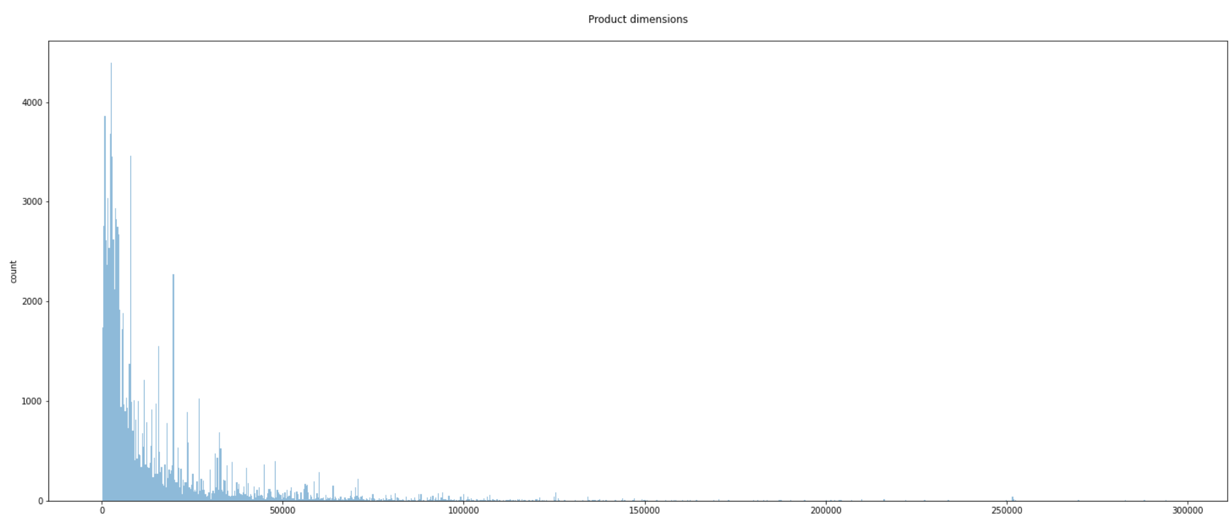
```
['order_id',
 'product_id',
 'order_item_id',
 'seller_id',
 'price',
```

```
'freight_value',
'product_category_name',
'product_name_lenght',
'product_description_lenght',
'product_photos_qty',
'product_weight_g',
'customer_id',
'order_status',
'order_purchase_timestamp',
'order_approved_at',
'order_carrier_delivery_date',
'order_customer_delivery_date',
'order_estimated_delivery_date',
'product_dim_cm']
```

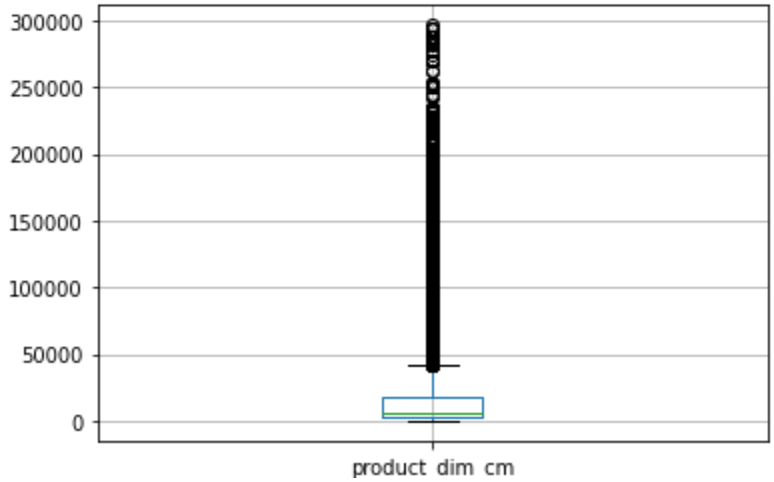
- ### Effect of product dimensions on the orders

In [12]:

```
product_dims = df_order_merged.select('product_dim_cm').toPandas()
fig = plt.figure(figsize=(25, 10))
ax = product_dims['product_dim_cm'].plot.hist(bins=1000, alpha=0.5)
ax.set_ylabel("count")
plt.title(f"Product dimensions\n")
plt.show()
product_dims.boxplot(column=['product_dim_cm'])
display(product_dims.describe())
```




| | product_dim_cm |
|---|---|
| **count** | 108643.000000 |
| **mean** | 15220.920750 |
| **std** | 23264.215598 |
| **min** | 168.000000 |
| **25%** | 2856.000000 |
| **50%** | 6552.000000 |
| **75%** | 18375.000000 |
| **max** | 296208.000000 |




# Geolocation

- ### Get the lat,lng for sellers and customers

In [13]:

```
df_geolocation.schema.names
#df_order_merged.join(df_order_merged)
df_order_merged = df_order_merged.join(df_sellers.drop('seller_city','seller_
df_order_merged = df_order_merged.join(df_customers.drop('customer_unique_id'
df_order_merged = df_order_merged.join(df_geolocation.selectExpr(" geo_zip_co
df_order_merged = df_order_merged.join(df_geolocation.selectExpr(" geo_zip_co
df_order_merged = df_order_merged.dropna()
display(get_columns([df_order_merged]))
```

```
{'df_order_merged': {'columns': ['customer_zip_code_prefix',
   'seller_zip_code_prefix',
   'customer_id',
   'seller_id',
   'order_id',
   'product_id',
   'order_item_id',
   'price',
   'freight_value',
   'product_category_name',
   'product_name_lenght',
   'product_description_lenght',
   'product_photos_qty',
   'product_weight_g',
   'order_status',
   'order_purchase_timestamp',
   'order_approved_at',
   'order_carrier_delivery_date',
   'order_customer_delivery_date',
   'order_estimated_delivery_date',
   'product_dim_cm',
   'seller_lat',
   'seller_lng',
   'customer_lat',
   'customer_lng'],
  'count': 108113}}
```

- ### Haversine calculation for distance
    - **convert all the lat,lng into radians and drop original lat,lng**

In [14]:

```
df_order_merged = df_order_merged.withColumn('seller_lat_rad',radians(df_orde
df_order_merged = df_order_merged.withColumn('seller_lng_rad',radians(df_orde
df_order_merged = df_order_merged.withColumn('customer_lat_rad',radians(df_or
df_order_merged = df_order_merged.withColumn('customer_lng_rad',radians(df_or

df_order_merged = df_order_merged.drop('seller_lat')
df_order_merged = df_order_merged.drop('seller_lng')
df_order_merged = df_order_merged.drop('customer_lat')
df_order_merged = df_order_merged.drop('customer_lng')

display(get_columns([df_order_merged]))
```

```
{'df_order_merged': {'columns': ['customer_zip_code_prefix',
   'seller_zip_code_prefix',
   'customer_id',
   'seller_id',
   'order_id',
   'product_id',
   'order_item_id',
   'price',
   'freight_value',
   'product_category_name',
   'product_name_lenght',
   'product_description_lenght',
   'product_photos_qty',
   'product_weight_g',
   'order_status',
   'order_purchase_timestamp',
   'order_approved_at',
   'order_carrier_delivery_date',
   'order_customer_delivery_date',
   'order_estimated_delivery_date',
   'product_dim_cm',
   'seller_lat_rad',
   'seller_lng_rad',
   'customer_lat_rad',
   'customer_lng_rad'],
  'count': 108113}}
```

- **Get the difference between lat, lng seller and customer**

In [15]:

```
df_order_merged = df_order_merged.withColumn('dlng',(df_order_merged['seller_
df_order_merged = df_order_merged.withColumn('dlat',(df_order_merged['seller_
display(get_columns([df_order_merged]))
```

```
{'df_order_merged': {'columns': ['customer_zip_code_prefix',
   'seller_zip_code_prefix',
   'customer_id',
   'seller_id',
   'order_id',
   'product_id',
   'order_item_id',
```

```
'price',
'freight_value',
'product_category_name',
'product_name_lenght',
'product_description_lenght',
'product_photos_qty',
'product_weight_g',
'order_status',
'order_purchase_timestamp',
'order_approved_at',
'order_carrier_delivery_date',
'order_customer_delivery_date',
'order_estimated_delivery_date',
'product_dim_cm',
'seller_lat_rad',
'seller_lng_rad',
'customer_lat_rad',
'customer_lng_rad',
'dlng',
'dlat'],
```

- **Calculations for haversine equation**

In [16]:

```
df_order_merged = df_order_merged.withColumn('dlng_sin', sin(df_order_merged[
df_order_merged = df_order_merged.withColumn('dlng_sin_square', pow(df_order_

df_order_merged = df_order_merged.withColumn('dlat_sin', sin(df_order_merged[
df_order_merged = df_order_merged.withColumn('dlat_sin_square', pow(df_order_

df_order_merged = df_order_merged.withColumn('seller_lat_rad_cos', cos(df_ord
df_order_merged = df_order_merged.withColumn('customer_lat_rad_cos', cos(df_o


df_order_merged = df_order_merged.withColumn('A',  df_order_merged['dlat_sin_

df_order_merged = df_order_merged.withColumn('A_sqrt', sqrt(df_order_merged['
df_order_merged = df_order_merged.withColumn('distance', 7912*asin(df_order_m


cal_list = [
            'dlng',
            'dlat',
            'dlng_sin',
            'dlat_sin',
            'dlng_sin_square',
            'dlat_sin_square',
            'seller_lat_rad_cos',
            'customer_lat_rad_cos',
            'seller_lat_rad',
            'customer_lat_rad',
            'seller_lng_rad',
            'seller_lat_rad',
            'customer_lng_rad',
            'customer_lat_rad',
            'A',
            'A_sqrt'
           ]
## Drop the temporary rows
for drop_col in cal_list:
    df_order_merged = df_order_merged.drop(drop_col)
```

- **Display the first 10 rows in the dataframe**

In [17]:

```
df_order_merged.limit(10)
```

Out[17]:

| customer_zip_code_prefix | seller_zip_code_prefix | customer_id | seller_id | |
|---|---|---|---|---|
| 02053 | 14940 | 8cdbc6c14192efc82... | 4a3ca9315b744ce9f... | 31688 |
| 02053 | 14940 | bb7874514104785ce... | d2374cbcbb3ca4ab1... | 7ab73 |
| 02053 | 07112 | 2c94ee4423f153e13... | 8581055ce74af1dab... | 09962 |
| 02053 | 07112 | 2c94ee4423f153e13... | 8581055ce74af1dab... | 09962 |

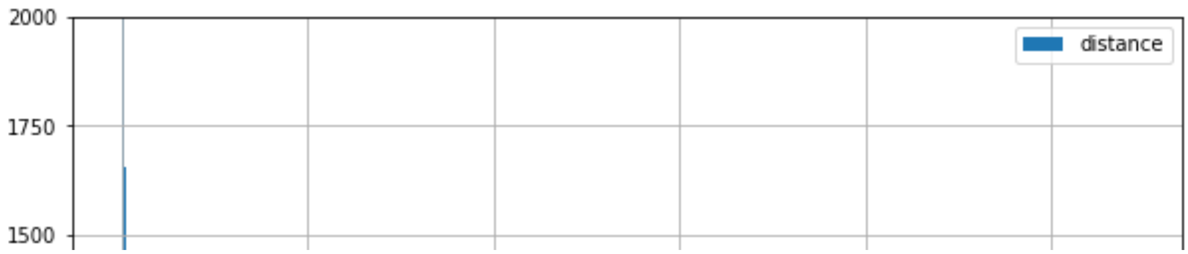
| | | | | |
|---|---|---|---|---|
| 02053 | 11701 | f0605edc06e3b81fd... | e9779976487b77c6d... | b7d6b3 |
| 02053 | 87050 | c1ddb7521d14db907... | 128639473a139ac0f... | 663830 |
| 02053 | 15025 | 774a68091890f5f1b... | 1f50f920176fa81da... | be787 |
| 02053 | 89180 | 57f0e44aca47fb9bc... | 519a7aa428f18d125... | b7b4a |
| 02053 | 89180 | d1cd0d62067d359ad... | 519a7aa428f18d125... | 27fd0 |

# Effect of distance on orders

In [18]:

```
df_dist = df_order_merged.select('distance').toPandas()
df_dist.plot.hist(bins=1500, ylim=(0,2000), grid=True, figsize=[10,8])
display(df_dist.describe())
```

| | distance |
|---|---|
| **count** | 108113.000000 |
| **mean** | 370.630199 |
| **std** | 366.329294 |
| **min** | 0.000000 |
| **25%** | 115.236940 |
| **50%** | 268.571504 |
| **75%** | 491.492344 |
| **max** | 5425.108268 |




## Merge the customer reviews

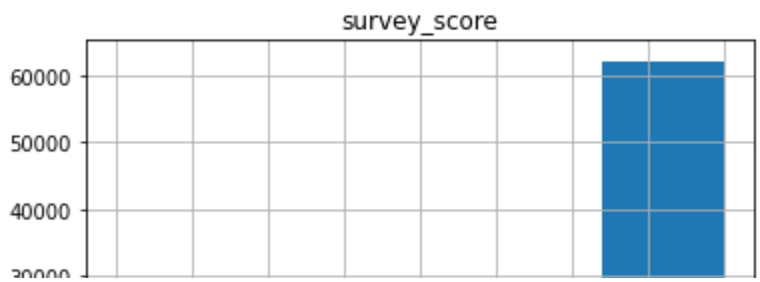
In [19]:
```
df_order_merged = df_order_merged.join(df_customer_reviews.select('order_id',
display(df_order_merged.limit(10))
```

| order_id | customer_zip_code_prefix | seller_zip_code_prefix | customer_id | |
|---|---|---|---|---|
| 3168875baaa7b1b7b... | 02053 | 14940 | 8cdbc6c14192efc82... | 4a3ca |
| 7ab737441b79ec93e... | 02053 | 14940 | bb7874514104785ce... | d2374 |
| 0996218f2d0c8ec0c... | 02053 | 07112 | 2c94ee4423f153e13... | 85810 |
| 0996218f2d0c8ec0c... | 02053 | 07112 | 2c94ee4423f153e13... | 85810 |
| b7d6b37701289908b... | 02053 | 11701 | f0605edc06e3b81fd... | e9779 |
| 663830a477534735b... | 02053 | 87050 | c1ddb7521d14db907... | 12863 |
| be787f15d899c8ef7... | 02053 | 15025 | 774a68091890f5f1b... | 1f50f |
| b7b4ac6cf1e3f2b55... | 02053 | 89180 | 57f0e44aca47fb9bc... | 519a7 |
| 27fd0c263b9d38e9c... | 02053 | 89180 | d1cd0d62067d359ad... | 519a7 |
| 989efe8965cc3848a... | 02943 | 14940 | 88b61cadc52207340... | 951e8 |

## Distribution plot of customer review

In [20]:
```
df_order_merged.select('survey_score').toPandas().hist(bins=5)
```

Out[20]:
```
array([[<AxesSubplot:title={'center':'survey_score'}>]], dtype=object)
```




# Timestamps

- **Merge the month**

In [21]:
```
get_month =  udf (lambda x: datetime.strptime(x, '%Y-%m-%d %H:%M:%S').month,
df_order_merged = df_order_merged.withColumn('month', get_month(col('order_pu

display(df_order_merged.limit(10))
```

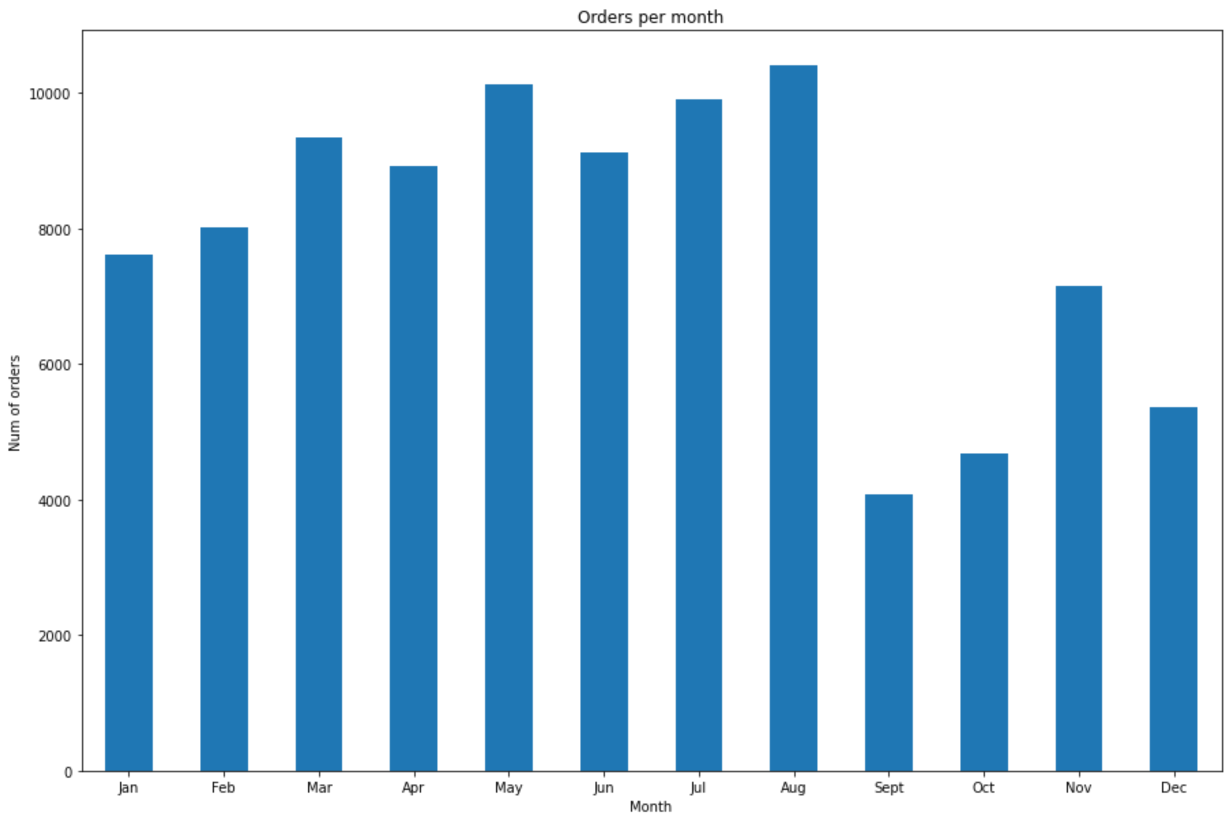
| order_id | customer_zip_code_prefix | seller_zip_code_prefix | customer_id | |
|---|---|---|---|---|
| 3168875baaa7b1b7b... | 02053 | 14940 | 8cdbc6c14192efc82... | 4a3ca |
| 7ab737441b79ec93e... | 02053 | 14940 | bb7874514104785ce... | d2374 |
| 0996218f2d0c8ec0c... | 02053 | 07112 | 2c94ee4423f153e13... | 85810 |
| 0996218f2d0c8ec0c... | 02053 | 07112 | 2c94ee4423f153e13... | 85810 |
| b7d6b37701289908b... | 02053 | 11701 | f0605edc06e3b81fd... | e9779 |
| 663830a477534735b... | 02053 | 87050 | c1ddb7521d14db907... | 12863 |
| be787f15d899c8ef7... | 02053 | 15025 | 774a68091890f5f1b... | 1f50f |
| b7b4ac6cf1e3f2b55... | 02053 | 89180 | 57f0e44aca47fb9bc... | 519a7 |
| 27fd0c263b9d38e9c... | 02053 | 89180 | d1cd0d62067d359ad... | 519a7 |
| 989efe8965cc3848a... | 02943 | 14940 | 88b61cadc52207340... | 951e8 |

display(df_order_merged.count()) display(df_order_merged.count()) display(df_order_merged.count()) display(df_order_merged.count()) display(df_order_merged.count()) display(df_order_merged.count())

- **Plot of orders per month**

In [22]:

```
df_pd_timestamps = df_order_merged.select('order_id','order_purchase_timestam
df_pd_timestamps.groupby(df_pd_timestamps['order_purchase_timestamp'].astype(
    .agg({"order_id": "nunique"})\
    .plot(figsize=(15,10), kind="bar",
          title="Orders per month",
          ylabel="Num of orders",
          xlabel="Month",
          legend=False)
plt.xticks(np.arange(0,12), ['Jan','Feb','Mar','Apr','May','Jun',
                             'Jul','Aug','Sept','Oct','Nov','Dec'],
           rotation='horizontal')
plt.show()
```




In [23]:

```
pp.pprint(get_columns([df_order_merged]))
```

```
{   'df_order_merged': {   'columns': [   'order_id',
                                          'customer_zip_code_prefix',
                                          'seller_zip_code_prefix',
                                          'customer_id',
                                          'seller_id',
                                          'product_id',
                                          'order_item_id',
                                          'price',
                                          'freight_value',
                                          'product_category_name',
                                          'product_name_lenght',
                                          'product_description_lenght',
                                          'product_photos_qty',
                                          'product_weight_g',
                                          'order_status',
```

```
'order_purchase_timestamp',
'order_approved_at',
'order_carrier_delivery_date',
'order_customer_delivery_date',
'order_estimated_delivery_date',
'product_dim_cm',
'distance',
'survey_score',
'month'],
```

# Create a incrementing id for product_catergory

In [24]:
```
df_product_category_name_translation = df_product_category_name_translation.w
df_order_merged = df_order_merged.join(df_product_category_name_translation.s
df_order_merged = df_order_merged.withColumn('cat_id', df_order_merged['cat_i
```

In [25]:
```
df_order_merged.schema
```

Out[25]:
```
StructType(List(StructField(product_category_name,StringType,true),StructFiel
d(order_id,StringType,true),StructField(customer_zip_code_prefix,StringType,t
rue),StructField(seller_zip_code_prefix,StringType,true),StructField(customer
_id,StringType,true),StructField(seller_id,StringType,true),StructField(produ
ct_id,StringType,true),StructField(order_item_id,StringType,true),StructField
(price,FloatType,true),StructField(freight_value,FloatType,true),StructField
(product_name_lenght,IntegerType,true),StructField(product_description_lengh
t,IntegerType,true),StructField(product_photos_qty,IntegerType,true),StructFi
eld(product_weight_g,IntegerType,true),StructField(order_status,StringType,tr
ue),StructField(order_purchase_timestamp,StringType,true),StructField(order_a
pproved_at,StringType,true),StructField(order_carrier_delivery_date,StringTyp
e,true),StructField(order_customer_delivery_date,StringType,true),StructField
(order_estimated_delivery_date,StringType,true),StructField(product_dim_cm,In
tegerType,true),StructField(distance,DoubleType,true),StructField(survey_scor
e,IntegerType,true),StructField(month,IntegerType,true),StructField(cat_id,In
tegerType,false)))
```

In [26]:
```
df_order_merged.limit(10)
```

Out[26]:

| product_category_name | order_id | customer_zip_code_prefix | seller_zip_code_prefix | |
|---|---|---|---|---|
| cama_mesa_banho | 3168875baaa7b1b7b... | 02053 | 14940 | 8c |
| cama_mesa_banho | 7ab737441b79ec93e... | 02053 | 14940 | bb7 |
| automotivo | 0996218f2d0c8ec0c... | 02053 | 07112 | 2c9 |
| automotivo | 0996218f2d0c8ec0c... | 02053 | 07112 | 2c9 |
| esporte_lazer | b7d6b37701289908b... | 02053 | 11701 | f0 |
| eletronicos | 663830a477534735b... | 02053 | 87050 | c1d |

| | | | | |
|---|---|---|---|---|
| ferramentas_jardim | be787f15d899c8ef7... | 02053 | 15025 | 77 |
| moveis_sala | b7b4ac6cf1e3f2b55... | 02053 | 89180 | 57 |
| moveis_sala | 27fd0c263b9d38e9c... | 02053 | 89180 | d1c |
| cama_mesa_banho | 989efe8965cc3848a | 02943 | 14940 | 88b |

# Using a smaller dataframe for the final K-Means model.

- ### Using a dataset split of 60%, 10%, 30% for the train, validation and test datasets
- ### To prevent the kernel crashes the train, test data sets have been restricted to 30,000 and 20,000 data points

In [27]:

```
features = ['distance', 'survey_score', 'month', 'price', 'freight_value' , '
features_ = ['distance', 'survey_score', 'month', 'price', 'freight_value' ,

df_order_merged_short = df_order_merged.select(features_)
#df_order_merged = df_order_merged.select(features)
VA = VectorAssembler(inputCols=features, outputCol='features')
transformed_data = VA.transform(df_order_merged_short)

train_df,val_df,  test_df = transformed_data.randomSplit([0.6, 0.1,0.3], seed
```

# Check the datatypes of the columns before fitting it to the model

In [28]:

```
display(df_order_merged_short.schema)
```

```
StructType(List(StructField(distance,DoubleType,true),StructField(survey_scor
e,IntegerType,true),StructField(month,IntegerType,true),StructField(price,Flo
atType,true),StructField(freight_value,FloatType,true),StructField(product_di
m_cm,IntegerType,true),StructField(product_weight_g,IntegerType,true),StructF
ield(product_photos_qty,IntegerType,true),StructField(cat_id,IntegerType,fals
e),StructField(product_id,StringType,true),StructField(customer_id,StringTyp
e,true)))
```

# Scale the train, validation, test datasets

In [29]:

```
print("Fit the trained data and create a scaler model")
scale = StandardScaler(inputCol='features',outputCol='scaled')

train_df = train_df.limit(N_TRAIN_DATA)
train_scaled_data = scale.fit(train_df)
train_scaled_data_output = train_scaled_data.transform(train_df)
train_scaled_data_output.show(2)

print("Fit the test data to the scaler model")
test_df = test_df.limit(N_TEST_DATA)
test_scaled_data_output = train_scaled_data.transform(test_df)
test_scaled_data_output.show(2)

print("Fit the validation data to the scaler model")
val_scaled_data_output = train_scaled_data.transform(val_df)
val_scaled_data_output.show(2)


print("Number of rows in train_df = {}, val_df = {}, test_df = {}".format(N_T
```

```
Fit the trained data and create a scaler model
+-----------------+------------+-----+-----+-------------+--------------+----
------------+-----------------+------+--------------------+-----------------
---+--------------------+--------------------+
|         distance|survey_score|month|price|freight_value|product_dim_cm|prod
uct_weight_g|product_photos_qty|cat_id|          product_id|         customer
_id|            features|              scaled|
+-----------------+------------+-----+-----+-------------+--------------+----
------------+-----------------+------+--------------------+-----------------
---+--------------------+--------------------+
|2.295458467223866|           5|    7|15.99|         7.39|          8000|
250|                 2|    12|c2c00c360a8407127...|01f841d4c59e8b763...|[2.29
545846722386...|[0.00628629665383...|
|3.021365960240162|           5|   12| 29.0|         8.72|          6000|
325|                 4|     7|170b033abb14ccc92...|683765c80b88aa6bb...|[3.02
136596024016...|[0.00827425239753...|
+-----------------+------------+-----+-----+-------------+--------------+----
------------+-----------------+------+--------------------+-----------------
---+--------------------+--------------------+
only showing top 2 rows

Fit the test data to the scaler model
+-----------------+------------+-----+-----+-------------+--------------+----
------------+-----------------+------+--------------------+-----------------
---+--------------------+--------------------+
|         distance|survey_score|month|price|freight_value|product_dim_cm|prod
uct_weight_g|product_photos_qty|cat_id|          product_id|         customer
_id|            features|              scaled|
+-----------------+------------+-----+-----+-------------+--------------+----
------------+-----------------+------+--------------------+-----------------
---+--------------------+--------------------+
|2.396908421333802|           4|    6|200.5|          9.6|         17496|
455|                 1|    15|6b75ce117b8fcc752...|71af36f53c0fe0b5b...|[2.39
690842133380...|[0.00656412546936...|
|6.430892166424179|           5|    6|24.99|         7.39|           816|
350|                 5|     8|a2da86fa759178e9e...|68cb7fbc85416655a...|[6.43
089216642417...|[0.01761151268219...|
+-----------------+------------+-----+-----+-------------+--------------+----
```

```
------------+-----------------+------+--------------------+-----------------
---+--------------------+--------------------+
only showing top 2 rows

Fit the validation data to the scaler model
+----------------+-----------+-----+-----+------------+-------------+----
------------+-----------------+------+--------------------+-----------------
---+--------------------+--------------------+
|        distance|survey_score|month|price|freight_value|product_dim_cm|prod
uct_weight_g|product_photos_qty|cat_id|          product_id|         customer
_id|            features|              scaled|
+----------------+-----------+-----+-----+------------+-------------+----
------------+-----------------+------+--------------------+-----------------
---+--------------------+--------------------+
|2.131367549634232|           5|    3|127.9|         8.66|           4693|
350|                 1|     1|1ca737c9f8f06b367...|7093a3faf4512a873...|[2.13
136754963423...|[0.00583692054840...|
|4.213825395287663|           5|    1|38.25|         9.94|           8400|
583|                 2|    14|72d3bf1d3a790f887...|0247131a290588492...|[4.21
382539528766...|[0.01153989795959...|
+----------------+-----------+-----+-----+------------+-------------+----
------------+-----------------+------+--------------------+-----------------
---+--------------------+--------------------+
only showing top 2 rows

Number of rows in train df = 30000  val df = 10847  test df = 20000
```

# Results

- ### Find the optimal 'k' for clustering

In [30]:

```
silhouette_score=[]
evaluator = ClusteringEvaluator(predictionCol='prediction', featuresCol='scal
                                metricName='silhouette', distanceMeasure='squ

for i in range(2,15):
    KMeans_algo=KMeans(featuresCol='scaled', k=i)
    KMeans_fit=KMeans_algo.fit(train_scaled_data_output)
    output=KMeans_fit.transform(train_scaled_data_output)
    score=evaluator.evaluate(output)
    silhouette_score.append(score)
    print("Silhouette Score for k = {} : {}".format(i,score))
```

```
Silhouette Score for k = 2 : 0.7154185741524448
Silhouette Score for k = 3 : 0.3098066990821567
Silhouette Score for k = 4 : 0.29358178371419563
Silhouette Score for k = 5 : 0.2987095209119094
Silhouette Score for k = 6 : 0.3041229055406726
Silhouette Score for k = 7 : 0.25531200898935774
Silhouette Score for k = 8 : 0.2844162143852261
Silhouette Score for k = 9 : 0.3003896273958905
Silhouette Score for k = 10 : 0.3003355105400472
Silhouette Score for k = 11 : 0.26685786770597814
Silhouette Score for k = 12 : 0.3005346901436662
Silhouette Score for k = 13 : 0.2608738004931076
Silhouette Score for k = 14 : 0.2477006753074689
```

- ### Plot of Cost against 'k'

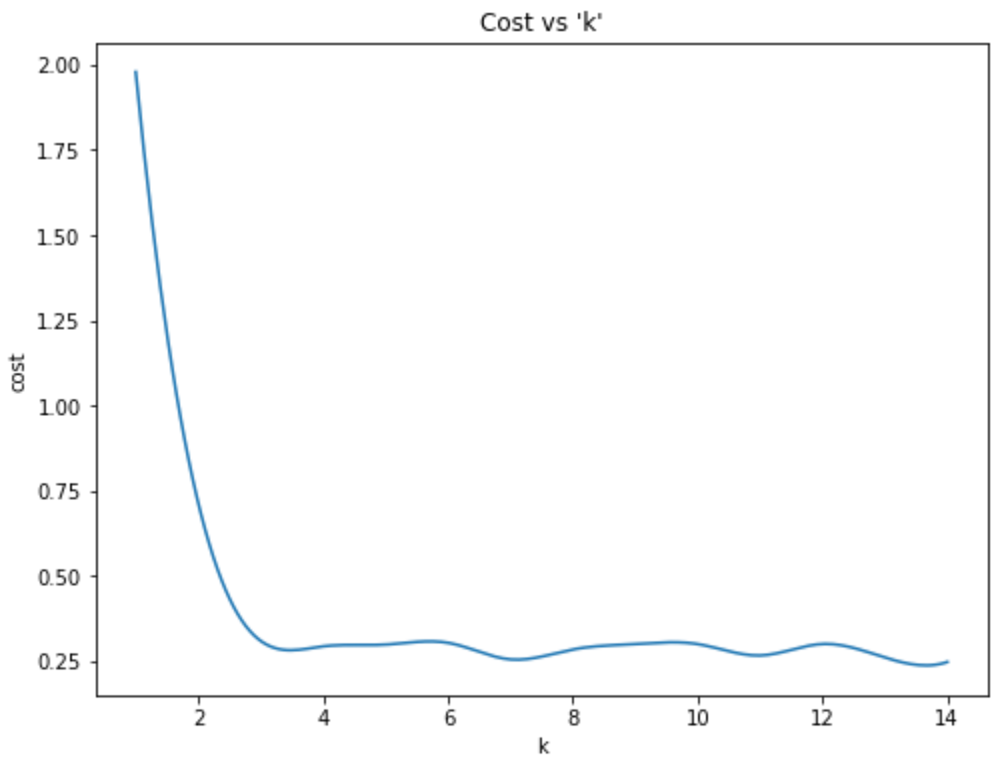
In [31]:

```
(x,y) = (range(2,15),silhouette_score)
x_new = np.linspace(1, 14, 200)
a_BSpline = make_interp_spline(x, y)
y_new = a_BSpline(x_new)

fig, ax = plt.subplots(1,1, figsize =(8,6))
ax.plot(x_new,y_new)
ax.set_title("Cost vs 'k'")
ax.set_xlabel('k')
ax.set_ylabel('cost')
```

Out[31]:

```
Text(0, 0.5, 'cost')
```




# It can be seen from the above plot that a optimal value of k = 4

In [32]:

```
KMeans_algo=KMeans(featuresCol='scaled', k=4)
KMeans_fit=KMeans_algo.fit(train_scaled_data_output)
train_output=KMeans_fit.transform(train_scaled_data_output)
score=evaluator.evaluate(train_output)
silhouette_score.append(score)
print("Silhouette Score for training data and k = {}: {}".format(4,score))
```

```
Silhouette Score for training data and k = 4: 0.29358178371419563
```

- ### Plot the clusters

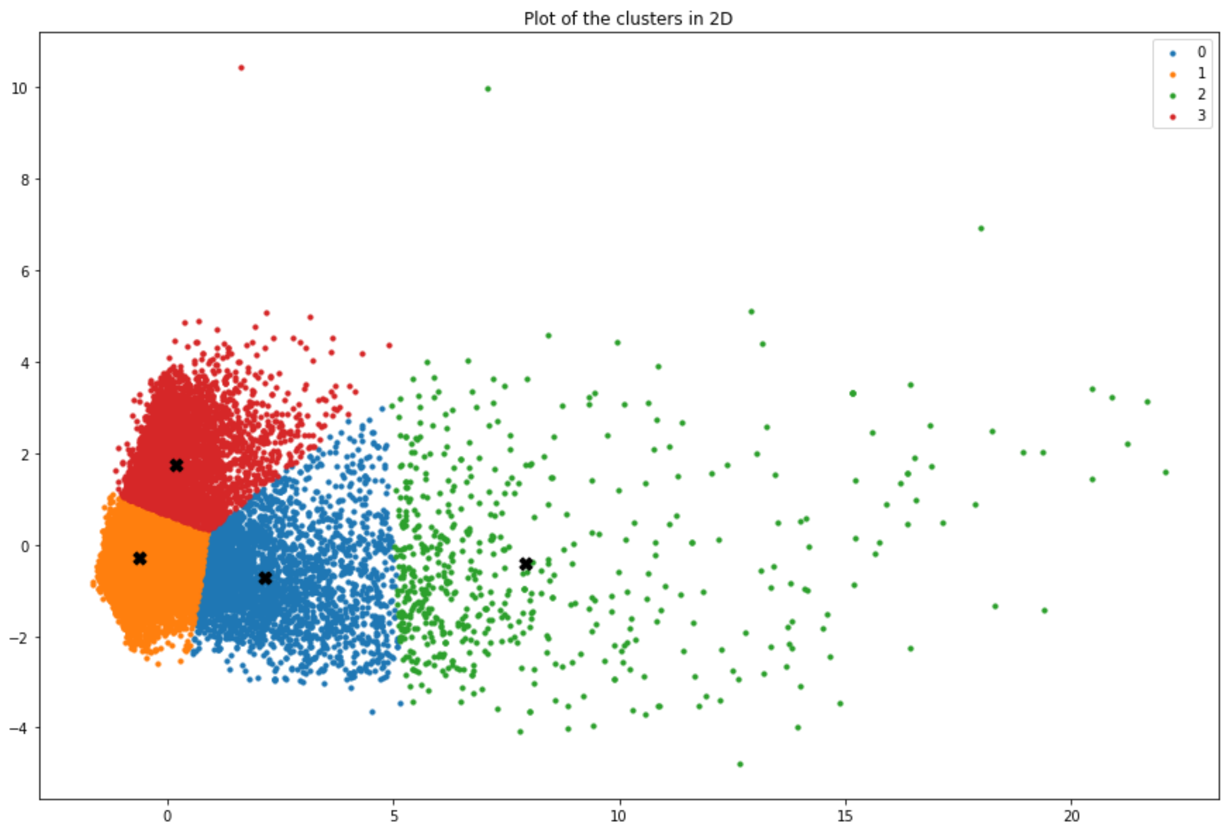
In [33]:

```
train_df = train_output.toPandas()
# unpack the dense scaled vectors
train_df_scaled = train_df['scaled'].apply(lambda x: pd.Series(x.toArray()))
# use PCA to reduce the dimension to 2
pca = PCA(2)
data_pca = pca.fit_transform(train_df_scaled)
kmeans_model = sklKMeans(n_clusters= 4)
label = kmeans_model.fit_predict(data_pca)
unique_labels = np.unique(label)
centroids = kmeans_model.cluster_centers_

fig, ax = plt.subplots(figsize=(15, 10))

for i in unique_labels:
    ax.scatter(data_pca[label == i , 0] , data_pca[label == i , 1] , label =

ax.scatter(centroids[:,0] , centroids[:,1] , s = 80, color = 'black', marker=
ax.set_title("Plot of the clusters in 2D")
plt.legend()
plt.show()
```




- ### Score for the validation data

In [34]:

```
val_output=KMeans_fit.transform(val_scaled_data_output)
score=evaluator.evaluate(val_output)
print("Silhouette Score for validation data:",score)
```

```
Silhouette Score for validation data: 0.2927932745091702
```

- ### Score for the test data

In [35]:

```
test_output=KMeans_fit.transform(test_scaled_data_output)
score=evaluator.evaluate(test_output)
print("Silhouette Score for test data:",score)
```

```
Silhouette Score for test data: 0.297981050113347
```

# Combine all the outputs together

In [36]:

```
Final_output = train_output.union(train_output.union(val_output)).dropna()
```

In [37]:

```
Final_output[Final_output['prediction']==3].sample(fraction=0.5)
```

Out[37]:

| distance | survey_score | month | price | freight_value | product_dim_cm | product_weight_ |
|---|---|---|---|---|---|---|
| 198.33453976369512 | 5 | 7 | 99.99 | 71.78 | 112251 | 825 |
| 210.36938402746296 | 4 | 11 | 1610.0 | 32.11 | 30912 | 351 |
| 252.22465480922648 | 5 | 1 | 289.0 | 46.48 | 53400 | 1015 |
| 262.16783233924406 | 4 | 11 | 139.99 | 32.01 | 64000 | 1070 |
| 308.96169782395975 | 3 | 8 | 795.0 | 63.65 | 51975 | 905 |
| 313.0098992106862 | 5 | 7 | 159.9 | 60.35 | 62640 | 747 |
| 313.23958502541706 | 5 | 5 | 219.0 | 54.06 | 83549 | 1620 |
| 313.88202851765914 | 5 | 8 | 149.99 | 57.53 | 31584 | 825 |
| 184.8796023402937 | 5 | 7 | 630.0 | 64.49 | 2560 | 2345 |
| 259.46477003230837 | 5 | 9 | 279.99 | 47.43 | 65664 | 825 |
| 425.2951521395577 | 4 | 4 | 89.9 | 23.21 | 83190 | 497 |
| 425.2951521395577 | 4 | 4 | 89.9 | 23.21 | 83190 | 497 |
| 458.45042934018784 | 4 | 7 | 134.0 | 52.95 | 92950 | 1375 |
| 678.4883856170538 | 5 | 8 | 99.99 | 41.09 | 42400 | 845 |
| 948.1341930199363 | 5 | 10 | 249.9 | 78.77 | 106580 | 565 |
| 1233.1601730870514 | 5 | 12 | 248.99 | 133.15 | 75000 | 1805 |
| 14.28207577143999 | 5 | 8 | 108.0 | 29.13 | 86640 | 1120 |
| 21.915648402363537 | 2 | 7 | 560.0 | 19.22 | 35301 | 875 |
| 224.84702076199 | 1 | 1 | 229.99 | 107.12 | 48608 | 1245 |
| 262.97240076801336 | 5 | 4 | 329.9 | 19.22 | 57420 | 960 |

only showing top 20 rows

# Product Recommendations

- ### Assume there were 5 purchases and we generated the test outputs
- ### Use the test output prediction labels and filter the Final_output dataframe
- ### Randomly choose 3 products from the cluster for each purchase

In [38]:
```
sample_test = test_output.limit(5)
```

In [39]:
```
purchase_list = sample_test.select('product_id', 'customer_id', 'prediction')
```

In [40]:
```
for item in purchase_list:
    print('Product : {} purchased by the customer :{}'.format(item[0],item[1]
    print('Top 3 recommended products for the specific customer')
    display(Final_output[Final_output['prediction']==item[2]].sample(fraction
```

```
Product : 6b75ce117b8fcc75289cb6cbe589de6c purchased by the customer :71af36f
53c0fe0b5b886ffad8154c5db
Top 3 recommended products for the specific customer
```

| distance | survey_score | month | price | freight_value | product_dim_cm | product_weight_g |
|---|---|---|---|---|---|---|
| 6.478096287460243 | 5 | 4 | 117.3 | 12.81 | 44890 | 4105 |
| 12.265322028224537 | 4 | 4 | 219.0 | 9.3 | 2700 | 292 |
| 30.60662347612332 | 5 | 12 | 79.99 | 8.75 | 8960 | 250 |

```
Product : a2da86fa759178e9e58e54aa1a144e59 purchased by the customer :68cb7fb
c85416655ad0499fcc7fdb9f7
Top 3 recommended products for the specific customer
```

| distance | survey_score | month | price | freight_value | product_dim_cm | product_weight_( |
|---|---|---|---|---|---|---|
| 6.271393215280431 | 5 | 4 | 29.9 | 7.39 | 15840 | 15( |
| 7.028794344256277 | 5 | 8 | 30.0 | 8.37 | 11270 | 50( |
| 14.914633111221498 | 5 | 1 | 134.17 | 27.38 | 43560 | 1040( |

```
Product : dfb97c88e066dc22165f31648efe1312 purchased by the customer :2c94ee4
423f153e13ce3fb15ac406a13
Top 3 recommended products for the specific customer
```

| distance | survey_score | month | price | freight_value | product_dim_cm | product_weight_g |
|---|---|---|---|---|---|---|
| 18.645394688770942 | 5 | 1 | 49.9 | 8.27 | 8000 | 100 |
| 26.103049476398382 | 5 | 3 | 110.0 | 13.05 | 50400 | 7800 |
| 26.528569471573803 | 4 | 1 | 155.0 | 8.83 | 5967 | 350 |

```
Product : 37f4d0bf85fbf875c920d460766d6a5c purchased by the customer :db7432c
b997db7083db6aaea715d3433
Top 3 recommended products for the specific customer
```

| distance | survey_score | month | price | freight_value | product_dim_cm | product_weight_g |
|---|---|---|---|---|---|---|
| 22.111542860294147 | 5 | 3 | 49.9 | 10.96 | 3136 | 600 |
| 24.10704168477952 | 5 | 12 | 118.6 | 8.09 | 2288 | 250 |
| 25.923252137933993 | 4 | 1 | 129.9 | 9.1 | 4800 | 550 |

```
Product : 2136c70bbe723d338fab53da3c03e6dc purchased by the customer :70a8cfb
1730fd53e5c15f2a62e1e5448
```

```
Top 3 recommended products for the specific customer
```

| distance | survey_score | month | price | freight_value | product_dim_cm | product_weight_( |
|---|---|---|---|---|---|---|
| 13.19973867236129 | 1 | 8 | 5.9 | 7.39 | 4590 | 25( |
| 16.74617105631575 | 1 | 5 | 579.99 | 16.96 | 16000 | 375( |
| 19.282037475200585 | 1 | 5 | 38.7 | 8.29 | 4096 | 45( |

# Stop Spark session

In [42]:

```
spark.stop()
```

In [ ]:

```
```