databricksCustomerSegmentation

```
import org.apache.spark
import pyspark
import seaborn as sns # for data visualization
import pandas as pd # for data analysis
import numpy as np # for numeric calculation
import matplotlib.pyplot as plt # for data visualization


from pyspark.sql import SparkSession
import pandas as pd


order_sparkDF = spark.read.csv("/FileStore/tables/orders.csv", header="true",
inferSchema="true")
orders_df = order_sparkDF.toPandas()
orders_df
```

	order_id	user_id	eval_set	order_number	order_dow	order_hour_of_day	days_since_pı
0	2539329	1	prior	1	2	8	
1	2398795	1	prior	2	3	7	
2	473747	1	prior	3	3	12	
3	2254736	1	prior	4	4	7	
4	431534	1	prior	5	4	15	
...	...	...	...	...	...	...	
3421078	2266710	206209	prior	10	5	18	
3421079	1854736	206209	prior	11	4	10	
3421080	626363	206209	prior	12	1	12	
3421081	2977660	206209	prior	13	1	12	
3421082	272231	206209	train	14	6	14	

3421083 rows × 7 columns

```
order_products_prior_sparkDF =
spark.read.csv("/FileStore/tables/order_products__prior.csv", header="true",
inferSchema="true")
order_products_prior_df = order_products_prior_sparkDF.toPandas()
order_products_prior_df
```

	order_id	product_id	add_to_cart_order	reordered
0	2	33120	1	1
1	2	28985	2	1
2	2	9327	3	0
3	2	45918	4	1
4	2	30035	5	0
...	...	...	...	...
32434484	3421083	39678	6	1
32434485	3421083	11352	7	0
32434486	3421083	4600	8	0
32434487	3421083	24852	9	1
32434488	3421083	5020	10	1

32434489 rows × 4 columns

```
urlA = 'https://drive.google.com/file/d/1W8bNivEj7H0WXqZx1X83fQEYz4A3XadY/view?
usp=sharing'
urlA2 = 'https://drive.google.com/uc?id=' + urlA.split('/')[-2]
aisles_df = pd.read_csv(urlA2)
aisles_df
```

	aisle_id	aisle
0	1	prepared soups salads
1	2	specialty cheeses
2	3	energy granola bars
3	4	instant foods
4	5	marinades meat preparation
...	...	...
129	130	hot cereal pancake mixes
130	131	dry pasta
131	132	beauty
132	133	muscles joints pain relief
133	134	specialty wines champagnes

134 rows × 2 columns

```
urlD = 'https://drive.google.com/file/d/1unatDL4jGx5CCHYN2Q9YnDjnq43AgtJp/view?
usp=sharing'
urlD2 = 'https://drive.google.com/uc?id=' + urlD.split('/')[-2]
departments_df = pd.read_csv(urlD2)
departments_df
```

	department_id	department
0	1	frozen
1	2	other
2	3	bakery
3	4	produce
4	5	alcohol
5	6	international
6	7	beverages
7	8	pets
8	9	dry goods pasta
9	10	bulk
10	11	personal care
11	12	meat seafood
12	13	pantry
13	14	breakfast
14	15	canned goods
15	16	dairy eggs
16	17	household
17	18	babies
18	19	snacks
19	20	deli
20	21	missing

```
urlOPT =
'https://drive.google.com/file/d/1IyZbHlrD8zXB8zhgx2XKxt812THThGRu/view?
usp=sharing'
urlOPT2 = 'https://drive.google.com/uc?id=' + urlOPT.split('/')[-2]
order_products_train = pd.read_csv(urlOPT2)
order_products_train
```

	order_id	product_id	add_to_cart_order	reordered
0	1	49302	1	1
1	1	11109	2	1
2	1	10246	3	0
3	1	49683	4	0
4	1	43633	5	1
...	...	...	...	...
1384612	3421063	14233	3	1
1384613	3421063	35548	4	1
1384614	3421070	35951	1	1
1384615	3421070	16953	2	1
1384616	3421070	4724	3	1

1384617 rows × 4 columns

```
urlP = 'https://drive.google.com/file/d/1Gkwkg56XgLzX_hyZDjEyHyRbcSjuWKp3/view?
usp=sharing'
urlP2 = 'https://drive.google.com/uc?id=' + urlP.split('/')[-2]
products = pd.read_csv(urlP2)
products
```

	product_id	product_name	aisle_id	department_id
0	1	Chocolate Sandwich Cookies	61	19
1	2	All-Seasons Salt	104	13
2	3	Robust Golden Unsweetened Oolong Tea	94	7
3	4	Smart Ones Classic Favorites Mini Rigatoni Wit...	38	1
4	5	Green Chile Anytime Sauce	5	13
...	...	...	...	...
49683	49684	Vodka, Triple Distilled, Twist of Vanilla	124	5
49684	49685	En Croute Roast Hazelnut Cranberry	42	1
49685	49686	Artisan Baguette	112	3
49686	49687	Smartblend Healthy Metabolism Dry Cat Food	41	8
49687	49688	Fresh Foaming Cleanser	73	11

49688 rows × 4 columns

```
#merging the data together
temp = pd.merge(order_products_prior_df, products, on=["product_id"])
temp = pd.merge(temp, orders_df, on=["order_id"])
temp = pd.merge(temp, aisles_df, on=["aisle_id"])
data = pd.merge(temp, departments_df, on=["department_id"])
del temp
data
```

	order_id	product_id	add_to_cart_order	reordered	product_name	aisle_id	department
0	2	33120	1	1	Organic Egg Whites	86	
1	26	33120	5	0	Organic Egg Whites	86	
2	120	33120	13	0	Organic Egg Whites	86	
3	327	33120	5	1	Organic Egg Whites	86	
4	390	33120	28	1	Organic Egg Whites	86	
...	...	...	...	...	...	...	
32434484	3243156	20731	1	0	Straight Sherry	134	
32434485	860862	30582	1	0	Natural Champagne	134	
32434486	1333472	27906	1	0	Imperial Champagne	134	

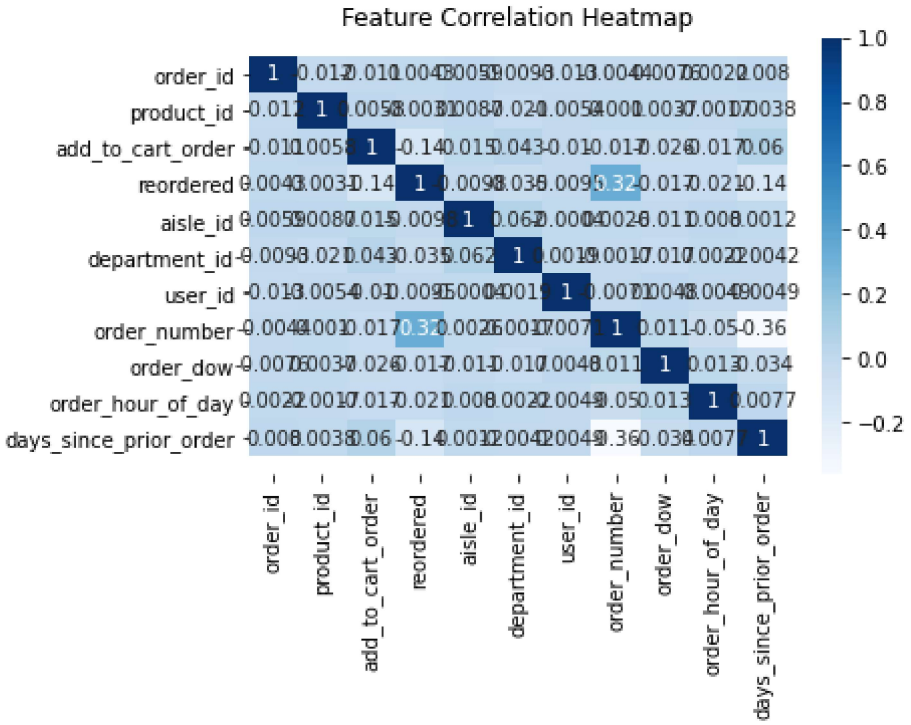
```
data = data.sample(frac = 0.0005)
```

```
data.corr()
```

	order_id	product_id	add_to_cart_order	reordered	aisle_id	department
order_id	1.000000	-0.011740	-0.010772	0.004317	0.005946	-0.009
product_id	-0.011740	1.000000	0.005812	-0.003062	0.008668	-0.020
add_to_cart_order	-0.010772	0.005812	1.000000	-0.136742	0.014590	0.043
reordered	0.004317	-0.003062	-0.136742	1.000000	-0.009840	-0.034
aisle_id	0.005946	0.008668	0.014590	-0.009840	1.000000	0.061
department_id	-0.009332	-0.020733	0.043004	-0.034592	0.061887	1.000
user_id	-0.012887	-0.005413	-0.010497	-0.009549	-0.000400	0.001
order_number	-0.004353	0.001016	-0.017469	0.319192	0.002632	-0.001
order_dow	-0.007574	0.003655	-0.025557	-0.017144	-0.010507	-0.016
order_hour_of_day	0.002175	-0.001675	-0.017339	-0.021478	0.007959	0.002
days_since_prior_order	0.008049	0.003848	0.060195	-0.142087	0.001155	-0.004

```
corr = data.corr()

sns.heatmap(corr, annot=True, cmap='Blues')
b, t = plt.ylim()
plt.ylim(b+0.5, t-0.5)
plt.title("Feature Correlation Heatmap")
plt.show()
```



```
data.columns
```

```
Out[13]: Index(['order_id', 'product_id', 'add_to_cart_order', 'reordered',
       'product_name', 'aisle_id', 'department_id', 'user_id', 'eval_set',
       'order_number', 'order_dow', 'order_hour_of_day',
       'days_since_prior_order', 'aisle', 'department'],
      dtype='object')
```

```
unwanted_cols =
['order_id','product_id','add_to_cart_order','eval_set','order_number','aisle_i
d','department_id']
```

```
data = data.drop(columns=[i for i in unwanted_cols])
```

```
data.head()
```

	reordered	product_name	user_id	order_dow	order_hour_of_day	days_since_prior_ord
24505735	1	Parmesan Cheese Crisps	66879	0	9	8
24719730	0	Organic 85% Cacao Dark Chocolate Bar	76923	0	11	Na
20290346	1	Alpine Spring Water	97035	1	19	24

```
sparkDF = spark.createDataFrame(data)
```

```
sparkDF.createOrReplaceTempView("kdata")
```

```
%fs rm -r dbfs:/user/hive/warehouse/kdata
```

```
res0: Boolean = false
```

```
%sql
select * from kdata
```

	reordered	product_name
1	1	Parmesan Cheese Crisps
2	0	Organic 85% Cacao Dark Chocolate Bar

3	1	Alpine Spring Water
4	0	Raspberries
5	1	Sweet Red Grape Tomatoes
6	0	Fresh Pressed Virgin Coconut Oil
7	0	Organic Rainbow Carrots

Truncated results, showing first 1000 rows.

```
df = sqlContext.sql("SELECT * from kdata")
```

```
df_original = df
```

```
df.show()
```

```
+---------+--------------------+-------+---------+-----------------+----------
-----------+--------------------+------------+
|reordered|        product_name|user_id|order_dow|order_hour_of_day|days_since
_prior_order|               aisle|  department|
+---------+--------------------+-------+---------+-----------------+----------
-----------+--------------------+------------+
|        1|Parmesan Cheese C...|  66879|        0|                9|
8.0|      chips pretzels|      snacks|
|        0|Organic 85% Cacao...|  76923|        0|               11|
null|     candy chocolate|      snacks|
|        1| Alpine Spring Water|  97035|        1|               19|
24.0|water seltzer spa...|   beverages|
|        0|         Raspberries| 102197|        3|               13|
25.0|    packaged produce|     produce|
|        1|Sweet Red Grape T...|  19390|        0|               16|
7.0|    fresh vegetables|     produce|
|        0|Fresh Pressed Vir...| 158634|        1|               12|
6.0|       oils vinegars|      pantry|
|        0|Organic Rainbow C...| 147839|        2|                8|
11.0|packaged vegetabl...|     produce|
|        1|Organic Boneless ...|  45720|        1|               21|
```

```
df.printSchema()
```

```
root
 |-- reordered: integer (nullable = true)
 |-- product_name: string (nullable = true)
 |-- user_id: integer (nullable = true)
 |-- order_dow: integer (nullable = true)
 |-- order_hour_of_day: integer (nullable = true)
 |-- days_since_prior_order: double (nullable = true)
```

```
 |-- aisle: string (nullable = true)
 |-- department: string (nullable = true)
```

```
# # drop user id cz we dont need it while clustering
# sparkDF= sparkDF.drop('user_id')
```

```
print((df.count(), len(df.columns)))
```

```
(16217, 8)
```

```
categoricalColumns = [item[0] for item in df.dtypes if
item[1].startswith('string') ]
```

```
categoricalColumns
```

```
Out[26]: ['product_name', 'aisle', 'department']
```

```
from pyspark.ml import Pipeline
from pyspark.ml.feature import StringIndexer, VectorAssembler, Imputer
```

```
c = 'days_since_prior_order'
sparkDF = df.withColumn(c, df[c].cast('long'))
```

```
df.printSchema()
```

```
root
 |-- reordered: integer (nullable = true)
 |-- product_name: string (nullable = true)
 |-- user_id: integer (nullable = true)
 |-- order_dow: integer (nullable = true)
 |-- order_hour_of_day: integer (nullable = true)
 |-- days_since_prior_order: double (nullable = true)
 |-- aisle: string (nullable = true)
 |-- department: string (nullable = true)
```

```
from collections import defaultdict

data_types = defaultdict(list)
for entry in df.schema.fields:
   data_types[str(entry.dataType)].append(entry.name)
```

```
strings_used = [var for var in data_types["StringType"]]

missing_data_fill = {}
for var in strings_used:
   missing_data_fill[var] = "missing"

df = df.fillna(missing_data_fill)


from pyspark.ml import Pipeline
from pyspark.ml.feature import OneHotEncoder, StringIndexer

stage_string = [StringIndexer(inputCol= c, outputCol= c+"_string_encoded") for
c in strings_used]
stage_one_hot = [OneHotEncoder(inputCol= c+"_string_encoded", outputCol= c+
"_one_hot") for c in strings_used]

ppl = Pipeline(stages= stage_string + stage_one_hot)
sparkDF = ppl.fit(df).transform(df)


sparkDF.show()
```

```
+---------+--------------------+-------+---------+-----------------+----------
-----------+--------------------+-----------+--------------------------+---
-----------------+------------------------+--------------------+------------
---+------------------+
|reordered|        product_name|user_id|order_dow|order_hour_of_day|days_since
_prior_order|               aisle|  department|product_name_string_encoded|ais
le_string_encoded|department_string_encoded|product_name_one_hot|   aisle_one_
hot|department_one_hot|
+---------+--------------------+-------+---------+-----------------+----------
-----------+--------------------+-----------+--------------------------+---
-----------------+------------------------+--------------------+------------
---+------------------+
|        1|Parmesan Cheese C...|  66879|        0|                9|
8.0|      chips pretzels|      snacks|                    1239.0|
8.0|                     2.0| (6215,[1239],[1.0])| (133,[8],[1.0])|    (20,
[2],[1.0])|
|        0|Organic 85% Cacao...|  76923|        0|               11|
null|     candy chocolate|      snacks|                     783.0|
30.0|                      2.0|  (6215,[783],[1.0])|(133,[30],[1.0])|    (20,
[2],[1.0])|
|        1| Alpine Spring Water|  97035|        1|               19|
```

```
sparkDF.printSchema()
```

```
root
 |-- reordered: integer (nullable = true)
 |-- product_name: string (nullable = false)
 |-- user_id: integer (nullable = true)
 |-- order_dow: integer (nullable = true)
 |-- order_hour_of_day: integer (nullable = true)
 |-- days_since_prior_order: double (nullable = true)
 |-- aisle: string (nullable = false)
 |-- department: string (nullable = false)
 |-- product_name_string_encoded: double (nullable = false)
 |-- aisle_string_encoded: double (nullable = false)
 |-- department_string_encoded: double (nullable = false)
 |-- product_name_one_hot: vector (nullable = true)
 |-- aisle_one_hot: vector (nullable = true)
 |-- department_one_hot: vector (nullable = true)
```

```
pri_col ='days_since_prior_order'
sparkDF = sparkDF.withColumn(pri_col,sparkDF[pri_col].cast('int'))
```

```
sparkDF.printSchema()
```

```
root
 |-- reordered: integer (nullable = true)
 |-- product_name: string (nullable = false)
 |-- user_id: integer (nullable = true)
 |-- order_dow: integer (nullable = true)
 |-- order_hour_of_day: integer (nullable = true)
 |-- days_since_prior_order: integer (nullable = true)
 |-- aisle: string (nullable = false)
 |-- department: string (nullable = false)
 |-- product_name_string_encoded: double (nullable = false)
 |-- aisle_string_encoded: double (nullable = false)
 |-- department_string_encoded: double (nullable = false)
 |-- product_name_one_hot: vector (nullable = true)
 |-- aisle_one_hot: vector (nullable = true)
 |-- department_one_hot: vector (nullable = true)
```

```
from collections import defaultdict

data_types = defaultdict(list)
for entry in sparkDF.schema.fields:
   data_types[str(entry.dataType)].append(entry.name)
```

```
for c in data_types["IntegerType"]:
  sparkDF = sparkDF.withColumn(c+ "_cast_to_double", sparkDF[c].cast("double"))

cast_vars = [var for var in  sparkDF.columns if
var.endswith("_cast_to_double")]
cast_vars_imputed  = [var+ "imputed" for var in cast_vars]
```

```
cast_vars
```

```
Out[39]: ['reordered_cast_to_double',
 'user_id_cast_to_double',
 'order_dow_cast_to_double',
 'order_hour_of_day_cast_to_double',
 'days_since_prior_order_cast_to_double']
```

```
# remove cz we dont want user id in clustering data features
cast_vars.remove('user_id_cast_to_double')
cast_vars_imputed.remove('user_id_cast_to_doubleimputed')
```

```
from pyspark.ml.feature import Imputer

imputer_for_cast_vars = Imputer(inputCols = cast_vars, outputCols =
cast_vars_imputed)
sparkDF = imputer_for_cast_vars.fit(sparkDF).transform(sparkDF)
```

```
sparkDF.show()
```

```
+---------+--------------------+-------+---------+-----------------+----------
-----------+--------------------+-----------+--------------------------+---
----------------+------------------------+--------------------+------------
---+------------------+------------------------+--------------------+-------
----------------+--------------------------------+--------------------------
----------+--------------------------------+-------------------------------+---
-----------------------------------+-----------------------------------------
---+
|reordered|        product_name|user_id|order_dow|order_hour_of_day|days_since
_prior_order|               aisle|  department|product_name_string_encoded|ais
le_string_encoded|department_string_encoded|product_name_one_hot|   aisle_one_
hot|department_one_hot|reordered_cast_to_double|user_id_cast_to_double|order_d
ow_cast_to_double|order_hour_of_day_cast_to_double|days_since_prior_order_cast
_to_double|reordered_cast_to_doubleimputed|order_dow_cast_to_doubleimputed|ord
er_hour_of_day_cast_to_doubleimputed|days_since_prior_order_cast_to_doubleimpu
```

```
ted
+---------+--------------------+-------+---------+-----------------+----------
-----------+--------------------+-----------+--------------------------+---
----------------+------------------------+--------------------+------------
---+------------------+------------------------+---------------------+-------
----------------+--------------------------------+--------------------------
```

```
from pyspark.ml.feature import VectorAssembler

features = cast_vars_imputed + [var + "_one_hot" for var in strings_used]

vector_assembler = VectorAssembler(inputCols = features, outputCol= "features")
assembled_data=vector_assembler.transform(sparkDF)


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

# silhouette_scores=[]
# evaluator = ClusteringEvaluator(featuresCol='features', \
# metricName='silhouette', distanceMeasure='squaredEuclidean')

# for K in range(2,5):

#     KMeans_=KMeans(featuresCol='features', k=K)

#     KMeans_fit=KMeans_.fit(assembled_data)

#     KMeans_transform=KMeans_fit.transform(assembled_data)

#     evaluation_score=evaluator.evaluate(KMeans_transform)

#     silhouette_scores.append(evaluation_score)



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


# import matplotlib.pyplot as plt
# fig, ax = plt.subplots(1,1, figsize =(10,8))
# ax.plot(range(2,5),silhouette_scores)
# ax.set_xlabel('Number of Clusters')
# ax.set_ylabel('Silhouette Score')
```

```
KMeans_=KMeans(featuresCol='features', k=3)
KMeans_Model=KMeans_.fit(assembled_data)
KMeans_Assignments=KMeans_Model.transform(assembled_data)


ctr=[]
centers = KMeans_Model.clusterCenters()
for center in centers:
    ctr.append(center)
    print(center)



[4.77741585e-01 2.76959826e+00 1.34028230e+01 ... 2.60586319e-03
 3.25732899e-03 1.95439739e-03]
[4.97542533e-01 2.61739130e+00 1.36207940e+01 ... 3.02457467e-03
 7.56143667e-04 7.56143667e-04]
[6.80049069e-01 2.76636556e+00 1.34173079e+01 ... 2.89952046e-03
 1.89584030e-03 8.92160143e-04]


# Evaluate clustering by computing Silhouette score

#If this number is negative, the data cannot be separated at all.
#Values closer to 1 indicate maximum separation.
#Values close to zero mean the data could barely be separated.
#In this example, 0.72 is not bad.

evaluator = ClusteringEvaluator()

silhouette = evaluator.evaluate(KMeans_Assignments)
print("Silhouette with squared euclidean distance = " + str(silhouette))

Silhouette with squared euclidean distance = 0.5013008724676526


transformed = KMeans_Assignments.select('user_id', 'prediction')
rows = transformed.collect()
print(rows[:3])

[Row(user_id=66879, prediction=2), Row(user_id=76923, prediction=0), Row(user_i
d=97035, prediction=1)]


rows
```

```
Out[51]: [Row(user_id=66879, prediction=2),
 Row(user_id=76923, prediction=0),
 Row(user_id=97035, prediction=1),
```

```
 Row(user_id=102197, prediction=1),
 Row(user_id=19390, prediction=2),
 Row(user_id=158634, prediction=2),
 Row(user_id=147839, prediction=0),
 Row(user_id=45720, prediction=0),
 Row(user_id=52228, prediction=2),
 Row(user_id=135922, prediction=2),
 Row(user_id=196031, prediction=0),
 Row(user_id=173381, prediction=2),
 Row(user_id=11199, prediction=1),
 Row(user_id=27805, prediction=1),
 Row(user_id=165030, prediction=1),
 Row(user_id=192761, prediction=2),
 Row(user_id=145755, prediction=1),
 Row(user_id=156492, prediction=0),
 Row(user_id=1704, prediction=2),
 Row(user_id=39851, prediction=0),
 Row(user_id=37716, prediction=2)
```

```
df_pred_less = spark.createDataFrame(rows)
df_pred_less.show()
```

```
+-------+----------+
|user_id|prediction|
+-------+----------+
|  66879|         2|
|  76923|         0|
|  97035|         1|
| 102197|         1|
|  19390|         2|
| 158634|         2|
| 147839|         0|
|  45720|         0|
|  52228|         2|
| 135922|         2|
| 196031|         0|
| 173381|         2|
|  11199|         1|
|  27805|         1|
| 165030|         1|
| 192761|         2|
| 145755|         1|
| 156492|         0|
```

```
df_pred_less
```

```
Out[53]: DataFrame[user_id: bigint, prediction: bigint]
```

```
sparkDF_original = spark.createDataFrame(data)
```

```
df_pred_join = df_pred_less.join(sparkDF_original, 'user_id')
df_pred_join.show()
```

```
+-------+----------+---------+--------------------+---------+-----------------
+----------------------+--------------------+------------+
|user_id|prediction|reordered|        product_name|order_dow|order_hour_of_day
|days_since_prior_order|               aisle|  department|
+-------+----------+---------+--------------------+---------+-----------------
+----------------------+--------------------+------------+
|  52412|         2|        1|Pure & Natural So...|        3|               13
|                   3.0|other creams cheeses|  dairy eggs|
|  30562|         1|        1|       Coconut Water|        3|                7
|                  30.0|       juice nectars|   beverages|
|  60033|         2|        1|Organic Balsamic ...|        1|               15
|                   0.0|       oils vinegars|      pantry|
|  60033|         2|        1|Organic Balsamic ...|        1|               15
|                   0.0|       oils vinegars|      pantry|
|  55474|         1|        1|Bag of Organic Ba...|        1|               17
|                  28.0|        fresh fruits|     produce|
| 176711|         0|        1|Organic Navel Orange|        2|                0
|                  18.0|        fresh fruits|     produce|
| 193283|         0|        0|Milk Chocolate Pe...|        5|               13
|                  10.0|     candy chocolate|      snacks|
| 197940|         0|        0|Orange & Lemon Fl...|        1|               15
```

```
print((df_pred_join.count(), len(df_pred_join.columns)))
```

```
(19589, 9)
```

```
sparkDF_original.printSchema()
```

```
root
 |-- reordered: integer (nullable = true)
 |-- product_name: string (nullable = true)
 |-- user_id: integer (nullable = true)
 |-- order_dow: integer (nullable = true)
 |-- order_hour_of_day: integer (nullable = true)
 |-- days_since_prior_order: double (nullable = true)
 |-- aisle: string (nullable = true)
 |-- department: string (nullable = true)
```

```
df_pred_join_fin = df_pred_join.drop_duplicates()
```

```
print((df_pred_join_fin.count(), len(df_pred_join_fin.columns)))
```

```
(17309, 9)
```

```
# from pyspark.ml.feature import PCA as PCAml
# pca = PCAml(k=2, inputCol="features", outputCol="pca")
# pca_model = pca.fit(assembled_data)
# pca_transformed = pca_model.transform(assembled_data)
```

```
  Cancelled
```

```
# import numpy as np
# x_pca = np.array(pca_transformed.rdd.map(lambda row: row.pca).collect())
```

```
  Command skipped
```

```
# cluster_assignment = np.array(KMeans_Assignments.rdd.map(lambda row:
row.prediction).collect()).reshape(-1,1)
```

```
  Command skipped
```

```
# import seaborn as sns
# import matplotlib.pyplot as plt

# pca_data = np.hstack((x_pca,cluster_assignment))

# pca_df = pd.DataFrame(data=pca_data, columns=("1st_principal",
"2nd_principal","cluster_assignment"))
# sns.FacetGrid(pca_df,hue="cluster_assignment", height=6).map(plt.scatter,
'1st_principal', '2nd_principal' ).add_legend()

# plt.show()
```

```
  Command skipped
```

Plot parallel

```
para_df_join = df_pred_less.join(sparkDF, 'user_id')
para_df_join.show()
```

```
+-------+----------+---------+--------------------+---------+-----------------
+----------------------+--------------------+------------+--------------------
```

```
-------+--------------------+------------------------+--------------------+--
--------------+------------------+------------------------+------------------
---+------------------------+--------------------------------+---------------
--------------------+--------------------------------+----------------------
-------+----------------------------------------+------------------------------
--------------+
|user_id|prediction|reordered|        product_name|order_dow|order_hour_of_day
|days_since_prior_order|               aisle|  department|product_name_string_
encoded|aisle_string_encoded|department_string_encoded|product_name_one_hot|
aisle_one_hot|department_one_hot|reordered_cast_to_double|user_id_cast_to_doub
le|order_dow_cast_to_double|order_hour_of_day_cast_to_double|days_since_prior_
order_cast_to_double|reordered_cast_to_doubleimputed|order_dow_cast_to_doublei
mputed|order_hour_of_day_cast_to_doubleimputed|days_since_prior_order_cast_to_
doubleimputed|
+-------+----------+---------+--------------------+---------+-----------------
+----------------------+--------------------+------------+--------------------
-------+--------------------+------------------------+--------------------+--
--------------+------------------+------------------------+------------------
```

```
para_df_join.printSchema()
```

```
root
 |-- user_id: long (nullable = true)
 |-- prediction: long (nullable = true)
 |-- reordered: integer (nullable = true)
 |-- product_name: string (nullable = false)
 |-- order_dow: integer (nullable = true)
 |-- order_hour_of_day: integer (nullable = true)
 |-- days_since_prior_order: integer (nullable = true)
 |-- aisle: string (nullable = false)
 |-- department: string (nullable = false)
 |-- product_name_string_encoded: double (nullable = false)
 |-- aisle_string_encoded: double (nullable = false)
 |-- department_string_encoded: double (nullable = false)
 |-- product_name_one_hot: vector (nullable = true)
 |-- aisle_one_hot: vector (nullable = true)
 |-- department_one_hot: vector (nullable = true)
 |-- reordered_cast_to_double: double (nullable = true)
 |-- user_id_cast_to_double: double (nullable = true)
 |-- order_dow_cast_to_double: double (nullable = true)
 |-- order_hour_of_day_cast_to_double: double (nullable = true)
 |-- days_since_prior_order_cast_to_double: double (nullable = true)
```

```
print((para_df_join.count(), len(para_df_join.columns)))
```

```
(19589, 24)
```

```
para_df_join_Fin = para_df_join.drop_duplicates()
```

```
print((para_df_join_Fin.count(), len(para_df_join_Fin.columns)))
```

```
(17309, 24)
```

```
PD_para_df_join_Fin = para_df_join_Fin.toPandas()
```

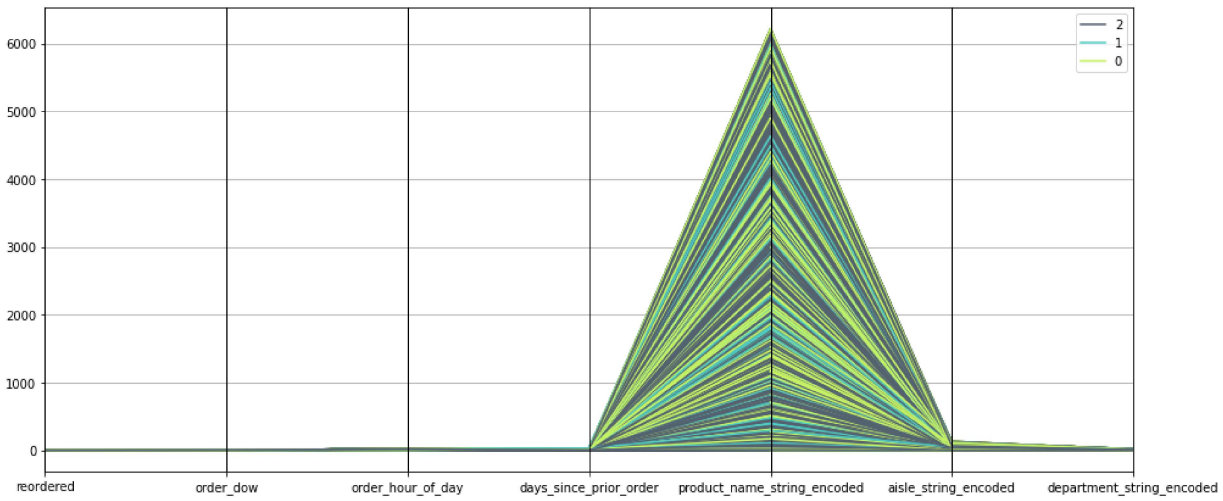
```
/databricks/spark/python/pyspark/sql/pandas/conversion.py:92: UserWarning: toPa
ndas attempted Arrow optimization because 'spark.sql.execution.arrow.pyspark.en
abled' is set to true; however, failed by the reason below:
  Unable to convert the field product_name_one_hot. If this column is not neces
sary, you may consider dropping it or converting to primitive type before the c
onversion.
Direct cause: Unsupported type in conversion to Arrow: VectorUDT
Attempting non-optimization as 'spark.sql.execution.arrow.pyspark.fallback.enab
led' is set to true.
  warnings.warn(msg)
```

```
plot_df =
PD_para_df_join_Fin[['prediction','reordered','order_dow','order_hour_of_day','
days_since_prior_order','product_name_string_encoded','aisle_string_encoded','d
epartment_string_encoded']]
```

```
plot_df.head()
```

	prediction	reordered	order_dow	order_hour_of_day	days_since_prior_order	product_name_str
0	2	1	3	13	3.0	
1	1	1	3	7	30.0	
2	2	1	1	15	0.0	
3	1	1	1	17	28.0	
4	0	1	2	0	18.0	

```
Out[79]: <AxesSubplot:>
```


2
1
0
6000
5000
4000
3000
2000
1000
0
reordered
order_dow
order_hour_of_day
days_since_prior_order
product_name_string_encoded
aisle_string_encoded
department_string_encoded