# Projet Big Data

Audric Girondin

## Introduciton

Dans un contexte économique de plus en plus compétitif, la capacité d'une entreprise à analyser et exploiter efficacement ses données de vente est cruciale pour maintenir sa position sur le marché. Ce projet s'inscrit dans une démarche d'exploration des ventes d'une entreprise spécialisée dans l'électronique, en utilisant un ensemble de données provenant de Kaggle. L'objectif principal est d'identifier les produits les plus performants, d'analyser les tendances de vente, et de comprendre les facteurs influençant les revenus.

Ce travail est réalisé dans le cadre de mon cours de Big Data, où j'apprends à gérer et analyser des données massives. L'utilisation de PySpark dans ce projet me permet de manipuler efficacement de grands volumes de données, ce qui est essentiel dans le traitement des données à grande échelle. Le projet comprend plusieurs étapes : nettoyage des données, analyse descriptive, tests statistiques et du machine learning. En particulier, l'analyse ANOVA a été utilisée pour évaluer les différences de prix moyens entre différents produits, tandis que le clustering via KMEANS a permis de regrouper certains produits. Sans oublier le traitement des données en flux avec PySpark Streaming, qui permet une analyse en temps réel des données.

## Configuration

```
#install pyspark
#!pip install pyspark
# Spark SQL
#!pip install pyspark[sql]
```

```
#import google drive
#from google.colab import drive
#drive.mount('/content/drive/')
```

```
# Import SparkSession
from pyspark.sql import SparkSession
# Create a Spark Session
spark = SparkSession.builder.master("local[*]").getOrCreate()
# Check Spark Session Information
spark
```

```
<pyspark.sql.session.SparkSession at 0x127482a00>
```

### Importation des librairies

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import udf, col, sum, count, to_date, month, year, split, mean,exp
from pyspark.sql.types import StringType, IntegerType
from pyspark.sql import Row
import seaborn as sns
import matplotlib.pyplot as plt
from pyspark.sql import functions as F
from datetime import datetime
from pyspark.ml.stat import Correlation
from pyspark.ml.feature import VectorAssembler
from scipy.stats import f
import numpy as np
import plotly.express as px
import plotly.graph_objs as go
from pyspark.ml.feature import StandardScaler
from pyspark.ml.clustering import KMeans
from pyspark.ml import Pipeline
```

### Chargement des données

```
# lecture du fichier
data = spark.read\
    .option("delimiter", ",")\
    .option("header", "true")\
    .option("inferSchema", "true")\
    .csv('New_Data.csv')
```

```
data.printSchema()
```

```
root
 |-- Order ID: integer (nullable = true)
 |-- Product: string (nullable = true)
 |-- Quantity Ordered: integer (nullable = true)
 |-- Price Each: double (nullable = true)
 |-- Order Date: string (nullable = true)
 |-- Purchase Address: string (nullable = true)
```

```
data.show(5)
```

```
+--------+--------------------+----------------+----------+--------------+--------------------
|Order ID|             Product|Quantity Ordered|Price Each|    Order Date|    Purchase Addres
+--------+--------------------+----------------+----------+--------------+--------------------
|  141234|              iPhone|               1|     700.0|01/22/19 21:25|944 Walnut St, Bo.
|  141235|Lightning Chargin...|               1|     14.95|01/28/19 14:15|185 Maple St, Por.
|  141236|    Wired Headphones|               2|     11.99|01/17/19 13:33|538 Adams St, San.
|  141237|    27in FHD Monitor|               1|    149.99|01/05/19 20:33|738 10th St, Los .
|  141238|    Wired Headphones|               1|     11.99|01/25/19 11:59|387 10th St, Aust.
+--------+--------------------+----------------+----------+--------------+--------------------
only showing top 5 rows
```

```
data_rdd = data.rdd
data_rdd.take(5)
```

```
[Row(Order ID=141234, Product='iPhone', Quantity Ordered=1, Price Each=700.0, Order Date='01/
 Row(Order ID=141235, Product='Lightning Charging Cable', Quantity Ordered=1, Price Each=14.9
 Row(Order ID=141236, Product='Wired Headphones', Quantity Ordered=2, Price Each=11.99, Order
 Row(Order ID=141237, Product='27in FHD Monitor', Quantity Ordered=1, Price Each=149.99, Orde
 Row(Order ID=141238, Product='Wired Headphones', Quantity Ordered=1, Price Each=11.99, Order
```

```
data.describe().show()
```

```
+-------+-----------------+------------+------------------+------------------+--------------+-
|summary|         Order ID|     Product|  Quantity Ordered|       Price Each|    Order Date|
+-------+-----------------+------------+------------------+------------------+--------------+-
|  count|           185950|      186304|            185950|            185950|        186304|
|   mean|230417.5693788653|        NULL|1.1243828986286637|184.39973476749391|          NULL|
```

```
| stddev|51512.73710999622|        NULL|0.4427926240286705|332.7313298843429|          NULL|
|    min|           141234|20in Monitor|                 1|             2.99|01/01/19 03:07|
|    max|           319670|      iPhone|                 9|           1700.0|    Order Date|
+-------+-----------------+------------+------------------+-----------------+--------------+
```

### Nettoyage des données

En appliquant ces opérations de nettoyage, nous nous assurons que les données utilisées sont pertinentes et précises, ce qui renforce la fiabilité des résultats de nos futures analyses.

```
# dropna
data = data.dropna()

def clean_product_name(product_name):
    return product_name.strip().upper()

clean_product_name_udf = udf(clean_product_name, StringType())

# Application de l'UDF pour nettoyer les noms de produits
data = data.withColumn("Product", clean_product_name_udf(data["Product"]))

# Filtrer les transactions improbables
data = data.filter((data['`Quantity Ordered`'] > 0) & (data['`Price Each`'] > 0))

data_rdd = data.rdd

data.show(5)
```

```
+--------+--------------------+---------------+----------+--------------+--------------------
|Order ID|             Product|Quantity Ordered|Price Each|    Order Date|    Purchase Addres
+--------+--------------------+---------------+----------+--------------+--------------------
|  141234|              IPHONE|              1|     700.0|01/22/19 21:25|944 Walnut St, Bo.
|  141235|LIGHTNING CHARGIN...|              1|     14.95|01/28/19 14:15|185 Maple St, Por.
|  141236|    WIRED HEADPHONES|              2|     11.99|01/17/19 13:33|538 Adams St, San.
|  141237|    27IN FHD MONITOR|              1|    149.99|01/05/19 20:33|738 10th St, Los .
|  141238|    WIRED HEADPHONES|              1|     11.99|01/25/19 11:59|387 10th St, Aust.
+--------+--------------------+---------------+----------+--------------+--------------------
only showing top 5 rows
```

## Analyse Descriptive

### Les 10 produits les plus vendus

Dans un premier temps, l'objectif est de déterminer les produits les plus vendus en regroupant les données par produit et en calculant la somme des quantités commandées pour chacun d'eux. En classant ces produits par ordre décroissant de quantités vendues, nous identifions les dix produits les plus populaires parmi les clients.

Cette analyse est essentielle pour comprendre les préférences des consommateurs et identifier les articles qui génèrent le plus de volume de ventes. En sachant quels produits sont les plus demandés, l'entreprise peut optimiser ses stratégies de stock, ajuster ses campagnes publicitaires pour mettre en avant ces produits, et mieux prévoir les besoins futurs en fonction des tendances de vente.

Les 10 produits les plus vendus sont (sous forme de df) :

```
# Grouper par produit et la somme des quantités
top_selling_products = data.groupBy("Product").agg(sum("`Quantity Ordered`").alias("TotalQua

# print des 10 produits les plus vendus

top_selling_products.orderBy("TotalQuantity", ascending=False).show(10)
```

```
+--------------------+-------------+
|             Product|TotalQuantity|
+--------------------+-------------+
|AAA BATTERIES (4-...|        31017|
|AA BATTERIES (4-P...|        27635|
|USB-C CHARGING CABLE|        23975|
|LIGHTNING CHARGIN...|        23217|
|    WIRED HEADPHONES|        20557|
|APPLE AIRPODS HEA...|        15661|
|BOSE SOUNDSPORT H...|        13457|
|    27IN FHD MONITOR|         7550|
|              IPHONE|         6849|
|27IN 4K GAMING MO...|         6244|
+--------------------+-------------+
only showing top 10 rows
```

Les 10 produits les plus vendus sont (sous forme de rdd) :

```
# Transformation de l'RDD pour extraire les colonnes nécessaires
product_quantity_rdd = data_rdd.map(lambda row: (row['Product'], row['Quantity Ordered']))

# Agréger les quantités par produit, en gérant les valeurs None
# Si x ou y est None, on le remplace par 0 avant l'addition
total_quantity_per_product_rdd = product_quantity_rdd.reduceByKey(lambda x, y: (x if x is not

# Convertir en un RDD de Row pour un affichage facile
top_selling_products_rdd = total_quantity_per_product_rdd.map(lambda x: Row(Product=x[0], Tot

# Trier les produits par quantité totale commandée en ordre décroissant
sorted_top_selling_products_rdd = top_selling_products_rdd.sortBy(lambda x: x['TotalQuantity

sorted_top_selling_products_rdd.take(10)
```

```
[Row(Product='AAA BATTERIES (4-PACK)', TotalQuantity=31017),
 Row(Product='AA BATTERIES (4-PACK)', TotalQuantity=27635),
 Row(Product='USB-C CHARGING CABLE', TotalQuantity=23975),
 Row(Product='LIGHTNING CHARGING CABLE', TotalQuantity=23217),
 Row(Product='WIRED HEADPHONES', TotalQuantity=20557),
 Row(Product='APPLE AIRPODS HEADPHONES', TotalQuantity=15661),
 Row(Product='BOSE SOUNDSPORT HEADPHONES', TotalQuantity=13457),
 Row(Product='27IN FHD MONITOR', TotalQuantity=7550),
 Row(Product='IPHONE', TotalQuantity=6849),
 Row(Product='27IN 4K GAMING MONITOR', TotalQuantity=6244)]
```

### Les 10 produits les plus rentables

Ici l'objectif est de calculer le revenu total généré par chaque produit en multipliant la quantité commandée par le prix unitaire pour chaque transaction. En regroupant ensuite les données par produit, nous sommes en mesure de déterminer quels sont les produits les plus rentables en termes de revenus générés.

Cette analyse est particulièrement utile pour identifier les produits phares de l'entreprise, c'est-à-dire ceux qui contribuent le plus au chiffre d'affaires. En comprenant quels produits sont les plus rentables, l'entreprise pourra donc optimiser son inventaire, concentrer ses efforts de marketing sur ces produits, et mieux allouer ses ressources pour maximiser les profits. De plus, cette information peut être utilisée pour faire des projections financières et ajuster les stratégies de vente en fonction des performances réelles des produits.

```
#Calculate total revenue for each product
data = data.withColumn("Revenue", col("Quantity Ordered") * col("Price Each"))

# Grouper par  produit et  calcul du revenue total
revenue_by_product = data.groupBy("Product").agg(sum("Revenue").alias("TotalRevenue"))

print("Les 10 produits les plus rentables sont :")
revenue_by_product.orderBy("TotalRevenue", ascending=False).show(10)
```

```
Les 10 produits les plus rentables sont :
+--------------------+------------------+
|             Product|      TotalRevenue|
+--------------------+------------------+
|  MACBOOK PRO LAPTOP|         8037600.0|
|              IPHONE|         4794300.0|
|     THINKPAD LAPTOP|4129958.6999999676|
|        GOOGLE PHONE|         3319200.0|
|27IN 4K GAMING MO...|2435097.5599999577|
|34IN ULTRAWIDE MO...|2355558.0099999583|
|APPLE AIRPODS HEA...|         2349150.0|
|       FLATSCREEN TV|         1445700.0|
|BOSE SOUNDSPORT H...|1345565.4299999368|
|    27IN FHD MONITOR|1132424.4999999707|
+--------------------+------------------+
only showing top 10 rows
```

### Tendance des ventes mensuelles

En ce qui concerne l'analyse suivante, l'objectif est d'examiner la tendance des ventes mensuelles de l'entreprise en visualisant l'évolution des revenus au fil du temps.

Elle permet de révéler des insights sur le comportement d'achat des clients, aider à identifier les mois les plus rentables, et permettre à l'entreprise de mieux planifier ses stratégies marketing et ses opérations commerciales en fonction des tendances observées. Voir Figure 1.

```
# Définir la fonction pour extraire le mois
def extract_month(order_date):
    date_obj = datetime.strptime(order_date, '%m/%d/%y %H:%M')
    return date_obj.month

# Définir la fonction pour extraire l'année
```

```
def extract_year(order_date):
    date_obj = datetime.strptime(order_date, '%m/%d/%y %H:%M')
    return date_obj.year

# Enregistrer les fonctions comme UDFs
extract_month_udf = udf(extract_month, IntegerType())
extract_year_udf = udf(extract_year, IntegerType())

# Appliquer les UDFs pour créer les colonnes Month et Year
data = data.withColumn("Month", extract_month_udf(data["Order Date"]))
data = data.withColumn("Year", extract_year_udf(data["Order Date"]))
```

```
# Grouper par mois et année pour voirs les revenues du mois
monthly_sales = data.groupBy("Month", "Year").agg(sum("Revenue").alias("TotalRevenue"))

# Show sales trends
monthly_sales.orderBy("Year", "Month").show()
```

```
+-----+----+------------------+
|Month|Year|      TotalRevenue|
+-----+----+------------------+
|    1|2019|1813586.4399999138|
|    2|2019|2202022.4199999087|
|    3|2019|2807100.3800003603|
|    4|2019|3390670.2400007015|
|    5|2019|3152606.7500005495|
|    6|2019|2577802.2600001753|
|    7|2019|2647775.7600002354|
|    8|2019| 2244467.879999913|
|    9|2019| 2097560.129999891|
|   10|2019|3736726.8800009675|
|   11|2019|3199603.2000005865|
|   12|2019| 4613443.340000139|
|    1|2020| 8670.289999999999|
+-----+----+------------------+
```

```
# Convertir en DataFrame Pandas
monthly_sales_pd = monthly_sales.toPandas()

# Combiner l'année et le mois en une seule colonne pour une meilleure visualisation
monthly_sales_pd['YearMonth'] = monthly_sales_pd['Year'].astype(str) + '-' + monthly_sales_pd
```

```
# Configuration du style Seaborn
sns.set(style="whitegrid")

# Création du graphique de tendance des ventes mensuelles
plt.figure(figsize=(12, 6))
sns.lineplot(
    data=monthly_sales_pd,
    x='YearMonth',
    y='TotalRevenue',
    marker='o',
    color='r'
)

# Configuration des axes et des labels
plt.title('Tendance des ventes mensuelles')
plt.xlabel('Year-Month')
plt.ylabel('Revenu total')
plt.xticks(rotation=45)
plt.grid(True)

# Affichage du graphique
plt.show()
```

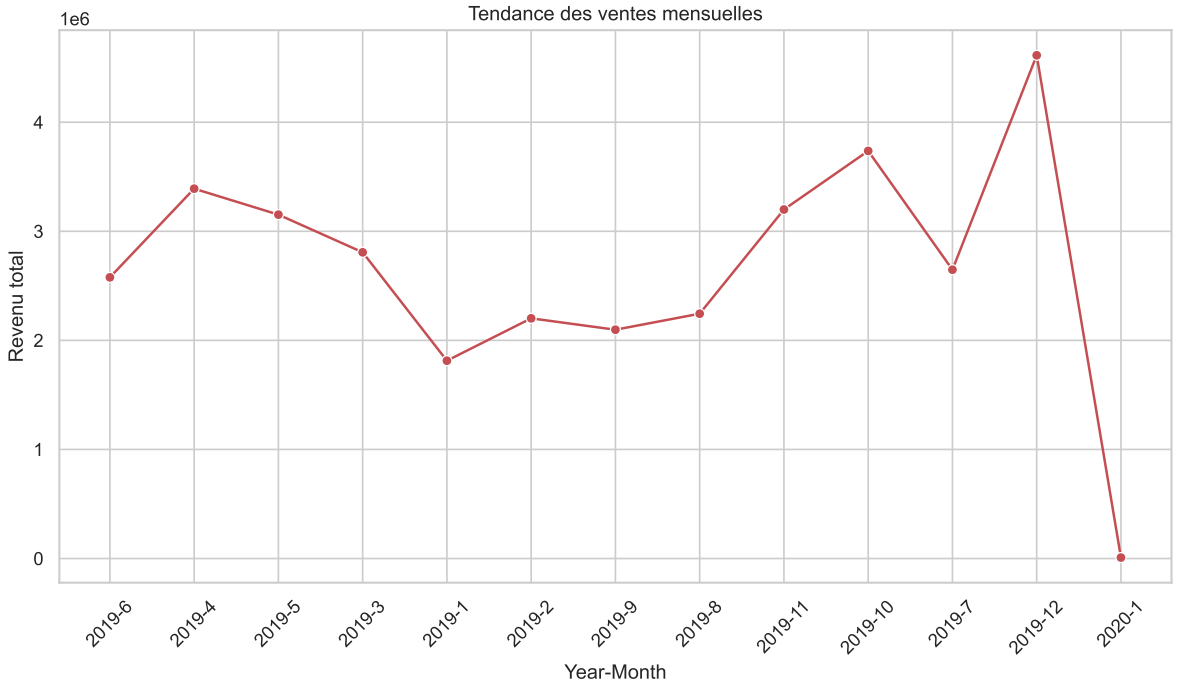


Figure 1: lineplot sur l'évolutions des revenus par mois

### Chiffre d'affaires des 5 principaux produits dans différentes villes

Nous cherchons à identifier les cinq produits les plus rentables, puis d'analyser leur performance en termes de revenus dans différentes villes.

Cette analyse permet de visualiser comment les produits les plus rentables se comportent dans différentes localités, ce qui peut révéler des tendances géographiques dans les préférences des clients. Une telle analyse est cruciale pour adapter les stratégies de vente et de marketing en fonction des régions, optimiser les opérations logistiques, et cibler les efforts commerciaux là où ils sont les plus efficaces. En visualisant ces données à l'aide d'un graphique en barres, nous obtenons un aperçu clair des performances des principaux produits dans les différentes villes. Voir Figure 2.

```
# Obtenir les 5 produits les plus rentables
top_5_products = revenue_by_product.orderBy("TotalRevenue", ascending=False).limit(5)

# Récupérer les noms des 5 produits les plus rentables
top_5_product_names = [row['Product'] for row in top_5_products.collect()]

# fonction pour extraire la ville à partir de l'adresse
```

```
def extract_city(purchase_address):
    try:
        return purchase_address.split(",")[1].strip()
    except IndexError:
        return None  # Au cas où le format ne correspondrait pas

# fonction comme UDF
extract_city_udf = udf(extract_city, StringType())

data = data.withColumn("City", extract_city_udf(data["Purchase Address"]))

# Filtrer les données pour ne conserver que les 5 produits les plus rentables
filtered_data = data.filter(data['Product'].isin(top_5_product_names))

# Regrouper par produit et ville pour obtenir le revenu par produit et par ville
product_city_performance = filtered_data.groupBy("City", "Product").agg(
    sum("Revenue").alias("TotalRevenue")
)

# Convertir en Pandas pour une analyse plus facile
product_city_performance_pd = product_city_performance.orderBy("City", "TotalRevenue", ascend

# Création du graphique avec Seaborn
plt.figure(figsize=(14, 8))
sns.set(style="whitegrid")

# Barplot avec Seaborn
sns.barplot(
    data=product_city_performance_pd,
    x="City",
    y="TotalRevenue",
    hue="Product",
    palette="tab10"
)

# Configuration des axes et des labels
plt.xlabel('Villes')
plt.ylabel('Revenue Total')
plt.title("Chiffre d'affaires des 5 principaux produits dans différentes villes")
plt.xticks(rotation=45)
plt.legend(title='Produits')
plt.grid(True)
```

```
# Affichage du graphique
plt.show()
```

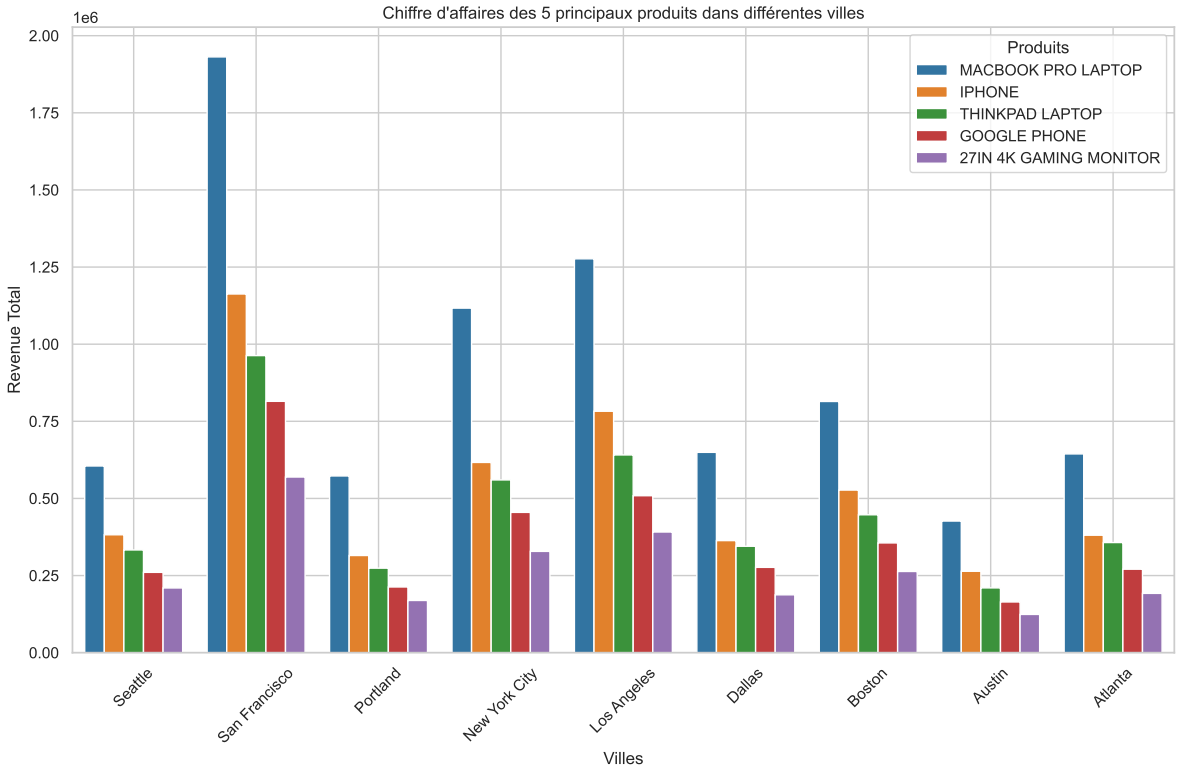


Figure 2: barplot sur les revenus des produits les plus rentables par villes

## Matrice de corrélation

Pour analyser la relation entre différentes variables numériques dans un jeu de données. En calculant la matrice de corrélation, on peut de déterminer la relation entre les colonnes sélectionnées, ce qui peut aider à identifier des tendances ou des dépendances significatives entre les variables telles que la quantité commandée et le prix unitaire.

```
def calculate_correlation_matrix(df, cols):
    """
    Calcule la matrice de corrélation pour les colonnes spécifiées provenant d'un dataframe

    Args:
    df : dataframe.
```

```
    cols (list of str): Liste des noms des colonnes pour lesquelles la corrélation doit être

    Returns:
    correlation_matrix: Matrice de corrélation.
    """

    # Sélection des colonnes pertinentes pour la corrélation
    assembler = VectorAssembler(inputCols=cols, outputCol="features")
    vector_data = assembler.transform(df).select("features")

    # Calcul de la matrice de corrélation
    correlation_matrix = Correlation.corr(vector_data, "features").head()[0]

    return correlation_matrix

# Exemple d'utilisation
columns = ["Quantity Ordered", "Price Each"]
correlation_matrix = calculate_correlation_matrix(data, columns)
print("Matrice de corrélation :\n", correlation_matrix)
```

```
Matrice de corrélation :
 DenseMatrix([[ 1.        , -0.14827234],
              [-0.14827234,  1.        ]])
```

### Tests statistiques

*a remplir*

### Machine Learning

#### Clustering

Ce code réalise une analyse des données de vente d'une entreprise en utilisant PySpark. L'objectif est de regrouper les produits en clusters en fonction de la quantité totale vendue et du prix unitaire moyen, afin d'identifier des tendances et des segments de produits similaires. Voir Figure 3.

```
# Conversion des colonnes en types numériques
df = data.withColumn("Quantity Ordered", col("Quantity Ordered").cast("float"))
df = data.withColumn("Price Each", col("Price Each").cast("float"))

# Agrégation des données par produit
product_group_df = df.groupBy("Product").agg(
    sum("Quantity Ordered").alias("Total Quantity Ordered"),
    mean("Price Each").alias("Price Each")
)

# Assembler les features en un vecteur
assembler = VectorAssembler(
    inputCols=["Total Quantity Ordered", "Price Each"],
    outputCol="features"
)

# Normalisation des données
scaler = StandardScaler(inputCol="features", outputCol="scaled_features", withStd=True, withM

# Appliquer KMeans
kmeans = KMeans(featuresCol='scaled_features', k=3, seed=42)

# Définir les étapes du pipeline
pipeline = Pipeline(stages=[assembler, scaler, kmeans])

#Entrainement du model
model = pipeline.fit(product_group_df)

#application du modele
clusters_df = model.transform(product_group_df)

# Renommer la colonne 'prediction' en 'cluster'
clusters_df = clusters_df.withColumnRenamed('prediction', 'cluster')

# Afficher les résultats
clusters_df.select("Product", "Total Quantity Ordered", "Price Each", "cluster").show()
```

```
+--------------------+----------------------+------------------+-------+
|             Product|Total Quantity Ordered|        Price Each|cluster|
+--------------------+----------------------+------------------+-------+
|27IN 4K GAMING MO...|                  6244|  389.989990234375|      1|
|    WIRED HEADPHONES|                 20557|11.989999771118164|      0|
```

```
|            LG DRYER|                  646|             600.0|      1|
|USB-C CHARGING CABLE|                23975|11.949999809265137|      0|
|AAA BATTERIES (4-...|                31017| 2.990000009536743|      0|
|  LG WASHING MACHINE|                  666|             600.0|      1|
|       FLATSCREEN TV|                 4819|             300.0|      1|
|LIGHTNING CHARGIN...|                23217|14.949999809265137|      0|
|              IPHONE|                 6849|             700.0|      1|
|34IN ULTRAWIDE MO...|                 6199|  379.989990234375|      1|
|  MACBOOK PRO LAPTOP|                 4728|            1700.0|      2|
|        GOOGLE PHONE|                 5532|             600.0|      1|
|BOSE SOUNDSPORT H...|                13457| 99.98999786376953|      0|
|APPLE AIRPODS HEA...|                15661|             150.0|      0|
|     THINKPAD LAPTOP|                 4130|  999.989990234375|      1|
|     VAREEBADD PHONE|                 2068|             400.0|      1|
|AA BATTERIES (4-P...|                27635|3.8399999141693115|      0|
|    27IN FHD MONITOR|                 7550|149.99000549316406|      1|
|        20IN MONITOR|                 4129|109.98999786376953|      1|
+--------------------+---------------------+------------------+-------+
```

```python
result_cluster_pd = clusters_df.toPandas()

# Créer le scatter plot interactif
fig = px.scatter(
    result_cluster_pd,
    x="Total Quantity Ordered",
    y="Price Each",
    color="cluster",
    hover_data=["Product"],  # Affiche le nom du produit lors du survol
    title="Clustering des Produits",
    color_continuous_scale=px.colors.sequential.Bluered

)

# Afficher la figure
fig.show()
```

### Flux de données en temps réel

Ce code est utile pour effectuer une analyse en temps réel des données reçues via une connexion socket. En utilisant des fenêtres temporelles, il permet d'agréger et de compter les occurrences

```
Unable to display output for mime type(s): text/html
```

(a) scatterplot sur le clustering des produits

```
Unable to display output for mime type(s): text/html
```

(b)

Figure 3

des mots sur des intervalles de 1 minute. Les résultats sont ensuite sauvegardés dans une table Spark pour une analyse ultérieure.

Adaptée à notre contexte, cette approche permet une réaction en temps réel aux tendances du marché en traitant les données de vente dès leur génération.

### Serveur Socket

```python
import socket
import time

# Configuration du serveur socket
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.bind(('localhost', 9999))  # Hôte et port
server_socket.listen(1)  # Permettre une seule connexion client

print("Serveur socket en attente de connexion...")
conn, addr = server_socket.accept()  # Accepter la connexion

print(f"Connexion établie avec {addr}")

# Envoyer des données à intervalles réguliers
phrases = ['hello world', 'this is a test', 'spark streaming with socke', 'structured stream
while True:
    for text in phrases:
        conn.send((text + "\n").encode())  # Envoyer un message avec un saut de ligne
        print(f"Envoyé : {text}")
        time.sleep(2)  # Pause de 2 secondes avant d'envoyer le prochain message

conn.close()
```