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](https://www.kaggle.com/datasets/darkovichcycy/bd-sales?resource=download&select=Sales_August_2019.csv) 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 (ANOVA), et du machine learning. En particulier, l’analyse ANOVA a été utilisée pour évaluer les différences de prix moyens entre différents produits, tandis que la régression linéaire a permis de prédire les revenus en fonction de certaines caractéristiques des transactions.

## 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 0x15d9beac0>

## 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 Address|  
+--------+--------------------+----------------+----------+--------------+--------------------+  
| 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='01/22/19 21:25', Purchase Address='944 Walnut St, Boston, MA 02215'),  
 Row(Order ID=141235, Product='Lightning Charging Cable', Quantity Ordered=1, Price Each=14.95, Order Date='01/28/19 14:15', Purchase Address='185 Maple St, Portland, OR 97035'),  
 Row(Order ID=141236, Product='Wired Headphones', Quantity Ordered=2, Price Each=11.99, Order Date='01/17/19 13:33', Purchase Address='538 Adams St, San Francisco, CA 94016'),  
 Row(Order ID=141237, Product='27in FHD Monitor', Quantity Ordered=1, Price Each=149.99, Order Date='01/05/19 20:33', Purchase Address='738 10th St, Los Angeles, CA 90001'),  
 Row(Order ID=141238, Product='Wired Headphones', Quantity Ordered=1, Price Each=11.99, Order Date='01/25/19 11:59', Purchase Address='387 10th St, Austin, TX 73301')]

data.describe().show()

+-------+-----------------+------------+------------------+-----------------+--------------+--------------------+  
|summary| Order ID| Product| Quantity Ordered| Price Each| Order Date| Purchase Address|  
+-------+-----------------+------------+------------------+-----------------+--------------+--------------------+  
| count| 185950| 186305| 185950| 185950| 186305| 186305|  
| mean|230417.5693788653| NULL|1.1243828986286637|184.3997347674939| NULL| NULL|  
| stddev|51512.73710999622| NULL|0.4427926240286707|332.7313298843435| NULL| NULL|  
| min| 141234|20in Monitor| 1| 2.99|01/01/19 03:07|1 11th St, Atlant...|  
| max| 319670| iPhone| 9| 1700.0| Order Date| Purchase Address|  
+-------+-----------------+------------+------------------+-----------------+--------------+--------------------+

## 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.

from pyspark.sql.functions import udf  
from pyspark.sql.types import StringType  
  
# 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 Address|  
+--------+--------------------+----------------+----------+--------------+--------------------+  
| 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) :

from pyspark.sql.functions import col, sum, count, to\_date, month, year, split  
import seaborn as sns  
import matplotlib.pyplot as plt  
  
# Grouper par produit et la somme des quantités  
top\_selling\_products = data.groupBy("Product").agg(sum("`Quantity Ordered`").alias("TotalQuantity"))  
  
# 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) :

from pyspark.sql import Row  
  
# 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 None else 0) + (y if y is not None else 0))  
  
# 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], TotalQuantity=x[1]))  
  
# Trier les produits par quantité totale commandée en ordre décroissant  
sorted\_top\_selling\_products\_rdd = top\_selling\_products\_rdd.sortBy(lambda x: x['TotalQuantity'], ascending=False)  
  
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](#fig-lineplot).

from pyspark.sql.functions import udf  
from pyspark.sql.types import IntegerType  
from datetime import datetime  
  
# 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['Month'].astype(str)  
  
# 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](#fig-barplot).

# 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", ascending=False).toPandas()  
  
# 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 |

## Correlation

from pyspark.ml.stat import Correlation  
from pyspark.ml.feature import VectorAssembler  
from pyspark.sql import SparkSession  
  
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 calculée.  
  
 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. ]])

## Test de l’Anova

Dans cette analyse, l’objectif est de réaliser un test ANOVA (Analyse de la Variance) pour déterminer s’il existe des différences significatives entre les revenus moyens générés par différents produits.

Le test ANOVA permet de comparer les moyennes des revenus des produits pour déterminer si les variations observées sont statistiquement significatives ou simplement dues au hasard. Cette analyse est essentielle pour identifier si certains produits se distinguent en termes de performance financière, ce qui peut orienter les stratégies de l’entreprise en matière de production, de marketing, et de gestion des ressources. En analysant les résultats du test ANOVA, nous pouvons rejeter ou accepter l’hypothèse nulle selon laquelle il n’y a pas de différence significative entre les groupes de produits. Cette information est cruciale pour prendre des décisions éclairées sur les produits à privilégier ou à repenser.

from pyspark.sql import functions as F  
from scipy.stats import f  
  
def anova\_test(df, group\_col, value\_col):  
 """  
 Effectue un test ANOVA sur un DataFrame PySpark.  
  
 Paramètres:  
 df : DataFrame PySpark  
 Le DataFrame contenant les données à analyser.  
 group\_col : str  
 Le nom de la colonne représentant les groupes/facteurs.  
 value\_col : str  
 Le nom de la colonne contenant les valeurs numériques à analyser.  
  
 Retourne:  
 dict :  
 Un dictionnaire contenant les statistiques SSB, SSW, MSB, MSW, la statistique F et la p-value.  
 SBB : somme des carrés inter-groupes  
 SSW : somme des carrés intra-groupes  
 MSB : les moyennes des carrés inter-groupes  
 MSW : les moyennes des carrés intra-groupes  
 La statistique F  
 La p-value.  
 """  
  
 # Calcul de la moyenne générale  
 mean\_total = df.agg(F.mean(value\_col)).first()[0]  
  
 # Calcul de la moyenne par groupe  
 df\_group\_mean = df.groupBy(group\_col).agg(F.mean(value\_col).alias('group\_mean'))  
  
 # Jointure des moyennes avec le DataFrame original  
 df\_with\_means = df.join(df\_group\_mean, on=group\_col)  
  
 # Calcul de la SSB (Sum of Squares Between)  
 df\_ssb = df\_with\_means.withColumn('ssb', (F.col('group\_mean') - mean\_total) \*\* 2)  
 ssb = df\_ssb.groupBy(group\_col).agg(F.sum('ssb').alias('ssb\_contrib')).agg(F.sum('ssb\_contrib')).first()[0]  
  
 # Calcul de la SSW (Sum of Squares Within)  
 df\_ssw = df\_with\_means.withColumn('ssw', (F.col(value\_col) - F.col('group\_mean')) \*\* 2)  
 ssw = df\_ssw.agg(F.sum('ssw')).first()[0]  
  
 # Calcul des degrés de liberté  
 n\_groups = df.select(group\_col).distinct().count()  
 n\_total = df.count()  
  
 df\_between = n\_groups - 1  
 df\_within = n\_total - n\_groups  
  
 # Calcul de MSB et MSW  
 msb = ssb / df\_between  
 msw = ssw / df\_within  
  
 # Calcul de la statistique F  
 F\_stat = msb / msw  
  
 # Calcul de la p-value  
 p\_value = 1 - f.cdf(F\_stat, df\_between, df\_within)  
  
 return {  
 'Somme des carrés inter-groupes': ssb,  
 'Somme des carrés intra-groupes': ssw,  
 'Moyennes des carrés inter-groupes': msb,  
 'Moyennes des carrés intra-groupes': msw,  
 'F\_stat': F\_stat,  
 'p\_value': p\_value  
 }  
  
#anova\_test = udf(anova\_test)  
# Exemple d'utilisation  
if \_\_name\_\_ == "\_\_main\_\_":  
  
 # Appel de la fonction  
 results = anova\_test(data, 'Product', 'Revenue')  
  
 # Affichage des résultats  
 print("Résultats de l'ANOVA :")  
 for key, value in results.items():  
 print(f"{key}: {value}")  
  
 # Conclusion  
 alpha = 0.05  
 if results['p\_value'] < alpha:  
 print("On rejette l'hypothèse nulle : il y a une différence significative entre les groupes.")  
 else:  
 print("On ne rejette pas l'hypothèse nulle : il n'y a pas de différence significative entre les groupes.")

Résultats de l'ANOVA :  
Somme des carrés inter-groupes: 20576932023.333424  
Somme des carrés intra-groupes: 32832159.868263796  
Moyennes des carrés inter-groupes: 1143162890.1851902  
Moyennes des carrés intra-groupes: 176.58249494846905  
F\_stat: 6473817.750274693  
p\_value: 1.1102230246251565e-16  
On rejette l'hypothèse nulle : il y a une différence significative entre les groupes.

## Machine Learning

### Régression Linéaire

L’objectif est de construire un modèle de régression linéaire pour prédire les revenus en fonction de certaines caractéristiques des transactions, telles que la quantité commandée et le prix unitaire.

from pyspark.ml.regression import LinearRegression  
from pyspark.ml.feature import VectorAssembler  
from pyspark.ml.evaluation import RegressionEvaluator  
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder  
  
# Preparation des données  
assembler = VectorAssembler(inputCols=["Quantity Ordered", "Price Each"], outputCol="features")  
output = assembler.transform(data)  
  
# Split le dataset en training et test sets  
train\_data, test\_data = output.randomSplit([0.8, 0.2])

lr = LinearRegression(featuresCol="features", labelCol="Revenue")  
  
paramGrid = ParamGridBuilder() \  
 .addGrid(lr.regParam, [0.1, 0.1, 0.5]) \  
 .addGrid(lr.elasticNetParam, [0.0, 0.5, 1.0]) \  
 .build()  
  
evaluator\_cv\_mse = RegressionEvaluator(labelCol="Revenue", predictionCol="prediction", metricName="mse")  
# Initialize CrossValidator  
crossval = CrossValidator(estimator=lr,  
 evaluator=evaluator\_cv\_mse,  
 estimatorParamMaps=paramGrid,  
 numFolds=5) # 5-fold cross-validation  
  
cv\_model = crossval.fit(train\_data)  
  
# Get the best model  
best\_model = cv\_model.bestModel  
  
print(f"Coefficients: {best\_model.coefficients}, Intercept: {best\_model.intercept}")  
print(f"R²: {best\_model.summary.r2}")

[Stage 338:============================> (2 + 2) / 4] [Stage 367:==============> (1 + 3) / 4] [Stage 396:> (0 + 4) / 4]

Coefficients: [6.605622200913777,1.000582054844649], Intercept: -6.453359762678648  
R²: 0.9986889531930573

#### Prédictions

Nous utilisons le modèle pour faire des prédictions sur les données de test et comparons les prédictions aux valeurs réelles de revenus. Cette étape permet d’évaluer l’exactitude du modèle et de voir comment les caractéristiques sélectionnées influencent les revenus.

# predictions sur le test sets  
predictions\_cv = best\_model.transform(test\_data)  
  
# comparaison predictions et valeurs réelles  
predictions\_cv.select("features", "Revenue", "prediction").show()

+------------+-------+------------------+  
| features|Revenue| prediction|  
+------------+-------+------------------+  
| [1.0,2.99]| 2.99|3.1440027822206282|  
| [1.0,11.95]| 11.95|12.109217993628683|  
|[1.0,149.99]| 149.99|150.22956484438404|  
| [1.0,300.0]| 300.0| 300.3268788916298|  
| [1.0,2.99]| 2.99|3.1440027822206282|  
|[1.0,389.99]| 389.99|390.36925800709975|  
| [1.0,11.99]| 11.99|12.149241275822469|  
| [1.0,11.95]| 11.95|12.109217993628683|  
| [1.0,11.95]| 11.95|12.109217993628683|  
| [1.0,14.95]| 14.95| 15.11096415816263|  
| [1.0,11.95]| 11.95|12.109217993628683|  
| [3.0,2.99]| 8.97| 16.35524718404818|  
|[1.0,149.99]| 149.99|150.22956484438404|  
| [1.0,99.99]| 99.99|100.20046210215158|  
| [1.0,11.95]| 11.95|12.109217993628683|  
|[1.0,999.99]| 999.99|1000.7243114623355|  
| [1.0,11.95]| 11.95|12.109217993628683|  
|[1.0,149.99]| 149.99|150.22956484438404|  
| [1.0,2.99]| 2.99|3.1440027822206282|  
| [1.0,2.99]| 2.99|3.1440027822206282|  
+------------+-------+------------------+  
only showing top 20 rows

#### Metriques

Une bonne performance du modèle de régression peut aider l’entreprise à prévoir les revenus futurs en fonction des variations de quantités commandées et de prix, optimisant ainsi les stratégies de tarification et de gestion des stocks.

Cette performance du modèle de régression est évaluée à travers des métriques telles que le coefficient de détermination (R²), l’erreur quadratique moyenne (MSE) ou encore l’erreur absolue moyenne (MAE). Un modèle avec un R² élevé indique que le modèle explique une grande partie de la variance des revenus, tandis que des valeurs faibles de MSE et de MAE suggèrent que les prédictions du modèle sont proches des valeurs réelles. Ainsi, si ces métriques montrent une performance satisfaisante, l’entreprise peut utiliser le modèle pour prévoir les revenus futurs en fonction des variations de quantités commandées et de prix, optimisant ainsi les stratégies de tarification et de gestion des stocks.

#Controle des metriques  
cv\_mse = evaluator\_cv\_mse.evaluate(predictions\_cv)  
print(f"Mean Squared Error (MSE) on test data: {cv\_mse}")  
  
r2\_test\_cv = evaluator\_cv\_mse.evaluate(predictions\_cv, {evaluator\_cv\_mse.metricName: "r2"})  
print(f"R² on test data: {r2\_test\_cv}")

Mean Squared Error (MSE) on test data: 257.8502519823125  
R² on test data: 0.997662280729423

#### Métriques

import numpy as np  
import plotly.express as px  
import plotly.graph\_objs as go  
from pyspark.ml.linalg import DenseVector  
from scipy.interpolate import griddata  
  
# Convertir les résultats en Pandas pour la visualisation  
preds = predictions\_cv.select("features", "Revenue", "prediction").toPandas()  
  
# Extraire les valeurs des deux features  
preds['feature1'] = preds['features'].apply(lambda x: x[0] if isinstance(x, DenseVector) else x[0])  
preds['feature2'] = preds['features'].apply(lambda x: x[1] if isinstance(x, DenseVector) else x[1])  
  
# Créer un scatter plot 3D pour les données originales  
trace1 = go.Scatter3d(  
 x=preds['feature1'],  
 y=preds['feature2'],  
 z=preds['Revenue'],  
 mode='markers',  
 marker=dict(  
 size=5,  
 color='blue',  
 ),  
 name='Revenue'  
)  
  
# Créer une grille pour les axes x et y  
x\_grid, y\_grid = np.meshgrid(np.unique(preds['feature1']), np.unique(preds['feature2']))  
  
# Interpoler les valeurs de prédiction pour correspondre à la grille  
z\_grid = griddata((preds['feature1'], preds['feature2']), preds['prediction'], (x\_grid, y\_grid), method='linear')  
  
# Créer un surface plot pour la régression prédite  
trace2 = go.Surface(  
 x=x\_grid,  
 y=y\_grid,  
 z=z\_grid,  
 colorscale='Viridis',  
 name='Regression Surface',  
 opacity=0.7  
)  
  
# Combiner les graphiques  
data\_grid = [trace1, trace2]  
  
# Créer le layout  
layout = go.Layout(  
 title='Régression linéaire multiple avec Plotly',  
 scene=dict(  
 xaxis\_title='Quantity Ordered',  
 yaxis\_title='Price Each',  
 zaxis\_title='Revenue'  
 )  
)  
  
# Créer la figure  
fig = go.Figure(data=data\_grid, layout=layout)  
  
# Afficher le graphique  
fig.show()

|  |  |  |
| --- | --- | --- |
| |  | | --- | | Unable to display output for mime type(s): text/html  (a) scatterplot sur la régression linéaire en 3D |      |  | | --- | | Unable to display output for mime type(s): text/html  (b) |   Figure 3 |

### Clustering

from pyspark.sql.functions import mean  
from pyspark.ml.feature import StandardScaler  
from pyspark.ml.clustering import KMeans  
  
# 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"  
)  
product\_features\_df = assembler.transform(product\_group\_df)  
  
# Normalisation des données  
scaler = StandardScaler(inputCol="features", outputCol="scaled\_features", withStd=True, withMean=False)  
scaler\_model = scaler.fit(product\_features\_df)  
scaled\_product\_df = scaler\_model.transform(product\_features\_df)  
  
# Appliquer KMeans  
kmeans = KMeans(featuresCol='scaled\_features', k=3, seed=42)  
model = kmeans.fit(scaled\_product\_df)  
clusters\_df = model.transform(scaled\_product\_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|  
+--------------------+----------------------+------------------+-------+

import plotly.express as px  
  
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()

|  |
| --- |
| Unable to display output for mime type(s): text/html  Figure 4: scatterplot sur le clustering des produits |