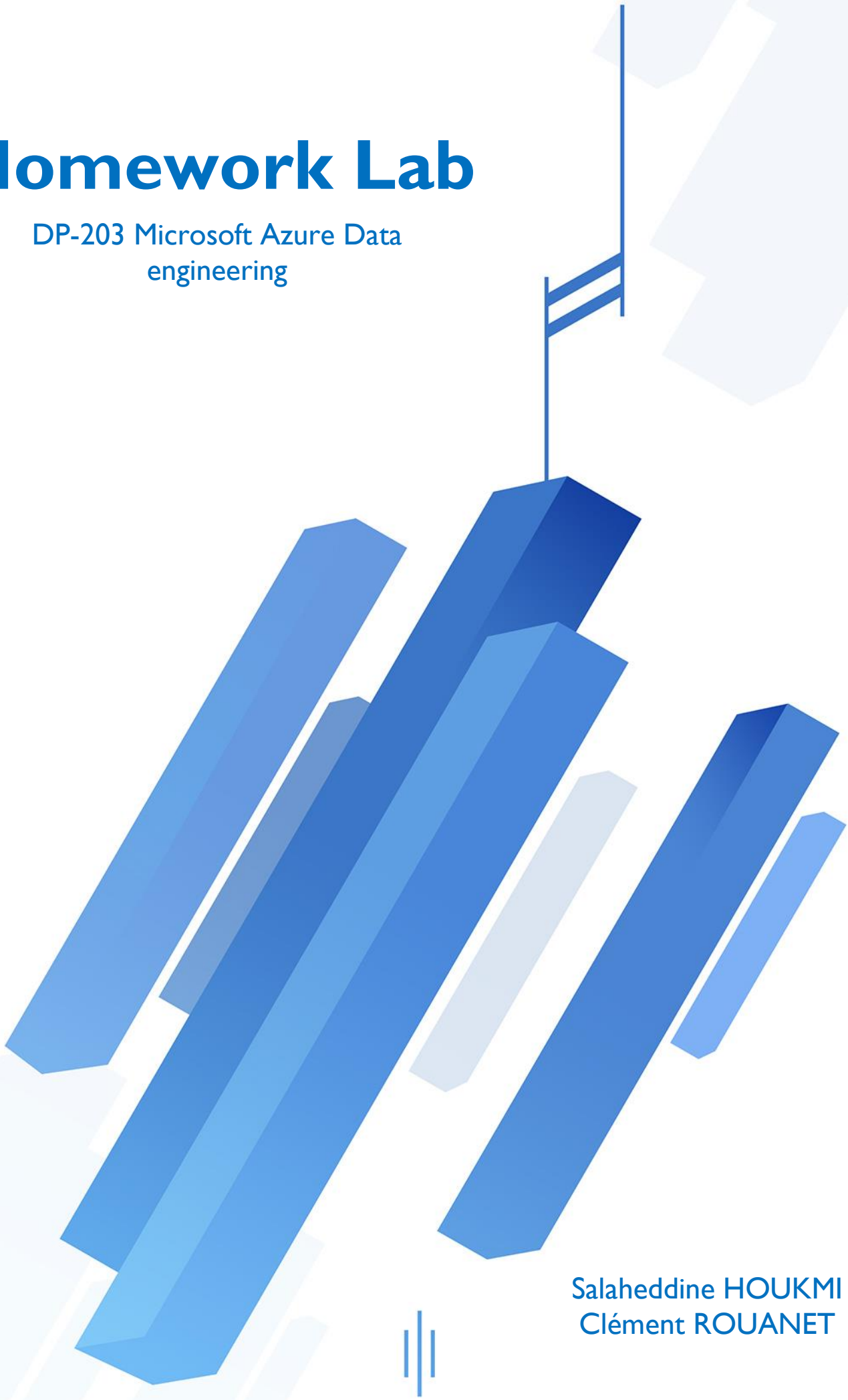
Homework Lab

DP-203 Microsoft Azure Data engineering



Salaheddine HOUKMI
Clément ROUANET

Sujet

Projet :

Notre initiative vise à implémenter et optimiser des solutions de stockage et de traitement de données sur le cloud, axées sur un objectif principal : **Trouver le "produit gagnant" et les emplacements stratégiques pour les publicités à partir du dataset de la plateforme de e-commerce brésilienne Olist.**

Pour démarrer un business en ligne, surtout dans le e-commerce, on commence par chercher une niche rentable en essayant de vendre plusieurs produits et en observant les résultats de chacun. Une fois le produit gagnant identifié, on lance nos publicités dans une région bien déterminée, en procédant également par essais. Toutes ces démarches nécessitent un grand budget.

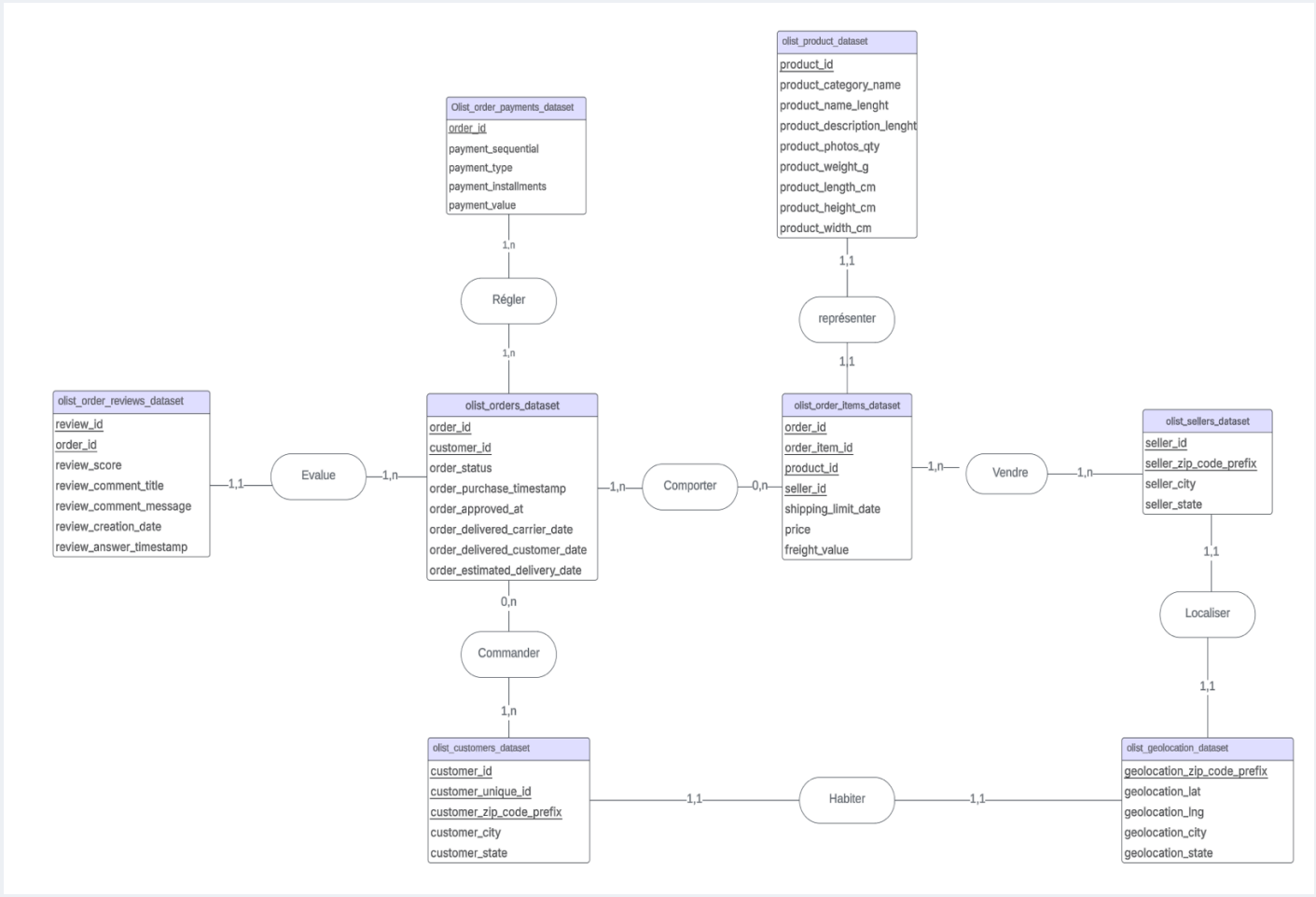
Objectif :

Notre objectif dans ce projet est de créer une solution de machine learning capable de prédire le produit gagnant en se basant sur plusieurs indicateurs tels que les ventes par catégories, les moyens de paiement, les avis et retours clients, etc., ainsi que de déterminer la région cible pour nos publicités, le tout sans nécessiter un grand budget. De plus, pour la création de notre boutique e-commerce, nous devons spécifier les moyens de paiement, ce qui représente un grand défi pour les professionnels du domaine. C'est pourquoi nous avons également décidé d'identifier les moyens de paiement les plus utilisés par région.

Jeux de données :

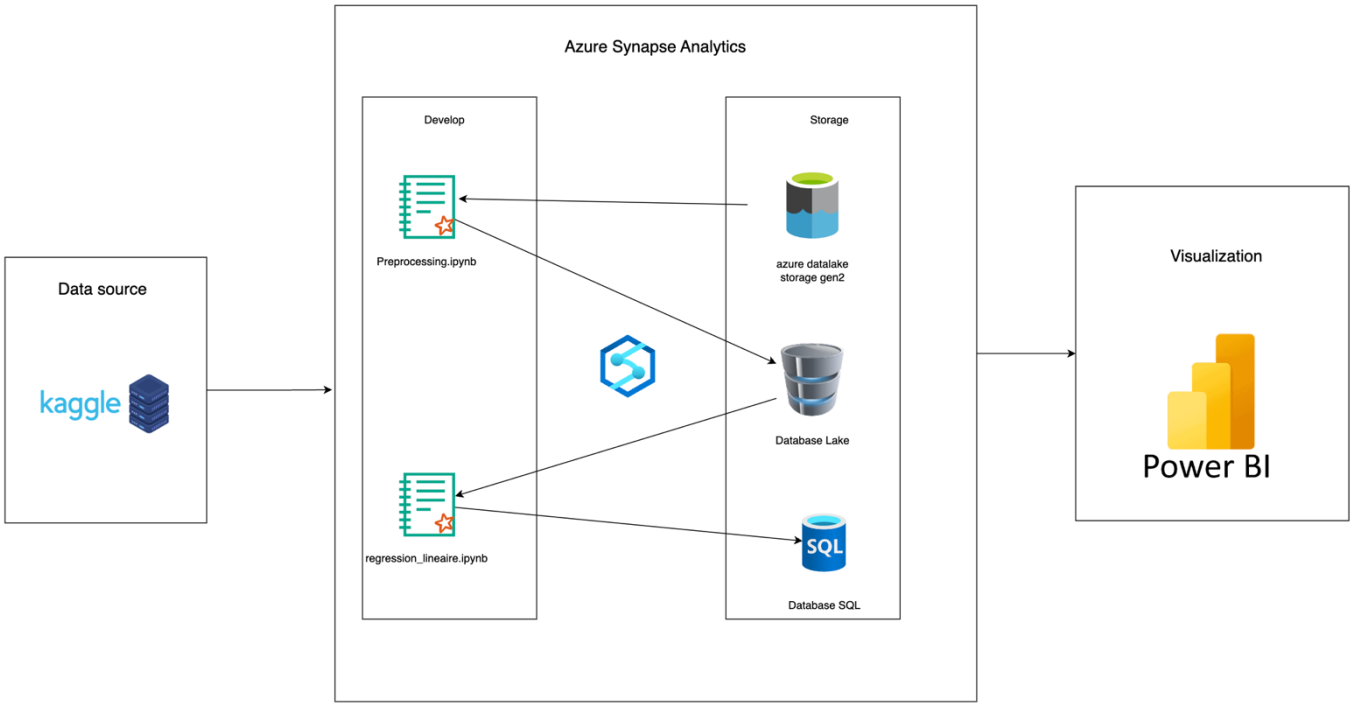
Le périmètre de notre projet sera le marché brésilien. L'ensemble des données a été généreusement fourni par Olist, le plus grand marché en ligne du Brésil. Olist connecte les petites entreprises de tout le pays aux grandes chaînes de distribution sans complications, et avec un seul contrat. Ces commerçants peuvent vendre leurs produits via la boutique Olist et les expédier directement aux clients en utilisant les partenaires logistiques d'Olist.
(Source de données : https://www.kaggle.com/datasets/olistbr/brazilian-ecommerce)

Les données sont divisées en plusieurs ensembles de données, pour une meilleure compréhension, voici un modèle conceptuel de données :


olist_product_dataset
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
Olist_order_payments_dataset
order_id
payment_sequential
payment_type
payment_installments
payment_value
1,n
Régler
1,n
1,1
représenter
1,1
olist_order_reviews_dataset
review_id
order_id
review_score
review_comment_title
review_comment_message
review_creation_date
review_answer_timestamp
1,1
Evalue
1,n
olist_orders_dataset
order_id
customer_id
order_status
order_purchase_timestamp
order_approved_at
order_delivered_carrier_date
order_delivered_customer_date
order_estimated_delivery_date
1,n
Comporter
0,n
olist_order_items_dataset
order_id
order_item_id
product_id
seller_id
shipping_limit_date
price
freight_value
1,n
Vendre
1,n
olist_sellers_dataset
seller_id
seller_zip_code_prefix
seller_city
seller_state
1,1
Localiser
1,1
0,n
Commander
1,n
olist_customers_dataset
customer_id
customer_unique_id
customer_zip_code_prefix
customer_city
customer_state
1,1
Habiter
1,1
olist_geolocation_dataset
geolocation_zip_code_prefix
geolocation_lat
geolocation_lng
geolocation_city
geolocation_state

Architecture et traitements :

1. Schéma d'architecture



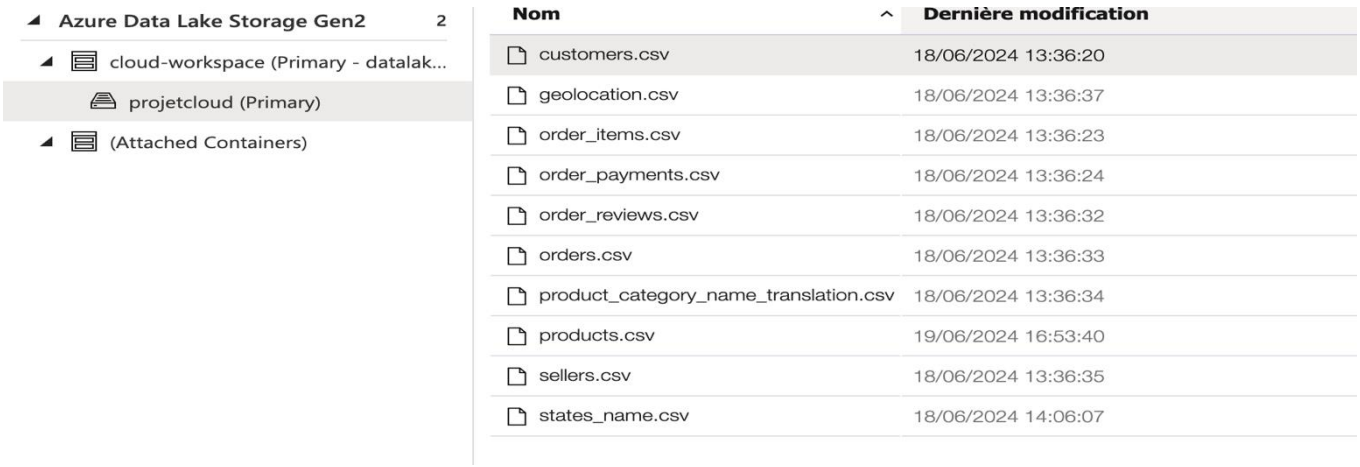
2. Collecte et stockage des données (Dataset) :

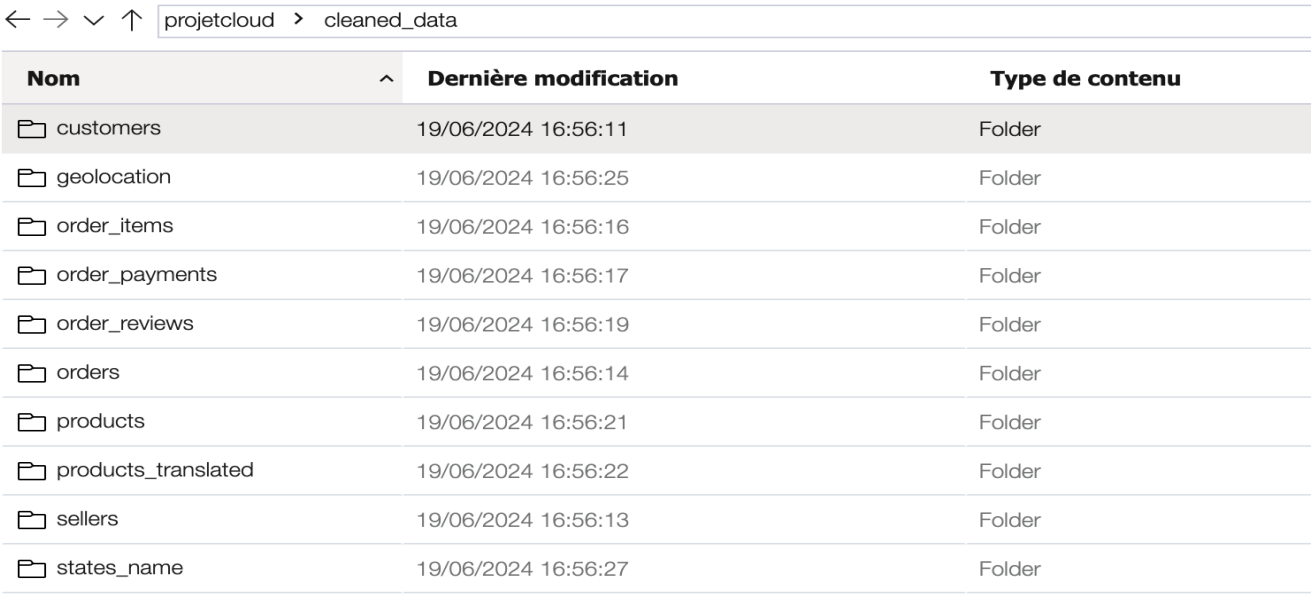
Nous opterons pour Azure, un fournisseur de services cloud renommé, comme notre infrastructure principale et Azure synapse comme service analytique pour nos traitements. Les données seront récupérées et stockées dans un contenaire sur data lake storage gen2 d'azure, garantissant ainsi une gestion sécurisée et évolutive de nos jeux de données.

Notre gestion de stockage sera comme suite :

- *Dossier raw_data* : Dossier contenant les données brutes sous forme de fichiers CSV.
- *Dossier cleaned_data (fichiers parquet)* : Dossier cintenant les données néttoyés (suppression des lignes dupliquées ,traitement de données aberrantes et nulles,) et transformées (traduction de la data du portuguais à l'anglais, changement des types de variables , ajout de colonnes opérationnels ,changement du type des fichiers du csv en parquet) .
- *Dossier Linear_regression* : Dossier contenant les données de prédiction issues de la régression linéaire.


Azure Data Lake Storage Gen2 2
cloud-workspace (Primary - datalak...
projetcloud (Primary)
(Attached Containers)
Nom
Dernière modification
Type de contenu
cleaned_data 18/06/2024 12:55:00 Folder
linear_regression 19/06/2024 18:43:15 Folder
raw_data 18/06/2024 12:54:53 Folder
synapse 18/06/2024 12:49:32 Folder


Azure Data Lake Storage Gen2 2
cloud-workspace (Primary - datalak...
projetcloud (Primary)
(Attached Containers)
Nom
Dernière modification
customers.csv 18/06/2024 13:36:20
geolocation.csv 18/06/2024 13:36:37
order_items.csv 18/06/2024 13:36:23
order_payments.csv 18/06/2024 13:36:24
order_reviews.csv 18/06/2024 13:36:32
orders.csv 18/06/2024 13:36:33
product_category_name_translation.csv 18/06/2024 13:36:34
products.csv 19/06/2024 16:53:40
sellers.csv 18/06/2024 13:36:35
states_name.csv 18/06/2024 14:06:07


projetcloud > cleaned_data
Nom
Dernière modification
Type de contenu
customers 19/06/2024 16:56:11 Folder
geolocation 19/06/2024 16:56:25 Folder
order_items 19/06/2024 16:56:16 Folder
order_payments 19/06/2024 16:56:17 Folder
order_reviews 19/06/2024 16:56:19 Folder
orders 19/06/2024 16:56:14 Folder
products 19/06/2024 16:56:21 Folder
products_translated 19/06/2024 16:56:22 Folder
sellers 19/06/2024 16:56:13 Folder
states_name 19/06/2024 16:56:27 Folder

1. Traitement de données :
   1.1. Nettoyage et transformation de données (Preprocessing) :

Le traitement de données est effectuée par le framework PySpark.
On a procédé comme suite :

- Nettoyage de données : Traitements des valeurs aberrantes et nulles , suppression des lignes dupliquées
- Transformation de données : Traduction de la data du portuguais à l'anglais, changement des types de variables , ajout de colonnes opérationnels ,changement du type des fichiers du csv au parquet.

```
# Suppression des valuers nulles de "order_reviews"
dataframes["order_reviews"] = dataframes["order_reviews"] \
    .filter(F.col("review_creation_date").isNotNull() & F.col("review_answer_timestamp").

# Vérification que toutes les valeurs nulles de order_reviews ont été supprimées
null_counts = count_nulls(dataframes["order_reviews"])
print(f"Null values in order_reviews DataFrame :")
null_counts.show(truncate=False)
```

✓ - Commande exécutée en 1 s 59 ms le 4:56:09 PM, 6/19/24

```
Null values in order_reviews DataFrame :
```

```
# Remplacement des valeurs nulles de "products" : String -> "Other"  &  Int -> -1
dataframes["products"] = dataframes["products"] \
    .withColumn("product_category_name", F.when(F.col("product_category_name").isNull(),
    .withColumn("product_name_length", F.when(F.col("product_name_length").isNull(), -1)
    .withColumn("product_description_length", F.when(F.col("product_description_length")
    .withColumn("product_photos_qty", F.when(F.col("product_photos_qty").isNull(), -1).c
    .withColumn("product_weight_g", F.when(F.col("product_weight_g").isNull(), -1).other
    .withColumn("product_length_cm", F.when(F.col("product_length_cm").isNull(), -1).oth
    .withColumn("product_height_cm", F.when(F.col("product_height_cm").isNull(), -1).oth
    .withColumn("product_width_cm", F.when(F.col("product_width_cm").isNull(), -1).other

# Vérification que toutes les valeurs nulles de products ont été remplacée
null_counts = count_nulls(dataframes["products"])
print(f"Null values in products DataFrame :")
null_counts.show(truncate=False)
```

Ajout champs

```
ataframes["order_items"] = dataframes["order_items"] \
   .withColumn("total_items_value", F.col("price") * F.col("order_item_id")) \
   .withColumn("total_freight_value", F.col("freight_value") * F.col("order_item_id")) \
   .withColumn("total_order_value", F.col("total_items_value") + F.col("total_freight_val

ataframes["order_items"].show(10)
```

✓ - Commande exécutée en 501 ms le 4:56:11 PM, 6/19/24

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

```python
# Jointure entre products et products_translated sur la colonne product_category_name
joined_df = dataframes["products"].alias("p") \
    .join(
        dataframes["products_translated"].alias("pt"),
        F.col("p.product_category_name") == F.col("pt.product_category_name"),
        "left"
    )

# Suppression du champs avec le om du produit en portugais
joined_df = joined_df.drop("product_category_name")

# Renommage de la colonne product_category_name_english
joined_df = joined_df.withColumnRenamed("product_category_name_english", "product_categor

# Réagencement des colonnes du DataFrame
new_columns = [joined_df.columns[0]] + ["product_category_name"] + [col for col in joined
dataframes["products"] = joined_df.select(new_columns)

# Mettre à jour le DataFrame dans le dictionnaire
dataframes["products"].show(5)

```

✓ - Commande exécutée en 1 s 106 ms le 4:56:00 PM, 6/19/24

1.2. Machine learning (régression linéaire) :

1.2.1. Traitements

Afin de réponde à notre sujet et d'arriver à notre objectif de trouver le niche gagnant des produits pour le ecommerce, nous avons opté pour un apprentissage supervisé qui se représente dans la régression linéaire pour prédire les ventes futures de chaque catégorie de produits dans les années 2019, 2020 et 2021.

Régression linéaire

```python
# Regression linéaire : prévision des prix
schema = StructType([
    StructField("category", StringType(), False),
    StructField("sales", IntegerType(), False),
    StructField("year", IntegerType(), False),
    StructField("month", IntegerType(), False),
    StructField("date", TimestampType(), True),
])

price_predictions = spark.createDataFrame(sc.emptyRDD(), schema)

price_predictions.show()
```

[43] ✓ - Commande exécutée en 2 s 776 ms le 9:26:47 PM, 6/19/24

```
+--------+-----+----+-----+----+
|category|sales|year|month|date|
+--------+-----+----+-----+----+
+--------+-----+----+-----+----+
```

```python
def regression_lineaire(df_train, future_date, category) :
    df_train = df_train.groupBy("year", "month") \
        .agg(F.sum("sales").alias("sales"))

    # Assembler les fonctionnalités
    assembler = VectorAssembler(
        inputCols=["year", "month"],
        outputCol="features"
    )

    # Transformer les données en utilisant l'assembler
    sales_data = assembler.transform(df_train)
    sales_data = sales_data.select("features", "sales")

    # Créer le modèle de régression linéaire
    lr = LinearRegression(featuresCol="features", labelCol="sales")

    # Ajuster le modèle aux données d'entraînement
    lr_model = lr.fit(sales_data)

    # Transformer les données en utilisant l'assembler
    df_predicted = spark.createDataFrame(future_date, ["year", "month"])
    df_predicted = assembler.transform(df_predicted).select("features")

    # Prédire les prix pour les dates futures
    df_predicted = lr_model.transform(df_predicted)

    # Assemblage des DataFrames train et predicted en RDD
    rdd = sc.parallelize(sales_data.union(df_predicted).rdd.collect()) \
        .map(lambda row: Row(features=row.features.toArray().tolist(), sales=row.sales)) \
        .map(lambda x: (int(x.features[0]), int(x.features[1]), x.sales))

    # Créer un DataFrame à partir de l'RDD avec les noms de colonnes appropriés
    df = rdd.toDF(["year", "month", "sales"]) \
        .withColumn("category", F.lit(category)) \
        .withColumn("date", F.to_date(F.concat(F.col("year"), F.lit("-"), F.col("month"), F.lit("-01")))) \
        .select("category", "sales", "year", "month", "date")
    return df
```

De plus, pour avoir une bonne visualisation et faciliter l'analyse des données nous avons généré également en plus, un dataframe avec les 10 meilleures catégories prédies au niveau de ventes.

```python
n = 10

# Récupération de la plus grande date de prédiction
max_date_row = price_predictions.agg(F.max("date")).head()
max_date = max_date_row[0]

# Récupération des n catégories ayant les meilleures ventes
best_categories_predicted = price_predictions.select("category") \
    .filter(F.col("date") == max_date) \
    .orderBy(F.col("sales").desc()) \
    .head(n)

# Mise sous forme de liste
best_categories_predicted = [c.category for c in best_categories_predicted]
best_categories_predicted
```

✓ - Commande exécutée en 51 s 634 ms le 9:36:28 PM, 6/19/24

```
['health_beauty',
 'watches_gifts',
 'bed_bath_table',
 'housewares',
 'computers_accessories',
 'sports_leisure',
 'furniture_decor',
 'auto',
 'baby',
 'telephony']
```

```python
# Création du dataframe avec uniquement les n meilleures catégories
price_predictions_top_category = price_predictions \
    .filter(F.col("category").isin(best_categories_predicted))

price_predictions_top_category.show()
```

1.2.2. Représentation graphique des prédictions. (Matplotlib)

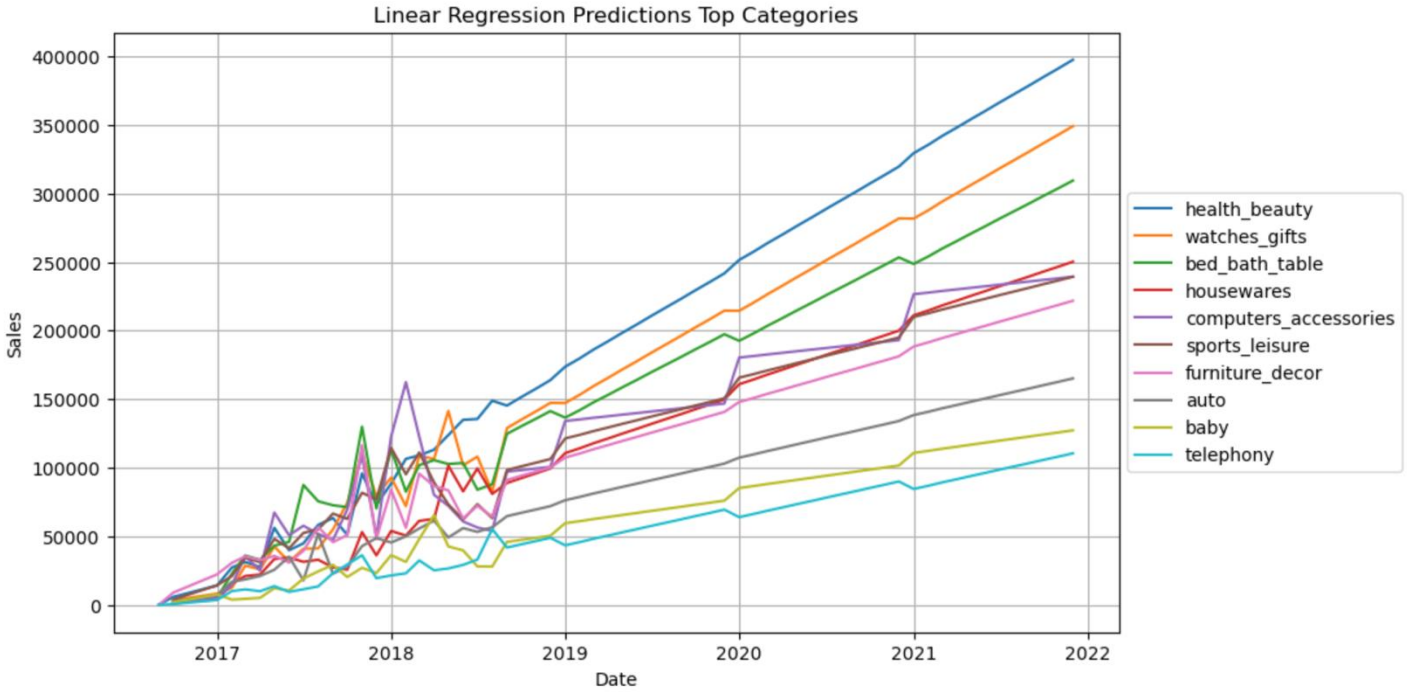
```python
# Initialiser la figure
plt.figure(figsize=(10, 6))

for category in best_categories_predicted :
    # Choisir la bonne catégorie et trier les données par date
    predictions_pd = price_predictions_top_category.filter(F.col("category") == category).orderBy("date").toPandas()

    # Tracer les données
    if not predictions_pd.empty:
        plt.plot(predictions_pd['date'], predictions_pd['sales'], label=category)


plt.xlabel('Date')
plt.ylabel('Sales')
plt.title('Linear Regression Predictions Top Categories')
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
plt.grid(True)
plt.show()
```

✓ - Commande exécutée en 12 s 827 ms le 9:37:58 PM, 6/19/24



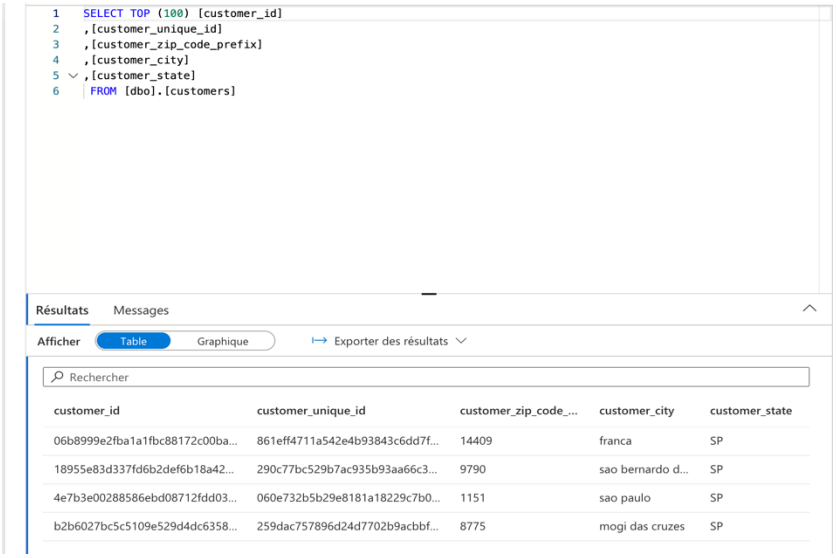
1.3. Mise en place d'un Datwarehouse.

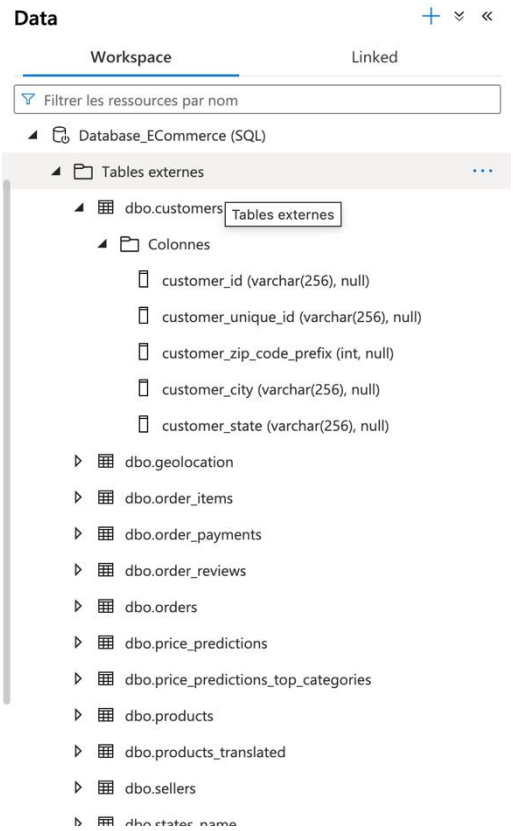
Après le traitement de la machine learning, tout les données ainsi que les données de prédictions sont sauvegardés dans la zone de staging (data lake storage gen2) sous forme de fichiers parquet.

```python
# Chemin de destination dans ADLS Gen2
predictions_path = "abfss://projetcloud@datalakecloud.dfs.core.windows.net/linear_regression/price_predictions"
top_category_predictions_path = "abfss://projetcloud@datalakecloud.dfs.core.windows.net/linear_regression/price_predictions_top_category"

# Écrire les DataFrame au format Parquet
price_predictions.write.mode("overwrite").parquet(predictions_path)
price_predictions_top_category.write.mode("overwrite").parquet(top_category_predictions_path)
```

Ensuite, pour avoir une bonne structuration de données, nous avons créé un datawarehouse (base de données SQL) qu'on a alimenté par nos fichiers parquets groupés par catégories sous forme de tables externes (dbo.customers, etc).

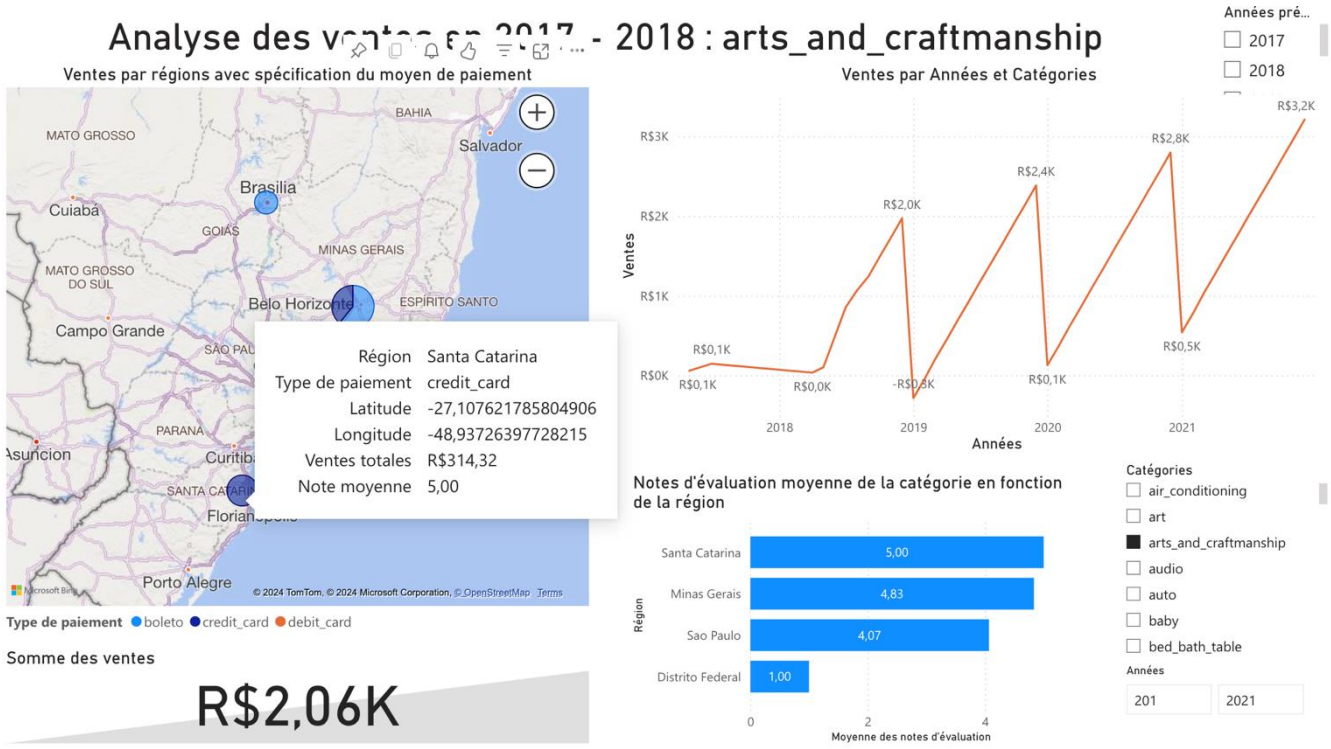




Visualisation de données (Rapport PowerBI).

À partir des données stockés dans le datawarehouse sur azure synapse, nous avons créé un rapport PowerBI, pour analyser les résultats de nos traitements sous formes de plusieurs visualisations :

- Carte : représente les régions avec les grandes ventes par catégories avec spécification du moyen de paiement.
- Courbe : représente les prédictions de ventes par année et catégorie
- Box KPI : représente la somme des vente par catégorie.
- Histogramme : représente les notes d'évaluation moyenne de la catégorie en fonction de région.
- Filtres : catégories et années.
- Titres dynamiques et visualisation dynamique.



Conclusion :

Nous avons conclu à partir de cette analyse que la catégorie "health_beauty" est la plus rentable et présente un grand potentiel pour l'avenir, avec un gain de 0,40M $R en décembre 2021, soit à peu près la moitié des gains cumulés de 2017 à 2019. Le moyen de paiement le plus utilisé est la carte de crédit, donc nous opterons pour une passerelle de paiement de type Stripe ou similaire pour notre boutique. Les produits de cette catégorie ont de bonnes notes, supérieures à 4/5, et sont vendus dans toutes les régions du Brésil, avec une majorité des ventes à São Paulo.