Rapport Pipeline Data Airbnb

1. **Introduction :**

Airbnb permet à des particuliers de louer tout ou une partie de leur propre habitation comme logement d'appoint. Le site offre une plateforme de recherche et de réservations entre la personne qui met à disposition son logement (le bailleur) et un locataire. Il couvre plus de 1,5 million d'annonces dans plus de 34 000 villes et 191 pays. De la création, en août 2008, jusqu'en juin 2012, plus de 10 millions de nuits ont été réservées sur Airbnb.

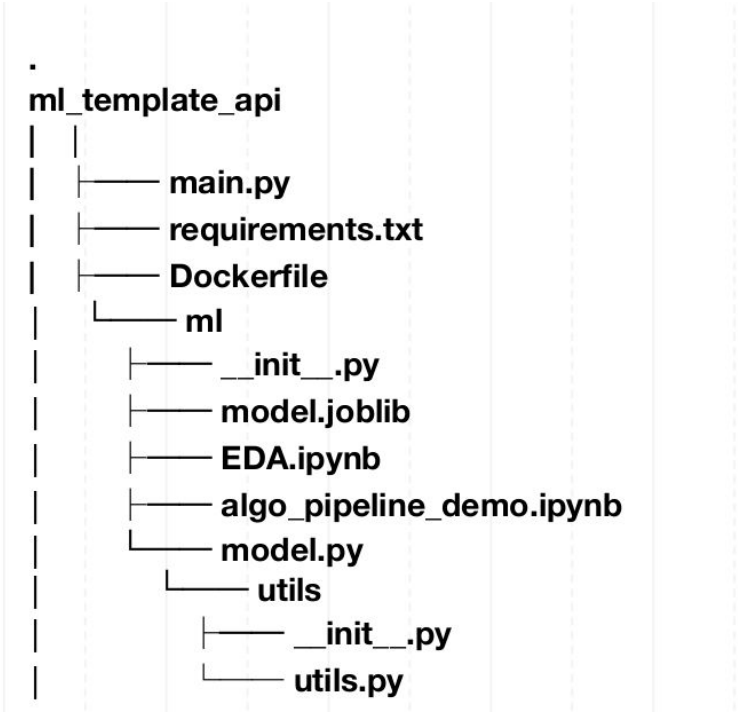
2. **But du Projet :**

On est un ingénieur pour un client et on doit présenter une mini pipeline data , de l'élaboration d'une problématique pertinente , la collecte de données , le développement d'une API , la paquetage Docker la tester et rédiger une documentation claire et explicite.

3. **Data set Utilisées :**

Price availability csv : https://storage.googleapis.com/h3-data/price_availability.csv
Listings Final csv : https://storage.googleapis.com/h3-data/listings_final.csv

4. **Architecture GPC :**



5. **Cleaning des données par un Model Builder :**

```python
#importer vos librairies
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import sklearn as sk
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import joblib

class DataHandler:
    def __init__(self) :
        self.dataprice = None
        self.datalisting = None
        self.merge = None
    def get_data(self):
    def group_data(self):
    def get_process_data(self):


class FeatureRecipe:

    def __init__(self,data:pd.DataFrame):

    def separate_variable_types(self) -> None:
    def drop_uselessf(self):
    def get_duplicates(self):
    def drop_duplicate(self):
    def drop_nanp(self, threshold: float):
        def deal_nanp(data:pd.DataFrame, threshold: float):
    def deal_dtime(self):
        """ TODO : Traiter les DateTime """
        pass

    def prepare_data(self, threshold: float):

class FeatureExtractor:
    """
    Feature Extractor class
    """
    def __init__(self, data: pd.DataFrame, flist: list):
```

```
        """
            Input : pandas.DataFrame, feature list to drop
            Output : X_train, X_test, y_train, y_test according to
sklearn.model_selection.train_test_split
        """
        self.df = data
        self.flist = flist
        self.X_train = None
        self.X_test = None
        self.y_train = None
        self.y_test = None

    def extract(self):
    def split(self,size:float,rand:int,y:str):
    def get_process_data(self,size:float,rand:int,y:str):

class ModelBuilder:
    """
        Class for train and print results of ml model
    """
    def __init__(self, model_path: str = None, save: bool = False):
        self.model_path = model_path
        self.save = save
        self.reg = LinearRegression()
        #self.time = DT.datetime.now()

    def __repr__(self): # class courier python , affichage par default
isntancié , methode __str__
        return f'Model Builder : model_path = {self.model_path} , save =
{self.save}.'

    def train(self,X,y):

    def predict_test(self, X) -> np.ndarray:  # certain ligne of X_test or
    def predict_from_dump(self, X) -> np.ndarray: #

    def save_model(self):
    def print_accuracy(self,X_test,y_test):
    def load_model(self):
```

Lien Github Code Source : https://github.com/Mohamed-Khalil67/Piple-line.git

6. Analyse graphique des données (EDA) :

- Dataset Final aprés cleaning

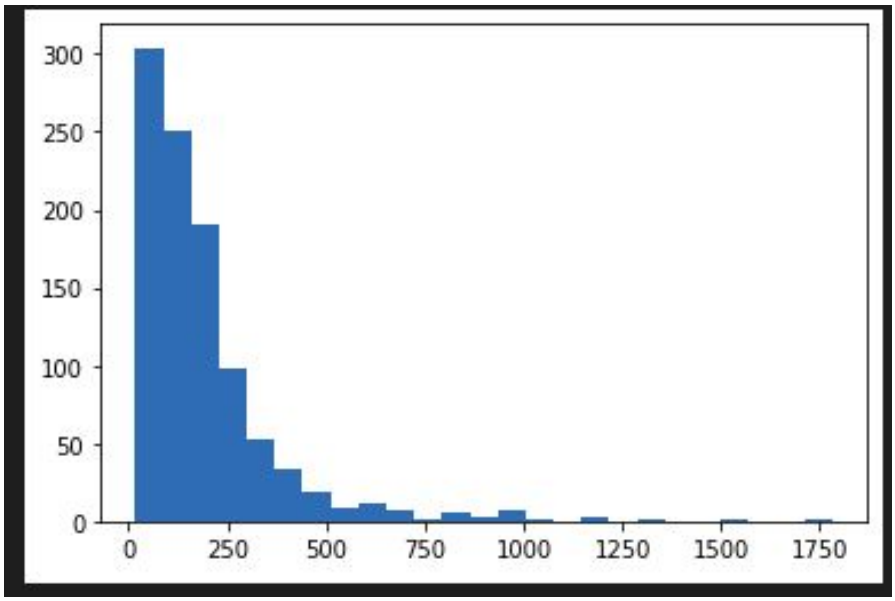
```
data_merge.describe()
```

	listing_id	local_price	latitude	longitude	person_capacity	beds	bedrooms	bathrooms	pricing_weekly_factor	pricing_monthly_factor
count	9.990000e+02	999.000000	999.000000	999.000000	999.000000	999.000000	999.000000	999.000000	999.000000	999.000000
mean	1.543276e+07	189.304613	48.864251	2.345202	3.669670	2.058058	1.326326	1.230230	0.942882	0.871809
std	9.305853e+06	184.425492	0.009824	0.031837	2.190568	1.500378	0.998806	0.524497	0.073776	0.162974
min	5.609300e+04	17.863724	48.844372	2.268992	1.000000	0.000000	0.000000	0.000000	0.600000	0.330000
25%	6.341818e+06	76.525248	48.856929	2.326608	2.000000	1.000000	1.000000	1.000000	0.900000	0.750000
50%	1.727835e+07	144.000000	48.864368	2.349152	3.000000	2.000000	1.000000	1.000000	1.000000	1.000000
75%	2.378911e+07	230.372596	48.871906	2.366973	4.000000	3.000000	2.000000	1.000000	1.000000	1.000000
max	2.879280e+07	1780.590674	48.882691	2.412700	16.000000	16.000000	6.000000	5.000000	1.000000	1.000000

```
{}
```

- Diagram de répartition des données (Type Gaussiennes) (Histogramme)

```
plt.hist(data_merge.local_price, bins = 25)
```



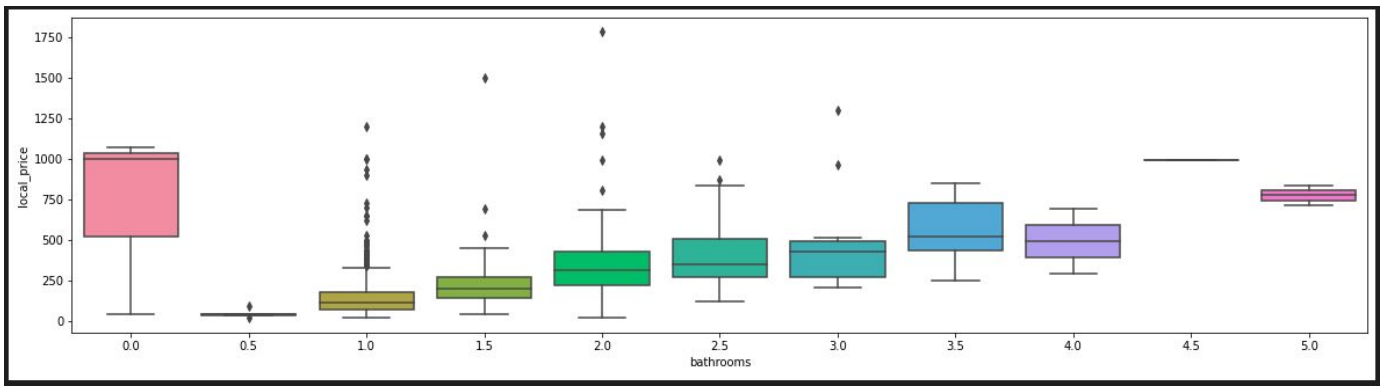
Commentaire sur Histogramme de données de local_price :
Ce qu'on observer :-

- Les nombres de prix des appartements de 0 à 250 sont plus hauts que les autres prix.

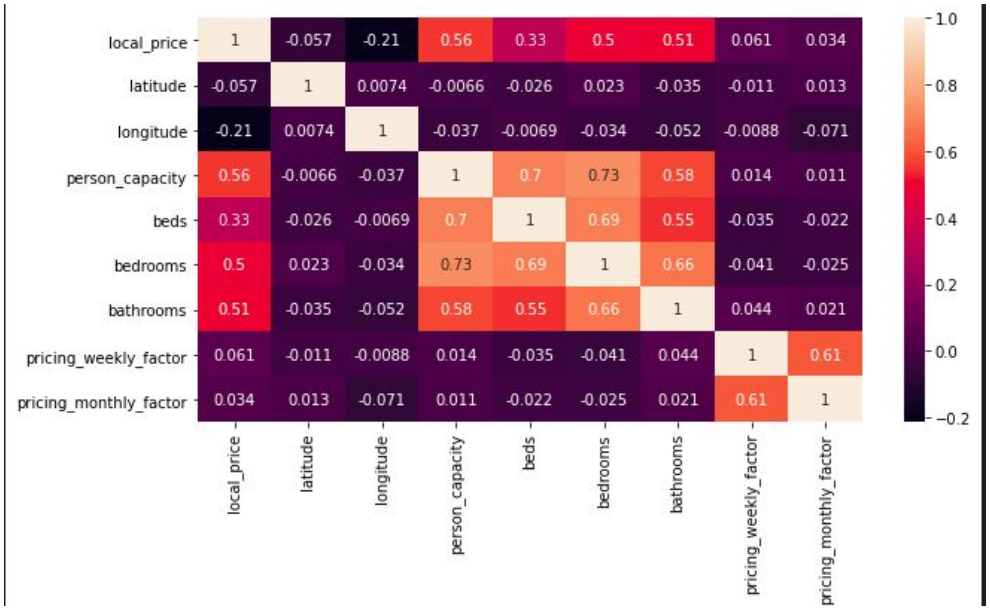
- Il y a une grande quantité des appartements qui sont pas chères et on peut faire plus d'études
- D'autres paramètres peuvent être utilisés : la position géographique ou la capacité des personnes et la surface de l'appartement.

    - Boîte de Moustache

```
fig=plt.gcf()
fig.set_size_inches(20,5)
sns.boxplot(x='bathrooms',y='local_price',data=data_merge);
```



    - Heatmap



```
data_corr=pd.DataFrame(data_merge,columns=['local_price','latitude','longitude','person_capacity','beds','bedrooms','bathrooms','pricing_weekly_factor',
'pricing_monthly_factor'])

sns.heatmap(data_corr.corr(),annot=True)
plt.show()
```

- Commentaire sur Heatmap Graphique :

On observe que le paramètre Beds par rapport aux autres paramètres

- Beds et person_capacity : 0.7
- Beds et bedrooms : 0.69

Ces valeurs signifie la précision de la compatibilité de ces paramètres qui est conforme au raisonnement que le nombre de lits conforme au nombre de chambres de sommeils.

7. **Model Building :**

```
from utils.utils import DataHandler,FeatureRecipe,FeatureExtractor

def DataManager(d:DataHandler=None, fr: FeatureRecipe=None,
fe:FeatureExtractor=None): # script model.py qui sort une model .joblib , alors
dans model.py datamanager model builder et va sortir un fichier .joblib
    """
        Fonction qui lie les 3 premières classes de la pipeline et qui return
FeatureExtractor.split(0.1)

    Args:
        d (DataHandler, optional): [description]. Defaults to None
        fr (FeatureRecipe, optional): [description]. Defaults to None
        fe (FeatureExtractor, optional): [description]. Defaults to None
    """
    print("Start Data Managing")
    d = DataHandler()
    d.get_process_data()
    print("Data Loaded")
    fr = FeatureRecipe(d.merge)
    print("Filtering data with threshold de 40%")
    fr.prepare_data(0.3)
    print("Filtering done")

    print("Feature Extracting ( Split )")
```

```
   flist =
['listing_id','city','neighborhood','latitude','longitude','is_rebookable','is_
new_listing','is_fully_refundable','is_host_highly_rated','is_business_travel_r
eady','pricing_weekly_factor','pricing_monthly_factor','name','type']
   print("Flist has been Chosen")
   fe = FeatureExtractor(d.merge,flist)
   X_train,X_test,y_train,y_test = fe.get_process_data(0.3,42,'local_price')

   return X_train,X_test,y_train,y_test
```

8. **API , Conteneurisation et déploiement GCP :**

```
import streamlit as st
import seaborn as sns
import pandas as pd
import numpy as np
import plotly.express as px
import matplotlib.pyplot as plt
#import os
from ml.utils.utils import
DataHandler,FeatureRecipe,FeatureExtractor,ModelBuilder

st.header(' Pipeline Data GCP " Cloud Computing " ')
st.markdown(" Two File Data CSV : listings_final.csv / price_availability.csv
")

print("Start Data Managing")
d = DataHandler()
d.get_process_data()
print("Data Loaded")
fr = FeatureRecipe(d.merge)
print("Filtering data with threshold de 40%")
fr.prepare_data(0.3)
print("Filtering done")

print("Feature Extracting ( Split )")
flist =
['listing_id','city','neighborhood','latitude','longitude','is_rebookable','is_
new_listing','is_fully_refundable','is_host_highly_rated','is_business_travel_r
eady','pricing_weekly_factor','pricing_monthly_factor','name','type']
print("Flist has been Chosen")
fe = FeatureExtractor(d.merge,flist)
X_train,X_test,y_train,y_test = fe.get_process_data(0.3,42,'local_price')
```

```
m = ModelBuilder('.') # model_path directory
m.train(X_train, y_train)
m.print_accuracy(X_train,y_train)

df = d.merge

st.title(" Airbnb Web Application")
st.markdown("Data Cleaning And Presentation of Airbnb")
st.header(" Airbnb Pricing Service Algorithme ")

if st.checkbox("Show DataSet Rows of header"):
   number = st.number_input("Number of Rows to View")
   st.dataframe(df.head(int(number)))

selec = []
for col in df.columns:
  selec.append(col)
option = st.multiselect("Selectionner les colonnes",(selec))

st.table(df[option].head())

st.header("Discription Choices Boxes :")

if st.button("Afficher le type des colonnes du dataset"):
   st.table(df[option].dtypes)
if st.button("La shape du dataset, par lignes et par colonnes"):
   st.write(df[option].shape)
if st.button("Afficher les statistiques descriptives du dataset"):
   st.write(df[option].describe())

st.header('Split Details :-')
st.subheader('X_train :')
st.write(X_train)
st.subheader('X_test :')
st.write(X_test)
st.subheader('y_train :')
st.write(y_train)
st.subheader('y_test :')
st.write(y_test)

st.subheader('Split Accuracy :')
st.write('Coefficient Accuracy : ',m.reg.score(X_test,y_test)*100,'%')
st.header("What kind of graph ?")
status = st.radio("Choose ur way of drawing
:",('Boxplot','Correlation','Distplot','Pairplot','Histogram'))
```

```
if status == 'Histogram':
   fig=plt.gcf()
   fig.set_size_inches(20,5)
   st.write(plt.hist(df.local_price, bins = 25))
   st.pyplot()
elif status == 'Boxplot':
   fig=plt.gcf()
   fig.set_size_inches(20,5)
   st.write(sns.boxplot(x='beds',y='local_price',data=df))
   st.pyplot()
elif status == 'Correlation':
   st.write(sns.heatmap(df.corr(), annot=True))
   st.pyplot()
elif status == 'Distplot':
   st.write(sns.distplot(df))
   st.pyplot()
elif status == 'Pairplot':
   st.write(sns.pairplot(df))
   st.pyplot()
```

Lien de Streamlit API sur VM instance créée par moi : http://104.199.46.74:8000/

9. **Pacquaging, POO & refractoring :**

Les ports utilisée pour Jupyter sur le VM instance est 8888 ,mais ça doit être lancé du vm avec ce ligne de command en terminal :

Jupyter notebook --no-browser --port=8888 --ip='0.0.0.0'

Le port 8000 est utilisé pour le Streamlit avec le commande suivant en terminal dans le VM instance (toujours active) :

Streamlit run main.py --server.port=8000

Link : **http://104.199.46.74:8000/**

Note book algo_pipeline_demo.ipynb :

```
Filtering done
Feature Extracting ( Split )
Flist has been Chosen
-- X and y Values to be put while discarding the flist that we dont need--
-- Extraction Done --
-- Using Split mehtode --
-- Afficher la shape de vos données --
 - - - Training Start - - -
 - - - Finish training - - -
 - - - Accuracy Printing - - -
Coeffecient Accuracy : 37.35359243633909 %
 - - - Finished Accuracy - - -
Model Builder : model_path = . , save = False.
 - - - Saving Model - - -
 - - - Finished Saving Model - - -
```

Commentaires Sure l'algolrithme de Datamanging et Model Building :

- Les datas utilisées sont celles de price availability et pricing
- De ce qu'on observe on a une accuracy de 37.35% :
    - Ce qui pas bien par rapport aux paramtres utilisées avec y
    - On aura une meilleur accuracy avec plus de parametres à utiliser ,
    - Les paramétres utilisées ici : prenant local price comme y et les X : beds , bedrooms , bathrooms , person_capacity.

10. <u>En conclusion :</u>

Un Set de Data Pipeline est une ensemble d'instructions qui extrait les données de différentes sources. C'est un processus automatisé : de prendre par exemple , des colonnes d'un certain base de données et le fait merger avec d'autres colonnes ; ce procédure s'appelle "job" et la pipeline se compose de plusieurs "job".

A la fin , le résultat obtenu à partir de ce modèle de pipeline nous a donné une résultat pas totalement satisfaisant avec une accuracy de 37.35 % pour le dataset Airbnb. Ce qui est une valeur de précision Balanced , mais ni underfitting et sûrement ni overfitting.

P.S. (de cours)

Adresses IPs :

127.0.0.1 = adresse pas local mais en cloud , l'autre connexion machine.
0.0.0.0 = adresse local du machine.
255.255.255.255 = broad Cast signifie tout le réseau.