RAPPORT TEST TECHNIQUE DATA CRUNCH

<< KENGO BENITO, Etudiant M2 IA/ML U.Paris >>

Sommaire

Introduction

Tictactrip, est un comparateur en ligne de réservation de voyage a travers l'Europe qui favorise l'intermodalité en combinant bus, train et covoiturage . Dans le cadre du processus de recrutement d'un stagiaire, l'équipe data a mis en place un test technique dans le but d'évaluer les capacités techniques et qualités interpersonnelles de leur futur stagiaire. Il a été ainsi mis à ma disposition , un ensemble de dataset, et j'ai pour mission d'exploiter ces données pour répondre aux sollicitations de l'équipe data.

1. Analyse du besoin

Les dataset fournis sont des fichiers au format CSV :

- ***ticket_data.csv*** : Contenant un historique de ticket

Ce dataset nous permet de suivre de bout en bout, un trajet. En effet, on retrouve des informations essentielles comme la présence de stations intermédiaires, qui sont des points de correspondances, ou les voyageurs peuvent changer de mode de transport ou de compagnie de voyage .

- ***cities.csv*** : les villes desservies par tictactrip

On y retrouve les coordonnées de chaque villes, qui nous permettent de calculer les distances entre la ville de départ, et la ville d'arrivé

- ***stations.csv*** : les stations desservies par tictactrip

On y retrouve également les coordonnées de chaque station, qui nous permettent de calculer les distances entre deux stations d'arrêt

- ***providers.csv*** : infos sur les différents providers

On y retrouve les informations essentielles sur la compagnie du voyage. Ce qui permet ainsi d'obtenir le type de convoi utilisé pour chaque trajet.

Les missions se présentent en 2 volets :

- Extraire des informations essentielles, type prix minimum, moyen et maximum et également la durée minimum, maximum et moyenne par trajet
- Déterminer le prix moyen et la durée moyenne d'un trajet , selon le type transport (train, bus …) et la tranche de distance du trajet (0-200 km, 201-800 km ..)

2. Nettoyage et préparation des données

- Remplacer les valeurs absentes : Lors de la lecture d'un CSV avec pandas, les valeurs vides prennent pour valeurs NaN, ce qui peut poser des difficultés à exploiter correctement les données. Il y' a lieu dès lors, de remplacer ces valeurs obtenues par des chaînes vides :

```
dataframe.replace(np.nan, '', regex=True)
```

- Convertir les dates : Les dates étant sous formes de chaînes de caractères, il est nécessaire de les convertir en type Datetime .

```
DATA_FORMAT = '%Y%m%d %H:%M:%S.%f'
pd.to_datetime('2017-10-13 14:00:00+00', format=DATA_FORMAT)
```

- Convertir les prix en euro (optionnels) : Afin d'avoir les prix sur un format plus courant, on peut diviser les prix par 100, et renommer la colonne de *price_in_cents* a *prices*

3. Implementation

a. Extraction d'informations essentielles par trajet

- **Prix :** Pour déterminer les prix minimum, maximum et moyen, on peut utiliser respectivement les fonction min(), max() , mean() de pandas, sur la colonne des prix .

```
min_price = ticket_data_df['prices'].min()
max_price = ticket_data_df['prices'].max()
avg_price = ticket_data_df['prices'].mean()
```

- **Durée :** Pour déterminer les durées minimum, maximum et moyen, on peut utiliser respectivement les fonctions min(), max() , mean() de pandas, sur une colonne durée qui sera créée en faisant la différence entre la date d'arrivée et la date de départ.

```
ticket_data_df['duration'] = ticket_data_df.apply(lambda x : x["arrival_ts"]-x["departure_ts"],axis=1)
ticket_data_df['duration_in_seconds'] = ticket_data_df.apply(lambda x : x['duration'].total_seconds(),axis=1)
min_duration = ticket_data_df['duration'].min()
max_duration = ticket_data_df['duration'].max()
avg_duration = ticket_data_df['duration'].mean()
```

b. Comparaison par groupe de type distance de trajet et type de transport

- **Déterminer les groupes de distance :** Afin de pouvoir déterminer les groupes de distances, on doit au préalable calculer les distances entre les villes . J'ai effectué une jointure entre les dataset des tickets et des villes, en croisant leur id.

  En répétant la même opération sur d_city, on obtient les coordonnées respectives des villes de départ et des villes d'arrivée pour chaque trajet dans le dataset de chaque trajet.

  La fonction `get_distancies()` a été définie pour calculer les distances entre deux coordonnées , en utilisant la librairie `geopy`.

```
def get_distancies(row):
    try:
      return geodesic((row["latitude_d_city"],row["longitude_d_city"]),
                      (row["latitude_o_city"],row["longitude_o_city"])).km
    except:
      return 0

ticket_data_df['distancies'] = ticket_data_df.apply(get_distancies,axis=1)
```

En fonction des distances obtenues, on va attribuer chaque trajet a un groupe de distance.

```
def set_distancies_group(row):
    if row["distancies"] >  2000:
      return "2000+km"
    elif row["distancies"] >  800 and row["distancies"] <=  2000:
      return "800-2000km"
    elif row["distancies"] >=  201 and row["distancies"] <=  800 :
      return "201-800km"
    elif row["distancies"] >=  0 and row["distancies"] <=  200:
      return "0-200km"

ticket_data_df['distancies_groups'] = ticket_data_df.apply(set_distancies_group,axis=1)
```

- **Déterminer les groupe de type de transport :**

  Ayant constaté que pour certains trajets, le type de transport pouvait changer aux stations intermédiaires, j'ai établi les groupes de type de transport, en fonction de cette information. On a donc des groupes de type de transport hybride (bus-train ...) , et des groupes uniques (bus, train ...). Les informations sur le transporteur, ayant été préalablement ajouté au dataset des ticket en faisant une jointure entre les tables concernés

```
def get_id_from_string(str) :
  return (str.replace('{','').replace('}','')).split(",")

def get_convoy_by_provider_id(id) :
    return (providers_df.loc[providers_df['id'] == int(id)]["transport_type"]).tolist()[0]

def get_convoy(row):
    convoy = row["transport_type"]
    if row["other_companies"] :
      mid_companies = get_id_from_string(row["other_companies"])
      if mid_companies :
          for id in mid_companies :
              c = get_convoy_by_provider_id(id)
              if c not in convoy :
                convoy = convoy + "-" + c
    return convoy

ticket_data_df['convoy'] = ticket_data_df.apply(get_convoy,axis=1)
```

- **Grouper par mode de transport, et par distance** : La fonction groupby de la librairie panda nous permet de regrouper par groupe de distance et convoies , en effectuant les agrégats max,min et mean sur les colonnes désirés a chaque fois.

```
d_max = ticket_data_df.groupby(["distancies_groups", "convoy"])["duration_in_seconds"].max().reset_index()
d_max["duration"] = pd.to_timedelta(d_max["duration_in_seconds"], unit='S')
del d_max["duration_in_seconds"]

d_min = ticket_data_df.groupby(["distancies_groups", "convoy"])["duration_in_seconds"].min().reset_index()
d_min["duration"] = pd.to_timedelta(d_min["duration_in_seconds"], unit='S')
del d_min["duration_in_seconds"]

d_mean = ticket_data_df.groupby(["distancies_groups", "convoy"])["duration_in_seconds"].mean().reset_index()
d_mean["duration"] = pd.to_timedelta(d_mean["duration_in_seconds"], unit='S')
del d_mean["duration_in_seconds"]

p_max = ticket_data_df.groupby(["distancies_groups", "convoy"])["prices"].max().reset_index()
p_min = ticket_data_df.groupby(["distancies_groups", "convoy"])["prices"].min().reset_index()
p_mean = ticket_data_df.groupby(["distancies_groups", "convoy"])["prices"].mean().reset_index()
```

4. Présentation des résultats

Pour répondre à la première question de ce test qui était d'extraire des informations essentielles, le code implémenté nous a fournis les résultats suivants :

Prix minimum par trajet	3.00 €
Prix moyen par trajet	43.82 €
Prix maximum par trajet	385.50 €

Durée minimum par trajet	0 jours 00:20:00 heures
Durée moyen par trajet	0 jours 07:04:37 heures
Durée maximum par trajet	20 jours 12:51:00 heures

Ensuite, pour ce qui etait de determiner la différence des prix et de durée par distance et par type de convois, on a obtenu le résultat suivant :

a. Prix min, max, moyen

Tranche de distances	Type de transport	Prix minimum	Prix moyen	Prix maximum
0-200 km	bus	8.50	21.35	229.00
	bus-train	9.90	31.35	63.30
	carpooling	3.00	11.77	128.50
	train	4.90	34.91	251.00
	train-bus	15.10	32.42	57.10
201-800 km	bus	10.00	34.60	145.96
	bus-train	23.50	81.44	224.80
	carpooling	8.50	32.31	138.00
	train	14.00	91.41	385.50

	train-bus	38.00	73.80	164.50
800-2000 km	bus	22.90	69.49	174.00
	carpooling	44.96	86.26	161.50
	train	19.40	154.63	375.50

b. Durée min,max et moyenne

Tranche de distances	Type de transport	Durée minimum	Durée moyenne	Durée maximum
0-200 km	bus	0 days 01:28:00	0 days 10:25:45.44	13 days 05:45:00
	bus-train	0 days 01:05:00	0 days 06:17:13.22	1 days 02:12:00
	carpooling	0 days 00:20:00	0 days 01:57:07.34	0 days 17:20:00
	train	0 days 00:39:00	0 days 04:25:59.87	1 days 10:27:00
	train-bus	0 days 01:40:00	0 days 03:29:30	0 days 12:24:00
201-800 km	bus	0 days 03:45:00	0 days 15:09:25.91	20 days 12:51:00
	bus-train	0 days 02:29:00	0 days 09:25:32.72	1 days 10:13:00
	carpooling	0 days 01:20:00	0 days 04:46:50.37	0 days 19:50:00
	train	0 days 01:08:00	0 days 07:39:33.29	2 days 00:27:00
	train-bus	0 days 03:55:00	0 days 12:56:20.25	1 days 07:20:00
800-2000 km	bus	0 days 12:15:00	1 days 03:32:10.02	14 days 17:00:00
	carpooling	0 days 07:50:00	0 days 13:06:00.83	1 days 05:10:00

	train	0 days 03:33:00	0 days 12:09:31.02	1 days 16:37:00

5. Difficultés rencontrés

- Prise en compte du dataset stations : Je n'ai pas réussi à trouver sous quel angle le dataset stations aurait pu nous être utile pour répondre efficacement au besoin demandé.
- Optimisation du code : Le code réalisé dans sa version initiale prenait un peu plus de temps, pour calculer les distances, il a fallu que je prépare des jointure en amont, afin d'utiliser directement les coordonnées des villes.
- Réaliser des agrégats sur des séries type timedelta. On ne peut pas faire des moyenne a partir de ces agrégats, raison pour laquelle j'ai jugés nécessaire de conserver les durées en secondes, pour faciliter les opérations de groupement

6. Axes d'améliorations

- Calcul de la distance : Le calcul de la distance n'est pas efficace pour l'instant. En effet, les distances sont calculées à l'aide d'une librairie `geopy` . On constate naturellement que les distances obtenues ne sont pas souvent toujours proches des distances réelles, car cette librairie calcule la distance dans

l’espace, tandis que dans des conditions réelles, une route a une trajectoire déjà bien définie. Comment obtenir de manière plus exacte les distances ?

Pour un tel projet, on peut mettre en place un script/cron qui calcule automatiquement les distances entre toutes les villes en arrière-plan, qui la stocke dans une base de données, avant de desservir cette information de manière plus rapide .

7. Bilan

Ce test était pour moi l’occasion de démontrer mes capacités techniques, mais également mes facultés d'analyse et de restitution de l'information. En effet, j’ai fait un total de 10 heures pleines sur ce projet qui s’est étendu sur 4 jours. L’analyse des données et des tâches, ayant pris 4 heures , j’ai passé 2 heures à implémenter, 2 heures à vérifier et optimiser et 2 dans la rédaction de ce rapport . Espérant ne pas être passé à côté de quelque chose d’important dans la réussite de ce test, j’ose croire que mes efforts seront récompensés .