Results:

1. Prix et Durée

Sur la base du fichier ticket_data.csv, le premier objectif est de trouver le prix le plus bas, le prix moyen et le prix le plus élevé, ainsi que le temps de trajet le plus bas, le plus élevé et le temps de trajet moyen pour chaque voyage.

Si j'ai bien compris, l'objectif est de regrouper les données par ville de départ (o_city) et par ville d'arrivée (d_city), le voyage étant défini comme le déplacement d'une ville à l'autre. Les résultats finaux seront présentés en conséquence.

1.1 Le prix

```
prix_result = ticket.groupby(['o_city', 'd_city'])['price_in_cents'] .agg(['min','mean','max'])
prix_result
```

		min	mean	max
o_city	d_city			
5	23	18600	20320.000000	22000
6	227	9860	11755.000000	13650
	504	2000	4042.666667	8920
	628	2600	2797.500000	3190
	845	700	864.626866	2420
...	...	...	...	...
11938	126	3000	4204.861111	5350
12124	1064	6000	6950.000000	7900
12166	857	5300	5300.000000	5300
12190	639	600	688.888889	850
	8937	9800	9800.000000	9800

1437 rows × 3 columns

1.2 Durée

		min	mean	max
o_city	d_city			
5	23	0 days 08:53:00	0 days 10:18:48	0 days 15:54:00
6	227	0 days 12:24:00	0 days 13:42:30	0 days 15:01:00
	504	0 days 05:36:00	0 days 08:17:24	0 days 12:20:00
	628	0 days 09:40:00	0 days 12:10:00	0 days 14:30:00
	845	0 days 01:00:00	0 days 01:19:54.626865671	0 days 04:11:00
...	...	...	...	...
11938	126	0 days 05:30:00	0 days 07:25:16.666666666	1 days 11:20:00
12124	1064	0 days 11:10:00	0 days 17:10:00	0 days 23:10:00
12166	857	0 days 21:55:00	0 days 21:55:00	0 days 21:55:00
12190	639	0 days 01:10:00	0 days 01:28:53.333333333	0 days 02:40:00
	8937	0 days 09:10:00	0 days 09:10:00	0 days 09:10:00

1437 rows × 3 columns

Durée en heures：

:

		min	mean	max
o_city	d_city			
5	23	8.883333	10.313333	15.900000
6	227	12.400000	13.708333	15.016667
	504	5.600000	8.290000	12.333333
	628	9.666667	12.166667	14.500000
	845	1.000000	1.331841	4.183333
...	...	...	...	...
11938	126	5.500000	7.421296	35.333333
12124	1064	11.166667	17.166667	23.166667
12166	857	21.916667	21.916667	21.916667
12190	639	1.166667	1.481481	2.666667
	8937	9.166667	9.166667	9.166667

1437 rows × 3 columns

2. La différence de prix et de temps s'explique par des distances et des modes de transport différents.

2.1 Distance en ligne droite

Tout d'abord, nous calculons la distance en ligne droite entre deux villes en nous basant sur la ville de départ (o_city) et la ville d'arrivée (d_city) du fichier ticket_data.csv, ainsi que sur les informations relatives à la latitude et à la longitude du fichier cities.csv.

Étant donné que les distances dans le tableau sont toutes inférieures à 2000 km, nous diviserons les données en trois groupes : 0-200 km, 201-800 km et 800-2000 km.

```
ticket_draft_0_200 = ticket_draft[ticket_draft['distance']<= 200]
ticket_draft_0_200.distance.agg(['min','max'])
```

```
min     18.962318
max    199.085694
Name: distance, dtype: float64
```

```
ticket_draft_201_800 = ticket_draft[(ticket_draft['distance']<= 800) & (ticket_draft['distance']> 200
ticket_draft_201_800.distance.agg(['min','max'])
```

```
min    200.566830
max    798.544558
Name: distance, dtype: float64
```

```
ticket_draft_801_2000 = ticket_draft[(ticket_draft['distance']<= 2000) & (ticket_draft['distance']> 8
ticket_draft_801_2000.distance.agg(['min','max'])
```

```
min     803.402293
max    1875.174971
Name: distance, dtype: float64
```

2.2 Définition des fonctions

Définir deux fonctions (prixAVG) et (tempsAVG) pour calculer la moyenne du prix et du temps pour différentes catégories sur la base de différentes distances.

```
def priceAVG_timeAVG (df):
    priceAVG = df.groupby('transport_type').price_in_cents.agg('mean').apply(lambda x: np.round(x, 2)
    timeAVG = df.groupby('transport_type').apply(lambda x: (x['arrival_ts'] - x['departure_ts']).agg(

    return priceAVG , timeAVG
```

```
priceAVG200 , timeAVG200 = priceAVG_timeAVG(ticket_draft_0_200)
```

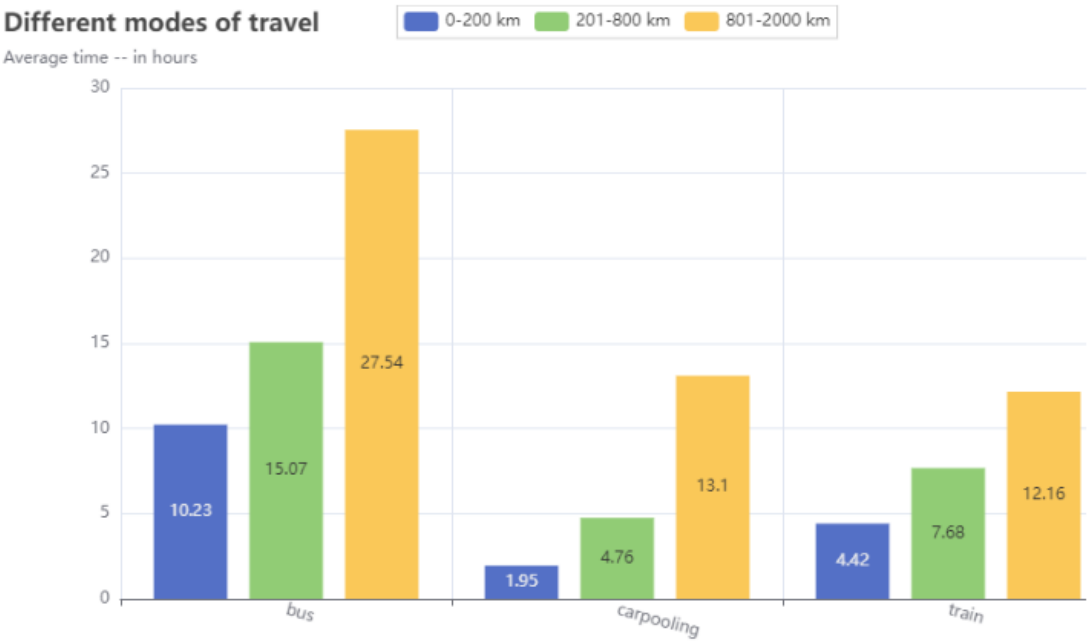
```
priceAVG800 , timeAVG800 = priceAVG_timeAVG(ticket_draft_201_800)
```

```
priceAVG2000 , timeAVG2000 = priceAVG_timeAVG(ticket_draft_801_2000)
```

```
print(priceAVG200 , timeAVG200 ,priceAVG800 , timeAVG800,priceAVG2000 , timeAVG2000)
```

```
transport_type
bus           2182.83
carpooling    1177.35
train         3487.90
Name: price_in_cents, dtype: float64 transport_type
bus           10.23
carpooling     1.95
train          4.42
dtype: float64 transport_type
bus           3494.85
carpooling    3218.06
train         9124.78
Name: price_in_cents, dtype: float64 transport_type
bus           15.07
carpooling     4.76
train          7.68
dtype: float64 transport_type
bus            6949.95
carpooling     8626.55
train         15463.75
Name: price_in_cents, dtype: float64 transport_type
bus           27.54
carpooling    13.10
train         12.16
dtype: float64
```

3. Présentation de "pyecharts" pour la visualisation de données



4. Faire des prévisions de prix

4.1 Utiliser dummie

Tout d'abord, nous utiliserons des variables "dummy" pour catégoriser le transport_type. Cela est nécessaire pour prédire les prix à l'avenir. Bien qu'il n'y ait que trois types de données dans ce cas, les remplacer serait possible, mais l'utilisation de variables "dummy" serait plus appropriée au cas où il y aurait plusieurs types de données à l'avenir.

4.2 Tout d'abord, une simple prédiction

La modification de "new_distance" ou "new_time" donnera des résultats différents.

```
import pandas as pd
from sklearn.linear_model import LinearRegression

# Créer un modèle de régression linéaire
regressor = LinearRegression()


X.columns = ['Feature 1', 'Feature 2', 'Feature 3', 'Feature 4', 'Feature 5']
X = dummied_df[['distance', 'time','bus','carpooling','train']]


y = dummied_df['price_in_cents']

# Entraîner le modèle de régression linéaire
regressor.fit(X, y)

new_distance = 500
new_time = 4


new_price = regressor.predict([[new_distance, new_time,0,0,1]])


new_price
```

```
D:\Anaconda\lib\site-packages\sklearn\base.py:450: UserWarning: X does not have valid feature names,
but LinearRegression was fitted with feature names
  warnings.warn(
```

```
array([9429.78429718])
```

4.3 Régression linéaire

Ensuite, un modèle simple d'apprentissage automatique est utilisé, le modèle est entraîné et les résultats sont obtenus :

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression


Forecast_prices = dummied_df


X_train, X_test, y_train, y_test = train_test_split(
    Forecast_prices[['distance', 'time','bus','carpooling','train']], Forecast_prices['price_in_cents']


model = LogisticRegression()
model.fit(X_train, y_train)


score = model.score(X_test, y_test)
print('Accuracy: {:.2f}%'.format(score * 100))
```

```
D:\Anaconda\lib\site-packages\sklearn\linear_model\_logistic.py:814: ConvergenceWarning: lbfgs faile
d to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
Accuracy: 3.93%
```

4.4 Apprentissage profond--MLP

J'ai choisi d'utiliser l'apprentissage profond car les résultats du modèle d'apprentissage automatique n'étaient pas aussi bons que je l'aurais souhaité. Cependant, en raison du volume de données, mon ordinateur n'a jamais exécuté les résultats, mais je suis sûr qu'il n'y a rien de mal dans mes idées et mon code, alors s'il vous plaît, essayez de trouver des résultats si vous le pouvez.

```
from  sklearn.neural_network  import  MLPClassifier
mlp = MLPClassifier(hidden_layer_sizes =(30 ,30 ,30),max_iter=300)

X_train, X_test, y_train, y_test = train_test_split(
    Forecast_prices[['distance', 'time','bus','carpooling','train']], Forecast_prices['price_in_cents']
history = mlp.fit(X_train, y_train)
```

```
score = model.score(X_test, y_test)
print('Accuracy: {:.2f}%'.format(score * 100))
```

```
y_pred = mlp.predict(X_test)

from  sklearn.metrics import  precision_score,recall_score,f1_score
print(classification_report(y_test,y_pred))
# OU SI VOUS PREFEREZ
print("precision = ",precision_score(y_test ,  y_pred))
print("rappel = ",recall_score(y_test ,  y_pred))
print("f1 = ",f1_score(y_test ,  y_pred))
```

Questions:

1. J'avais l'intention d'utiliser databricks pour la prédiction des prix, mais pour une raison ou une autre le site continue à être buggé, j'espère pouvoir l'essayer à l'avenir.
2. La colonne 'id' dans le tableau providers.csv est égale à la colonne 'company' dans ticket_data.csv, donc que signifie 'company_id' ?