Prédiction de la durée d'un trajet en taxi

Ce jeu de données est issu d'une compétition sur kaggle. Les données ont été au préalable échantillonnées et nettoyées pour les besoins de cette compétition et l'objectif était prédire la durée de chaque trajet sur la base des attributs individuels des trajets. On y retrouve les colonnes suivantes:

- id : identifiant du trajet
- vendor_id : code indiquant le prestataire associé à l'enregistrement du voyage
- pickup_datetime : la date et l'heure auxquelles le compteur a été enclenché
- dropoff_datetime : la date et l'heure auxquelles le compteur a été arrêté
- passenger_count : nombre de passagers lors du trajet
- pickup_longitude : longitude à laquelle le compteur a été déclenché
- pickup_latitude : latitude à laquelle le compteur a été déclenché
- dropoff_longitude : longitude à laquelle le compteur a été arrêté
- dropoff_latitude : latitude à laquelle le compteur a été arrêté
- store_and_fwd_flag : Cet indicateur indique si les informations du trajet ont été transmis en temps réel au serveur ou non. Y=stockées en mémoire puis récupérées; N=envoie en temps réel
- trip_duration : durée du trajet (en secondes)

Analyse exploratoire

In [1]:
```python
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import datetime
import numpy as np
import math
%matplotlib inline
```

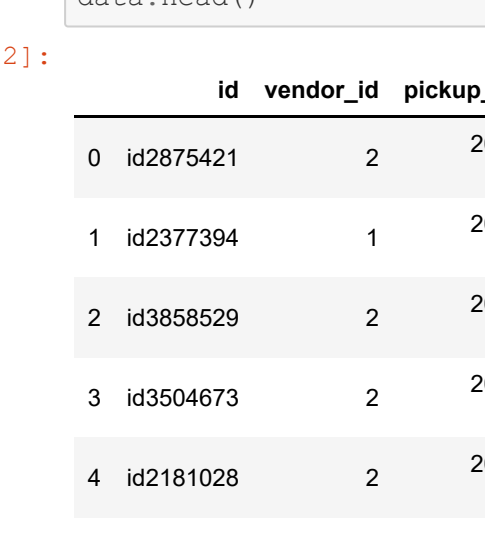
In [2]:
```python
data = pd.read_csv('train.csv')
data.head()
```

Out[2]:

	id	vendor_id	pickup_datetime	dropoff_datetime	passenger_count	pickup_longitude	pickup_latitude	dropoff_longitude	dropo
0	id2875421	2	2016-03-14 17:24:55	2016-03-14 17:32:30	1	-73.982155	40.767937	-73.964630	
1	id2377394	1	2016-06-12 00:43:35	2016-06-12 00:54:38	1	-73.980415	40.738564	-73.999481	
2	id3858529	2	2016-01-19 11:35:24	2016-01-19 12:10:48	1	-73.979027	40.763939	-74.005333	
3	id3504673	2	2016-04-06 19:32:31	2016-04-06 19:39:40	1	-74.010040	40.719971	-74.012268	
4	id2181028	2	2016-03-26 13:30:55	2016-03-26 13:38:10	1	-73.973053	40.793209	-73.972923	

In [3]:
```python
# On vérifie s'il n'y a pas de valeurs manquantes dans le jeu de données.
sns.heatmap(data.isnull(),yticklabels=False,cbar=False,cmap='viridis')
```

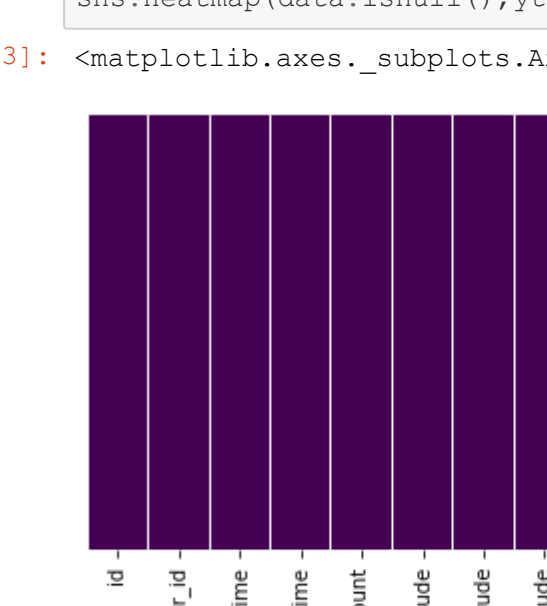
Out[3]: `<matplotlib.axes._subplots.AxesSubplot at 0x22404e952c8>`



Il n'y a pas de valeurs manquantes dans le jeu de données. Pour chaque trajet, toutes les informations ont été renseignées.

In [4]:
```python
# distribution de la durée de trajet
sns.distplot(data['trip_duration']/60)
```

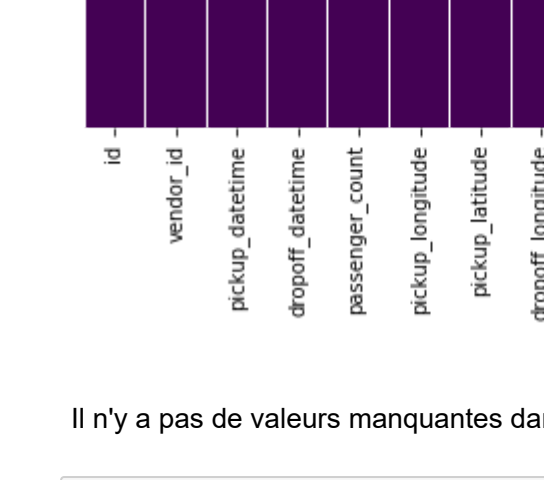
Out[4]: `<matplotlib.axes._subplots.AxesSubplot at 0x22404f53e88>`



On remarque que la plupart des trajets ont une durée comprise entre 0 et 5000s. Nous allons donc zoomer sur cette fenêtre.

In [5]:
```python
sns.distplot(data[data['trip_duration']/3600<2]['trip_duration'])
```

Out[5]: `<matplotlib.axes._subplots.AxesSubplot at 0x22403f2c488>`



In [6]:
```python
quantiles = np.quantile(data['trip_duration'], (0.05,0.1,0.5,0.9,0.95))
tab = pd.DataFrame(['5%','10%','50%','90%','95%']).T
tab = tab.append(pd.DataFrame(quantiles).T, ignore_index=True)
tab.columns = tab.loc[0,:]
tab.drop(0, axis=0, inplace=True)
print('Durée moyenne des trajets: ', data['trip_duration'].mean())
print('Ecart type: ', data['trip_duration'].std())
tab
```

```
Durée moyenne des trajets:  959.4922729603659
Ecart type:  5237.431724497702
```

Out[6]:

	5%	10%	50%	90%	95%
1	180	245	662	1634	2104

Nous avons tracé la distribution de la durée de trajet et la durée moyenne d'un trajet est de 959s (environ 16mn) avec un écart type de 5237s (environ 87mn). Nous avons aussi constaté la présence de trajets avec des durées très élevées qui nous semblent aberrantes. En effet, il y a par exemple des trajets de plus de 30 000s (environ 8h), ce qui est étonnant pour des trajets sur New York. Cependant, ils constituent un faible nombre puisque 95% des trajets du jeu de données font moins de 2104s (environ 35mn).

Feature engineering

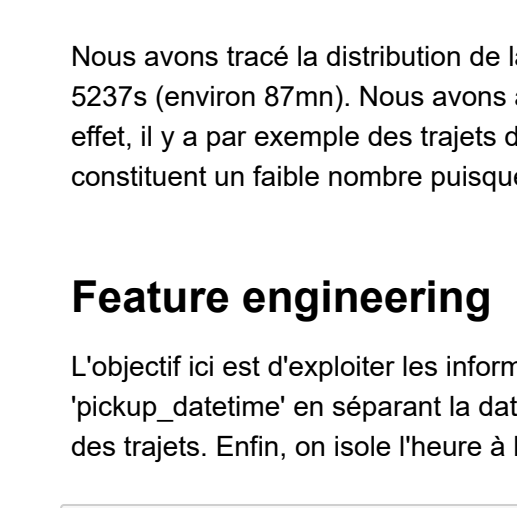
L'objectif ici est d'exploiter les informations sur les trajets pour en expliciter de nouvelles. On commence d'abord par la variable 'pickup_datetime' en séparant la date et le temps en deux variables distinctes. Ensuite, on récupère le jour de la semaine ainsi que le mois des trajets. Enfin, on isole l'heure à laquelle le compteur a été déclenché (début du trajet).

In [3]:
```python
# On isole la date et l'heure du trajet
data['pickup_date'] = data['pickup_datetime'].apply(lambda x : x.split()[0])
data['pickup_time'] = data['pickup_datetime'].apply(lambda x : x.split()[1])
data.head()
```

Out[3]:

	id	vendor_id	pickup_datetime	dropoff_datetime	passenger_count	pickup_longitude	pickup_latitude	dropoff_longitude	dropo
0	id2875421	2	2016-03-14 17:24:55	2016-03-14 17:32:30	1	-73.982155	40.767937	-73.964630	
1	id2377394	1	2016-06-12 00:43:35	2016-06-12 00:54:38	1	-73.980415	40.738564	-73.999481	
2	id3858529	2	2016-01-19 11:35:24	2016-01-19 12:10:48	1	-73.979027	40.763939	-74.005333	
3	id3504673	2	2016-04-06 19:32:31	2016-04-06 19:39:40	1	-74.010040	40.719971	-74.012268	
4	id2181028	2	2016-03-26 13:30:55	2016-03-26 13:38:10	1	-73.973053	40.793209	-73.972923	

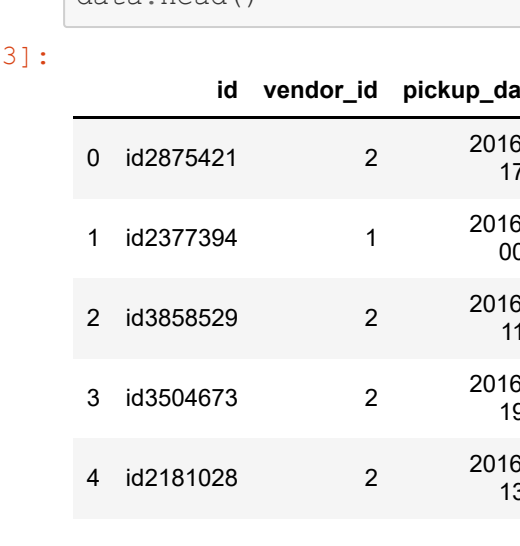
In [4]:
```python
# On isole le mois et on trace la distribution de la durée moyenne selon le mois
data['pickup_month'] = data['pickup_date'].apply(lambda x : x.split('-')[1])
result = data.groupby('pickup_month')['trip_duration'].aggregate(np.mean).reset_index()
#ax = sns.barplot(x='pickup_month',y='trip_duration',data=result,order=result['pickup_month'])
ax = sns.barplot(x='pickup_month',y='trip_duration',data=result)
ax.set(xlabel='mois', ylabel=r'durée moyenne', title=r'Durée moyenne du trajet selon le mois')
plt.show()
```



D'après ce graphique, on constate que selon le mois où le trajet est effectué, sa durée moyenne varie. Le mois a donc une influence sur la durée du trajet.

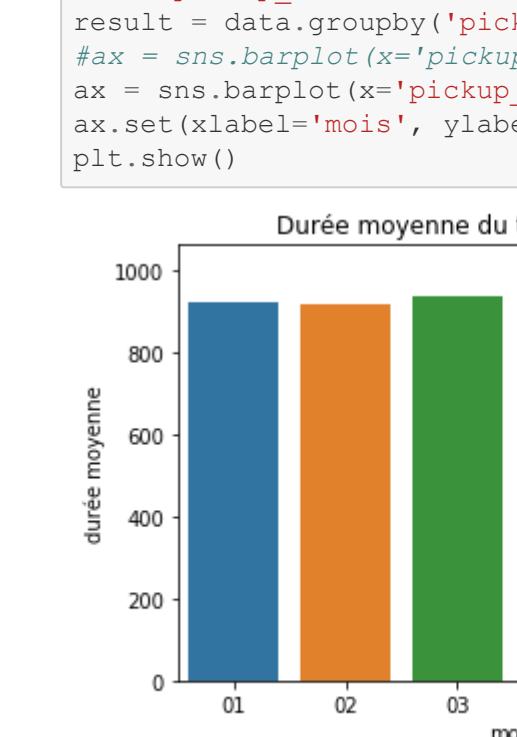
In [5]:
```python
# Récupération du jour de la semaine où le trajet a eu lieu
def get_day(x):
    correspondance = ['Mon','Tus','Wed','Thu','Fri','Sat','Sun']
    int_day = datetime.datetime(int(x.split('-')[0]), int(x.split('-')[1]), int(x.split('-')[2])).weekd
ay()
    return correspondance[int_day]
```

In [8]:
```python
data['pickup_day'] = data['pickup_date'].apply(get_day)
result = data.groupby(["pickup_day"])['trip_duration'].aggregate(np.mean).reset_index()
ax = sns.barplot(x='pickup_day',y='trip_duration',data=result, order=['Mon','Tus','Wed','Thu','Fri','Sa
t','Sun'])
ax.set(xlabel='jour', ylabel=r'durée moyenne', title=r'Durée moyenne du trajet selon le jour de la sema
ine')
plt.show()
```



D'après ce graphique, on remarque que le jour de la semaine a aussi une incidence sur la durée d'un trajet. En effet, le jeudi, le vendredi et le samedi sont les jours où les trajets en taxi sont susceptibles de durer plus longtemps. En revanche, le dimanche et le lundi sont les jours où les trajets en taxi sont susceptibles de durer moins longtemps.

In [9]:
```python
# On isole le moment de la journée (heure) où le trajet démarre
data['pickup_hour'] = data['pickup_time'].apply(lambda x : x.split(':')[0])
result = data.groupby(["pickup_hour"])['trip_duration'].aggregate(np.mean).reset_index()
ax = sns.barplot(x='pickup_hour',y='trip_duration',data=result)
ax.set(xlabel=r'heure de la journée', ylabel=r'durée moyenne', title=r"Durée moyenne du trajet selon
 l'heure de la journée")
plt.show()
```



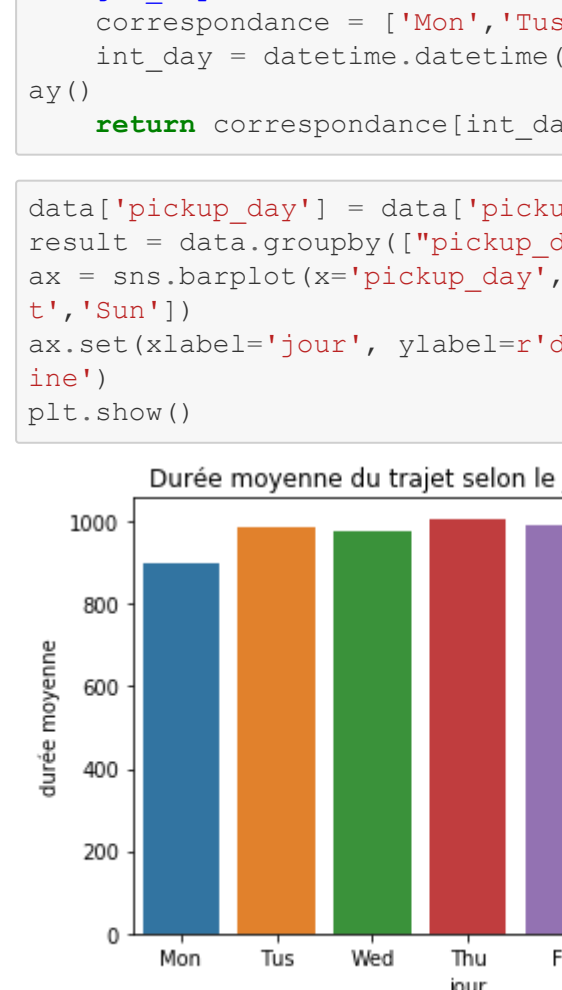
Selon le moment de la journée où on prend le taxi, la durée moyenne du trajet diffère. Elle est plus élevée par exemple les après-midis entre 12h et 18h.

In [11]:
```python
# On calcule la distance à vol d'oiseau entre le point de départ et la destination.
from math import sin, cos, sqrt, atan2, radians
def compute_distance(row):
    try:
        R = 6373.0
        lat1 = radians(row['dropoff_latitude'])
        lon1 = radians(row['dropoff_longitude'])
        lat2 = radians(row['pickup_latitude'])
        lon2 = radians(row['pickup_longitude'])
        dlat = lat1 - lat2
        dlon = lon1 - lon2
        a = sin(dlat / 2)**2 + cos(lat1) * cos(lat2) * sin(dlon / 2)**2
        c = 2 * atan2(sqrt(a), sqrt(1 - a))
        return R * c
    except KeyError:
        print(row)
```

In [12]:
```python
data['trip_dist'] = data.apply(compute_distance,axis=1)
```

In [13]:
```python
sns.jointplot(x='trip_dist',y='trip_duration',data=data,kind='scatter')
```

Out[13]: `<seaborn.axisgrid.JointGrid at 0x2813558f348>`



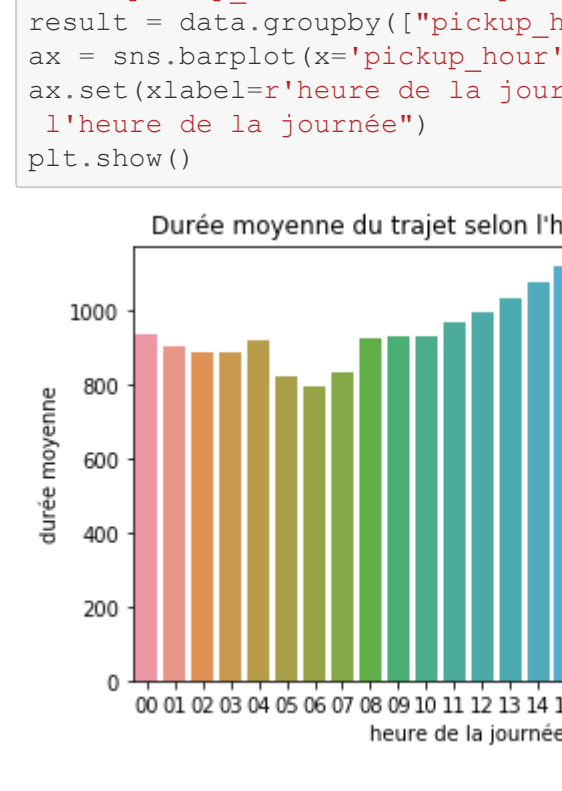
Ce graphique représente la durée de trajet par rapport à la distance. Comme on l'a remarqué plus haut avec la distribution de la durée des trajet, on a la confirmartion que certains d'entre eux étaient aberrants. En effet, on voit sur ce graphique que des trajets de moins de 100km avaient une durée de plus de 1 500 000s (environ 416h). De plus on remarque des trajets de 0km, ce qui est impossible. On va donc supprimer ces valeurs par la suite. Pour se faire, nous allons calculer la vitesse moyenne de ces trajets et on supprimera les trajets avec une vitesse moyenne égale à 0km/h ou supérieure à 130km/h. Le choix du seuil à 130km/h est dû au fait que c'est la vitesse maximale autorisée dans l'état de New York.

In [14]:
```python
# une fois la distance à vol d'oiseau calculée, on calcule la vitesse moyenne des trajets
data['spead'] = data['trip_dist']/(data['trip_duration']/3600)
data[(data['spead']<130) & (data['spead']!=0) & (data['trip_duration']/3600<3)].shape[0]/data.shape[0]*
100
```

Out[14]: `99.44263302080563`

In [15]:
```python
sns.jointplot(x='trip_dist',y='trip_duration',data=data[(data['spead']<130) & (data['spead']!=0) & (dat
a['trip_duration']/3600<3)],kind='scatter')
#ax.set(xlabel='distance', ylabel=r'durée du trajet', title=r'Durée du trajet en fonction de la distanc
e')
#plt.show()
```

Out[15]: `<seaborn.axisgrid.JointGrid at 0x2812e61e2c8>`



Nous avons tracé la durée de trajet par rapport à la distance pour les trajets de moins de 3h avec une distance supérieure à 0 et on remarque que nous n'avons pas une relation de linéarité stricte entre la distance et la durée d'un trajet. Le choix de retenir que les trajets de moins de 3h permet de supprimer les trajets aberrants mentionnés plus haut tout en conservant presque toute la donnée (99.4 %).

Régression Linéaire

In [17]:
```python
data[(data['trip_duration']/3600<3) & (data['spead']<130) & (data['spead']!=0)].shape[0]
```

Out[17]: `1450514`

In [18]:
```python
data_bis = data.copy()
data_bis = data_bis[(data_bis['spead']<130) & (data_bis['spead']!=0) & (data_bis['trip_duration']/3600<
6)]
data_bis.drop(['id','vendor_id','pickup_datetime','dropoff_datetime','pickup_date','pickup_time','spea
d'], axis=1, inplace=True)
data_bis.head()
```

Out[18]:

	passenger_count	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	store_and_fwd_flag	trip_duration	pickup_montl
0	1	-73.982155	40.767937	-73.964630	40.765602	N	455	0:
1	1	-73.980415	40.738564	-73.999481	40.731152	N	663	0(
2	1	-73.979027	40.763939	-74.005333	40.710087	N	2124	0
3	1	-74.010040	40.719971	-74.012268	40.706718	N	429	0‹
4	1	-73.973053	40.793209	-73.972923	40.782520	N	435	0:

In [19]:
```python
data_bis['pickup_month'] = data_bis['pickup_month'].astype("category")
data_bis['pickup_hour'] = data_bis['pickup_hour'].astype("category")
data_bis['pickup_day'] = data_bis['pickup_day'].astype("category")
data_bis['store_and_fwd_flag'] = data_bis['store_and_fwd_flag'].astype("category")
data_bis.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1450565 entries, 0 to 1458643
Data columns (total 11 columns):
passenger_count       1450565 non-null int64
pickup_longitude      1450565 non-null float64
pickup_latitude       1450565 non-null float64
dropoff_longitude     1450565 non-null float64
dropoff_latitude      1450565 non-null float64
store_and_fwd_flag    1450565 non-null category
trip_duration         1450565 non-null int64
pickup_month          1450565 non-null category
pickup_day            1450565 non-null category
pickup_hour           1450565 non-null category
trip_dist             1450565 non-null float64
dtypes: category(4), float64(5), int64(2)
memory usage: 94.1 MB
```

In [20]:
```python
data_bis = pd.get_dummies(data_bis)
data_bis.columns
```

Out[20]:
```
Index(['passenger_count', 'pickup_longitude', 'pickup_latitude',
       'dropoff_longitude', 'dropoff_latitude', 'trip_duration', 'trip_dist',
       'store_and_fwd_flag_N', 'store_and_fwd_flag_Y', 'pickup_month_01',
       'pickup_month_02', 'pickup_month_03', 'pickup_month_04',
       'pickup_month_05', 'pickup_month_06', 'pickup_day_Fri',
       'pickup_day_Mon', 'pickup_day_Sat', 'pickup_day_Sun', 'pickup_day_Thu',
       'pickup_day_Tus', 'pickup_day_Wed', 'pickup_hour_00', 'pickup_hour_01',
       'pickup_hour_02', 'pickup_hour_03', 'pickup_hour_04', 'pickup_hour_05',
       'pickup_hour_06', 'pickup_hour_07', 'pickup_hour_08', 'pickup_hour_09',
       'pickup_hour_10', 'pickup_hour_11', 'pickup_hour_12', 'pickup_hour_13',
       'pickup_hour_14', 'pickup_hour_15', 'pickup_hour_16', 'pickup_hour_17',
       'pickup_hour_18', 'pickup_hour_19', 'pickup_hour_20', 'pickup_hour_21',
       'pickup_hour_22', 'pickup_hour_23'],
      dtype='object')
```

In [21]:
```python
data_bis.drop('store_and_fwd_flag_Y', axis=1, inplace=True)
print(data_bis.columns)
data_bis.shape
```

```
Index(['passenger_count', 'pickup_longitude', 'pickup_latitude',
       'dropoff_longitude', 'dropoff_latitude', 'trip_duration', 'trip_dist',
       'store_and_fwd_flag_N', 'pickup_month_01', 'pickup_month_02',
       'pickup_month_03', 'pickup_month_04', 'pickup_month_05',
       'pickup_month_06', 'pickup_day_Fri', 'pickup_day_Mon', 'pickup_day_Sat',
       'pickup_day_Sun', 'pickup_day_Thu', 'pickup_day_Tus', 'pickup_day_Wed',
       'pickup_hour_00', 'pickup_hour_01', 'pickup_hour_02', 'pickup_hour_03',
       'pickup_hour_04', 'pickup_hour_05', 'pickup_hour_06', 'pickup_hour_07',
       'pickup_hour_08', 'pickup_hour_09', 'pickup_hour_10', 'pickup_hour_11',
       'pickup_hour_12', 'pickup_hour_13', 'pickup_hour_14', 'pickup_hour_15',
       'pickup_hour_16', 'pickup_hour_17', 'pickup_hour_18', 'pickup_hour_19',
       'pickup_hour_20', 'pickup_hour_21', 'pickup_hour_22', 'pickup_hour_23'],
      dtype='object')
```

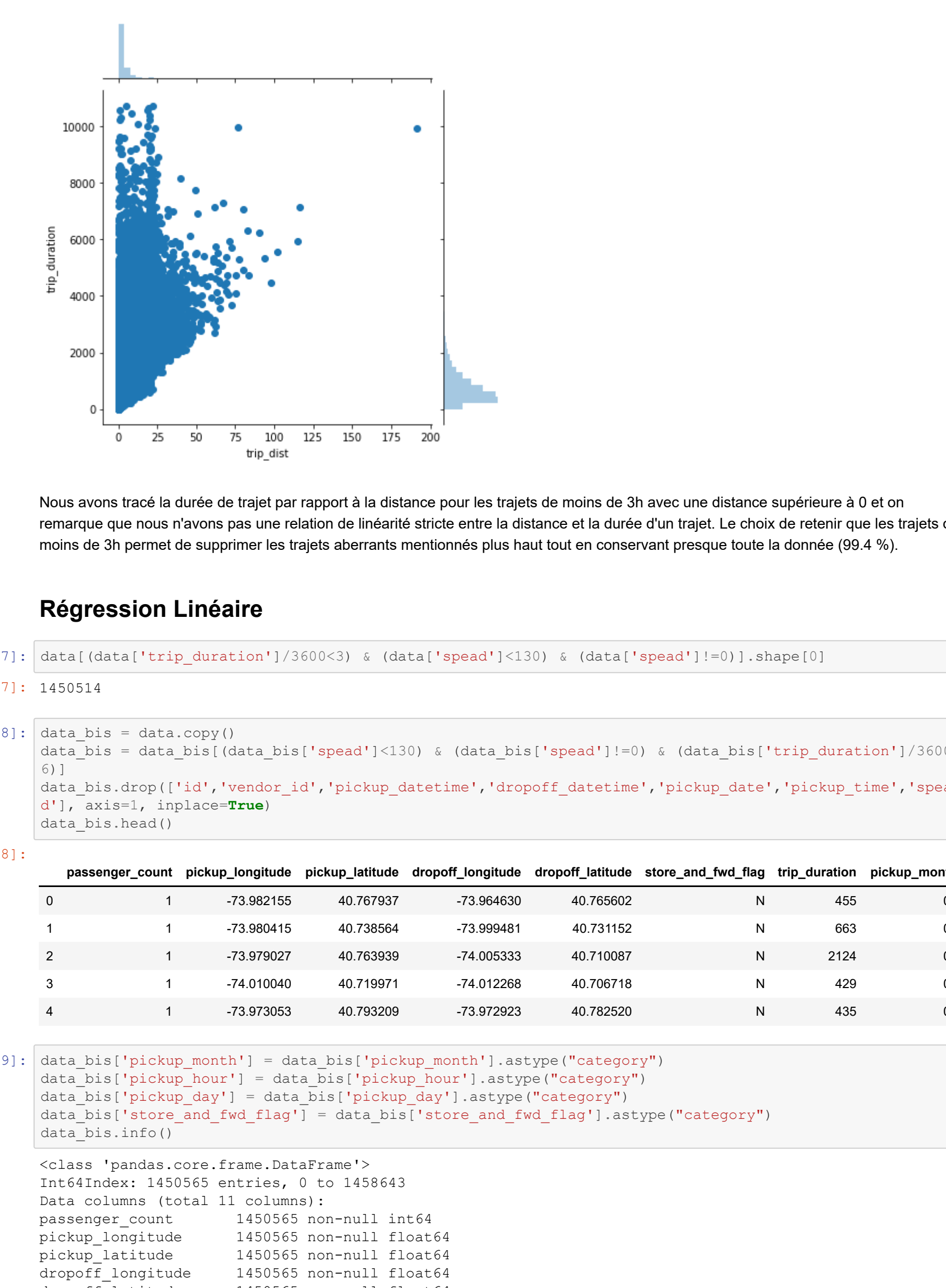
Out[21]: `(1450565, 45)`

In [22]:
```python
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    data_bis.drop('trip_duration',axis=1), data_bis['trip_duration'], test_size=0.2, random_state=101)
```

In [26]:
```python
from sklearn.linear_model import LinearRegression
lm = LinearRegression()
lm.fit(X_train,y_train)

plt.figure(figsize=(15,10))
plt.xticks(rotation=90)
plt.bar(X_train.columns, lm.coef_)
```

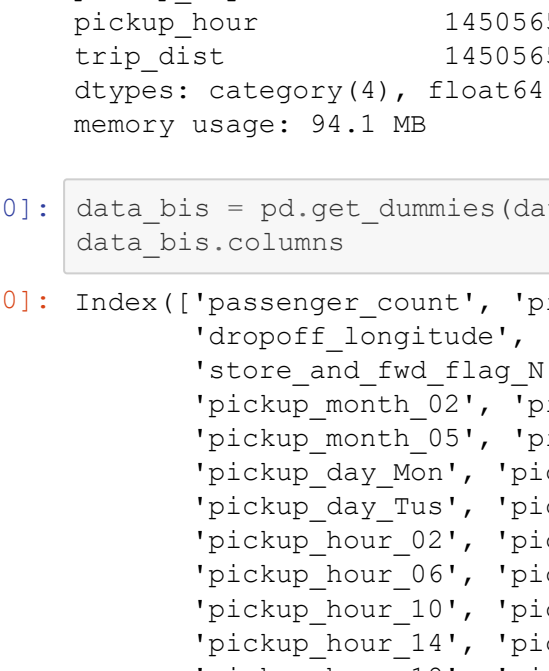
Out[26]: `<BarContainer object of 44 artists>`



In [27]:
```python
predictions = lm.predict(X_test)
```

In [28]:
```python
plt.scatter(y_test,predictions)
```

Out[28]: `<matplotlib.collections.PathCollection at 0x2810a537e08>`



In [ ]:
```python
from sklearn import metrics
print('MSE:', metrics.mean_squared_error(y_test, predictions))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, predictions))/60)
```

L'écart moyen de prédiction est égal à 6.71mn pour ce modèle.

Random Forest

In [17]:
```python
data_bis = data.copy()
data_bis = data_bis[(data_bis['spead']<130) & (data_bis['spead']!=0) & (data_bis['trip_duration']/3600<
6)]

def get_day(x):
    return datetime.datetime(int(x.split('-')[0]), int(x.split('-')[1]), int(x.split('-')[2])).weekday
()

data_bis['pickup_day'] = data_bis['pickup_date'].apply(get_day)
data_bis.drop(['id','vendor_id','pickup_datetime','dropoff_datetime','pickup_date','pickup_time','spea
d'], axis=1, inplace=True)
data_bis.head()
```

Out[17]:

	passenger_count	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	store_and_fwd_flag	trip_duration	pickup_montl
0	1	-73.982155	40.767937	-73.964630	40.765602	N	455	0:
1	1	-73.980415	40.738564	-73.999481	40.731152	N	663	0(
2	1	-73.979027	40.763939	-74.005333	40.710087	N	2124	0
3	1	-74.010040	40.719971	-74.012268	40.706718	N	429	0‹
4	1	-73.973053	40.793209	-73.972923	40.782520	N	435	0:

In [18]:
```python
data_bis['pickup_month'] = data_bis['pickup_month'].astype("category")
data_bis['pickup_hour'] = data_bis['pickup_hour'].astype("category")
data_bis['pickup_day'] = data_bis['pickup_day'].astype("category")
data_bis['store_and_fwd_flag'] = data_bis['store_and_fwd_flag'].astype("category")
data_bis['store_and_fwd_flag'] = pd.get_dummies(data_bis['store_and_fwd_flag'],drop_first=True)
data_bis.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1450565 entries, 0 to 1458643
Data columns (total 11 columns):
passenger_count       1450565 non-null int64
pickup_longitude      1450565 non-null float64
pickup_latitude       1450565 non-null float64
dropoff_longitude     1450565 non-null float64
dropoff_latitude      1450565 non-null float64
store_and_fwd_flag    1450565 non-null uint8
trip_duration         1450565 non-null int64
pickup_month          1450565 non-null category
pickup_day            1450565 non-null category
pickup_hour           1450565 non-null category
trip_dist             1450565 non-null float64
dtypes: category(3), float64(5), int64(2), uint8(1)
memory usage: 94.1 MB
```

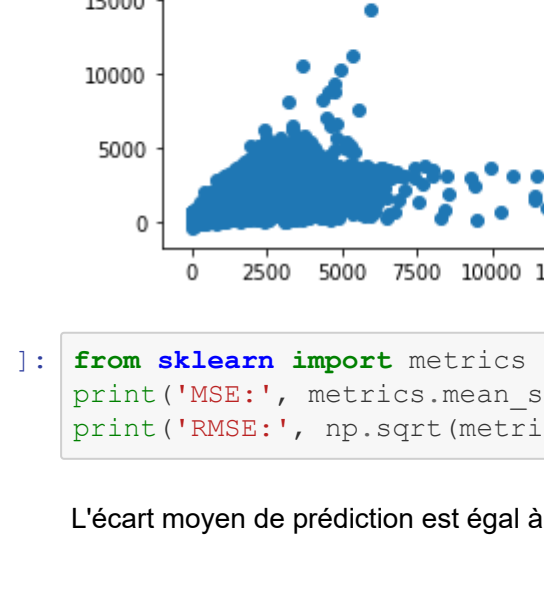
In [19]:
```python
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    data_bis.drop('trip_duration',axis=1), data_bis['trip_duration'], test_size=0.2, random_state=101)
```

In [20]:
```python
from sklearn.ensemble import RandomForestRegressor
rfc = RandomForestRegressor(n_estimators=100)
rfc.fit(X_train, y_train)
```

Out[20]: `RandomForestRegressor()`

In [23]:
```python
y_pred = rfc.predict(X_test)
plt.scatter(y_test,y_pred)
```

Out[23]: `<matplotlib.collections.PathCollection at 0x23323a34688>`



In [24]:
```python
from sklearn import metrics
print('MSE:', metrics.mean_squared_error(y_test, y_pred))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, y_pred))/60)
```

```
MSE: 90637.30564498388
RMSE: 5.0176717056647755
```

Par rapport à la régression linéaire, on obtient un meilleur résultat avec un écart moyen de prédiction égal à 5.01mn. Cependant dans les deux cas, on voit que la prédiction en fonction de la valeur réelle ne suit pas une distribution parfaitement linéaire. Cela nous suggére qu'on peut encore gagner en précision avec d'autres modèles. On ne le fera pas dans ce notebook car l'objectif étant seulement d'illustrer une démarche.

In [ ]:
```python
# Les variables importantes dans le modèle random forest
from sklearn.inspection import permutation_importance
import matplotlib.pyplot as plt

result = permutation_importance(rfc, X_test, y_test, n_repeats=10,
                                random_state=42, n_jobs=-1)
sorted_idx = result.importances_mean.argsort()

fig, ax = plt.subplots()
ax.boxplot(result.importances[sorted_idx].T,
           vert=False, labels=X_test.columns[sorted_idx])
ax.set_title("Permutation Importances (test set)")
fig.tight_layout()
plt.show()
```

```
C:\Users\mouha\Anaconda3\lib\site-packages\joblib\externals\loky\process_executor.py:706: UserWarnin
g: A worker stopped while some jobs were given to the executor. This can be caused by a too short wor
ker timeout or by a memory leak.
  "timeout or by a memory leak.", UserWarning
```

Ici, l'objectif était de déterminer les variables influant grandement sur la prédiction de la durée du trajet. Pour cela, on utilise l'approche appelée Permutation feature importance qui consiste à mesurer directement l'importance des variables en observant comment le remaniement aléatoire (préservant ainsi la distribution de la variable) de chaque variable influence la performance du modèle. C'est une approche qui est gourmande en terme de calcul. Or, mon ordinateur n'est pas assez puissant.