Machine Learning Use cases TD1

In [ ]:
```
!sudo apt-get install build-essential swig
!pip install auto-sklearn==0.14.7
```

```
Building wheels for collected packages: auto-sklearn, pynisher, smac, liac-arff
  Building wheel for auto-sklearn (setup.py) ... done
  Created wheel for auto-sklearn: filename=auto_sklearn-0.14.7-py3-none-any.whl si
ze=6602873 sha256=4c0aa4814081c6105365eca41067e64cb00b0bda815e9e2e872a6ef54e18930f
  Stored in directory: /root/.cache/pip/wheels/ba/43/5c/2fbe6fd19e3af314cbc4aa8083
78068d8ddd6792064f4a2448
  Building wheel for pynisher (setup.py) ... done
  Created wheel for pynisher: filename=pynisher-0.6.4-py3-none-any.whl size=7043 s
ha256=df65e6d9891cee79ddd8fab3825f2076ea0c0ea390831275850b927f110cb3ef
  Stored in directory: /root/.cache/pip/wheels/42/71/95/7555ec3253e1ba8add72ae5feb
f1b015d297f3b73ba296d6f6
  Building wheel for smac (setup.py) ... done
  Created wheel for smac: filename=smac-1.2-py3-none-any.whl size=215933 sha256=1f
5352f65223bdfc0f1f05aa04a691701bcbe6e45faad6df0b52f387774e4858
  Stored in directory: /root/.cache/pip/wheels/ad/95/67/6afc6b04d3715070c853d0a9d7
c7b1fb822def38671dfbbb9f
  Building wheel for liac-arff (setup.py) ... done
  Created wheel for liac-arff: filename=liac_arff-2.5.0-py3-none-any.whl size=1173
2 sha256=861dc23422217022d252a506fff90d5d26e43c2c5f14d5537196154259a1995a
  Stored in directory: /root/.cache/pip/wheels/1f/0f/15/332ca86cbebf25ddf98518caaf
887945fbe1712b97a0f2493b
Successfully built auto-sklearn pynisher smac liac-arff
Installing collected packages: scikit-learn, pyrfr, pynisher, emcee, ConfigSpace,
smac, liac-arff, distro, auto-sklearn
  Attempting uninstall: scikit-learn
    Found existing installation: scikit-learn 1.0.2
    Uninstalling scikit-learn-1.0.2:
      Successfully uninstalled scikit-learn-1.0.2
ERROR: pip's dependency resolver does not currently take into account all the pack
ages that are installed. This behaviour is the source of the following dependency
 conflicts.
yellowbrick 1.5 requires scikit-learn>=1.0.0, but you have scikit-learn 0.24.2 whi
ch is incompatible.
Successfully installed ConfigSpace-0.4.21 auto-sklearn-0.14.7 distro-1.7.0 emcee-
3.1.3 liac-arff-2.5.0 pynisher-0.6.4 pyrfr-0.8.3 scikit-learn-0.24.2 smac-1.2
```

In [20]:

```
#Exécuter la cellule une seconde fois en cas de message d'erreur pour le module au
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib
import pickle
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.metrics import confusion_matrix
from sklearn import preprocessing

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras import regularizers

#import autokeras as ak

from tensorflow.keras.models import save_model
from tensorflow.keras.models import load_model
from tensorflow.keras.optimizers import SGD


#import autosklearn
#import autosklearn.regression


from matplotlib import gridspec
```

```
from matplotlib import rcParams
#Paramètres graphiques
rcParams['figure.figsize'] = 15,8 #Taille de la figure affichée
sns.set_style("darkgrid")#style de l'arrière plan de seaborn
sns.set_palette("pastel")#Couleurs utilisées dans les graphiques
matplotlib.rcParams["figure.dpi"] = 300 #dpi = dot per inch , résolution des graph
pd.set_option('display.max_columns', 500) #Nombre de colonne maximum du dataframe

import sklearn.datasets
import sklearn.metrics
from sklearn.model_selection import train_test_split

import warnings
warnings.simplefilter(action='ignore')
```

In [ ]:

```
#fonctionne avec sklearn <1.0
def error_analysis(model, X_test, y_test):

    results=X_test.copy(deep=True).reset_index()
    results["pred"]=model.predict(X_test)
    results.loc[results["pred"]<0,"pred"]=0
    results["round_pred"]=results["pred"].round()
    results["true_value"]=y_test.tolist()
    results["Erreur"]=results["pred"]-results["true_value"]
    results["Erreur absolue"]=abs(results["Erreur"])
    results["true_value_5j"]=results["true_value"]
    results.loc[results["true_value_5j"]>=5,"true_value_5j"]=5

    results["pred_5j"]=results["round_pred"]
    results.loc[results["pred_5j"]>=5,"pred_5j"]=5

    results["round_down_pred"]=np.floor(results["pred_5j"])
    results["round_up_pred"]=results["round_down_pred"]+1

    #MAE
    print("Erreur moyenne absolue du modèle : " + str(round(results["Erreur absolu

    #Accuracy une classe par jour
    #On arrondi les résultats pour voir le nombre de prédictions correctes
    import sklearn
    acc = sklearn.metrics.accuracy_score(results["true_value"],results["round_pred
    print("Accuracy 1 : " + str(round(acc,4)))


    #Accuracy avec la classe +5 jours
    acc = sklearn.metrics.accuracy_score(results["true_value_5j"],results["pred_5j
    print("Accuracy 2 : " + str(round(acc,4)))

    #Accuracy avec des classes de 2 jours :
    acc = results[(results["true_value_5j"]<=results["round_up_pred"]) & (results[
    print("Accuracy 3 : "+str(round(acc,4)))



    print("`\n\n\n")
    #subplots avec une ligne pour les données globales et une ligne par provider :
    providers=results.groupby("provider").agg("count").sort_values("true_value",as

    Rows=len(providers)+1
    cols=4

    rows_labels=["Global"] + providers
```

```
    width_ratios= [4,3]+[2 for i in range(cols-2)]
    heigth_ratios =[2 for i in range(Rows)]

    #fig=plt.figure(1,figsize=(15,5*Rows))
    fig,axes= plt.subplots(Rows,cols ,figsize=(15,5*Rows), gridspec_kw={'width_rat
                                                                        'height_ra
                                                                        'wspace': 
                                                                        'hspace': 
    Position = [i for i in range(1, 5*(len(providers)+1))]

    #Données globales du modèle :
    #Courbe mae
    ax = axes[0,0]
    temp=results.groupby("date_commande").agg("mean").reset_index().sort_values("da
    sns.lineplot(data=temp, x="date_commande",y="Erreur absolue",ax=ax)
    ax.fill_between(temp["date_commande"],temp["Erreur absolue"],alpha=0.2)
    ax.set_xticklabels(labels=[] ,rotation = 90)
    ax.set_title("MAE par date de commande")

    #Matrice de confusion :
    ax = axes[0,1]
    array =confusion_matrix(results["true_value_5j"], results["pred_5j"],normalize=

    df_cm = pd.DataFrame(array, range(len(array[0])), range(len(array[0])))
    disp=sns.heatmap(df_cm,
                     annot=True,
                     annot_kws={"size": 7,"color":"white"},
                     cmap=plt.cm.Blues,
                     fmt=".2f",
                     cbar=False,
                     ax=ax
                     )
    disp.set_xlabel("Valeurs prédites")
    disp.set_ylabel("Valeurs réelles")
    disp.set_title("Matrice de confusion normalisée\nAccuracy 2\n",fontsize=8)



    acc1 = sklearn.metrics.accuracy_score(results["true_value"],results["round_pred
    acc2 = sklearn.metrics.accuracy_score(results["true_value_5j"],results["pred_5j
    acc3 = results[(results["true_value_5j"]<=results["round_up_pred"]) & (results

    df_pie=pd.DataFrame([[acc1,acc2,acc3],[1-acc1,1-acc2, 1-acc3]], columns=["Accu

    #Pie chart accuracy 2 :
    y="Accuracy 2"
    ax = axes[0,2]
    df_pie.plot.pie(y=y, autopct='%1.1f%%', ax=ax)
    ax.set_title(y+str("\n"))
    ax.get_legend().remove()
    ax.get_yaxis().set_visible(False)


    #Pie chart accuracy 3 :
    y="Accuracy 3"
    ax = axes[0,3]
    df_pie.plot.pie(y=y, autopct='%1.1f%%', ax=ax)
    ax.set_title(y+str("\n"))
    ax.get_legend().remove()
    ax.get_yaxis().set_visible(False)


    #Pour chaque provider :
    for i in range(1,len(providers)+1) :
```

```
        #Courbe mae
        results_provider= results[results["provider"]==providers[i-1]]
        ax = axes[i,0]
        temp=results_provider.groupby("date_commande").agg("mean").reset_index().s
        sns.lineplot(data=temp, x="date_commande",y="Erreur absolue",ax=ax)
        ax.fill_between(temp["date_commande"],temp["Erreur absolue"],alpha=0.2)
        ax.set_xticklabels(labels=[] ,rotation = 90)
        ax.set_title("MAE par date de commande")

        #Matrice de confusion :
        ax = axes[i,1]
        array =confusion_matrix(results_provider["true_value_5j"], results_provide

        df_cm = pd.DataFrame(array, range(len(array[0])), range(len(array[0])))
        disp=sns.heatmap(df_cm,
                         annot=True,
                         annot_kws={"size": 7,"color":"white"},
                         cmap=plt.cm.Blues,
                         fmt=".2f",
                         cbar=False,
                         ax=ax
                         )
        disp.set_xlabel("Valeurs prédites")
        disp.set_ylabel("Valeurs réelles")
        disp.set_title("Matrice de confusion normalisée\nAccuracy 2\n",fontsize=8)

        acc1 = sklearn.metrics.accuracy_score(results_provider["true_value"],result
        acc2 = sklearn.metrics.accuracy_score(results_provider["true_value_5j"],re
        acc3 = results_provider[(results_provider["true_value_5j"]<=results_provid

        df_pie=pd.DataFrame([[acc1,acc2,acc3],[1-acc1,1-acc2, 1-acc3]], columns=["

        #Pie chart accuracy 2 :
        y="Accuracy 2"
        ax = axes[i,2]
        df_pie.plot.pie(y=y, autopct='%1.1f%%', ax=ax)
        ax.set_title(y+str("\n"))
        ax.get_legend().remove()
        ax.get_yaxis().set_visible(False)


        #Pie chart accuracy 3 :
        y="Accuracy 3"
        ax = axes[i,3]
        df_pie.plot.pie(y=y, autopct='%1.1f%%', ax=ax)
        ax.set_title(y+str("\n"))
        ax.get_legend().remove()
        ax.get_yaxis().set_visible(False)




    for ax, row in zip(axes[:,0], rows_labels):
        ax.set_ylabel(str(row)+"    ", rotation=0, fontsize=20)


    fig.tight_layout()
    fig.show()
```

In [ ]:
```
def check_nan(df):
    for i in df.columns.tolist():
        print("Valeurs nan dans "+str(i)+" : "+str(df[i].isna().sum()))

#Pour sklearn 1.0
def error_analysis_sklearn10(model, X_test, y_test):

    results=X_test.copy(deep=True).reset_index()
    results["pred"]=model.predict(X_test)
    results.loc[results["pred"]<0,"pred"]=0
    results["round_pred"]=results["pred"].round()
    results["true_value"]=y_test.tolist()
    results["Erreur"]=results["pred"]-results["true_value"]
    results["Erreur absolue"]=abs(results["Erreur"])
    results["true_value_5j"]=results["true_value"]
    results.loc[results["true_value_5j"]>=5,"true_value_5j"]=5

    results["pred_5j"]=results["round_pred"]
    results.loc[results["pred_5j"]>=5,"pred_5j"]=5

    results["round_down_pred"]=np.floor(results["pred_5j"])
    results["round_up_pred"]=results["round_down_pred"]+1

    #MAE
    print("Erreur moyenne absolue du modèle : " + str(round(results["Erreur absolue

    #Accuracy une classe par jour
    #On arrondi les résultats pour voir le nombre de prédictions correctes
    import sklearn
    acc = sklearn.metrics.accuracy_score(results["true_value"],results["round_pred
    print("Accuracy 1 : " + str(round(acc,4)))


    #Accuracy avec la classe +5 jours
    acc = sklearn.metrics.accuracy_score(results["true_value_5j"],results["pred_5j
    print("Accuracy 2 : " + str(round(acc,4)))

    #Accuracy avec des classes de 2 jours :
    acc = results[(results["true_value_5j"]<=results["round_up_pred"]) & (results["
    print("Accuracy 3 : "+str(round(acc,4)))



    print("`\n\n\n")
    #subplots avec une ligne pour les données globales et une ligne par provider :
    providers=results.groupby("provider").agg("count").sort_values("true_value",as

    Rows=len(providers)+1
    cols=4

    rows_labels=["Global"] + providers

    width_ratios= [2]+[1 for i in range(cols-1)]
    heigth_ratios =[1 for i in range(Rows)]

    #fig=plt.figure(1,figsize=(15,5*Rows))
    fig,axes= plt.subplots(Rows,cols ,figsize=(15,5*Rows), gridspec_kw={'width_rat
                                                                        'height_ra
                                                                        'wspace': 
                                                                        'hspace': 

    Position = [i for i in range(1, 5*(len(providers)+1))]

    #Données globales du modèle :
    #Courbe mae
```

```
    ax = axes[0,0]
    temp=results.groupby("date_commande").agg("mean").reset_index().sort_values("da
    sns.lineplot(data=temp, x="date_commande",y="Erreur absolue",ax=ax)
    ax.fill_between(temp["date_commande"],temp["Erreur absolue"],alpha=0.2)
    ax.set_xticklabels(labels=[] ,rotation = 90)
    ax.set_title("MAE par date de commande")

    #Matrice de confusion :
    ax = axes[0,1]
    disp = ConfusionMatrixDisplay.from_predictions(
        results["true_value_5j"],results["pred_5j"],

        display_labels=results["true_value_5j"].drop_duplicates().sort_values().to
        cmap=plt.cm.Blues,
        normalize="true",
        colorbar=False,
        ax=ax,
        include_values=False,
    )
    disp.ax_.set_title("Matrice de confusion normalisée\nAccuracy 2\n")



    acc1 = sklearn.metrics.accuracy_score(results["true_value"],results["round_pred
    acc2 = sklearn.metrics.accuracy_score(results["true_value_5j"],results["pred_5j
    acc3 = results[(results["true_value_5j"]<=results["round_up_pred"]) & (results

    df_pie=pd.DataFrame([[acc1,acc2,acc3],[1-acc1,1-acc2, 1-acc3]], columns=["Accur

    #Pie chart accuracy 2 :
    y="Accuracy 2"
    ax = axes[0,2]
    df_pie.plot.pie(y=y, autopct='%1.1f%%', ax=ax)
    ax.set_title(y+str("\n"))
    ax.get_legend().remove()
    ax.get_yaxis().set_visible(False)


    #Pie chart accuracy 3 :
    y="Accuracy 3"
    ax = axes[0,3]
    df_pie.plot.pie(y=y, autopct='%1.1f%%', ax=ax)
    ax.set_title(y+str("\n"))
    ax.get_legend().remove()
    ax.get_yaxis().set_visible(False)


    #Pour chaque provider :
    for i in range(1,len(providers)+1) :
        #Courbe mae
        results_provider= results[results["provider"]==providers[i-1]]
        ax = axes[i,0]
        temp=results_provider.groupby("date_commande").agg("mean").reset_index().so
        sns.lineplot(data=temp, x="date_commande",y="Erreur absolue",ax=ax)
        ax.fill_between(temp["date_commande"],temp["Erreur absolue"],alpha=0.2)
        ax.set_xticklabels(labels=[] ,rotation = 90)
        ax.set_title("MAE par date de commande")

        #Matrice de confusion :
        ax = axes[i,1]
        disp = ConfusionMatrixDisplay.from_predictions(
            results_provider["true_value_5j"],results_provider["pred_5j"],
```

```
            #display_labels=[i in range(0,6)],
            cmap=plt.cm.Blues,
            normalize="true",
            colorbar=False,
            ax=ax,
            include_values=False,
        )
        disp.ax_.set_title("Matrice de confusion normalisée\nAccuracy 2\n")

        acc1 = sklearn.metrics.accuracy_score(results_provider["true_value"],result
        acc2 = sklearn.metrics.accuracy_score(results_provider["true_value_5j"],res
        acc3 = results_provider[(results_provider["true_value_5j"]<=results_provide

        df_pie=pd.DataFrame([[acc1,acc2,acc3],[1-acc1,1-acc2, 1-acc3]], columns=["A

        #Pie chart accuracy 2 :
        y="Accuracy 2"
        ax = axes[i,2]
        df_pie.plot.pie(y=y, autopct='%1.1f%%', ax=ax)
        ax.set_title(y+str("\n"))
        ax.get_legend().remove()
        ax.get_yaxis().set_visible(False)


        #Pie chart accuracy 3 :
        y="Accuracy 3"
        ax = axes[i,3]
        df_pie.plot.pie(y=y, autopct='%1.1f%%', ax=ax)
        ax.set_title(y+str("\n"))
        ax.get_legend().remove()
        ax.get_yaxis().set_visible(False)







    for ax, row in zip(axes[:,0], rows_labels):
        ax.set_ylabel(str(row)+"    ", rotation=0, fontsize=20)


    fig.tight_layout()
    fig.show()

# Create a grid : initialize it
def grid_var_date(var, results) :

    g = sns.FacetGrid(results.reset_index().groupby(["date_commande","provider"]).a

    # Add the line over the area with the plot function
    g.map(plt.plot, 'date_commande', var)


    # Fill the area with fill_between
    g.map(plt.fill_between, 'date_commande', var, alpha=0.2).set_titles("{col_name}

    # Control the title of each facet
    g.set_titles("{col_name}")

    for ax in g.axes.flat:
        labels = ax.get_xticklabels() # get x labels
        ax.set_xticklabels(labels, rotation=90) # set new labels
```

```
    # Add a title for the whole plot
    plt.subplots_adjust(top=0.85)
    g.fig.suptitle(str(var)+" moyenne quotidienne pour chaque provider\n",fontsize=

    # Show the graph
    plt.show()
```

Lecture des données

In [2]:

```
dataset=pd.read_csv("./data/logistique.csv",sep=";")

dataset["dateexpe"]=dataset["dateexpe"].astype(str)


dataset['split'] = dataset['dateexpe'].apply(lambda row : int(row.split('/')[0]))
dataset[(dataset['split'] == 11) | (dataset['split'] == 12)]
dataset['dateexpe'][(dataset['split'] == 11) | (dataset['split'] == 12)] = dataset
dataset = dataset.drop(columns=['split'])


df=dataset.copy(deep=True)
df.columns=["date_heure_commande","provider","date_expe"]
df=df.replace("2019","19", regex=True)

# new data frame with split value columns
list_date_heure = df["date_heure_commande"].str.split(" ", n = 1, expand = True)

# making separate first name column from new data frame
df["date_commande"]= list_date_heure[0]
df["horaire_commande"]=list_date_heure[1]

list_heure_min = df["horaire_commande"].str.split(":", n = 1, expand = True)
# making separate first name column from new data frame
df["heure_commande"]= list_heure_min[0]
df["min_commande"]=list_heure_min[1]
df.drop(columns=["horaire_commande"],inplace=True)

df["date_expe"]=df["date_expe"].str.split(" ", n = 1, expand = True)[0]

df["date_commande"]=pd.to_datetime(df["date_commande"], format="%d/%m/%y")

df["date_expe"]=pd.to_datetime(df["date_expe"], format="%d/%m/%y")

df["gap"]=(df["date_expe"]-df["date_commande"]).dt.days

df["jour_expe"]=df["date_expe"].dt.dayofweek
df["jour_commande"]=df["date_commande"].dt.dayofweek

df["id_date"]=df['date_commande'].dt.year.astype(int)*365*24*60+df['date_commande'
df["id_date"]=df["id_date"]//(60*24) # On arrondi au minuit inférieur le plus proch

#On considère que les colis sont expédiés à midi
df["id_date_expe"]=(df['date_expe'].dt.year.astype(int)*365*24*60+df['date_expe'].
df["id_date_expe"] = df["id_date_expe"]//(60*24)

df["date_expe"]=df["date_expe"].dt.date

df["provider"]=df["provider"].astype(str)


df.drop(columns=["date_heure_commande"],inplace=True)
```

```
df["heure_commande"]=df["heure_commande"].astype(int)
df["min_commande"]=df["min_commande"].astype(int)

#drop des lignes où la date d'expédition est avant la date de commande
df=df[df["gap"]>=0]

df.head()
```

```
C:\Users\qnava\AppData\Local\Temp\ipykernel_4240\2983625325.py:8: SettingWithCopyW
arning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  dataset['dateexpe'][(dataset['split'] == 11) | (dataset['split'] == 12)] = datas
et['dateexpe'][(dataset['split'] == 11) | (dataset['split'] == 12)].apply(lambda r
ow : '/'.join([row.split('/')[1], row.split('/')[0], row.split('/')[2]]))
```

Out[2]:

	provider	date_expe	date_commande	heure_commande	min_commande	gap	jour_expe	jou
0	48	2019-11-02	2019-11-01	11	40	1	5	
1	48	2019-11-02	2019-11-01	11	40	1	5	
2	48	2019-11-02	2019-11-01	10	24	1	5	
3	48	2019-11-02	2019-11-01	14	24	1	5	
4	48	2019-11-02	2019-11-01	13	28	1	5	

Observations

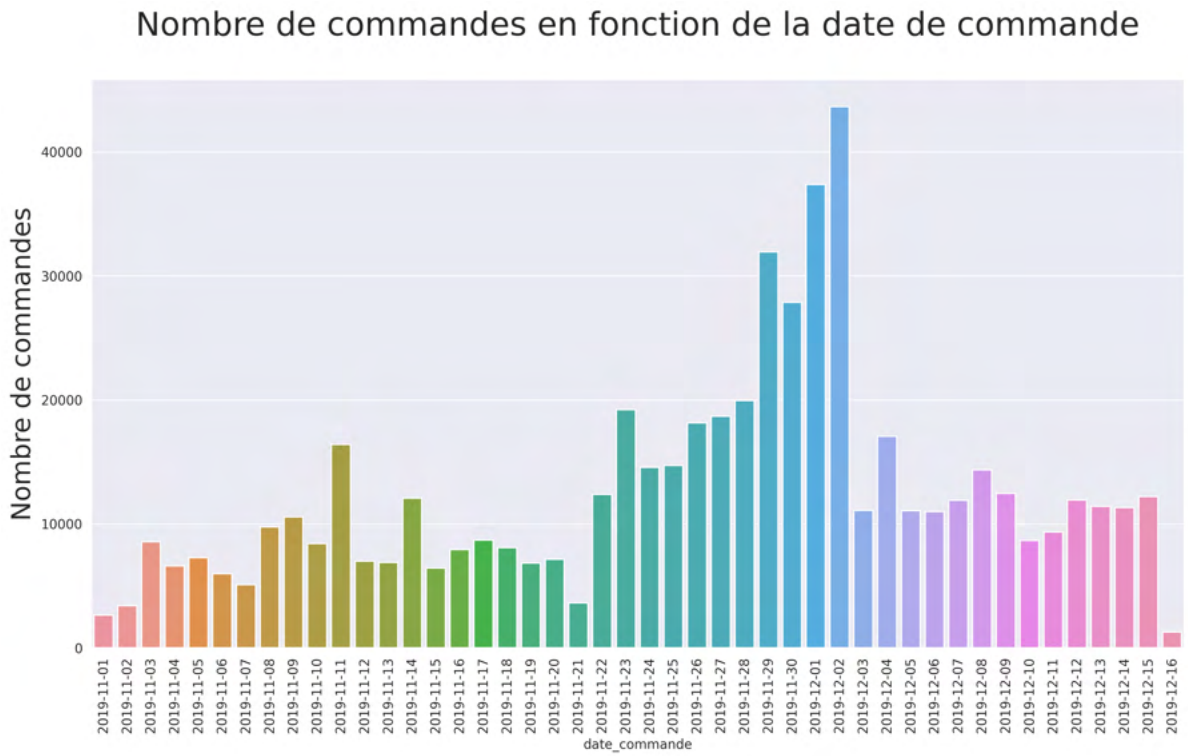
In [ ]:
```
df.describe()
```

Out[ ]:

	heure_commande	min_commande	gap	jour_expe	jour_commande	
count	572807.000000	572807.000000	572807.000000	572807.000000	572807.000000	572807.
mean	15.333858	28.669850	2.317587	2.415630	3.156486	737265.
std	4.758604	17.720156	1.236174	1.882427	2.133388	11.
min	0.000000	0.000000	0.000000	0.000000	0.000000	737240.
25%	12.000000	13.000000	1.000000	1.000000	1.000000	737258.
50%	16.000000	29.000000	2.000000	2.000000	3.000000	737268.
75%	19.000000	44.000000	3.000000	4.000000	5.000000	737273.
max	23.000000	59.000000	15.000000	6.000000	6.000000	737285.

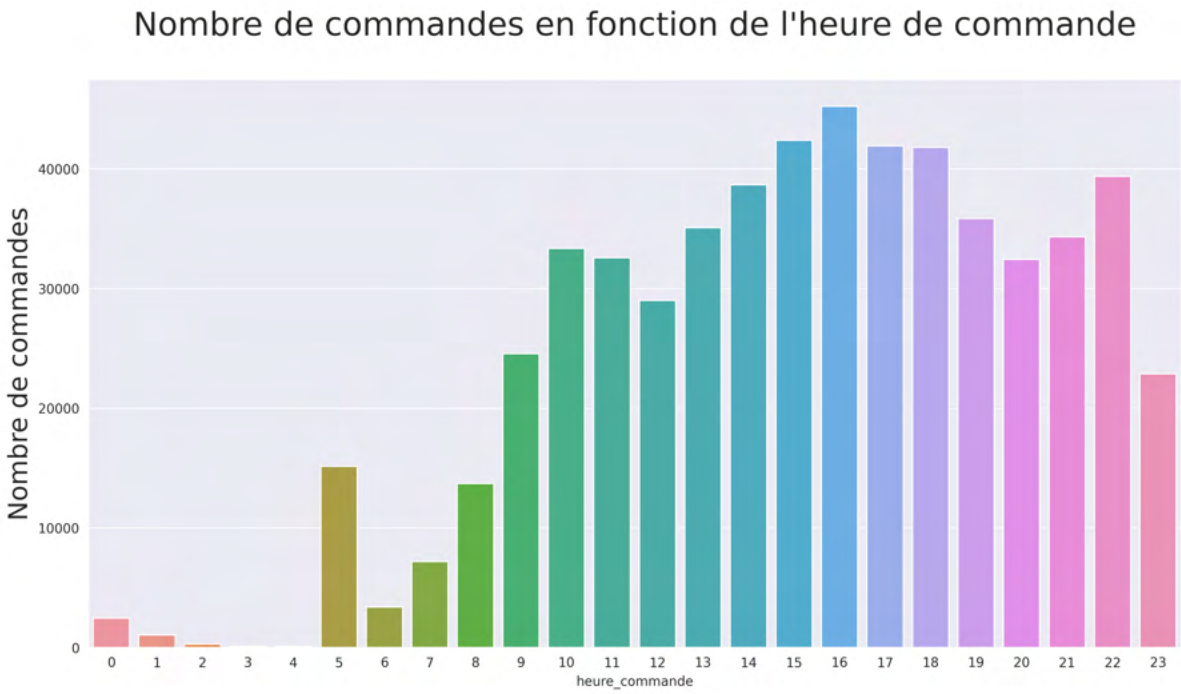
In [ ]:
```
param="date_commande"
titre ="Nombre de commandes en fonction de la date de commande\n"
ylabel="Nombre de commandes"
```

```
a=sns.barplot(data=df.groupby(param).agg("count").reset_index().sort_values(param)
              x=param,
              y="gap")

labels= df.groupby(param).agg("count").reset_index()[param].sort_values().dt.date
a.set_ylabel(ylabel, fontsize=20)
a.set_title(titre, fontsize=25)
a.set_xticklabels(labels=labels ,rotation = 90)
plt.show()
```

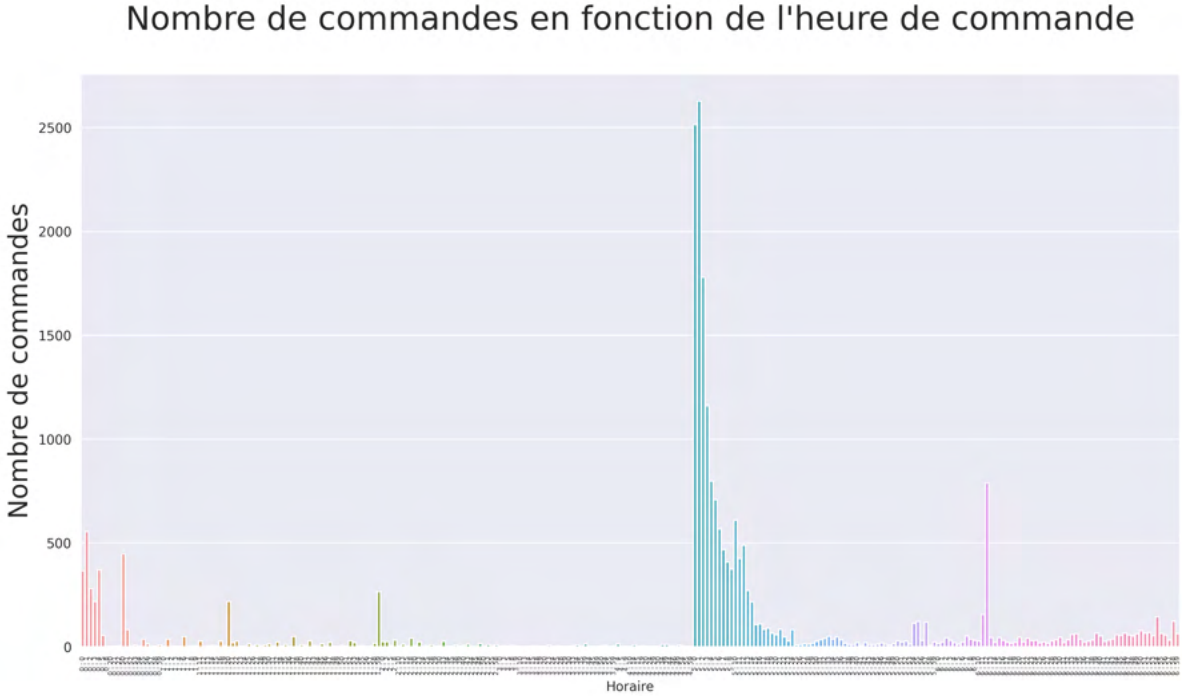


In [ ]:

```
param="heure_commande"
titre ="Nombre de commandes en fonction de l'heure de commande\n"
ylabel="Nombre de commandes"

a=sns.barplot(data=df.groupby(param).agg("count").reset_index().sort_values(param)
              x=param,
              y="gap")


a.set_ylabel(ylabel, fontsize=20)
a.set_title(titre, fontsize=25)
plt.show()
```



In [ ]:
```
param="heure_commande"
titre ="Nombre de commandes en fonction de l'heure de commande\n"
ylabel="Nombre de commandes"

df_temp=df[df["heure_commande"].isin([0,1,2,3,4,5,6])].groupby(["heure_commande","r
df_temp["Horaire"] = df_temp["heure_commande"].astype(str) + " : "+df_temp["min_cor

a=sns.barplot(data=df_temp,
              x="Horaire",
              y="gap")
labels=df_temp["Horaire"]
a.set_xticklabels(labels=labels ,rotation = 90,fontsize=5)

a.set_ylabel(ylabel, fontsize=20)
a.set_title(titre, fontsize=25)
plt.show()
```



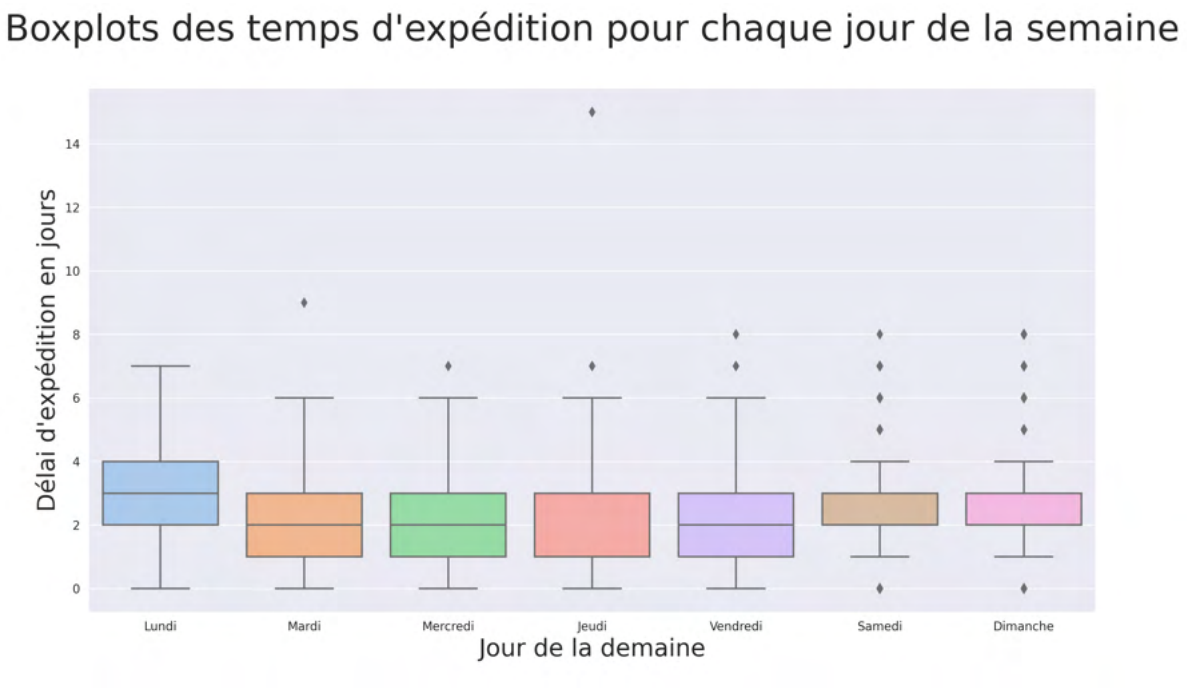
On observe qu'il y a un nombre de commandes élevé à 5 h du matin et

presque nul à 3 et 4 heures

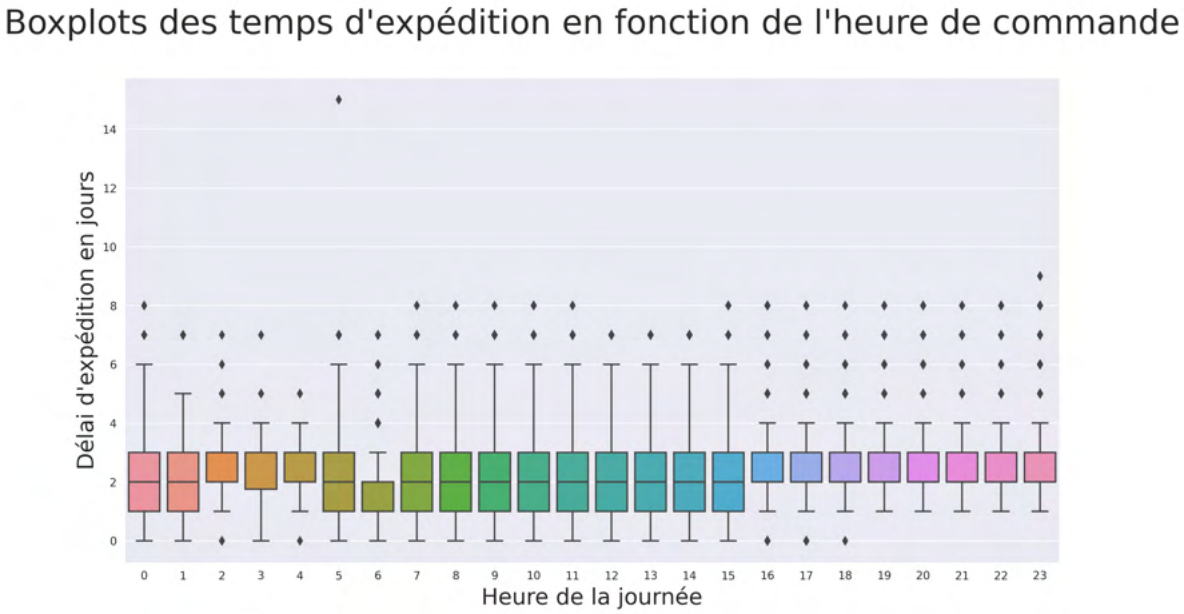
Boxplots dates de commande :

Boxplots entre les jours de la semaine et le délai de livraison

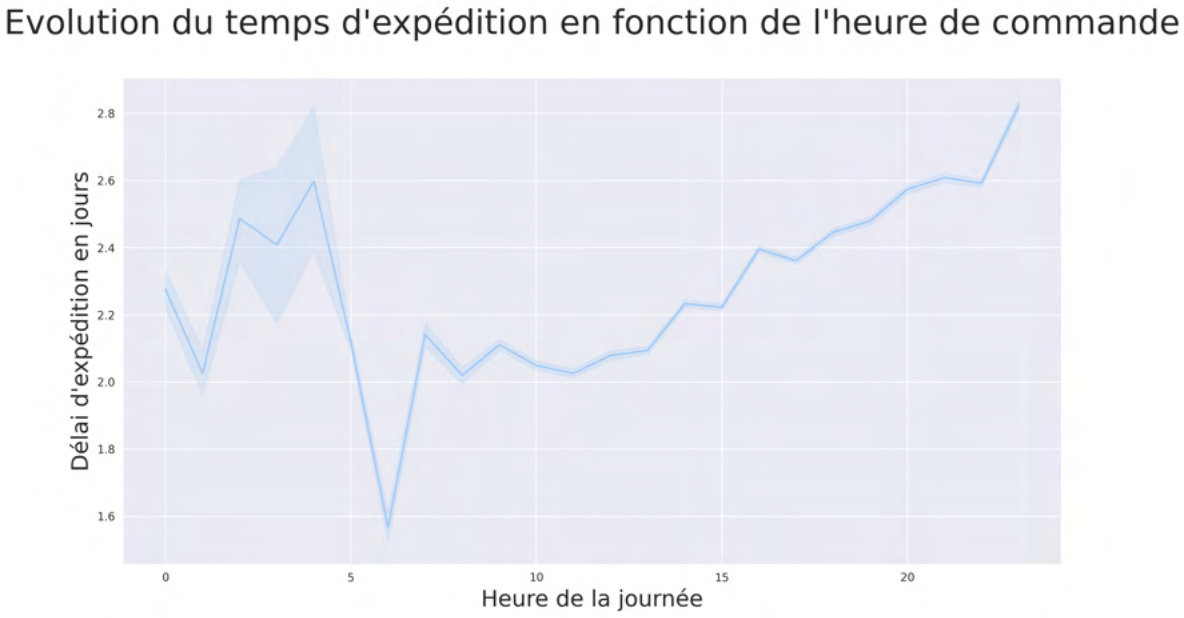
In [ ]:
```
var ="jour_commande"
sns.boxplot(data=df,
            x=var,
            y="gap",)
            #order=df.groupby("provider").agg("count")[["gap"]].sort_values("gap",

plt.title("Boxplots des temps d'expédition pour chaque jour de la semaine\n",fonts
plt.ylabel("Délai d'expédition en jours", fontsize=20)
plt.xlabel("Jour de la demaine\n",fontsize=20)
plt.xticks(labels=["Lundi","Mardi","Mercredi","Jeudi","Vendredi","Samedi","Dimanch
plt.show()
```



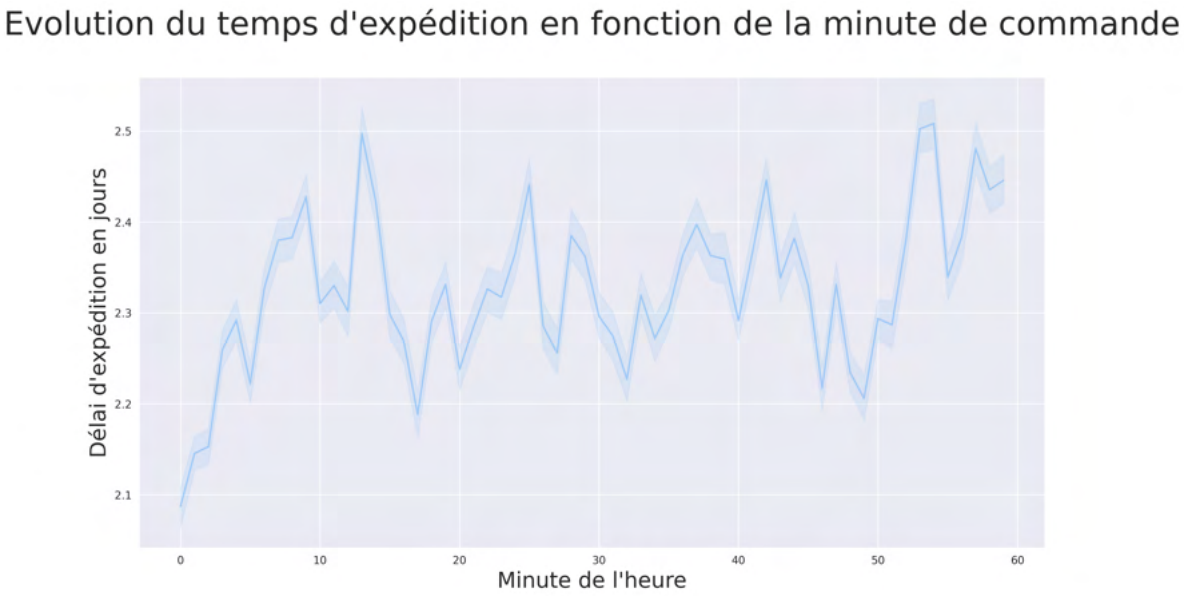
In [ ]:
```
var ="heure_commande"
sns.boxplot(data=df,
            x=var,
            y="gap",)
            #order=df.groupby("provider").agg("count")[["gap"]].sort_values("gap",

plt.title("Boxplots des temps d'expédition en fonction de l'heure de commande\n",f
plt.ylabel("Délai d'expédition en jours", fontsize=20)
plt.xlabel("Heure de la journée\n",fontsize=20)
plt.show()
```



In [ ]:
```
sns.lineplot(data=df, x="heure_commande",y="gap")

plt.title("Evolution du temps d'expédition en fonction de l'heure de commande\n",f
plt.ylabel("Délai d'expédition en jours", fontsize=20)
plt.xlabel("Heure de la journée\n",fontsize=20)
plt.show()
```

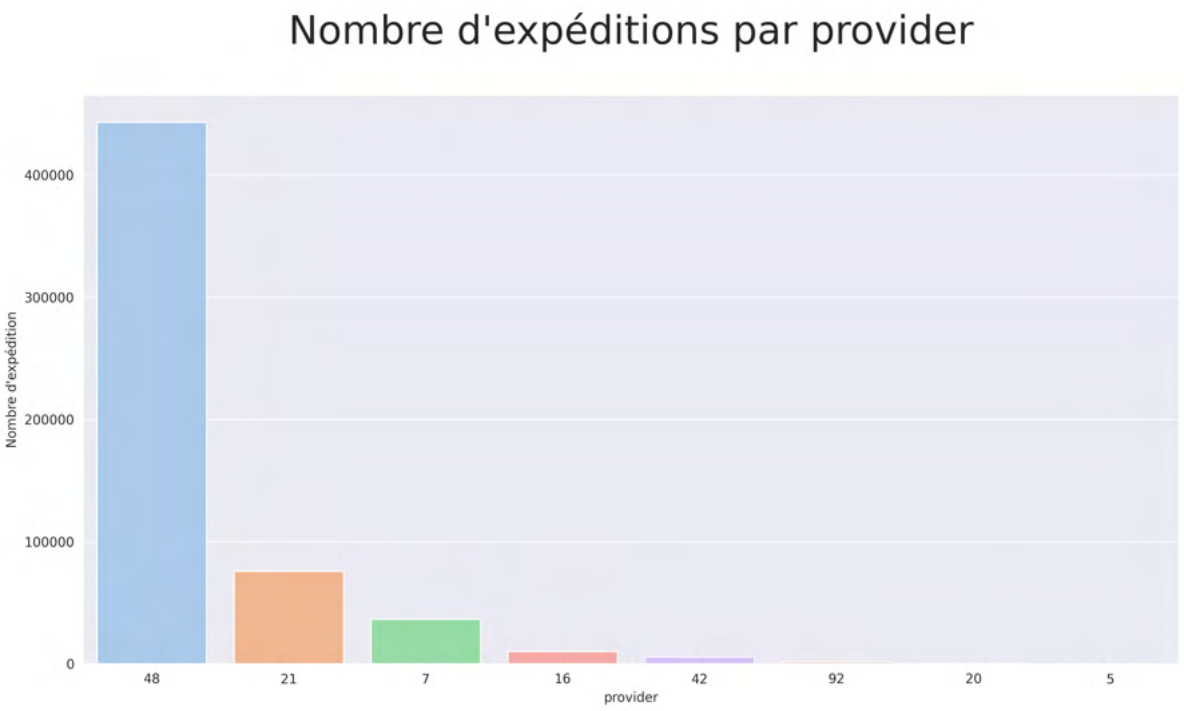


In [ ]:
```
sns.lineplot(data=df, x="min_commande",y="gap")

plt.title("Evolution du temps d'expédition en fonction de la minute de commande\n"
plt.ylabel("Délai d'expédition en jours", fontsize=20)
plt.xlabel("Minute de l'heure\n",fontsize=20)
plt.show()
```

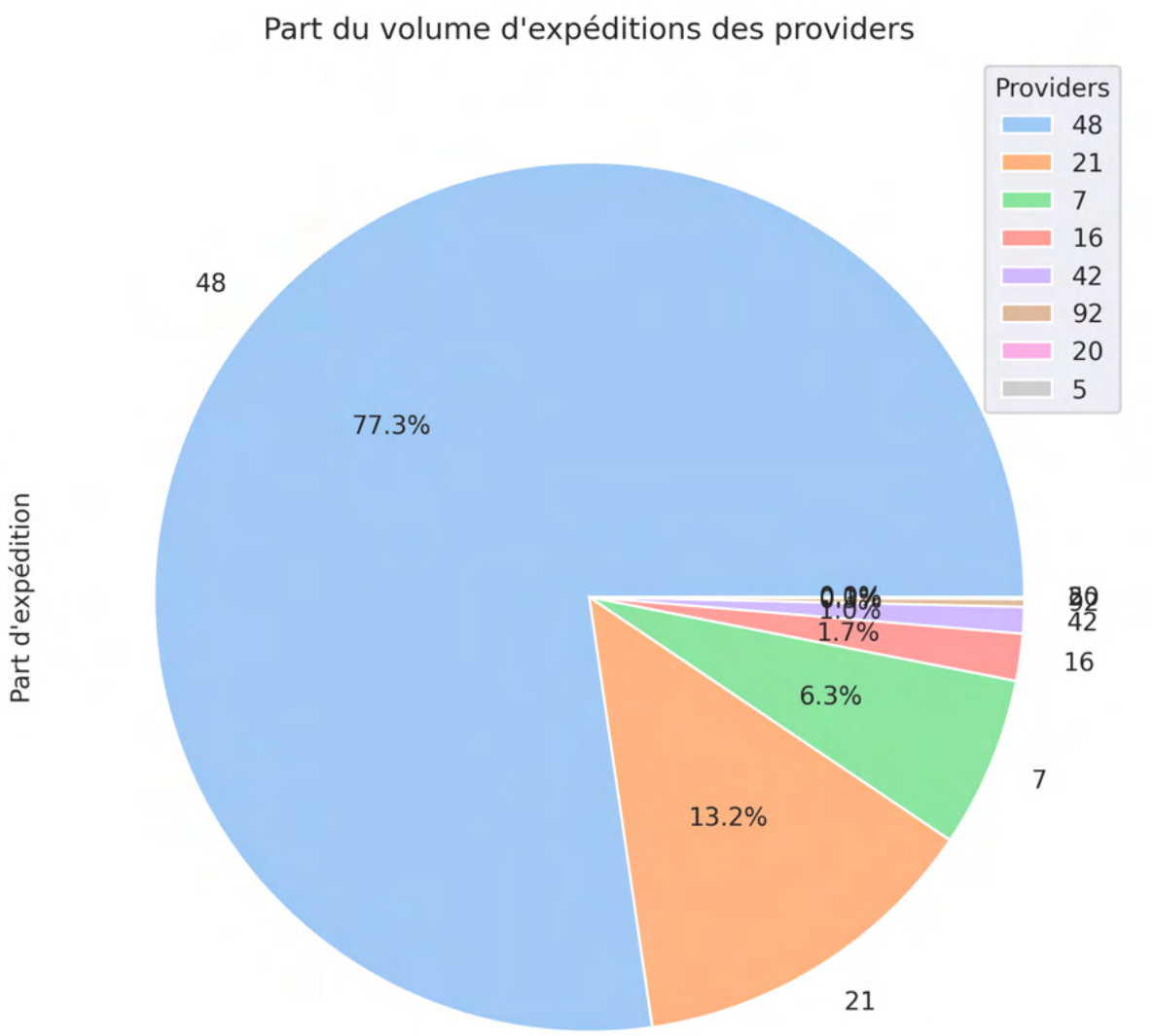


Observations providers :

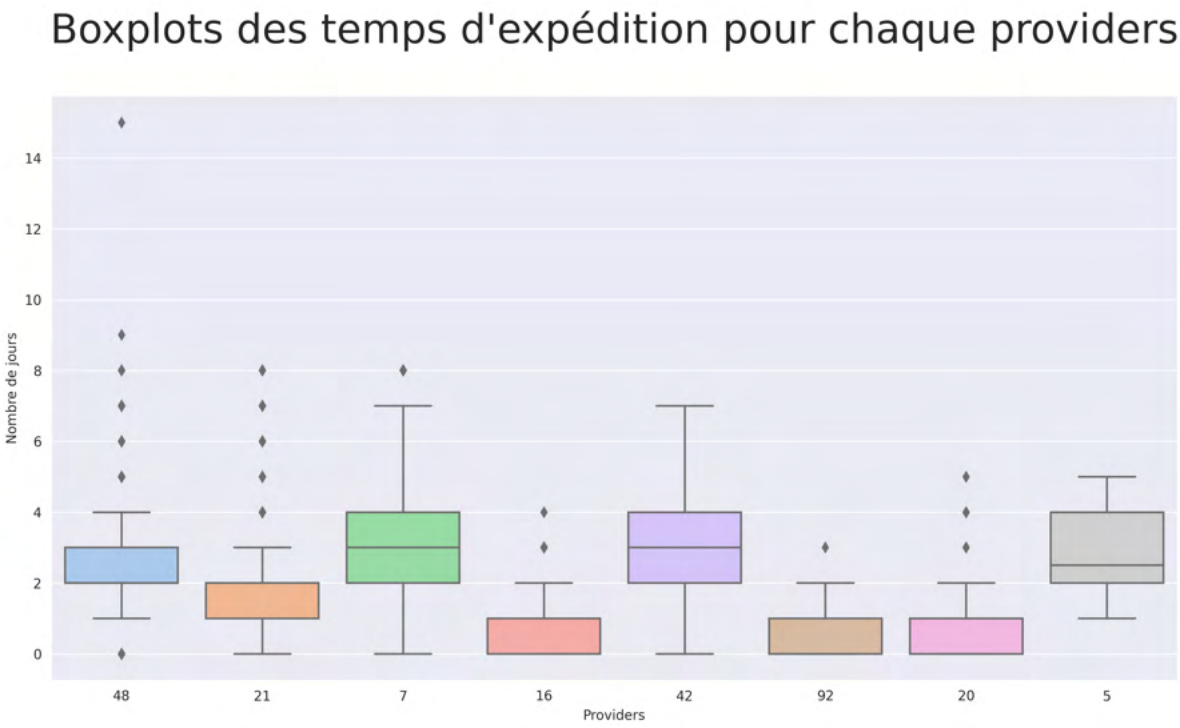
In [ ]:
```
df_temp=df.groupby("provider").agg("count")[["gap"]]
df_temp.columns=["Nombre d'expédition"]
sns.barplot(data=df_temp.reset_index().sort_values("Nombre d'expédition",ascending=
plt.title("Nombre d'expéditions par provider\n",fontsize=30)
plt.show()
```



In [ ]:
```
df_temp=df.groupby("provider").agg("count")[["gap"]].sort_values("gap",ascending=Fa
df_temp.columns=["Part d'expédition"]

df_temp.plot.pie(y="Part d'expédition", autopct='%1.1f%%')
plt.legend(title="Providers")
plt.title("Part du volume d'expéditions des providers")
plt.show()
```

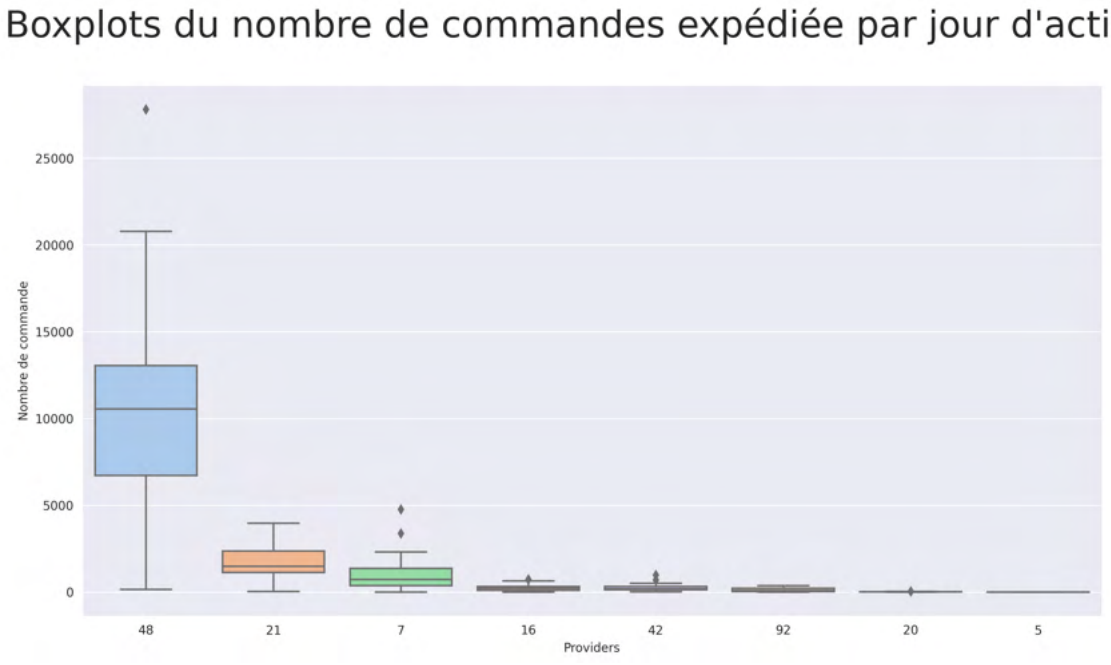


In [ ]:
```
sns.boxplot(data=df,
            x="provider",
            y="gap",
            order=df.groupby("provider").agg("count")[["gap"]].sort_values("gap",as

plt.title("Boxplots des temps d'expédition pour chaque providers\n",fontsize=30)
plt.ylabel("Nombre de jours")
plt.xlabel("Providers")
plt.show()
```



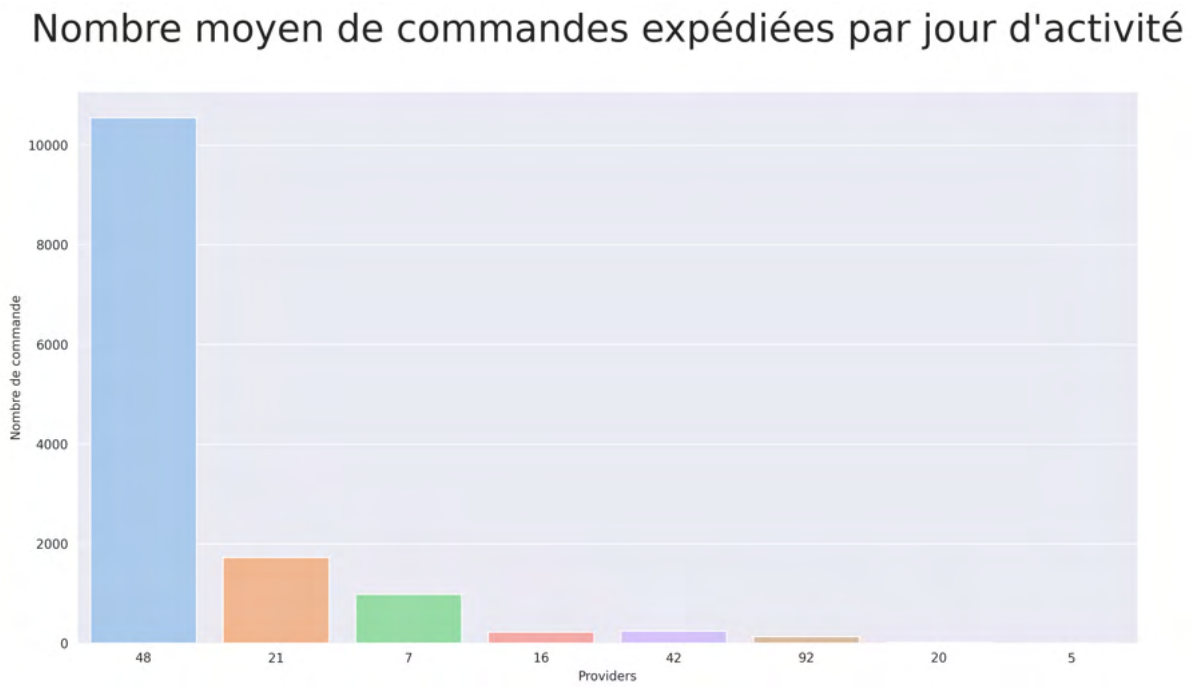
Nombre de commandes expédiées par date de livraison

In [ ]:
```
df_temp = df.groupby(["provider","date_expe"]).agg("count").reset_index()[["provide
sns.boxplot(data=df_temp,
            x="provider",
            y="gap",
            order=df.groupby("provider").agg("count")[["gap"]].sort_values("gap",as

plt.title("Boxplots du nombre de commandes expédiée par jour d'activité\n",fontsize
plt.ylabel("Nombre de commande")
plt.xlabel("Providers")
plt.show()
```



In [ ]:
```
df_temp = df.groupby(["provider","date_expe"]).agg("count").reset_index()[["provide
sns.barplot(data=df_temp,
            x="provider",
```

```
            y="gap",
            order=df.groupby("provider").agg("count")[["gap"]].sort_values("gap",as

plt.title("Nombre moyen de commandes expédiées par jour d'activité\n",fontsize=30)
plt.ylabel("Nombre de commande")
plt.xlabel("Providers")
plt.show()
```



In [ ]:
```
df_temp.sort_values("gap",ascending=False)
```

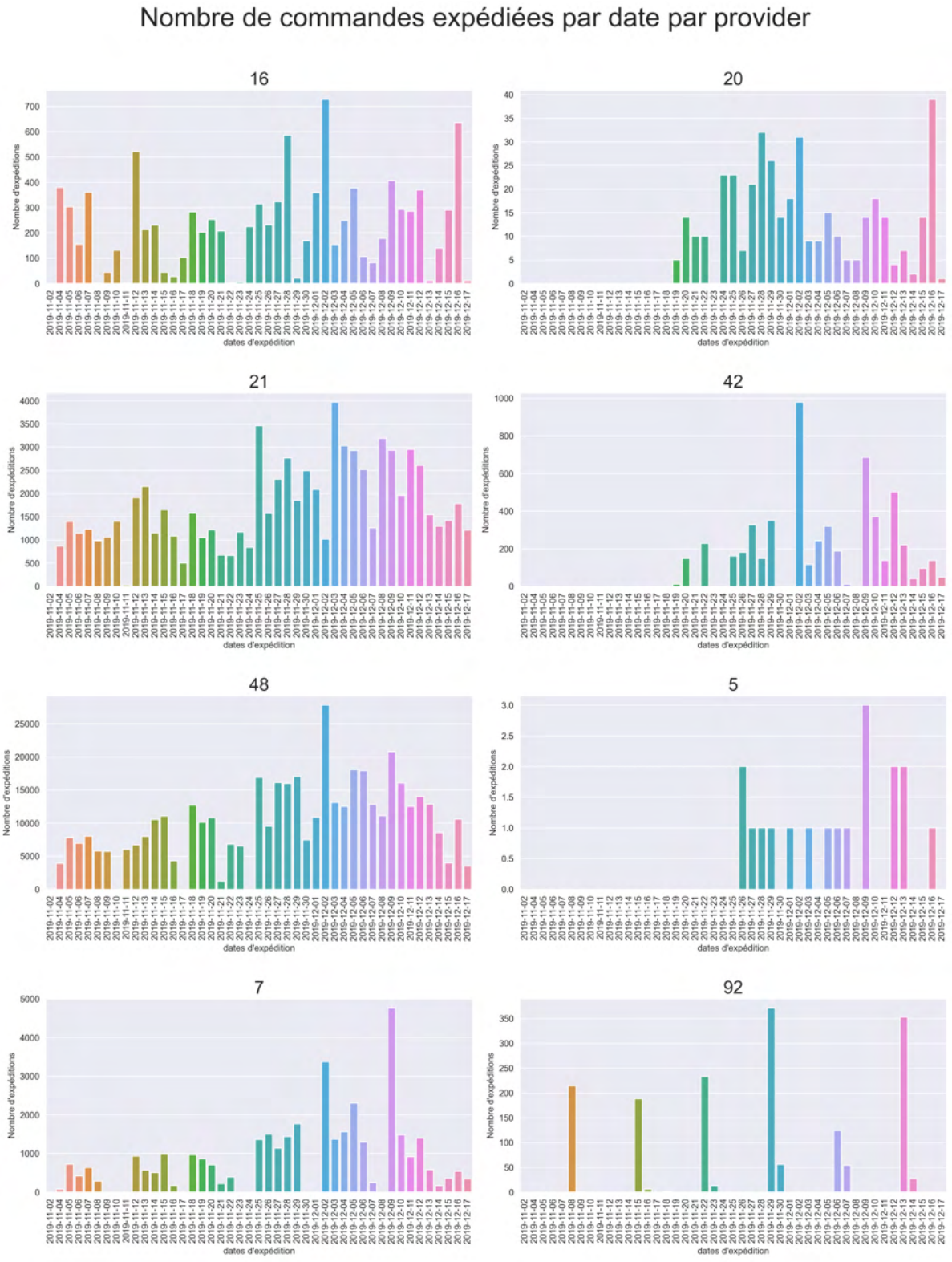
Out[ ]:

	provider	gap
4	48	10544.785714
2	21	1723.795455
6	7	982.729730
3	42	245.521739
0	16	227.568182
7	92	136.666667
1	20	14.285714
5	5	1.384615

In [4]:
```
def subplots_auto_barplot(df,var_filtre, cols,xvar,yvar,rotate=False, title=""):
    distinct_values=df[var_filtre].unique().tolist()
    Tot = len(distinct_values)
    Rows = Tot // cols
    if Tot % cols != 0:
        Rows += 1

    Position = range(1,Tot + 1)

    fig = plt.figure(1,figsize=(15, 5*Rows))
    for k in range(Tot):
        ax = fig.add_subplot(Rows,cols,Position[k])

        #à modifier en fonction de ce que l'on souhaite afficher
```

```
        df_temp=df[df[var_filtre]==distinct_values[k]]

        sns.barplot(data=df_temp, x=xvar,y=yvar, ax=ax)

        if rotate :
            labels=df_temp[xvar]
            ax.set_xticklabels(labels=labels ,rotation = 90)
        ax.set_title("\n"+str(distinct_values[k]),fontsize=20)
    plt.suptitle(title+"\n", fontsize=30)
    plt.tight_layout()
    plt.show()
```

In [21]:
```
df_temp=df.groupby(["provider","date_expe"]).agg("count")[["gap"]]
df_temp=df_temp.reindex( pd.MultiIndex.from_product([df_temp.reset_index().sort_va
                                                     df_temp.reset_index().sort_values("da
                                                     names=['provider', 'date_expe']),fill_

df_temp["date_expe"]=pd.to_datetime(df_temp["date_expe"])
df_temp["date_expe"]=df_temp["date_expe"].dt.date
df_temp.columns =["Provider","dates d'expédition","Nombre d'expéditions"]

subplots_auto_barplot(df=df_temp,
                      var_filtre="Provider",
                      cols=2,
                      xvar="dates d'expédition",
                      yvar="Nombre d'expéditions",
                      rotate=True,
                      title="Nombre de commandes expédiées par date par provider")
```



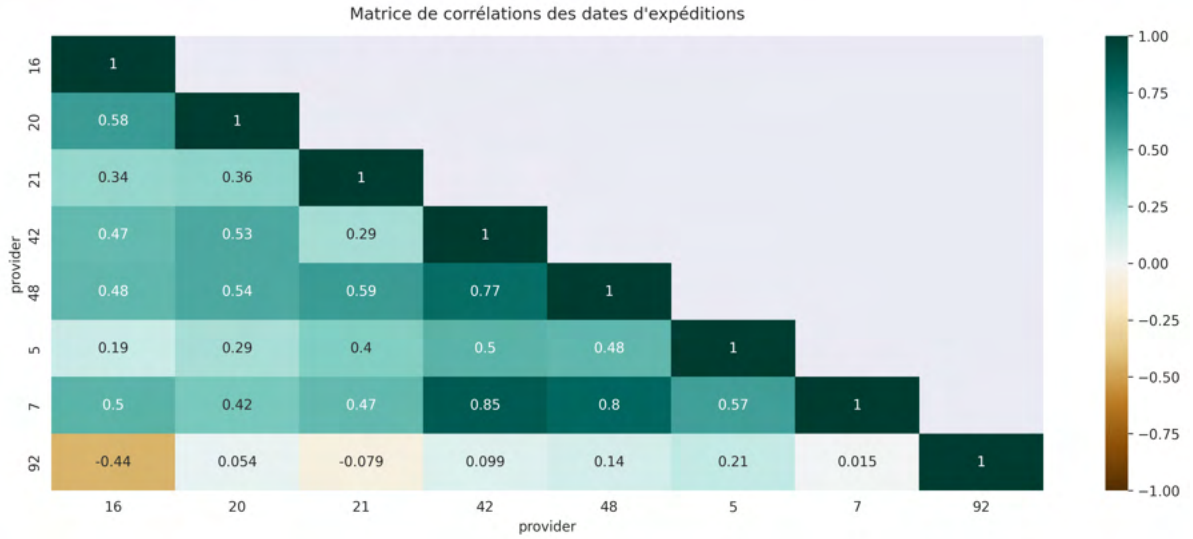
On remarque que les providers ont des comportements différents pour les expéditions des commandes

Les providers 5,20 et 42 ont des comportements étranges, on ne sait pas s'ils livrent toute l'année

Corrélations entre les providers :

In [ ]:
```
df_corr=df.groupby(["date_expe","provider"]).agg("count").reset_index()[["date_expe

# Increase the size of the heatmap.
plt.figure(figsize=(16, 6))
```

```
# Store heatmap object in a variable to easily access it when you want to include m
# Set the range of values to be displayed on the colormap from -1 to 1, and set the
heatmap = sns.heatmap(df_corr.corr(), vmin=-1, vmax=1, annot=True,cmap='BrBG', mask

# Give a title to the heatmap. Pad defines the distance of the title from the top o
heatmap.set_title("Matrice de corrélations des dates d'expéditions", fontdict={'fon
```

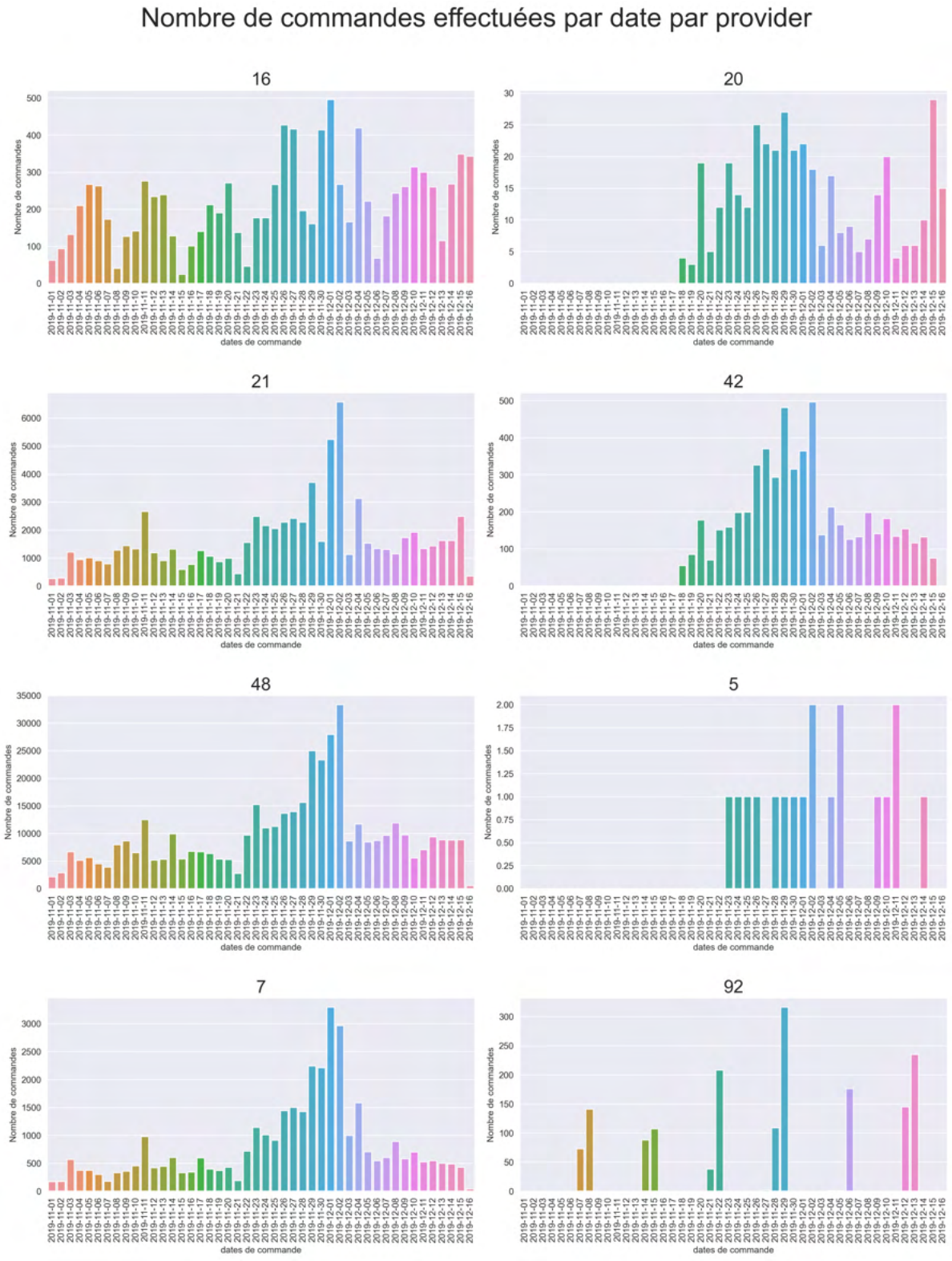


Corrélations positives => viennent sûrement du fait que lorsqu'il y a une augmentation des commandes plusieurs services augmente leur nombre de livraisons

Corrélations négatives => 92 expédie les vendredis et samedi uniquement et 16 ne livre presque pas les vendredis

Pour les dates de commande

In [22]:
```
df_temp=df.groupby(["provider","date_commande"]).agg("count")[["gap"]]
df_temp=df_temp.reindex( pd.MultiIndex.from_product([df_temp.reset_index().sort_val
                                                     df_temp.reset_index().sort_values("dat
                                                     names=['provider', 'date_commande']),

df_temp["date_commande"]=df_temp["date_commande"].dt.date
df_temp.columns =["Provider","dates de commande","Nombre de commandes"]

subplots_auto_barplot(df=df_temp,
                      var_filtre="Provider",
                      cols=2,
                      xvar="dates de commande",
                      yvar="Nombre de commandes",
                      rotate=True,
                      title="Nombre de commandes effectuées par date par provider"
```



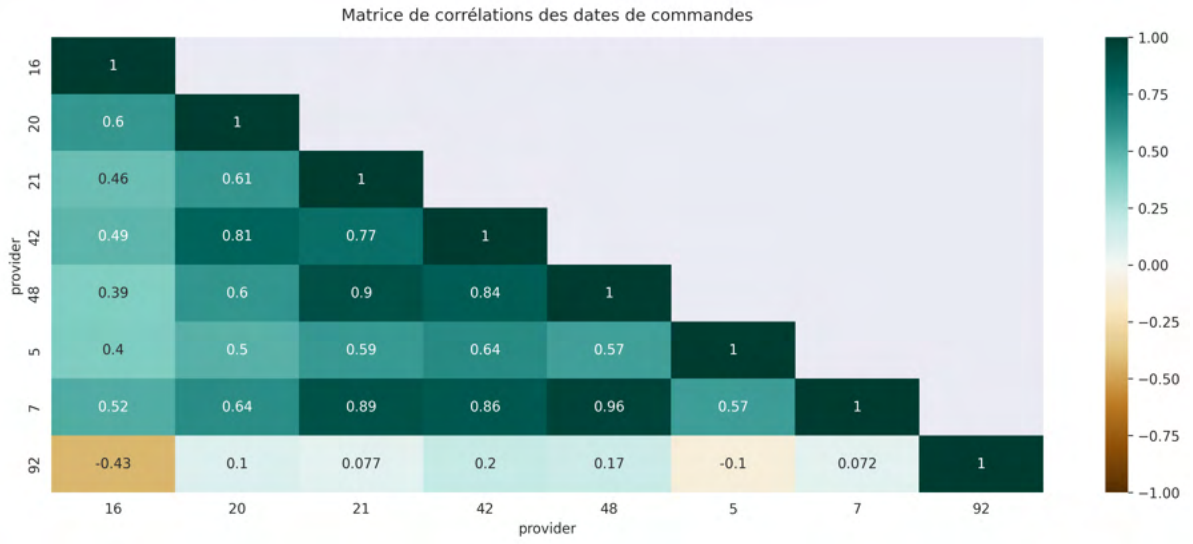
On remarque que les providers ont des comportements différents pour les expéditions des commandes

Les providers 5,20 et 42 ont des comportements étranges, on ne sait pas s'ils livrent toute l'année

92 ne prends les commandes que les jeudi et vendredi et ne fait les expéditions que les vendredis et samedis ?

Corrélations entre les providers :

In [ ]:
```
df_corr=df.groupby(["date_commande","provider"]).agg("count").reset_index()[["date_

# Increase the size of the heatmap.
plt.figure(figsize=(16, 6))


# Store heatmap object in a variable to easily access it when you want to include r
# Set the range of values to be displayed on the colormap from -1 to 1, and set the
heatmap = sns.heatmap(df_corr.corr(), vmin=-1, vmax=1, annot=True,cmap='BrBG', masl


# Give a title to the heatmap. Pad defines the distance of the title from the top o
heatmap.set_title("Matrice de corrélations des dates de commandes", fontdict={'font
```



Corrélations positives => viennent sûrement du fait que lorsqu'il y a une augmentation des commandes plusieurs services augmente leur nombre de livraisons

Corrélations négatives => 92 expédie les vendredis et samedi uniquement et 16 ne livre presque pas les vendredis

Feature extraction

(E) correspond aux variables utilisées pour l'embedding

-Stock (par provider concerné et total) (E) (stock_provider, stock_total)

-Stock de tous les providers ([id_provider]_delay)

-Nombre de jours d'activité par provider et total (sur les nb_jours derniers jours)(E) (jour_act_provider, jour_act_total)

-Nombre de commandes expédiées au cours des X derniers jours par provider et total (X est le délai d'expédition moyen)(E) (nb_expe_provider, nb_expe_total)

-Nombre de commandes expédiées au cours des X derniers jours (X est le délai d'expédition moyen) de tous les providers ([id_provider]_nb_expe)

-Quartiles de délai de livraison (sur les nb_jours derniers jours) pour le provider et total (E) ([quartile]_provider, [quartile]_total)

-Nombre moyen de commandes livrées par jours de la semaine au cours des nb*jours derniers jours (E) (jour*[nb*jour], jour* [nb_jour]_tous_provider)

-Indicateurs de types d'expédition (E)(coef*expe*[type_expe])

Calcul des variables

In [ ]:
```
nb_jours =15
jours_min =3 #Nombre de journées d'expédition minimum pour que l'algo calcule autor
dict_days={}

#Variables qui ne seront pas utilisées pendant l'embedding :

#Stock restant pour chaque provider
for provider in df["provider"].unique().tolist():
    ser_temp=df[df["provider"]==provider].groupby("id_date").agg("count").sort_inde
    ser_temp_expe=df[df["provider"]==provider].groupby(['id_date_expe']).agg("count
    dict_stock={}

    for i in range(len(ser_temp)) :
        dict_stock[ser_temp.index[i]]=ser_temp.iloc[:i].sum() - ser_temp_expe[ser_t

    df[str(provider)+"_delay"]=df["id_date"].map(dict_stock)


#Nombre de commandes envoyées au cours des X derniers jours pour chaque provider
for provider in df["provider"].unique().tolist():
    mean_delay = df.groupby("provider").agg("mean")["gap"].apply(np.ceil).loc[provi

    indexes=df["id_date"].unique().tolist()
    ser_temp_expe=df[df["provider"]==provider].groupby(['id_date_expe']).agg("count
    dict_activity={}
    for i in indexes :
        dict_activity[i]=ser_temp_expe.iloc[(ser_temp_expe.index>=i-mean_delay) &
    df[str(provider)+"_nb_expe"]=df["id_date"].map(dict_activity)



#Paramétrages manuels des types de service de livraison initiaux
start_settings={
    "48":"rapide",
    "21":"rapide",
    "7" :"normale",
    "16": "express",
    "42":"normale",
    "92":"express",
    "20":"express",
    "5":"lente"
}



#Génération des différents types de clusters
from sklearn.cluster import KMeans
type_expe = KMeans(n_clusters=4)
data = df.set_index("provider")[["gap"]]

type_expe.fit(data)
data["cluster"]=type_expe.predict(data)

df_temp = data.reset_index().groupby(["cluster"]).agg("mean").sort_values("gap")[['
df_temp["type_expe"]="None"
```

```
df_temp.iloc[0,1]="express"
df_temp.iloc[1,1]="rapide"
df_temp.iloc[2,1]="normale"
df_temp.iloc[3,1]="lente"
dict_cluster_type_expe= df_temp["type_expe"].to_dict()

#On labelise toutes les commandes avec ce clustering :
df["expedition"]= type_expe.predict(df.set_index("provider")[["gap"]])
df["expedition"]=df["expedition"].map(dict_cluster_type_expe)

print("Résultats du clustering :")
df_temp=data.reset_index().groupby(["cluster"]).agg(nb_commandes=('gap', 'count'),
df_temp["Type expedition"]=df_temp["cluster"].map(dict_cluster_type_expe)
print(df_temp.drop(columns=["cluster"]))

#Initialisation des colonnes
#stock :
df["stock_provider"]=0
df["stock_total"]=0

#Nombre de jours d'activité sur les nb_jours derniers
df["jours_act_provider"]=0
df["jours_act_total"]=0

#Nombre de commandes expédiées au cours des X derniers jours (X est le délai d'expe
df["nb_expe_provider"]=0
df["nb_expe_total"]=0

#Quartiles des délais de livraison au cours des nb_jours derniers jours pour chaque
df["min_provider"]=0
df["Q1_provider"]=0
df["med_provider"]=0
df["Q3_provider"]=0
df["max_provider"]=0

#Quartiles des délais de livraison au cours des nb_jours derniers jours total :
df["min_total"]=0
df["Q1_total"]=0
df["med_total"]=0
df["Q3_total"]=0
df["max_total"]=0

#Nombre moyen de commandes expédiées par jours de la semaine au cours des nb_jours
df["jour_0"]=0
df["jour_1"]=0
df["jour_2"]=0
df["jour_3"]=0
df["jour_4"]=0
df["jour_5"]=0
df["jour_6"]=0

#Nombre moyen de commandes expédiées par jours de la semaine au cours des nb_jours
df["jour_0_tous_provider"]=0
df["jour_1_tous_provider"]=0
df["jour_2_tous_provider"]=0
df["jour_3_tous_provider"]=0
df["jour_4_tous_provider"]=0
df["jour_5_tous_provider"]=0
df["jour_6_tous_provider"]=0

#Part des commandes pour chaque type de livraison par provider :
df["coef_expe_express"]=0
df["coef_expe_rapide"]=0
df["coef_expe_normale"]=0
```

```
df["coef_expe_lente"]=0




for provider in df["provider"].unique().tolist():# Pour chaque provider

    #Variables temporaires :

    df_provider=df[df["provider"]==provider]#dataframe avec uniquement les lignes p

    nb_commandes_jour_commande=df_provider.groupby("id_date").agg("count").sort_in
    nb_commandes_jour_expe=df_provider.groupby(['id_date_expe']).agg("count").sort_

    indexes=np.sort(df_provider["id_date"].unique()).tolist()#Liste des dates de c

    min_date_expe=df_provider["id_date_expe"].min()#première date d'expédition du p

    #variables temporaires propres au calcul du nombre de jours d'activité
    df_activity=df_provider[["id_date_expe"]].drop_duplicates().sort_values("id_da
    df_activity["unit"]=1
    ser_expe_activity= df_activity.set_index("id_date_expe").sort_index()["unit"]#

    #Variables temporaires propres au calcul du nombre de commandes expédiées au c
    ser_expe_gap=df_provider.set_index("id_date").sort_index()["gap"]#Série avec d



    #Dictionnaires de mapping :
    dict_stock={}
    dict_activity={}
    dict_nb_commandes_X_days={}
    dict_min={}
    dict_Q1={}
    dict_med={}
    dict_Q3={}
    dict_max={}
    dict_days[provider]={}
    dict_coef_express={}
    dict_coef_rapide={}
    dict_coef_normale={}
    dict_coef_lente={}

    #Code pour calculer chacune des variables :


    for i in indexes : #Pour chaque date de commande

        borne_inferieure = (i-nb_jours)*(i-nb_jours>min_date_expe)#date du jour

        #Calcul du stock de la date de commande
        dict_stock[i] = nb_commandes_jour_commande.loc[nb_commandes_jour_commande.

        #Nombre de jours en activité sur les nb_jours derniers jours:
        dict_activity[i] = len(ser_expe_activity.loc[(borne_inferieure<=ser_expe_a

        #Nombre de commandes expédiées au cours des X derniers jours (X est le délc
        mean_delay=np.ceil(ser_expe_gap.loc[ser_expe_gap.index<i].mean())#délai moy
        dict_nb_commandes_X_days[i]=nb_commandes_jour_expe.iloc[(nb_commandes_jour_

        #Quartiles du provider :
        ser_borned=nb_commandes_jour_expe.loc[(nb_commandes_jour_expe.index>=borne_
        dict_min[i]=ser_borned.quantile(0)
        dict_Q1[i]=ser_borned.quantile(0.25)
```

```
        dict_med[i]=ser_borned.quantile(0.5)
        dict_Q3[i]=ser_borned.quantile(0.75)
        dict_max[i]=ser_borned.quantile(1)

        #Nombre moyen de commandes livrées par jour de la semaine :
        dict_days[provider][i]= df_provider[(df_provider["id_date"]>=borne_inferieu

        #Coefficients des types des livraisons :
        if len(nb_commandes_jour_expe.loc[nb_commandes_jour_expe.index<i])>=jours_
            condition = (df_provider["id_date"]<i) & (df_provider["id_date"]>=borne
            df_temp =(df_provider[condition].groupby(["expedition"]).agg("count")[
            dict_coef_express[i] =df_temp.loc["express"].iloc[0]
            dict_coef_rapide[i] =df_temp.loc["rapide"].iloc[0]
            dict_coef_normale[i]=df_temp.loc["normale"].iloc[0]
            dict_coef_lente[i]=df_temp.loc["lente"].iloc[0]
        else :
            #On utilise le dictionnaire pour mapper
            type_expe_start=start_settings[provider]
            if type_expe_start=="express" :
                dict_coef_express[i]=1
            if type_expe_start=="rapide":
                dict_coef_rapide[i]=1
            if type_expe_start=="normale":
                dict_coef_lente[i]=1
            if type_expe_start=="lente":
                dict_coef_normale[i]=1

    #Mapping des variables dans df :
    df.loc[df["provider"]==provider,"stock_provider"]=df["id_date"].map(dict_stock
    df.loc[df["provider"]==provider,"jours_act_provider"]=df["id_date"].map(dict_a
    df.loc[df["provider"]==provider,"nb_expe_provider"]=df["id_date"].map(dict_nb_
    df.loc[df["provider"]==provider,"min_provider"]=df["id_date"].map(dict_min)
    df.loc[df["provider"]==provider,"Q1_provider"]=df["id_date"].map(dict_Q1)
    df.loc[df["provider"]==provider,"med_provider"]=df["id_date"].map(dict_med)
    df.loc[df["provider"]==provider,"Q3_provider"]=df["id_date"].map(dict_Q3)
    df.loc[df["provider"]==provider,"max_provider"]=df["id_date"].map(dict_max)
    df.loc[df["provider"]==provider,"coef_expe_express"]=df["id_date"].map(dict_co
    df.loc[df["provider"]==provider,"coef_expe_rapide"]=df["id_date"].map(dict_coe
    df.loc[df["provider"]==provider,"coef_expe_normale"]=df["id_date"].map(dict_co
    df.loc[df["provider"]==provider,"coef_expe_lente"]=df["id_date"].map(dict_coef_

for provider in dict_days :
        for id_date in dict_days[provider] :
            for jour in dict_days[provider][id_date]:
                df.loc[(df["provider"]==provider) & (df["id_date"]==id_date),"jour_




#Pour les variables totales :
#Variables temporaires :


nb_commandes_jour_commande=df.groupby("id_date").agg("count").sort_index()["gap"]#
nb_commandes_jour_expe=df.groupby(['id_date_expe']).agg("count").sort_index()["gap

indexes=np.sort(df["id_date"].unique()).tolist()#Liste des dates de commande

min_date_expe=df["id_date_expe"].min()#première date d'expédition

#variables temporaires propres au calcul du nombre de jours d'activité
df_activity=df[["id_date_expe"]].drop_duplicates().sort_values("id_date_expe")#Dat
df_activity["unit"]=1
ser_expe_activity= df_activity.set_index("id_date_expe").sort_index()["unit"]#Séri
```

```
#Variables temporaires propres au calcul du nombre de commandes expédiées au cours
ser_expe_gap=df.set_index("id_date").sort_index()["gap"]#Série avec date d'expe en



#Dictionnaires de mapping :
dict_stock={}
dict_activity={}
dict_nb_commandes_X_days={}
dict_min={}
dict_Q1={}
dict_med={}
dict_Q3={}
dict_max={}
dict_days={}

#Code pour calculer chacune des variables :


for i in indexes : #Pour chaque date de commande

    borne_inferieure = (i-nb_jours)*(i-nb_jours>min_date_expe)

    #Calcul du stock de la date de commande
    dict_stock[i] = nb_commandes_jour_commande.loc[nb_commandes_jour_commande.index

    #Nombre de jours en activité sur les nb_jours derniers jours:
    dict_activity[i] = len(ser_expe_activity.loc[(borne_inferieure<=ser_expe_activi

    #Nombre de commandes expédiées au cours des X derniers jours (X est le délai d
    mean_delay=np.ceil(ser_expe_gap.loc[ser_expe_gap.index<i].mean())#délai moyen
    dict_nb_commandes_X_days[i]=nb_commandes_jour_expe.iloc[(nb_commandes_jour_expe

    #Quartiles  :
    ser_borned=nb_commandes_jour_expe.loc[(nb_commandes_jour_expe.index>=borne_infe
    dict_min[i]=ser_borned.quantile(0)
    dict_Q1[i]=ser_borned.quantile(0.25)
    dict_med[i]=ser_borned.quantile(0.5)
    dict_Q3[i]=ser_borned.quantile(0.75)
    dict_max[i]=ser_borned.quantile(1)

    #Nombre moyen de commandes livrées par jour de la semaine :
    dict_days[i]= df[(df["id_date"]>=borne_inferieure) & (df["id_date"]<i)].groupby

#Mapping des variables dans df :

df["stock_total"]=df["id_date"].map(dict_stock)
df["jours_act_total"]=df["id_date"].map(dict_activity)
df["nb_expe_total"]=df["id_date"].map(dict_nb_commandes_X_days)
df["min_total"]=df["id_date"].map(dict_min)
df["Q1_total"]=df["id_date"].map(dict_Q1)
df["med_total"]=df["id_date"].map(dict_med)
df["Q3_total"]=df["id_date"].map(dict_Q3)
df["max_total"]=df["id_date"].map(dict_max)


for id_date in dict_days :
    for jour in dict_days[id_date]:
        df.loc[ df["id_date"]==id_date,"jour_"+str(jour)+"_tous_provider"]=dict_day

df=df[~df["provider"].isna()]
df=df.fillna(0)
print(df.shape)
```

```
df.to_csv("./data/data_transformed_min"+str(jours_min)+"_nb_jours"+str(nb_jours)+"
df.head()
```

```
Résultats du clustering :
   nb_commandes  temps_expe_moyen  variance_temps_expe Type expedition
2        155286          0.906791             0.290726         express
0        201413          2.000000             0.000000          rapide
3        123320          3.000000             0.000000         normale
1         92788          4.461051             0.727220           lente
(572807, 61)
```

Out[ ]:

	provider	date_expe	date_commande	heure_commande	min_commande	gap	jour_expe	jou
0	48	2019-11-02	2019-11-01	11	40	1	5	
1	48	2019-11-02	2019-11-01	11	40	1	5	
2	48	2019-11-02	2019-11-01	10	24	1	5	
3	48	2019-11-02	2019-11-01	14	24	1	5	
4	48	2019-11-02	2019-11-01	13	28	1	5	

In [ ]:
```
df=pd.read_csv("./data/data_transformed_min3_nb_jours15.csv")
df["date_expe"]=pd.to_datetime(df["date_expe"])
df["date_commande"]=pd.to_datetime(df["date_commande"])
```

Observations sur les variables extraites :

In [ ]:
```
df.groupby(["date_commande"]).agg("mean")[['48_delay', '16_delay', '21_delay', '7_
       '92_delay', '42_delay', '20_delay', '5_delay']].describe()
```

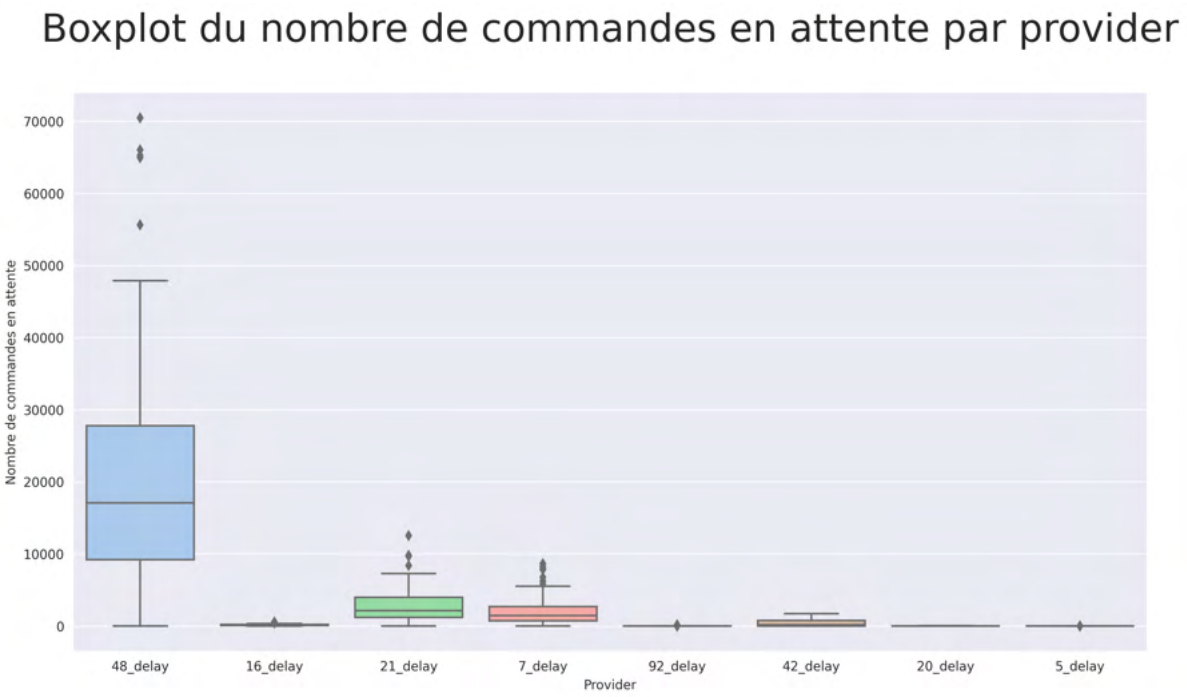
Out[ ]:

	48_delay	16_delay	21_delay	7_delay	92_delay	42_delay	20_delay
count	46.000000	46.000000	46.000000	46.000000	46.000000	46.000000	46.000000
mean	22707.456522	147.543478	3094.760870	2363.021739	10.000000	411.195652	6.326087
std	18785.482206	122.666070	2838.229269	2452.436996	31.009676	496.892683	7.630507
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	9197.250000	66.000000	1186.750000	705.250000	0.000000	0.000000	0.000000
50%	17080.000000	118.500000	2125.500000	1440.000000	0.000000	155.000000	3.000000
75%	27753.750000	184.500000	3962.250000	2666.500000	0.000000	752.000000	10.000000
max	70499.000000	542.000000	12520.000000	8628.000000	145.000000	1689.000000	25.000000

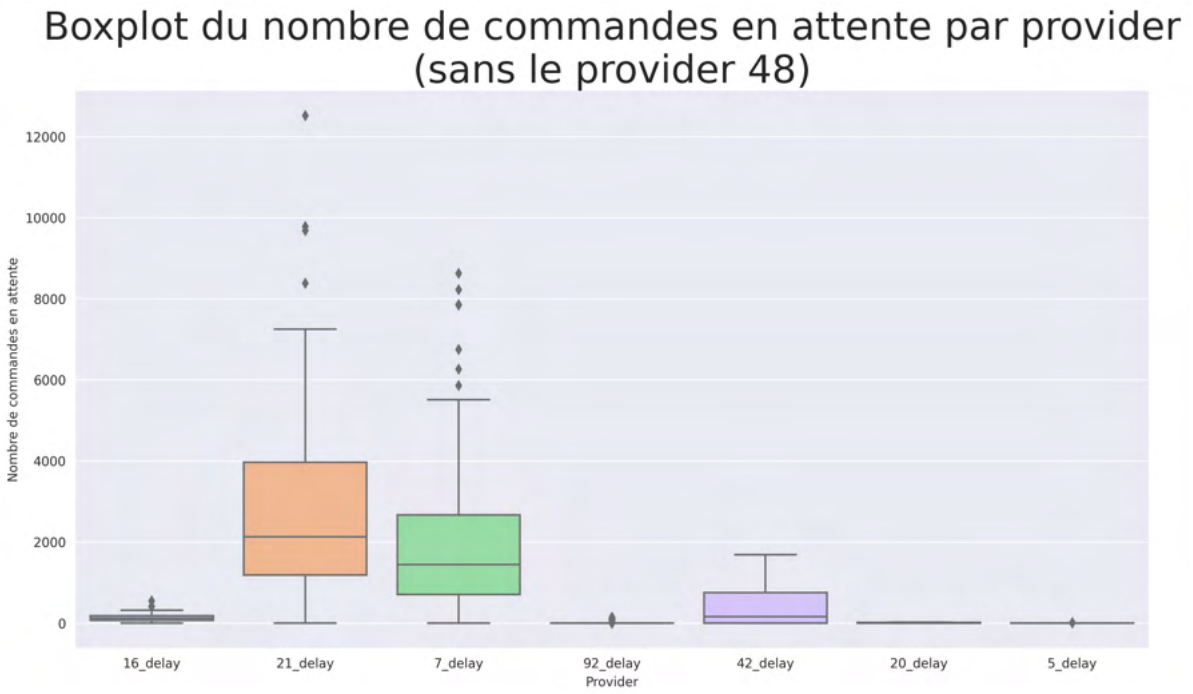
Observations des commandes en attente pour les providers

```
df_temp=df[["date_commande","heure_commande","min_commande",'48_delay', '16_delay',
```

In [ ]:
```
       '92_delay', '42_delay', '20_delay', '5_delay']].drop_duplicates().sort_valu



sns.boxplot(data=df_temp.groupby(["date_commande"]).agg("mean").sort_index().drop(
plt.title("Boxplot du nombre de commandes en attente par provider\n",fontsize=30)
plt.xlabel("Provider")
plt.ylabel("Nombre de commandes en attente")
plt.show()
```



In [ ]:
```
df_temp=df[["date_commande","heure_commande","min_commande",'16_delay', '21_delay',
       '92_delay', '42_delay', '20_delay', '5_delay']].drop_duplicates().sort_valu



sns.boxplot(data=df_temp.groupby(["date_commande"]).agg("mean").sort_index().drop(
plt.title("Boxplot du nombre de commandes en attente par provider\n(sans le provid
plt.xlabel("Provider")
plt.ylabel("Nombre de commandes en attente")
plt.show()
```

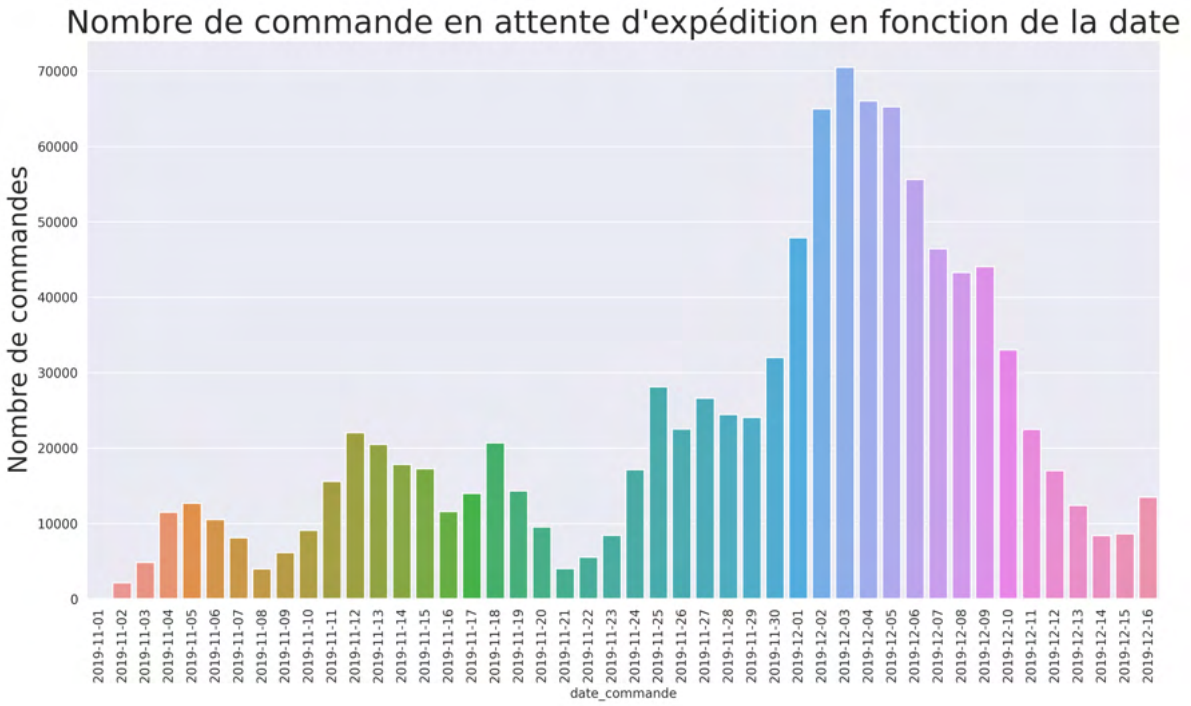


On observe clairement que le volume de commande en attente dépend du volume de commandes traitées par le provider

In [ ]:
```
def subplots_auto_barplot_delay(df,var_filtre, cols,xvar,yvar,rotate=False, title='
    distinct_values=df[var_filtre].unique().tolist()
    Tot = len(distinct_values)
    Rows = Tot // cols
    if Tot % cols != 0:
        Rows += 1

    Position = range(1,Tot + 1)

    fig = plt.figure(1,figsize=(15, 5*Rows))
    for k in range(Tot):
        ax = fig.add_subplot(Rows,cols,Position[k])

        #à modifier en fonction de ce que l'on souhaite afficher

        df_temp=df[df[var_filtre]==distinct_values[k]]

        sns.barplot(data=df_temp, x=xvar,y=yvar, ax=ax)

        if rotate :
            labels=df_temp[xvar]
            ax.set_xticklabels(labels=labels ,rotation = 90)
        ax.set_title("\n"+str(distinct_values[k]),fontsize=20)
    plt.suptitle(title+"\n", fontsize=30)
    plt.tight_layout()
    plt.show()
```

In [ ]:
```
col_name="48_delay"

a=sns.barplot(data=df.groupby(["date_commande"]).agg("mean")[['48_delay', '16_delay
       '92_delay', '42_delay', '20_delay', '5_delay']].reset_index(),
            x="date_commande",
            y=col_name)


labels= df.groupby("date_commande").agg("count").reset_index()["date_commande"].sor
a.set_ylabel(ylabel, fontsize=20)
```

```
a.set_title("Nombre de commande en attente d'expédition en fonction de la date", f
a.set_xticklabels(labels=labels ,rotation = 90)
plt.show()
```



Corrélation temps d'expédition , nombre de commandes en attente

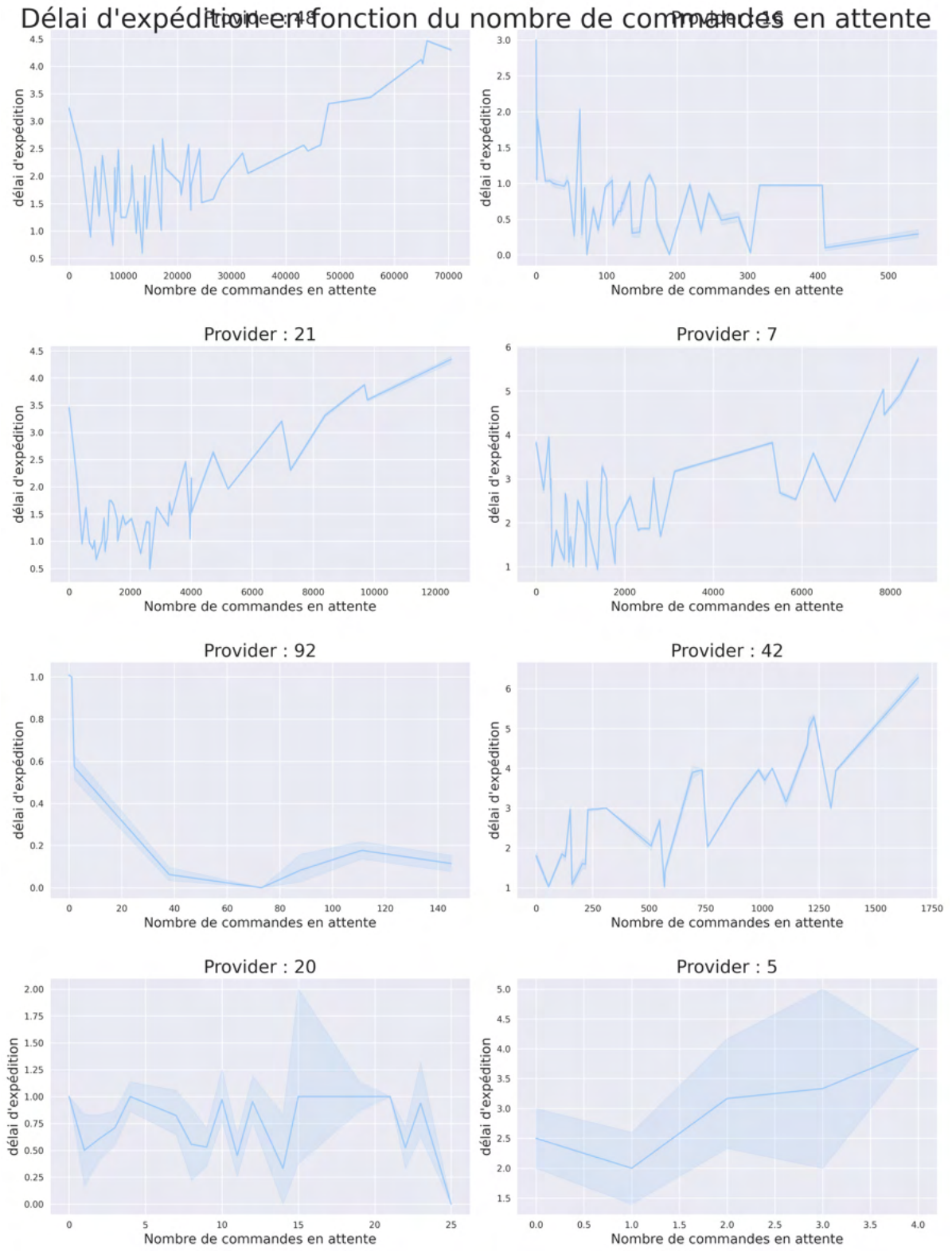
In [ ]:
```
providers=df["provider"].unique()

for provider in providers :
    df_temp=df[df["provider"]==provider]
    print("Corrélations gap x delay"+str(provider)+" = "+str(df_temp["gap"].corr(d
```

```
Corrélations gap x delay48 = 0.7056886761323119
Corrélations gap x delay16 = -0.14882602094083316
Corrélations gap x delay21 = 0.7316566013612563
Corrélations gap x delay7 = 0.6984580328171969
Corrélations gap x delay92 = -0.5641905055991568
Corrélations gap x delay42 = 0.7271638334268107
Corrélations gap x delay20 = -0.00430315485921003
Corrélations gap x delay5 = 0.4953271028037383
```

Corrélations positives fortes pour 48, 21,7,42,5

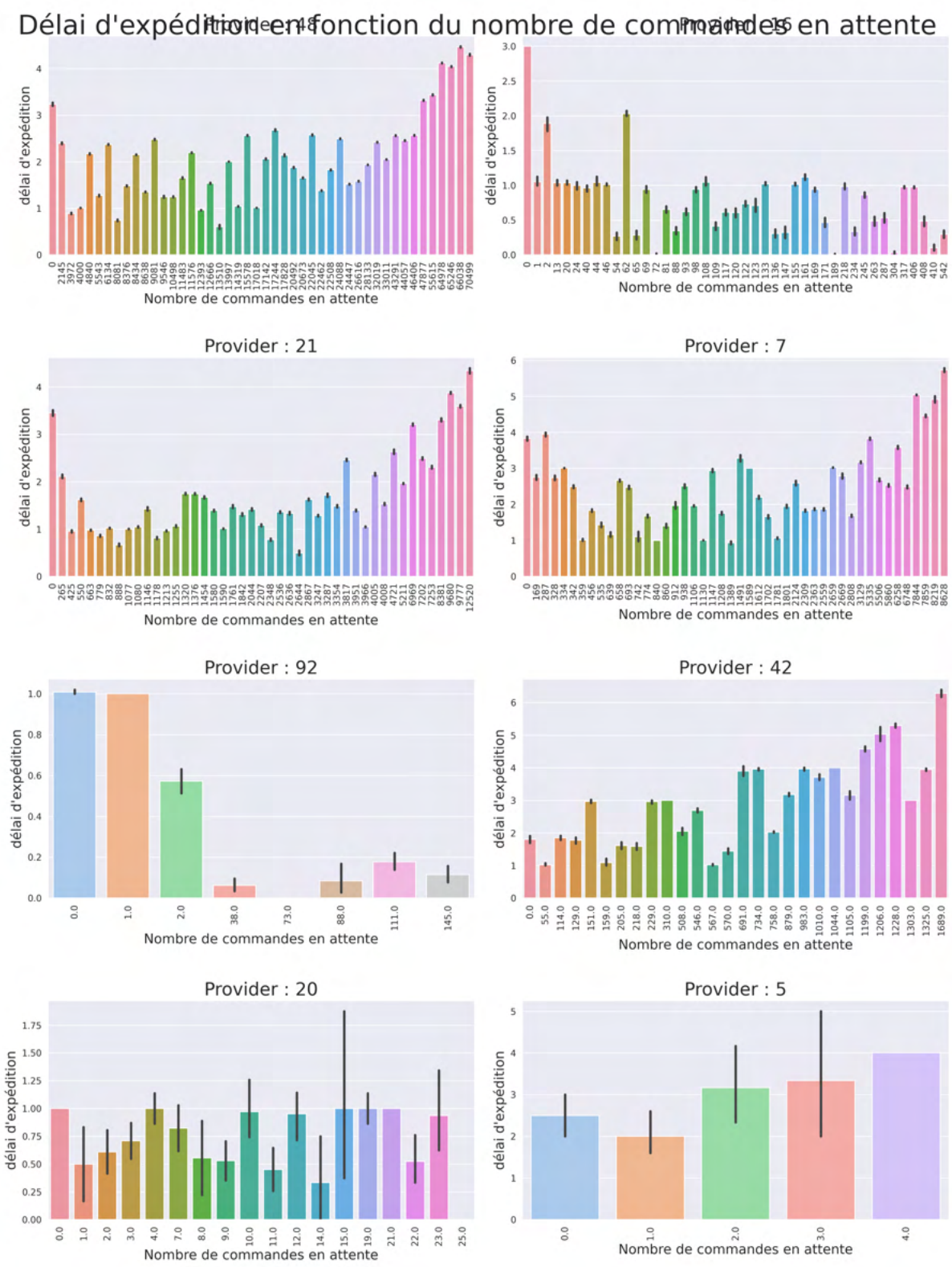
In [ ]:
```
cols=2

providers=df["provider"].unique()
Tot = len(providers)
Rows = Tot // cols

if Tot % cols != 0:
    Rows += 1

Position = range(1,Tot + 1)

fig = plt.figure(1,figsize=(15, 5*Rows))
for k in range(Tot):
    ax = fig.add_subplot(Rows,cols,Position[k])
```

```
    #à modifier en fonction de ce que l'on souhaite afficher
    sns.lineplot(data=df[df["provider"]==providers[k]], x=str(providers[k])+"_delay

    ax.set_title("\nProvider : "+str(providers[k]),fontsize=20)
    ax.set_ylabel("délai d'expédition",fontsize=15)
    ax.set_xlabel("Nombre de commandes en attente",fontsize=15)
plt.suptitle("Délai d'expédition en fonction du nombre de commandes en attente\n",
plt.tight_layout()
plt.show()
```



In [ ]:
```
cols=2

providers=df["provider"].unique()
Tot = len(providers)
```

```
Rows = Tot // cols

if Tot % cols != 0:
    Rows += 1

Position = range(1,Tot + 1)

fig = plt.figure(1,figsize=(15, 5*Rows))
for k in range(Tot):
    ax = fig.add_subplot(Rows,cols,Position[k])

    sns.barplot(data=df[df["provider"]==providers[k]], x=str(providers[k])+"_delay'

    ax.set_title("\nProvider : "+str(providers[k]),fontsize=20)
    ax.set_ylabel("délai d'expédition",fontsize=15)
    ax.set_xlabel("Nombre de commandes en attente",fontsize=15)
    labels=df[df["provider"]==providers[k]][str(providers[k])+"_delay"].drop_duplic
    ax.set_xticklabels(labels=labels ,rotation = 90)
plt.suptitle("Délai d'expédition en fonction du nombre de commandes en attente\n",
plt.tight_layout()
plt.show()
```



Conclusion :

Providers avec des comportements différents

Certains semblent êtres spécialisés dans la livraison rapide

Corrélations entre le nombre d'expédition et le délai d'expédition :

In [ ]:
```
providers=df["provider"].unique()
```

```
for provider in providers :
    df_temp=df[df["provider"]==provider]
    print("Corrélations gap x nb_expe "+str(provider)+" = "+str(df_temp["gap"].cor
```

```
Corrélations gap x nb_expe 48 = 0.2871030779204106
Corrélations gap x nb_expe 16 = -0.2932036632585558
Corrélations gap x nb_expe 21 = 0.3133099233612255
Corrélations gap x nb_expe 7 = 0.12032258410028246
Corrélations gap x nb_expe 92 = nan
Corrélations gap x nb_expe 42 = 0.08415866189466298
Corrélations gap x nb_expe 20 = -0.07367518132801622
Corrélations gap x nb_expe 5 = -0.21102123475450804
```

Corrélations faibles mais existantes pour 48, 16, 21 et 5

Machine learning

Prédiction classique :

On essaye de prédire le délai d'expédition de manière classique pour avoir une idée des performances que l'on peut attendre

Chargement et mise en forme des données :

In [ ]:

```
df=pd.read_csv("./data/data_transformed_min3_nb_jours15.csv")
df["date_expe"]=pd.to_datetime(df["date_expe"])
df["date_commande"]=pd.to_datetime(df["date_commande"])
df=df.sort_values("date_commande")

#Remarque :
#Nous n'utilisons pas le mois ou la semaine de l'année car nous n'avons que 6
#semaines de données et pas une année complète


# Drop des 2 premiers jours car les stocks sont partiellement faux :
df=df.set_index("id_date")

df=df.drop(index=[737240,737241]).reset_index()

#Variables que l'on sélectionne :
features = ['provider', 'heure_commande',
            'min_commande', 'jour_commande','stock_provider','stock_total',
            'nb_expe_provider','nb_expe_total',
             '48_delay', '16_delay', '21_delay', '7_delay',
            '92_delay', '42_delay', '20_delay', '5_delay','48_nb_expe',
       '16_nb_expe', '21_nb_expe', '7_nb_expe', '92_nb_expe', '42_nb_expe',
       '20_nb_expe', '5_nb_expe']

to_predict="gap"

X=df.set_index("date_commande")[features]
y=df[to_predict]

#On converti les colonnes en catégories
for i in range(4):
    X.iloc[:,i]=X.iloc[:,i].astype("category")
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1, test_size

print("X_train  : "+str(X_train.shape))
print("X_test  : "+str(X_test.shape))
print("y_train  : "+str(y_train.shape))
print("y_test  : "+str(y_test.shape))
X_train.head()
```

```
X_train  : (425073, 24)
X_test  : (141692, 24)
y_train  : (425073,)
y_test  : (141692,)
```

Out[ ]:

date_commande	provider	heure_commande	min_commande	jour_commande	stock_provider	st
2019-11-11	7	14	55	0	1147.0	
2019-11-28	48	12	21	3	24447.0	
2019-11-18	48	20	25	0	20673.0	
2019-12-01	48	21	56	6	47877.0	
2019-11-23	48	7	49	5	8434.0	

Modèle de référence

Fit du modèle

In [ ]:
```
path = './Modèles/ref_grad_boost.pkl'
```

In [ ]:
```
from sklearn.ensemble import GradientBoostingRegressor
ref = GradientBoostingRegressor(random_state=0)
ref.fit(X_train, y_train)

#Sauvegarde du modèle :
with open(path, 'wb') as f:
    pickle.dump(ref, f)
```

Accuracy 1 : Accuracy du modèle une classe par jour

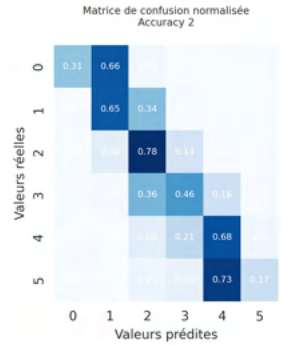
Accuracy 2 : Accuracy du modèle une classe par jour et une classe + de 5 jours

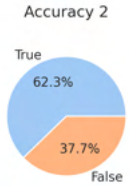
Accuracy 3 : Accuracy du modèle avec chaque classe qui est un encadrement de 2 jours et + 5 de jours

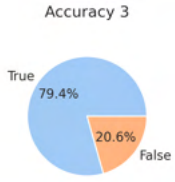
In [ ]:
```
#Lecture du modèle
with open(path, 'rb') as f:
    model = pickle.load(f)

#Analyse des erreurs
error_analysis(model, X_test, y_test)
```

```
Erreur moyenne absolue du modèle : 0.4778 jours
Accuracy 1 : 0.6173
Accuracy 2 : 0.6225
Accuracy 3 : 0.7941
```

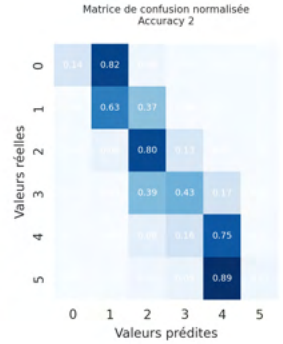
Global

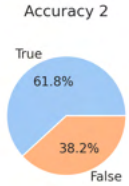


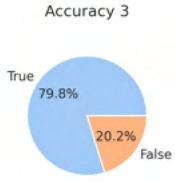




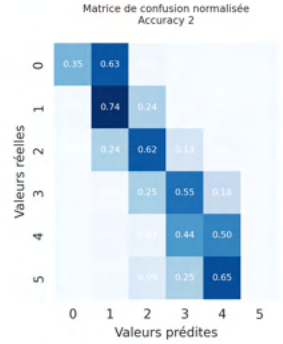
48

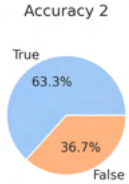


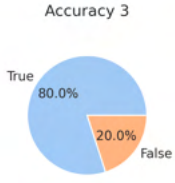




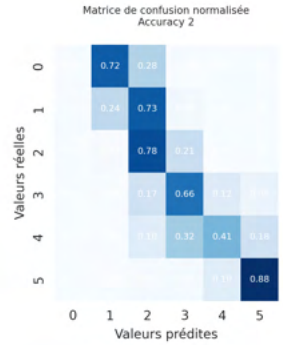
21

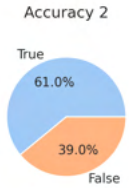


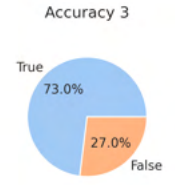




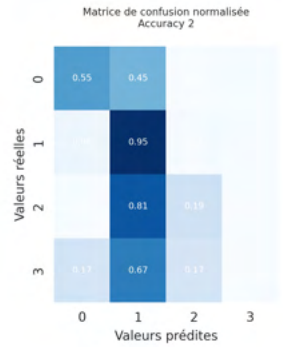
7

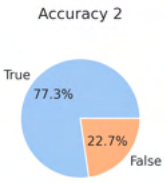


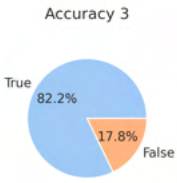




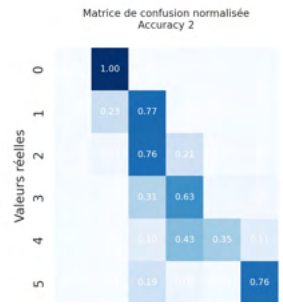
16

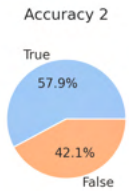


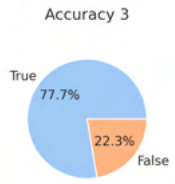


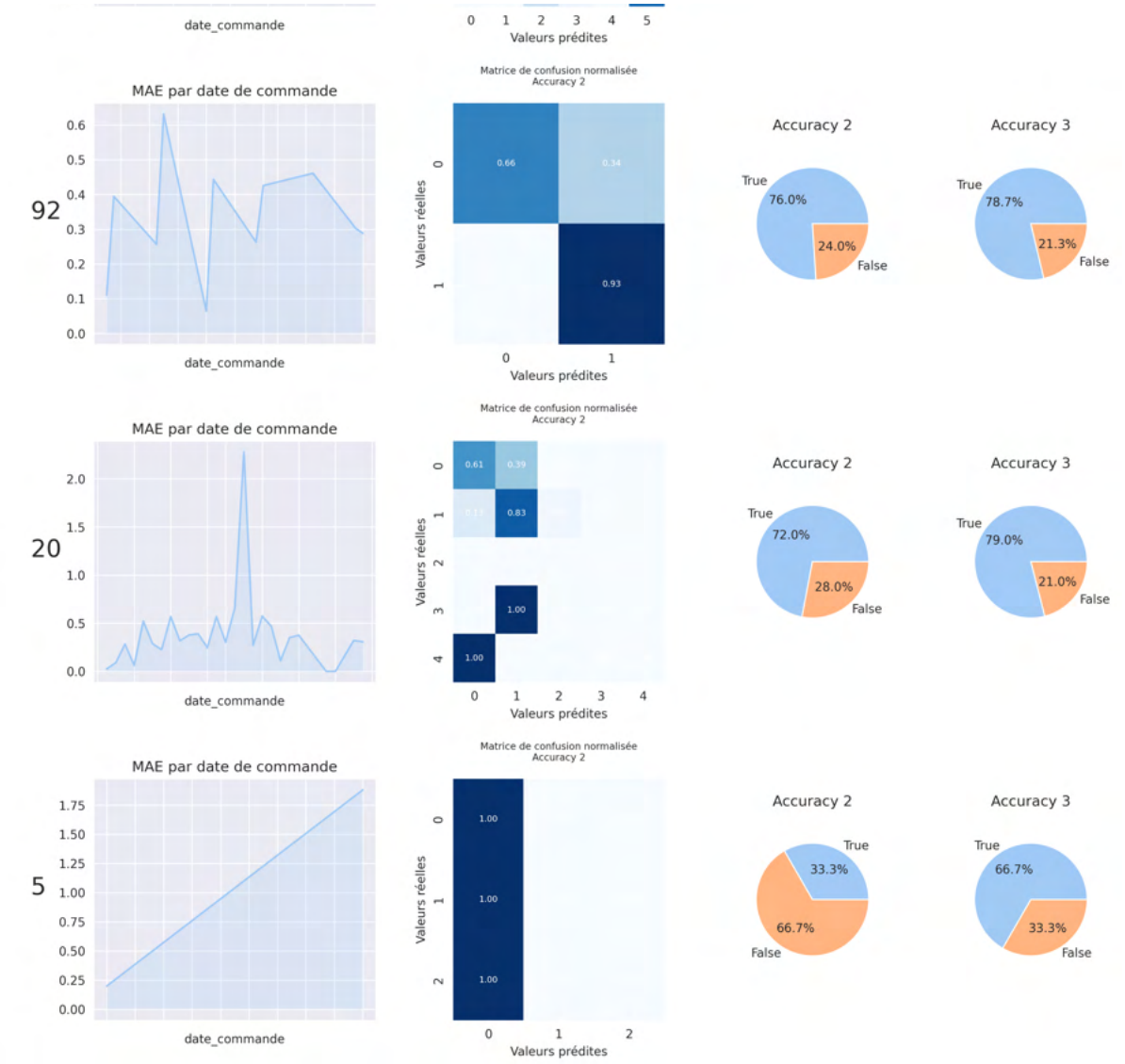


42









Le modèle de référence se trompe en moyenne de 0.478 jour et 61.7% de bonnes prédictions

En faisant des classes de 2 jours et en regroupant toutes les prédiction supérieures ou égales à 5 jours nous obtenons 79% de bonnes réponses

Auto sklearn

In [ ]:
```
path = './Modèles/automl_regression1_1h_all_data.pkl'
```

In [ ]:
```
#On utilise auto sklearn pour générer le modèle
#On fait tourner la librairie pendant 3h puis on observe les résultats
#20h40
import autosklearn.regression
automl = autosklearn.regression.AutoSklearnRegressor(
  time_left_for_this_task=1*60*60,
  per_run_time_limit=300,
  n_jobs=-1,
  resampling_strategy="cv",
  resampling_strategy_arguments={'folds':3},
  metric=autosklearn.metrics.mean_absolute_error
)
automl.fit(X_train, y_train)
```

Out[ ]:
```
AutoSklearnRegressor(metric=mean_absolute_error, n_jobs=-1,
                     per_run_time_limit=300, resampling_strategy='cv',
                     resampling_strategy_arguments={'folds': 3})
```

In [ ]:
```
# save model
automl.refit(X_train,y_train)
with open(path, 'wb') as f:
    pickle.dump(automl, f)
```

In [ ]:
```
#Lecture du modèle
with open(path, 'rb') as f:
    model = pickle.load(f)
model.leaderboard(detailed = True, ensemble_only=False)[["rank","ensemble_weight",'
                                                         ,"data_preprocessors","fea
```

Out[ ]:

model_id	rank	ensemble_weight	type	cost	duration	train_loss	data_prepro
35	1	1.0	gradient_boosting	0.426275	191.562846	0.420373	
20	2	0.0	gradient_boosting	0.430616	232.605614	0.426443	
30	3	0.0	gradient_boosting	0.454860	113.527705	0.454173	
45	4	0.0	decision_tree	0.496486	39.494354	0.496133	
37	5	0.0	gradient_boosting	0.496505	191.794844	0.496165	
6	6	0.0	sgd	0.605582	59.204566	0.605409	
40	7	0.0	decision_tree	0.655639	12.667222	0.655618	
43	8	0.0	gradient_boosting	0.698761	219.587639	0.694502	
28	9	0.0	gradient_boosting	0.909802	18.292460	0.909760	
23	10	0.0	gradient_boosting	0.952707	93.545448	0.952684	

In [ ]:
```
automl_results = pd.DataFrame(model.cv_results_).sort_values(by = 'mean_test_score

automl_results.head(10)
```

Out[ ]:

	mean_test_score	mean_fit_time	params	rank_test_scores	status	param
33	0.426275	191.562846	{'data_preprocessor:__choice__': 'feature_type...	1	Success	
18	0.430616	232.605614	{'data_preprocessor:__choice__': 'feature_type...	2	Success	
28	0.454860	113.527705	{'data_preprocessor:__choice__': 'feature_type...	3	Success	
43	0.496486	39.494354	{'data_preprocessor:__choice__': 'feature_type...	4	Success	
35	0.496505	191.794844	{'data_preprocessor:__choice__': 'feature_type...	5	Success	
4	0.605582	59.204566	{'data_preprocessor:__choice__': 'feature_type...	6	Success	
38	0.655639	12.667222	{'data_preprocessor:__choice__': 'feature_type...	7	Success	
41	0.698761	219.587639	{'data_preprocessor:__choice__': 'feature_type...	8	Success	
26	0.909802	18.292460	{'data_preprocessor:__choice__': 'feature_type...	9	Success	
21	0.952707	93.545448	{'data_preprocessor:__choice__': 'feature_type...	10	Success	

Observations de la composition de l'ensemble du modèle

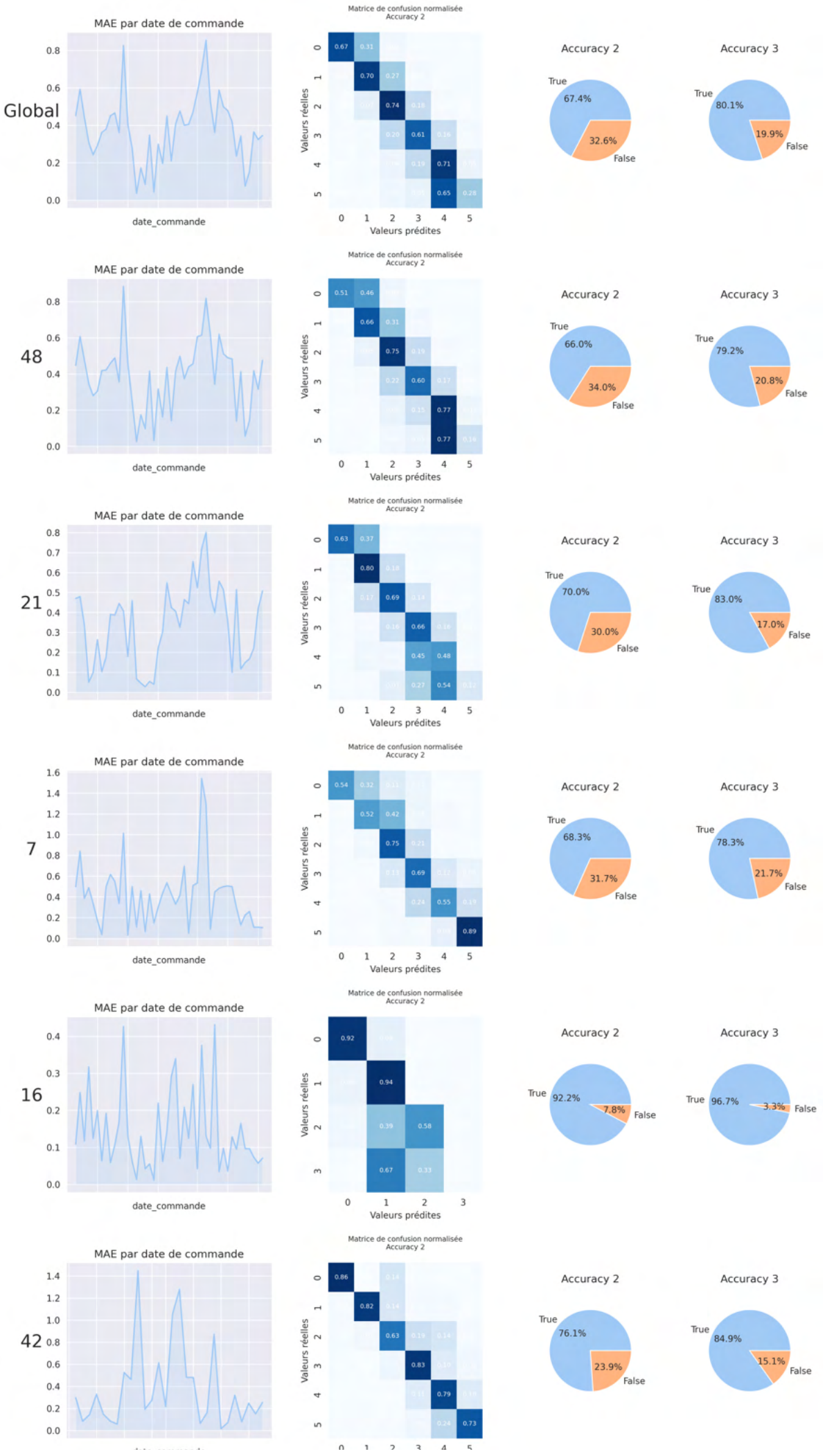
In [ ]:
```
rank = model.leaderboard(detailed = True, ensemble_only=False)
rank["feature_preprocessors"]= rank["feature_preprocessors"].apply(lambda x : x[0])
rank.groupby(["type","feature_preprocessors"]).agg(["mean",'count'])['rank'].sort_
```

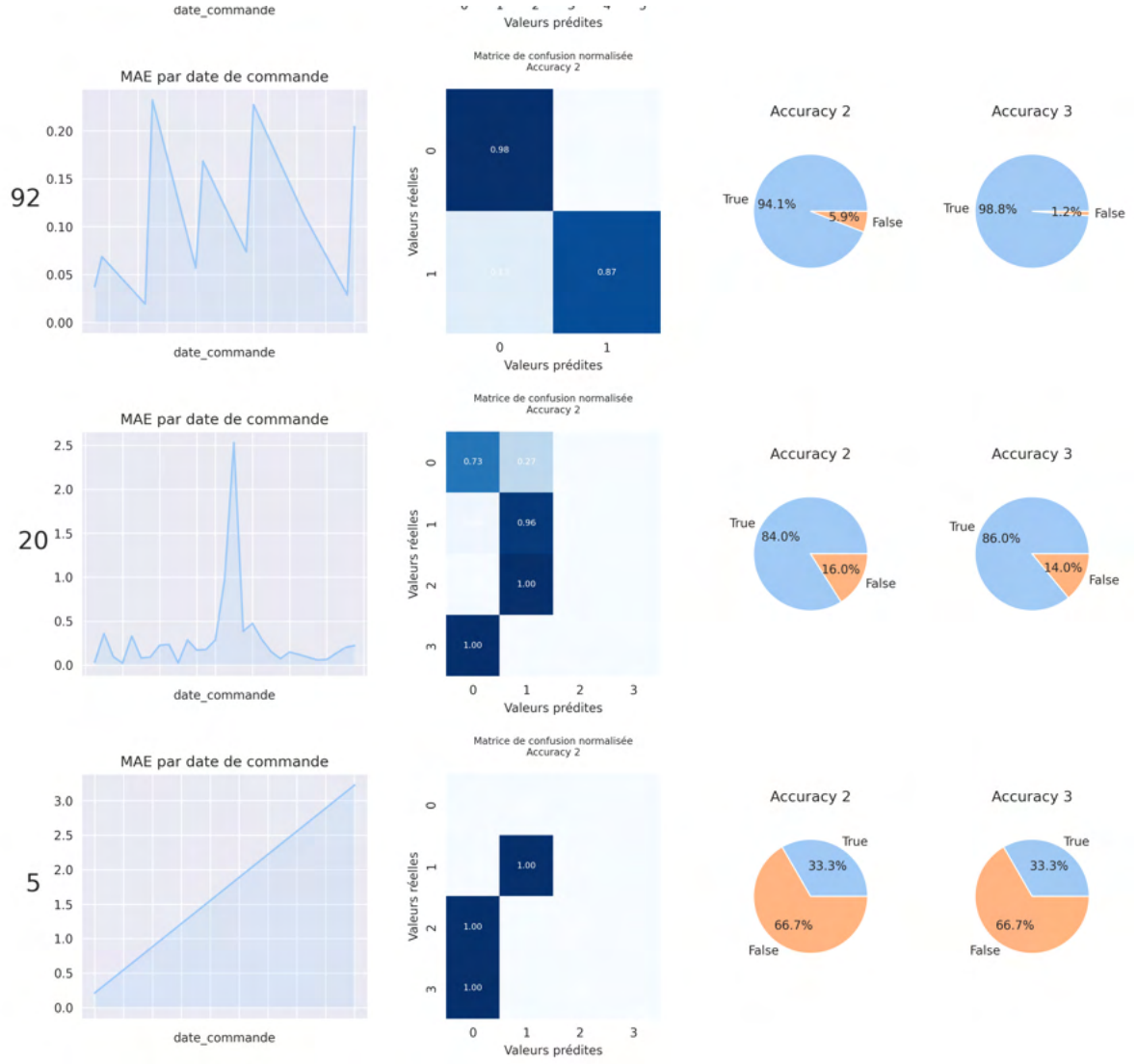
Out[ ]:

		mean	count
type	**feature_preprocessors**		
gradient_boosting	**no_preprocessing**	1.0	1
	feature_agglomeration	3.5	2
decision_tree	**select_percentile_regression**	5.5	2
sgd	**select_rates_regression**	6.0	1
gradient_boosting	**pca**	8.0	1
	select_rates_regression	9.6	5
	polynomial	10.0	1
ard_regression	**polynomial**	14.0	1
mlp	**no_preprocessing**	15.0	1

In [ ]:
```
#Analyse des erreurs
error_analysis(model, X_test, y_test)
```

```
Erreur moyenne absolue du modèle : 0.4228 jours
Accuracy 1 : 0.6678
Accuracy 2 : 0.6737
Accuracy 3 : 0.8007
```


MAE par date de commande
Global
date_commande
Matrice de confusion normalisée
Accuracy 2
Valeurs réelles
Valeurs prédites
Accuracy 2
True
67.4%
32.6%
False
Accuracy 3
True
80.1%
19.9%
False
MAE par date de commande
48
date_commande
Matrice de confusion normalisée
Accuracy 2
Valeurs réelles
Valeurs prédites
Accuracy 2
True
66.0%
34.0%
False
Accuracy 3
True
79.2%
20.8%
False
MAE par date de commande
21
date_commande
Matrice de confusion normalisée
Accuracy 2
Valeurs réelles
Valeurs prédites
Accuracy 2
True
70.0%
30.0%
False
Accuracy 3
True
83.0%
17.0%
False
MAE par date de commande
7
date_commande
Matrice de confusion normalisée
Accuracy 2
Valeurs réelles
Valeurs prédites
Accuracy 2
True
68.3%
31.7%
False
Accuracy 3
True
78.3%
21.7%
False
MAE par date de commande
16
date_commande
Matrice de confusion normalisée
Accuracy 2
Valeurs réelles
Valeurs prédites
Accuracy 2
True
92.2%
7.8%
False
Accuracy 3
True
96.7%
3.3%
False
MAE par date de commande
42
Matrice de confusion normalisée
Accuracy 2
Valeurs réelles
Accuracy 2
True
76.1%
23.9%
False
Accuracy 3
True
84.9%
15.1%
False



Les résultats du modèle sont plutôt bon , nous observons qu'il a de très bonnes prédictions pour les providers 92, 16 et 20. En revanche il n'arrive pas à prédire le provider 5

On observe que les modèles avec les meilleurs résultats sont : gradient boosting , knn et decision tree

Les features preprocessors qui semblent le mieux fonctionner sont : feature_agglomeration, "no_preprocessing",select_rates_regression et pca

On relance l'algorithme en lui demandant de se focaliser uniquement sur ces modèles , pour espérer avoir un ensemble plus optimisé

Deuxième essais automl

In [ ]:
```
df=pd.read_csv("./data/data_transformed_min3_nb_jours15.csv")
df["date_expe"]=pd.to_datetime(df["date_expe"])
df["date_commande"]=pd.to_datetime(df["date_commande"])
df=df.sort_values("date_commande")

#Remarque :
#Nous n'utilisons pas le mois ou la semaine de l'année car nous n'avons que 6
#semaines de données et pas une année complète
```

```
# Drop des 2 premiers jours car les stocks sont partiellement faux :
df=df.set_index("id_date")

df=df.drop(index=[737240,737241]).reset_index()

#Variables que l'on sélectionne :
features = ['provider', 'heure_commande',
            'min_commande', 'jour_commande','stock_provider','stock_total',
            'nb_expe_provider','nb_expe_total',
             '48_delay', '16_delay', '21_delay', '7_delay',
            '92_delay', '42_delay', '20_delay', '5_delay','48_nb_expe',
       '16_nb_expe', '21_nb_expe', '7_nb_expe', '92_nb_expe', '42_nb_expe',
       '20_nb_expe', '5_nb_expe']

to_predict="gap"

X=df.set_index("date_commande")[features]
y=df[to_predict]

#On converti les colonnes en catégories
for i in range(4):
    X.iloc[:,i]=X.iloc[:,i].astype("category")

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1, test_size

print("X_train  : "+str(X_train.shape))
print("X_test  : "+str(X_test.shape))
print("y_train  : "+str(y_train.shape))
print("y_test  : "+str(y_test.shape))
X_train.head()
```

```
X_train  : (425073, 24)
X_test  : (141692, 24)
y_train  : (425073,)
y_test  : (141692,)
```

Out[ ]:

date_commande	provider	heure_commande	min_commande	jour_commande	stock_provider	st
2019-11-11	7	14	55	0	1147.0	
2019-11-28	48	12	21	3	24447.0	
2019-11-18	48	20	25	0	20673.0	
2019-12-01	48	21	56	6	47877.0	
2019-11-23	48	7	49	5	8434.0	

In [ ]:
```
path = './Modèles/automl_regression2_1h_all_data.pkl'
```

In [ ]:
```
#On utilise auto sklearn pour générer le modèle
import autosklearn.regression
automl = autosklearn.regression.AutoSklearnRegressor(
  time_left_for_this_task=3*60*60,
  per_run_time_limit=420,
  n_jobs=-1,
  resampling_strategy="cv",
  resampling_strategy_arguments={'folds':3},
  metric=autosklearn.metrics.mean_absolute_error,
  include = {
      'regressor':  ["gradient_boosting","k_nearest_neighbors","decision_tree"],
```

```
      'feature_preprocessor': ['no_preprocessing','feature_agglomeration',"pca","s
  }
)
automl.fit(X_train, y_train)

automl.refit(X_train,y_train)

# save model
import pickle
with open(path, 'wb') as f:
    pickle.dump(automl, f)
```

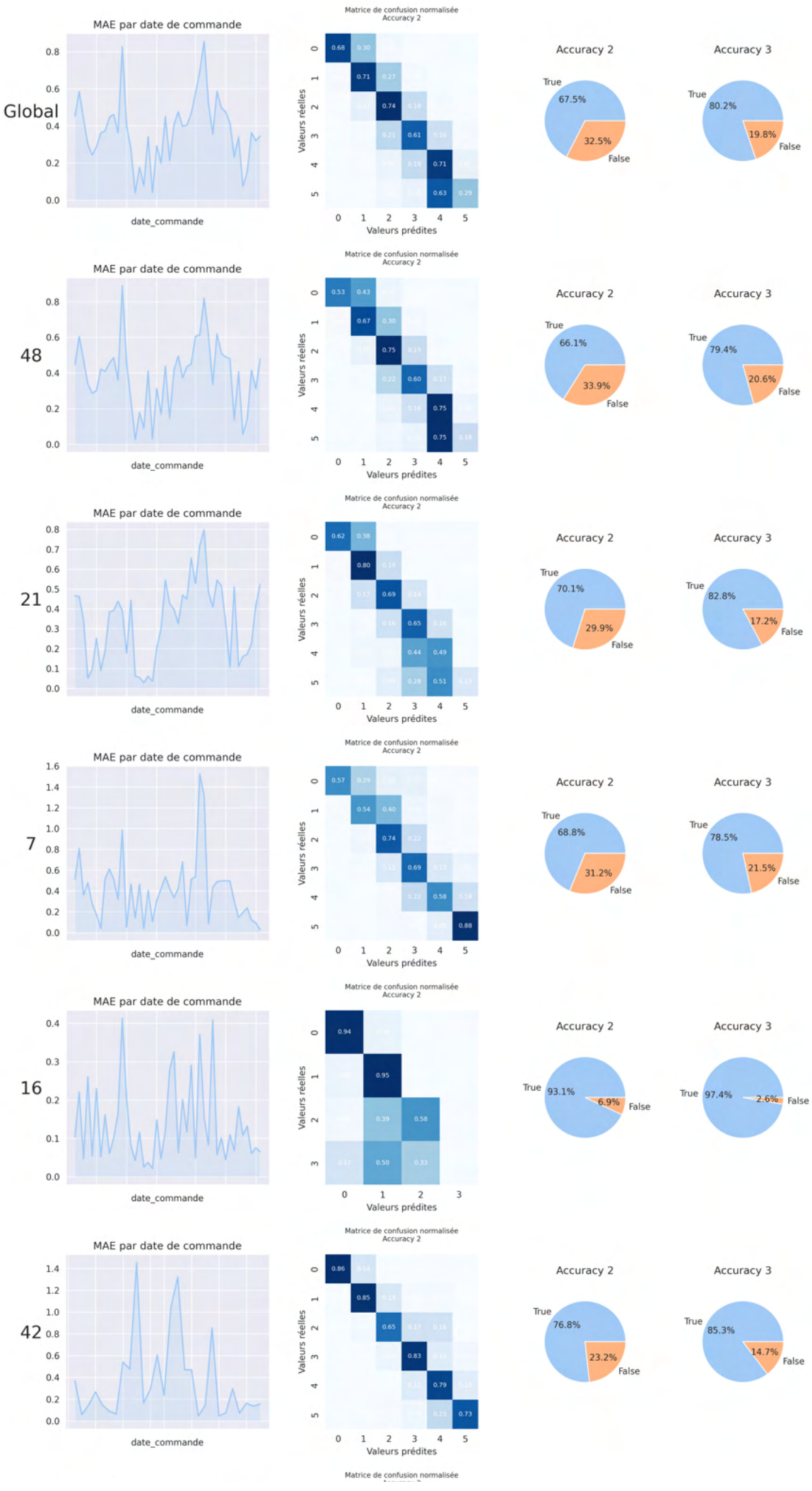
In [ ]:
```
model.leaderboard(detailed = True, ensemble_only=False)[["rank","ensemble_weight",'
                                                         ,"data_preprocessors","fea
```

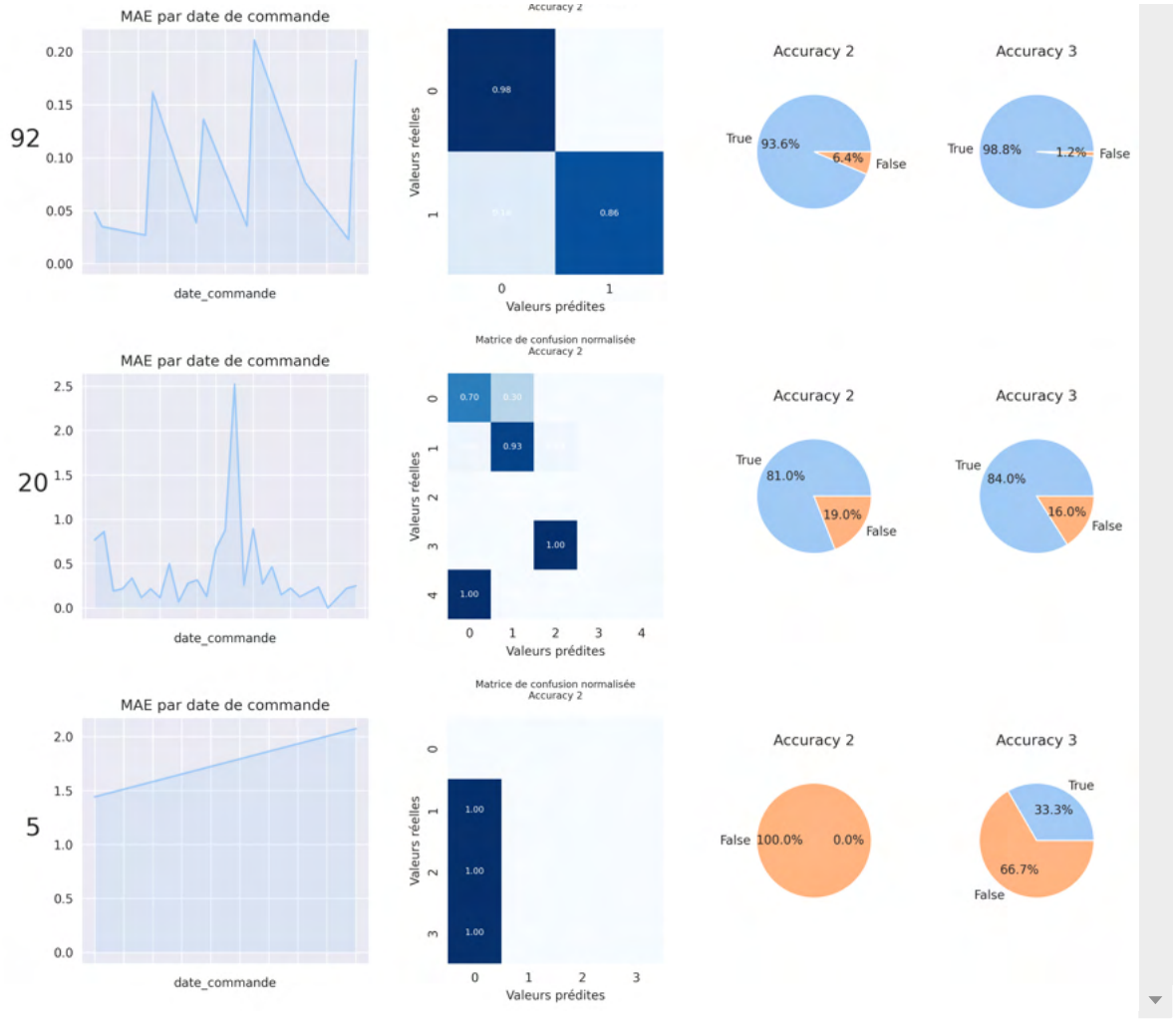
Out[ ]:

model_id	rank	ensemble_weight	type	cost	duration	train_loss	data_prepro
35	1	1.0	gradient_boosting	0.426275	191.562846	0.420373	
20	2	0.0	gradient_boosting	0.430616	232.605614	0.426443	
30	3	0.0	gradient_boosting	0.454860	113.527705	0.454173	
45	4	0.0	decision_tree	0.496486	39.494354	0.496133	
37	5	0.0	gradient_boosting	0.496505	191.794844	0.496165	
6	6	0.0	sgd	0.605582	59.204566	0.605409	
40	7	0.0	decision_tree	0.655639	12.667222	0.655618	
43	8	0.0	gradient_boosting	0.698761	219.587639	0.694502	
28	9	0.0	gradient_boosting	0.909802	18.292460	0.909760	
23	10	0.0	gradient_boosting	0.952707	93.545448	0.952684	

In [ ]:
```
# load model
import pickle
with open(path, 'rb') as f:
    model = pickle.load(f)

error_analysis(model,X_test,y_test)
```

```
Erreur moyenne absolue du modèle : 0.4211 jours
Accuracy 1 : 0.6688
Accuracy 2 : 0.6746
Accuracy 3 : 0.802
```


Global
MAE par date de commande
date_commande
Matrice de confusion normalisée
Accuracy 2
Valeurs réelles
Valeurs prédites
Accuracy 2
True
67.5%
32.5%
False
Accuracy 3
True
80.2%
19.8%
False
48
MAE par date de commande
date_commande
Matrice de confusion normalisée
Accuracy 2
Valeurs réelles
Valeurs prédites
Accuracy 2
True
66.1%
33.9%
False
Accuracy 3
True
79.4%
20.6%
False
21
MAE par date de commande
date_commande
Matrice de confusion normalisée
Accuracy 2
Valeurs réelles
Valeurs prédites
Accuracy 2
True
70.1%
29.9%
False
Accuracy 3
True
82.8%
17.2%
False
7
MAE par date de commande
date_commande
Matrice de confusion normalisée
Accuracy 2
Valeurs réelles
Valeurs prédites
Accuracy 2
True
68.8%
31.2%
False
Accuracy 3
True
78.5%
21.5%
False
16
MAE par date de commande
date_commande
Matrice de confusion normalisée
Accuracy 2
Valeurs réelles
Valeurs prédites
Accuracy 2
True
93.1%
6.9%
False
Accuracy 3
True
97.4%
2.6%
False
42
MAE par date de commande
date_commande
Matrice de confusion normalisée
Accuracy 2
Valeurs réelles
Valeurs prédites
Accuracy 2
True
76.8%
23.2%
False
Accuracy 3
True
85.3%
14.7%
False

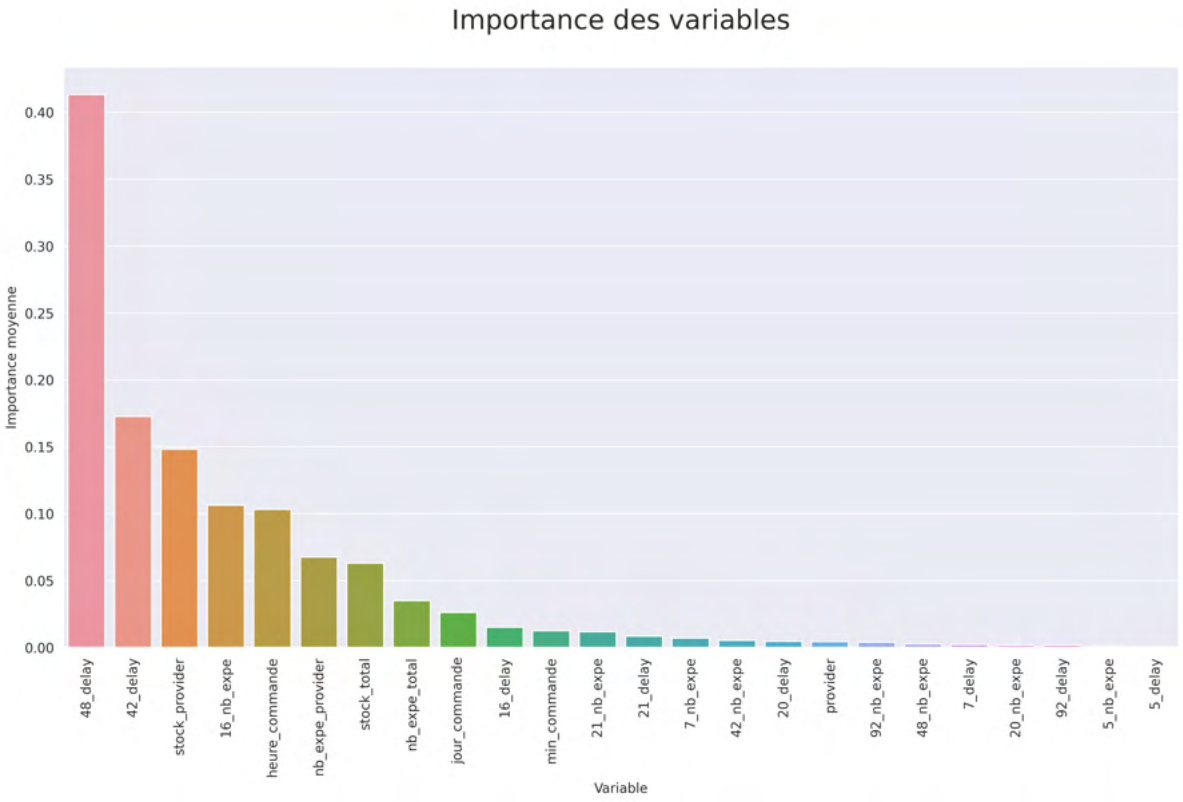


Feature importance analysis

Permutation importance :

"Since auto-sklearn implements the scikit-learn interface, it can be used with the scikit-learn's inspection module. So, now we first look at the permutation importance, which defines the decrease in a model score when a given feature is randomly permuted. So, the higher the score, the more does the model's predictions depend on this feature."
https://automl.github.io/auto-sklearn/master/examples/40_advanced/example_inspect_predictions.html

In [ ]:
```
from sklearn.inspection import plot_partial_dependence, permutation_importance
r = permutation_importance(model, X_test, y_test, n_repeats=5,n_jobs=-1, random_sta
sort_idx = r.importances_mean.argsort()
temp=pd.DataFrame(r["importances_mean"], index= X.columns.tolist(), columns=["impor
temp["importances_mean"]=temp["importances_mean"]*(-1)

sns.barplot(data=temp.sort_values("importances_mean",ascending=False).reset_index(
            x="Variable",
            y="Importance moyenne")
plt.xticks(rotation=90)
plt.title("Importance des variables\n",fontsize=20)
plt.show()
```



Test embedding des providers :

Mise en forme des données

On prend les 34 premiers jours pour le train et le reste pour le test

In [5]:
```
#Paramètres :
to_index=["provider","date_commande"]
to_drop=["expedition","jour_expe","id_date","id_date_expe","date_expe"]
to_predict="gap"
features=["heure_commande","min_commande","jour_commande",
          #Variables des providers
          "stock_provider","jours_act_provider","nb_expe_provider","min_provider",
          "Q3_provider","max_provider","jour_0","jour_1","jour_2","jour_3","jour_4
          "coef_expe_rapide","coef_expe_normale","coef_expe_lente",
          #Variables totales :
          "stock_total","jours_act_total","nb_expe_total","min_total","Q1_total","
          "jour_1_tous_provider","jour_2_tous_provider","jour_3_tous_provider","jou
          "jour_6_tous_provider"]

to_encode=["jour_commande","heure_commande"]

data=pd.read_csv("./data/data_transformed_min3_nb_jours15.csv")
data=data.sort_values(["id_date","heure_commande","min_commande","id_date_expe"])
df_index= data[to_index]

jours_train=34#On prend les 34 premières journées du dataframe dans le jeu de trai
index_split_train_test= len(data[(data["id_date"]-data["id_date"].min())<=jours_tra

X=data.set_index(to_index).drop(columns=to_drop)[features]
X=pd.get_dummies(X, columns=to_encode)

y=data[to_predict]
```

```
print("X : ",X.shape)
print("y : ",y.shape)

X_train,X_test =X.iloc[:index_split_train_test], X.iloc[index_split_train_test:]
y_train, y_test = y.iloc[:index_split_train_test], y.iloc[index_split_train_test:]

print("X_train : ",X_train.shape)
print("X_test : ",X_test.shape)
print("y_train : ",y_train.shape)
print("y_test : ",y_test.shape)
```

```
X :  (572807, 65)
y :  (572807,)
X_train :  (456945, 65)
X_test :  (115862, 65)
y_train :  (456945,)
y_test :  (115862,)
```

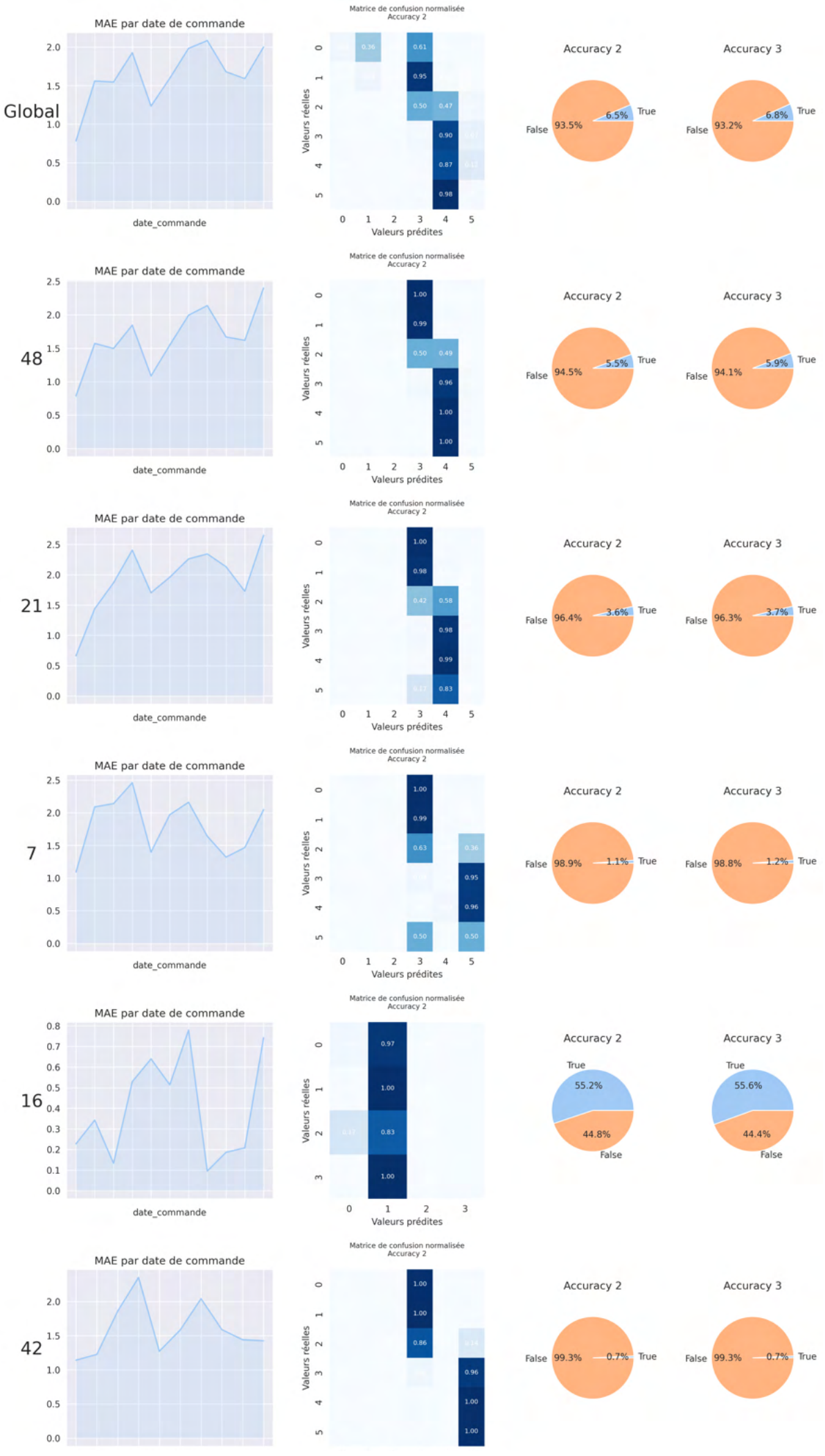
Modèle de référence :

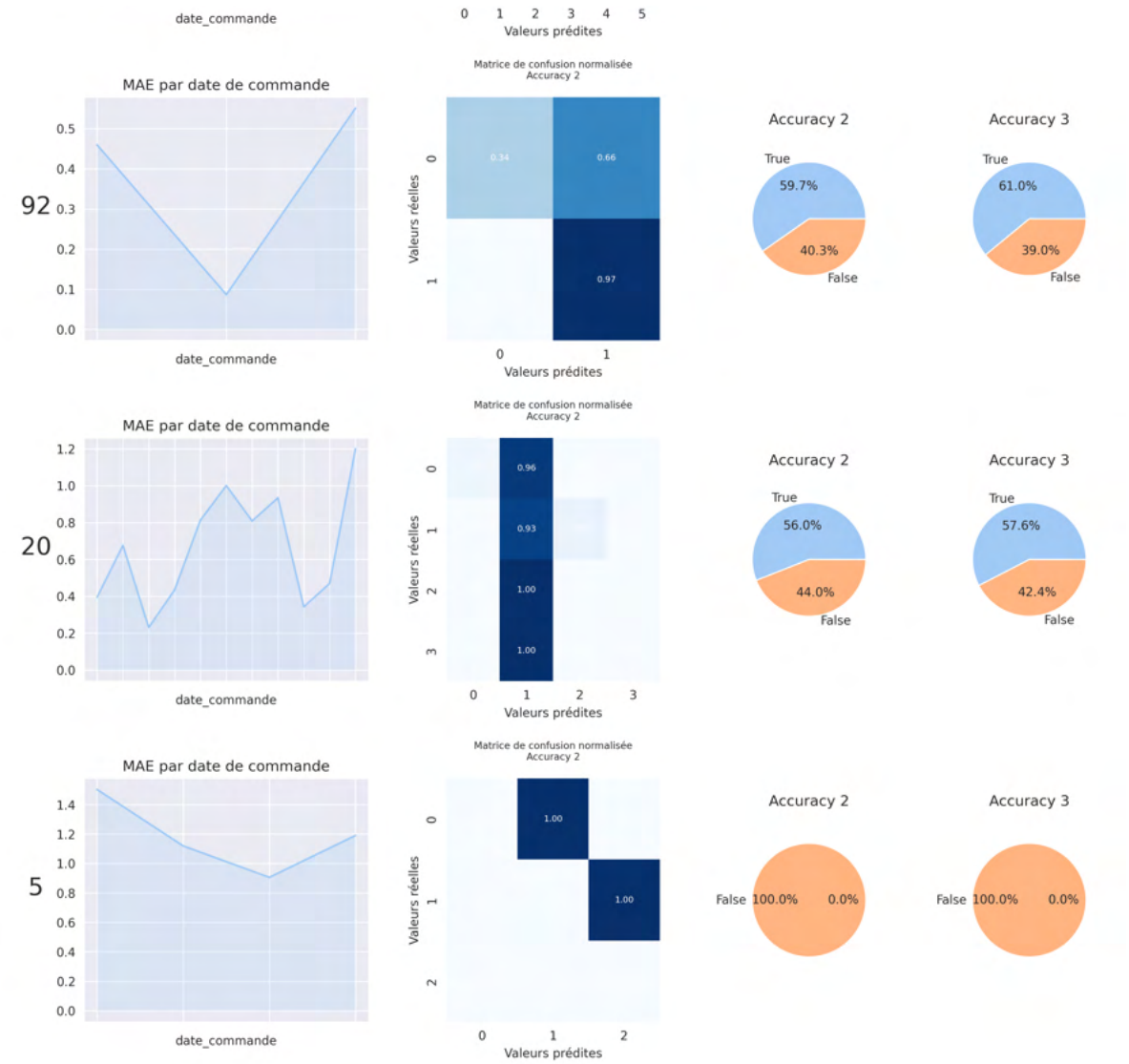
In [ ]:
```
path="./Modèles/ref_grad_boost_embed_ts.csv"
```

In [ ]:
```
from sklearn.ensemble import GradientBoostingRegressor
ref = GradientBoostingRegressor(random_state=0)
ref.fit(X_train, y_train)

#Sauvegarde du modèle :
with open(path, 'wb') as f:
    pickle.dump(ref, f)
```

In [ ]:
```
#Lecture du modèle
with open(path, 'rb') as f:
    model = pickle.load(f)

#Analyse des erreurs
error_analysis(model, X_test, y_test)
```

```
Erreur moyenne absolue du modèle : 1.6185 jours
Accuracy 1 : 0.0646
Accuracy 2 : 0.0646
Accuracy 3 : 0.0678
```


Global
MAE par date de commande
date_commande
Matrice de confusion normalisée
Accuracy 2
Valeurs réelles
Valeurs prédites
Accuracy 2
False
True
93.5%
6.5%
Accuracy 3
93.2%
6.8%
48
94.5%
5.5%
94.1%
5.9%
21
96.4%
3.6%
96.3%
3.7%
7
98.9%
1.1%
98.8%
1.2%
16
55.2%
44.8%
55.6%
44.4%
42
99.3%
0.7%



Le modèle ne fonctionne pas du tout, peut être à cause du nombre de variables

Deep learning

Mise en forme des données

In [6]:
```
#Paramètres :
to_index=["provider","date_commande"]
to_drop=["expedition","jour_expe","id_date","id_date_expe","date_expe"]
to_predict="gap"
features=["heure_commande","min_commande","jour_commande",
          #Variables des providers
          "stock_provider","jours_act_provider","nb_expe_provider","min_provider",
          "Q3_provider","max_provider","jour_0","jour_1","jour_2","jour_3","jour_4
          "coef_expe_rapide","coef_expe_normale","coef_expe_lente",
          #Variables totales :
          "stock_total","jours_act_total","nb_expe_total","min_total","Q1_total","
          "jour_1_tous_provider","jour_2_tous_provider","jour_3_tous_provider","jou
          "jour_6_tous_provider"]

to_encode=["jour_commande","heure_commande"]

data=pd.read_csv("./data/data_transformed_min3_nb_jours15.csv")
data=data.sort_values(["id_date","heure_commande","min_commande","id_date_expe"])
df_index= data[to_index]
```

```
jours_train=34#On prend les 34 premières journées du dataframe dans le jeu de trai
index_split_train_test= len(data[(data["id_date"]-data["id_date"].min())<=jours_tra

X=data.set_index(to_index).drop(columns=to_drop)[features]
X=pd.get_dummies(X, columns=to_encode)

y=data[to_predict]

print("X : ",X.shape)
print("y : ",y.shape)

X_train,X_test =X.iloc[:index_split_train_test], X.iloc[index_split_train_test:]
y_train, y_test = y.iloc[:index_split_train_test], y.iloc[index_split_train_test:]

print("X_train : ",X_train.shape)
print("X_test : ",X_test.shape)
print("y_train : ",y_train.shape)
print("y_test : ",y_test.shape)
```

```
X :  (572807, 65)
y :  (572807,)
X_train :  (456945, 65)
X_test :  (115862, 65)
y_train :  (456945,)
y_test :  (115862,)
```

Normalisation :

MinMaxScaler

In [7]:
```
scaler = preprocessing.MinMaxScaler()
scaler.fit(X)
X_train_scaled =scaler.transform(X_train)
X_test_scaled=scaler.transform(X_test)
```

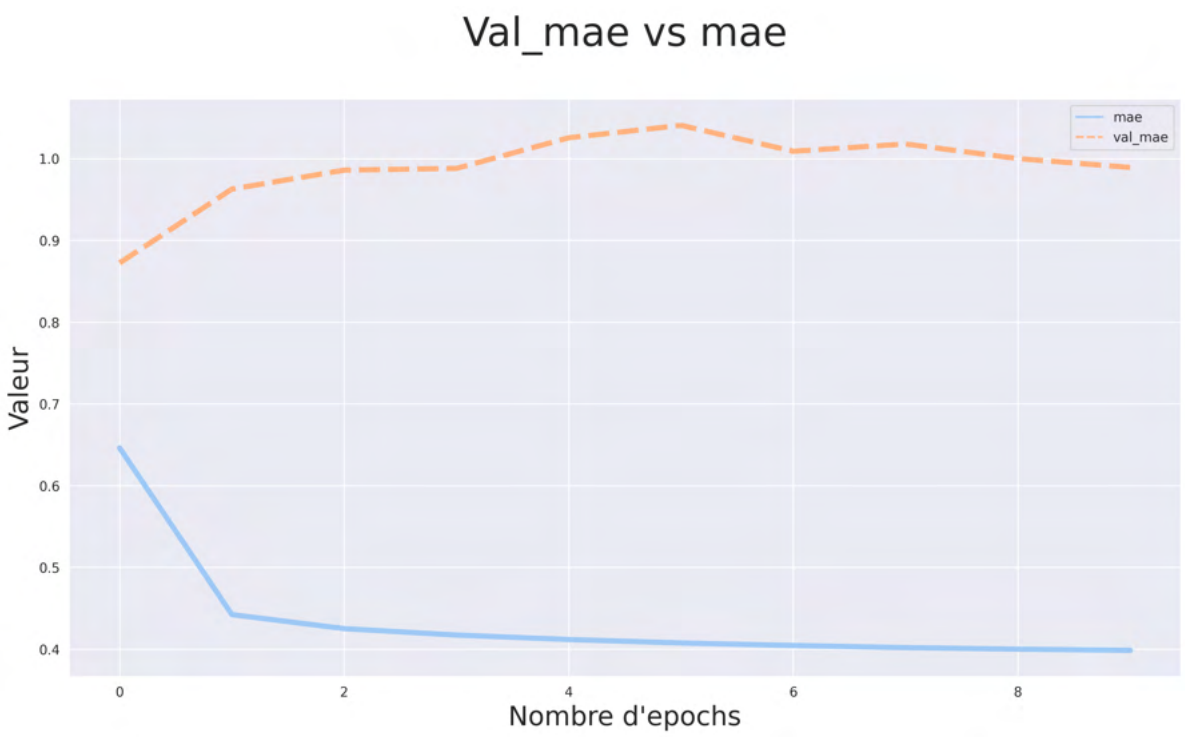
Modèle de référence

In [8]:
```
model1=keras.Sequential([
        keras.layers.Dense(8, activation='relu',input_shape=(len(X_train.columns),
        #keras.layers.Dropout(0.5),
        keras.layers.Dense(8, activation='relu',kernel_regularizer=regularizers.L1
        keras.layers.Dense(8, activation='relu',kernel_regularizer=regularizers.L1
        keras.layers.Dense(8, activation='relu',kernel_regularizer=regularizers.L1
        keras.layers.Dense(1, activation ='relu')
])

model1.compile(optimizer='Adam',
                       loss='mae',
                       metrics=['mae','mse'], )

history = model1.fit(X_train_scaled,y_train, epochs =10,validation_split=0.2, batch
results= pd.DataFrame(history.history).reset_index()
results = results.rename(columns={"index":"epochs"}).set_index("epochs")
```

```
Epoch 1/10
714/714 [==============================] - 4s 4ms/step - loss: 0.6605 - mae: 0.646
0 - mse: 0.8079 - val_loss: 0.8870 - val_mae: 0.8723 - val_mse: 1.2536
Epoch 2/10
714/714 [==============================] - 2s 3ms/step - loss: 0.4570 - mae: 0.442
2 - mse: 0.4388 - val_loss: 0.9775 - val_mae: 0.9626 - val_mse: 1.6088
Epoch 3/10
714/714 [==============================] - 2s 3ms/step - loss: 0.4399 - mae: 0.425
0 - mse: 0.4332 - val_loss: 1.0007 - val_mae: 0.9858 - val_mse: 1.6797
Epoch 4/10
714/714 [==============================] - 2s 3ms/step - loss: 0.4321 - mae: 0.417
2 - mse: 0.4305 - val_loss: 1.0028 - val_mae: 0.9879 - val_mse: 1.6860
Epoch 5/10
714/714 [==============================] - 2s 3ms/step - loss: 0.4266 - mae: 0.411
7 - mse: 0.4283 - val_loss: 1.0403 - val_mae: 1.0254 - val_mse: 1.8373
Epoch 6/10
714/714 [==============================] - 2s 3ms/step - loss: 0.4225 - mae: 0.407
5 - mse: 0.4260 - val_loss: 1.0556 - val_mae: 1.0406 - val_mse: 1.9141
Epoch 7/10
714/714 [==============================] - 3s 4ms/step - loss: 0.4195 - mae: 0.404
4 - mse: 0.4240 - val_loss: 1.0240 - val_mae: 1.0089 - val_mse: 1.7678
Epoch 8/10
714/714 [==============================] - 3s 4ms/step - loss: 0.4171 - mae: 0.401
9 - mse: 0.4219 - val_loss: 1.0331 - val_mae: 1.0178 - val_mse: 1.8253
Epoch 9/10
714/714 [==============================] - 3s 5ms/step - loss: 0.4153 - mae: 0.400
0 - mse: 0.4207 - val_loss: 1.0153 - val_mae: 0.9999 - val_mse: 1.7570
Epoch 10/10
714/714 [==============================] - 2s 3ms/step - loss: 0.4138 - mae: 0.398
5 - mse: 0.4195 - val_loss: 1.0045 - val_mae: 0.9891 - val_mse: 1.7173
```

In [9]:

```
sns.lineplot(data=results[["mae","val_mae"]],lw=4)
plt.title("Val_mae vs mae\n", fontsize=30)
plt.ylabel("Valeur",fontsize=20)
plt.xlabel("Nombre d'epochs",fontsize=20)
plt.show()
```



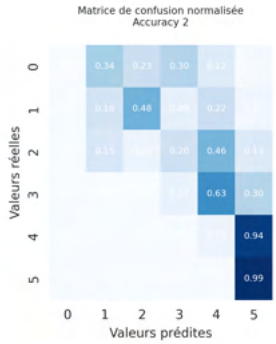
In [10]:

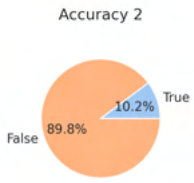
```
X_test_date=pd.DataFrame(X_test_scaled,columns=X_test.columns.tolist())
X_test_date["date_commande"]=df_index.iloc[index_split_train_test:,1].tolist()
X_test_date["provider"]=df_index.iloc[index_split_train_test:,0].tolist()
```

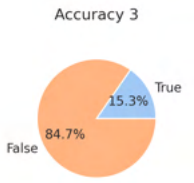
```
X_test_date["date_commande"]=pd.to_datetime(X_test_date["date_commande"])
X_test_date=X_test_date.set_index(["date_commande","provider"])

error_analysis(model1, X_test_date, y_test)
```

```
Erreur moyenne absolue du modèle : 1.5606 jours
Accuracy 1 : 0.0997
Accuracy 2 : 0.1023
Accuracy 3 : 0.1531
```

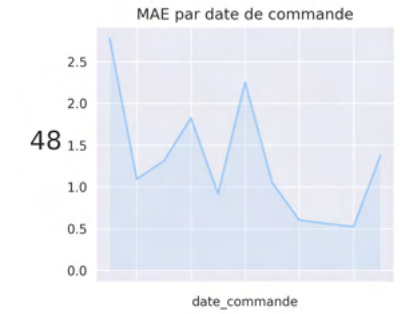
Global

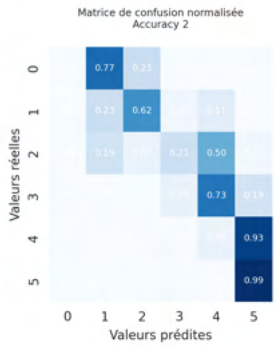


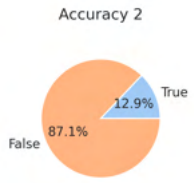


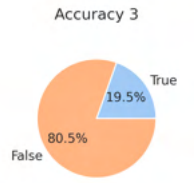


48

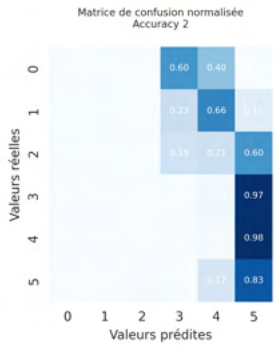


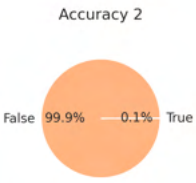


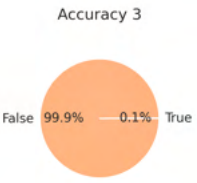




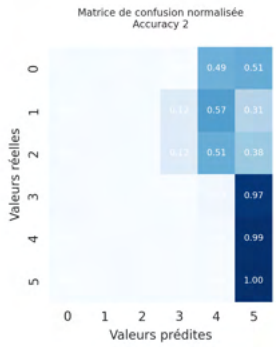
21

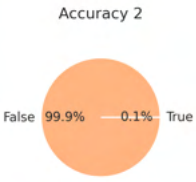


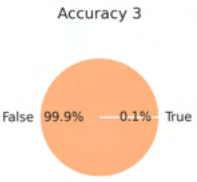




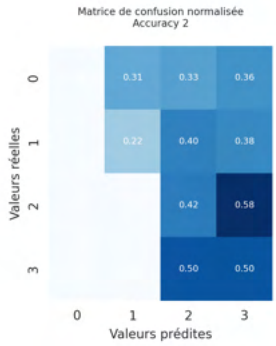
7

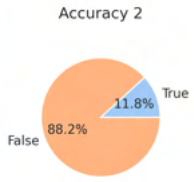


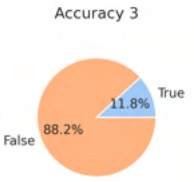




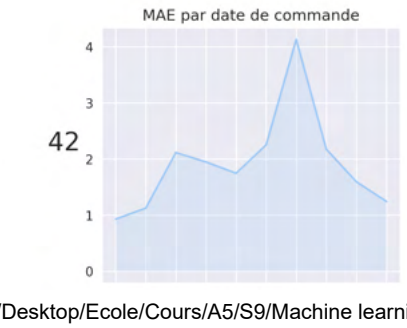
16

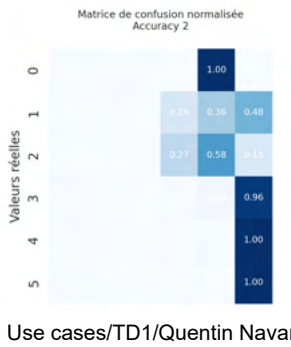


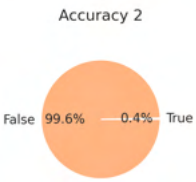


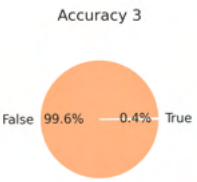


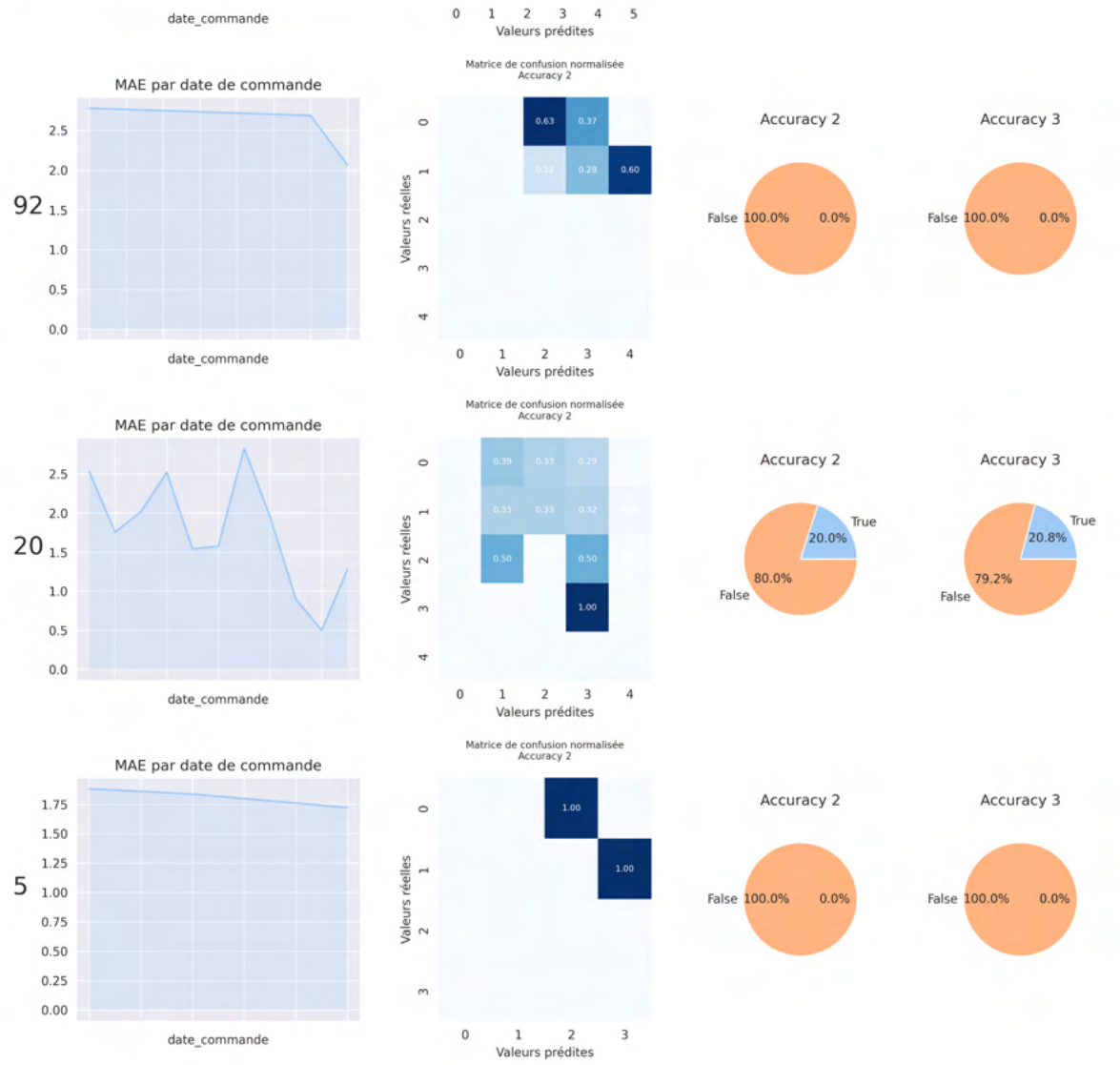
42











Le modèle overfit énormément, c'est sûrement en partie dû au fait que le set de train et de test sont trop différents

Pour essayer d'obtenir un meilleur score , nous rééssayons avec un split train test aléatoire. Cependant il faut garder en mémoire que si les données dont nous disposons ne sont pas réprésentatives du reste de l'année le modèle aura des difficultés à généraliser en production.

Modèle avec Split traint test classique

In [11]:
```
import sklearn.datasets
import sklearn.metrics
from sklearn.model_selection import train_test_split
#import autosklearn

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)
df_index=X_test.reset_index()[["provider","date_commande"]]

print("X_train  : "+str(X_train.shape))
print("X_test  : "+str(X_test.shape))
print("y_train  : "+str(y_train.shape))
print("y_test  : "+str(y_test.shape))
```

```
X_train  : (429605, 65)
X_test  : (143202, 65)
y_train  : (429605,)
y_test  : (143202,)
```

Normalisation (MinMaxScaler)

In [12]:
```
scaler = preprocessing.MinMaxScaler()
scaler.fit(X)
X_train_scaled =scaler.transform(X_train)
X_test_scaled=scaler.transform(X_test)
```

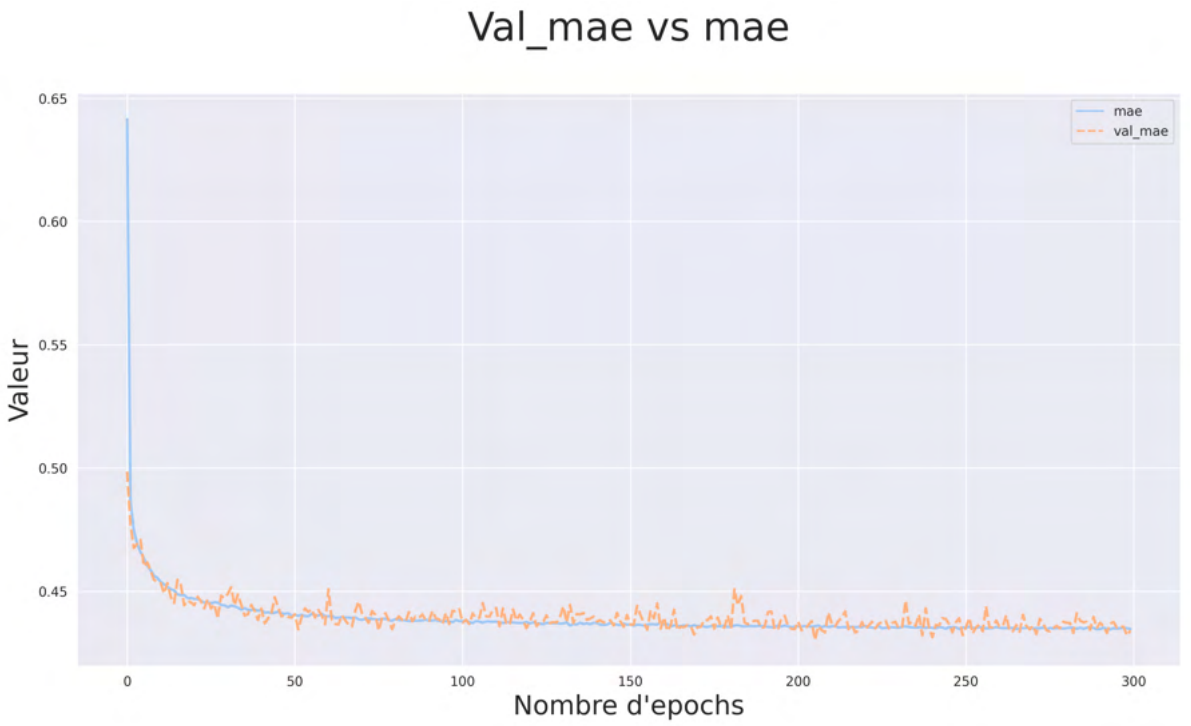
Modèle :

In [13]:
```
model1=keras.Sequential([
        keras.layers.Dense(128, activation='relu',input_shape=(len(X_train.columns)
        #keras.layers.Dropout(0.5),
        keras.layers.Dense(128, activation='relu',kernel_regularizer=regularizers.l
        keras.layers.Dense(64, activation='relu',kernel_regularizer=regularizers.L1
        keras.layers.Dense(64, activation='relu',kernel_regularizer=regularizers.L1
        keras.layers.Dense(32, activation='relu',kernel_regularizer=regularizers.L1
        keras.layers.Dense(16, activation='relu',kernel_regularizer=regularizers.L1
        keras.layers.Dense(1, activation ='relu')
])

model1.compile(optimizer =keras.optimizers.Adamax(learning_rate=0.001),
                      loss='mse',
                      metrics=['mae',], )
```

In [14]:
```
history = model1.fit(X_train_scaled,y_train, epochs =300,validation_split=0.2, bat
results= pd.DataFrame(history.history).reset_index()
results = results.rename(columns={"index":"epochs"}).set_index("epochs")
```

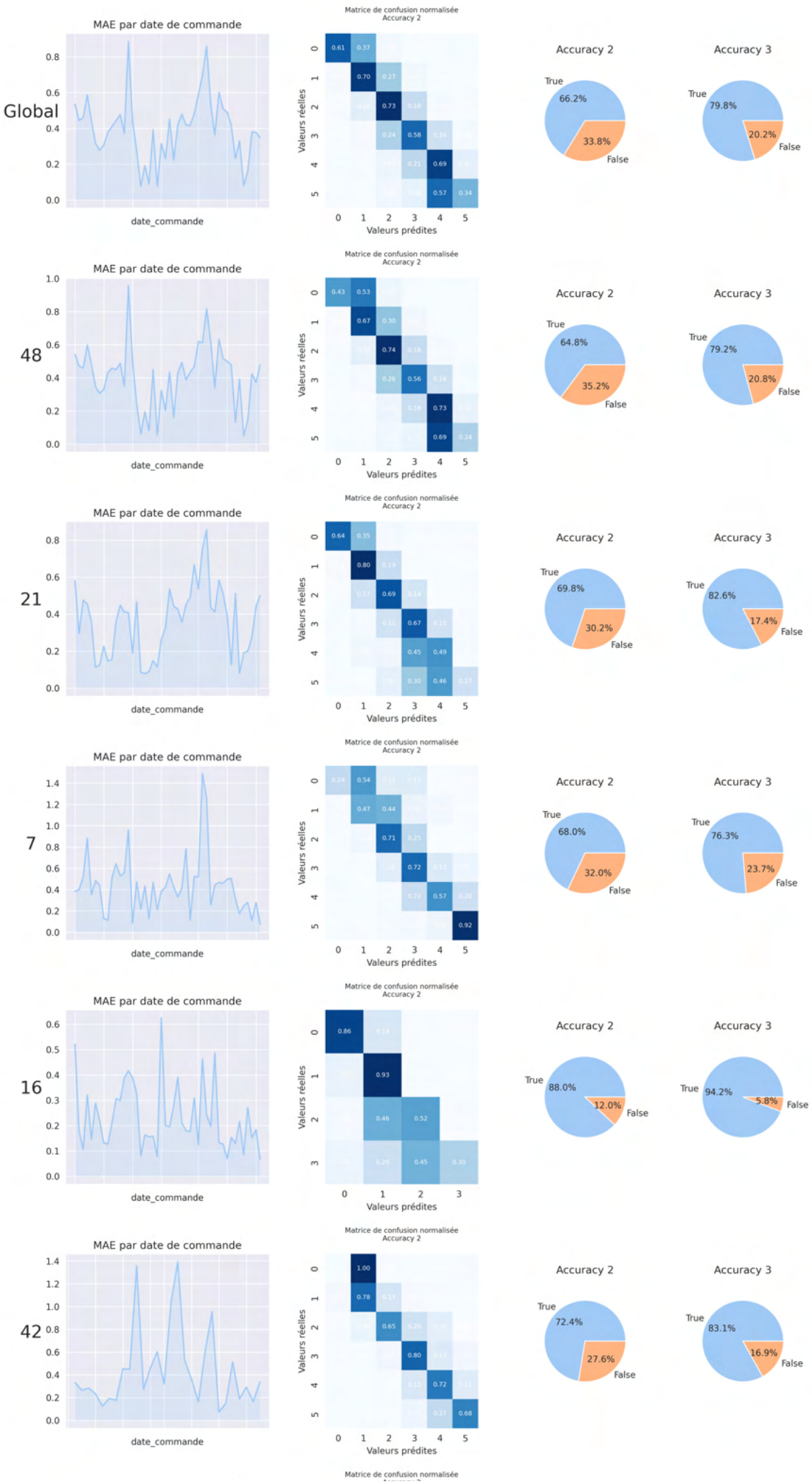
```
50 - val_loss: 0.3872 - val_mae: 0.4320
Epoch 300/300
672/672 [==============================] - 9s 13ms/step - loss: 0.3851 - mae: 0.43
47 - val_loss: 0.3877 - val_mae: 0.4340
```

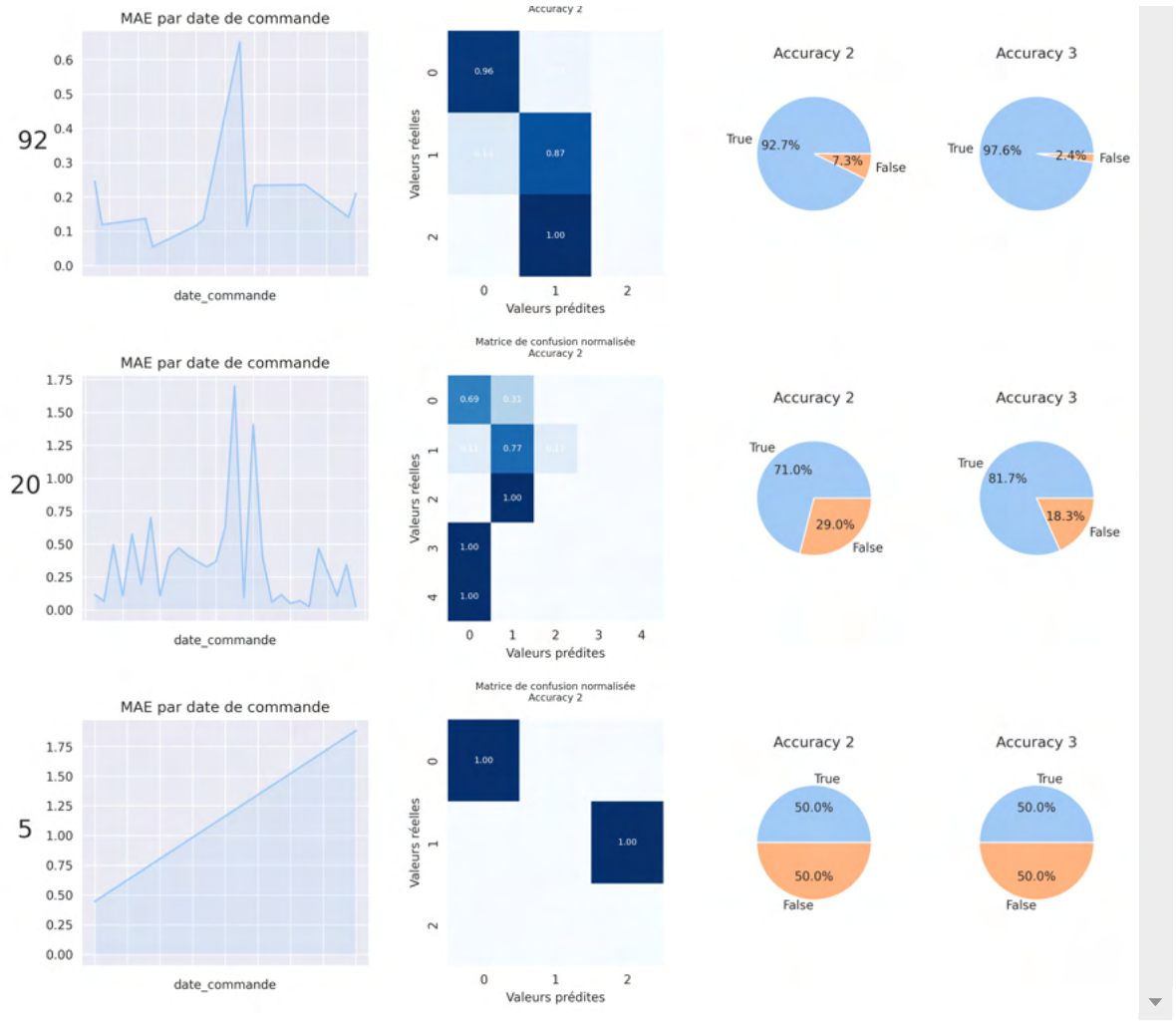
In [15]:
```
sns.lineplot(data=results[["mae","val_mae"]],lw=2)
plt.title("Val_mae vs mae\n", fontsize=30)
plt.ylabel("Valeur",fontsize=20)
plt.xlabel("Nombre d'epochs",fontsize=20)
plt.show()
```



In [16]:
```
X_test_date=pd.DataFrame(X_test_scaled,columns=X_test.columns.tolist())
X_test_date["date_commande"]=df_index["date_commande"]
X_test_date["provider"]=df_index["provider"]
X_test_date["date_commande"]=pd.to_datetime(X_test_date["date_commande"])
X_test_date=X_test_date.set_index(["date_commande","provider"])

error_analysis(model1, X_test_date, y_test)
```

```
Erreur moyenne absolue du modèle : 0.4365 jours
Accuracy 1 : 0.6565
Accuracy 2 : 0.6625
Accuracy 3 : 0.7983
```


Global
MAE par date de commande
date_commande
Matrice de confusion normalisée
Accuracy 2
Valeurs réelles
Valeurs prédites
Accuracy 2
True
66.2%
33.8%
False
Accuracy 3
True
79.8%
20.2%
False
48
MAE par date de commande
date_commande
Matrice de confusion normalisée
Accuracy 2
Valeurs réelles
Valeurs prédites
Accuracy 2
True
64.8%
35.2%
False
Accuracy 3
True
79.2%
20.8%
False
21
MAE par date de commande
date_commande
Matrice de confusion normalisée
Accuracy 2
Valeurs réelles
Valeurs prédites
Accuracy 2
True
69.8%
30.2%
False
Accuracy 3
True
82.6%
17.4%
False
7
MAE par date de commande
date_commande
Matrice de confusion normalisée
Accuracy 2
Valeurs réelles
Valeurs prédites
Accuracy 2
True
68.0%
32.0%
False
Accuracy 3
True
76.3%
23.7%
False
16
MAE par date de commande
date_commande
Matrice de confusion normalisée
Accuracy 2
Valeurs réelles
Valeurs prédites
Accuracy 2
True
88.0%
12.0%
False
Accuracy 3
True
94.2%
5.8%
False
42
MAE par date de commande
date_commande
Matrice de confusion normalisée
Accuracy 2
Valeurs réelles
Valeurs prédites
Accuracy 2
True
72.4%
27.6%
False
Accuracy 3
True
83.1%
16.9%
False



Nous obtenons de meilleurs résultats qui sont plus proches des premiers modèles de machine learning mais cela ne permet toujours par de prouver que le modèle est capable de prédire le temps d'expédition d'un provider qu'il n'a jamais rencontré

Split train test classique en isolant les providers 42 et 92 :

In [17]:
```
#Paramètres :
to_index=["provider","date_commande"]
to_drop=["expedition","jour_expe","id_date","id_date_expe","date_expe"]
to_predict="gap"
features=["heure_commande","min_commande","jour_commande",
          #Variables des providers
          "stock_provider","jours_act_provider","nb_expe_provider","min_provider",'
          "Q3_provider","max_provider","jour_0","jour_1","jour_2","jour_3","jour_4"
          "coef_expe_rapide","coef_expe_normale","coef_expe_lente",
          #Variables totales :
          "stock_total","jours_act_total","nb_expe_total","min_total","Q1_total","r
          "jour_1_tous_provider","jour_2_tous_provider","jour_3_tous_provider","jou
          "jour_6_tous_provider"]
category=["jour_commande","heure_commande"]

providers_isoles=[42,92]

data=pd.read_csv("./data/data_transformed_min3_nb_jours15.csv")
data=data.sort_values(["id_date","heure_commande","min_commande","id_date_expe"])
df_index= data[to_index]

provider_seul=data[data["provider"].isin(providers_isoles)].set_index(to_index)
data2=data[~data["provider"].isin(providers_isoles)].set_index(to_index)
```

```
X=data2.drop(columns=to_drop)[features]
X=pd.get_dummies(X, columns=category)

y=data2[to_predict]

X_provider_seul=provider_seul[features]
X_provider_seul=pd.get_dummies(X_provider_seul, columns=category)
y_provider_seul=provider_seul[to_predict]

print("X : ",X.shape)
print("y : ",y.shape)
print("X_provider_seul : ",X_provider_seul.shape)
print("y_provider_seul : ",y_provider_seul.shape)
```

```
X :  (565520, 65)
y :  (565520,)
X_provider_seul :  (7287, 65)
y_provider_seul :  (7287,)
```

In [18]:
```
import sklearn.datasets
import sklearn.metrics
from sklearn.model_selection import train_test_split
#import autosklearn

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.18)

index_provider_seul=X_provider_seul.index
index_test= X_test.index


X_train=X_train.values.astype('float32')
X_test=X_test.values.astype('float32')
y_train=y_train.values.astype('float32')
y_test=y_test.values.astype('float32')

print("X_train  : "+str(X_train.shape))
print("X_test  : "+str(X_test.shape))
print("X_test_provider_seul : "+str(X_provider_seul.shape))
print("y_train  : "+str(y_train.shape))
print("y_test  : "+str(y_test.shape))
print("y_test_provider_seul : "+str(y_provider_seul.shape))
```

```
X_train  : (463726, 65)
X_test  : (101794, 65)
X_test_provider_seul : (7287, 65)
y_train  : (463726,)
y_test  : (101794,)
y_test_provider_seul : (7287,)
```

Normalisation MinMaxScaler

In [19]:
```
scaler = preprocessing.MinMaxScaler()
scaler.fit(X)
X_train_scaled =scaler.transform(X_train)
X_test_scaled=scaler.transform(X_test)
X_test_provider_seul=scaler.transform(X_provider_seul)
```

Modèle

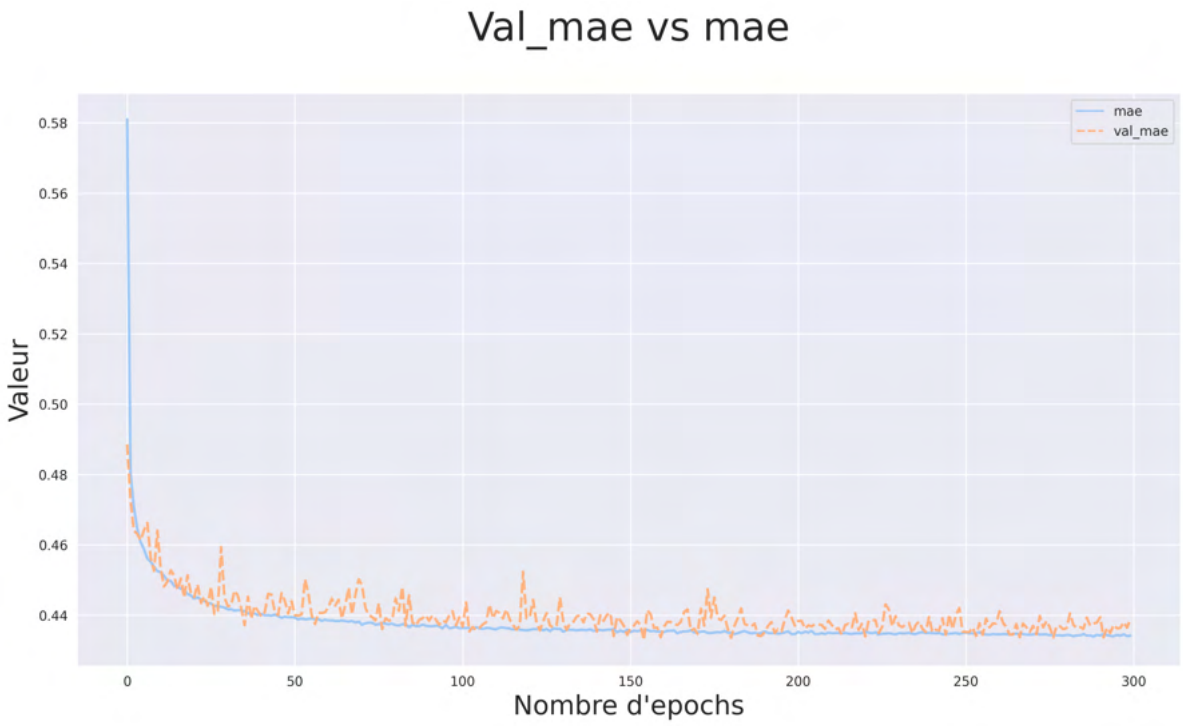
On essaye avec le même modèle que précédemment :

In [22]:
```
model1=keras.Sequential([
        keras.layers.Dense(128, activation='relu',input_shape=(X_train.shape[1],),
        #keras.layers.Dropout(0.5),
        keras.layers.Dense(128, activation='relu',kernel_regularizer=regularizers.
        keras.layers.Dense(64, activation='relu',kernel_regularizer=regularizers.L
        keras.layers.Dense(64, activation='relu',kernel_regularizer=regularizers.L
        keras.layers.Dense(32, activation='relu',kernel_regularizer=regularizers.L
        keras.layers.Dense(16, activation='relu',kernel_regularizer=regularizers.L
        keras.layers.Dense(1, activation ='relu')
])

model1.compile(optimizer =keras.optimizers.Adamax(learning_rate=0.001),
                      loss='mse',
                      metrics=['mae',], )
```

In [23]:
```
history = model1.fit(X_train_scaled,y_train, epochs =300,validation_split=0.2, bat
results= pd.DataFrame(history.history).reset_index()
results = results.rename(columns={"index":"epochs"}).set_index("epochs")
```

```
41 - val_loss: 0.3912 - val_mae: 0.4364
Epoch 300/300
725/725 [==============================] - 8s 11ms/step - loss: 0.3847 - mae: 0.43
41 - val_loss: 0.3940 - val_mae: 0.4389
```

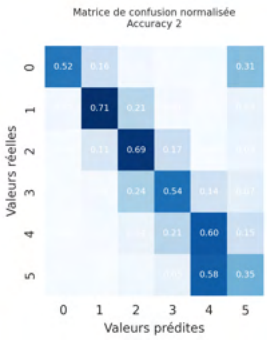
In [24]:
```
sns.lineplot(data=results[["mae","val_mae"]],lw=2)
plt.title("Val_mae vs mae\n", fontsize=30)
plt.ylabel("Valeur",fontsize=20)
plt.xlabel("Nombre d'epochs",fontsize=20)
plt.show()
```

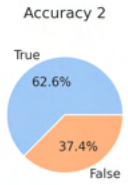


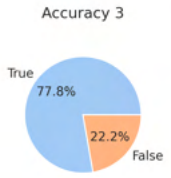
In [25]:
```
X_test_date=pd.DataFrame(X_test_scaled, columns=X.columns.tolist())
X_test_date.index=index_test
X_test_date=X_test_date.reset_index()

X_provider_seul_date=pd.DataFrame(X_provider_seul, columns=X.columns.tolist())
X_provider_seul_date.index=index_provider_seul
X_provider_seul_date=X_provider_seul_date.reset_index()

X_test_date["date_commande"]=pd.to_datetime(X_test_date["date_commande"])
X_provider_seul_date["date_commande"]=pd.to_datetime(X_provider_seul_date["date_cor

X_test_date=pd.concat([X_test_date,X_provider_seul_date],axis=0).set_index(["date_
error_analysis(model1, X_test_date, pd.Series(list(y_test)+list(y_provider_seul)))
```

```
Erreur moyenne absolue du modèle : 4984.8001 jours
Accuracy 1 : 0.6107
Accuracy 2 : 0.6256
Accuracy 3 : 0.7778
```

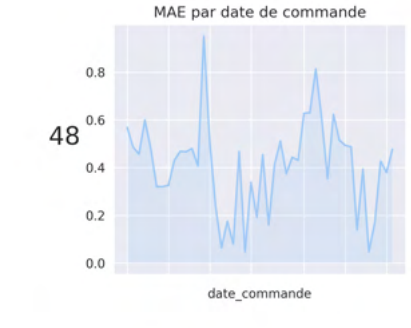
Global

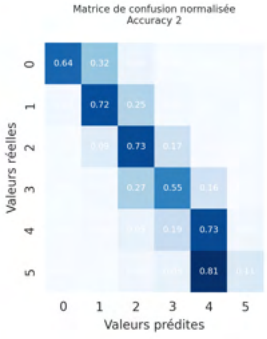


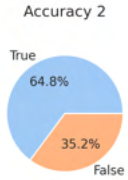


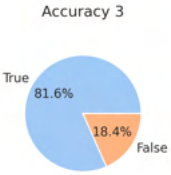


48

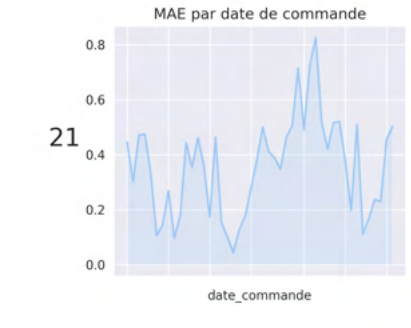


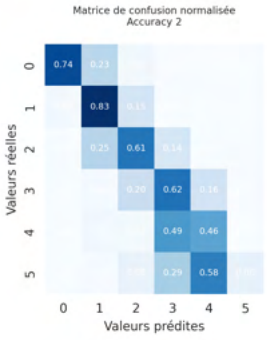


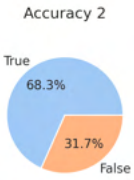


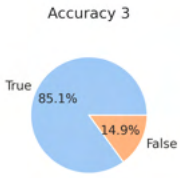


21

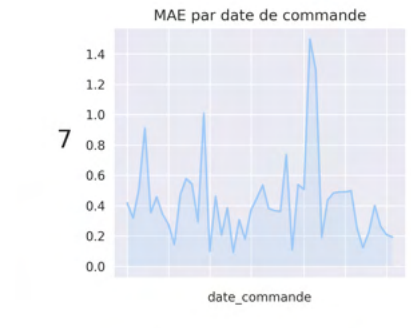


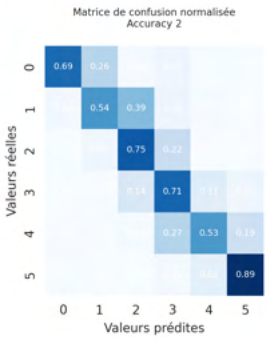


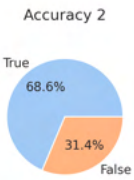


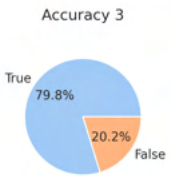


7

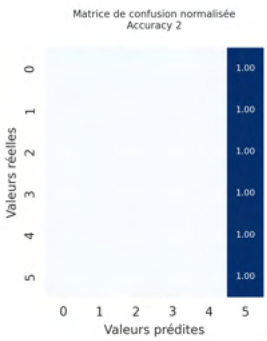


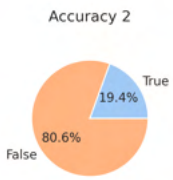


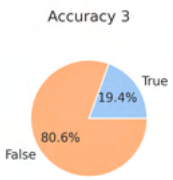




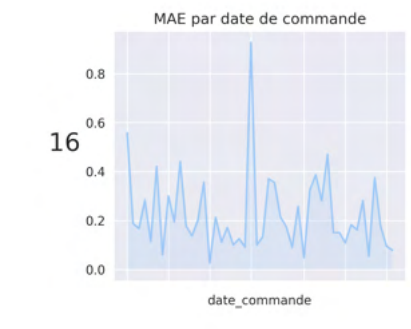
42

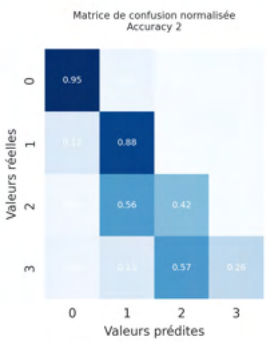


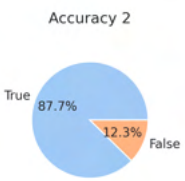


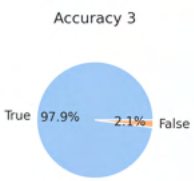


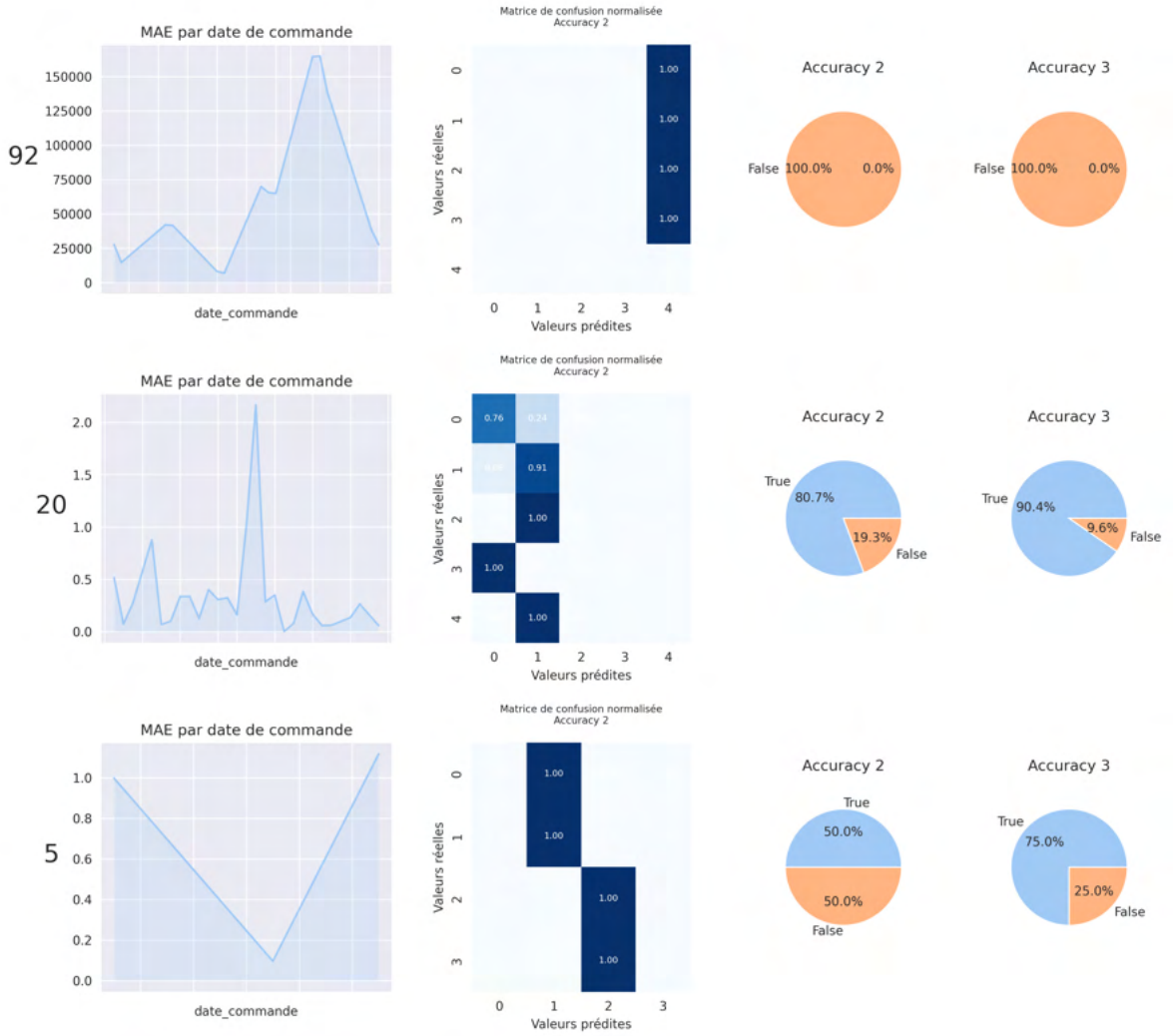
16











In [26]:
```
print('MAE des providers isolés : ', abs(model1.predict(X_provider_seul)-y_provide
```

```
MAE des providers isolés :  74612.68138814198
```

L'embedding des providers ne fonctionne pas du tout sur les providers qui ne sont pas dans le set de train