PROJET

# DATA ANALYST

**“ Prédiction du temps de réponse d’un véhicule de la Brigade des Pompiers de Londres ”**


**Auteurs**

BARRERA CEBRIAN, Ivan
COULANDREAU, Franck
HAEGEMAN, Cédric
HUBERT, Arnaud

**Chef de cohorte**

KASSEL, Raphael

DataScientest • com

**Promotion Février 2021 “ Bootcamp ”**

# INDEX

# I. ABSTRACT

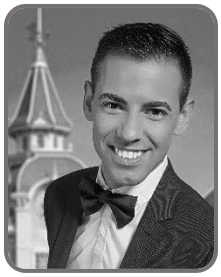
L'objectif de ce projet est d'analyser et/ou d'estimer les temps de réponse et de mobilisation de la Brigade des Pompiers de Londres (London Fire Brigade, LFB). La brigade des pompiers de Londres est le service d'incendie et de sauvetage le plus actif du Royaume-Uni et l'une des plus grandes organisations de lutte contre l'incendie et de sauvetage au monde.

Le *premier jeu de données* fourni contient les détails de chaque incident traité depuis janvier 2017 jusqu'à janvier 2021. Des informations sont fournies sur la date et le lieu de l'incident ainsi que sur le type d'incident traité.

Le *second jeu de données* contient les détails de chaque camion de pompiers envoyé sur les lieux d'un incident depuis janvier 2017 jusqu'à janvier 2021. Des informations sont fournies sur l'appareil mobilisé, son lieu de déploiement et les heures d'arrivée sur les lieux de l'incident.

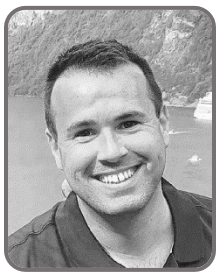
**Key words :** brigade, pompiers, Londres, mobilisation, incident, incendie, caserne, intervention, temps, LFB, London, fire.

## Auteurs du projet



Ivan
BARRERA CEBRIAN

Franck
COULANDREAU



Cédric
HAEGEMAN

Arnaud
HUBERT

## Niveau d'expertise autour du sujet à étudier

Ayant des profils professionnels très différents, la première approche de ce projet nous a permis d'avoir des points de vue complémentaires sur le sujet adressé. Bien que nous n'ayons pas d'expertise directe sur l'organisation d'une caserne de pompiers, ni sur la problématique de prévision d'un temps.

Afin d'approfondir nos connaissances, nous avons fait des recherches dans ce domaine en nous renseignant par exemple sur le nombre de casernes, la composition géographique de Londres ainsi que ses différents quartiers, le langage technique utilisé dans le métier, etc. L'ensemble de ces informations nous ont permis d'avoir une meilleure vision sur la problématique adressée.

En dehors de notre propre étude, nous n'avons pas pris contact avec des experts du métier de pompier, mais notre chef de cohorte, Raphael KASSEL, nous a bien guidé dans le côté technique afin d'appliquer nos connaissances dans l'analyse des données et proposer un modèle de prédiction de prévision d'un temps.

## II. EXPLORATION ET SELECTION DES DONNEES

Comme il a été décrit précédemment dans l'Abstract, nous allons explorer et analyser en détail les 2 jeux des données correspondant aux « Incidents » et à la « Mobilisation » :

- Premier jeu de données - Incidents : *LFB_Incident_data_from_January_2017.xlsx*

Ce dataset est composé d'un total de 38 variables et 420 244 lignes :

1. IncidentNumber
2. DateOfCall
3. CalYear
4. TimeOfCall
5. HourOfCall
6. IncidentGroup
7. StopCodeDescription
8. SpecialServiceType
9. PropertyCategory
10. PropertyType
11. AddressQualifier
12. Postcode_full
13. Postcode_district
14. UPRN
15. USRN
16. IncGeo_BoroughCode
17. IncGeo_BoroughName
18. ProperCase
19. IncGeo_WardCode
20. IncGeo_WardName
21. IncGeo_WardNameNew
22. Easting_m
23. Northing_m
24. Easting_rounded
25. Northing_rounded
26. Latitude
27. Longitude
28. FRS
29. IncidentStationGround
30. FirstPumpArriving_AttendanceTime
31. FirstPumpArriving_DeployedFromStation
32. SecondPumpArriving_AttendanceTime
33. SecondPumpArriving_DeployedFromStation
34. NumStationsWithPumpsAttending
35. NumPumpsAttending
36. PumpCount
37. PumpHoursRoundUp
38. Notional Cost (£)

- Deuxième jeu de données - Mobilisation : *LFB_Mobilisation_data_from_January_2017.xlsx*

Ce dataset est composé d'un total de 22 variables et 618 071 lignes :

1. IncidentNumber
2. ResourceMobilisationId
3. Resource_Code
4. PerformanceReporting
5. DateAndTimeMobilised
6. DateAndTimeMobile
7. TimeMobileTimezoneId
8. DateAndTimeArrived
9. TimeArrivedTimezoneId
10. AttendanceTimeSeconds
11. DateAndTimeLeft
12. TimeLeftTimezoneId
13. DateAndTimeReturned
14. TimeReturnedTimezoneId
15. DeployedFromStation_Code
16. DeployedFromStation_Name
17. DeployedFromLocation
18. PumpOrder
19. PlusCode_Code
20. PlusCode_Description
21. DelayCodeId
22. DelayCode_Description

Nous avons un total de 60 variables et 1 038 315 lignes, mais pour atteindre l'objectif de cette étude, l'estimation du temps de réponse et la mobilisation de la Brigade des Pompiers de Londres, nous avons fusionné les deux datasets grâce à la variable « *IncidentNumber* », qui nous servira d'index.

Le nouveau DataFrame issue de la fusion, appelé *LFB_analysis_py.csv,* contient la sélection des 31 variables suivantes et 840 488 lignes :

**Table 1.** *Sélection des variables pertinentes pour l'analyse*

| N° | NOM DES VARIABLES | DESCRIPTION DES VARIABLES |
|---|---|---|
| 1 | IncidentNumber | Numéro de l'incident |
| 2 | DateOfCall | Date de l'appel JJ/MM/YYYY |
| 3 | TimeOfCall | Temps de l'appel HH/MM/SS |
| 4 | IncidentGroup | Catégorie d'incident |
| 5 | StopCodeDescription | Sous-catégorie d'incident |
| 6 | SpecialServiceType | Type de service spécial |
| 7 | PropertyCategory | Type de propriété |
| 8 | PropertyType | Sous-catégorie du type de propriété |
| 9 | AddressQualifier | Qualité du renseignement sur la localisation de l'incident |
| 10 | Postcode_district | Code district |
| 11 | IncGeo_BoroughCode | Code arrondissement |
| 12 | ProperCase | Nom arrondissement en minuscule |
| 13 | IncGeo_WardCode | Code du quartier |
| 14 | IncGeo_WardNameNew | Nom du quartier |
| 15 | Latitude | Coordonnées GPS Latitude |
| 16 | Longitude | Coordonnées GPS Longitude |
| 17 | **FirstPumpArriving_AttendanceTime** | **Délai d'arrivée de la 1ère brigade en secondes** |
| 18 | FirstPumpArriving_DeployedFromStation | Nom de la brigade d'origine 1ère brigade |
| 19 | SecondPumpArriving_AttendanceTime | Délai d'arrivée de la 2ème brigade en secondes |
| 20 | SecondPumpArriving_DeployedFromStation | Nom de la brigade d'origine 2ème brigade |
| 21 | NumStationsWithPumpsAttending | Nombre de la brigade engagée pour 1 même incident |
| 22 | NumPumpsAttending | Nombre de camions déployés de la brigade pour 1 même incident |
| 23 | PumpCount | Nombre total de camions déployés toutes brigades confondues pour 1 même incident |
| 24 | PumpHoursRoundUp | Nombre d'heures d'intervention cumulées |
| 25 | Notional Cost (£) | Coût théorique de l'intervention |
| 26 | DateAndTimeMobile | Date et heure à laquelle les pompiers partent de la caserne |
| 27 | DateAndTimeArrived | Date et heure à laquelle les pompiers sont arrivés sur le lieu de l'incident |
| 28 | DateAndTimeLeft | Heure de départ des pompiers du lieu de l'incident |
| 29 | DeployedFromStation_Code | Code identifiant de la caserne |
| 30 | DeployedFromStation_Name | Nom de la brigade (nom du quartier) |
| 31 | DelayCodeId | Code identifiant de la raison du délai |

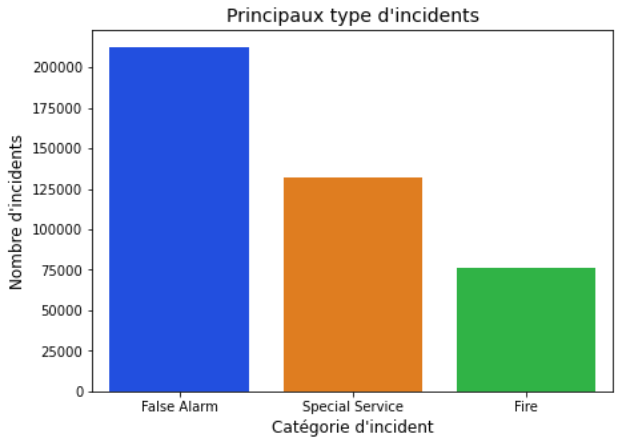
Notre variable cible pour créer notre modèle de prédiction du temps de réponse et de mobilisation de la Brigade des Pompiers de Londres sera **« FirstPumpArriving_AttendanceTime »**, qui nous proportionne le délai de mobilisation et d'arrivée de la première brigade. Les valeurs de cette variable sont données en secondes. (*Se référer à notre tableau des 'Stats' Excel pour la distribution des valeurs*)

Dans le chapitre suivant, nous allons croiser différentes variables afin d'en tirer des conclusions pour pouvoir justifier la sélection des nos 31 variables parmi les 60 que nous avions au départ.

## III. DATAVIZ'

Après notre analyse de l'ensemble des variables nous vous présentons une **analyse générale des données** :

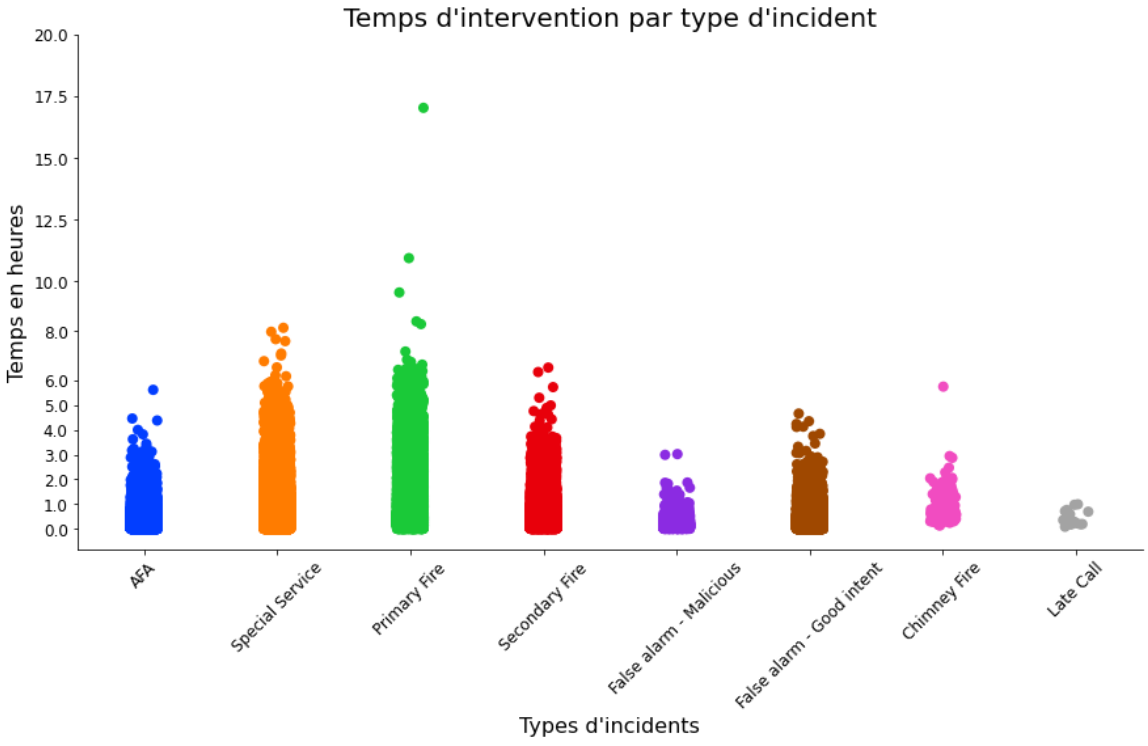
Graphique 1. *Principaux types d'incidents en nombre absolu*




Nous pouvons constater que la majorité des incidents signalés sont des fausses alarmes (automatique ou bien manuelle.)

Viennent ensuite les 'Special Service', puis en dernière position les incendies.

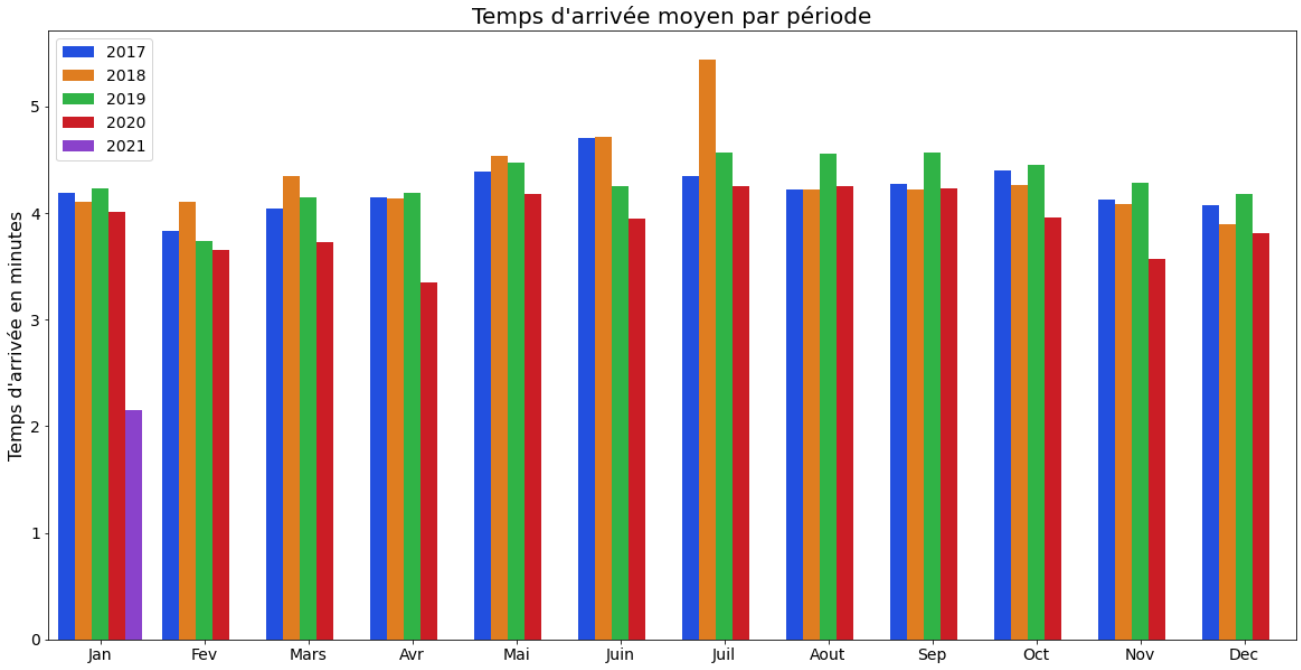
Graphique 2. *Temps d'intervention par type d'incident*




Nous constatons que les temps d'interventions par type d'incident sont généralement de 2 heures, avec des temps plus élevés comme pour les 'Incendies' et 'Special Services' (inondation, suicide, ouverture d'ascenseurs…)

Dans un autre côté, les incendies sont les types d'incident qui peuvent retenir les pompiers plusieurs dizaines d'heures. Nous pouvons visualiser aussi que les 3 types d'alarmes engrangent énormément de temps cumulé inutile.

Graphique 3. *Temps d'arrivée moyen par période de l'année*

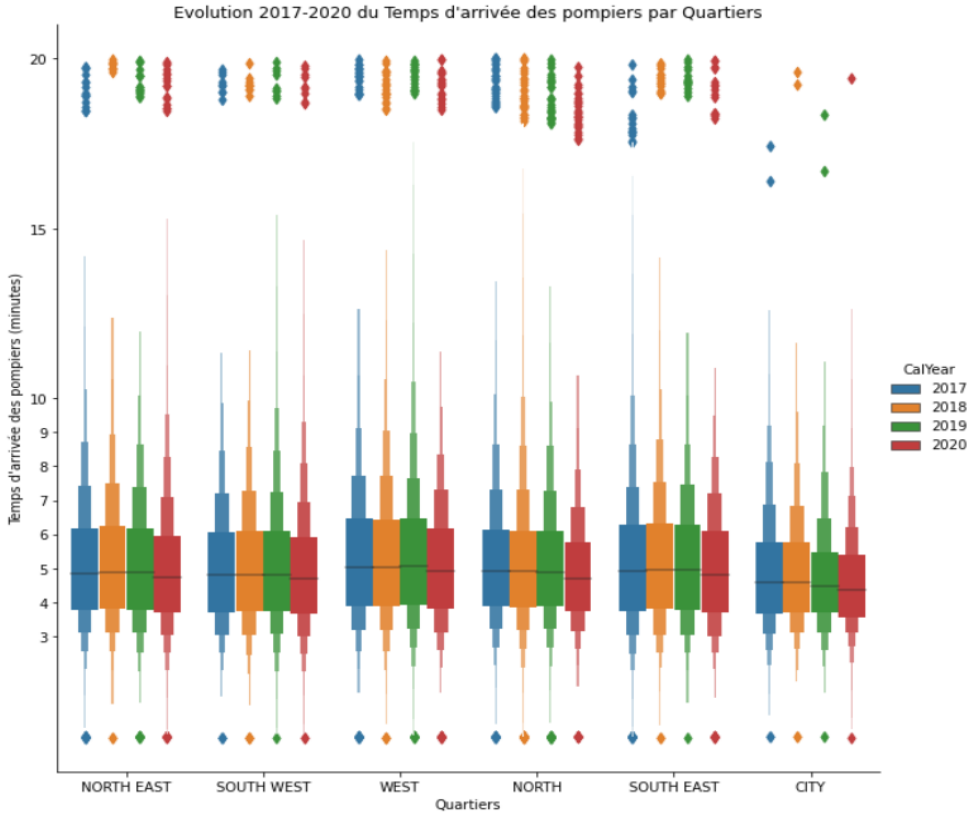



* Notre dataset contient des données jusqu'au 12/01/2021

Nous pouvons remarquer que le temps d'arrivée moyen fluctue selon la période de l'année, avec des temps élevés sur la période estivale, et en diminution sur la période hivernale.

Nous constatons également un pic sur le mois de juillet 2018.

Graphique 4. *Évolution du temps moyen d'arrivée des pompiers par secteurs géographiques*




Pour mieux comprendre le graphique, *Le Grand Londres* est composé de 33 quartiers qu'il est commun de regrouper en 6 secteurs géographiques :

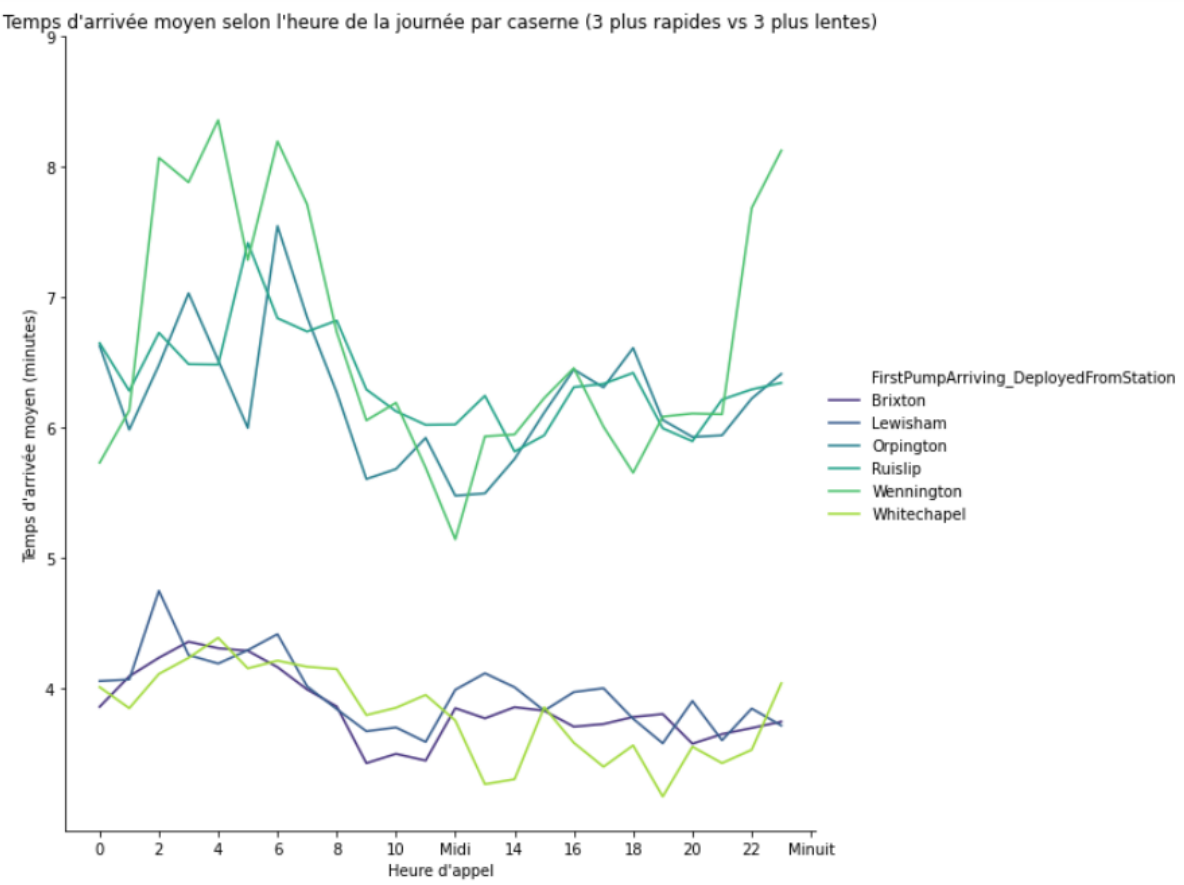
- **SOUTH EAST :** Bexley, Bromley, Greenwich, Lewisham, Southwark.
- **NORTH EAST :** Havering, Redbridge, Newham, Barking & Dagenham,Waltham Forest, Tower Hamlets.
- **NORTH :** Enfield, Barnet, Haringey, Hackney, Islington, Camden, Westminster.
- **WEST :** Harrow, Hillingdon, Brent, Ealing, Hounslow, Hammersmith & Fulham, Kensington & Chelsea.
- **SOUTH WEST :** Richmond upon Thames, Kingston upon Thames, Wandsworth, Merton, Lambeth, Sutton, Croydon.
- **CITY :** City of London.


Cartographie : *"Secteurs du Grand Londres"*

Pour les 6 secteurs géographiques du Grand Londres on peut constater que la médiane des temps d'arrivée des pompiers se situe aux alentours de 5 minutes et que dans la majorité des cas les pompiers mettent entre 4 et 6 minutes entre une alerte et leur arrivée sur les lieux. On observe que les pompiers sur le secteur de la City mettent moins de temps à intervenir que dans les autres secteurs.

On peut également noter que pour les 6 secteurs géographiques le temps d'arrivée des pompiers a diminué sur l'année 2020, probablement à cause de la circulation plus fluide observée pendant les périodes de confinement liées à la crise sanitaire du Covid19.

Graphique 5. *Temps moyen d'arrivée des pompiers par caserne*




Parmi les 105 casernes du Grand Londres, on constate que le temps d'arrivée moyen de la première équipe sur les lieux d'un incident peut varier significativement entre les casernes.

Une différence de l'ordre de 2 à 3 minutes est observée entre les casernes qui interviennent le plus rapidement et celles qui mettent le plus de temps (quel que soit l'heure de l'appel dans la journée). On peut imaginer facilement que sur un départ d'incendie ces 2 à 3 minutes peuvent avoir de lourdes conséquences.

Pour construire cette visualisation il a fallu ne tenir compte que des incidents pour lesquels c'est la caserne du secteur qui est intervenue. En effet, il arrive parfois qu'une caserne extérieure au secteur arrive en premier sur les lieux d'un incident avec un temps forcément un peu supérieur à ses temps habituels.

**Table 2.** *Étude de corrélation entre le temps d'arrivée des pompiers et la caserne du lieu de l'incident (ANOVA)*

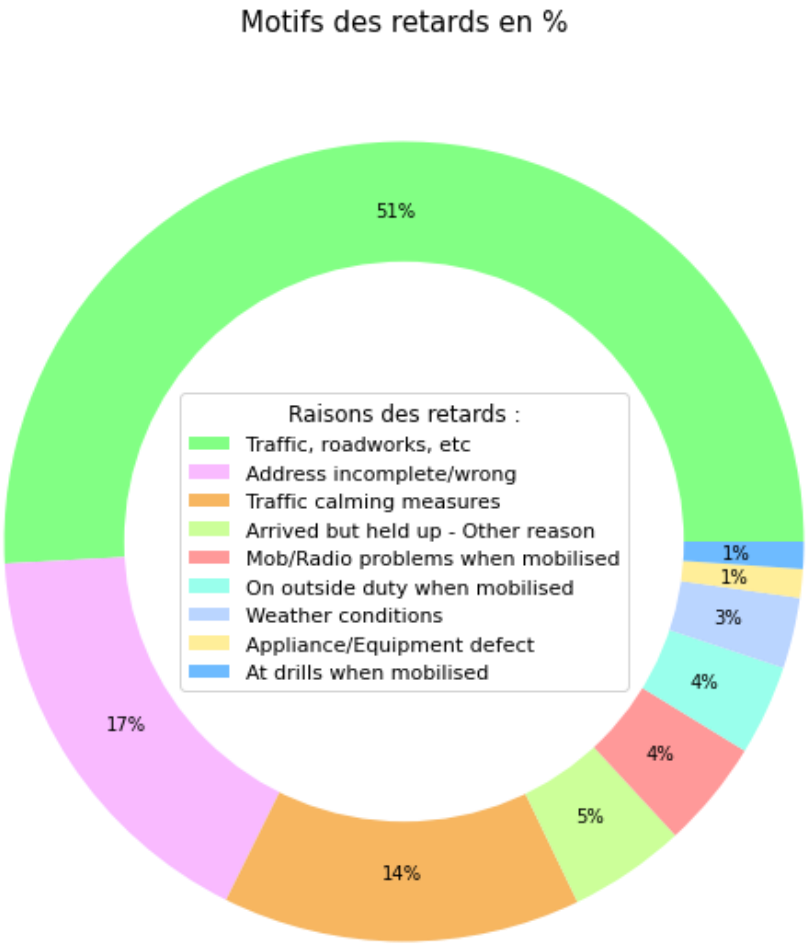
| | df | sum_sq | mean_sq | F | PR(>F) |
|---|---|---|---|---|---|
| IncidentStationGround | 101.0 | 3.744967e+08 | 3.707889e+06 | 224.890618 | 0.0 |
| Residual | 389079.0 | 6.414948e+09 | 1.648752e+04 | NaN | NaN |

La p-value (PR(>F)) à 0.0 est inférieure à 5 % donc on peut rejeter l'hypothèse selon laquelle *IncidentStationGround* (caserne du lieu d'un incident) n'influe pas sur *FirstPumpArriving_AttendanceTime* (temps d'arrivée des pompiers sur les lieux).

Par la suite et après une analyse approfondie de l'ensemble des variables, nous vous présentons les différentes représentations graphiques sur **l'impact du temps de réponse et mobilisations** des pompiers.

Cette deuxième partie de la représentation va servir comme introduction pour ensuite créer des modèles de prédiction à l'aide de Machine Learning.

Graphique 6. *Motifs des retards*

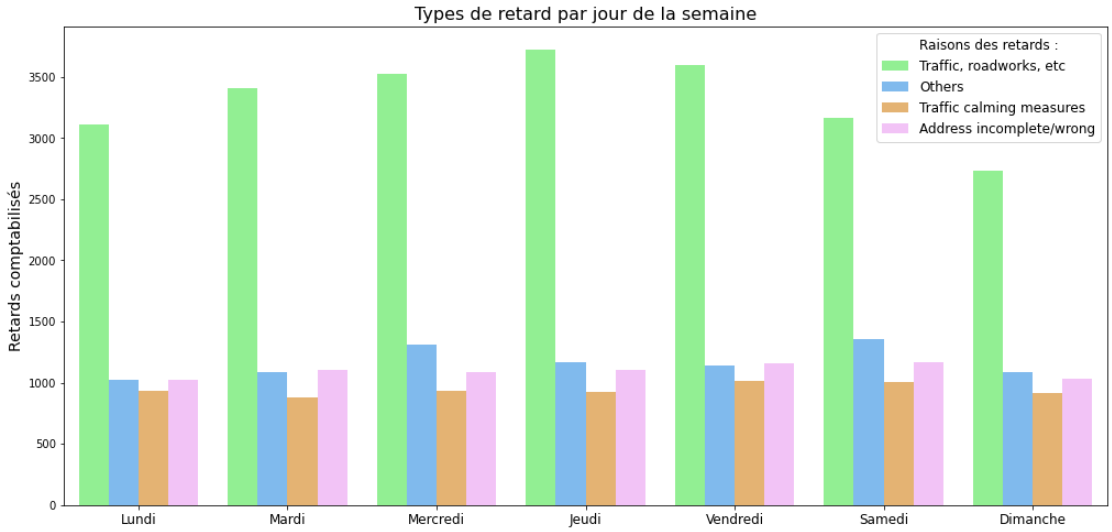



Nous constatons que les embouteillages représentent une grande majorité des raisons de retard avec 51%, suivi par 17% des anomalies d'adresses et, enfin, 14% des retards sont dus aux mesures de régulation de la circulation.

Le reste des motifs reste très minoritaire et donc nous allons les regrouper dans la variable 'Autres' pour le reste de cette analyse.

Nous allons maintenant étudier le type de retard par période de la semaine et de la journée et enfin calculer leurs temps moyens.

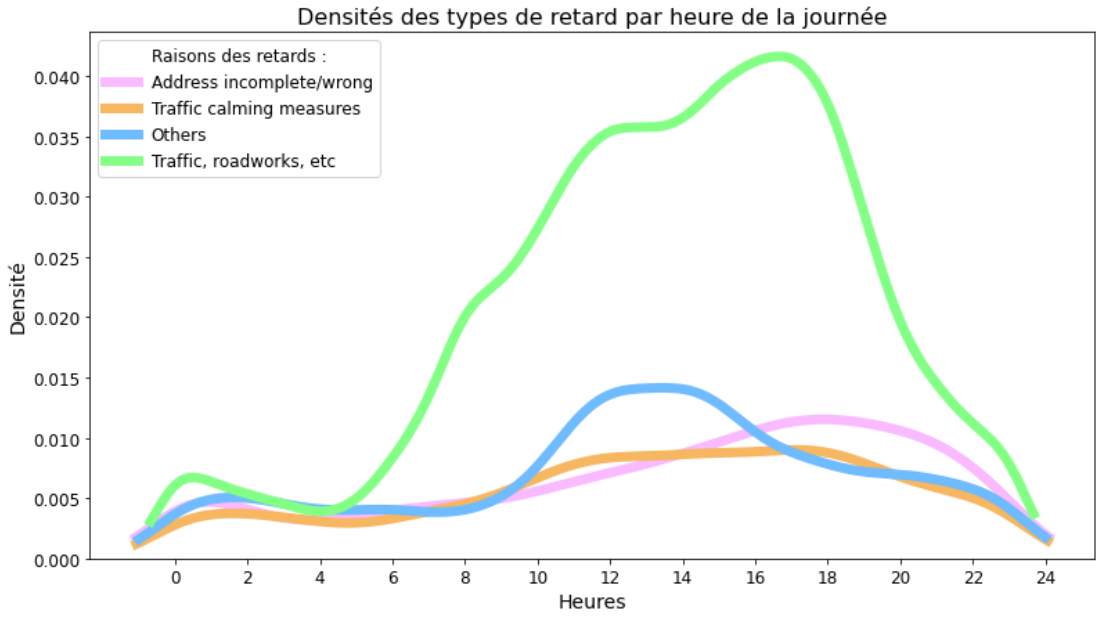
Graphique 7. *Types de retard par jour de la semaine*




Comme le démontrait le graphique précédent, les embouteillages sont fortement représentés, ceux-ci connaissent un pic en milieu de semaine et une régression le week-end.

Une fois regroupé, la variable « *Autres* » représente la 2eme cause des retards, avec les journées du mercredi et samedi en pic.

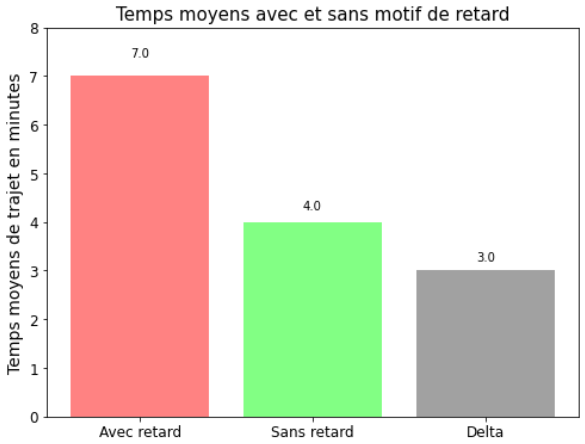
Graphique 8. *Densités des types de retard par heure de la journée*




Intéressons-nous à la variable la plus forte : « Embouteillage ». Sans surprise, celle-ci est très prononcée lors des heures de travail, entre 7h et 20h.

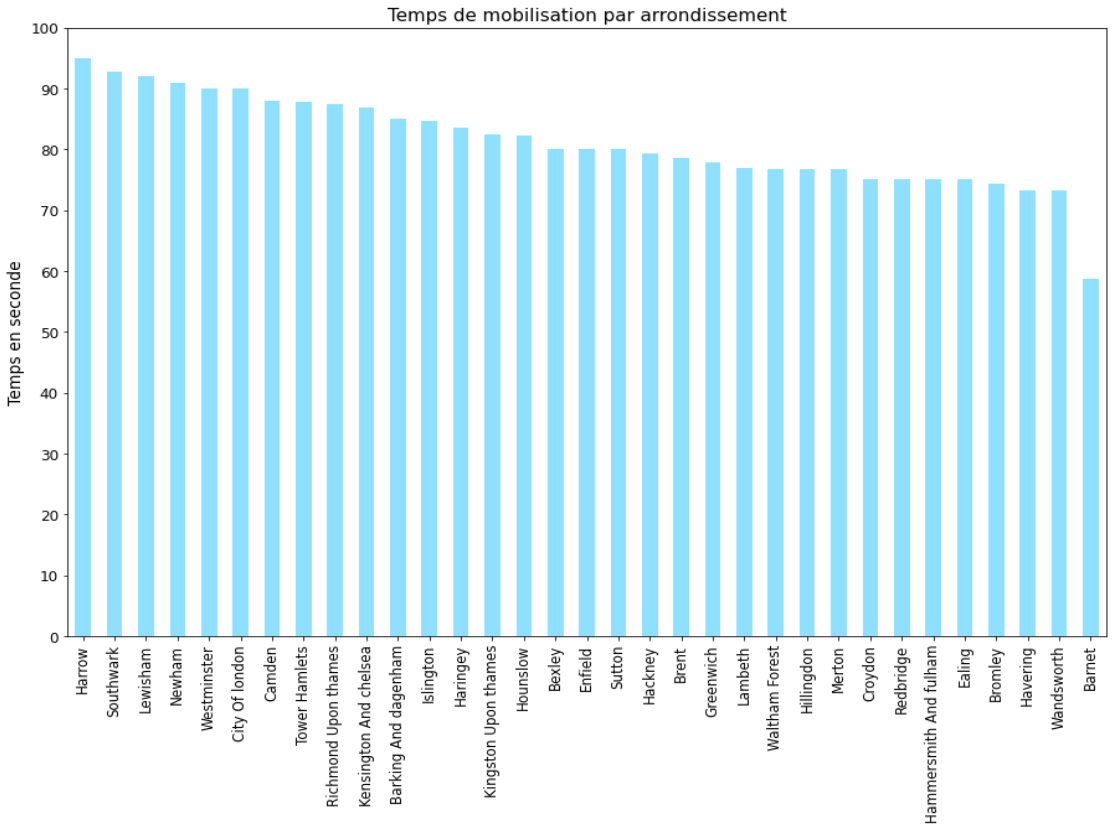
C'est ensuite la variable « *Autres* » qui prend le relais avec un pic entre 10h et 16h. Et enfin la variable « *Anomalies d'adresses* », qui se distingue en soirée.

Graphique 9. *Temps moyens avec et sans motifs de retard*




Une intervention avec retard mettra en moyenne 7 minutes, contre 4 minutes en temps normal, entre le départ et l'arrivée des pompiers sur les lieux des incidents. Il y a donc une moyenne de 3 minutes dès qu'un retard est enregistré.

Graphique 10. *Temps de mobilisation par arrondissement*

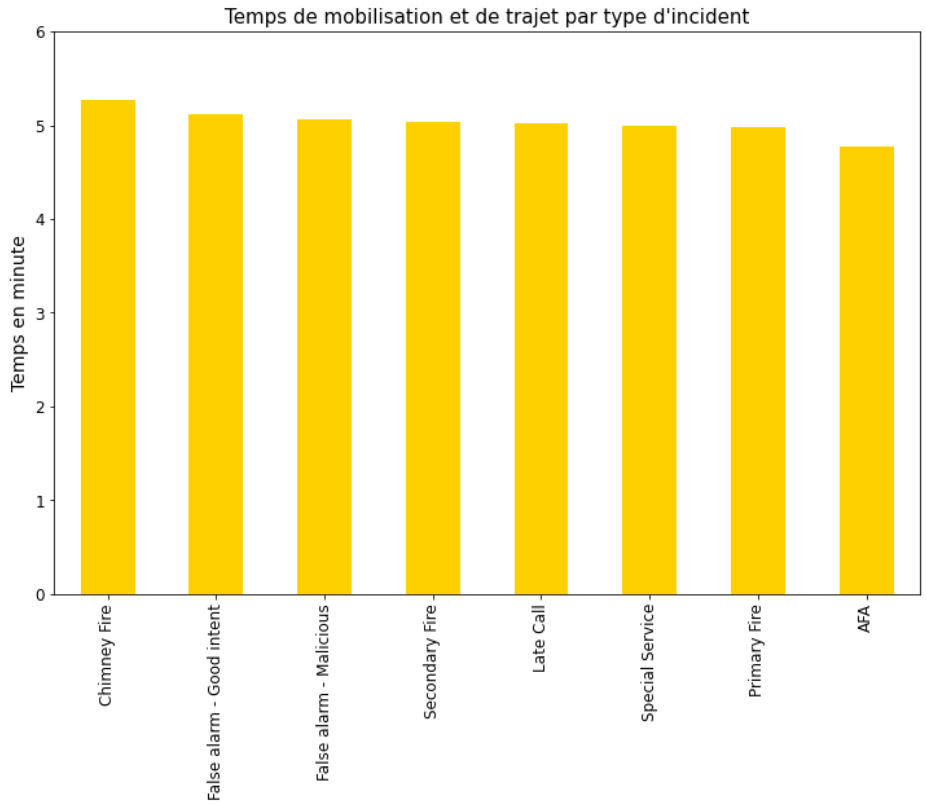



Ce graphique représente par arrondissement, le temps moyen écoulé entre l'heure de l'appel et l'heure de départ des pompiers.

Nous constatons un delta moyen de 37 secondes entre les 2 arrondissements en valeurs extrêmes.

La moyenne générale est de 80 secondes.

Graphique 11. *Temps de mobilisation et de trajet par type d'incident*




Ce graphique représente par type d'incident, le temps moyen écoulé entre le moment de l'appel et l'arrivée sur les lieux. Afin d'avoir les valeurs les plus réelles, les données n'incluent aucun type de retard.

Au vu du graphique, nous pouvons en déduire que les temps de mobilisation et de trajet restent les mêmes pour n'importe quel type d'incident avec un delta de 29 secondes entre les 2 valeurs extrêmes.

Les équipes et le matériel sont donc déjà prêt pour tous types d'interventions.

## - Résumé de l'analyse -

Par ces graphiques, nous avons pu obtenir une meilleure représentation visuelle de la répartition générale de notre dataset. Nous avons également pu étudier en détail, l'impact de certains facteurs tels que :

- Les motifs de retard.
- Les types d'interventions, selon l'hypothèse que certaines d'entre elles pourraient nécessiter une préparation spécifique et donc ralentir les délais d'intervention.
- Les temps de mobilisation par arrondissement, afin d'évaluer la performance par secteur géographique.

Ces DataViz' ont démontré que seule la variable « *retards* » influait sur le temps d'arrivée moyen et de manière modérée. Nous allons donc, au vu de ces premières informations visuelles, étudier des modèles de prédictions.

# IV. MACHINE LEARNING

## • Première itération

Pour la partie Machine Learning, nous avons essayé d'estimer le temps d'arrivée des pompiers à partir de l'heure d'appel à leurs services.

La variable cible, étant une variable continue, nous avons choisi d'entrainer des méthodes de Régression Linéaires : LinearRegression, RidgeCV et LassoCV.

La variable cible identifiée est :

- FirstPumpArriving_AttendanceTime

Les variables numériques de nos datasets que nous avons retenues sont :

- DateOfCall retravaillée en CalYear, CalMonth, CalDay, CalWeekDay
- HourOfCall
- Easting_rounded
- Northing_rounded

Les variables catégorielles de nos datasets que nous avons retenues sont :

- IncidentGroup
- StopCodeDescription
- SpecialServiceType
- AddressQualifier
- ProperCase
- IncidentStationGround
- FirstPumpArriving_DeployedFromStation
- DeployedFromLocation
- DelayCodeId

De plus nous avons enrichi nos données en rajoutant des variables calculées :

- Casernes_quartier : le nombre de casernes du quartier de l'incident (ex. dans le quartier de Westminster il y a 8 casernes)

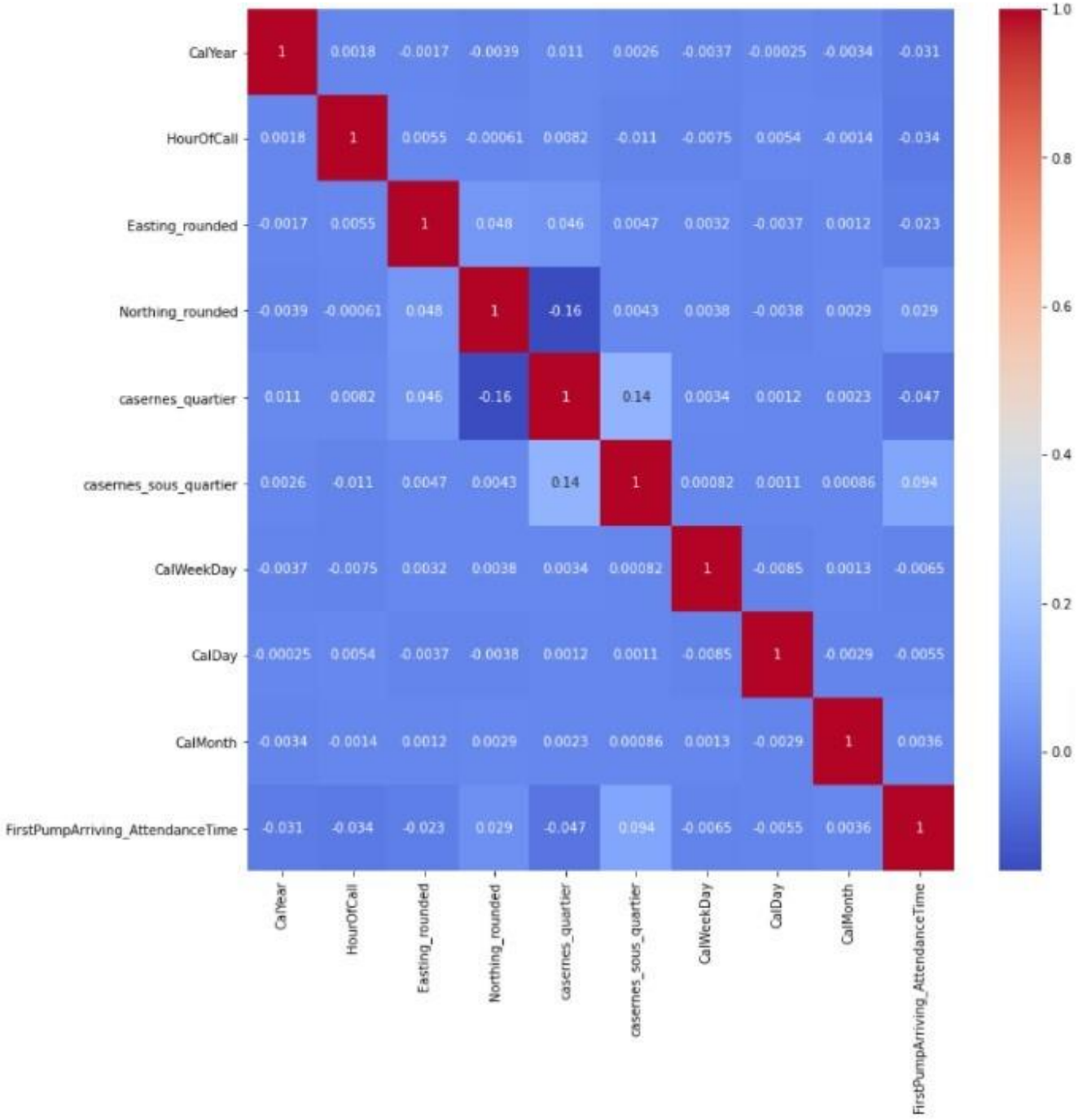
- Casernes_sous_quartier : le nombre de casernes du sous quartier de l'incident (ex. dans le sous quartier de Barkingside il y a 2 casernes)

## Modèles de Machine Learning expérimentés

La heatmap de corrélation des variables numériques ne nous a pas permis de trouver une variable fortement corrélée avec notre cible.

La corrélation la plus élevée concerne la variable « *casernes_sous_quartier* » avec un coefficient de 0.094 ce qui est trop faible pour faire un modèle de Régression Linéaire simple.

Graphique 12. *Heatmap (1ère itération)*



- Modèle LinearRegression

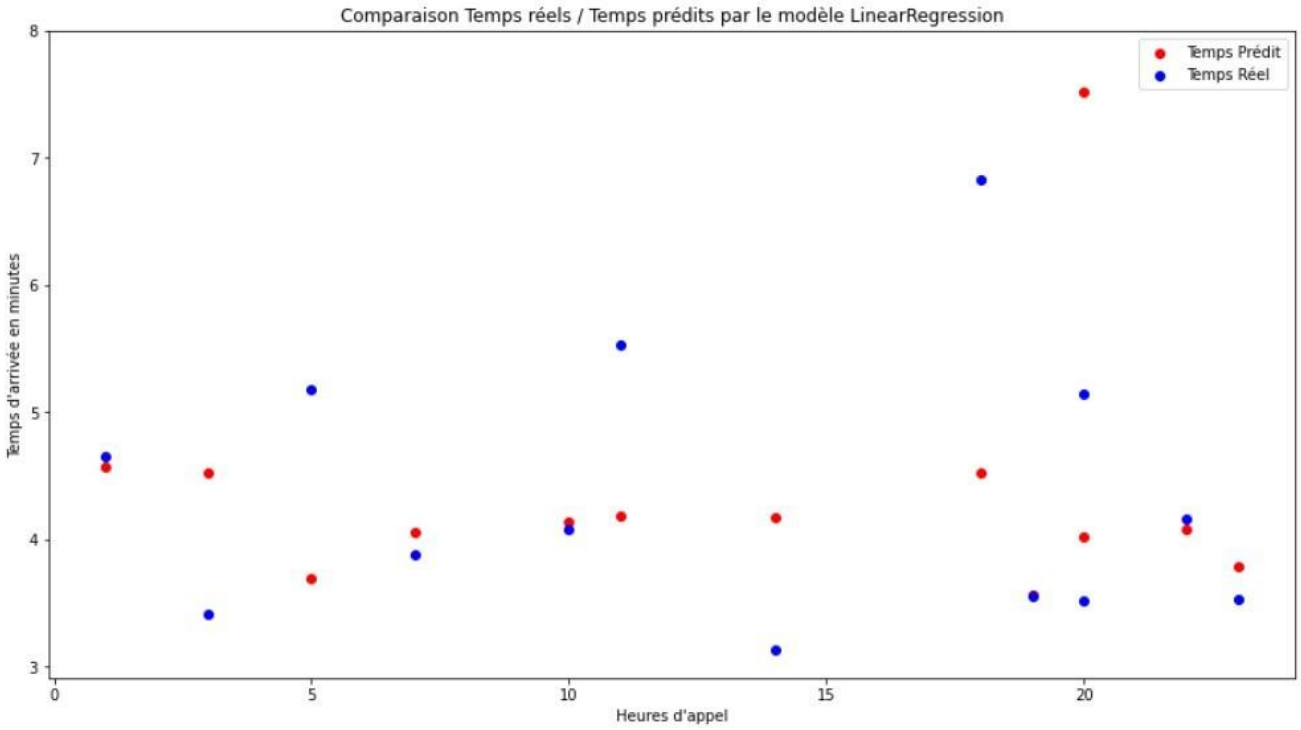
Le coefficient de détermination du modèle sur l'échantillon d'entrainement est de 0.5839818857279384.

Le coefficient de détermination du modèle sur l'échantillon de test est de 0.5786472154140925.

Les scores sont faibles mais relativement proches entre nos 2 échantillons.

Le graphique suivant nous montre sur une journée de l'échantillon de test les écarts entre les temps réels d'arrivée des pompiers et les temps prédits par le modèle *LinearRegression* à différentes heures de la journée.

Graphique 13. *Visualisation Modèle LinearRegression*



- Modèle RidgeCV

Le score du modèle sur l'échantillon d'entrainement est de 0.5839438555837259.

Le score du modèle sur l'échantillon de test est de 0.5787123523986961.

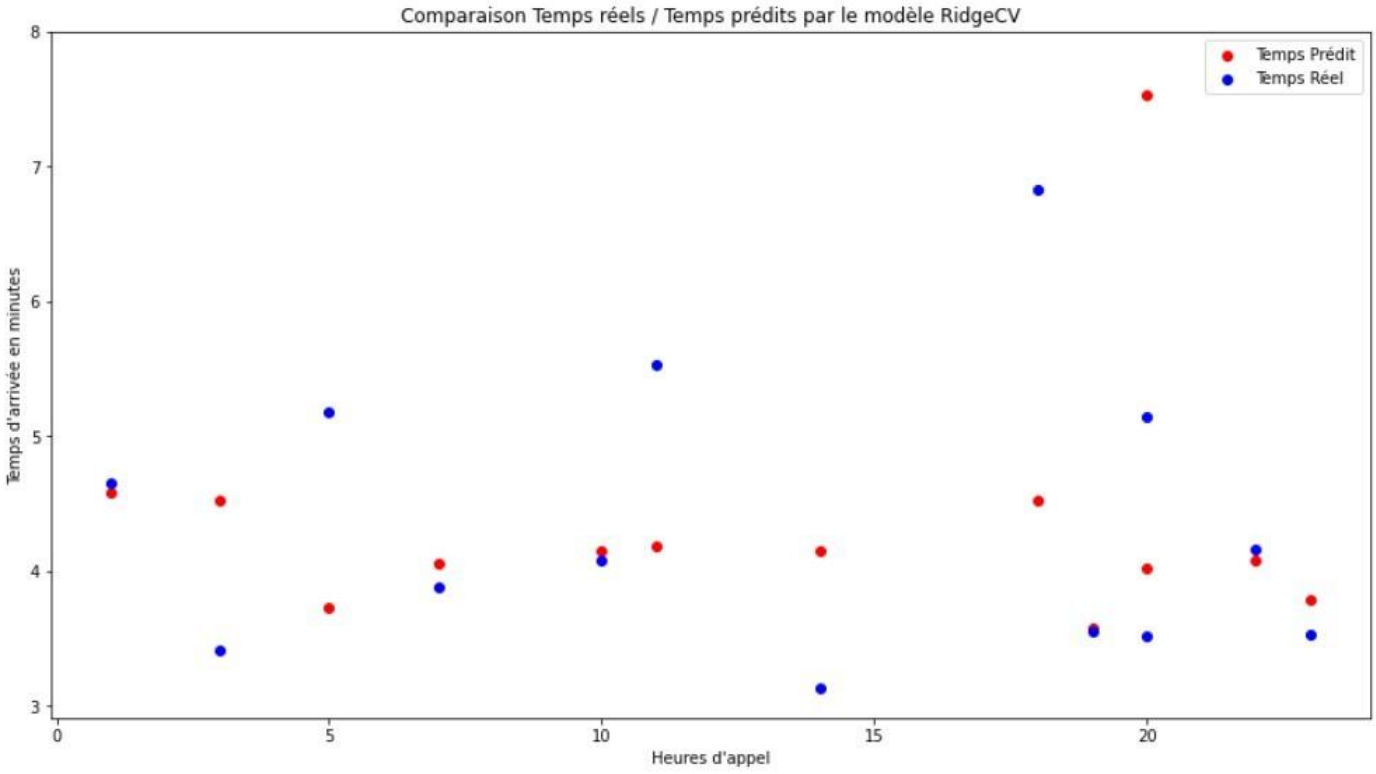
Le mean_squared_error sur l'échantillon d'entrainement est de 0.4150893945887835.

Le mean_squared_error sur l'échantillon de test est de 0.4252015806664894.

Les scores sont faibles mais relativement proches entre nos 2 échantillons.

Le graphique suivant nous montre sur une journée de l'échantillon de test les écarts entre les temps réels d'arrivée des pompiers et les temps prédits par le modèle *RidgeCV* à différentes heures de la journée.

Graphique 14. *Visualisation Modèle RidgeCV*



- Modèle LassoCV

Le score du modèle sur l'échantillon d'entrainement est de 0.5805036551202245

Le score du modèle sur l'échantillon de test est de 0.5762807159318282

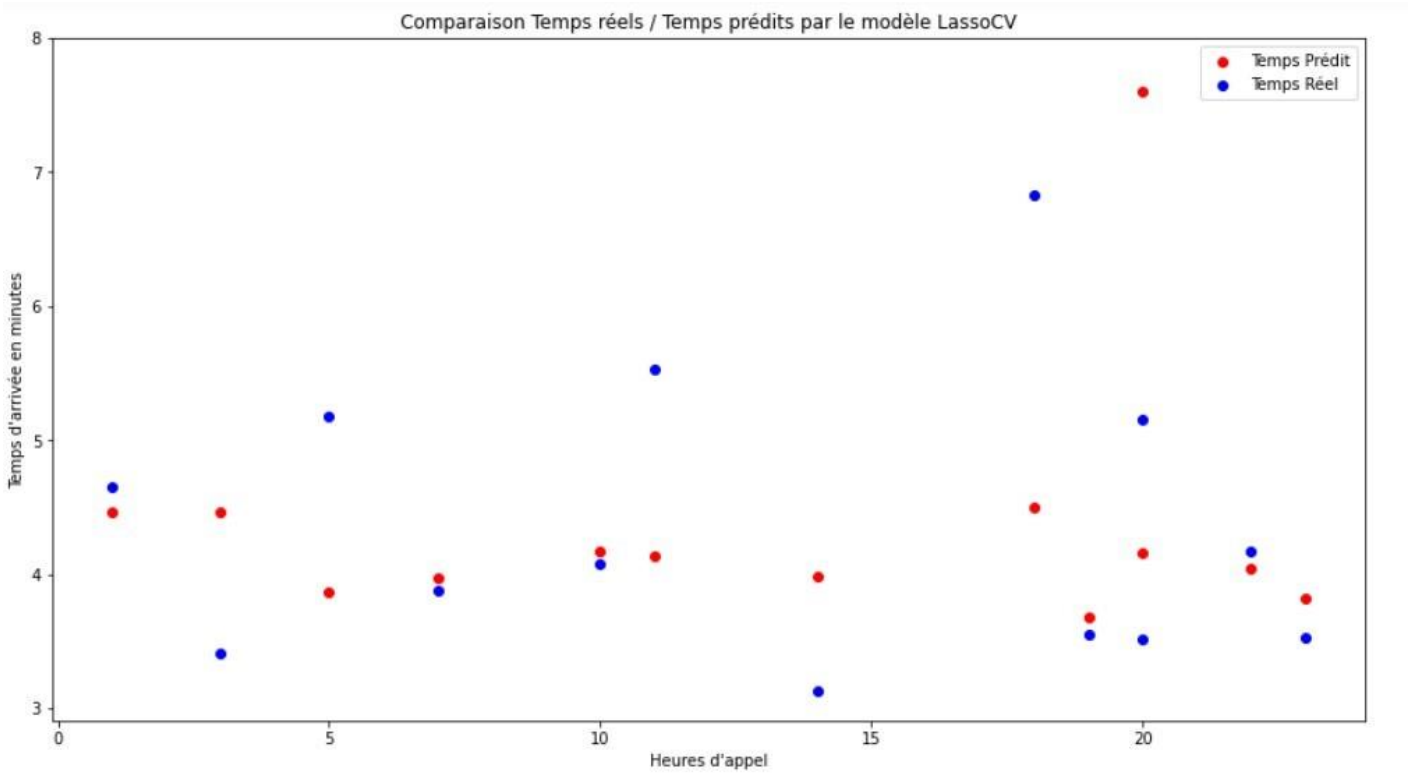
Le mean_squared_error sur l'échantillon d'entrainement est de 0.41852160138784983

Le mean_squared_error sur l'échantillon de test est de 0.4276558080220872

Les scores sont faibles mais relativement proches entre nos 2 échantillons.

Le graphique suivant nous montre sur une journée de l'échantillon de test les écarts entre les temps réels d'arrivée des pompiers et les temps prédits par le modèle *LassoCV* à différentes heures de la journée.

Graphique 15. *Visualisation Modèle LassoCV*



- ## Deuxième itération

Pour améliorer les performances de nos modèles basés sur la Régression Linéaire nous sommes partis de la matrice de corrélation de nos variables numériques. Malgré l'ajout des variables sur le nombre de casernes par quartier et par sous-quartier il fallait construire de nouvelles variables susceptibles d'être retenues par nos modèles de régression.

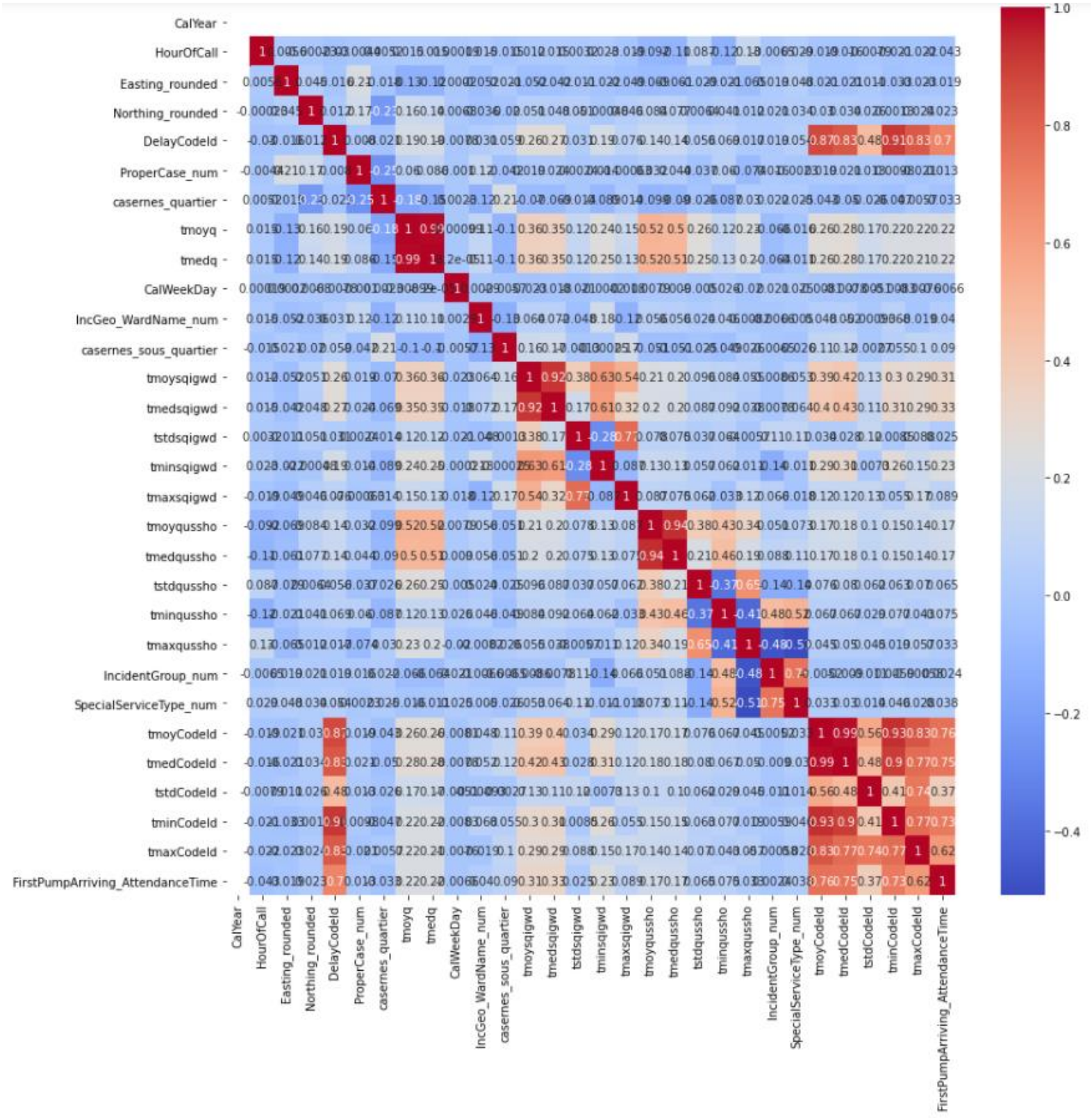
C'est pourquoi nous avons décidé d'ajouter des données statistiques de l'année antérieure à nos échantillons d'entrainement et de test.

Pour cela nous nous sommes référés aux variables identifiées lors de notre phase d'exploration des données (Exploration et Dataviz) et qui montraient des variations significatives de notre variable cible.

Nous avons donc retenu le quartier ('ProperCase'), le sous-quartier ('IncGeo_WardName'), la catégorie d'incident ('IncidentGroup'), le jour de la semaine calculé à partir de la date de l'appel ('DateOfCall'), le type de service spécial ('SpecialServiceType'), l'heure d'appel ('HourOfCall') et enfin le type de retard lorsque renseigné ('DelayCodeId'). A partir de ces variables catégorielles nous avons calculé les temps moyens observés sur l'année antérieure à nos échantillons d'entrainement et de test. Exemple : temps moyens observés en 2019 utilisés pour nos échantillons d'entrainement et de test sur l'année 2020.

La heatmap de corrélation obtenue à l'issue de cette phase d'enrichissement montre bien que les statistiques de l'année antérieure sont susceptibles d'aider nos modèles de prédiction.

Graphique 16. *Heatmap (2ème itération)*



- Modèle LinearRegression – 2ème itération

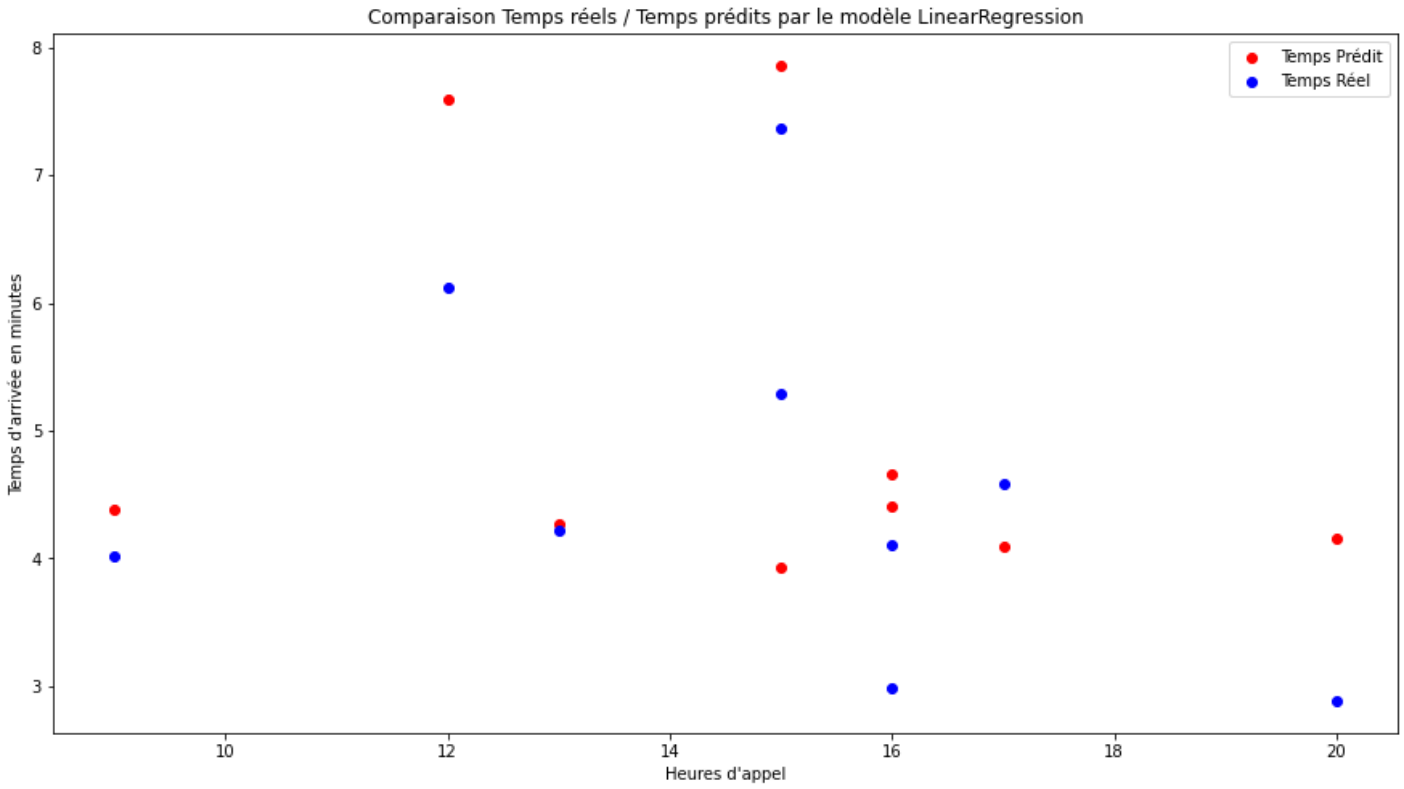
Le coefficient de détermination du modèle sur l'échantillon d'entrainement est de 0.5855081424710049
Le coefficient de détermination du modèle sur l'échantillon de test est de 0.5784596619364113

Le mean_squared_error sur l'échantillon d'entrainement est de 0.4164908522480654
Le mean_squared_error sur l'échantillon de test est de 0.41230732117478597

Les scores sont faibles mais relativement proches entre nos 2 échantillons.

Le graphique suivant nous montre sur le quartier de Merton pour un lundi et pour des services spéciaux de l'échantillon de test les écarts entre les temps réels d'arrivée des pompiers et les temps prédits par le modèle LinearRegression à différentes heures de la journée.

Graphique 17. *Visualisation Modèle LinearRegression (2ème itération)*



- Modèle SGDRegressor – 2ème itération

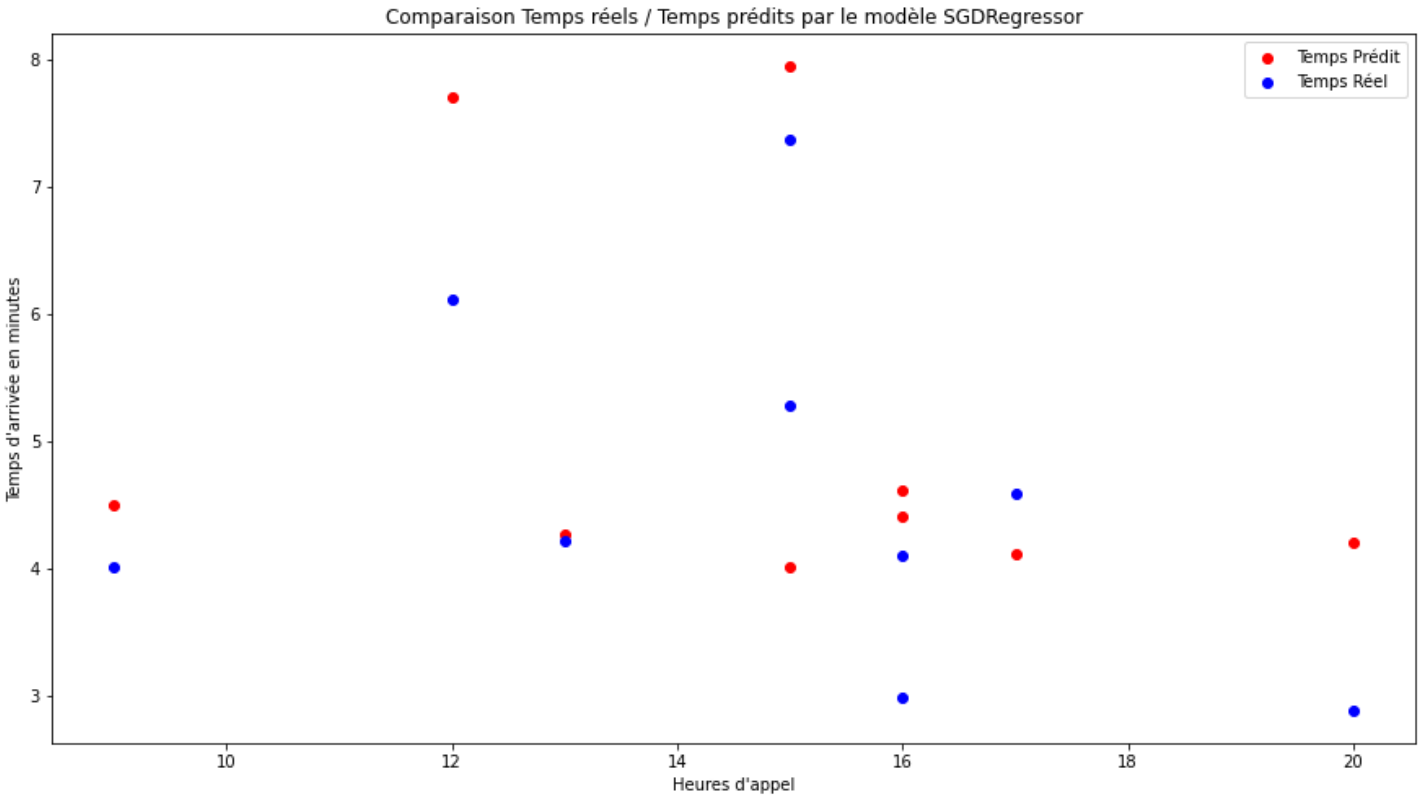
Le score du modèle sur l'échantillon d'entrainement est de 0.583734580320258
Le score du modèle sur l'échantillon de test est de 0.5775630142437822

Le mean_squared_error sur l'échantillon d'entrainement est de 0.41548348577161687
Le mean_squared_error sur l'échantillon de test est de 0.4256085013915583

Les scores sont faibles mais relativement proches entre nos 2 échantillons.

Le graphique suivant nous montre sur le quartier de Merton pour un lundi et pour des services spéciaux de l'échantillon de test les écarts entre les temps réels d'arrivée des pompiers et les temps prédits par le modèle SGDRegressor à différentes heures de la journée.

Graphique 18. *Visualisation Modèle SGDRegressor (2ème itération)*



- Modèle GradientBoostingRegressor – 2ème itération

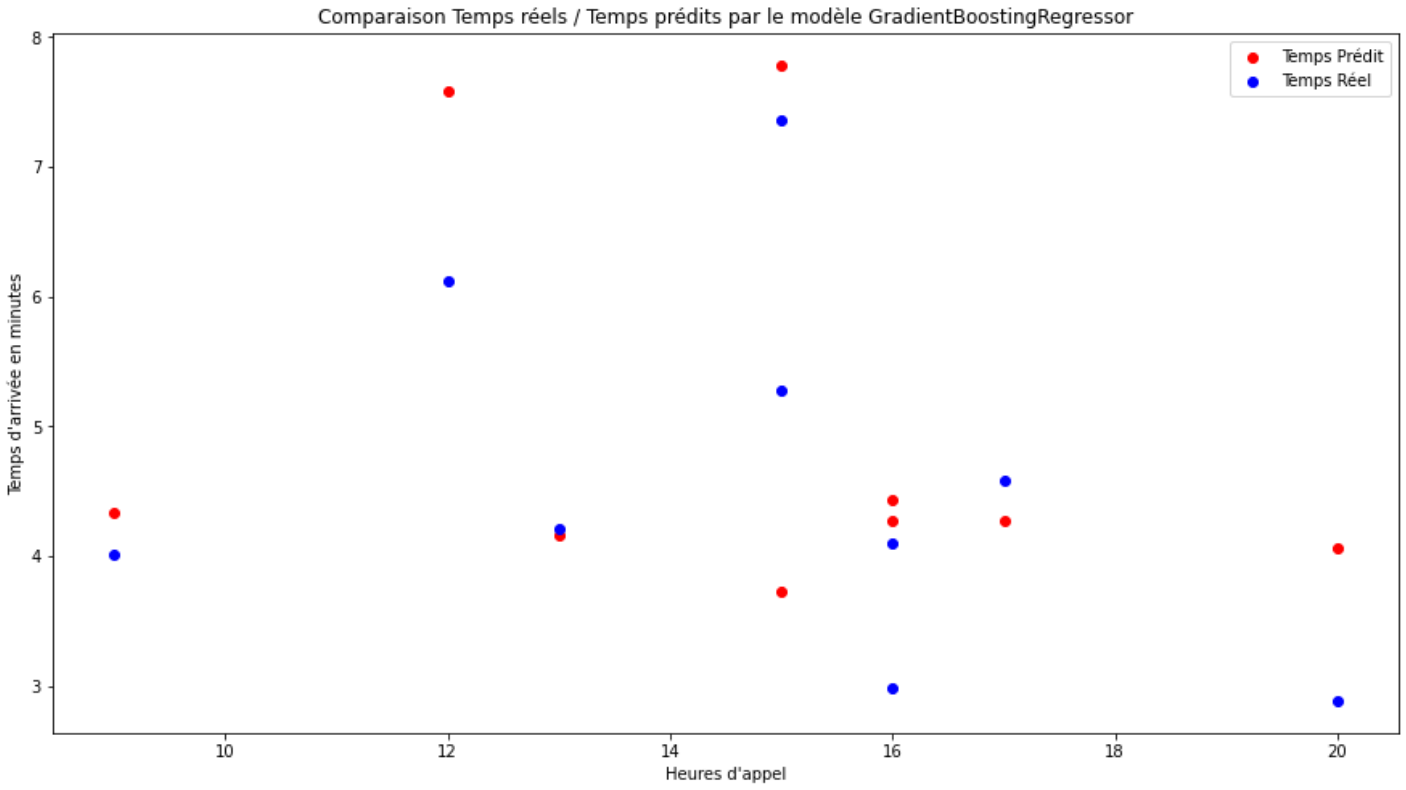
Le score du modèle sur l'échantillon d'entrainement est de 0.6156006822570951
Le score du modèle sur l'échantillon de test est de 0.6110778763441289

Le mean_squared_error sur l'échantillon d'entrainement est de 0.386885187097919
Le mean_squared_error sur l'échantillon de test est de 0.3788480879452526

Les scores et les erreurs quadratiques moyennes sont un peu meilleurs que les modèles précédents. De plus les scores et les erreurs quadratiques moyennes sont très proches entre les échantillons d'entrainement et de test ce qui montre une certaine stabilité du modèle.

Le graphique suivant nous montre sur le quartier de Merton pour un lundi et pour des services spéciaux de l'échantillon de test les écarts entre les temps réels d'arrivée des pompiers et les temps prédits par le modèle GradientBoostingRegressor à différentes heures de la journée.

Graphique 19. *Visualisation Modèle GradientBoostingRegressor (2ème itération)*



## - Bilan 2ème itération -

Le modèle avec les meilleures performances est le modèle *GradientBoostingRegressor*.

Les scores sont légèrement meilleurs que pour les autres modèles.

L'apport des nouvelles colonnes calculées à partir des variables catégorielles significatives a été bénéfique.

Il reste quelques grands écarts entre la réalité et les prédictions qui selon nous proviennent d'aléas externes à nos données.

On observe que de nombreuses prédictions sur 2020 donnent des temps d'arrivée des pompiers supérieures à la réalité probablement à cause des conditions de circulation meilleures en 2020 par rapport à 2019 du fait des effets de la crise sanitaire.

# V. CONCLUSIONS

L'objectif principal du projet, qui est « La prédiction du temps d'intervention des Pompiers de Londres », a été atteint :

Le modèle de Machine Learning que nous avons retenu avec la meilleure performance est le :

- Gradient Boosting Regressor, avec un coefficient de détermination de 62 %.

Ce modèle se rapproche de la réalité, avec une marge de progression potentielle de 38% à ce jour.
Nous pensons que ce modèle pourrait encore être amélioré en confortant les calculs de prédiction avec l'étude des variables et facteurs suivants :

- Superficie / Densité de la population par quartier (règle sur les implantations de casernes de pompiers en fonction de la densité et / ou superficie d'intervention.)
- Informations sur le trafic (ratio de feux rouges par nombre d'habitants, signalétique routière, etc…)
- Kilométrage des routes

L'ensemble de ce projet pourrait être utile dans un futur proche, à la fois pour :

- Outil orienté utilisateur / App :

Aujourd'hui, seul le numéro des secours permet de rentrer en contact avec les pompiers.
Nous pouvons imaginer, demain, qu'une fois raccroché, une application mobile puisse prendre le relais afin de prédire le temps d'arrivée des pompiers, géolocaliser le véhicule et informer l'utilisateur, par exemple, des 1$^{ers}$ gestes de secours à effectuer en attendant l'arrivée sur les lieux et donc de sauver des vies.
Cela pourrait également servir de support de communication à la communauté sur les actions et l'actualité des pompiers de leurs quartiers.

- Outil orienté pompiers :

- Améliorer l'organisation entre casernes (classement de l'efficacité des temps d'arrivée).

- Faciliter et automatiser une partie du travail de la personne chargée de prendre les appels en pouvant prédire le temps d'arrivée aux victimes.

- Proposer notre projet de prédiction à la brigade de pompiers de Paris.

- Appliquer notre modèle à d'autres services comme les ambulances, la police, etc…, où le temps d'intervention joue un rôle crucial.

## VI. DATA SOURCE

Le jeu de données utilisé pour cette étude provient du site officiel du Gouvernement du Royaume-Uni, London Datastore, London Fire Brigade (LFB).

Ces données sont de libre accès comme indiqué ci-dessous :

Le premier jeu de données (“Incidents”) :
https://data.london.gov.uk/dataset/london-fire-brigade-incident-records

Le second jeu de données (“Mobilisation”) :
https://data.london.gov.uk/dataset/london-fire-brigade-mobilisation-records

LFB
LONDON FIRE BRIGADE

### DIFFICULTES RENCONTREES LORS DU PROJET

- Quel a été le principal verrou scientifique rencontré lors de ce projet ?

  Le principal verrou a été de trouver le bon modèle de prédiction.

- Difficultés rencontrées pour le développement du projet et comment elles nous ont ralenti dans notre avancement :

  - Jeux de données : Nous avions dès le départ une très grande volumétrie de données avec un *shape* de 1 038 315 lignes et 60 colonnes. La sélection et le traitement des données pertinentes en lien avec notre objectif a demandé un certain temps.

  - IT : La puissance computationnelle nous a ralenti également du fait de faire tourner des algorithmes avec un jeu de données de ces dimensions dans Python.

  - Temps : Combiner le rythme entre les modules de formation et la mise en pratique du projet a été un grand défi.

# VII. ANNEXES

Dans les annexes de ce projet se trouvent le diagramme de GANTT ainsi que les codes Python pour chaque représentation graphique et tables :

Graphique 20. *Répartition de l'effort du projet sur la durée et dans l'équipe*

## INDEX DES CODES : REPRESENTATIONS GRAPHIQUES ET TABLES

- Dataviz’



- Machine Learning

Première itération



Deuxième itération



- Tables



- Diagramme de Gantt

## Importation et nettoyage rapide des données

```
%matplotlib inline
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_palette("bright")

# Importation
df_inci = pd.read_csv(filepath_or_buffer="LFB_incidents.csv", sep = ";", low_memory=False)
df_mob = pd.read_csv(filepath_or_buffer="LFB_mobilisations.csv", sep = ";", low_memory=False)

# Supression des colonnes inutiles
df_inci_drop = df_inci.drop(["CalYear", "Postcode_full", "UPRN", "USRN", "IncGeo_BoroughName", "IncGeo_BoroughName", "IncGeo_WardName", "Easting_m",
"Northing_m", "Easting_rounded", "Northing_rounded", "FRS", "IncidentStationGround"], axis=1)

df_mob_drop = df_mob.drop(["ResourceMobilisationId", "Resource_Code", "PerformanceReporting", "DateAndTimeMobilised", "TimeMobileTimezoneId",
"TimeArrivedTimezoneId", "AttendanceTimeSeconds", "TimeLeftTimezoneId", "DateAndTimeReturned", "TimeReturnedTimezoneId", "DeployedFromLocation",
"PumpOrder", "PlusCode_Code", "PlusCode_Description"], axis=1)

# Merge
df_lfb = df_inci_drop.merge(df_mob_drop, on = "IncidentNumber", how = "left")

# Conversion des données concernées
df_lfb['DateOfCall']= pd.to_datetime(df_lfb['DateOfCall'])
df_lfb.head()
```

### Figure 1. *Principaux types d'incidents en nombre absolu*

```
# Création et paramètres du graphique
plt.figure(figsize=(7,5))
typeincident = sns.countplot(x="IncidentGroup", data=df_lfb, order = df_lfb['IncidentGroup'].value_counts().index)
plt.title("Nombre de type d'incidents principaux", fontsize=14)
plt.xlabel("Catégorie d'incident", fontsize=12)
plt.ylabel("Count", fontsize=12)
plt.show();
```

### Graphique 2. *Temps d'intervention par type d'incident*

```
#Temps d'intervention par type d'incident
from datetime import datetime
dftime = df_lfb[["StopCodeDescription", "DateAndTimeArrived"]]
dfarr = df_lfb["DateAndTimeArrived"]
dfarr = pd.to_datetime(dfarr)
dftime["arrived"] = dfarr.dt.strftime("%H:%M:%S")
dfdep = df_lfb["DateAndTimeLeft"]
dfdep = pd.to_datetime(dfdep)
dftime["departure"] = dfdep.dt.strftime("%H:%M:%S")
dftime = dftime.drop(columns=["DateAndTimeArrived"])
dftime = dftime.dropna(axis = 0, how = "any")
dftime['diff'] = dftime["departure"].apply(lambda x: datetime.strptime(x,'%H:%M:%S')) - dftime["arrived"].apply(lambda x: datetime.strptime(x,'%H:%M:%S'))
dftime['seconds'] = dftime['diff'].dt.total_seconds()
dftime = dftime.drop(dftime[dftime["seconds"] < 0].index)
value=3600
dftime['min']=(dftime['seconds']/value).round(2)

#Création et paramètres du graphique
sns.catplot(x = 'StopCodeDescription', y = 'min', height=6, aspect=2, s=8, data = dftime)
plt.yticks([0,1,2,3,4,5,6,8,10,12.5,15,17.5,20], fontsize=12)
plt.xticks(rotation = 45, fontsize=12)
plt.title("Temps d'intervention par type d'incident", fontsize=20)
plt.xlabel("Types d'incidents", fontsize=16)
plt.ylabel("Temps en heures", fontsize=16)
plt.show();
```

Graphique 3. *Temps d'arrivée moyen par période de l'année*

```
# Séléction des données qui nous intéressent
dfmeantime = df_lfb[["FirstPumpArriving_AttendanceTime", "DateOfCall"]]
dfmeantime['year'] = dfmeantime['DateOfCall'].dt.year
dfmeantime['month'] = dfmeantime['DateOfCall'].dt.month
dftime = dfmeantime.drop(columns=["DateOfCall"])
dftime = dftime.dropna(axis = 0, how = "any")
dftime['FirstPumpArriving_AttendanceTime'] = dftime['FirstPumpArriving_AttendanceTime']/60
dftime['FirstPumpArriving_AttendanceTime'] = dftime['FirstPumpArriving_AttendanceTime']/10000
dfmeanfinal=dftime.groupby(["month","year"]).sum()
dfmeanfinal=dfmeanfinal.reset_index()
dfmeanfinal = dfmeanfinal.drop(dfmeanfinal[dfmeanfinal["FirstPumpArriving_AttendanceTime"] < 1].index)

# Création et parametres du graphique
fig, ax = plt.subplots(figsize=(20,10))
sns.barplot(x="month", y="FirstPumpArriving_AttendanceTime", hue="year", data=dfmeanfinal, ax=ax)
mois=['Jan','Fev','Mars','Avr',"Mai","Juin",'Juil','Aout','Sep','Oct','Nov','Dec']
ax.set_xticklabels(mois, fontsize=14)
plt.yticks(fontsize=14)
plt.title("Temps d'arrivée moyen par période", fontsize=20)
plt.xlabel(None)
plt.ylabel("Temps d'arrivée en minutes", fontsize=16)
pd.options.mode.chained_assignment = None
plt.legend(prop={'size': 14})
plt.show();
```

Graphique 4. *Évolution du temps moyen d'arrivée des pompiers par secteurs géographiques*

```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns

# Chargement du dataset des incidents
df = pd.read_excel('LFB Incident data from January 2017.xlsx')

# Création nouveau dataframe de travail dfw en retirant les colonnes inutiles pour la dataviz
dfw = df.drop(['Postcode_full', 'UPRN', 'USRN', 'IncGeo_BoroughCode', 'ProperCase', 'IncGeo_WardCode', 'Easting_m', 'Northing_m',
      'Easting_rounded', 'Northing_rounded', 'FRS', 'NumStationsWithPumpsAttending', 'NumPumpsAttending', 'PumpCount',
      'IncGeo_WardNameNew', 'Latitude', 'Longitude', 'PumpHoursRoundUp', 'Notional Cost (£)'], axis=1)

# Utilisation du script pour détecter les colonnes avec des valeurs manquantes
 def valeur_manquante(df):
     flag=0
     for col in df.columns:
          if df[col].isna().sum() > 0:
             flag=1
             print(f"'{col}": {df[col].isna().sum()} valeurs manquantes')
     if flag==0:
        print("Le dataset ne contient plus de valeurs manquantes, bien joué.")
 valeur_manquante(dfw)

# Suppression des lignes avec temps d'arrivée manquant et des lignes avec caserne d'origine manquante
dfw.dropna(subset = ['FirstPumpArriving_AttendanceTime'], axis=0, inplace = True)
dfw.dropna(subset = ['FirstPumpArriving_DeployedFromStation'], axis=0, inplace = True)

# Création nouvelle colonne quartier initialisée à 'aucun'
dfw['quartier'] = 'aucun'

# Valorisation de quartier à partir des cartes London_boroughs et de la carte de découpage des quartiers
dfw.loc[dfw.IncGeo_BoroughName.isin(['BEXLEY','BROMLEY','GREENWICH','LEWISHAM','SOUTHWARK']), 'quartier'] = 'SOUTH EAST'
dfw.loc[dfw.IncGeo_BoroughName.isin(['HAVERING','REDBRIDGE','NEWHAM','BARKING AND DAGENHAM','WALTHAM FOREST', 'TOWER HAMLETS']), 'quartier'] =
'NORTH EAST'
dfw.loc[dfw.IncGeo_BoroughName.isin(['ENFIELD','BARNET','HARINGEY','HACKNEY','ISLINGTON', 'CAMDEN','WESTMINSTER']), 'quartier'] = 'NORTH'
dfw.loc[dfw.IncGeo_BoroughName.isin(['HARROW','HILLINGDON','BRENT','EALING','HOUNSLOW', 'HAMMERSMITH AND FULHAM','KENSINGTON AND
CHELSEA']), 'quartier'] = 'WEST'
dfw.loc[dfw.IncGeo_BoroughName.isin(['RICHMOND UPON THAMES','KINGSTON UPON THAMES','WANDSWORTH','MERTON','LAMBETH',
'SUTTON','CROYDON']), 'quartier'] = 'SOUTH WEST'
dfw.loc[dfw.IncGeo_BoroughName.isin(['CITY OF LONDON']), 'quartier'] = 'CITY'

# Dataviz' Evolution 2017-2020 du Temps
sns.catplot(x='quartier', y='FirstPumpArriving_AttendanceTime',
          kind='boxen', height=9, hue='CalYear',
          data=dfw)
plt.title('Evolution 2017-2020 du Temps d\'arrivée des pompiers par Quartiers')
plt.xlabel('Quartiers')
plt.ylabel('Temps d\'arrivée des pompiers (minutes
plt.yticks([180, 240, 300, 360, 420, 480, 540, 600, 900, 1200], [3, 4, 5, 6, 7, 8, 9, 10, 15, 20]);
```

Graphique 5. *Temps moyen d'arrivée des pompiers par caserne*

```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns

# Chargement du dataset des incidents
df = pd.read_excel('LFB Incident data from January 2017.xlsx')

# Création nouveau dataframe de travail dfw en retirant les colonnes inutiles pour la dataviz
dfw = df.drop(['Postcode_full', 'UPRN', 'USRN', 'IncGeo_BoroughCode', 'ProperCase', 'IncGeo_WardCode', 'Easting_m', 'Northing_m',
      'Easting_rounded', 'Northing_rounded', 'FRS', 'NumStationsWithPumpsAttending', 'NumPumpsAttending', 'PumpCount',
      'IncGeo_WardNameNew', 'Latitude', 'Longitude', 'PumpHoursRoundUp', 'Notional Cost (£)'], axis=1)

# Utilisation du script pour détecter les colonnes avec des valeurs manquantes
  def valeur_manquante(df):
      flag=0
      for col in df.columns:
            if df[col].isna().sum() > 0:
               flag=1
               print(f'"{col}": {df[col].isna().sum()} valeurs manquantes')
      if flag==0:
         print("Le dataset ne contient plus de valeurs manquantes, bien joué.")
  valeur_manquante(dfw)

# Suppression des lignes avec temps d'arrivée manquant et des lignes avec caserne d'origine manquante
dfw.dropna(subset = ['FirstPumpArriving_AttendanceTime'], axis=0, inplace = True)
dfw.dropna(subset = ['FirstPumpArriving_DeployedFromStation'], axis=0, inplace = True)

# Récupération des noms des casernes les plus rapides et les plus lentes
# On ne prend que les incidents pour lesquels la caserne qui est intervenue est celle du secteur de l'incident pour ne pas défavoriser les casernes intervenues ailleurs
res = pd.DataFrame(dfw[dfw.IncidentStationGround == dfw.FirstPumpArriving_DeployedFromStation].groupby(['FirstPumpArriving_DeployedFromStation'], as_index =
False).agg({'FirstPumpArriving_AttendanceTime' : 'median'}) )
print("Récupération des 3 casernes les plus rapides :\n")
display(res.sort_values('FirstPumpArriving_AttendanceTime', ascending=True).head(3))
print("Récupération des 3 casernes les plus lentes :\n")
display(res.sort_values('FirstPumpArriving_AttendanceTime', ascending=False).head(3))

# Récupération des infos pour la dataviz dans res_h
res_h = pd.DataFrame(dfw[dfw.IncidentStationGround == dfw.FirstPumpArriving_DeployedFromStation].groupby(['FirstPumpArriving_DeployedFromStation',
'HourOfCall'], as_index = False).agg({'FirstPumpArriving_AttendanceTime' : 'mean'}) )

# Dataviz' Temps d'arrivée moyen selon l'heure de la journée par caserne (3 plus rapides vs 3 plus lentes)
plt.figure(figsize=(6,20))
sns.relplot(x = 'HourOfCall', y = 'FirstPumpArriving_AttendanceTime',kind='line',
          palette = 'viridis', height=8,
          data = res_h[res_h.FirstPumpArriving_DeployedFromStation.isin(['Orpington', 'Wennington', 'Ruislip', 'Lewisham', 'Brixton', 'Whitechapel'])],
          hue = 'FirstPumpArriving_DeployedFromStation' )
plt.title('Temps d\'arrivée moyen selon l\'heure de la journée par caserne (3 plus rapides vs 3 plus lentes)')
plt.xlabel('Heure d\'appel')
plt.ylabel('Temps d\'arrivée moyen (minutes)')
plt.yticks([240, 300, 360, 420, 480, 540], [4, 5, 6, 7, 8, 9])
plt.xticks([0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24], ['0','2','4','6','8','10','Midi','14','16','18','20','22','Minuit']);
```

## Graphique 6. *Motifs des retards*

```
# Séléction des données qui nous intéressent
counts = df_lfb['DelayCode_Description'].value_counts()
dfdelay = df_lfb.loc[df_lfb['DelayCode_Description'].isin(counts.index[counts < 25000])]
dfdelaylast = dfdelay['DelayCode_Description'].value_counts()

# Création et paramètres du graphique
dfdelaylast.plot.pie(y='dfdelaylast', figsize=(10.5, 10.),labels=None,  autopct = lambda x: str(round(x,)) + '%',colors =
['#82ff84','#f9baff','#f7b660','#ccff99','#ff9999','#99ffec','#bad5ff','#ffee99','#6ebbff'], pctdistance = 0.83)
centre_circle = plt.Circle((0,0),0.7,fc='white')
fig = plt.gcf()
fig.gca().add_artist(centre_circle)
plt.title("Motifs des retards en %", fontsize=15)
plt.ylabel(None)
labels=dfdelaylast.index.unique()
plt.legend(labels, loc="center",title = "Raisons des retards :", title_fontsize='large', prop={'size': 11})
plt.show();
```

## Graphique 7. *Types de retard par jour de la semaine*

```
# Séléction des données qui nous interessent
dflate = df_lfb[["HourOfCall", "DateOfCall", "DelayCode_Description"]]
dflate['day'] = pd.to_datetime(dflate["DateOfCall"])
dflate['day'] = dflate['day'].apply(lambda x: x.weekday())
dflate = dflate.dropna(axis = 0, how = "any")
dflate = dflate.drop(columns=["DateOfCall"])
dflate = dflate.loc[dflate['DelayCode_Description'].isin(counts.index[counts < 25000])]
dflate['DelayCode_Description'] = dflate['DelayCode_Description'].replace({'At drills when mobilised':'Others', 'On outside duty when mobilised':'Others',
'Appliance/Equipment defect':'Others', 'Weather conditions':'Others', 'Mob/Radio problems when mobilised':'Others', 'Arrived but held up - Other reason':'Others'})

# Création du graphique
fig, ax = plt.subplots(figsize=(17,8))
sns.countplot(x='day', hue = 'DelayCode_Description', palette=["#82ff84", "#6ebbff", "#f7b660", "#f9baff"], data=dflate)
jours=['Lundi','Mardi','Mercredi','Jeudi',"Vendredi","Samedi",'Dimanche']
ax.set_xticklabels(jours, fontsize=12)
plt.title("Types de retard par jour de la semaine", fontsize=16)
plt.ylabel("Retards comptabilisés", fontsize=14)
plt.xlabel(None)
plt.legend(prop={'size': 12}, title = "Raisons des retards :", title_fontsize='large')
plt.show();
```

## Graphique 8. *Densités des types de retard par heure de la journée*

```
# Création du graphique reprenant les informations du graphique précédent
fig, ax = plt.subplots(figsize=(13,7))
sns.kdeplot(x='HourOfCall', hue = 'DelayCode_Description', palette=["#82ff84", "#6ebbff", "#f7b660", "#f9baff"], cut=1, linewidth=7, data=dflate)
plt.yticks(fontsize=12)
plt.xticks([0,2,4,6,8,10,12,14,16,18,20,22,24],fontsize=12)
plt.title("Types de retard par heure de la journée", fontsize=16)
plt.xlabel("Heures", fontsize=14)
plt.ylabel("Nombre de retard comptabilisé X10", fontsize=14)
liste=['Address incomplete/wrong','Traffic calming measures', 'Others','Traffic, roadworks, etc']
plt.legend(liste, loc='upper left', prop={'size': 12},title = "Raisons des retards :", title_fontsize='large')
plt.show();
```

## Graphique 9. *Temps moyens avec et sans motifs de retard*

```
# Sélection des données qui nous intéressent pour le delta du temps des interventions sans retard
dfsd = df_lfb[['DateAndTimeMobile', 'DateAndTimeArrived', 'DelayCode_Description']]
dfsd['DelayCode_Description']=dfsd['DelayCode_Description'].replace('Not held up', 1)
dfsd['DelayCode_Description']=dfsd['DelayCode_Description'].fillna(1)
dfsd = dfsd.dropna(axis = 0, how = "any")
dfsd = (dfsd.loc[dfsd['DelayCode_Description'] == 1])

dfsd['DateAndTimeMobile'] = pd.to_datetime(dfsd['DateAndTimeMobile'])
dfsd['DateAndTimeMobile']= dfsd['DateAndTimeMobile'].dt.strftime("%H:%M:%S")
dfsd['DateAndTimeArrived'] = pd.to_datetime(dfsd['DateAndTimeArrived'])
dfsd['DateAndTimeArrived']= dfsd['DateAndTimeArrived'].dt.strftime("%H:%M:%S")

dfsd['diff'] = dfsd['DateAndTimeArrived'].apply(lambda x: datetime.strptime(x,'%H:%M:%S'))  - dfsd['DateAndTimeMobile'].apply(lambda x: datetime.strptime(x,'%H:%M:%S'))
dfsd['diff seconds'] = dfsd['diff'].dt.total_seconds()
dfsd = dfsd.drop(dfsd[dfsd["diff seconds"] < 0].index)
value=60
dfsd['min']=(dfsd['diff seconds']/value).round(2)
dfsd = dfsd.drop(columns=["DateAndTimeMobile", "DateAndTimeArrived", "diff", 'diff seconds'])

# Séléction des données qui nous intéressent pour le delta du temps des interventions avec retards
dfad = df_lfb[['DateAndTimeMobile', 'DateAndTimeArrived', 'DelayCode_Description']]
dfad['DelayCode_Description']=dfad['DelayCode_Description'].replace('Not held up', 1)
dfad['DelayCode_Description']=dfad['DelayCode_Description'].fillna(1)
dfad = dfad.dropna(axis = 0, how = "any")
dfad = (dfad.loc[dfad['DelayCode_Description'] != 1])

dfad['DateAndTimeMobile'] = pd.to_datetime(dfad['DateAndTimeMobile'])
dfad['DateAndTimeMobile']= dfad['DateAndTimeMobile'].dt.strftime("%H:%M:%S")
dfad['DateAndTimeArrived'] = pd.to_datetime(dfad['DateAndTimeArrived'])
dfad['DateAndTimeArrived']= dfad['DateAndTimeArrived'].dt.strftime("%H:%M:%S")

dfad['diff'] = dfad['DateAndTimeArrived'].apply(lambda x: datetime.strptime(x,'%H:%M:%S'))  - dfad['DateAndTimeMobile'].apply(lambda x: datetime.strptime(x,'%H:%M:%S'))
dfad['diff seconds'] = dfad['diff'].dt.total_seconds()
dfad = dfad.drop(dfad[dfad["diff seconds"] < 0].index)
value=60
dfad['min']=(dfad['diff seconds']/value).round(2)
dfad = dfad.drop(columns=["DateAndTimeMobile", "DateAndTimeArrived", "diff", 'diff seconds'])

# Création du graphique
AD = dfad["min"].mean()
AD = np.round(AD,0)
SD = dfsd["min"].mean()
SD = np.round(SD,0)
Delta = AD – SD

fig, ax = plt.subplots()
fig.set_size_inches(8, 6)
bar_x = ['Avec retard', 'Sans retard', 'Delta']
bar_height = [AD,SD,Delta]
bar_tick_label = ['Avec retard', 'Sans retard', 'Delta']
bar_label = [AD,SD,Delta]

bar_plot = plt.bar(bar_x,bar_height,tick_label=bar_tick_label)

def autolabel(rects):
    for idx,rect in enumerate(bar_plot):
        height = rect.get_height()
        ax.text(rect.get_x() + rect.get_width()/2., 1.05*height,
                bar_label[idx],
                ha='center', va='bottom', rotation=0)

autolabel(bar_plot)

plt.ylim(0,8)
ax.set_ylabel('Temps moyens de trajet en minutes', fontsize=14)
ax.set_title('Temps moyens avec et sans motif de retard', fontsize=15)
ax.bar(x,y, color = ["#ff8282", '#82ff84', '#a1a1a1'])
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
plt.show();
```

## Graphique 10. *Temps de mobilisation par arrondissement*

```
# Séléction des données qui nous intéressent
dfrespbri = df_lfb[['ProperCase', 'TimeOfCall', 'DateAndTimeMobile']]
dfrespbri['DateAndTimeMobile'] = pd.to_datetime(dfrespbri['DateAndTimeMobile'])
dfrespbri['DateAndTimeMobile'] = dfrespbri['DateAndTimeMobile'].dt.strftime("%H:%M:%S")
dfrespbri = dfrespbri.dropna(axis = 0, how = "any")

dfrespbri['diff'] = dfrespbri['DateAndTimeMobile'].apply(lambda x: datetime.strptime(x,'%H:%M:%S')) - dfrespbri['TimeOfCall'].apply(lambda x:
datetime.strptime(x,'%H:%M:%S'))
dfrespbri['diff seconds'] = dfrespbri['diff'].dt.total_seconds()
dfrespbri = dfrespbri.drop(dfrespbri[dfrespbri["diff seconds"] < 0].index)
dfrespbri = dfrespbri.drop(columns=["TimeOfCall", "DateAndTimeMobile", "diff"])
dfrespbri = dfrespbri.drop(dfrespbri[dfrespbri["diff seconds"] > 10].index)
dfrespbri = dfrespbri.groupby("ProperCase").mean()
dfrespbri = dfrespbri.sort_values(by = 'diff seconds', ascending = False)
dfrespbri = dfrespbri.reset_index()
dfrespbri['diff seconds'] = dfrespbri['diff seconds']*10

# Création du graphique
ax = dfrespbri[['ProperCase','diff seconds']].plot(kind='bar', figsize=(15, 10), fontsize=12, color="#8FDFFF")
ax.set_title('Temps de mobilisation par arrondissement', fontsize=15)
ax.get_legend().remove()
ax.set_xticklabels(dfrespbri.ProperCase)
ax.set_ylabel('Temps en seconde', fontsize=14)
plt.yticks([0,10,20,30,40,50,60,70,80,90,100],fontsize=12)
plt.show();
```

## Graphique 11. *Temps d'intervention par type d'incident*

```
# Séléction des données qui nous intéressent
dfrespbri = df_lfb[['ProperCase', 'TimeOfCall', 'DateAndTimeMobile']]
#Séléction des données qui nous interessent
dftpi = df_lfb[['TimeOfCall','DateAndTimeArrived' ,'DelayCode_Description', 'StopCodeDescription']]
dftpi['DelayCode_Description']=dftpi['DelayCode_Description'].replace('Not held up', 1)
dftpi['DelayCode_Description']=dftpi['DelayCode_Description'].fillna(1)
dftpi = dftpi.dropna(axis = 0, how = "any")
dftpi = (dftpi.loc[dftpi['DelayCode_Description'] == 1])
dftpi['DateAndTimeArrived'] = pd.to_datetime(dftpi['DateAndTimeArrived'])
dftpi['DateAndTimeArrived']= dftpi['DateAndTimeArrived'].dt.strftime("%H:%M:%S")dftpi['diff'] = dftpi['DateAndTimeArrived'].apply(lambda x:
datetime.strptime(x,'%H:%M:%S')) - dftpi['TimeOfCall'].apply(lambda x: datetime.strptime(x,'%H:%M:%S'))
dftpi['diff seconds'] = dftpi['diff'].dt.total_seconds()
dftpi = dftpi.drop(dftpi[dftpi["diff seconds"] < 0].index)value=60
dftpi['min']=(dftpi['diff seconds']/value).round(2)
dftpi = dftpi.drop(dftpi[dftpi["min"] > 15].index)
dftpi = dftpi.drop(columns=["TimeOfCall", "DateAndTimeArrived", "diff", "DelayCode_Description","diff seconds"])
dftpi = dftpi.groupby("StopCodeDescription").mean()
dftpi = dftpi.sort_values(by = 'min', ascending = False)
dftpi = dftpi.reset_index()

# Création du graphique
ax = dftpi[['StopCodeDescription','min']].plot(kind='bar', figsize=(12, 8), fontsize=12, color="#FFD000")
ax.set_title("Temps de mobilisation et de trajet par type d'incident", fontsize=15)
ax.get_legend().remove()
ax.set_xticklabels(dftpi.StopCodeDescription)
ax.set_ylabel('Temps en minute', fontsize=14)
plt.yticks([0,1,2,3,4,5,6],fontsize=12)
plt.show();
```

Graphique 12. *Heatmap (1ère itération)*

```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import Ridge, LassoCV

# Chargement du dataset des Incidents
dfi = pd.read_excel('LFB Incident data from January 2017.xlsx')

# Chargement du dataset des Mobilisations
dfm = pd.read_excel('LFB Mobilisation data from January 2017.xlsx')

# Fusion des datasets sur la colonne commune IncidentNumber (on ne garde que les incidents en commun avec inner)
fusion = dfi.merge(right = dfm, on = 'IncidentNumber', how = 'inner')

# chargement du dataset de travail à partir de la fusion
# On ne retient que les lignes qui correspondent au premier camion (PumpOrder = 1) de la première équipe de pompiers arrivée sur les lieux
dfw = fusion[(fusion.FirstPumpArriving_DeployedFromStation == fusion.DeployedFromStation_Name)&(fusion.PumpOrder == 1)]

# on souhaite ajouter au dataset le nombre de casernes du quartier de l'incident (ex. dans le quartier de Westminster il y a 8 casernes)
# dans le but d'enrichir notre dataset afin d'aider les modèles de prédiction
df_quartiers = pd.DataFrame(columns=('ProperCase','casernes_quartier'))

for indice, quartier in enumerate(dfw.ProperCase.unique()):
    # pour chaque quartier on récupère son nom
    df_quartiers.loc[indice,'ProperCase'] = quartier
    # on calcule le nombre de casernes différentes du quartier
    nb_cas_pot = dfw[dfw.ProperCase == quartier].IncidentStationGround.nunique()
    # on l'enregistre dans le nouveau dataframe
    df_quartiers.loc[indice,'casernes_quartier'] = nb_cas_pot

# on ajoute la colonne casernes_quartier au dataframe de travail par un merge sur la colonne commune ProperCase
dfw = dfw.merge(right = df_quartiers, on = 'ProperCase' , how = 'inner')

# on souhaite ajouter au dataset le nombre de casernes du sous quartier de l'incident (ex. dans le sous quartier de Barkingside
# dans le but d'enrichir notre dataset afin d'aider les modèles de prédiction
df_sous_quartiers = pd.DataFrame(columns=('IncGeo_WardName','casernes_sous_quartier'))

for indice, sous_quartier in enumerate(dfw.IncGeo_WardName.unique()):
    # pour chaque sous quartier on récupère son nom
    df_sous_quartiers.loc[indice,'IncGeo_WardName'] = sous_quartier
    # on calcule le nombre de casernes différentes du sous quartier
    nb_cas_pot = dfw[dfw.IncGeo_WardName == sous_quartier].IncidentStationGround.nunique()
    # on l'enregistre dans le nouveau dataframe
    df_sous_quartiers.loc[indice,'casernes_sous_quartier'] = nb_cas_pot

# on ajoute la colonne casernes_sous_quartier au dataframe de travail par un merge sur la colonne commune IncGeo_WardName
dfw = dfw.merge(right = df_sous_quartiers, on = 'IncGeo_WardName' , how =
```

```
# on supprime les colonnes inutiles
dfw = dfw.drop(['IncidentNumber','TimeOfCall','PropertyType','Postcode_full', 'Postcode_district','UPRN', 'USRN', 'IncGeo_BoroughCode','IncGeo_BoroughName',
            'IncGeo_WardCode', 'IncGeo_WardName','IncGeo_WardNameNew','Easting_m','Northing_m','Latitude','Longitude','FRS',
            'SecondPumpArriving_AttendanceTime','SecondPumpArriving_DeployedFromStation',
            'NumStationsWithPumpsAttending','NumPumpsAttending','PumpCount','PumpHoursRoundUp','Notional Cost (£)',
            'ResourceMobilisationId','Resource_Code','PerformanceReporting','DateAndTimeMobilised',
            'DateAndTimeMobile', 'TimeMobileTimezoneId', 'DateAndTimeArrived',
            'TimeArrivedTimezoneId', 'AttendanceTimeSeconds', 'DateAndTimeLeft',
            'TimeLeftTimezoneId', 'DateAndTimeReturned', 'TimeReturnedTimezoneId',
            'DeployedFromStation_Code', 'DeployedFromStation_Name','PumpOrder', 'PlusCode_Code','PlusCode_Description', 'DelayCode_Description']
           , axis=1)

# on transforme le type des variables numériques object en entiers
dfw.casernes_quartier = dfw.casernes_quartier.astype('int64')
dfw.casernes_sous_quartier = dfw.casernes_sous_quartier.astype('int64')

# on rajoute le jour de la semaine, le jour et le mois de l'appel avant de supprimer les variables de type datetime
dfw['CalWeekDay'] = pd.to_datetime(dfw['DateOfCall']).dt.weekday
dfw['CalDay'] = pd.to_datetime(dfw['DateOfCall']).dt.day
dfw['CalMonth'] = pd.to_datetime(dfw['DateOfCall']).dt.month

# La variable DelayCodeId est une catégorielle donc on change son type en chaine de caractères
dfw.DelayCodeId = dfw.DelayCodeId.astype('str')

# on récupère dans num_data toutes les colonnes numériques
num_data = dfw.select_dtypes(include=['float64','int64'])
# print(len(num_data.columns),"colonnes numériques")
# pour chaque colonne numérique
for col in num_data:
   # on la supprime du dataset de travail
   dfw = dfw.drop(col, axis=1)

# on réorganise l'ordre des colonnes de num_data pour avoir la variable cible en dernier
num_d1 = num_data[['CalYear','HourOfCall','Easting_rounded','Northing_rounded']]
num_d2 = num_data[['casernes_quartier','casernes_sous_quartier','CalWeekDay','CalDay','CalMonth']]
num_d3 = num_data[['FirstPumpArriving_AttendanceTime']]

num_data = num_d1.join(num_d2.join(num_d3))

# on affiche la heatmap des corrélations
cor = num_data.corr()

fig, ax = plt.subplots(figsize=(12,12))
sns.heatmap(cor, annot= True, ax= ax, cmap="coolwarm");

from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import Ridge, LassoCV

# on normalise les variables numériques avec StandardScaler
scaler = preprocessing.StandardScaler().fit(num_data)
num_data[num_data.columns] = pd.DataFrame(scaler.transform(num_data), columns=num_data.columns, index= num_data.index)

# on rajoute à dfw les colonnes numériques normalisées
dfw = dfw.join(num_data)

# on récupère dans cat_data toutes les colonnes catégorielles
cat_data = dfw.select_dtypes(include=['O'])
#print(len(cat_data.columns),"colonnes catégorielles")
```

```
# pour chaque colonne catégorielle
for col in cat_data:
    # on crée des variables indicatrices autant que de valeurs différentes et on nomme les nouvelles colonnes
    # avec comme préfixe le nom de la colonne d'origine
    dfw = dfw.join(pd.get_dummies(dfw[col], prefix=col))
    # on supprime la colonne initiale de dfw pour plus tard joindre toutes les variables indicatrices de cat_data à dfw
    dfw = dfw.drop(col, axis=1)

# on sépare la cible dans target et les autres colonnes de dfw dans feats (on en profite pour supprimer la colonne DateOfCall)
target = dfw.FirstPumpArriving_AttendanceTime
feats = dfw.drop(['DateOfCall',  'FirstPumpArriving_AttendanceTime'], axis=1)

# on découpe les données de dfw en échantillons d'entrainement (80%) et de test final (20%)
X_train, X_test, y_train, y_test = train_test_split(feats, target, test_size=0.2)print(table)
```

Graphique 13. *Visualisation Modèle LinearRegression (1ère itération)*

```
from sklearn.linear_model import LinearRegression

# création du modèle
lr = LinearRegression()
lr.fit(X_train, y_train)


# R**2 entrainement + proche de 1 possible
print("Coefficient de détermination du modèle :", lr.score(X_train, y_train))

## Coefficient de détermination du modèle : 0.5839818857279384

# R**2 test + proche de 1 possible
lr.score(X_test,y_test)

## 0.5786472154140925

# on stocke les prédictions du modèle pour X_test dans pred_test
pred_test = lr.predict(X_test)
# plt.scatter(pred_test, y_test)
# plt.plot((y_test.min(),y_test.max()), (y_test.min(),y_test.max()))

# Reconstitution de l'échantillon de test avec la colonne cible dans le but de dénormaliser pour visualiser les écarts entrainement / test
num_X1 = X_test[['CalYear','HourOfCall','Easting_rounded','Northing_rounded','casernes_quartier','casernes_sous_quartier','CalWeekDay','CalDay','CalMonth']]
num_X3 = pd.DataFrame(y_test)[['FirstPumpArriving_AttendanceTime']]

# num_X_test avec la cible réelle FirstPumpArriving_AttendanceTime
num_X_test = num_X1.join(num_X3)

# Reconstitution de l'échantillon de test avec la colonne prédite dans le but de dénormaliser pour visualiser les écarts entrainement / test
num_X1 = X_test[['CalYear','HourOfCall','Easting_rounded','Northing_rounded','casernes_quartier','casernes_sous_quartier','CalWeekDay','CalDay','CalMonth']]
num_X4 = pd.DataFrame(pred_test, columns=['FirstPumpArriving_AttendTime_pred'])

# num_X_pred avec la cible prédite FirstPumpArriving_AttendTime_pred
num_X_pred = num_X1.join(num_X4)

# on dénormalise les deux dataframes
num_X_test_inverse = pd.DataFrame(scaler.inverse_transform(num_X_test), columns=num_X_test.columns, index= num_X_test.index)
num_X_pred_inverse = pd.DataFrame(scaler.inverse_transform(num_X_pred), columns=num_X_pred.columns, index= num_X_pred.index)

# on rajoute la colonne cible prédite à num_X_test_inverse
num_X_test_inverse = num_X_test_inverse.join(num_X_pred_inverse.FirstPumpArriving_AttendTime_pred)

# On extrait une journée du test avec la cible et la prédiction
res = num_X_test_inverse[(-pd.isnull(num_X_test_inverse.FirstPumpArriving_AttendTime_pred))&(num_X_test_inverse.CalYear == 2020)&(num_X_test_inverse.CalMon
th == 11)&(num_X_test_inverse.CalDay == 14)][['CalDay','CalMonth','HourOfCall', 'FirstPumpArriving_AttendanceTime','FirstPumpArriving_AttendTime_pred' ]].sort_valu
es(['CalDay','HourOfCall']).head(30)

# on affiche les écarts dans un graphique pour cette journée
plt.figure(figsize=(15,8))
plt.scatter(res.HourOfCall, res.FirstPumpArriving_AttendTime_pred, c='r', label='Temps Prédit')
plt.scatter(res.HourOfCall, res.FirstPumpArriving_AttendanceTime, c='b', label='Temps Réel')
plt.xlabel('Heures d\'appel')
plt.ylabel('Temps d\'arrivée en minutes')
plt.yticks([180,240,300,360,420,480],[3,4,5,6,7,8])
plt.title('Comparaison Temps réels / Temps prédits par le modèle LinearRegression')
plt.legend();
```

Graphique 14. *Visualisation Modèle RidgeCV (1ère itération)*

```
from sklearn.linear_model import RidgeCV

# création du modèle
ridge_reg = RidgeCV(alphas= (0.001, 0.01, 0.1, 0.3, 0.7, 1, 10, 50, 100))
ridge_reg.fit(X_train, y_train)

print( "alpha sélectionné par c-v :" ,ridge_reg.alpha_)
print("score train :", ridge_reg.score(X_train, y_train))
print("score test :", ridge_reg.score(X_test, y_test))

## alpha sélectionné par c-v : 10.0
## score train : 0.5839438555837259
## score test : 0.5787123523986961

ridge_pred_train = ridge_reg.predict(X_train)
ridge_pred_test = ridge_reg.predict(X_test)

print("mse train:", mean_squared_error(ridge_pred_train, y_train))
print("mse test:", mean_squared_error(ridge_pred_test, y_test))

## mse train: 0.4150893945887835
## mse test: 0.4252015806664894

# on stocke les prédictions du modèle pour X_test dans pred_test
pred_test = ridge_reg.predict(X_test)
# plt.scatter(pred_test, y_test)
# plt.plot((y_test.min(),y_test.max()), (y_test.min(),y_test.max()))

# Reconstitution de l'échantillon de test avec la colonne cible dans le but de dénormaliser pour visualiser les écarts entrainement / test
num_X1 = X_test[['CalYear','HourOfCall','Easting_rounded','Northing_rounded','casernes_quartier','casernes_sous_quartier','CalWeekDay','CalDay','CalMonth']]
num_X3 = pd.DataFrame(y_test)[['FirstPumpArriving_AttendanceTime']]

# num_X_test avec la cible réelle FirstPumpArriving_AttendanceTime
num_X = num_X1.join(num_X3)

# Reconstitution de l'échantillon de test avec la colonne prédite dans le but de dénormaliser pour visualiser les écarts entrainement / test
num_X1 = X_test[['CalYear','HourOfCall','Easting_rounded','Northing_rounded','casernes_quartier','casernes_sous_quartier','CalWeekDay','CalDay','CalMonth']]
num_X4 = pd.DataFrame(pred_test, columns=['FirstPumpArriving_AttendTime_pred'])

# num_X_pred avec la cible prédite FirstPumpArriving_AttendTime_pred
num_X_pred = num_X1.join(num_X4)

# on dénormalise les deux dataframes
num_X_inverse = pd.DataFrame(scaler.inverse_transform(num_X), columns=num_X.columns, index= num_X.index)
num_X_pred_inverse = pd.DataFrame(scaler.inverse_transform(num_X_pred), columns=num_X_pred.columns, index= num_X_pred.index)

# on rajoute la colonne cible prédite à num_X_test_inverse
num_X_inverse = num_X_inverse.join(num_X_pred_inverse.FirstPumpArriving_AttendTime_pred)

# display(num_X_inverse[(-pd.isnull(num_X_inverse.FirstPumpArriving_AttendTime_pred))&(num_X_inverse.CalYear == 2020)&(num_X_inverse.CalMonth ==
11)&(num_X_inverse.CalDay ==14)][['CalDay','CalMonth','HourOfCall', 'FirstPumpArriving_AttendanceTime','FirstPumpArriving_AttendTime_pred' ]].sort_valu
es(['CalDay','HourOfCall']).head(30))
res = num_X_inverse[(-pd.isnull(num_X_inverse.FirstPumpArriving_AttendTime_pred))&(num_X_inverse.CalYear == 2020)&(num_X_inverse.CalMonth == 11)&(num_X_
inverse.CalDay ==14)][['CalDay','CalMonth','HourOfCall', 'FirstPumpArriving_AttendanceTime','FirstPumpArriving_AttendTime_pred' ]].sort_values(['CalDay','HourOfCall'])
.head(30)

# on affiche les écarts dans un graphique pour cette journée
plt.figure(figsize=(15,8))
plt.scatter(res.HourOfCall, res.FirstPumpArriving_AttendTime_pred, c='r', label='Temps Prédit')
plt.scatter(res.HourOfCall, res.FirstPumpArriving_AttendanceTime, c='b', label='Temps Réel')
plt.xlabel('Heures d\'appel')
plt.ylabel('Temps d\'arrivée en minutes')
plt.yticks([180,240,300,360,420,480],[3,4,5,6,7,8])
plt.title('Comparaison Temps réels / Temps prédits par le modèle RidgeCV')
plt.legend();
```

Graphique 15. *Visualisation Modèle LassoCV (1ère itération)*

```
# Création du modèle de régression LassoCV
model_lasso = LassoCV().fit(X_train, y_train)

# Prédictions à partir de X_test
pred_test = model_lasso.predict(X_test)

# Affichage des performances du modèle score R carré + mean_squared_error
print("score test:", model_lasso.score(X_test, y_test))
print("mse test:", mean_squared_error(pred_test, y_test))


## score test: 0.5762807159318282
## mse test: 0.4276558080220872

# Reconstitution de l'échantillon de test avec la colonne cible dans le but de dénormaliser pour visualiser les écarts entrainement / test
num_X1 = X_test[['CalYear','HourOfCall','Easting_rounded','Northing_rounded','casernes_quartier','casernes_sous_quartier','CalWeekDay','CalDay','CalMonth']]
num_X3 = pd.DataFrame(y_test)[['FirstPumpArriving_AttendanceTime']]

# num_X_test avec la cible réelle FirstPumpArriving_AttendanceTime
num_X_test = num_X1.join(num_X3)

# Reconstitution de l'échantillon de test avec la colonne prédite dans le but de dénormaliser pour visualiser les écarts entrainement / test
num_X1 = X_test[['CalYear','HourOfCall','Easting_rounded','Northing_rounded','casernes_quartier','casernes_sous_quartier','CalWeekDay','CalDay','CalMonth']]
num_X4 = pd.DataFrame(pred_test, columns=['FirstPumpArriving_AttendTime_pred'])

# num_X_pred avec la cible prédite FirstPumpArriving_AttendTime_pred
num_X_pred = num_X1.join(num_X4)

# on dénormalise les deux dataframes
num_X_test_inverse = pd.DataFrame(scaler.inverse_transform(num_X_test), columns=num_X_test.columns, index= num_X_test.index)
num_X_pred_inverse = pd.DataFrame(scaler.inverse_transform(num_X_pred), columns=num_X_pred.columns, index= num_X_pred.index)

# on rajoute la colonne cible prédite à num_X_test_inverse
num_X_test_inverse = num_X_test_inverse.join(num_X_pred_inverse.FirstPumpArriving_AttendTime_pred)

# display(num_X_test_inverse[(-pd.isnull(num_X_test_inverse.FirstPumpArriving_AttendTime_pred))&(num_X_test_inverse.CalYear == 2020)&(num_X_test_inverse.CalMonth == 11)&(num_X_test_inverse.CalDay ==14)][['CalDay','CalMonth','HourOfCall', 'FirstPumpArriving_AttendanceTime','FirstPumpArriving_AttendTime_pred' ]].sort_values(['CalDay','HourOfCall']).head(30))
res = num_X_test_inverse[(-pd.isnull(num_X_test_inverse.FirstPumpArriving_AttendTime_pred))&(num_X_test_inverse.CalYear == 2020)&(num_X_test_inverse.CalMonth == 11)&(num_X_test_inverse.CalDay ==14)][['CalDay','CalMonth','HourOfCall', 'FirstPumpArriving_AttendanceTime','FirstPumpArriving_AttendTime_pred' ]].sort_values(['CalDay','HourOfCall']).head(30)

# on affiche les écarts dans un graphique pour cette journée
plt.figure(figsize=(15,8))
plt.scatter(res.HourOfCall, res.FirstPumpArriving_AttendTime_pred, c='r', label='Temps Prédit')
plt.scatter(res.HourOfCall, res.FirstPumpArriving_AttendanceTime, c='b', label='Temps Réel')
plt.xlabel('Heures d\'appel')
plt.ylabel('Temps d\'arrivée en minutes')
plt.yticks([180,240,300,360,420,480],[3,4,5,6,7,8])
plt.title('Comparaison Temps réels / Temps prédits par le modèle LassoCV')
plt.legend();
print(table)
```

Graphique 16. *Heatmap (2ème itération)*

```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import Ridge, LassoCV

# Chargement du dataset des Incidents
dfi = pd.read_excel('LFB Incident data from January 2017.xlsx')

# Chargement du dataset des Mobilisations
dfm = pd.read_excel('LFB Mobilisation data from January 2017.xlsx')

# Fusion des datasets sur la colonne commune IncidentNumber (on ne garde que les incidents en commun avec inner)
fusion = dfi.merge(right = dfm, on = 'IncidentNumber', how = 'inner')

# chargement du dataset de travail à partir de la fusion
# On ne retient que les lignes qui correspondent au premier camion (PumpOrder = 1) de la première équipe de pompiers arrivée sur les lieux
dfw = fusion[(fusion.FirstPumpArriving_DeployedFromStation == fusion.DeployedFromStation_Name)&(fusion.PumpOrder == 1)]

#On travaille sur l'année précédente pour calculer des statistiques
dfw = dfw[dfw.CalYear == 2019]

# on souhaite ajouter au dataset le nombre de casernes du quartier de l'incident (ex. dans le quartier de Westminster il y a 8 casernes)
# dans le but d'enrichir notre dataset afin d'aider les modèles de prédiction
# on ajoute également les temps moyens  et médians par casernes
df_cas_quart = pd.DataFrame(columns=('ProperCase', 'ProperCase_num' ,'casernes_quartier', 'tmoyq', 'tmedq'))

for indice, quartier in enumerate(dfw.ProperCase.unique()):
    #print(indice)
    #print(quartier)
    df_cas_quart.loc[indice,'ProperCase'] = quartier
    df_cas_quart.loc[indice,'ProperCase_num'] = indice
    nb_cas_pot = dfw[dfw.ProperCase == quartier].IncidentStationGround.nunique()
    #print("casernes potentielles ",nb_cas_pot)
    df_cas_quart.loc[indice,'casernes_quartier'] = nb_cas_pot
    tps_moy = dfw[dfw.ProperCase == quartier].FirstPumpArriving_AttendanceTime.mean()
    df_cas_quart.loc[indice,'tmoyq'] = tps_moy
    tps_med = dfw[dfw.ProperCase == quartier].FirstPumpArriving_AttendanceTime.median()
    df_cas_quart.loc[indice,'tmedq'] = tps_med
    #print("temps moyen ",tps_moy)

#display(df_cas_quart)
# on sauvegarde ces statistiques dans un csv
df_cas_quart.to_csv('df_cas_quart_2019.csv')

# on ajoute une colonne jour de la semaine
dfw['CalWeekDay'] = pd.to_datetime(dfw['DateOfCall']).dt.weekday

# on souhaite ajouter au dataset le nombre de casernes du sous quartier de l'incident
# dans le but d'enrichir notre dataset afin d'aider les modèles de prédiction
# on ajoute également les temps moyens, médians, écart type, min et max par sous quartier, type d'incident et jour de la semaine
df_sqigwd = pd.DataFrame(columns=('IncGeo_WardName', 'IncGeo_WardName_num','casernes_sous_quartier','IncidentGroup','CalWeekDay','tmoysqigwd','tmedsqigwd','tstdsqigwd','tminsqi
gwd','tmaxsqigwd'))

num_sq = 0
indice = 0
```

```
for sous_quartier in dfw.IncGeo_WardName.unique():
    nb_cas_pot = dfw[dfw.IncGeo_WardName == sous_quartier].IncidentStationGround.nunique()
    #print(indice)
    for ig in dfw[dfw.IncGeo_WardName == sous_quartier].IncidentGroup.unique():
        for k, wd in enumerate(dfw[(dfw.IncGeo_WardName == sous_quartier)&(dfw.IncidentGroup == ig)].CalWeekDay.unique()):
            #print("casernes potentielles ",nb_cas_pot)
            df_sqigwd.loc[indice+k,'IncGeo_WardName'] = sous_quartier
            df_sqigwd.loc[indice+k,'IncGeo_WardName_num'] = num_sq
            df_sqigwd.loc[indice+k,'casernes_sous_quartier'] = nb_cas_pot
            df_sqigwd.loc[indice+k,'IncidentGroup'] = ig
            df_sqigwd.loc[indice+k,'CalWeekDay'] = wd
            tps_moy = dfw[(dfw.IncGeo_WardName == sous_quartier)&(dfw.IncidentGroup == ig)&(dfw.CalWeekDay == wd)].FirstPumpArriving_AttendanceTime.mean()
            df_sqigwd.loc[indice+k,'tmoysqigwd'] = tps_moy
            tps_med = dfw[(dfw.IncGeo_WardName == sous_quartier)&(dfw.IncidentGroup == ig)&(dfw.CalWeekDay == wd)].FirstPumpArriving_AttendanceTime.median()
            df_sqigwd.loc[indice+k,'tmedsqigwd'] = tps_med
            tps_std = dfw[(dfw.IncGeo_WardName == sous_quartier)&(dfw.IncidentGroup == ig)&(dfw.CalWeekDay == wd)].FirstPumpArriving_AttendanceTime.std()
            df_sqigwd.loc[indice+k,'tstdsqigwd'] = tps_std
            tps_min = dfw[(dfw.IncGeo_WardName == sous_quartier)&(dfw.IncidentGroup == ig)&(dfw.CalWeekDay == wd)].FirstPumpArriving_AttendanceTime.min()
            df_sqigwd.loc[indice+k,'tminsqigwd'] = tps_min
            tps_max = dfw[(dfw.IncGeo_WardName == sous_quartier)&(dfw.IncidentGroup == ig)&(dfw.CalWeekDay == wd)].FirstPumpArriving_AttendanceTime.max()
            df_sqigwd.loc[indice+k,'tmaxsqigwd'] = tps_max
            #tps_moy = dfw[dfw.ProperCase == quartier].FirstPumpArriving_AttendanceTime.mean()
            #print("temps moyen ",tps_moy)
        indice +=k+1
    num_sq += 1

#display(df_sqigwd.head(10))

# on remplace les écarts types null par 0
df_sqigwd.tstdsqigwd.fillna(0, inplace=True)
# on sauvegarde ces statistiques dans un csv
df_sqigwd.to_csv('df_sqigwd_2019.csv')

# on souhaite ajouter au dataset les temps moyens, médians, écart type, min et max par quartier, heure de l'incident et service spécial
# dans le but d'enrichir notre dataset afin d'aider les modèles de prédiction
df_quartiers = pd.DataFrame(columns=('ProperCase','SpecialServiceType', 'HourOfCall','tmoyqussho', 'tmedqussho', 'tstdqussho','tminqussho', 'tmaxqussho'))

ind = 0

dfw.SpecialServiceType.fillna('None', inplace=True)

for quartier in dfw.ProperCase.unique():
    #nb_cas_pot = dfw[dfw.ProperCase == quartier].IncidentStationGround.nunique()
    #print(quartier)
    for heure in range(0,24):
        for i, spst in enumerate(dfw[(dfw.ProperCase == quartier)&(dfw.HourOfCall == heure)].SpecialServiceType.unique()):
            df_quartiers.loc[ind+i,'ProperCase'] = quartier
            df_quartiers.loc[ind+i,'SpecialServiceType'] = spst
            df_quartiers.loc[ind+i,'HourOfCall'] = heure
            #print("casernes potentielles ",nb_cas_pot)
            #df_quartiers.loc[indice,'casernes_quartier'] = nb_cas_pot
            tps_moy = dfw[(dfw.ProperCase == quartier)&(dfw.SpecialServiceType == spst)&(dfw.HourOfCall == heure)].FirstPumpArriving_AttendanceTime.mean()
            df_quartiers.loc[ind+i,'tmoyqussho'] = tps_moy
            tps_med = dfw[(dfw.ProperCase == quartier)&(dfw.SpecialServiceType == spst)&(dfw.HourOfCall == heure)].FirstPumpArriving_AttendanceTime.median()
            df_quartiers.loc[ind+i,'tmedqussho'] = tps_med
            tps_std = dfw[(dfw.ProperCase == quartier)&(dfw.SpecialServiceType == spst)&(dfw.HourOfCall == heure)].FirstPumpArriving_AttendanceTime.std()
            df_quartiers.loc[ind+i,'tstdqussho'] = tps_std
            tps_min = dfw[(dfw.ProperCase == quartier)&(dfw.SpecialServiceType == spst)&(dfw.HourOfCall == heure)].FirstPumpArriving_AttendanceTime.min()
            df_quartiers.loc[ind+i,'tminqussho'] = tps_min
            tps_max = dfw[(dfw.ProperCase == quartier)&(dfw.SpecialServiceType == spst)&(dfw.HourOfCall == heure)].FirstPumpArriving_AttendanceTime.max()
            df_quartiers.loc[ind+i,'tmaxqussho'] = tps_max
            #print("temps moyen ",tps_moy)
        ind +=i+1

#display(df_quartiers)
# on remplace les écarts types null par 0
df_quartiers.tstdqussho.fillna(0, inplace=True)
# on sauvegarde ces statistiques dans un csv
df_quartiers.to_csv('df_quartiers_2019.csv')
```

```
# on souhaite ajouter au dataset les temps moyens, médians, écart type, min et max par sous quartier et delaycodeId
# dans le but d'enrichir notre dataset afin d'aider les modèles de prédiction
df_sqid = pd.DataFrame(columns=('IncGeo_WardName','DelayCodeId','tmoyCodeId', 'tmedCodeId', 'tstdCodeId','tminCodeId', 'tmaxCodeId'))
# on remplace les DelayCodeId null par 0
dfw.DelayCodeId.fillna(0, inplace=True)

indice = 0

for sous_quartier in dfw.IncGeo_WardName.unique():
    for n, codeid in enumerate(dfw[(dfw.IncGeo_WardName == sous_quartier)].DelayCodeId.unique()):
        #print("casernes potentielles ",nb_cas_pot)
        df_sqid.loc[indice+n,'IncGeo_WardName'] = sous_quartier
        df_sqid.loc[indice+n,'DelayCodeId'] = codeid
        tps_moy = dfw[(dfw.IncGeo_WardName == sous_quartier)&(dfw.DelayCodeId == codeid)].FirstPumpArriving_AttendanceTime.mean()
        df_sqid.loc[indice+n,'tmoyCodeId'] = tps_moy
        tps_med = dfw[(dfw.IncGeo_WardName == sous_quartier)&(dfw.DelayCodeId == codeid)].FirstPumpArriving_AttendanceTime.median()
        df_sqid.loc[indice+n,'tmedCodeId'] = tps_med
        tps_std = dfw[(dfw.IncGeo_WardName == sous_quartier)&(dfw.DelayCodeId == codeid)].FirstPumpArriving_AttendanceTime.std()
        df_sqid.loc[indice+n,'tstdCodeId'] = tps_std
        tps_min = dfw[(dfw.IncGeo_WardName == sous_quartier)&(dfw.DelayCodeId == codeid)].FirstPumpArriving_AttendanceTime.min()
        df_sqid.loc[indice+n,'tminCodeId'] = tps_min
        tps_max = dfw[(dfw.IncGeo_WardName == sous_quartier)&(dfw.DelayCodeId == codeid)].FirstPumpArriving_AttendanceTime.max()
        df_sqid.loc[indice+n,'tmaxCodeId'] = tps_max
        #tps_moy = dfw[dfw.ProperCase == quartier].FirstPumpArriving_AttendanceTime.mean()
        #print("temps moyen ",tps_moy)
    indice +=n+1

#display(df_sqid.head(10))
# on remplace les écarts types null par 0
df_sqid.tstdCodeId.fillna(0, inplace=True)
# on sauvegarde ces statistiques dans un csv
df_sqid.to_csv('df_sqid_2019.csv')

# on numérote les différents types d'incident dans un DataFrame
df_ig = pd.DataFrame(columns=('IncidentGroup', 'IncidentGroup_num' ))

for j, ig in enumerate(dfw.IncidentGroup.unique()):
    df_ig.loc[j,'IncidentGroup'] = ig
    df_ig.loc[j,'IncidentGroup_num'] = j

#display(df_ig)

# on numérote les différents services spéciaux dans un DataFrame
df_sst = pd.DataFrame(columns=('SpecialServiceType', 'SpecialServiceType_num' ))

for m, sst in enumerate(dfw.SpecialServiceType.unique()):
    df_sst.loc[m,'SpecialServiceType'] = sst
    df_sst.loc[m,'SpecialServiceType_num'] = m

#display(df_sst)

# on repart de la fusion d'origine
dfw = fusion[(fusion.FirstPumpArriving_DeployedFromStation == fusion.DeployedFromStation_Name)&(fusion.PumpOrder == 1)]

# cette fois ci on charge les incidents de 2020
dfw = dfw[dfw.CalYear == 2020]
#dfw.shape

# on ajoute la colonne casernes_quartier, temps moyens et médians au dataframe de travail par un merge sur la colonne commune ProperCase
dfw = dfw.merge(right = df_cas_quart, on = 'ProperCase' , how = 'inner')

#dfw.shape

# on ajoute le jour de la semaine
dfw['CalWeekDay'] = pd.to_datetime(dfw['DateOfCall']).dt.weekday

# on ajoute la colonne casernes_sous_quartier, temps moyens, médians, écart type, min, max au dataframe de travail par un merge sur les colonnes communes
# IncGeo_WardName, IncidentGroup et CalWeekDay
dfw = dfw.merge(right = df_sqigwd, on=["IncGeo_WardName", "IncidentGroup","CalWeekDay"] , how = 'inner')

#dfw.shape
```

```
# on remplace les services spéciaux nulls par 0
dfw.SpecialServiceType.fillna('None', inplace=True)
# on ajoute les colonnes temps moyens, médians, écart type, min, max au dataframe de travail par un merge sur les colonnes communes
# ProperCase, SpecialServiceType, HourOfCall
dfw = dfw.merge(right = df_quartiers, on=["ProperCase", "SpecialServiceType","HourOfCall"] , how = 'inner')

#dfw.shape

# on ajoute la colonne IncidentGroup_num au dataframe de travail par un merge sur la colonne commune IncidentGroup
dfw = dfw.merge(right = df_ig, on=["IncidentGroup"] , how = 'inner')

#dfw.shape

# on ajoute la colonne SpecialServiceType_num au dataframe de travail par un merge sur la colonne commune SpecialServiceType
dfw = dfw.merge(right = df_sst, on=["SpecialServiceType"] , how = 'inner')

#dfw.shape

# on remplace les DelayCodeId à null par 0
dfw.DelayCodeId.fillna(0, inplace=True)
# on ajoute les colonnes temps moyens, médians, écart type, min, max au dataframe de travail par un merge sur les colonnes communes
# IncGeo_WardName, DelayCodeId
dfw = dfw.merge(right = df_sqid, on=["IncGeo_WardName", "DelayCodeId"] , how = 'inner')

#dfw.shape

# on supprime les colonnes devenues inutiles
dfw = dfw.drop(['IncidentNumber','DateOfCall','TimeOfCall','IncidentGroup','StopCodeDescription','SpecialServiceType','PropertyCategory','PropertyType','AddressQualifier','Postcode_full',
        'Postcode_district','UPRN', 'USRN', 'IncGeo_BoroughCode','IncGeo_BoroughName','ProperCase','IncGeo_WardCode',
        'IncGeo_WardName','IncGeo_WardNameNew','Easting_m','Northing_m','Latitude','Longitude','FRS','IncidentStationGround','FirstPumpArriving_DeployedFromStation',
        'SecondPumpArriving_AttendanceTime','SecondPumpArriving_DeployedFromStation',
        'NumStationsWithPumpsAttending','NumPumpsAttending','PumpCount','PumpHoursRoundUp','Notional Cost (£)',
        'ResourceMobilisationId','Resource_Code','PerformanceReporting','DateAndTimeMobilised',
        'DateAndTimeMobile', 'TimeMobileTimezoneId', 'DateAndTimeArrived',
        'TimeArrivedTimezoneId', 'AttendanceTimeSeconds', 'DateAndTimeLeft',
        'TimeLeftTimezoneId', 'DateAndTimeReturned', 'TimeReturnedTimezoneId',
        'DeployedFromStation_Code', 'DeployedFromStation_Name','DeployedFromLocation','PumpOrder', 'PlusCode_Code','PlusCode_Description', 'DelayCode_Description']
       , axis=1)

# on transforme les nouvelles colonnes en numérique
dfw.casernes_quartier = dfw.casernes_quartier.astype('int64')
dfw.casernes_sous_quartier = dfw.casernes_sous_quartier.astype('int64')

# on transforme les nouvelles colonnes en numérique
dfw.HourOfCall = dfw.HourOfCall.astype('int64')
dfw.ProperCase_num = dfw.ProperCase_num.astype('int64')
dfw.CalWeekDay = dfw.CalWeekDay.astype('int64')
dfw.IncGeo_WardName_num = dfw.IncGeo_WardName_num.astype('int64')
dfw.IncidentGroup_num = dfw.IncidentGroup_num.astype('int64')
dfw.SpecialServiceType_num = dfw.SpecialServiceType_num.astype('int64')
# on transforme les nouvelles colonnes en numérique
dfw.tmoyq = dfw.tmoyq.astype('float64')
dfw.tmedq = dfw.tmedq.astype('float64')
dfw.tmoysqigwd = dfw.tmoysqigwd.astype('float64')
dfw.tmedsqigwd = dfw.tmedsqigwd.astype('float64')
dfw.tstdsqigwd = dfw.tstdsqigwd.astype('float64')
dfw.tminsqigwd = dfw.tminsqigwd.astype('float64')
dfw.tmaxsqigwd = dfw.tmaxsqigwd.astype('float64')
dfw.tmoyqussho = dfw.tmoyqussho.astype('float64')
dfw.tmedqussho = dfw.tmedqussho.astype('float64')
dfw.tstdqussho = dfw.tstdqussho.astype('float64')
dfw.tminqussho = dfw.tminqussho.astype('float64')
dfw.tmaxqussho = dfw.tmaxqussho.astype('float64')
dfw.tmoyCodeId = dfw.tmoyCodeId.astype('float64')
dfw.tmedCodeId = dfw.tmedCodeId.astype('float64')
dfw.tstdCodeId = dfw.tstdCodeId.astype('float64')
dfw.tminCodeId = dfw.tminCodeId.astype('float64')
dfw.tmaxCodeId = dfw.tmaxCodeId.astype('float64')
dfw.DelayCodeId = dfw.DelayCodeId.astype('int64')
```

```
num_data = dfw.select_dtypes(include=['float64','int64'])
#print(len(num_data.columns),"colonnes numériques")
# pour chaque colonne numérique
for col in num_data:
    # on supprime la colonne initiale de dfw pour plus tard joindre toutes les colonnes numériques
    # centrées réduites de num_data à dfw
    dfw = dfw.drop(col, axis=1)

#num_data.columns

# reconstruction de num_data en mettant la target en dernière colonne
num_d1 = num_data[['CalYear','HourOfCall','Easting_rounded','Northing_rounded']]
num_d2 = num_data[['DelayCodeId', 'ProperCase_num',
       'casernes_quartier', 'tmoyq', 'tmedq', 'CalWeekDay',
       'IncGeo_WardName_num', 'casernes_sous_quartier', 'tmoysqigwd',
       'tmedsqigwd', 'tstdsqigwd', 'tminsqigwd', 'tmaxsqigwd', 'tmoyqussho',
       'tmedqussho', 'tstdqussho', 'tminqussho', 'tmaxqussho',
       'IncidentGroup_num', 'SpecialServiceType_num', 'tmoyCodeId',
       'tmedCodeId', 'tstdCodeId', 'tminCodeId', 'tmaxCodeId']]
num_d3 = num_data[['FirstPumpArriving_AttendanceTime']]

num_data = num_d1.join(num_d2.join(num_d3))

#num_data.head()

#num_data.shape


# Matrice de corrélation
cor = num_data.corr()

fig, ax = plt.subplots(figsize=(14,14))
sns.heatmap(cor, annot= True, ax= ax, cmap="coolwarm");
```

Graphique 17. *Visualisation Modèle LinearRegression (2ème itération)*

```
# Normalisation de num_data
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import Ridge, LassoCV

scaler = preprocessing.StandardScaler().fit(num_data)
num_data[num_data.columns] = pd.DataFrame(scaler.transform(num_data), columns=num_data.columns, index= num_data.index)

# Récupération de num_data dans dfw
dfw = dfw.join(num_data)
#dfw.head(10)

# découpage target et features
target = dfw.FirstPumpArriving_AttendanceTime
feats = dfw.drop(['FirstPumpArriving_AttendanceTime'], axis=1)

# on fabrique des échantillons d'entrainement et de test
X_train, X_test, y_train, y_test = train_test_split(feats, target, test_size=0.2)

# Création du modèle LinearRegression
from sklearn.linear_model import LinearRegression

lr = LinearRegression()
# entrainement du modèle
lr.fit(X_train, y_train)

# R**2  + proche de 1 possible
print("Coefficient de détermination du modèle :", lr.score(X_train, y_train))

# affichage du score
lr.score(X_test,y_test)

# on stocke les prédictions du modèle pour X_test dans pred_test
pred_test = lr.predict(X_test)

# reconstitution des dataframes normalisés X_test + y_test et X_test + y_pred avec la target en dernière colonnes
num_X1 = X_test[['CalYear','HourOfCall','Easting_rounded','Northing_rounded']]
num_X2 = X_test[['DelayCodeId', 'ProperCase_num',
      'casernes_quartier', 'tmoyq', 'tmedq', 'CalWeekDay',
      'IncGeo_WardName_num', 'casernes_sous_quartier', 'tmoysqigwd',
      'tmedsqigwd', 'tstdsqigwd', 'tminsqigwd', 'tmaxsqigwd', 'tmoyqussho',
      'tmedqussho', 'tstdqussho', 'tminqussho', 'tmaxqussho',
      'IncidentGroup_num', 'SpecialServiceType_num', 'tmoyCodeId',
      'tmedCodeId', 'tstdCodeId', 'tminCodeId', 'tmaxCodeId']]
num_X1 = num_X1.join(num_X2)
num_X3 = pd.DataFrame(y_test)[['FirstPumpArriving_AttendanceTime']]

num_X_test = num_X1.join(num_X3)

num_X_pred = num_X1.assign(FirstPumpArriving_AttendTime_pred=pd.DataFrame(pred_test, columns=['FirstPumpArriving_AttendTime_pred']).values)

# dénormalisation de num_X_test et num_X_pred pour récupérer les temps réels et prédits
num_X_test_inverse = pd.DataFrame(scaler.inverse_transform(num_X_test), columns=num_X_test.columns, index= num_X_test.index)
num_X_pred_inverse = pd.DataFrame(scaler.inverse_transform(num_X_pred), columns=num_X_pred.columns, index= num_X_pred.index)

# concaténation du temps prédit à num_x_test_inverse pour avoir les 2 colonnes dans le même dataframe
num_X_test_inverse = num_X_test_inverse.join(num_X_pred_inverse.FirstPumpArriving_AttendTime_pred)

# affichage de quelques lignes pour les mettre dans un graphique
display(num_X_test_inverse[(num_X_test_inverse.ProperCase_num == 11)&(num_X_test_inverse.IncidentGroup_num == 2)&(num_X_test_inverse.CalWeekDay == 0)].sort_values(['CalWeekDay','Hour
OfCall']).head(20))

# récupératiob des mêmes données dans res
res = num_X_test_inverse[(num_X_test_inverse.ProperCase_num == 11)&(num_X_test_inverse.IncidentGroup_num == 2)&(num_X_test_inverse.CalWeekDay == 0)][['CalWeekDay','HourOfCall', 'FirstPu
mpArriving_AttendanceTime','FirstPumpArriving_AttendTime_pred' ]].sort_values(['CalWeekDay','HourOfCall']).head(30)

# Affichage des comparaisons Temps réels / Temps prédits par le modèle
plt.figure(figsize=(15,8))
plt.scatter(res.HourOfCall, res.FirstPumpArriving_AttendTime_pred, c='r', label='Temps Prédit')
plt.scatter(res.HourOfCall, res.FirstPumpArriving_AttendanceTime, c='b', label='Temps Réel')
plt.xlabel('Heures d\'appel')
plt.ylabel('Temps d\'arrivée en minutes')
plt.yticks([180,240,300,360,420,480],[3,4,5,6,7,8])
plt.title('Comparaison Temps réels / Temps prédits par le modèle LinearRegression')
plt.legend();
```

Graphique 18. *Visualisation Modèle SGDRegressor (2ème itération)*

```
# Création du modèle SGDRegressor
from sklearn.linear_model import SGDRegressor

sgd_reg = SGDRegressor()
# entrainement du modèle
sgd_reg.fit(X_train, y_train)


#print( "alpha sélectionné par c-v :" ,ridge_reg.alpha_)
print("score train :", sgd_reg.score(X_train, y_train))
print("score test :", sgd_reg.score(X_test, y_test))

sgd_reg_train = sgd_reg.predict(X_train)
sgd_reg_test = sgd_reg.predict(X_test)

print("mse train:", mean_squared_error(sgd_reg_train, y_train))
print("mse test:", mean_squared_error(sgd_reg_test, y_test))

# on stocke les prédictions du modèle pour X_test dans pred_test
pred_test = sgd_reg.predict(X_test)

# reconstitution des dataframes normalisés X_test + y_test et X_test + y_pred avec la target en dernière colonnes
num_X1 = X_test[['CalYear','HourOfCall','Easting_rounded','Northing_rounded']]
num_X2 = X_test[['DelayCodeId', 'ProperCase_num',
      'casernes_quartier', 'tmoyq', 'tmedq', 'CalWeekDay',
      'IncGeo_WardName_num', 'casernes_sous_quartier', 'tmoysqigwd',
      'tmedsqigwd', 'tstdsqigwd', 'tminsqigwd', 'tmaxsqigwd', 'tmoyqussho',
      'tmedqussho', 'tstdqussho', 'tminqussho', 'tmaxqussho',
      'IncidentGroup_num', 'SpecialServiceType_num', 'tmoyCodeId',
      'tmedCodeId', 'tstdCodeId', 'tminCodeId', 'tmaxCodeId']]
num_X1 = num_X1.join(num_X2)
num_X3 = pd.DataFrame(y_test)[['FirstPumpArriving_AttendanceTime']]

num_X_test = num_X1.join(num_X3)

num_X_pred = num_X1.assign(FirstPumpArriving_AttendTime_pred=pd.DataFrame(pred_test, columns=['FirstPumpArriving_AttendTime_pred']).values)

# dénormalisation de num_X_test et num_X_pred pour récupérer les temps réels et prédits
num_X_test_inverse = pd.DataFrame(scaler.inverse_transform(num_X_test), columns=num_X_test.columns, index= num_X_test.index)
num_X_pred_inverse = pd.DataFrame(scaler.inverse_transform(num_X_pred), columns=num_X_pred.columns, index= num_X_pred.index)

# concaténation du temps prédit à num_x_test_inverse pour avoir les 2 colonnes dans le même dataframe
num_X_test_inverse = num_X_test_inverse.join(num_X_pred_inverse.FirstPumpArriving_AttendTime_pred)


# affichage de quelques lignes pour les mettre dans un graphique
display(num_X_test_inverse[(num_X_test_inverse.ProperCase_num == 11)&(num_X_test_inverse.IncidentGroup_num == 2)&(num_X_test_inverse.CalWeekDay == 0)].sort_values(['CalWee
kDay','HourOfCall']).head(20))
# récupératiob des mêmes données dans res
res = num_X_test_inverse[(num_X_test_inverse.ProperCase_num == 11)&(num_X_test_inverse.IncidentGroup_num == 2)&(num_X_test_inverse.CalWeekDay == 0)][['CalWeekDay','HourOf
Call', 'FirstPumpArriving_AttendanceTime','FirstPumpArriving_AttendTime_pred' ]].sort_values(['CalWeekDay','HourOfCall']).head(30)


# Affichage des comparaisons Temps réels / Temps prédits par le modèle
plt.figure(figsize=(15,8))
plt.scatter(res.HourOfCall, res.FirstPumpArriving_AttendTime_pred, c='r', label='Temps Prédit')
plt.scatter(res.HourOfCall, res.FirstPumpArriving_AttendanceTime, c='b', label='Temps Réel')
plt.xlabel('Heures d\'appel')
plt.ylabel('Temps d\'arrivée en minutes')
plt.yticks([180,240,300,360,420,480],[3,4,5,6,7,8])
plt.title('Comparaison Temps réels / Temps prédits par le modèle SGDRegressor')
plt.legend();
```

Graphique 19. *Visualisation Modèle GradientBoostingRegressor (2ème itération)*

```
# Création du modèle GradientBoostingRegressor
from sklearn.ensemble import GradientBoostingRegressor

gb_reg = GradientBoostingRegressor()
# entrainement du modèle
gb_reg.fit(X_train, y_train)

#print( "alpha sélectionné par c-v :" ,ridge_reg.alpha_)
print("score train :", gb_reg.score(X_train, y_train))
print("score test :", gb_reg.score(X_test, y_test))

##score train : 0.6156006822570951
##score test : 0.6110778763441289

gb_reg_train = gb_reg.predict(X_train)
gb_reg_test = gb_reg.predict(X_test)

print("mse train:", mean_squared_error(gb_reg_train, y_train))
print("mse test:", mean_squared_error(gb_reg_test, y_test))

##mse train: 0.3868851870979195
##mse test: 0.3788480879452526

# on stocke les prédictions du modèle pour X_test dans pred_test
pred_test = gb_reg.predict(X_test)

# reconstitution des dataframes normalisés X_test + y_test et X_test + y_pred avec la target en dernière colonnes
num_X1 = X_test[['CalYear','HourOfCall','Easting_rounded','Northing_rounded']]
num_X2 = X_test[['DelayCodeId', 'ProperCase_num',
      'casernes_quartier', 'tmoyq', 'tmedq', 'CalWeekDay',
      'IncGeo_WardName_num', 'casernes_sous_quartier', 'tmoysqigwd',
      'tmedsqigwd', 'tstdsqigwd', 'tminsqigwd', 'tmaxsqigwd', 'tmoyqussho',
      'tmedqussho', 'tstdqussho', 'tminqussho', 'tmaxqussho',
      'IncidentGroup_num', 'SpecialServiceType_num', 'tmoyCodeId',
      'tmedCodeId', 'tstdCodeId', 'tminCodeId', 'tmaxCodeId']]
num_X1 = num_X1.join(num_X2)
num_X3 = pd.DataFrame(y_test)[['FirstPumpArriving_AttendanceTime']]

num_X_test = num_X1.join(num_X3)

num_X_pred = num_X1.assign(FirstPumpArriving_AttendTime_pred=pd.DataFrame(pred_test, columns=['FirstPumpArriving_AttendTime_pred']).values)

num_X_pred.shape

# dénormalisation de num_X_test et num_X_pred pour récupérer les temps réels et prédits
num_X_test_inverse = pd.DataFrame(scaler.inverse_transform(num_X_test), columns=num_X_test.columns, index= num_X_test.index)
num_X_pred_inverse = pd.DataFrame(scaler.inverse_transform(num_X_pred), columns=num_X_pred.columns, index= num_X_pred.index)

# concaténation du temps prédit à num_x_test_inverse pour avoir les 2 colonnes dans le même dataframe
num_X_test_inverse = num_X_test_inverse.join(num_X_pred_inverse.FirstPumpArriving_AttendTime_pred)

# affichage de quelques lignes pour les mettre dans un graphique
display(num_X_test_inverse[(num_X_test_inverse.ProperCase_num == 11)&(num_X_test_inverse.IncidentGroup_num == 2)&(num_X_test_inverse.CalWeekDay == 0)].sort_values(['CalWeekDay','HourOfCall']).head(20))
# récupératiob des mêmes données dans res
res = num_X_test_inverse[(num_X_test_inverse.ProperCase_num == 11)&(num_X_test_inverse.IncidentGroup_num == 2)&(num_X_test_inverse.CalWeekDay == 0)][['CalWeekDay','HourOfCall', 'FirstPumpArriving_AttendanceTime','FirstPumpArriving_AttendTime_pred' ]].sort_values(['CalWeekDay','HourOfCall']).head(30)

# Affichage des comparaisons Temps réels / Temps prédits par le modèle
plt.figure(figsize=(15,8))
plt.scatter(res.HourOfCall, res.FirstPumpArriving_AttendTime_pred, c='r', label='Temps Prédit')
plt.scatter(res.HourOfCall, res.FirstPumpArriving_AttendanceTime, c='b', label='Temps Réel')
plt.xlabel('Heures d\'appel')
plt.ylabel('Temps d\'arrivée en minutes')
plt.yticks([180,240,300,360,420,480],[3,4,5,6,7,8])
plt.title('Comparaison Temps réels / Temps prédits par le modèle GradientBoostingRegressor')
plt.legend();
```

Table 2. Étude de corrélation entre le temps d'arrivée des pompiers et la caserne du lieu de l'incident (ANOVA)

```
import statsmodels.api

# Variable continue testée : FirstPumpArriving_AttendanceTime
# Variable catégorielle testée : IncidentStationGround
result = statsmodels.formula.api.ols('FirstPumpArriving_AttendanceTime ~ IncidentStationGround', data = dfw).fit()
table = statsmodels.api.stats.anova_lm(result)
```

Graphique 20. *Répartition de l'effort du projet sur la durée et dans l'équipe*

```
import plotly.express as px
import pandas as pd

# TOTAL EQUIPE
dfequipe = pd.DataFrame([
    dict(Task="Equipe", Start='2021-02-23', Finish='2021-02-24', Resource="Zoom Calls"),
    dict(Task="Equipe", Start='2021-02-25', Finish='2021-02-26', Resource="Zoom Calls"),
    dict(Task="Equipe", Start='2021-03-01', Finish='2021-03-02', Resource="Zoom Calls"),
    dict(Task="Equipe", Start='2021-03-04', Finish='2021-03-05', Resource="Zoom Calls"),
    dict(Task="Equipe", Start='2021-03-12', Finish='2021-03-16', Resource="Zoom Calls"),
    dict(Task="Equipe", Start='2021-03-19', Finish='2021-03-20', Resource="Zoom Calls"),
    dict(Task="Equipe", Start='2021-03-22', Finish='2021-03-23', Resource="Zoom Calls"),
    dict(Task="Equipe", Start='2021-03-24', Finish='2021-03-25', Resource="Zoom Calls"),
    dict(Task="Equipe", Start='2021-03-27', Finish='2021-03-28', Resource="Zoom Calls"),
    dict(Task="Equipe", Start='2021-03-29', Finish='2021-03-30', Resource="Zoom Calls"),
    dict(Task="Equipe", Start='2021-03-31', Finish='2021-04-01', Resource="Zoom Calls"),
    dict(Task="Equipe", Start='2021-02-22', Finish='2021-03-09', Resource="Traitement des données"),
    dict(Task="Equipe", Start='2021-02-28', Finish='2021-03-16', Resource="DataViz"),
    dict(Task="Equipe", Start='2021-03-25', Finish='2021-03-27', Resource="DataViz"),
    dict(Task="Equipe", Start='2021-03-08', Finish='2021-03-16', Resource="Itération 1"),
    dict(Task="Equipe", Start='2021-03-16', Finish='2021-03-25', Resource="Itération 2"),
    dict(Task="Equipe", Start='2021-03-04', Finish='2021-04-01', Resource="Structuration du rapport"),
    dict(Task="Equipe", Start='2021-03-27', Finish='2021-04-07', Resource="Streamlit")])

fig = px.timeline(dfequipe, x_start="Start", x_end="Finish", y="Resource", color="Resource", width=850, height=350)
fig.update_layout(showlegend=False, title = "Temps total equipe", title_x=0.56)
fig.update_yaxes(title="", tickfont_family="Arial Black")
fig.show()
print(table)
```

```
import plotly.express as px
import pandas as pd

# ARNAUD
dfarnaud = pd.DataFrame([
   dict(Task="Arnaud", Start='2021-02-23', Finish='2021-02-24', Resource="Zoom Calls"),
   dict(Task="Arnaud", Start='2021-02-25', Finish='2021-02-26', Resource="Zoom Calls"),
   dict(Task="Arnaud", Start='2021-03-01', Finish='2021-03-02', Resource="Zoom Calls"),
   dict(Task="Arnaud", Start='2021-03-04', Finish='2021-03-05', Resource="Zoom Calls"),
   dict(Task="Arnaud", Start='2021-03-12', Finish='2021-03-16', Resource="Zoom Calls"),
   dict(Task="Arnaud", Start='2021-03-19', Finish='2021-03-20', Resource="Zoom Calls"),
   dict(Task="Arnaud", Start='2021-03-22', Finish='2021-03-23', Resource="Zoom Calls"),
   dict(Task="Arnaud", Start='2021-03-24', Finish='2021-03-25', Resource="Zoom Calls"),
   dict(Task="Arnaud", Start='2021-03-27', Finish='2021-03-28', Resource="Zoom Calls"),
   dict(Task="Arnaud", Start='2021-03-29', Finish='2021-03-30', Resource="Zoom Calls"),
   dict(Task="Arnaud", Start='2021-03-31', Finish='2021-04-01', Resource="Zoom Calls"),
   dict(Task="Arnaud", Start='2021-02-22', Finish='2021-03-07', Resource="Traitement des données"),
   dict(Task="Arnaud", Start='2021-03-02', Finish='2021-03-02', Resource="DataViz"),
   dict(Task="Arnaud", Start='2021-03-08', Finish='2021-03-16', Resource="Itération 1"),
   dict(Task="Arnaud", Start='2021-03-16', Finish='2021-03-25', Resource="Itération 2"),
   dict(Task="Arnaud", Start='2021-02-22', Finish='2021-02-22', Resource="Structuration du rapport"),
   dict(Task="Arnaud", Start='2021-03-27', Finish='2021-04-07', Resource="Streamlit")])

fig = px.timeline(dfequipe, x_start="Start", x_end="Finish", y="Resource", color="Resource", width=850, height=350)
fig.update_layout(showlegend=False, title = "Temps total equipe", title_x=0.56)
fig.update_yaxes(title="", tickfont_family="Arial Black")
fig.show()
print(table)


# CEDRIC
dfcedric = pd.DataFrame([
   dict(Task="Cédric", Start='2021-02-23', Finish='2021-02-24', Resource="Zoom Calls"),
   dict(Task="Cédric", Start='2021-02-25', Finish='2021-02-26', Resource="Zoom Calls"),
   dict(Task="Cédric", Start='2021-03-01', Finish='2021-03-02', Resource="Zoom Calls"),
   dict(Task="Cédric", Start='2021-03-04', Finish='2021-03-05', Resource="Zoom Calls"),
   dict(Task="Cédric", Start='2021-03-12', Finish='2021-03-16', Resource="Zoom Calls"),
   dict(Task="Cédric", Start='2021-03-19', Finish='2021-03-20', Resource="Zoom Calls"),
   dict(Task="Cédric", Start='2021-03-22', Finish='2021-03-23', Resource="Zoom Calls"),
   dict(Task="Cédric", Start='2021-03-24', Finish='2021-03-25', Resource="Zoom Calls"),
   dict(Task="Cédric", Start='2021-03-27', Finish='2021-03-28', Resource="Zoom Calls"),
   dict(Task="Cédric", Start='2021-03-29', Finish='2021-03-30', Resource="Zoom Calls"),
   dict(Task="Cédric", Start='2021-03-31', Finish='2021-04-01', Resource="Zoom Calls"),
   dict(Task="Cédric", Start='2021-02-22', Finish='2021-03-09', Resource="Traitement des données"),
   dict(Task="Cédric", Start='2021-02-28', Finish='2021-03-16', Resource="DataViz"),
   dict(Task="Cédric", Start='2021-03-25', Finish='2021-03-27', Resource="DataViz"),
   dict(Task="Cédric", Start='2021-03-12', Finish='2021-03-13', Resource="Itération 1"),
   dict(Task="Cédric", Start='2021-03-14', Finish='2021-03-15', Resource="Itération 1"),
   dict(Task="Cédric", Start='2021-03-19', Finish='2021-03-20', Resource="Itération 2"),
   dict(Task="Cédric", Start='2021-03-22', Finish='2021-03-23', Resource="Itération 2"),
   dict(Task="Cédric", Start='2021-03-14', Finish='2021-03-19', Resource="Structuration du rapport"),
   dict(Task="Cédric", Start='2021-03-24', Finish='2021-03-27', Resource="Structuration du rapport"),
   dict(Task="Cédric", Start='2021-03-31', Finish='2021-04-01', Resource="Structuration du rapport"),
   dict(Task="Cédric", Start='2021-03-30', Finish='2021-04-07', Resource="Streamlit")])

fig = px.timeline(dfequipe, x_start="Start", x_end="Finish", y="Resource", color="Resource", width=850, height=350)
fig.update_layout(showlegend=False, title = "Temps total equipe", title_x=0.56)
fig.update_yaxes(title="", tickfont_family="Arial Black")
fig.show()
print(table)
```

```
import plotly.express as px
import pandas as pd

# FRANCK
dffranck = pd.DataFrame([
    dict(Task="Franck", Start='2021-02-23', Finish='2021-02-24', Resource="Zoom Calls"),
    dict(Task="Franck", Start='2021-02-25', Finish='2021-02-26', Resource="Zoom Calls"),
    dict(Task="Franck", Start='2021-03-01', Finish='2021-03-02', Resource="Zoom Calls"),
    dict(Task="Franck", Start='2021-03-04', Finish='2021-03-05', Resource="Zoom Calls"),
    dict(Task="Franck", Start='2021-03-12', Finish='2021-03-16', Resource="Zoom Calls"),
    dict(Task="Franck", Start='2021-03-19', Finish='2021-03-20', Resource="Zoom Calls"),
    dict(Task="Franck", Start='2021-03-22', Finish='2021-03-23', Resource="Zoom Calls"),
    dict(Task="Franck", Start='2021-03-24', Finish='2021-03-25', Resource="Zoom Calls"),
    dict(Task="Franck", Start='2021-03-27', Finish='2021-03-28', Resource="Zoom Calls"),
    dict(Task="Franck", Start='2021-03-29', Finish='2021-03-30', Resource="Zoom Calls"),
    dict(Task="Franck", Start='2021-03-31', Finish='2021-04-01', Resource="Zoom Calls"),
    dict(Task="Franck", Start='2021-02-23', Finish='2021-02-26', Resource="Traitement des données"),
    dict(Task="Franck", Start='2021-03-02', Finish='2021-03-04', Resource="DataViz"),
    dict(Task="Franck", Start='2021-03-06', Finish='2021-03-08', Resource="DataViz"),
    dict(Task="Franck", Start='2021-03-10', Finish='2021-03-16', Resource="Itération 1"),
    dict(Task="Franck", Start='2021-03-17', Finish='2021-03-25', Resource="Itération 2"),
    dict(Task="Franck", Start='2021-02-22', Finish='2021-02-22', Resource="Structuration du rapport"),
    dict(Task="Franck", Start='2021-03-27', Finish='2021-04-07', Resource="Streamlit")])

fig = px.timeline(dfequipe, x_start="Start", x_end="Finish", y="Resource", color="Resource", width=850, height=350)
fig.update_layout(showlegend=False, title = "Temps total equipe", title_x=0.56)
fig.update_yaxes(title="", tickfont_family="Arial Black")
fig.show()
print(table)


# IVAN
dfivan = pd.DataFrame([
    dict(Task="Ivan", Start='2021-02-23', Finish='2021-02-24', Resource="Zoom Calls"),
    dict(Task="Ivan", Start='2021-02-25', Finish='2021-02-26', Resource="Zoom Calls"),
    dict(Task="Ivan", Start='2021-03-01', Finish='2021-03-02', Resource="Zoom Calls"),
    dict(Task="Ivan", Start='2021-03-04', Finish='2021-03-05', Resource="Zoom Calls"),
    dict(Task="Ivan", Start='2021-03-12', Finish='2021-03-16', Resource="Zoom Calls"),
    dict(Task="Ivan", Start='2021-03-19', Finish='2021-03-20', Resource="Zoom Calls"),
    dict(Task="Ivan", Start='2021-03-22', Finish='2021-03-23', Resource="Zoom Calls"),
    dict(Task="Ivan", Start='2021-03-24', Finish='2021-03-25', Resource="Zoom Calls"),
    dict(Task="Ivan", Start='2021-03-27', Finish='2021-03-28', Resource="Zoom Calls"),
    dict(Task="Ivan", Start='2021-03-29', Finish='2021-03-30', Resource="Zoom Calls"),
    dict(Task="Ivan", Start='2021-03-31', Finish='2021-04-01', Resource="Zoom Calls"),
    dict(Task="Ivan", Start='2021-02-22', Finish='2021-03-04', Resource="Traitement des données"),
    dict(Task="Ivan", Start='2021-03-08', Finish='2021-03-16', Resource="DataViz"),
    dict(Task="Ivan", Start='2021-03-12', Finish='2021-03-13', Resource="Itération 1"),
    dict(Task="Ivan", Start='2021-03-14', Finish='2021-03-15', Resource="Itération 1"),
    dict(Task="Ivan", Start='2021-03-19', Finish='2021-03-20', Resource="Itération 2"),
    dict(Task="Ivan", Start='2021-03-22', Finish='2021-03-23', Resource="Itération 2"),
    dict(Task="Ivan", Start='2021-03-04', Finish='2021-04-01', Resource="Structuration du rapport"),
    dict(Task="Ivan", Start='2021-03-30', Finish='2021-04-07', Resource="Streamlit")])

fig = px.timeline(dfequipe, x_start="Start", x_end="Finish", y="Resource", color="Resource", width=850, height=350)
fig.update_layout(showlegend=False, title = "Temps total equipe", title_x=0.56)
fig.update_yaxes(title="", tickfont_family="Arial Black")
fig.show()
print(table)
```