

PROJET POMPIERS

Lorine Beaumont
Weronika Nouyrigat
Vincent Ruelle

Table des Matières

1. Introduction p.2

- 1.1. Les Données
- 1.2. Visualisation des données
- 1.3. Pre-Processing et features engineering
- 1.4. Encoding

2. Premier Modèle (Régression) p.12

- 2.1. Préparation du dataset et sélection des features
- 2.2. Implémentation des modèles de Régression
- 2.3. Interpretation des résultats

3. Deuxième Modèle (Classification) p.16

- 3.1. Préparation du dataset et sélection des features
- 3.2. Implémentation du modèles de Classification
- 3.3. Interpretation des résultats

4. Conclusion p.22

1. Introduction

La Brigade des Pompiers de Londres, avec 5000 personnes, est le plus grand service d'incendie et de secours du pays, protégeant le Grand Londres. Suite aux incendies meurtriers passés, des investissements croissants ont été réalisés, le temps de réaction étant crucial. Les incendies, souvent débutant par une petite flamme contrôlable, se propagent avec le temps, aggravant les conditions de lutte. Le facteur essentiel est le temps de réponse des pompiers, crucial pour la survie et la minimisation des dégâts.

Ces dernières années, l'intégration de l'IA dans les services d'incendie s'avère prometteuse. L'IA permet la prévision des incendies et la prise de décisions en temps réel grâce à l'analyse des données. Les algorithmes identifient les risques, prédisent la propagation, aidant les pompiers à élaborer des stratégies proactives et à allouer les ressources de manière optimale. L'IA fournit également des données en temps réel améliorant la connaissance de la situation.

1.1. Les données

Nous avons deux types de dataset à notre disposition: les *Incidents Records*, contenant 39 colonnes avec des informations sur les incidents ou les pompiers de Londres sont intervenus et les *Mobilisation Records*, contenant 22 colonnes avec des informations sur le temps de réaction des engins des pompiers.

Nous avons effectué une première exploration des données avec le template fourni au lien suivant:

<https://docs.google.com/spreadsheets/d/1XELFoghztrGdhST8fi0HdE1Acme6IVFP3kB3VrEkml/edit#gid=0>

1.2. Visualisation des données/Premières analyses des données en images

Fig. 1. Nombre d'incidents par jour sur les années 2009-2017

La figure 1 montre les incidents et appels cumulés par jour de 2009 à 2017. On remarque un nombre d'interventions comparable chaque année, avec quelques pics ponctuels en 2009, 2013 et 2016, probablement dus à des événements majeurs. Une hausse d'incidents est également observable chaque été.

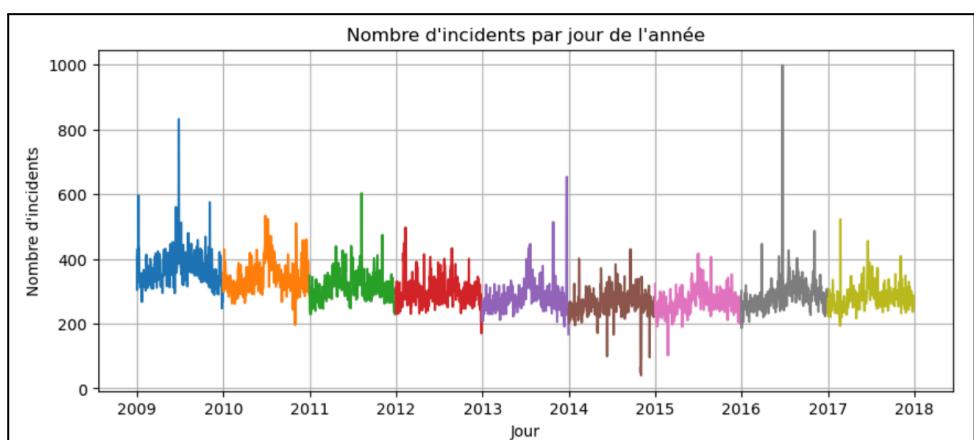


Fig. 2. Pourcentages d'incidents sur les années 2009-2017 selon les trois catégories d'appels (Fire, Special Service, False Alarm)

Les graphiques circulaires de la figure 2 montrent que la répartition en pourcentage par type d'appel est relativement constante. Près de la moitié des incidents sont des fausses alertes, un point important à souligner.

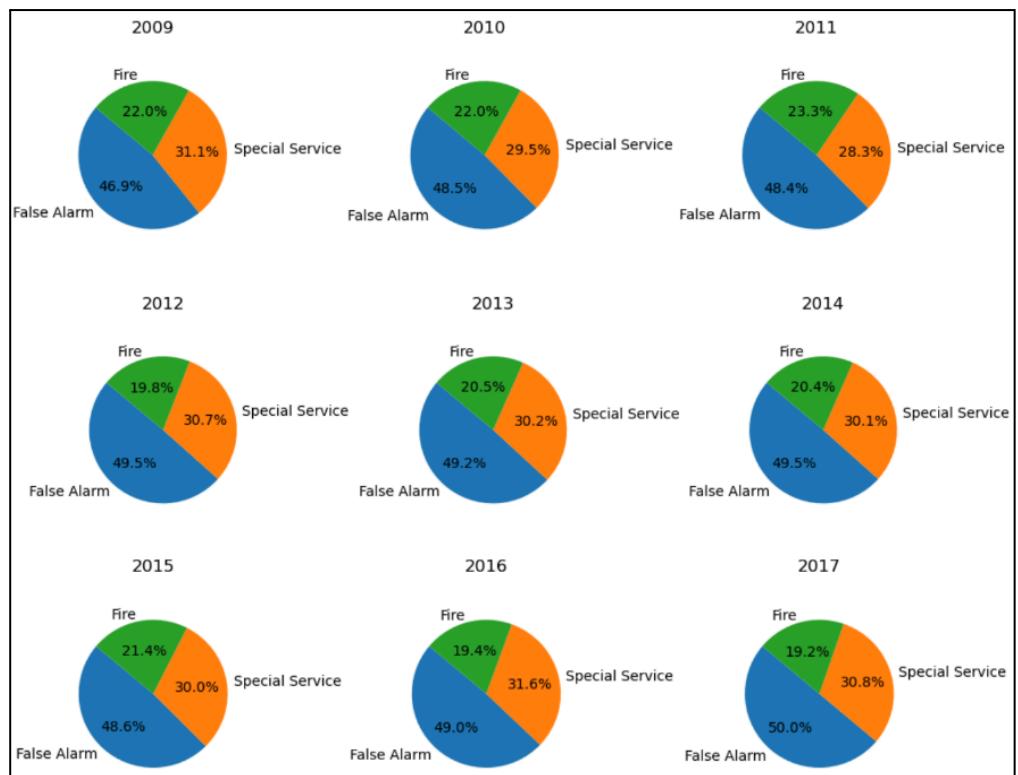


Fig. 3. Le temps moyen de réaction en fonction de l'heure d'appel

Le graphique montre le temps de réponse de la première brigade en fonction de l'heure d'appel. On constate des temps de réaction accrus en matinée (4h-7h) et en début d'après-midi (12h-17h).

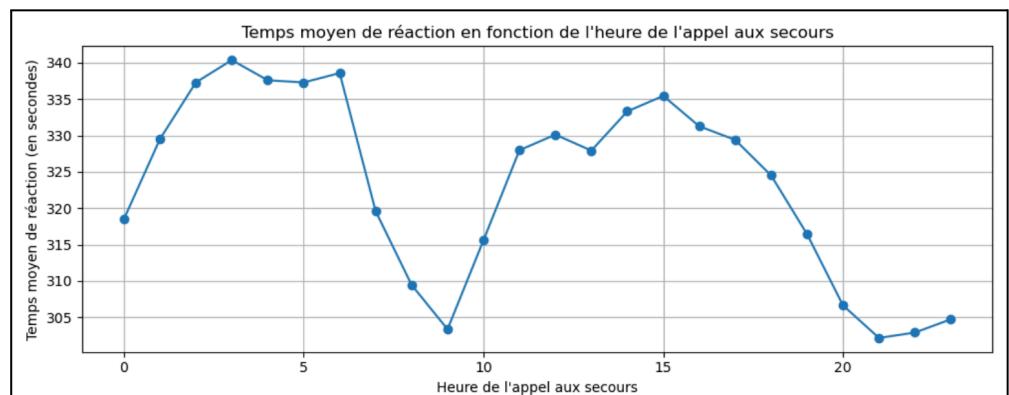


Fig. 4. Le nombre d'interventions en fonction de l'heure de l'appel aux secours

Ce graphique montre que la majorité des interventions ont lieu pendant la journée.

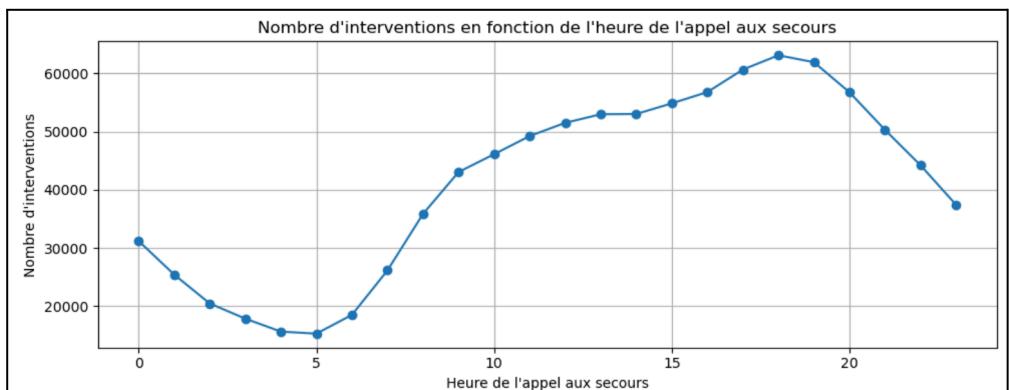


Fig. 5. Nombre d'interventions par arrondissement de Londres, avec la distinction entre les trois types d'incidents

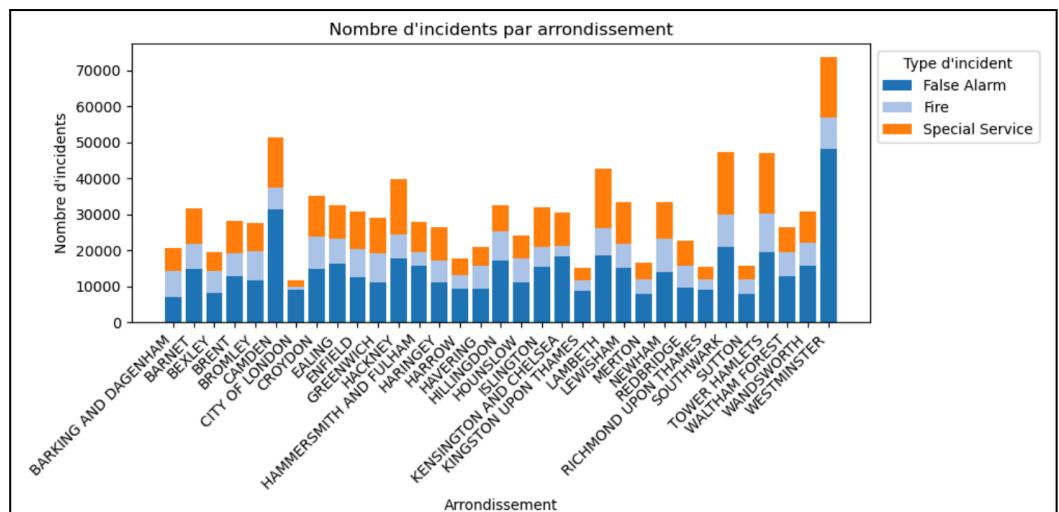


Fig. 6. Le nombre d'incidents par quartier de la ville de Londres avec une coloration des barres en fonction du prix au m².

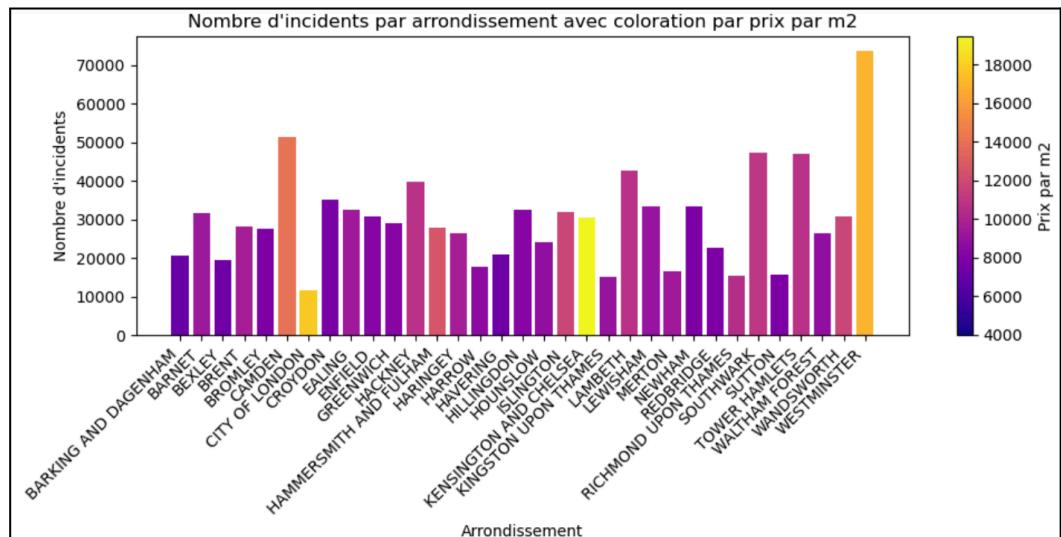


Fig. 7. Le nombre d'habitants par arrondissement de Londres

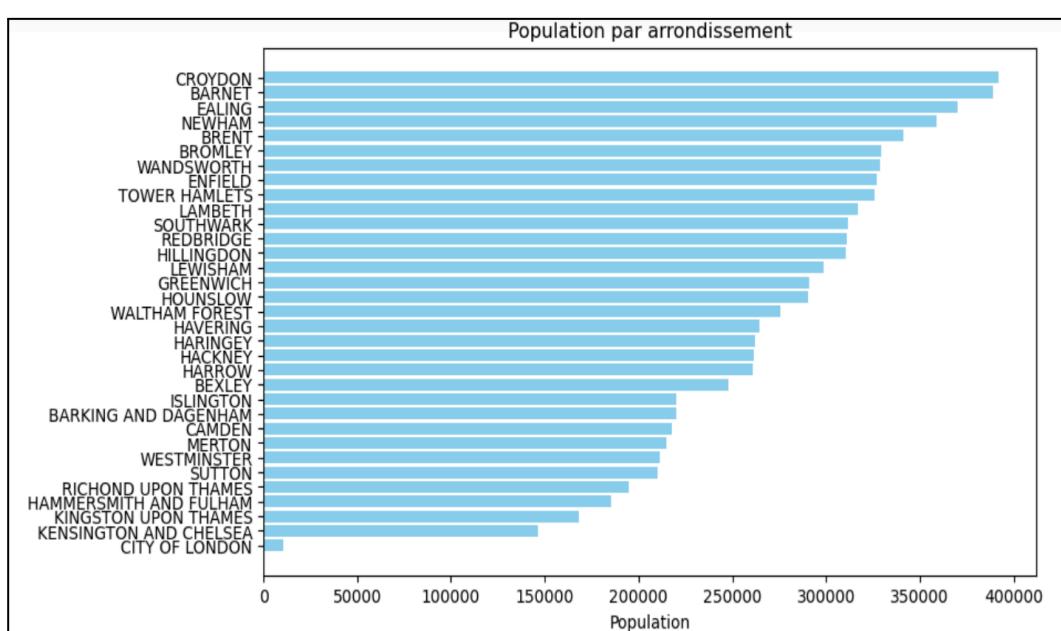


Fig. 8. Distribution des temps de réaction des 1er et 2eme camions arrivés

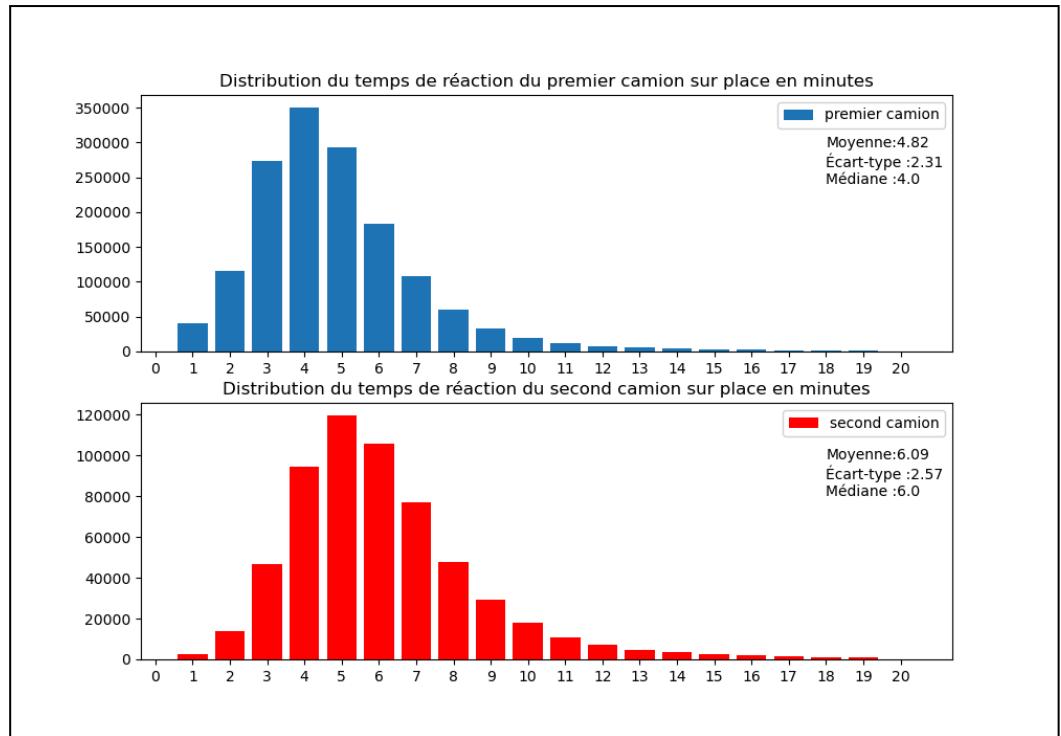


Fig. 9. Plan choroplète avec l'emplacement des casernes dans Londres

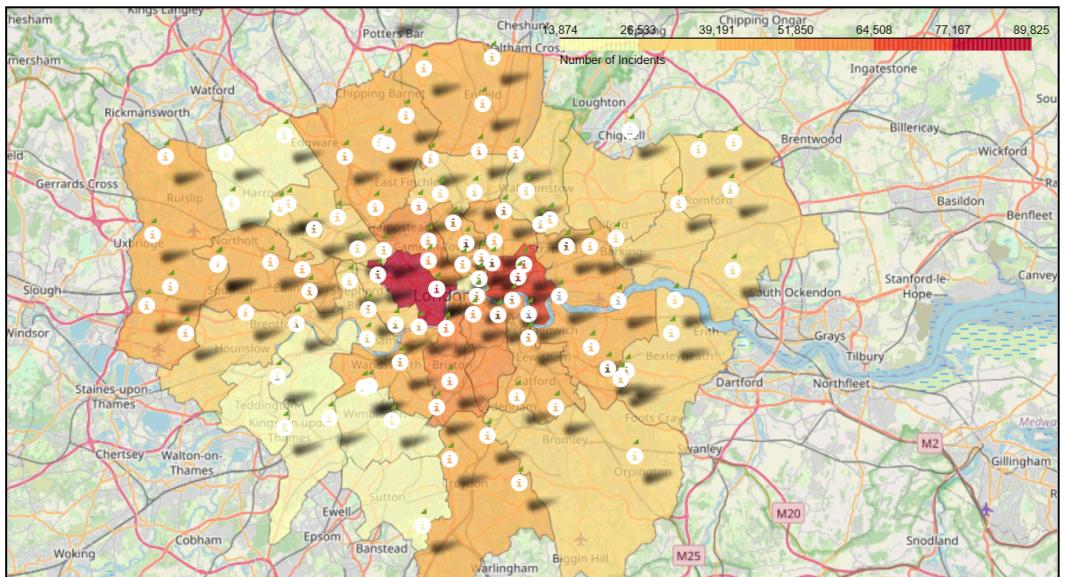
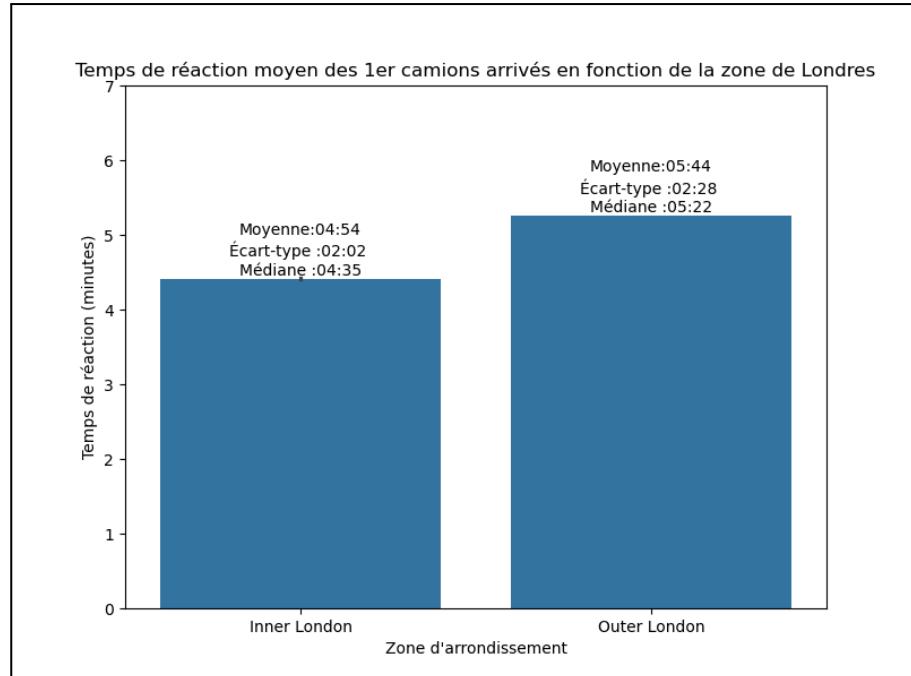


Fig. 10. Temps de réaction moyen des premiers camions en fonction de la zone Londres

Londres est composée de 33 arrondissements divisés en deux zones : Inner London (centre) et Outer London (périphérie). On remarque que les arrondissements dans le centre de Londres bénéficient d'un meilleur temps de réponse. Cela peut s'expliquer par la plus grande concentration de stations au centre de Londres



1.3. Pre-Processing et features engineering

1.3.1. Les données

Deux jeux de données sont utilisés : **Incidents** (détails des incidents) et **Mobilisation** (camions envoyés). Leur fusion sur le N° d'incident est complexe car une ligne dans Incidents correspond à un incident, tandis qu'une ligne dans Mobilisation correspond à un camion.

Le DataFrame Mobilisation contient l'ordre d'arrivée des camions. Un nouveau DataFrame a été créé avec les variables d'intérêt pour les 1er et 2e camions arrivés, chaque ligne correspondant à un N° d'incident.

L'objectif est de prédire le temps de réponse de la 1ère équipe (premier camion), ce qui inclut le temps de mobilisation suite à l'appel et le trajet vers l'incident.

1.3.2. Ajout de variables

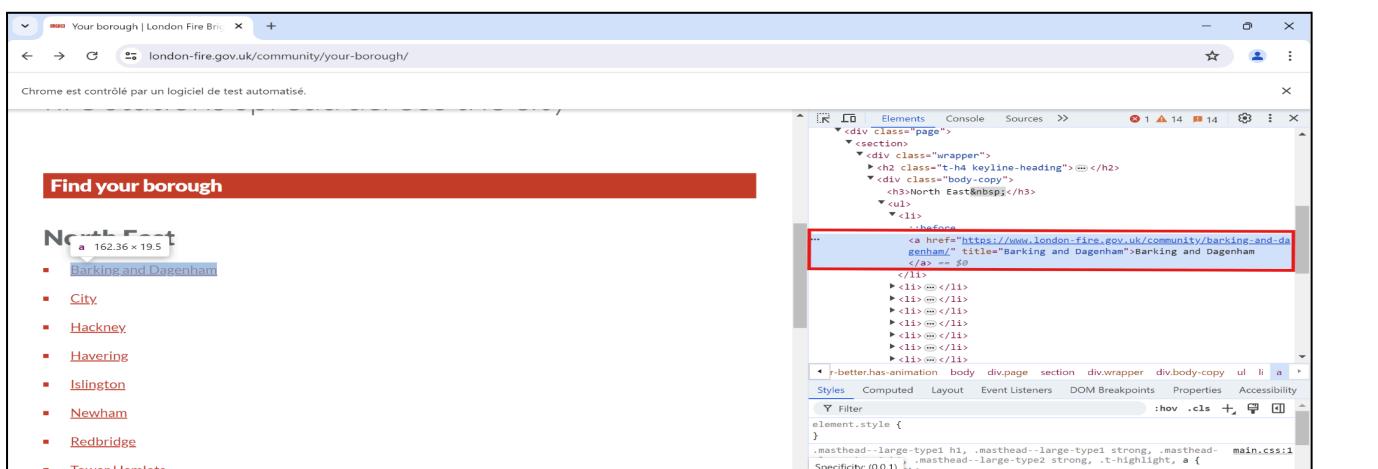
Les données ont été enrichies avec diverses variables pertinentes pour prédire le temps de réponse: La distinction entre Inner London et Outer London impacte les temps d'arrivée moyens (Figure 10). L'arrondissement de l'incident est donc un prédicteur intéressant.

Les données temporelles extraites sont le mois, le jour, la période de la journée (matin, après-midi, soir, nuit) et les heures de pointe (matin: 7h-10h, après-midi: 17h-19h). La circulation différant selon ces périodes, elles peuvent influencer le temps de réponse. Durant les confinements dûs au Covid-19 à Londres en 2020, le trafic a été fortement réduit, il a été pensé que le temps de réponse des pompiers pouvait être réduit. Une colonne précisant si l'incident a lieu pendant un confinement a donc été rajoutée.

Lorsqu'on évalue le temps de trajet, la distance parcourue est primordiale. Pour la calculer, les coordonnées GPS de la station intervenant et du lieu de l'incident sont

nécessaires. Celles des incidents sont déjà disponibles dans les données. Pour les casernes, les adresses ont d'abord dûes être récupérer par scraping sur le site de la London Fire Brigade (LFB), puis converties en coordonnées. Une première approche simpliste utilisait la distance à vol d'oiseau avec geopy. Ensuite, la distance routière a été calculée grâce à OSRM (Open Source Routing Machine), un outil open source exploitant les données OpenStreetMap pour déterminer les itinéraires les plus courts.

Scraping et récupération des coordonnées des casernes : Le scraping a été fait avec la librairie Selenium. La première étape a été d'ouvrir la page du site de la LFB qui recense les quartiers de Londres. Pour chaque quartier, il y a un lien cliquable, qu'on récupère. Ces liens renvoient aux pages contenant les informations des stations du quartier.



```

from selenium import webdriver
from selenium.webdriver.common.by import By
from webdriver_manager.chrome import ChromeDriverManager

service = webdriver.ChromeService()
driver = webdriver.Chrome(service=service)

# 1.Récupérer les liens pour chaque quartier

# ouvre la page des quartiers
driver.get("https://www.london-fire.gov.uk/community/your-borough/")
# Récupère les éléments webs de chaque quartiers. Contient les noms des quartiers et les liens vers les infos des quartiers
quartiers_html = driver.find_elements(By.XPATH, "/html/body/div[3]/section[1]/div/div/ul/li/a")
# Liste des noms des quartiers
quartier_1er_page = []
# Liste des liens vers les adresses de casernes
liens = []

# récupe les noms et des liens vers chaque quartier
for nom_quartier in quartiers_html :
    quartier_1er_page.append(nom_quartier.text)
    liens.append(nom_quartier.get_attribute('href'))

print(quartier[0:3])
print(liens[0:3])
✓ 0.0s

['Barking and Dagenham', 'Barking and Dagenham', 'The City of London']
['https://www.london-fire.gov.uk/community/barking-and-dagenham/', 'https://www.london-fire.gov.uk/community/the-city-of-london/', 'https://www.london-fire.gov.uk/community/hackney/']

```

Extrait de la page des quartiers avec l'élément récupéré (encadré rouge), code pour récupérer les liens et extrait des résultats

Une fois sur la page d'un quartier, on cherche l'élément de la page contenant l'adresse de chaque station. Si il y plusieurs stations dans un quartier, les adresses sont disponibles en cliquant sur le nom de chaque station. Il a donc fallu récupérer les liens de navigation afin d'accéder aux informations de chaque station.

Station addresses in Barking and Dagenham

BARKING | DAGENHAM

Barking

📍 Alfred's Way, IG11 0BB

[Get directions >](#)



Station addresses in Barking and Dagenham

BARKING | DAGENHAM

Dagenham

📍 70 Rainham Road North,
Dagenham, RM10 7ES

[Get directions >](#)



Barking and Dagenham

place for everyone.™

Borough Commander for Barking and Dagenham

Station addresses in Barking and Dagenham

a.tabs__nav-link.js-tab-link 123.84 × 57.8

BARKING | DAGENHAM

Dagenham

📍 70 Rainham Road North,
Dagenham, RM10 7ES



Skip to ▾

DevTools is now available in French!
Always match Chrome's language [Switch DevTools to French](#) [Don't show again](#)

Elements Console Sources 3 110 65

```

<section class="bg-white">
  <div class="wrapper">
    <h2 class="t-h4 keyline-heading">...</h2>
    <div class="tabs js-tabs js-tabs--closed-mobile no-page-gutter@s">
      &gt;
        <nav class="tabs__nav">
          <col class="tabs__nav-list bare-list">
            <li class="tabs__nav-item">
              <a href="#fbTabs_12tb" class="tabs__nav-link js-tab-link">Barking </span> == $0
                <span class="tabs__nav-text">Barking </span>
              </a>
              &gt;&gt;
            </li>
            <li class="tabs__nav-item">...</li>
          </ol>
        </nav>
        <div class="tabs__content bg-grey">...</div>
        <div class="tabs__content bg-grey">...</div>
      </div>
    </div>
  </div>
</section>

```

Styles Computed Layout Event Listeners DOM Breakpoints Properties Accessibility

Filter :hov .cls + ⌂

```
# 2.Dans chaque liens, scrapper les adresses stations

# quartier, nom de station et adresse
quartier = []
nom_station = []
adress = []

for lien in liens:
    # pour chaque quartier, redirection vers le lien contenant les infos des stations
    driver.get(lien)
    # Récupère l'élément web de chaque station présente dans un quartier
    stations = driver.find_elements(By.XPATH, "/html/body/div[1]/section[3]/div/div/nav/ol/li/a")
    for i,station in enumerate(stations) :
        # nom de quartier
        quartier.append(driver.find_element(By.XPATH, "/html/body/div[1]/div[1]/div[1]/div[1]/h1").text)
        # récupère les noms des stations
        nom_station.append(station.text)
        # clique sur l'onglet avec le nom de station
        station.click()
        # récup de l'adresse de la station
        adress.append(driver.find_element(By.XPATH, f"/html/body/div[1]/section[3]/div/div/div[{i+1}]/section/div/div/div[2]/address").text)
```

[Page contenant les informations des stations d'un quartier et code pour récupérer les adresses](#)

Les adresses ont ensuite été converties en coordonnées GPS grâce à l'API Bing™ Maps REST dont l'une des fonctionnalités permet de récupérer les coordonnées GPS. Les coordonnées sont récupérées en envoyant une requête avec un URL de la forme :

http://dev.virtualearth.net/REST/v1/Locations/UK/q='{adresse}'?&c=en-GB&userRegion=GB&key={cle_bing}.

L'argument UK permet de préciser que le pays des adresses est l'Angleterre. (Une tentative sans cet argument avait mené à l'obtention de coordonnées situées dans d'autre pays). La requête renvoie un dictionnaire au format JSON d'où on récupère les coordonnées.

```
import requests
from tqdm import tqdm

# Formate les adresses au bon format : remplacer les espaces par des %20
query = []
for adresse in df.Adresse :
    query.append(adresse.replace(" ","%20"))

latitude = []
longitude = []

for i,adresse in enumerate(tqdm(query)) :

    url = f"http://dev.virtualearth.net/REST/v1/Locations/UK/q='{query[i]}'?&c=en-GB&userRegion=GB&key={cle_bing}"
    reponse = requests.get(url)
    data = reponse.json()

    # si la requete n'aboutie pas, met des NA
    if reponse.status_code != 200:
        print("bad request index",i)
        latitude.append(np.nan)
        longitude.append(np.nan)

    # récupère les latitude et longitude
    else :
        latitude.append(data["resourceSets"][0]["resources"][0]["geocodePoints"][0]["coordinates"][0])
        longitude.append(data["resourceSets"][0]["resources"][0]["geocodePoints"][0]["coordinates"][1])
```

```
0.09720517724807863],
'name': "Alfred's Way, Barking, IG11 0BB, England, United Kingdom",
'point': {'type': 'Point', 'coordinates': [51.5298076, 0.0889271]},
'address': {'addressLine': "Alfred's Way",
'adminDistrict': 'England',
'adminDistrict2': 'Greater London',
'countryRegion': 'United Kingdom',
'formattedAddress': "Alfred's Way, Barking, IG11 0BB, England, United Kingdom",
'locality': 'Barking',
'postalCode': 'IG11 0BB'},
'confidence': 'Medium',
'entityType': 'RoadBlock',
'geocodePoints': [{ 'type': 'Point',
'coordinates': [51.5298076, 0.0889271]},
```

Code pour récupérer les coordonnées et exemple de dictionnaire issu de la requête à l'API

	Quartier	Nom_station	Adresse	latitude	longitude
0	Barking and Dagenham	Barking	Alfred's Way, IG11 0BB	51.529808	0.088927
1	Barking and Dagenham	Dagenham	70 Rainham Road North, Dagenham, RM10 7ES	51.559509	0.156807
2	The City of London	Dowgate	94-95 Upper Thames Street, EC4R 3UE	51.510415	-0.089807

Extrait du DataFrame contenant les adresses et coordonnées des stations

Calcul des trajets

station-incident : Avec les coordonnées des stations et des lieux d'intervention, nous avons pu calculer la distance à parcourir par les camions. L'API gratuite OSRM a été utilisée pour obtenir ces distances routières.

```
### ETAPE 3: on calcule les trajets entre les points GPS des casernes déployées et les coordonnées GPS des lieux d'intervention

import requests

# Exemple de fonction pour appeler OSRM en ligne
def calculer_distance_osrm(depart_lat, depart_lng, arrivee_lat, arrivee_lng):
    url = f'http://router.project-osrm.org/route/v1/driving/{depart_lng},{depart_lat};{arrivee_lng},{arrivee_lat}?overview=full&geometries=geojson'
    response = requests.get(url)

    if response.status_code == 200:
        data = response.json()
        distance = data['routes'][0]['distance'] / 1000.0 # distance en kilomètres
        duration = data['routes'][0]['duration'] # durée en secondes
        return distance, duration
    else:
        return None, None
```

Ce sont des calculs qui se sont révélés être très coûteux en temps de calcul, mais ont permis d'enrichir les données dont on disposait au début du projet par la distance à parcourir en kilomètres et le temps de trajet en secondes (sans prendre en compte le trafic routier).

1.3.3. Gestion des valeurs manquantes

Pour nos modèles de prédiction, nous nous intéressons aux variables disponibles avant le départ de la brigade, soit les informations temporelles (date, heure), sur le lieu (localisation, type de propriété), le type d'incident (incendie, secours) et la distance caserne-incident. Les lignes ayant des valeurs manquantes pour ces variables indépendantes ou la variable cible ont été supprimées.

Pour les analyses, nous considérons les variables potentiellement connues avant le départ de la brigade : temporelles, sur le lieu d'intervention, le type d'incident et la distance caserne-incident.

1.4. Encoding

Les données catégorielles doivent être encodées numériquement pour les algorithmes de machine learning. Le choix de la méthode d'encodage impacte les performances, donc plusieurs approches ont été explorées:

One Hot Encoding: transforme chaque valeur catégorielle en colonne binaire, générant un nombre très élevé de colonnes pour nos données. Chaque colonne correspond à une valeur unique de la variable catégorielle et contient des 0 et des 1 pour indiquer la présence ou l'absence de cette valeur dans la ligne d'observation. Vu le nombre élevé de variables catégorielles, cette méthode d'encoding générera un nombre trop important de colonnes et des sparse matrix trop grandes pour être facilement exploitées. Nous avons donc rejeté cette méthode d'encoding.

Label Encoding: attribue une valeur numérique unique de 1 à N à chaque catégorie. Puisque cette méthode attribue des variables numériques, les algorithmes peuvent considérer ces chiffres comme ayant un ordre ou une relation particulière entre eux. Nous avons donc conclu que cette méthode serait inadaptée pour les données non ordonnées comme les nôtres.

Mean Encoding: remplace chaque catégorie d'une variable par la moyenne de la variable cible pour cette catégorie, capturant la relation entre catégorie et cible. *Retenue comme la plus appropriée.*

En conclusion, l'encodage Mean Encoding a été appliqué à toutes les données catégorielles puisqu'il présente l'avantage de ne

```
# Colonnes à encoder
categorical_cols = ['IncidentGroup', 'PropertyCategory', 'IncGeo_BoroughName', 'IncGeo_WardName',
                    'IncidentStationGround', 'FirstPumpArriving_DeployedFromStation', 'Nom_station']

for col in categorical_cols:
    # Calculer La moyenne de La colonne cible pour chaque catégorie
    mean_encoded_col = train_data.groupby(col)['FirstPumpArriving_AttendanceTime'].mean()
    global_mean = y_train.mean()

    # Remplacer les valeurs de la colonne catégorielle par les moyennes calculées
    X_train[col + '_mean_encoded'] = X_train[col].map(mean_encoded_col).fillna(global_mean)
    X_test[col + '_mean_encoded'] = X_test[col].map(mean_encoded_col).fillna(global_mean)
```

pas augmenter le volume des données et aide à un apprentissage plus rapide. En effet, avec le mean encoding, nous gardons qu'une seule colonne de données alors que d'autres méthodes telles que le one hot ou l'encodage cyclique fournissent plusieurs colonnes de données qui risquent d'alourdir notre dataset. Nous avons procédé en utilisant une boucle *for* pour encoder nos datasets rapidement selon la méthode d'encodage choisie.

Le Mean Encoding utilisant les valeurs de la variable cible pour remplacer les modalités d'une variable catégorielle, cela expose à des risques d'overfitting. Le package Feature-engine propose une méthode MeanEncoder gérant ce problème. Pour éviter la fuite de données, les moyennes sont calculées sur l'entraînement puis appliquées au test, comme un transformateur. Le MeanEncoder applique également un smoothing pour pallier les déséquilibres de fréquences entre modalités, réduisant ainsi l'overfitting.

```
from feature_engine.encoding import MeanEncoder

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)

# Mean encoding avec feat-engine
mean_enc = MeanEncoder(smoothing='auto', unseen='encode')
X_train = mean_enc.fit_transform(X_train, y_train)
X_test = mean_enc.transform(X_test)
```

Cyclic Encoding: Nos datasets contiennent des données temporelles sur plusieurs années, mois, jours et heures. Ainsi, le Cyclic Encoding s'est avéré nécessaire. Cette technique représente les variables de manière continue et cyclique, plutôt que catégorielle ordinaire. Elle est particulièrement utile lorsque la périodicité des données temporelles est cruciale, comme dans l'analyse saisonnière, les séries temporelles ou toute application où les valeurs temporelles se répètent régulièrement. Le Cyclic Encoding vise à traiter ce type de problématique spécifique.

Pour résumer notre processus d'encodage, les données temporelles ont été encodées de façon cyclique et les autres données catégorielles - par le mean encoding.

2. Premier Modèle - Régression

2.1. Préparation du dataset et sélection des features

Le dataframe initial comportait 80 colonnes et 726 401 lignes. Un tri des données d'entrée les plus pertinentes pour prédire le temps de trajet était nécessaire. Nous avons donc procédé à sélectionner les features importantes par inspection. Tenant compte de notre objectif: (*prédiction du temps de réponse de la première brigade arrivant sur un incident à la suite d'un appel*) nous avons sélectionné les features connues par les services d'urgences a priori au moment de l'appel. Les données sur la distance, le temps de trajet calculés via OSRM, le type d'incident, les arrondissements, quartiers, casernes et les informations temporelles ont été conservées. Ce jeu de données a ensuite été traité dans un premier modèle de régression pour la prédiction.

Après avoir établi le dataframe, le cyclic encoding a été appliqué à la variable 'DateTimeOfCall' et le mean encoding à toutes les données catégorielles . La donnée

'FirstPumpArriving_AttendanceTime' est la variable cible. Pour éviter des calculs coûteux en temps et en ressources, seulement les interventions qui ont eu lieu après 2020 ont été prises en compte. Le Dataframe final avait donc 172156 lignes.

#	Column	Non-Null Count	Dtype
0	IncidentGroup	172156 non-null	object
1	PropertyCategory	172156 non-null	object
2	IncGeo_BoroughName	172156 non-null	object
3	IncGeo_WardName	172156 non-null	object
4	IncidentStationGround	172156 non-null	object
5	FirstPumpArriving_AttendanceTime	172156 non-null	float64
6	FirstPumpArriving_DeployedFromStation	172156 non-null	object
7	DateTimeOfCall	172156 non-null	object
8	Nom_station	172156 non-null	object
9	Distance_km	172156 non-null	float64
10	Duree_trajet_sec	172156 non-null	float64

dtypes: float64(3), object(8)
memory usage: 14.4+ MB

2.2. Implémentation du modèle (Régression)

La régression linéaire est l'une des techniques fondamentales de modélisation statistique utilisée à établir une relation linéaire entre plusieurs variables indépendantes et la variable cible afin de prédire cette dernière.

Plusieurs modèles de régression ont été évalués avec les métriques R2, MSE et MAE. Un modèle linéaire simple a d'abord été testé, suivi de régularisations avec/sans régression polynomiale pour contrôler la sélection de variables et l'overfitting. GridSearchCV a permis d'optimiser les hyperparamètres. La MAE étant la métrique comparative pour prédire le temps d'arrivée, le tableau présente les métriques de chaque modèle ainsi que la différence de MSE entre entraînement/test pour quantifier l'overfitting. Les résultats pour chaque modèle implémentés dans cette première étape sont reportés dans le tableau ci dessous:

	score R2 train	score R2 test	MAE train	MAE test	Diff MAE train-test	MSE train	MSE test	Diff MSE train-test
Modele lr naif	0.510004	0.509775	59.467610	59.467610	0.035747	8086.337690	8144.844281	-58.506591
Modele ridge	0.510004	0.509776	59.468496	59.432912	0.035584	8086.335341	8144.828315	-58.492974
Modele lasso	0.510004	0.509776	59.468170	59.432595	0.035574	8086.335486	8144.813042	-58.477555
Modele elastic	0.510004	0.509776	59.467728	59.432215	0.035513	8086.337291	8144.828825	-58.491534
Modele elastic poly	0.534601	0.528529	57.339846	57.605011	-0.265165	7680.402135	7833.258714	-152.856579

Tab 1 : Résumé des performances des modèles de régression sur l'entraînement et le test

Nous pouvons constater que les les métriques d'évaluation d'un modèle indiquent que:
le MSE élevé suggère que les prédictions du modèle sont assez éloignées des valeurs réelles en termes absolus.

le R² au alentours de 0.50 montre que le modèle ne parvient pas à expliquer une grande partie de la variance de la variable cible.

le MAE élevé montre que les prédictions ont des écarts importants avec les valeurs réelles

Nous pouvons en conclure que le modèle de régression linéaire a un efficacité très limitée en ce qui concerne le modèle étudié. Nous allons donc explorer d'autres types de modèles à la recherche de meilleures performances.

Le Random Forest est un algorithme d'apprentissage supervisé combinant de multiples arbres de décision. Chaque arbre est entraîné sur un sous-ensemble aléatoire des données avec un sous-ensemble aléatoire des variables. Cette méthode d'ensemble améliore les performances en agrégant les prédictions des différents arbres. Elle s'applique à la régression et la classification. Le Random Forest fait partie des techniques ensemblistes qui combinent plusieurs modèles pour de meilleures prédictions qu'un seul modèle. Nous obtenons les résultats suivants avec un modèle RandomForestRegressor optimisé avec GridSearch. Les tableaux ci dessus montrent la recherche des meilleurs paramètres pour Random Forest sur le modèle étudié

Nous observons une légère dégradation des résultats obtenus

```
Mean Squared Error: 11939.420537395725
Root Mean Squared Error: 109.26765549509939
R^2 Score: 0.43289980566325625
```

```
Fitting 5 folds for each of 108 candidates, totalling 540 fits
Meilleurs paramètres trouvés:
{'max_depth': None, 'min_samples_leaf': 4, 'min_samples_split': 10, 'n_estimators': 200}
```

Nous observons que 43% des données test on été bien prédites et que l'erreur moyenne sur la prédiction est de 109 secondes, donc un peu moins de deux minutes. Etant donné que le temps de trajet de l'arrivée des secours est parfois une question de sauver la santé ou la vie de quelqu'un, les résultats obtenus ne sont pas satisfaisants.

Réseau de neurones : des modèles d'apprentissage automatique inspirés du fonctionnement du cerveau humain. Ils sont composés de couches de neurones artificiels (ou unités) interconnectées. Les résultats de cet algorithme sur notre dataset sont reportés ci contre.

Nous observons que les résultats sont encore moins bons que sur le Random Forest. Dans ce cas, il est intéressant d'explorer le modèle XGBoost.

Mean Squared Error: 12982.620250121394
Root Mean Squared Error: 113.94130177473572
R ² Score: 0.38334976611436844

XGBoost: l'Extreme Gradient Boosting est un puissant algorithme d'apprentissage supervisé basé sur des arbres de décision. Il est largement utilisé pour les tâches de classification et de régression en raison de sa capacité à gérer des ensembles de données de grande taille et sa robustesse face aux problèmes de surapprentissage (overfitting).

Train Mean Squared Error: 5618.232883895958
Train Root Mean Squared Error: 74.95487231592058
Train R ² Score: 0.7263679095910744
Test Mean Squared Error: 12120.547385999414
Test Root Mean Squared Error: 110.0933575925424
Test R ² Score: 0.4242966183711234

Les résultats montrent une bonne performance sur les données d'entraînement mais une performance significativement inférieure sur les données de test, ce qui indique que le modèle peut souffrir de surapprentissage (overfitting). C'est pourquoi on procède à une recherche de paramètres optimaux pour que le modèle acquiert de meilleures performances. Les résultats de l'algorithme suite à la recherche de paramètres optimaux via GridSearch sont reportés ci contre.

Train Mean Squared Error: 10481.570231508129
Train Root Mean Squared Error: 102.37954010205422
Train R ² Score: 0.4895024765818077
Test Mean Squared Error: 11850.168727354378
Test Root Mean Squared Error: 108.8584802730333
Test R ² Score: 0.4371391000796665

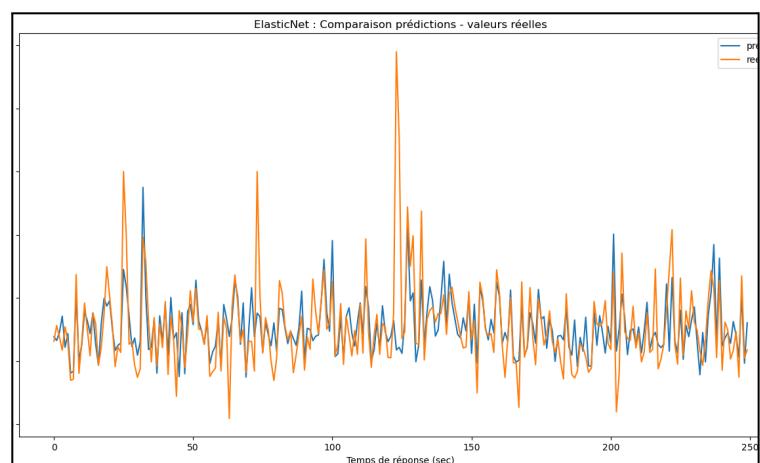
Nous pouvons voir que les prédictions données par les modèles d'apprentissage proposés ci-dessus ne sont pas concluantes. Ceci peut être expliqué par la grande disparité des temps de trajets. En effet, il est très difficile de prévoir ce genre de données à la seconde près. C'est pourquoi dans un deuxième temps les modèles de classifications vont être testés.

2.3. Interprétation des résultats

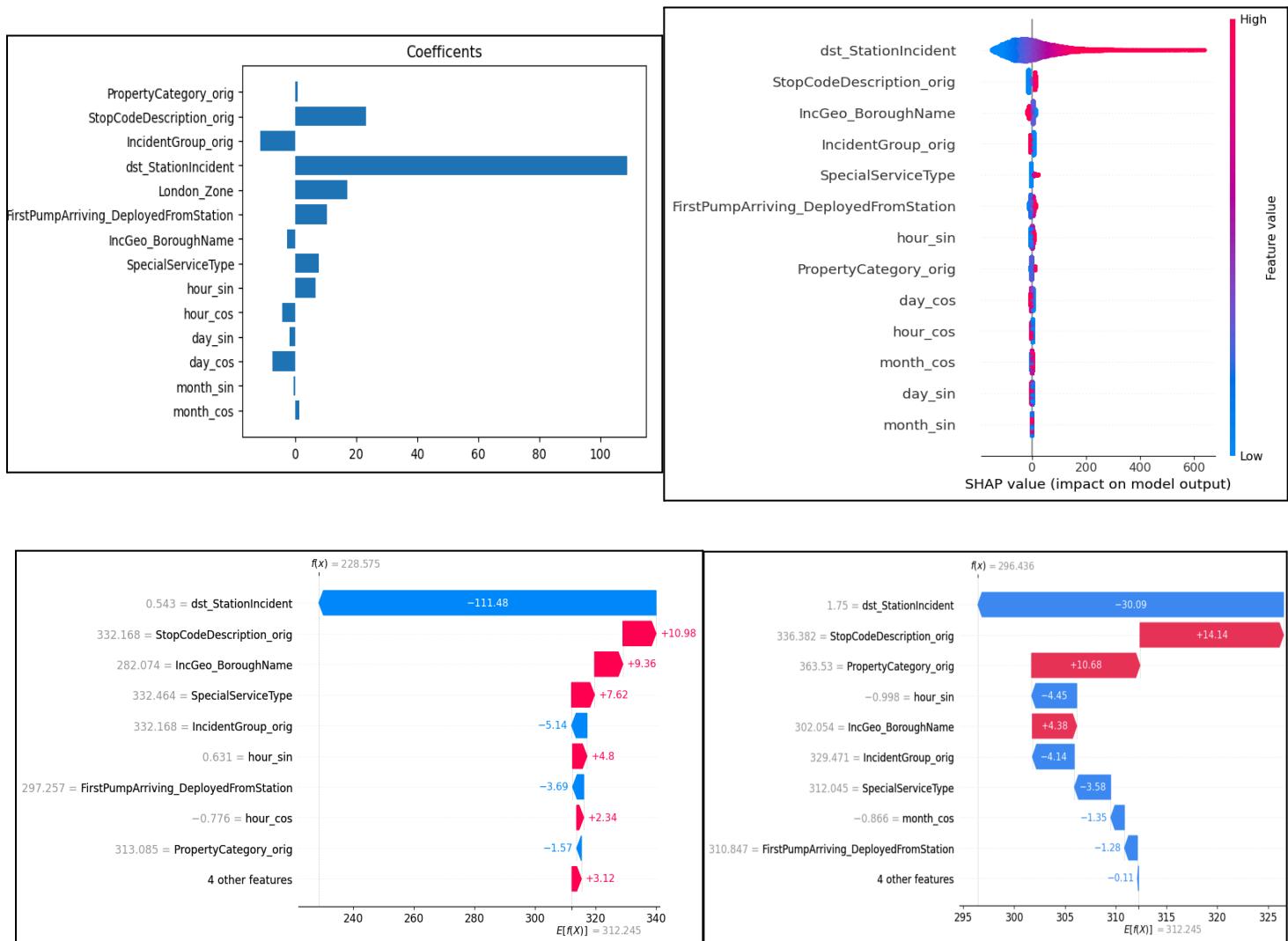
Nous observons donc que les modèles de régressions dépasse difficilement le 50% de variance expliquée (Tab 1 page 13).

Cependant, avec une MAE d'environ 1 minute, l'erreur moyenne peut être considérée comme plutôt acceptable au vu de l'objectif.

Les modèles prédisent assez bien les valeurs intermédiaires, mais pas les valeurs extrêmes. La distance à parcourir est la variable la plus importante pour les prédictions, avec un coefficient très important par rapport aux



autres variables. Une visualisation des résultats avec le package shap permet de mieux visualiser comment chaque variable impacte les prédictions. On constate que lorsque la distance augmente, la valeur de la prédiction augmente et inversement. Certaines valeurs extrêmes supérieures du temps de réponse s'expliquent par le fait que les pompiers peuvent être soumis à des évènements qui ralentissent leur interventions (trafic routier ou météo par exemple). Les erreurs faites sur ces prédictions peuvent alors s'expliquer par le fait que le modèle a du mal à prédire si la brigade a été retardée avec les variables disponibles et se base donc majoritairement sur la distance à parcourir.



Model ElasticNet : coefficients des variables , modification de la valeur SHAP en fonction de la valeur des variables et deux exemples de comment les variables ont impactées la prédiction du modèle

Si on prend quelques exemples de comment chaque variable a influé sur une prédiction du modèle, c'est généralement la distance qui a la plus la plus forte influence. Même si les autres variables impactent globalement moins fortement les variations, elles peuvent être intéressantes à conserver si on veut un grand niveau de précision dans les prédictions.

3. Deuxième Modèle - Algorithmes de Classification

3.1. Préparation du dataset et sélection des features

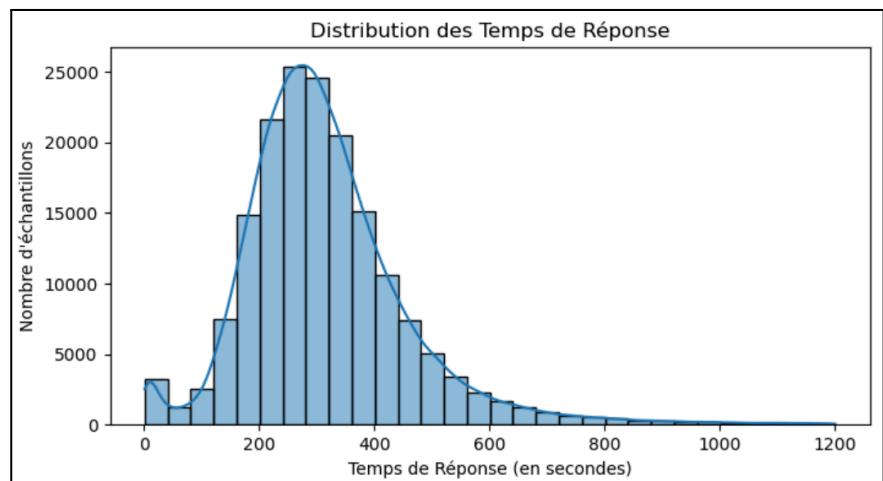
Nous avons constaté qu'un modèle de régression, même sous de formes différentes, ne produit pas les prédictions espérées. Pour la prochaine étape du projet, nous avons donc décidé d'explorer des algorithmes de classification sur nos données afin de voir si ces modèles sont plus performants dans la prédiction du temps de réponse des pompiers.

Nous avons réutilisé le dataset préparé lors du premier modèle mais nous avons effectué un traitement à la colonne cible afin de créer plusieurs classes. Pour effectuer le meilleur découpage, nous regardons la distribution du temps de réponse dans notre dataset.

Pour les modèles de classification, les temps de réponse ont été regroupés en classes. Deux découpages ont été testés pour mesurer les performances:

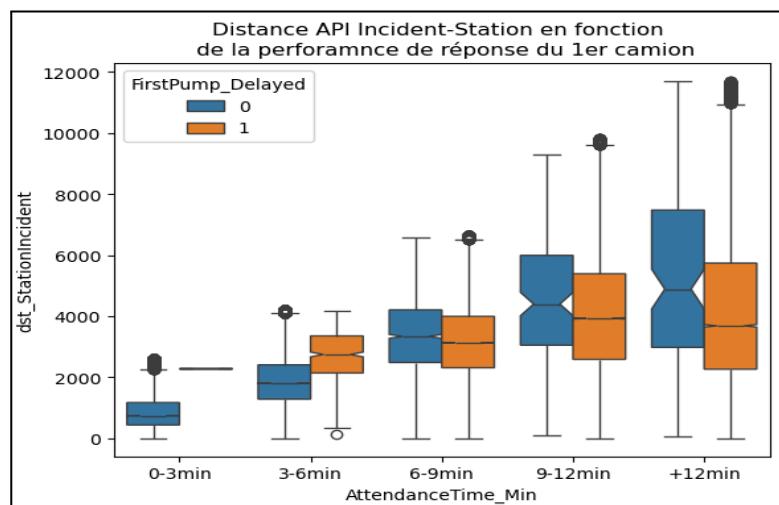
Découpage en 5 classes: [0-3], [3-6], [6-9], [9-12] minutes et plus de 12 minutes.

Découpage en classes de taille égale



Nous avons également trouvé que depuis 2009, la Brigade de Londres a des standards de temps de réponse: Ils visent un temps de réponse moyen de 6 minutes, dans les 10 minutes dans 90% des cas et dans les 12 minutes dans 92% des cas. Les découpages suivent cette logique sur les performances attendues.

Nous avons créé un boxplot des temps de réponse en fonction de la distance caserne - incident. Nous avons supprimé les valeurs extrêmes supérieures. Le boxplot illustre bien notre intuition qui est de dire que le temps de réponse est plus long plus la distance caserne - incident est longue



3.2. Implémentation du modèle de classification

3.2.1. Découpage initial

Une première tentative de classification naïve a été faite avec un algorithme Random Forest sans prendre en compte le fait que les classes peuvent être déséquilibrées. Le pourcentage de bonnes prédictions moyennes est correct (71%). Mais lorsqu'on calcul la balanced accuracy (rappel moyenne de chaque classe), le modèle tombe à 40% de bonnes prédictions pour le test. En effet, quand on regarde les classes individuellement, on voit que les scores des classes les plus minoritaires (3 et 4) sont globalement moins bons que les majoritaires(0,1 et 2).

Rapport de classification et matrice de confusion de la RandomForest naïve

Par la suite, différentes méthodes ont été testées pour traiter le déséquilibre de classe. L'optimisation des hyperparamètres des modèles a été faite par cross-validation avec *GridSearchCV* avec comme critère de performance '*balance_accuracy*'. Les résultats ont été évalués avec les métriques du package *imblearn balanced_accuracy_score* et *classification_report_imbalanced*.

predits	0-3min	3-6min	6-9min	9-12min	+12min	
reels						
0-3min	2414	4080	17	0	1	
3-6min	1140	39882	3059	15	6	
6-9min	65	8061	8657	140	9	
9-12min	11	674	1680	449	53	
+12min	9	293	387	172	85	

	pre	rec	spe	f1	geo	iba	sup
0	0.66	0.37	0.98	0.48	0.60	0.34	6512
1	0.75	0.90	0.52	0.82	0.69	0.49	44102
2	0.63	0.51	0.91	0.56	0.68	0.44	16932
3	0.58	0.16	1.00	0.25	0.39	0.14	2867
4	0.55	0.09	1.00	0.15	0.30	0.08	946
avg / total	0.71	0.72	0.68	0.70	0.66	0.44	71359

Arbre de Décision: Pour gérer le déséquilibre de classe, une première approche a été d'utiliser le paramètres *class_weight* du classifieur *DecisionTreeClassifier* qui donne un poids au classe inversement proportionnel à leur fréquence. Les résultats sont meilleurs pour les classes minoritaires que le modèle naïf sans pour autant être excellents.

Rapport de classification et matrice de confusion de l'arbre de décision avec équilibre du poids des classes

	pre	rec	spe	f1	geo	iba	sup
0-3min	0.65	0.39	0.98	0.49	0.62	0.36	6540
3-6min	0.76	0.90	0.54	0.82	0.70	0.50	44066
6-9min	0.63	0.53	0.90	0.58	0.69	0.46	16859
9-12min	0.56	0.19	0.99	0.28	0.43	0.17	2863
+12min	0.58	0.07	1.00	0.13	0.27	0.07	1031
avg / total	0.71	0.72	0.69	0.70	0.67	0.46	71359

BalancedRandomForestClassifier: Un classifieur *BalancedRandomForestClassifier* permet de prendre en compte le déséquilibre des données en faisant des bootstrap sur les classes minoritaires (les échantillons sont rééquilibrés entre les différentes classes). Les résultats obtenus ne sont pas meilleurs qu'avec l'arbre de décision.

	pre	rec	spe	f1	geo	iba	sup	predicts	0-3min	3-6min	6-9min	9-12min	+12min
0-3min	0.32	0.83	0.82	0.46	0.83	0.68	6540	reels					
3-6min	0.81	0.52	0.81	0.64	0.65	0.41	44066	0-3min	5451	1031	20	0	38
6-9min	0.49	0.44	0.86	0.46	0.61	0.36	16859	3-6min	11106	23062	7079	160	2659
9-12min	0.26	0.33	0.96	0.29	0.57	0.30	2863	6-9min	563	3791	7387	2309	2809
+12min	0.07	0.45	0.91	0.12	0.64	0.39	1031	9-12min	85	319	589	955	915
avg / total	0.66	0.52	0.83	0.56	0.65	0.42	71359	+12min	68	120	123	256	464

Rapport de classification et matrice de confusion de l'arbre de décision avec équilibre du poids des classes

Pour tenter d'améliorer les résultats, on termine avec des algorithmes de Boosting et Bagging.

AdaBoostClassifier: Les modèles AdaBoost permettent de gérer des données déséquilibrées en entraînant un classifieur faible plusieurs fois en corrigeant le poids des mauvaises classifications et en rééchantillonnant automatiquement les classes. Un arbre de décision est utilisé comme estimateur de base.

Les résultats obtenus ne sont pas très concluants et à peine meilleur que les modèles précédents.

Rapport de classification du modèle AdaBoost

	pre	rec	spe	f1	geo	iba	sup
0-3min	0.36	0.79	0.86	0.49	0.82	0.68	6540
3-6min	0.82	0.60	0.78	0.69	0.69	0.46	44066
6-9min	0.48	0.47	0.84	0.48	0.63	0.38	16859
9-12min	0.23	0.41	0.94	0.29	0.62	0.37	2863
+12min	0.12	0.32	0.96	0.17	0.56	0.29	1031
avg / total	0.66	0.58	0.81	0.60	0.68	0.46	71359

RUSBoostClassifier : Le RUSBoostClassifier intègre un sous-échantillonnage aléatoire à un modèle AdaBoost. Les résultats obtenus sont légèrement moins bons que ceux de l'AdaBoost.

	blc_acc_train	blc_acc_test	geo_mean_train	geo_mean_test	f1_train	f1_test
RF_naive	0.998691	0.416857	0.999158	0.708516	0.999234	0.702885
DTC	0.517661	0.506485	0.675009	0.672611	0.584196	0.580935
Blc_RandomForest	0.535216	0.515723	0.661978	0.657836	0.562430	0.557715
AdaBoostDTC	0.564206	0.520313	0.690414	0.684919	0.606371	0.599955
RUSBoostClassifier	0.548834	0.518822	0.683902	0.679137	0.596526	0.590996

Résumé des performances des modèles sur l'entraînement et le test (geo_mean : moyenne géométrique, blc_acc : balanced accuracy, f1 : score F1)

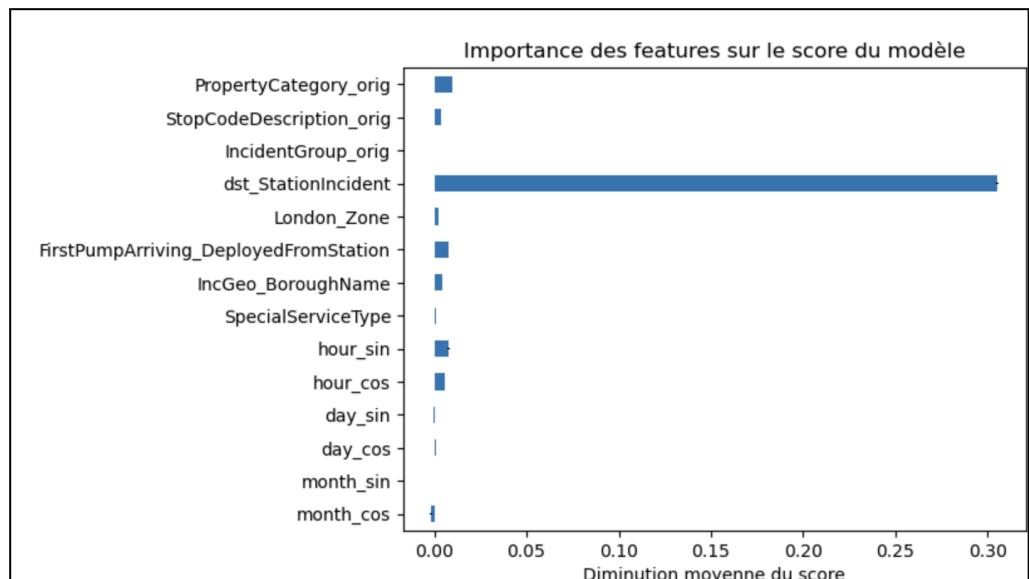
Parmi les modèles testés pour ce premier découpage, lorsqu'on regarde le taux de bonne prédiction moyen, on dépasse à peine les 50%. Les meilleurs résultats sont obtenus avec le modèle AdaBoost.

predits	0-3min	3-6min	6-9min	9-12min	+12min	reels	0-3min	3-6min	6-9min	9-12min	+12min
0-3min	0.793119	0.198165	0.005199	0.000000	0.003517	0-3min	0.355031	0.040919	0.002109	0.000000	0.006145
3-6min	0.200563	0.587846	0.172264	0.009781	0.029547	3-6min	0.604928	0.817883	0.470906	0.082662	0.347849
6-9min	0.027166	0.235779	0.456077	0.196097	0.084880	6-9min	0.031348	0.125505	0.476985	0.634062	0.382314
9-12min	0.025498	0.119804	0.229130	0.406217	0.219350	9-12min	0.004997	0.010830	0.040695	0.223053	0.167780
+12min	0.052376	0.149370	0.145490	0.304559	0.348206	+12min	0.003696	0.004862	0.009305	0.060222	0.095912

a) Rappel à gauche (%sur les lignes) et b) précision à droite (%sur les colonnes) sur chaque classe

Ce tableau permet de voir plus précisément si chaque classe est bien détectée par le modèle (a), et combien de fois le modèle a juste lorsqu'il prédit une classe (b). Étant donné que nos classes sont ordinaires, il est intéressant de voir si quand le modèle fait une erreur, il tombe plus ou moins loin du résultat correct. Lorsque le modèle fait une erreur sur la détection d'une classe (a), la prédiction va le plus souvent se trouver dans la classe juste inférieure ou supérieure. Même si le modèle manque souvent de précision (b), si on regarde en détail, il a tendance à surestimer le temps de réponse, sauf pour la classe 0-3 min où il surestime en 3-6 min. Si on garde en tête notre objectif de prévoir le temps d'arriver des premiers pompiers sur un incident, ce n'est pas une mauvaise chose de donner une estimation erronée si au final la brigade arrive plus rapidement.

Comme supposé en premier lieu et montré dans le graphique ci contre, c'est bien la distance entre la station intervenant et le lieu d'incident qui impact le plus les prédictions du modèles



3.2.2. Nouveaux découpages

Suite à ces résultats, un nouveau découpage équilibre a été effectué. Cinq classes équilibrées ont été déterminées pour éviter un éventuel biais dans nos modèles de

classification. Ceux-ci favoriseraient les classes majoritaires au détriment des minoritaires. Nous évitons donc ce type de biais avec un découpage en 5 classes, chaque classe regroupant 20% des incidents de notre dataset. Les classes sont illustrées dans le tableau.

Comme avec le premier découpage, nous avons appliqué plusieurs algorithmes différents sur le dataset.

Random Forest: comme avant, nous avons commencé avec un algorithme Random Forest qui donne les résultats ci-contre. Les résultats ne s'améliorent pas. Nous notons une accuracy inférieure à 50% et des valeurs de précision, recall et f1 insatisfaisantes.

Bin 1: 1.0 - 211.0 secondes, Pourcentage : 19.86%
Bin 2: 211.0 - 269.0 secondes, Pourcentage : 19.99%
Bin 3: 269.0 - 325.0 secondes, Pourcentage : 20.08%
Bin 4: 325.0 - 405.0 secondes, Pourcentage : 20.06%
Bin 5: 405.0 - 1200.0 secondes, Pourcentage : 20.00%
Total des pourcentages : 100.00%

Accuracy: 0.4866748762750064					
Classification Report:					
	precision	recall	f1-score	support	
1	0.64	0.63	0.64	8671	
2	0.40	0.47	0.44	8608	
3	0.37	0.37	0.37	8618	
4	0.41	0.38	0.40	8504	
5	0.64	0.57	0.60	8638	
accuracy			0.49	43039	
macro avg	0.49	0.49	0.49	43039	
weighted avg	0.49	0.49	0.49	43039	
Classe prédictive	1	2	3	4	5
Classe réelle					
1	5506	2061	539	323	242
2	1872	4086	1879	558	213
3	672	2389	3152	1844	561
4	318	1033	2121	3272	1760
5	297	553	865	1993	4930

Régression Logistique Multi-Classes: Extension de la régression logistique binaire, elle permet de prédire l'appartenance d'un échantillon à l'une des K classes possibles ($K > 2$). Contrairement à la binaire qui prédit une probabilité pour deux classes, la multi-classes attribue une probabilité à chacune des K classes. L'échantillon est alors classé dans la classe ayant la probabilité la plus élevée. Les résultats de cet algorithme sont dans le tableau reporté ci après. Nous ne relevons aucune amélioration des résultats avec cet algorithme. L'accuracy est même inférieure que la Random Forest ainsi que les metrics de precision, recall et f1. Nous considérons donc cet algorithme comme insatisfaisant.

Accuracy: 0.42949417969748366					
Classification Report:					
	precision	recall	f1-score	support	
1	0.50	0.68	0.58	8671	
2	0.34	0.29	0.31	8608	
3	0.32	0.25	0.28	8618	
4	0.37	0.32	0.34	8504	
5	0.53	0.60	0.56	8638	
accuracy			0.43	43039	
macro avg	0.41	0.43	0.42	43039	
weighted avg	0.41	0.43	0.42	43039	
Classe prédictive	1	2	3	4	5
Classe réelle					
1	5901	1353	568	301	548
2	3216	2516	1608	753	515
3	1414	1959	2197	1908	1140
4	683	1108	1626	2721	2366
5	481	547	812	1648	5150

XGBoost : Bibliothèque performante pour la classification, gérant efficacement les données complexes. En optimisant les hyperparamètres et utilisant la validation croisée, XGBoost s'avère très efficace pour divers problèmes de classification. Son principal atout est de combiner plusieurs modèles faibles en un modèle puissant, pouvant améliorer les performances de prédiction. Nous constatons des résultats légerement

Train Accuracy: 0.5990535715668734					
Test Accuracy: 0.4863728246474128					
Rapport de Classification :					
	precision	recall	f1-score	support	
0	0.64	0.64	0.64	8679	
1	0.40	0.47	0.43	8625	
2	0.37	0.36	0.36	8632	
3	0.41	0.38	0.39	8569	
4	0.62	0.58	0.60	8534	
accuracy			0.49	43039	
macro avg	0.49	0.49	0.49	43039	
weighted avg	0.49	0.49	0.49	43039	
Classe prédictive	0	1	2	3	4
Classe réelle					
0	5596	1970	552	315	246
1	1900	4044	1853	612	216
2	685	2362	3079	1924	582
3	349	1081	1946	3224	1969
4	282	587	860	1815	4990

supérieurs aux algorithmes précédents mais nous cherchons tout de même de meilleures performances

Classification avec les réseaux de neurones : Il s'agit ici de modèles d'apprentissage automatique inspirés par la structure du cerveau humain. Ils sont composés de couches de neurones (ou nœuds) qui sont connectées entre elles. Ces modèles sont particulièrement efficaces pour les tâches de classification lorsqu'ils sont correctement configurés et entraînés. Nous constatons ici aussi des résultats équivalents aux algorithmes précédents

Rapport de Classification :					
	precision	recall	f1-score	support	
0	0.62	0.59	0.60	8671	
1	0.39	0.42	0.40	8608	
2	0.34	0.36	0.35	8618	
3	0.39	0.38	0.38	8504	
4	0.60	0.55	0.57	8638	
accuracy				0.46	43039
macro avg	0.47	0.46	0.46	43039	
weighted avg	0.47	0.46	0.46	43039	
Classe prédictive	0	1	2	3	4
Classe réelle					
0	5094	2125	776	363	313
1	1853	3588	2159	681	327
2	658	2106	3131	1995	728
3	325	930	2188	3211	1850
4	285	489	1019	2090	4755

Pour chercher de meilleures performances, nous avons vérifié comment les différents modèles réagissent au nombre de classes selon lesquelles est partagée la variable cible. Voici un tableau de comparaison entre trois découpages différents, toujours choisis de façon à ce qu'ils soient équitables.

	BINS=5	BINS=4	BINS=3
Regression Logistique Multi Classees	Accuracy = 0.429	Accuracy = 0.499	Accuracy = 0.606
Random Forest Classifier	Accuracy = 0.487	Accuracy = 0.557	Accuracy = 0.657
Réseau de Neurones	Accuracy = 0.457	Accuracy = 0.531	Accuracy = 0.628
XGBoost	Train Accuracy = 0.54 Test Accuracy = 0.48	Train Accuracy = 0.61 Test Accuracy = 0.56	Train Accuracy = 0.69 Test Accuracy = 0.66
Intervalles	1s -> 211s 211s -> 269s 269s -> 325s 325s -> 405s 405s -> 1200s	1s -> 227s 227s -> 296s 296s -> 379s 379s -> 1200s	1s -> 251s 251s -> 346s 346s -> 1200s

Nous constatons que les performances des prédictions s'améliorent de façon considérable, si le nombre de classes est réduit. Ceci reste vrai pour tous les modèles testés.

Le modèle XGBoost, avec Bins = 3 est l'algorithme avec les meilleurs résultats. les tableaux ci contre montrent les résultats. Le but de cette division en classes était qu'il soit équilibré pour éviter le surapprentissage des modèles étudiés. Comme nous pouvons le voir sur le graphique qui recense toutes les réponses notées dans le dataframe (page 5), la majorité des réactions des casernes sont comprises entre 3 et 7 minutes. Cela indique

Train Accuracy: 0.6937661191012803 Test Accuracy: 0.6599363368107996 Rapport de Classification :					
	precision	recall	f1-score	support	
0	0.71	0.73	0.72	14414	
1	0.54	0.57	0.56	14175	
2	0.74	0.68	0.70	14450	
accuracy				0.66	43039
macro avg	0.66	0.66	0.66	43039	
weighted avg	0.66	0.66	0.66	43039	
Classe prédictive	0	1	2		
Classe réelle					
0	10561	3170	683		
1	3279	8087	2809		
2	1076	3619	9755		

que si on veut considérer un nombre de classes plus grand (par exemple Bins = 5), tout en les gardant équilibrées, les intervalles de temps qu'englobe chacune des classes du milieu (bins 2, 3 et 4) est d'environ 60 secondes. Il est donc difficile d'effectuer des prévisions précises pour des laps de temps aussi petits. En réduisant le nombre d'intervalles, nous évitons ce genre de complications, et il est possible d'avoir de meilleurs résultats. Nous pouvons voir que les intervalles optimaux seraient bin1 = 0-4 minutes, bin2 = 4-6 minutes bin3 = plus de 6 minutes.

3.3. Interprétation des résultats

Aucun des modèles de classification n'a permis d'obtenir de résultats vraiment satisfaisants. En vue de notre objectif, les performances de prédictions sur les temps de réponse les plus courts sont les plus intéressantes à regarder. Les erreurs faites sur les temps de réponse les plus longs peuvent être moins importantes à prendre en compte comparée aux erreurs sur les temps de réponse courts: Par exemple, si on prédit un temps de réponse entre 9 et 12 minutes alors que le camion arrive en 6 minutes, on peut considérer que l'erreur n'est pas importante car au final la performance de la brigade intervenant est meilleure. Une erreur faite sur une prédiction pessimiste du temps de réponse serait donc moins importante à prendre en compte qu'une prédiction optimiste. Il aurait donc été intéressant d'essayer de minimiser les erreurs de prédiction pour les classes représentant les meilleures performances d'arrivée en leur donnant différents poids. Lorsqu'un modèle fait une prédiction avec un écart d'une classe par rapport au résultat réel, un autre point intéressant aurait été de regarder le temps de réponse en secondes pour vérifier l'importance de l'écart avec les limites d'une classe. Par exemple, pour le premier type de découpage, un temps de réponse de 170 secondes a probablement des similitudes avec un temps de 190 secondes. Mais l'un appartient à la classe [0-3] et l'autre à la classe [3-6]. Il peut donc être compliqué pour les modèles d'être précis dans ces cas de figure.

4. Conclusion et Ouverture

Ce projet a exploré différentes approches de modélisation, allant des régressions linéaires aux algorithmes de classification en passant par des méthodes d'ensemble comme les forêts aléatoires et le Boosting.

Bien que les résultats des modèles de régression soient mitigés, avec une prédiction acceptable des valeurs intermédiaires mais des difficultés avec les extrêmes, ils ont permis d'identifier la distance comme facteur prépondérant du temps de réponse. Les modèles de classification, malgré des performances généralement inférieures à 60% d'accuracy, ont fourni des informations intéressantes sur la criticité des erreurs de prédiction. Une sous-estimation du temps de réponse est préférable à une surestimation dans un contexte opérationnel.

Si ce projet n'a pas permis de résoudre entièrement la problématique de prédiction précise des temps de réponse, il a néanmoins ouvert des pistes prometteuses. L'intégration de données supplémentaires sur le trafic routier et les conditions météorologiques pourrait améliorer les performances des modèles. De plus, une approche visant à minimiser les erreurs

sur les temps de réponse les plus courts, cruciaux pour sauver des vies, mériteraient d'être explorée.

En fin de compte, ce projet démontre le potentiel de l'IA et de l'analyse de données pour optimiser les opérations des services d'urgence, tout en soulignant la complexité inhérente à la prédiction précise de phénomènes dynamiques et influencés par de multiples facteurs. Les réflexions et suggestions des lecteurs sont les bienvenues pour faire progresser ces travaux.

Bibliographie:

Sampalis, J. S., Lavoie, A., Williams, J. I., Mulder, D. S., & Kalina, M. (1993). Impact of on-site care, prehospital time, and level of in-hospital care on survival in severely injured patients. *The Journal of Trauma: Injury, Infection, and Critical Care*, 34(2), 252–261.

Feero, S., Hedges, J. R., Simmons, E., & Irwin, L. (1995). Does out-of-hospital EMS time affect trauma survival? *The American Journal of Emergency Medicine*, 13(2), 133–135.

Xin, J., & Huang, C. (2013). Fire risk analysis of residential buildings based on scenario clusters and its application in fire risk management. *Fire Safety Journal*, 62, 72–78.