



# Hackathon




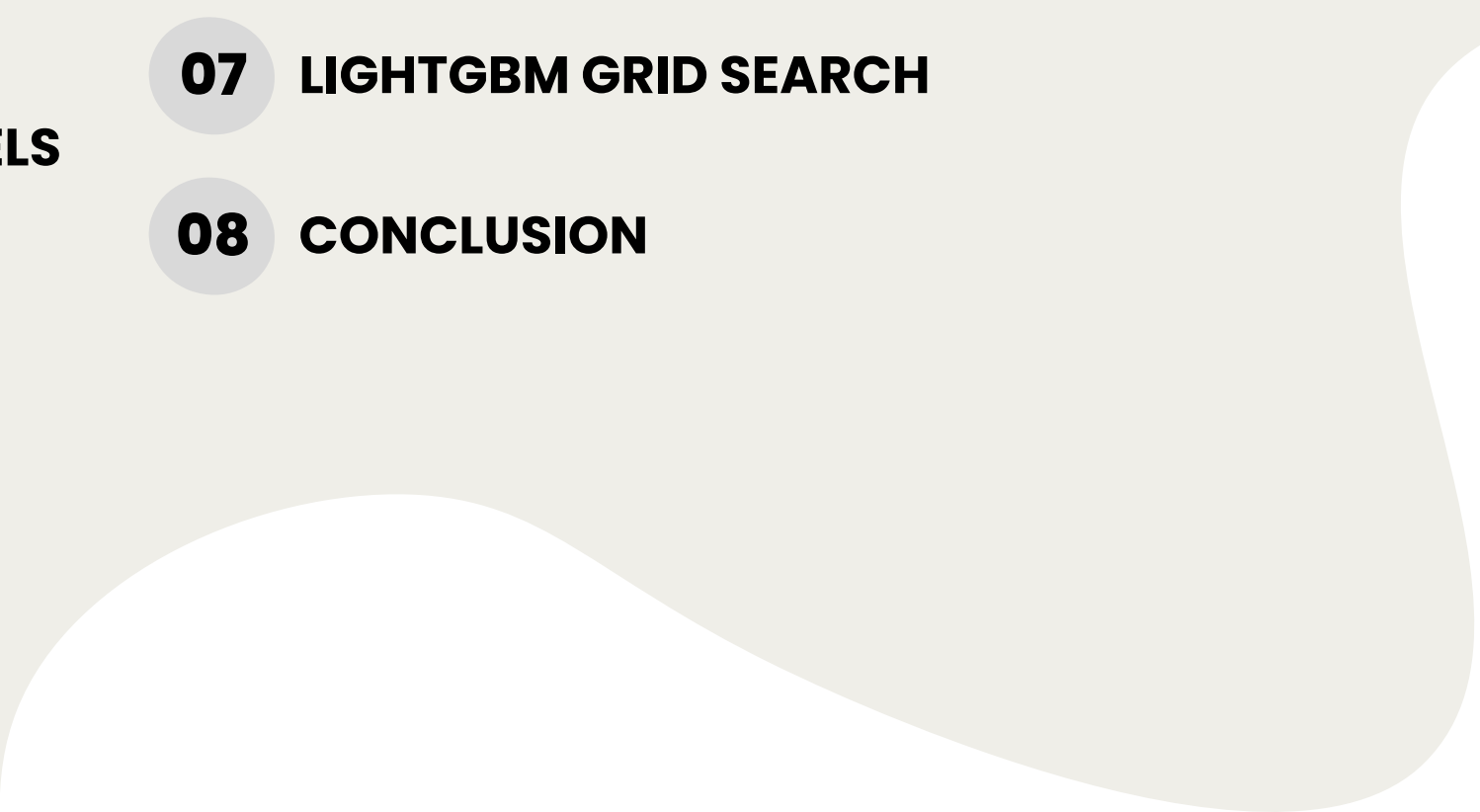

# Eleven Strategy

14/02/2024

Hans Célestin  
Le Breton Titouan  
Vielzeuf Charles  
Wauquiez Mathis


# Sommaire

---

- 
- 
- 
- 
- 
- 01 ORGANISATION DU TRAVAIL (GITHUB, BRANCH, RÉPARTITION ET PRIORISATION)**
  - 02 TRAITEMENTS DES INSTANCES (ONE HOT, RENORMALISATIONS, MEANS...)**
  - 03 PREMIERS RESULTATS AVEC AVG, ET VOISINAGE DE DONNEES**
  - 04 ENTRAINEMENT DE MLP, TESTS SUR DIFFERENTS INPUTS, DIFFERENTS MODELS ET CHOIX DES HYPERPARAMETRES**
  - 05 APPROCHE PAR ATTRACTION POUR LES JOURS DE LA SEMAINE**
  - 06 ESSAI RAPIDE DE DIFFÉRENTS MODÈLES (RANDOM FORESTS, LIGHTGBM, XGBOOST, CATBOOST)**
  - 07 LIGHTGBM GRID SEARCH**
  - 08 CONCLUSION**

# Organisation du travail

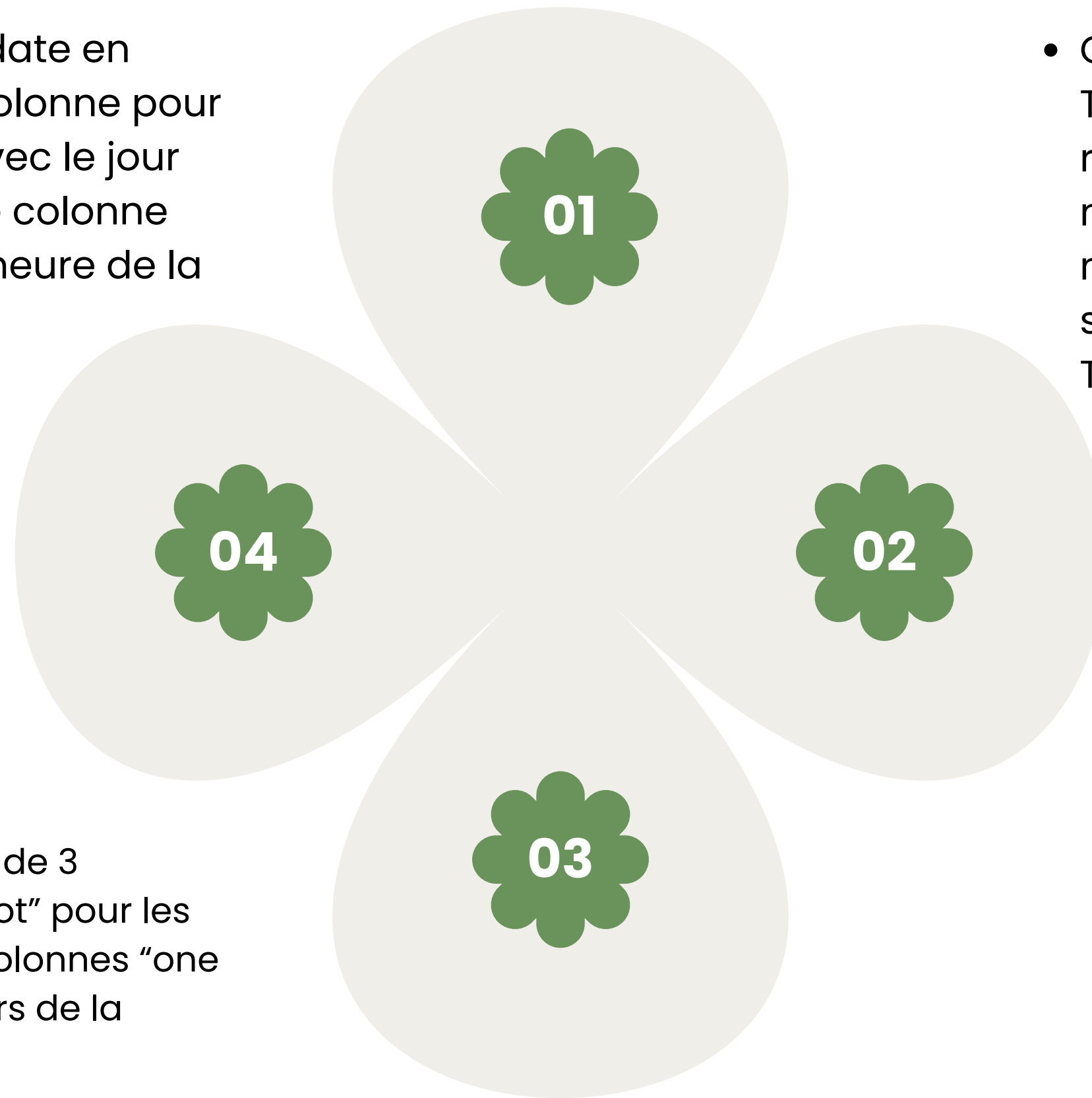
## METHODES

- 
- 01 Discussions préliminaires sur la manière d'aborder le problème
  - 02 Travail efficace sur Github
  - 03 Répartition des tâches pour avancer plus rapidement

# Traitement des instances

- Gestion de la date en prenant une colonne pour le mois, une avec le jour du mois, et une colonne pour la demi-heure de la journée

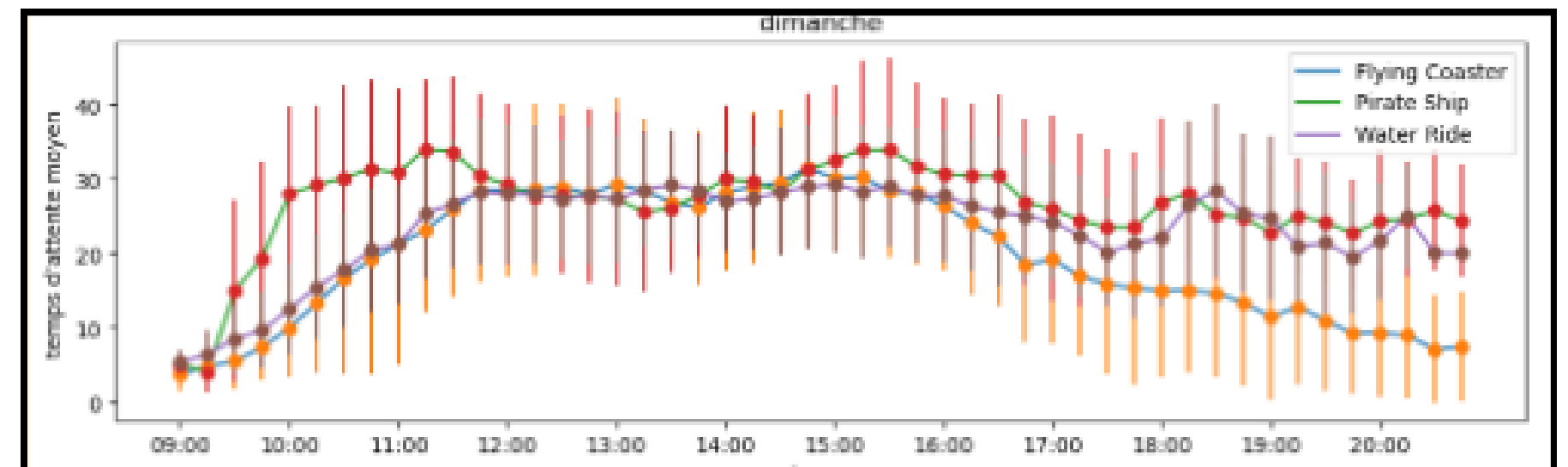
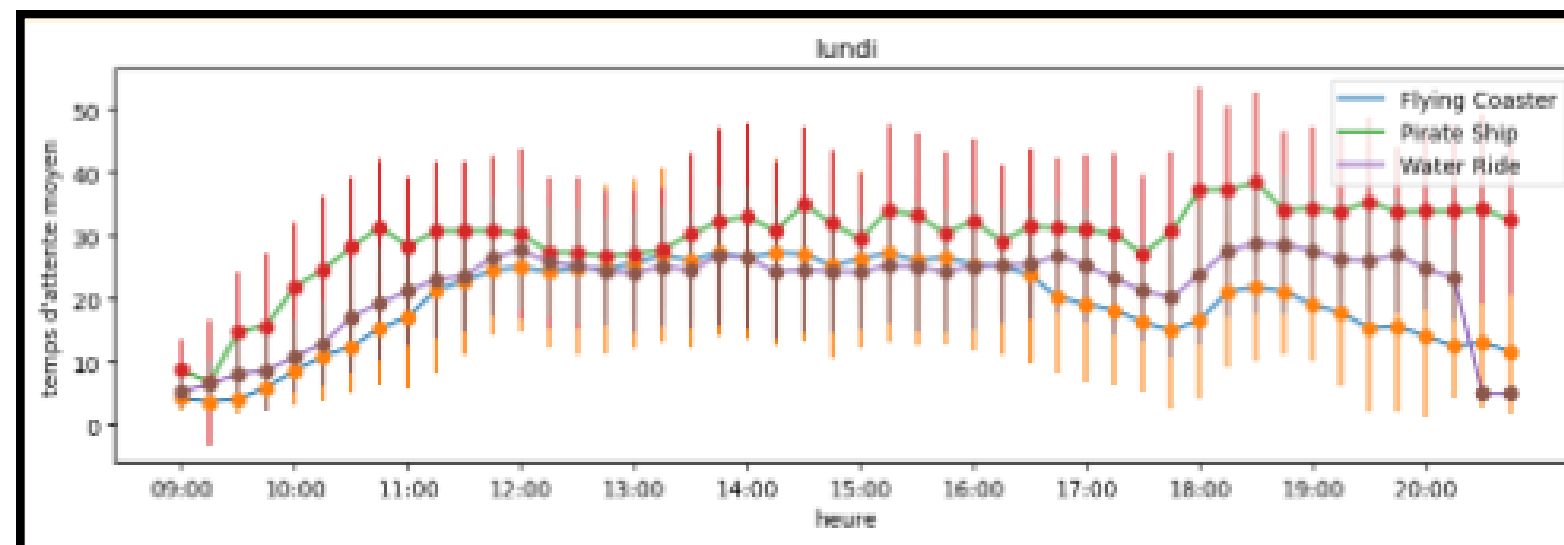
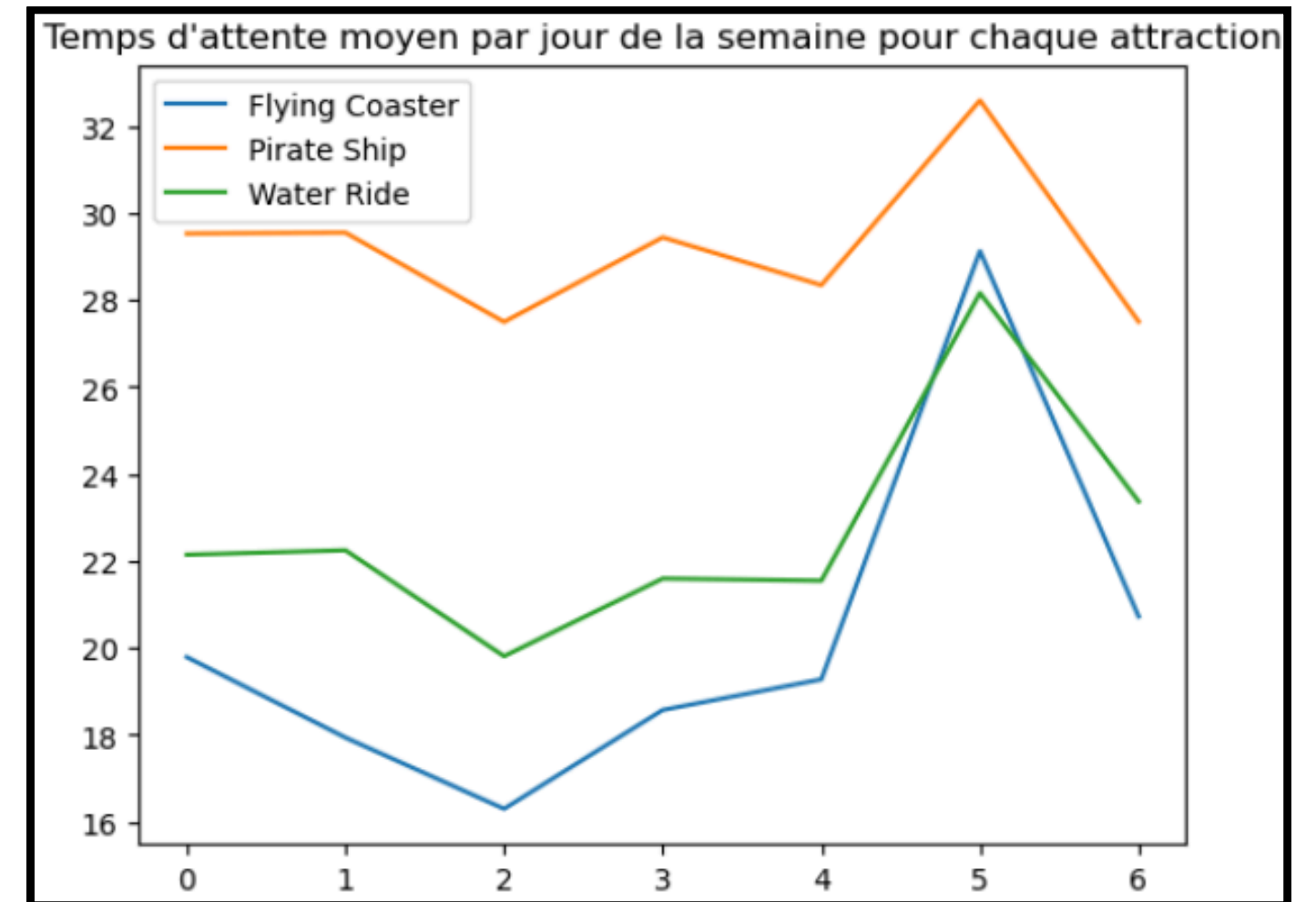
- Implémentation de 3 colonnes "one hot" pour les manèges / de colonnes "one hot" pour les jours de la semaine



- Gestion de Time\_to\_parade\_1 en remplaçant les valeurs manquantes par la moyenne des valeurs et suppression de Time\_to\_parade\_2
- Renormalisation de données (sauf les "one hot")

# Gestion plus fine des données

- 01 • Colonnes “one hot” sur les jours de la semaine, division en catégories semaine / week-end
- 02 • Implémentation des jours fériés avec la librairie python “holidays”
- 03 • Implémentation des données météorologiques (les plus pertinentes : la pluie, la température)
- 04 • Divisions différentes des temps de la journée (minutes, 1/4 d’heures, 1/2 heures, heures)



# Premiers resultats avec AVG, et voisinage de donnees

01 idee: chercher pour les données de validation les donnees **les plus proches** dans la table de train

02 **les plus proches** = clé {date+manège} -> améliorations possibles via pondération des autres données clés (pluie, jour semaine...)

03 Scores pas très bons (20 environ sans temps d attente (tt à zéro)) et **YOUR SCORE: RMSE = 13.45** quand on prend la ligne la plus proche avec clé {date+manège}

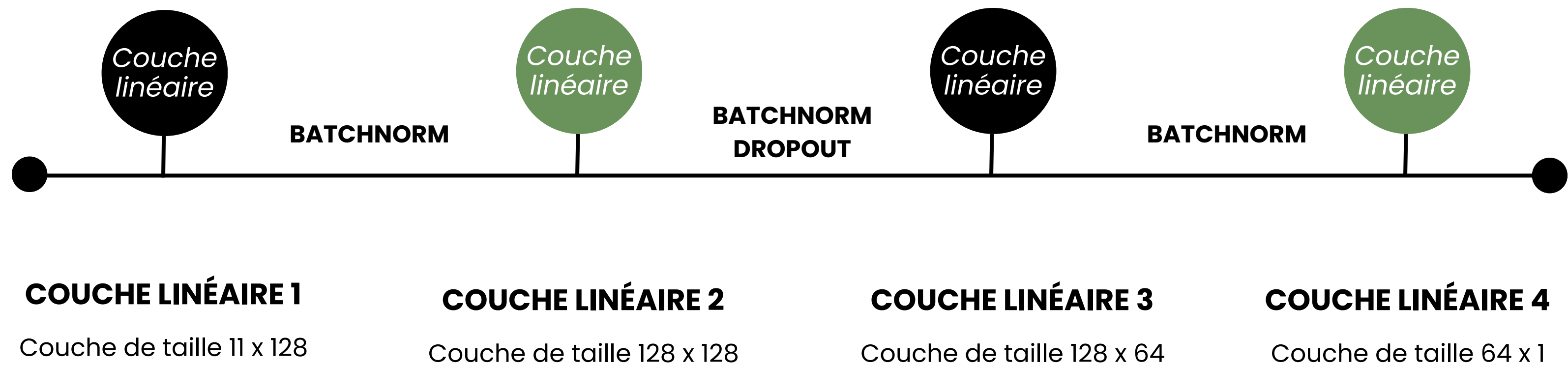
```
# PEU CONCLUANT - Idée : pour chaque date de validation, trouver la date la plus proche dans wtg_times et prendre le temps d'attente associé

# create the distance map btwn validation['DATETIME'] and wtg_times['DATETIME']
distance = np.abs(wtg_times['DATETIME'].values[:, None] - validation['DATETIME'].values[None, :])
distance2 = np.abs(wtg_times['DATETIME'].values[:, None] - validation['DATETIME'].values[None, :] - 2)
# get the index of the minimum distance for each row
index = np.argmin(distance, axis=0)
index2 = np.argmin(distance2, axis=0)

# get the index of the minimum value between distance and distance2
index = np.where(distance[index, np.arange(len(index))] < distance2[index2, np.arange(len(index2))], index, index2)

# get the corresponding datetime
closest_datetime = wtg_times['DATETIME'].iloc[index]
# get the corresponding waiting time
closest_waiting_time = wtg_times['CURRENT_WAIT_TIME'].iloc[index]
```

# Le réseau MLP





# Autres méthodes



Approche par attraction



Essai rapide de différents modèles (random forests, lightgbm, xgboost, catboost)



lightgbm: grille de recherche pour trouver les meilleurs hyperparamètres / approche par attraction



# Résultats

	RANDOM FORESTS	LightGBM	CatBoost	XGBOOST
RMSE sur le jeu de test	8.37	7.15 / 6.70	10.08	7.82
RMSE sur le jeu de validation	9.96	8.70 / 8.62	11.15	8.81

# CONCLUSION ET AMELIORATION POSSIBLES

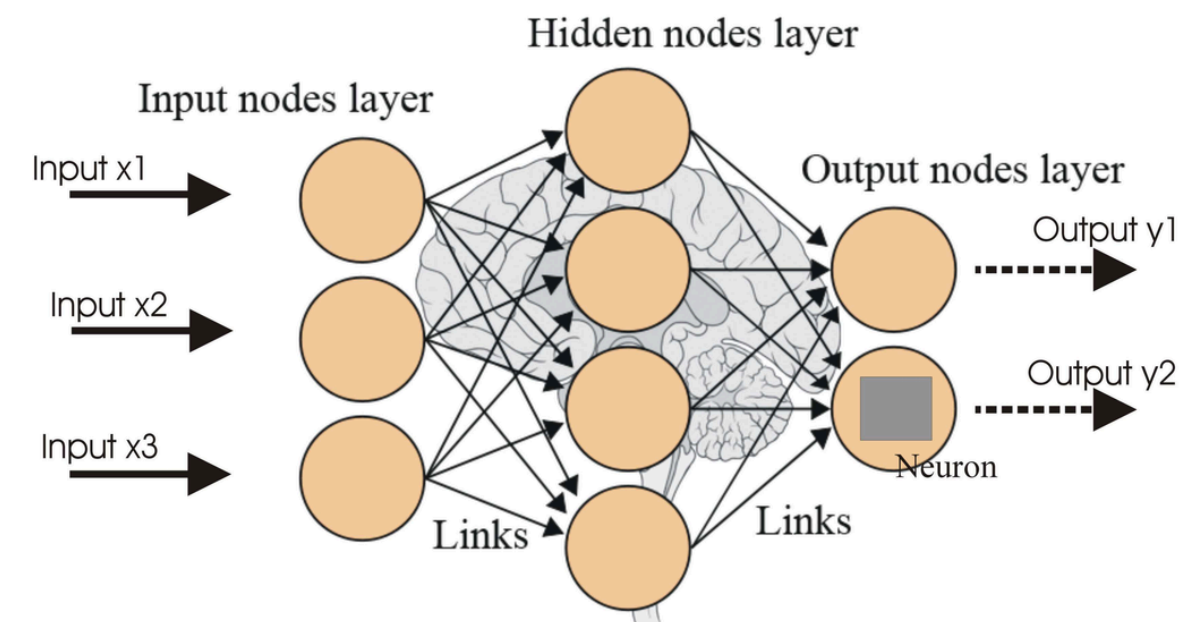
Nombre de valeurs manquantes par colonne dans Train sur un total de 37018 lignes et 9 colonnes	
DATETIME	0
ENTITY_DESCRIPTION_SHORT	0
ADJUST_CAPACITY	0
DOWNTIME	0
CURRENT_WAIT_TIME	0
TIME_TO_PARADE_1	14024
TIME_TO_PARADE_2	30806
TIME_TO_NIGHT_SHOW	14024
WAIT_TIME_IN_2H	0
dtype: int64	

+ Code

+ Markdown

```
print("Nombre de valeurs manquantes par colonne dans Test "+str(Test.shape[0])+" lignes et "+str(Test.shape[1])+" colonnes ")
Test.isnull().sum()
```

Nombre de valeurs manquantes par colonne dans Test 2444 lignes et 8 colonnes	
DATETIME	0
ENTITY_DESCRIPTION_SHORT	0
ADJUST_CAPACITY	0
DOWNTIME	0
CURRENT_WAIT_TIME	0
TIME_TO_PARADE_1	1080
TIME_TO_PARADE_2	2098
TIME_TO_NIGHT_SHOW	1080
dtype: int64	



**SPÉCIFICITÉ DU JEU DE DONNÉES: DÉCOUPER EN PLUSIEURS TRAINING (ENVIRON LA MOITIÉ DE DONNÉES RENSEIGNÉES SUR CERTAINES COLONNES: SPLIT PLUTOT QUE METTRE DES AVG°**

**TRAINING PLUS CONSÉQUENT, DATA AUGMENTATION, MEILLEURE GESTION DES INPUTS...**