Report of the project: Prediction les ventes hebdomadaries

I – Data preprocessing:

Fichier features.csv

Nous avons de données de type MarkDown1-5 avec tout valeur manquante est marquée d'un NA. Nous remplaçons ces valeurs NA par valeur 0 comme rien liées à démarques promotionnelles.

Données de manquant du CPI et Unemployment sont remplacés par ses moyen.

```
#fill markdown by 0
df_features$MarkDown1[is.na(df_features$MarkDown1)] <- 0
df_features$MarkDown2[is.na(df_features$MarkDown2)] <- 0
df_features$MarkDown3[is.na(df_features$MarkDown3)] <- 0
df_features$MarkDown4[is.na(df_features$MarkDown4)] <- 0
df_features$MarkDown5[is.na(df_features$MarkDown5)] <- 0

# fill CPI and Unemployement
df_features$CPI[is.na(df_features$CPI)] <- mean(df_features$CPI, na.rm = TRUE)
df_features$Unemployment[is.na(df_features$Unemployment)] <- mean(df_features$Unemployment, na.rm = TRUE)
```

Fichier stores.csv

Nous avons la colonne Type qui est catégorial variable. Nous le transformons sous forme numérique à la colonne StoreType.

Fichier train.csv/ test.csv :

Nous fusionnons fichier de stores et features avec fichier de train.csv

Après, nous transformons colonne Date sous type Date pour facile à l'utiliser. Nous créons des nouvelles colonnes pour avoir nouveau attribut pour forecasting. Comme

Days_of30 : la valeur d'une journée dans un année en comptant que chaque mois a 30 jours.

dayHoliday : la valeur d'une journée de Holiday dans un année en comptant que chaque mois a 30 jours. Autre journée sans Holiday égal à 0.

Year : l'année à partir colonne Date

Day : le jour à partir colonne Date

Month : le mois à partir colonne Date

II – Méthode d'évaluation :

Nous utilisons RMSE, la racine de l'erreur moyenne quadratique pour faire le choix de modèle principalement.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\epsilon_i)^2}$$

Autre indicateur est pour mieux comprendre et caractériser la qualité d'un modèle de prédiction.

III – Approche :

Nous pensons qu'il y a deux façons de forecasting :

- Première façon, nous considérons type de store et département de store sont des catégories. Alors, le modèle de forecasting va appliquer sur tout le magasin, toutes les données.
- Deuxième façon, nous faisons de filtre par Store et Département. C'est-à-dire, nous faisons de forecasting chaque magasin dans la façon indépendant. Le modèle de forecasting va appliquer sur un magasin unique identifié par son Département et numéro de Store.

La première façon donne le modèle plus générique et adapté au besoin de forecasting tout magasin. Mais par rapport la puissance de notre ordinateur, nous décidons à choisir ***deuxième approche*** pour avoir moins de temps d'attente d'entrainement des modèles et nous gardons la première façon après.

IV – Analyse - store 1 dans département 1 (fichier analysis_store1_dept1.R) :

Cette partie nous présentons notre travail avec le parti de train dans le fichier train.csv associé avec store et feature.

Nous choisissons le store numéro 1 au Département 1 dans le fichier de train.csv pour commencer. Pour diviser les données en une partie d'entrainement et une partie de test, nous choisissions que les données de 3 dernier mois sont pour le test *suivi la demande de prédire sur un horizon de 3 mois*. Le reste sont pour la partie d'entrainement.

Grace à ces commandes, nous savons que le dernier jour est à 26/10/2012 et le premier jour est à 02/05/2010. Donc, le test contient données le mois 8, 9 ,10 du années 2012.

```
#date max 2012-10-26 in train dataset
train[which.max(as.POSIXct(train$Date)), ]
#date min 2010-02-05 in train dataset
train[which.min(as.POSIXct(train$Date)), ]
```

```
train_ml<-dept_1_store_1 %>% filter( (dept_1_store_1$year < 2012) | (dept_1_store_1$year >= 2012 & dept_1_store_1$month < 8 ) )
test_ml <-dept_1_store_1 %>% filter( dept_1_store_1$year >= 2012 & dept_1_store_1$month >= 8  )
```

Les modèles de séries temporelles supposent que les données de séries temporelles donné en stationnaire signifie qu'il a la moyenne constante et la variance constante. Si les données n'est pas stationnaire, nous n'appliquons pas les modèles de séries temporelles. Nous utilisons le test de Dicky Fuller pour vérifier si la série est stationnaire ou non. Voici sa mise en œuvre.

```
adf.test(train$Weekly_Sales)
```

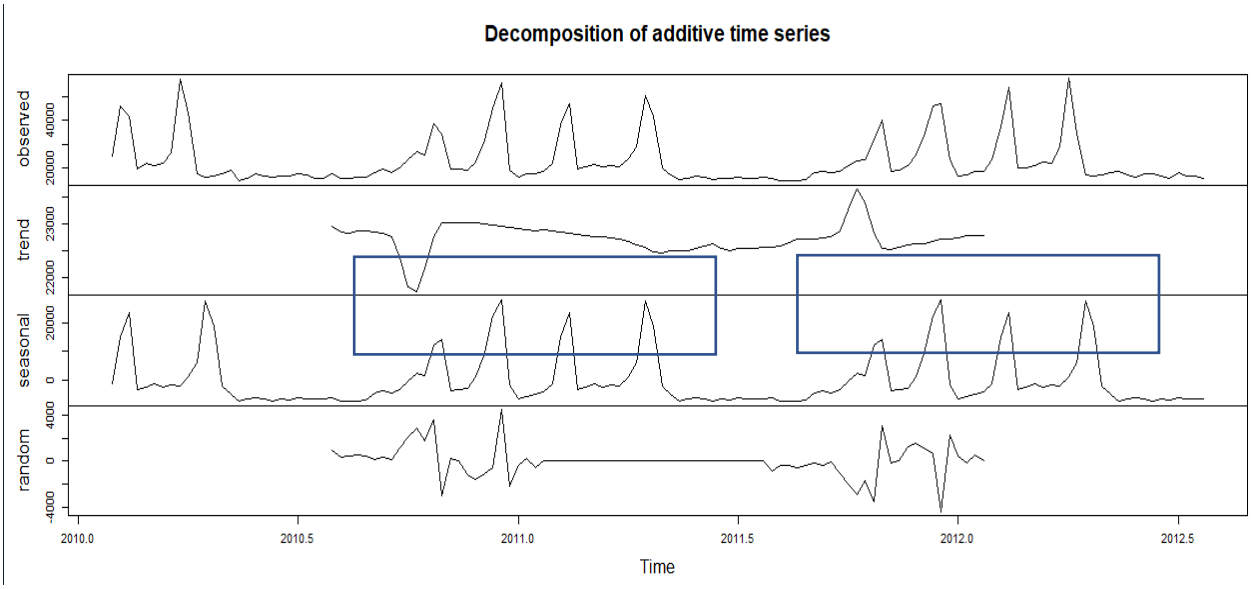
```
        Augmented Dickey-Fuller Test

data:  train$Weekly_Sales
Dickey-Fuller = -20.575, Lag order = 74, p-value = 0.01
alternative hypothesis: stationary

Warning message:
In adf.test(train$Weekly_Sales) : p-value smaller than printed p-value
```

Comme nous pouvons voir que notre p-value est inférieure à 0.05 et même inférieure à 0,01, nous pouvons donc dire avec une assez bonne confiance que nous pouvons rejeter le null hypotheis (unit-root, non-stationary). De plus, notre ADF est bien inférieur à notre valeur de confiance de 1%, nous avons donc une autre confirmation que nous pouvons rejeter l'hypothèse nulle parce que notre donnée est stationary.

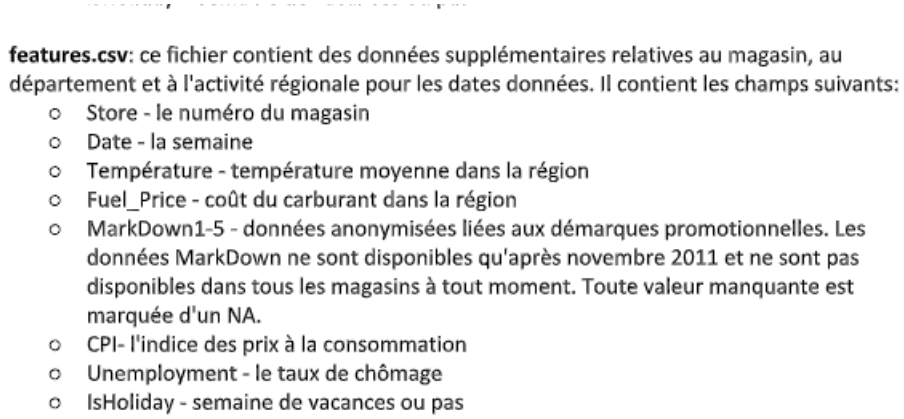
Nous décomposons notre donnée et nous trouvons qu'il y a de « seasonal » dans notre donnés avec le pattern particulier. Donc, il y a 2 remarques sur notre données, seasonal et stationary. Seasonal veut dire qu'il y a de cycle à court terme répétitif de la série. Ces 2 remarque nous aide à appliquer pour choisir le type d'algorithme plus adapté.



Des Holiday de notre dataset qui sont concentre sont de Christmas, Thanksgiving et les deux autre à février et septembre. Ça nous donne une vision que comment les vacance affect aux ventes.

	<chr>	<date>
1	holiday	2010-02-12
2	holiday	2010-09-10
3	holiday	2010-11-26
4	holiday	2010-12-31
5	holiday	2011-02-11
6	holiday	2011-09-09
7	holiday	2011-11-25
8	holiday	2011-12-30

Nous trouvons que le fichier de features.csv nous donne des infos données supplémentaires relatives au magasin, au département et l'activité régionale pour les dates données.



features.csv: ce fichier contient des données supplémentaires relatives au magasin, au département et à l'activité régionale pour les dates données. Il contient les champs suivants:

- Store - le numéro du magasin
- Date - la semaine
- Température - température moyenne dans la région
- Fuel_Price - coût du carburant dans la région
- MarkDown1-5 - données anonymisées liées aux démarques promotionnelles. Les données MarkDown ne sont disponibles qu'après novembre 2011 et ne sont pas disponibles dans tous les magasins à tout moment. Toute valeur manquante est marquée d'un NA.
- CPI- l'indice des prix à la consommation
- Unemployment - le taux de chômage
- IsHoliday - semaine de vacances ou pas

Mais ce type d'info, nous n'avons pas eu toujours pour faire forecasting. Par exemple, pour avoir le température moyenne de la région d'une journée, il faut passer la journée ou avoir une méthode de prédiction de température. L'info plus facile que nous avons toujours est la journée que nous voulons faire forecasting avec. Donc, nous voulons tester le modèle avec données plus détaillé avec tous ses infos supplémentaires (modèle nommé avec « _xreg », données nommées avec « _ml » dans le code) et aussi modèle avec info de journée (modèle nommé sans « _xreg », données nommées avec « _ts » dans le code).

Il y a quelque petite transformation pour adapter le données suivi le prototype de chaque algorithme aussi (post fixe « _ts » pour le prototype algorithme time series et post fixe « _ml » pour le prototype machine learning).

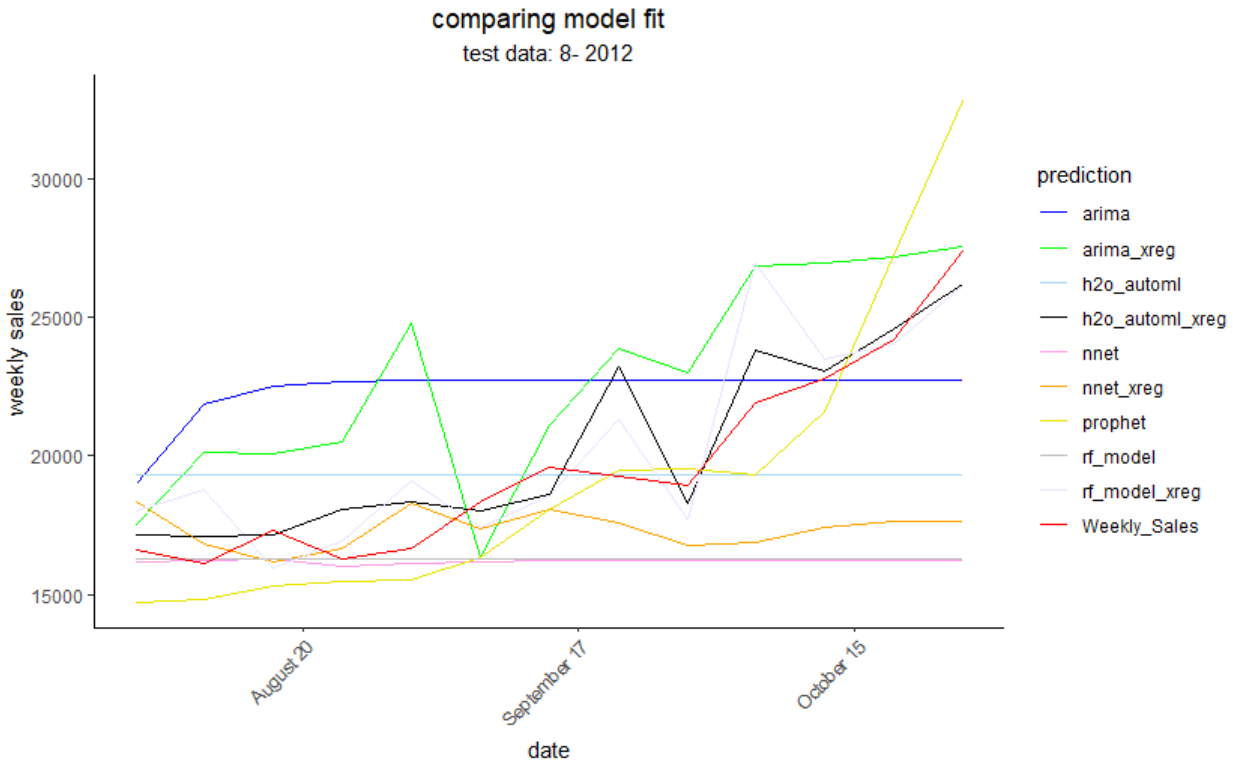
Grace à le choix d'un seul magasin, nous pouvons supprimer de catégorise sans changement par rapport le temps comme le numéro, la taille (« Size »), le type, le département de magasin.

Nous utilisons des méthode suivi la demande de tester les méthodes statistiques basées sur les séries temporelles (ARIMA, H-W,Prophet, TBATS….) , puis les méthodes de machine learning (RF, XBGoost, GBM ….) et un nouveau méthode de automatique généré l'algorithme automl du library H2O.

Dans ce travail, nous utilisons :

+ Neural Network Time Series Forecasts (nnetar)
+ H2O's AutoML
+ Prophet
+ ARIMA
+ Random Forest

Voici la comparaison entre d'algorithmes



Voici les paramètres pour comparaison

	nnet	nnet_xreg	arima_xreg	arima	prophet	h2o_automl
Mean Error	3449.75	2284.15	-3117.37	-2705.25	395.88	310.87
Root Mean Square Error	4783.54	4016.35	3955.49	4139.87	2229.55	3351.75
Mean Absolute Error	3463.76	2963.17	3424.06	3651.58	1824.64	2638.91
Mean Absolute Percent Error	0.16	0.13	0.18	0.20	0.09	0.13
Mean Percent Error	0.15	0.09	-0.17	-0.16	0.03	-0.01

	h2o_automl_xreg	rf_model	rf_model_xreg
Mean Error	-632.40	3384.89	-695.09
Root Mean Square Error	1516.82	4753.43	2014.29
Mean Absolute Error	1148.03	3406.97	1610.69
Mean Absolute Percent Error	0.06	0.15	0.08
Mean Percent Error	-0.04	0.15	-0.04

Nous trouvons que les deux modèle H2O's AutoML et random forest et prophet ont les moins RMSE.

Nous trouvons aussi que le modèle qui utiliser seul le critère de la journée est moins bon que de sa version avec tout le critère. Sauf Prophet est très bonne avec ces données de la journée et indication la journée de la vacance aussi.

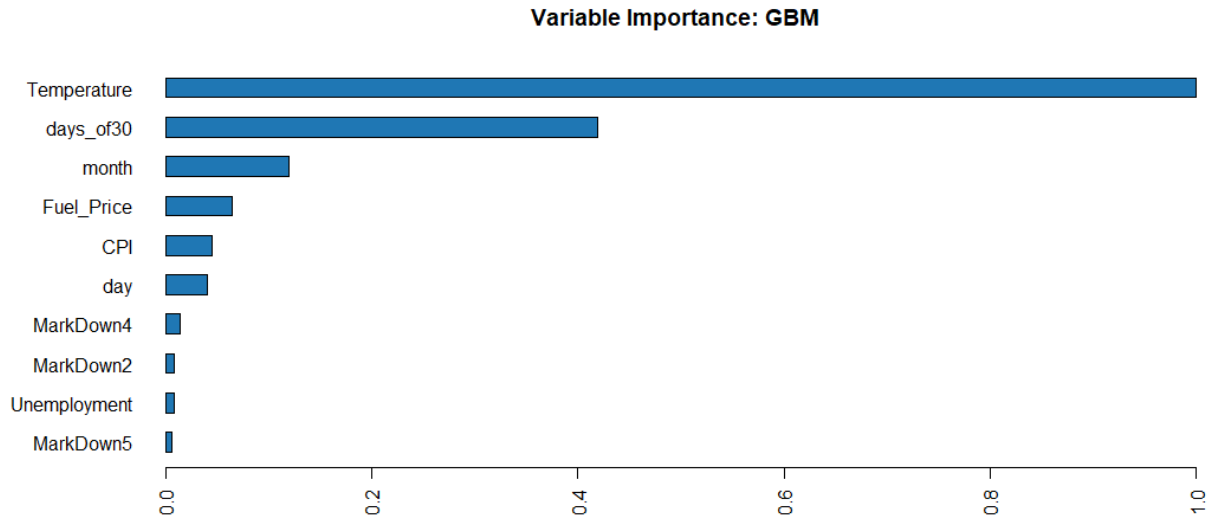
Le modèle H2O automl est le choix parmi le modèle XGBoost , GBM , DeepLearning((Fully-connected multi-layer artificial neural network), etc même si avec ses modele stacked ensemble(ensemble de modèles empilés).

Nous pouvons voir le résultat des modèles d'utiliser dans AUTOML H2O

```
                                       model_id mean_residual_deviance     rmse
1         GBM_grid__1_AutoML_20200208_105417_model_50               2300757 1516.825
2         GBM_grid__1_AutoML_20200208_105417_model_62               2505905 1583.005
3         GBM_grid__1_AutoML_20200208_105417_model_75               3104731 1762.025
4         GBM_grid__1_AutoML_20200208_105417_model_19               3189813 1786.005
5         GBM_grid__1_AutoML_20200208_105417_model_72               3231799 1797.720
6 StackedEnsemble_BestOfFamily_AutoML_20200208_105417               3666611 1914.840
      mse      mae      rmsle
1 2300757 1148.034 0.07513147
2 2505905 1303.011 0.08108449
3 3104731 1512.924 0.08349711
4 3189813 1569.271 0.08766762
5 3231799 1413.852 0.09576087
6 3666611 1464.771 0.08509736
```

Donc nous savons que notre modèle h2o_automl_xreg est un type de modèle GBM déjà bien régler hyperparamètres.

Nous aimons aussi voir le feature plus important qui affecte le forecasting.

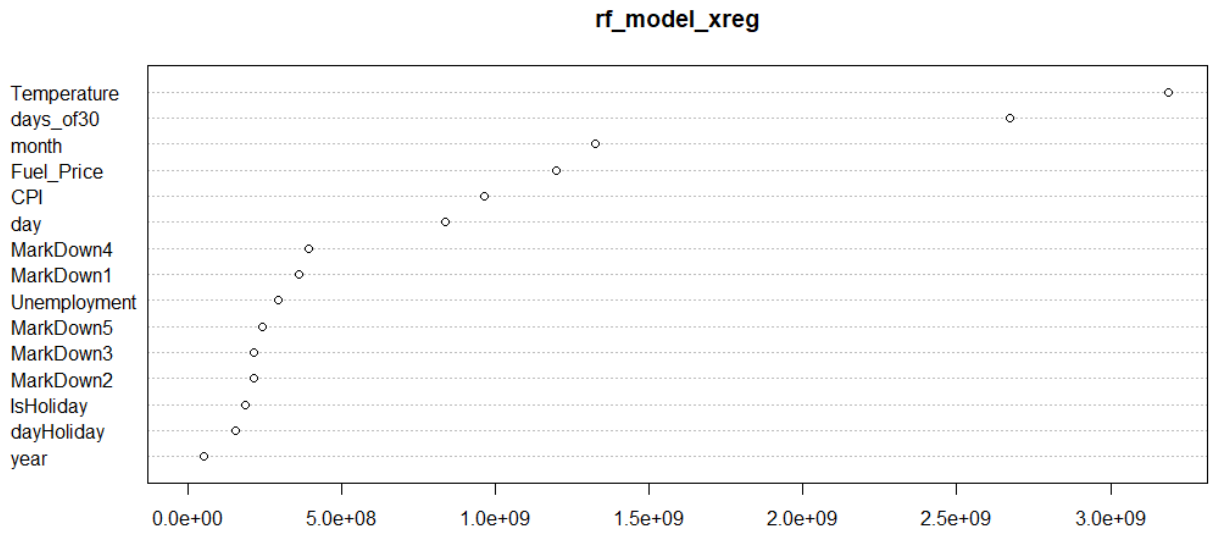


Nous pouvons voir que le temperature affect le plus sur le ventes. Avec le climate, et temperature different, nous allons avoir le besoisn different.

Et l'orde du journée de affect aussi (nous pouvons penser sur d'autre holiday qui n'est pas mentioné dans notre données) parce que les faeture IsHoliday et dayHoliday n'est pas apparu. Ou, nous pouvons penser sur le journée qui est près de Holiday serais un critere très bonne.

Le cout de carburant est aussi affecte le vente.

Avec d'autre modèle comme random forest nous pouvons voir le similarité avec H2O AUTOML.



V – Test (automl.R, randomforest.R, Prophet.R) :

Cette partie nous présentons notre travail avec le parti de test dans le fichier train.csv associé avec store et feature.

Pour choisir un algorithme de lancer le test, non seulement il a le meilleur RMSE mais aussi le temps de réalisation est important. Après quelques essais de minimiser le temps d'algorithme suivi parameters « max_runtime_secs », nous trouvons que l'algorithme auto-ml marche bien avec le max_runtime_secs plus que 120s. Donc, il faut compter 2 min pour une fois de lancer algorithme. Basé sur notre approche et il y a plus que 3000 magasins à prédire, nous décidons de n'utiliser pas Automl. Voici des images de notre justification

```
java.lang.NullPointerException
        at water.MRTask.dfork(MRTask.java:453)
        at water.MRTask.doAll(MRTask.java:390)
        at water.MRTask.doAll(MRTask.java:397)
        at hex.glm.GLMModel.predictScoreImpl(GLMModel.java:1687)
        at hex.Model.score(Model.java:1470)
        at hex.ensemble.StackedEnsembleModel.predictScoreImpl(StackedEnsembleModel.java:146)
        at hex.Model.score(Model.java:1470)
        at water.api.ModelMetricsHandler$1.compute2(ModelMetricsHandler.java:381)
        at water.H2O$H2OCountedCompleter.compute(H2O.java:1455)
        at jsr166y.CountedCompleter.exec(CountedCompleter.java:468)
        at jsr166y.ForkJoinTask.doExec(ForkJoinTask.java:263)
        at jsr166y.ForkJoinPool$WorkQueue.runTask(ForkJoinPool.java:974)
        at jsr166y.ForkJoinPool.runWorker(ForkJoinPool.java:1477)
        at jsr166y.ForkJoinWorkerThread.run(ForkJoinWorkerThread.java:104)

Error: java.lang.NullPointerException
```

Erreur comme le temps de lancer auto-ml est très limité (moins 60s)

```
09:41:47.972: AutoML:
09:41:53.983: Skipping
AutoML_20200209_094147
[1] "1 46"

09:43:03.339: AutoML:
09:43:09.349: Skipping
AutoML_20200209_094303
[1] "1 17"
```

Exemple de temps de chaque fois de lancer

D'autre problème que nous devons traiter est le cas de store dans une département n'est pas apparu dans notre dataset d'entrainement. Voyons l'exemple du magasin avec numéro de département associé qui est seul dans le dataset de test.

```
list_store_dept_test <- unique(final_test$Dept_Store)
list_store_dept_train <- unique(train$Dept_Store)

for (val in list_store_dept_test){
  if (!(val %in% list_store_dept_train))
  {
    print(val)

  }

}
```

```
+ }
[1] "10 99"
[1] "18 43"
[1] "24 43"
[1] "25 99"
[1] "34 39"
[1] "36 30"
[1] "37 29"
[1] "42 30"
[1] "45 39"
[1] "5 99"
[1] "9 99"
```

Pour résoudre ce problème, voici le code dans le fichier randomforest.R , nous créons un seuil minimal de record du magasin dans la base d'entrainement (ici est 10).

```
dept_store_train <- train %>% filter(Dept_Store == val)
#create a limit of record in training data
if (nrow(dept_store_train) > 10)
{
  #data preprocessing for prototype algorithm
  dept_store_test <- final_test_tmp %>% filter(Dept_Store == val )
  train_ml <- dept_store_train %>%  select(-Date,-Store,-Dept,-Type,-StoreType,-Size, -Dept_Store)
  test_ml <- dept_store_test %>% select(-Date,-Store,-Dept,-Type,-StoreType,-Size,-Dept_Store)
  #training model
  rf_model_xreg <-  randomForest( Weekly_Sales ~., data = train_ml, ntree=150, replace=TRUE, mtry=5)
  #runing prediction
  rf_pred_xreg <- predict(rf_model_xreg, test_ml)
  #save résultat
  dept_store_test$Predict <- rf_pred_xreg
  dept_store_test <- dept_store_test%>% select(Dept_Store,Date, Predict)
  csv_file$Predict <- replace(csv_file$Predict, (csv_file$Date %in% dept_store_test$Date) & (csv_file$Dept_Stor
  #save model
  name_file <- c("./model/type1",val,"final_model.rds")
  name_file <- paste(name_file, collapse="_")
  saveRDS(rf_model_xreg, name_file)
```

Si nous n'avons pas assez le seuil, nous faisons forecasting par département. Nous ajoutons info de la taille et le type de magasins pour faire forecasting. Une fois comme ça, nous avons plus de données de d'autre magasins mais avec la taille et le type diffèrent.

```
  saveRDS(rf_model_xreg, name_file)
}else{
  print(2)
  s <- strsplit(val, " ", fixed = TRUE)
  #get data by departement
  dept_tmp = as.numeric(s[[1]][2] )
  dept_store_train <- train[train$Dept == dept_tmp,]
  dept_store_test <- final_test_tmp %>% filter(Dept_Store == val )
  #not remove size and storetype
  train_ml <- dept_store_train %>% select(-Date,-Store,-Dept,-Type,-Dept_Store )
  test_ml <- dept_store_test %>% select(-Date,-Store,-Dept,-Type,-Dept_Store )
  #training model
  rf_model_xreg <-  randomForest( Weekly_Sales ~., data = train_ml, ntree=150, replace=TRUE, mtry=5)
  #runing prediction
  rf_pred_xreg <- predict(rf_model_xreg, test_ml)
  #save résultat
  dept_store_test$Predict <- rf_pred_xreg
  dept_store_test <- dept_store_test%>% select(Dept_Store,Date, Predict)
  csv_file$Predict <- replace(csv_file$Predict, (csv_file$Date %in% dept_store_test$Date) & (csv_file$Dept_Store %
  #save model
  name_file <- c("./model/type2",val,"final_model.rds")
  name_file <- paste(name_file, collapse="_")
  saveRDS(rf_model_xreg, name_file)
```

Avec cette solution, nous pouvons traiter le cas de données de magasin est très peu (<10 records de semaine approximante données dans 3 mois à un magasin).

Les algorithmes que nous réussissons à lancer prophet et random forest. Le tuning de random foret est suivi la donnée de store 1 dans département 1 avec sa performance qui donne le moins RMSE. Donc, le résultat que nous pensons plus prometteuse est à l'algorithme prophet avec tous les paramètre de configuration chez FaceBook .

Nous gaspillons le temps sur auto-ml mais le besoin de 120 s pour une fois de lancement est difficile à faire (déjà essai avec le temps diffèrent).

VI – Perspective :

Voici quelques idées que nous pensons à ajouter dans l'avenir :

- Fais modèle forecasting par données avec info département similaire
- L'effet de Vacance est plus interéssé à continuer, nous pouvons ajouter plus info de vacance non seulement 4 vacances dans le dataset ou une méthode de pénaliser d'erreur dans la semaine de vacance pour valoriser plus la journée de vacance.
- Appliquer le méthode dimension réduction pour voir le component principal et faire forecasting avec eux
- Comme auto-ml, faire un modèle de boosting ensemble d'algorithme de notre algorithme
- Tuning de algorithm de random forest pour chaque magasin avec la base d'entrainement.