I’m coding R and python by Jupyter notebook, support by Kaggle notebook.

1. Data overview

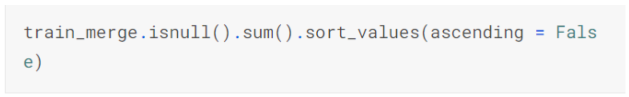
1.1 Data description

To make the most accurate regression for the sale price, I'm using all the datasets: W21_store_info.csv, W21_test.csv and W21_train.csv. W21_train.csv has training information for the store, like store, day of week. W21_test.csv has testing information for the store, exclude sales and customers. W21_store_info has supplemental information for the details of the store, according to different store ID. It includes store type, competition distance and promo interval.

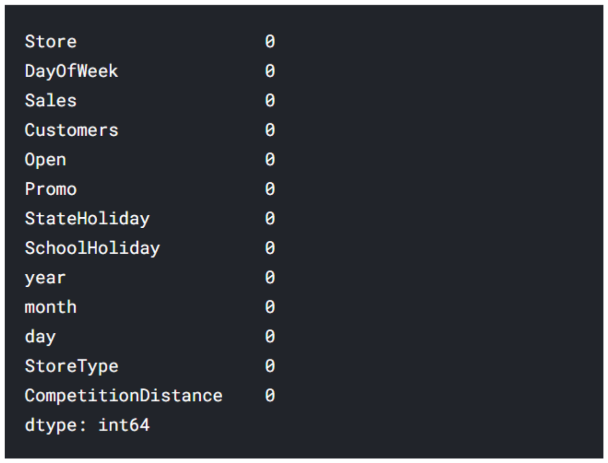
Since there is a common column of store (store ID), I combine W21_train.csv and W21_store_info.csv; W21_test.csv and W21_store_info.csv by the common column store.

At the start, we know most of the columns are numerical value, only the Date looks like a date object. So, I separated the date column to these columns: year, month, date, according to the date column's year, month, date.

1.2 statistics & visualization



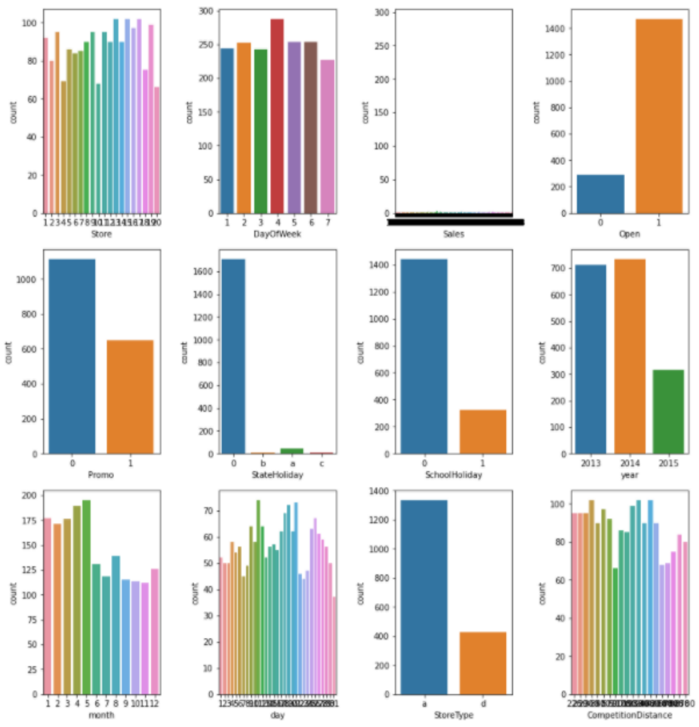
```
train_merge.isnull().sum().sort_values(ascending = Fals
e)
```



```
Store                  0
DayOfWeek              0
Sales                  0
Customers              0
Open                   0
Promo                  0
StateHoliday           0
SchoolHoliday          0
year                   0
month                  0
day                    0
StoreType              0
CompetitionDistance    0
dtype: int64
```

There are no null values in the dataset. For now, Date and StoreType are cateogrical type, the others are numeric type. Then, randomly sample 10% of the dataset for visualization, to lower the computational cost.

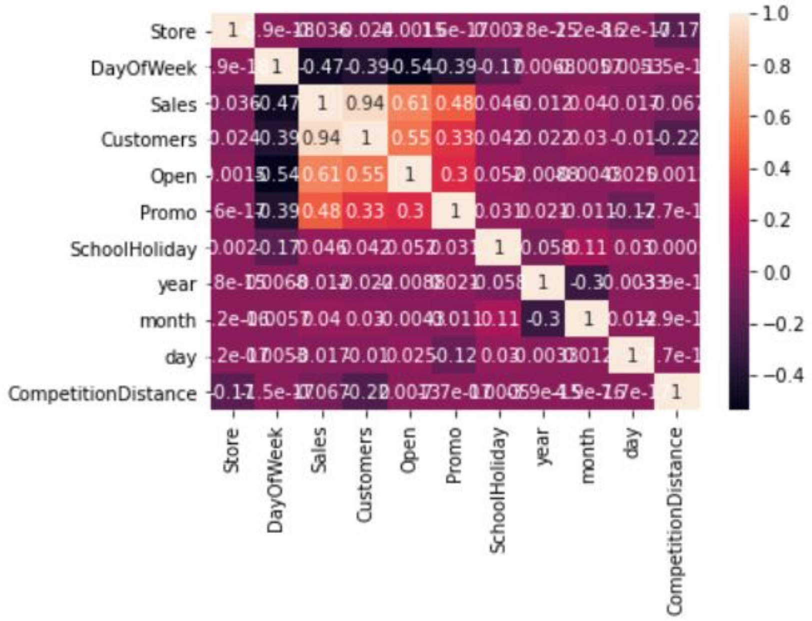
Histogram for each of the covariates:



Sales, customers and Competition distance are numerical continuous data. Date of week, open, promo, School holiday are numerical discrete data. The others are categorical Also, States holiday is sparse data, with most count as 0. Delete it.

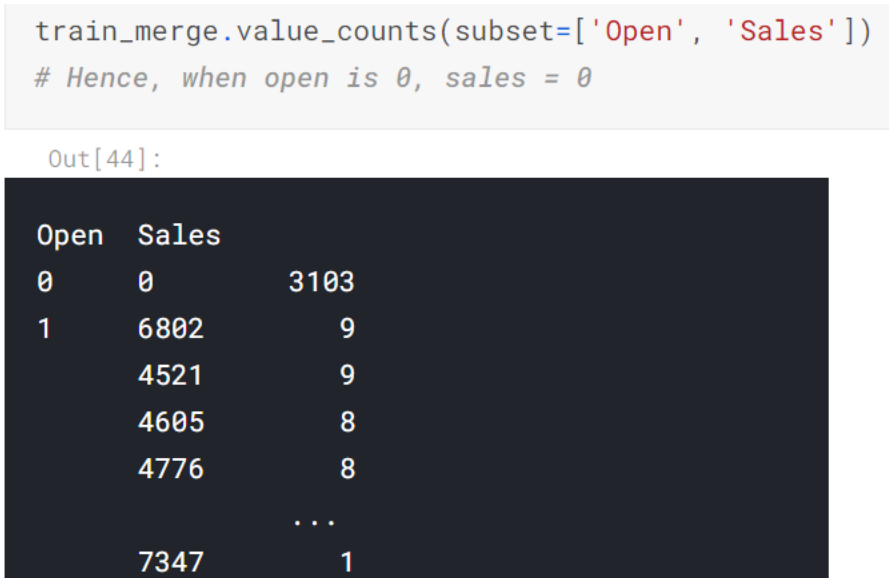
To reduce multicollinearity, delete one of the pairs of columns that has correlation above 0.95

Correlation matrix:



From the observation, Sales and customer has a very high correlation, and it’s < 0.95. Since this column is not in testing dataset, we can use other columns predict customers first.

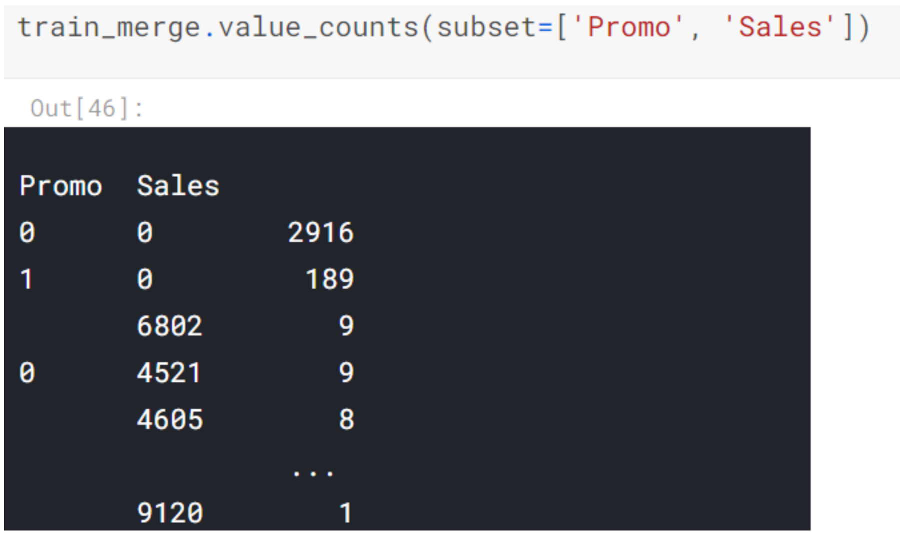
Also, from the heat map, open and promo has high correlation with sales. Let's find the relationship. Since open and promo are values of 0 and 1, check if open/protmo is 0, sales is 0.



```
train_merge.value_counts(subset=['Open', 'Sales'])
# Hence, when open is 0, sales = 0
```

```
Out[44]:
Open  Sales
0     0        3103
1     6802        9
      4521        9
      4605        8
      4776        8
                ...
      7347        1
```

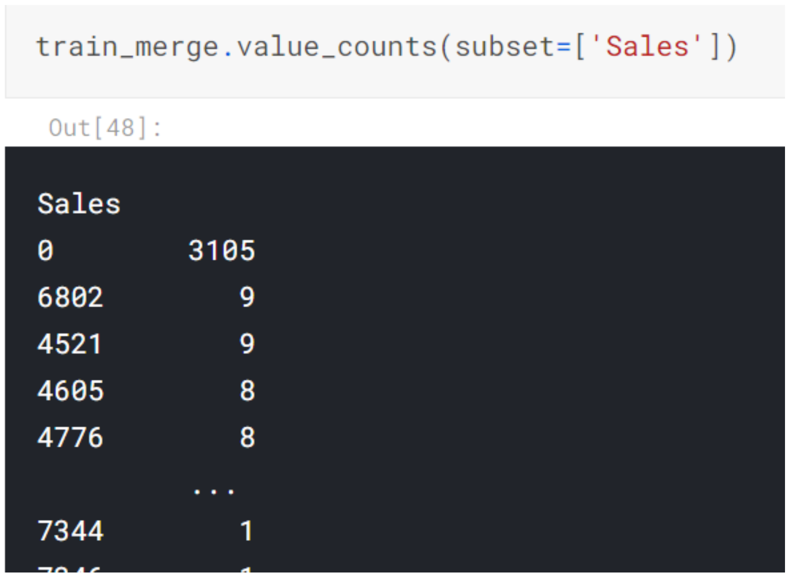
It shows when open is 0, sales are 0. The other values are evenly distributed.



```
train_merge.value_counts(subset=['Promo', 'Sales'])
```

```
Out[46]:
Promo  Sales
0      0        2916
1      0         189
       6802        9
0      4521        9
       4605        8
                ...
       9120        1
```

It shows there is 2916 rows, when promo is 0, sales is 0. Other values are evenly distributed.

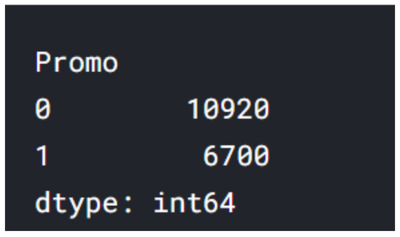


```
train_merge.value_counts(subset=['Sales'])
```

```
Out[48]:
```

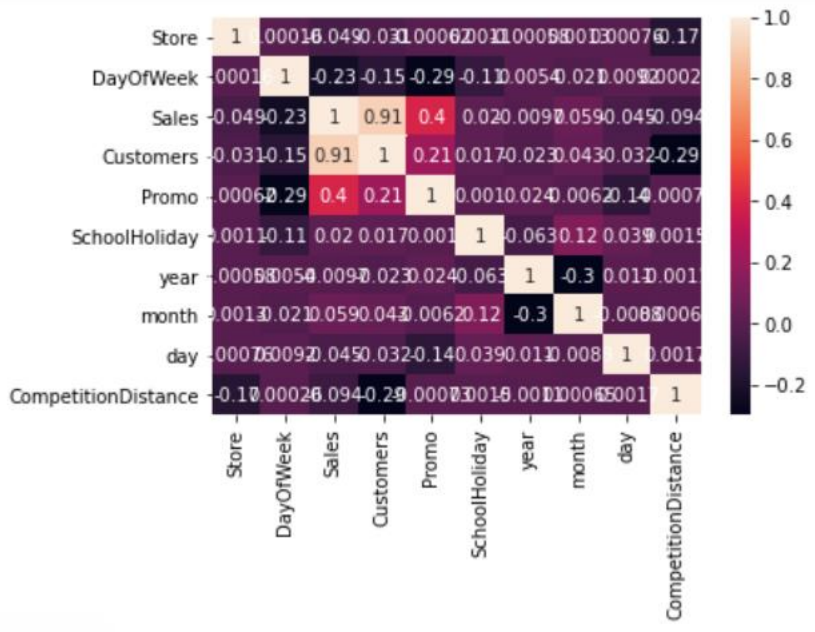
```
Sales
0       3105
6802       9
4521       9
4605       8
4776       8
        ...
7344       1
```

Totally sales has 3105 rows, there is 3103 rows when open is 0, sales are 0. Since there is a very high percentage for open is 0, sales are 0 (close to 100%). So, predict the sale price as 0, when the open is 0.

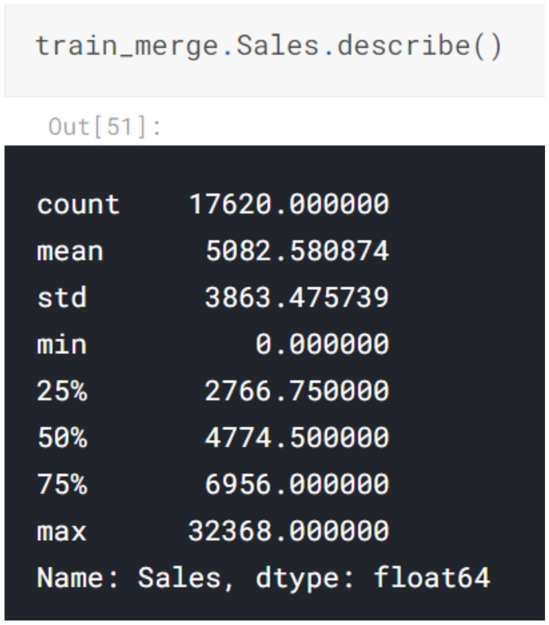


```
Promo
0       10920
1        6700
dtype: int64
```

In Promo column, the percentage of 2916/10920 when promo is 0, sales is 0, just leave it here.



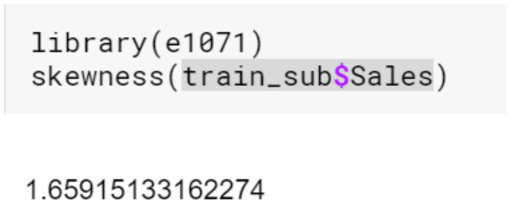
Here is the heat map when open is not 0.



```
train_merge.Sales.describe()
```

```
Out[51]:
count    17620.000000
mean      5082.580874
std       3863.475739
min          0.000000
25%       2766.750000
50%       4774.500000
75%       6956.000000
max      32368.000000
Name: Sales, dtype: float64
```

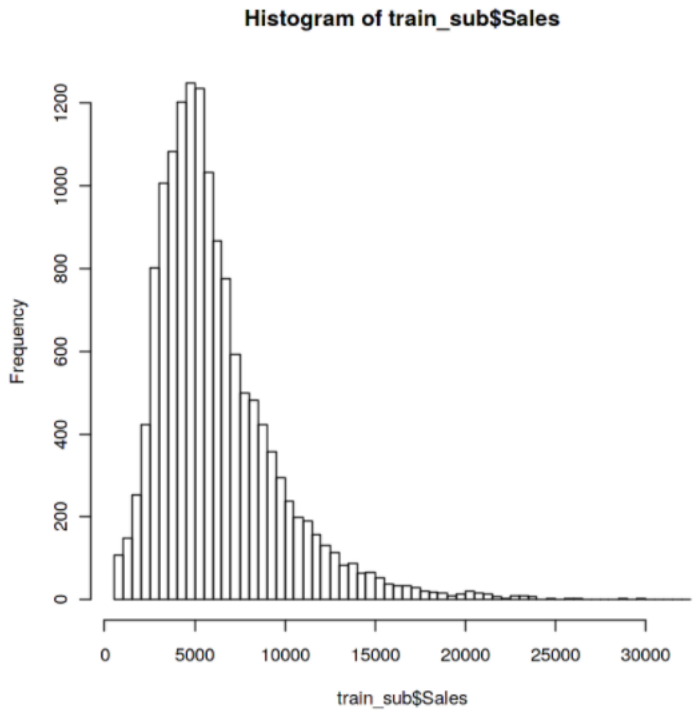
Here is the basic statistics for the target. Looks like the data’s skewness is not good. We can try log scale to reduce the skewness.



```
library(e1071)
skewness(train_sub$Sales)
```

```
1.65915133162274
```

We can see the skewness is 1.659, which is not good. The range(-0.5,0.5) is acceptable for the skewness.

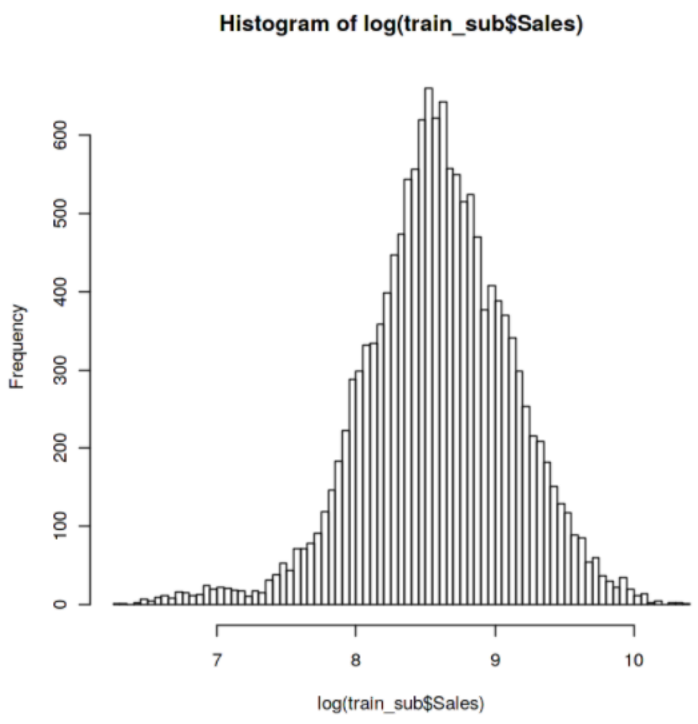


Distribution for sales

```
skewness(log(train_sub$Sales))
```

```
-0.321170665678229
```

It looks like log scale is better.



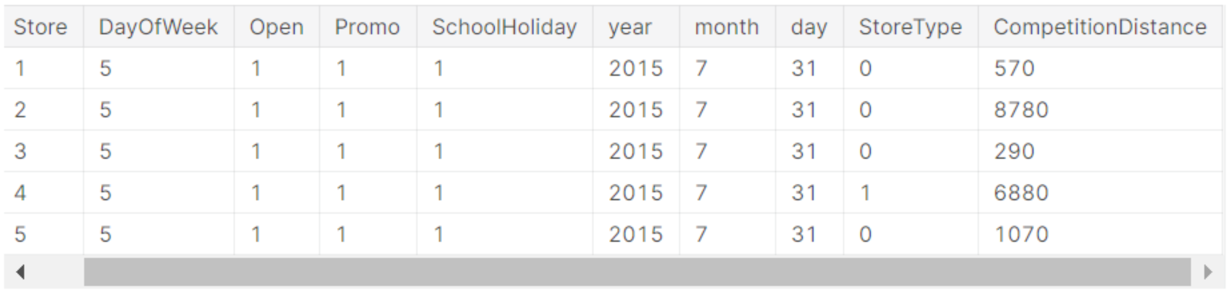
The shape looks better, normally distributed for the sales column.

So, we can try to use the log scale, or other scale function on the target variable Sales.

1.3 Data transformation

As I mentioned before, I transformed the data from data object to three columns. Other than that, I made the transformation for categorical variables. For the column StoreType, it is a column with two variables, two letters. Since sometimes some regression methods cannot directly works with strings, I use label encoder in python, change it to value 0 or 1.

Current data frame for training data:



Store	DayOfWeek	Open	Promo	SchoolHoliday	year	month	day	StoreType	CompetitionDistance
1	5	1	1	1	2015	7	31	0	570
2	5	1	1	1	2015	7	31	0	8780
3	5	1	1	1	2015	7	31	0	290
4	5	1	1	1	2015	7	31	1	6880
5	5	1	1	1	2015	7	31	0	1070

1.4 short conclusion:

Now I have a cleaned dataset to use, also explore some patterns in the data above, from the histogram, correlation map, statistical summary, skewness. Then, I need to make feature selection of the models.

2. Feature selection

Select meaningful features is important for most of the regression models. It's a way to reduce the number of features and complexity of the model. By removing the meaningless features, our model performance might be higher. But now, we only have 10 variables. My idea is:

Use these variables create more variables by using combination of each two variables, factor, and select important features from these.

2.1 Create more variables

I use factor() and combinations: Sales ~(factor(Store) + factor(DayOfWeek) + Customers + Promo + SchoolHoliday + factor(year) + factor(month) + day + StoreType + CompetitionDistance)^2 to generate as many variable as I can. (for example, in models, y~(a+b+c)^2 = y~a*b + b*c+ a*c)

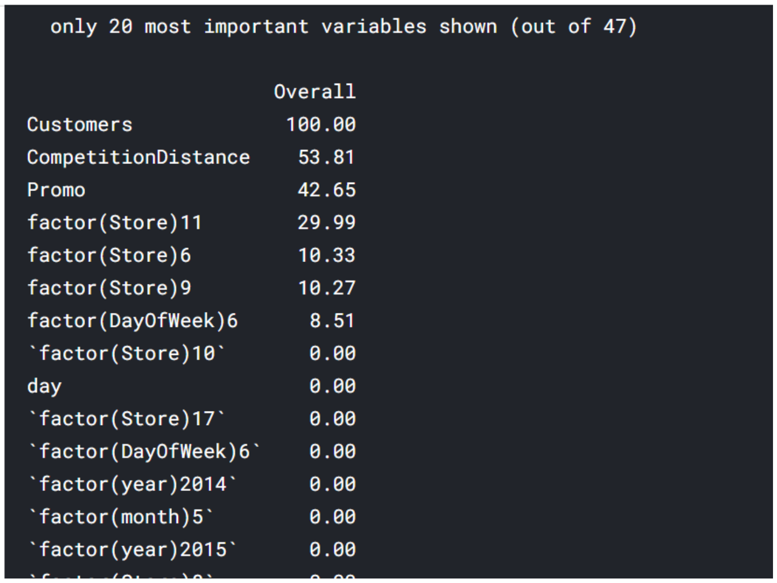
Factor and interaction are important because: factor() Compares each level of a variable to the reference level, which will let the each level of the variable has the same prediction power. Interaction effect combines two independent variables and create a new meaningful variable.

Then select important important features by following ways:

2.2 Variable importance

By using caret package, I'm selecting variables by variable importance, using a tree method 'rpart'. RPART implement the Recursive Partitioning And Regression Tree, it recursively creates a decision tree that strives to correctly classify menbers of the population by splitting it into sub-populations, based on several dichotomous independent variables. [1]

I wrote the code to select several top important features:

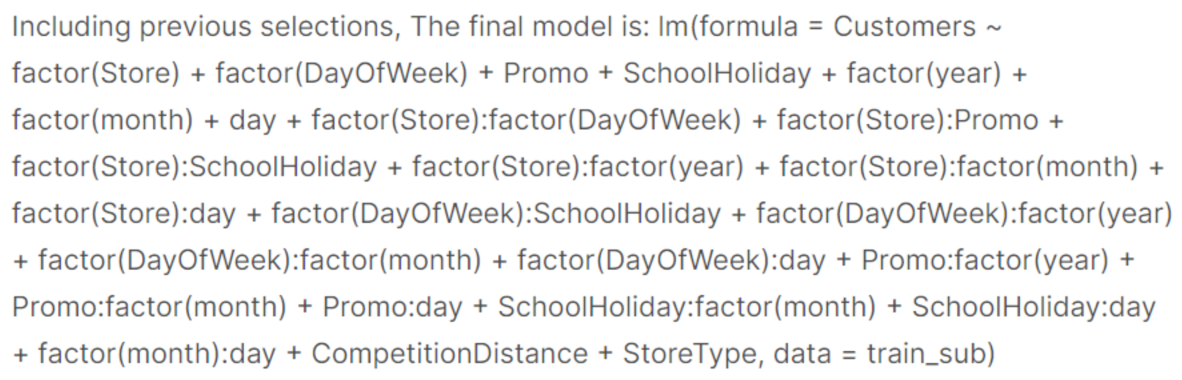

```
  only 20 most important variables shown (out of 47)

                     Overall
Customers             100.00
CompetitionDistance    53.81
Promo                  42.65
factor(Store)11        29.99
factor(Store)6         10.33
factor(Store)9         10.27
factor(DayOfWeek)6      8.51
`factor(Store)10`       0.00
day                     0.00
`factor(Store)17`       0.00
`factor(DayOfWeek)6`    0.00
`factor(year)2014`      0.00
`factor(month)5`        0.00
`factor(year)2015`      0.00
```

So, by this method, It shows that Customers, CompetitionDistance, Promo, factor(Store), factor(DayOfWeek) should be selected.

2.3 Stepwise feature selection

Stepwise feature selection was developed based on linear regression. This method uses a sequence of steps, let feature to enter or leave the regression model, one variable at a time. When the model needs to leave the model, it should have p value > 0.15. When the model needs to enter the model, it should have p value < 0.15.

By this methods, I pass through the variables that I generated in 2.1 by step() function. After generates the result, combine the result from 2.2, if the function in 2.3 not select variables from the result in 2.2. Here is the solution:



```
Including previous selections, The final model is: lm(formula = Customers ~
factor(Store) + factor(DayOfWeek) + Promo + SchoolHoliday + factor(year) +
factor(month) + day + factor(Store):factor(DayOfWeek) + factor(Store):Promo +
factor(Store):SchoolHoliday + factor(Store):factor(year) + factor(Store):factor(month) +
factor(Store):day + factor(DayOfWeek):SchoolHoliday + factor(DayOfWeek):factor(year)
+ factor(DayOfWeek):factor(month) + factor(DayOfWeek):day + Promo:factor(year) +
Promo:factor(month) + Promo:day + SchoolHoliday:factor(month) + SchoolHoliday:day
+ factor(month):day + CompetitionDistance + StoreType, data = train_sub)
```

3. Simple prediction

In this step, I randomly split the training data set into 75% training, 25% testing to test different models and use RMSPE score. I'm trying some models to predict the result, to find the best model among all the choices.

In step 2, feature selection might not be helpful for tree methods since tree method itself can select important variables. But for other linear methods, boosting methods, it is still worth to use selected variables to do the prediction. I'm discussing each models first, then show the result table at last.

3.1 linear regression

For linear regression, we try to fit a linear function of features to minimizes the sum of squared residuals. It's simple to implement and easier to interpret the output coefficients. But linear regression assumes a linear relationship between dependent and independent variables, we are not sure if the data satisfy the assumption.rigid regression

3.2 Rigid regression

Rigid regression made some improvements from linear regression. It solves the problem in linear regression, the regression coefficients is not unique (when p >n, p is X's full column rank – 1). The ridge regression has a tuning parameter λ ∈ [0,∞], which controls the trade-off between bias and variance. I choose λ by its smallest cv error.

3.3 Lasso regression

Unlike ridge regression, the lasso penalizes the l1 norm instead of l2 norm of the regression coefficient. For both rigid and lasso, they can avoid overfitting. They could be applied even when number of features is larger than amount of data.

3.4 random forest

Random forest is based on the bagging algorithm and uses an ensemble learning technique. It grows the decision tree to full size and combines the output of all the trees. So, it can reduce the overfitting problem and minimize the variance. I use hyper parameter method to choose parameter mtr, number of variables available for splitting at each tree node.

3.5 Boosting

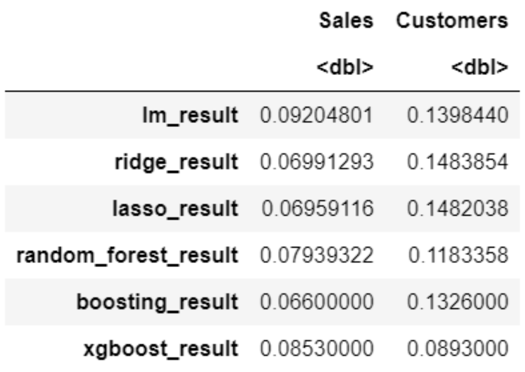
Boosting is an iterative technique that adjusts the weight of an observation based on the last classification. If an observation is classified incorrectly, it tries to increase the weight of this observation.

3.6 XGB boost

XGBoost is an implementation of gradient boosted decision trees designed for speed and performance. XGBoost library implements the gradient boosting decision tree algorithm. Gradient boosting is a type of machine learning boosting. It relies on the intuition that the best possible next model, when combined with previous models, minimizes the overall prediction error. [2]

3.7 Result

The results for the above models are:

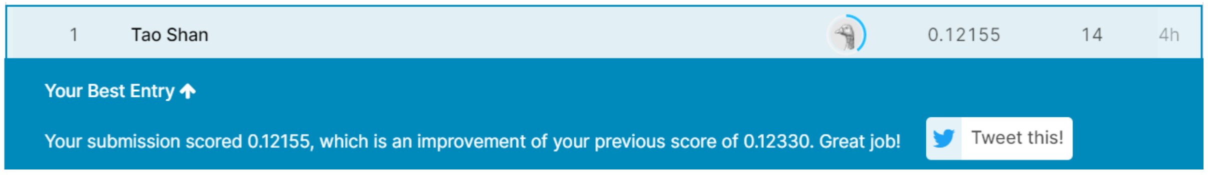


	Sales	Customers
	<dbl>	<dbl>
lm_result	0.09204801	0.1398440
ridge_result	0.06991293	0.1483854
lasso_result	0.06959116	0.1482038
random_forest_result	0.07939322	0.1183358
boosting_result	0.06600000	0.1326000
xgboost_result	0.08530000	0.0893000

The first column is the prediction performance to predict variable Sales. The second column is the prediction performance to predict variable Customers. (by training data, random sample 75% training data and 25% testing data, set.seed = 123)

By the table above, it is reasonable choose ridge regression, lasso regression, random forest, boosting, XGboost for sales column, XGboost for customers column.

Such as using xgboost for both sales and customers, the result is following:


1 Tao Shan 0.12155 14 4h

Your Best Entry

Your submission scored 0.12155, which is an improvement of your previous score of 0.12330. Great job!

3.8 Summary

By discussing with above models, I will try to use ridge regression, lasso regression, random forest, boosting, XGboost for sales column, XGboost for customers column. Since the combinations has the best accuracies.

4. Predictor Choice

linear_regression_0316.csv 0.16056

By these selected variables in step 2, and use simple linear regression with default setting, initial score is about 0.161. Then I will choose the best predictors for our selected models. I'm trying to decide which parameter is the best. For some of the predictor that allows hyper pruning parameters, I'm choosing some of the parameters for hyper pruning parameters to get an optimal model.

For the following method, the first one is predictor for Sales, the second one is predictor for Customers. I'm only discuss XGBoost when predict customers, and it is in part 4.4. In other parts, I'm discussing methods when predict sales.

4.1 Lasso + XGB (final_result.ipynb)

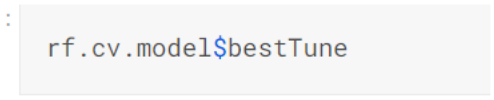
Name	Submitted	Wait time	Execution time	Score
LassoXGB_0321.csv	just now	1 seconds	0 seconds	0.13669

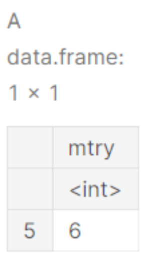
For Lasso regression, I was testing performance for this model when using first standard deviation rule or smallest cv error as lambda parameter. When testing lasso regression's RMSPE score in step 3, the smallest cv error's performance is always better than first standard deviation rule. So, I choose lambda by smallest cv error. And the result is 0.13669. As a linear method, it's much better than linear regression.

4.2 Random forest + XGB (final_result.ipynb)

Name	Submitted	Wait time	Execution time	Score
RFXGB_0321.csv	just now	1 seconds	0 seconds	0.12487

In random forest, I'm doing hyper parameter pruning for mtry: number of variables randomly sampled as candidates at each split.


```
rf.cv.model$bestTune
```


A
data.frame:
1 × 1

	mtry
	<int>
5	6

By the result, the best parameter for mtry is 6. By passing this parameter in random forest, the result is 0.12487, which is better than linear methods.

4.3 Boosting + XGB (final_result2.ipynb)

Name	Submitted	Wait time	Execution time	Score
BoostingXGB_0322.csv	just now	1 seconds	0 seconds	0.12553

I'm making hyperparameter pruning for boosting:

Interaction.depth: a number of splits it has to perform on a tree. My choice: c(2,3,4,5)

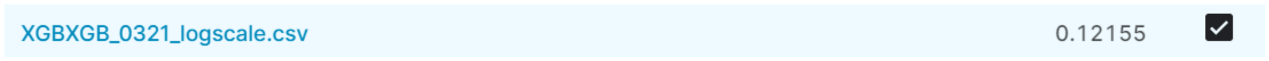
n.trees: the total number of tree to fit. My choice: c(200,400,600,800,1000)

shrinkage: learning rate or step-size reduction. My choice: c(0.1,0.05)

The best parameter is:

So, by using the best parameter, the final score is 0.12553

4.4 XGB + XGB (final_decision_xgb_20210321.ipynb)



XGBXGB_0321_logscale.csv	0.12155	✓

XGBoost has these parameters: [3]

eta: learning rate. Eta shrinks the feature weights to make the boosting process more conservative. My choice: c(0.01,0.05,0.1)

Gamma: Minimum loss reduction required to make a further partition on a leaf node of the tree. My pruning choice: c(0,0.01, 0.05, 0.1, 0.5, 0.7, 0.9, 1.0)

Max_depth: max depth of the tree. Increase this value will let the model become more complex My choice: c(2,4,6)

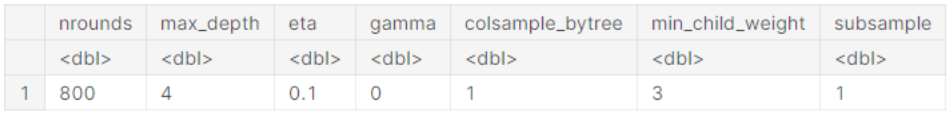
Min_child_weight: Minimum sum of instance weight (hessian) needed in a child. The larger the value is, the more conservative the algorithm will be. My choice: c(1,2,3)

Colsample_bytree: subsample ratio of columns when constructing each tree

My choice: c(0.4,0.6,0.8,1)

Nrounds: number of rounds for boosting step. My choice: c(50,100,400,800)

Here is the hyperparameter pruning result:

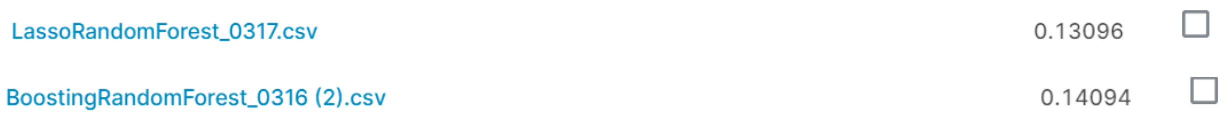


	nrounds	max_depth	eta	gamma	colsample_bytree	min_child_weight	subsample
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1	800	4	0.1	0	1	3	1

For other parameters, the tree method is the tree construction algorithm used in XGBoost. Like auto method, it uses a heuristic to choose the fastest method. For a large dataset, 'approx' is a good parameter for chosen. Approx parameter approximate greedy algorithm using quantile sketch and gradient histogram. Booster is specifying which booster to use, gbtree or gblinear. Gblinear uses linear regression with l1&l2 shrinkage, and gbtree uses a regression tree as a week learner.

By log scaler and hyper pruning parameter with these parameters, I get the optimal score of RMSPE 0.12155.

4.4 Other choices:



LassoRandomForest_0317.csv	0.13096	☐
BoostingRandomForest_0316 (2).csv	0.14094	☐

The result for Lasso + Random forest is 0.13, which is better than Lasso + XGBoost.

The result for Boosting + Random Forest is 0.14, which is worse than Boosting + XGBoost.

It seems like Random forest is another good predictor to predict Customers.

5. Summary:

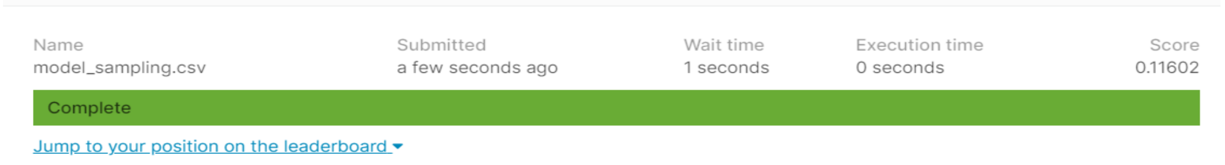
In summary, by data exploration, data preprocessing, feature selection, and test different models, the best solution is XGB + XGB and the score is 0.12155. (temporarily 1st place of this course).

In data exploration, I found that when open is 0, sales should be predicted as 0, and it's reasonable to use log scale for the target variable. Customer is reasonable to predict first in testing set, since it has high correlation with target variable. Also, for feature selection, I selected variables by feature importance and stepwise feature selection. Then, I test different model's performance in the training dataset and compare them to select the best combination. At last, by hyper parameter pruning, select parameter for each model with the best performance.

I did not use PCA in this dataset since the number of attributes is small. The other ways that could improve the model is:

1. Model sampling

   Use several best model's results, and take the average of them, or add the result together by weighting. By combining the result of 25% of random forest, 25% of boosting, 25% of XGB boosting and 25% of Lasso, the result is:



Name	Submitted	Wait time	Execution time	Score
model_sampling.csv	a few seconds ago	1 seconds	0 seconds	0.11602

Complete

Jump to your position on the leaderboard

   It can improve model performance because for RMSPE, sometimes we overestimate or underestimate the score, the method can help us reduce the error. For example, if the actual price is 6000, we made predictions of 5900, 5950, 6050, 6100 (these numbers have similar error rate), the average will be 6000 which has no error. Any single models will contain some of the error in this case (when the prediction result is all higher or lower than actual, it will not help the result, but also does not make the result worse).   So that is why sometimes single model's performance is worse than combine the result together, though this single model is the best among all models.

2. Cross validation

   Cross validation can reduce overfitting and selection bias. So a possible way is: Spilt the training dataset to 5 folders. In each iteration, combine 4 folders as training data, build the model then predict the result. Totally we have 5 iterations, add the result and divide by 5. (Some code is wrote in python [5])

3. Neural network

   In neural network, we want to create a computational system that could solve problems like a human brain. Neural network can learn and model the relationships between inputs and outputs that are nonlinear and complex; make generalizations and inferences; reveal hidden relationships, patterns and predictions; and model highly volatile data. [4] Hence, there is what we can try:

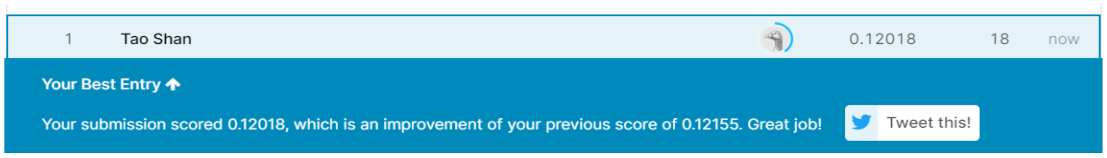
   Python: MLP Regressor in sklearn, CNN, RNN, transfer learning in Keras package

   R: Package neuralnet

4. Delete Customer?

   Customer column has high correlation with Sales, and it's valuable to predict it first, since test dataset does not have this column. However, sometimes it may increase bias, the accuracy to predict customers cannot be 100% correct.

   When I try XGboost and delete the customer column, I get the result higher than above methods. This might happen because the accuracy of predict customers. There is RMPSE about 0.08 when use XGB regression predict customers by 75% of training dataset (in part 3.7). So, we can try to delete this column and see the response answer.


1 Tao Shan 0.12018 18 now

Your Best Entry

Your submission scored 0.12018, which is an improvement of your previous score of 0.12155. Great job!

Reference:

[1]
https://en.wikipedia.org/wiki/Recursive_partitioning#:~:text=Recursive%20partitioning%20is%20a%20statistical,on%20several%20dichotomous%20independent%20variables.

[2]

https://www.displayr.com/gradient-boosting-the-coolest-kid-on-the-machine-learning-block/

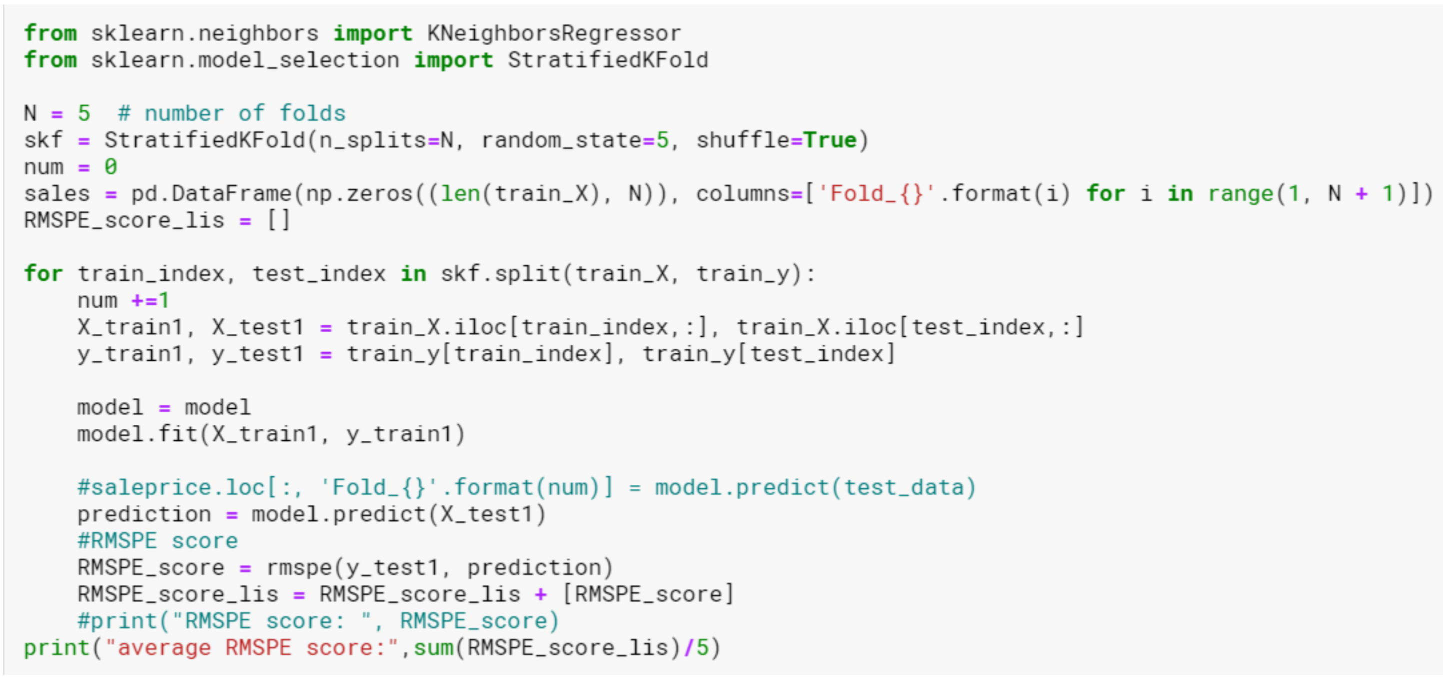
[3]

https://xgboost.readthedocs.io/en/latest/parameter.html

[4]

https://www.sas.com/en_ca/insights/analytics/neural-networks.html#:~:text=What%20they%20are%20and%20why,time%20%E2%80%93%20continuously%20learn%20and%20improve.

[5]


```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import StratifiedKFold

N = 5  # number of folds
skf = StratifiedKFold(n_splits=N, random_state=5, shuffle=True)
num = 0
sales = pd.DataFrame(np.zeros((len(train_X), N)), columns=['Fold_{}'.format(i) for i in range(1, N + 1)])
RMSPE_score_lis = []

for train_index, test_index in skf.split(train_X, train_y):
    num +=1
    X_train1, X_test1 = train_X.iloc[train_index,:], train_X.iloc[test_index,:]
    y_train1, y_test1 = train_y[train_index], train_y[test_index]

    model = model
    model.fit(X_train1, y_train1)

    #saleprice.loc[:, 'Fold_{}'.format(num)] = model.predict(test_data)
    prediction = model.predict(X_test1)
    #RMSPE score
    RMSPE_score = rmspe(y_test1, prediction)
    RMSPE_score_lis = RMSPE_score_lis + [RMSPE_score]
    #print("RMSPE score: ", RMSPE_score)
print("average RMSPE score:",sum(RMSPE_score_lis)/5)
```