SALE PREDICTION REPORT

Writer: Shifan Huang (Teammate: Pink)

Introduction:

In this report, we try to forecast sales with time-series using 3 supervised machine learning models: Ridge, Gradient Boosting and Random Forest. After testing MSE as well as accuracy score of each model, we decide to use the Random Forest model to do the prediction for future sales.

Business Problem:

We need to predict sales for the thousands of product families sold at Favorita stores located in Ecuador so that we can get to know economic trend in the future.

Data Description:

We have several files for data including store data, oil price, holiday, time, promotion information and product information that we need to merge and get the effective variables we need for our prediction.

Data Preparation:

We merged all the files and get the data we'll need for our analysis. This is how our raw data looks like:

	id	date	store_nbr	family	sales	onpromotion	city	state	type_x	cluster	transactions	type_y	locale	locale_name	description	transferred	dcoilwtico
0	73062	2013-02-11	1	AUTOMOTIVE	0.0	0	Quito	Pichincha	D	13	396	Holiday	National	Ecuador	Carnaval	False	97.01
1	73063	2013-02-11	1	BABY CARE	0.0	0	Quito	Pichincha	D	13	396	Holiday	National	Ecuador	Carnaval	False	97.01
2	73064	2013-02-11	1	BEAUTY	0.0	0	Quito	Pichincha	D	13	396	Holiday	National	Ecuador	Carnaval	False	97.01
3	73065	2013-02-11	1	BEVERAGES	172.0	0	Quito	Pichincha	D	13	396	Holiday	National	Ecuador	Carnaval	False	97.01
4	73066	2013-02-11	1	BOOKS	0.0	0	Quito	Pichincha	D	13	396	Holiday	National	Ecuador	Carnaval	False	97.01

We did several steps to clean our data:

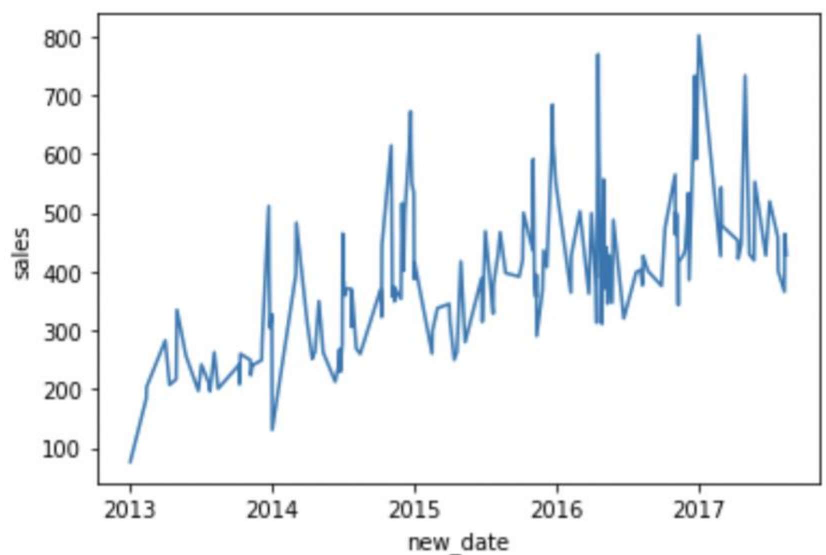
1) Replace the null values in the daily oil price with its mean value
2) Reformate data to the standard format
3) Categorize string values in family and store num so that we won't have a lot dummy variables when analysis

e.g., In variable “family”, we replaced 'AUTOMOTIVE', 'HARDWARE', 'LAWN AND GARDEN', 'PLAYERS AND ELECTRONICS' with ‘Tools’ so that we would have more data for each category for more accurate prediction.

After that, we applied one-hot encoder to encode the categorical features including date as well as family. We split our data into two parts, 70% for training and the rest 30% for testing.

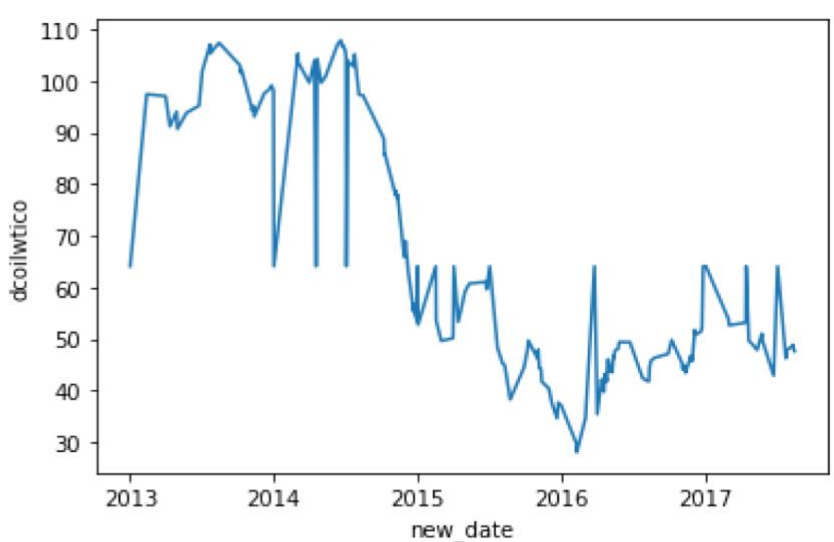
Data Exploration:

1)



This graph shows our sales trend along with the timeline. It’s clear that store sales have an overall positive trend. Now we would like to know if it’s highly related with daily oil price.

2)



This graph is the daily oil price along with the timeline. According to the regression, they are not highly related. That's why we didn't take this variable into prediction.

Model 1: Ridge

We start with simplest model: Ridge, to penalize the parameters in our linear model. Firstly, we set the alpha at 0.5 and did cross validation on it with 10 splits. The Mean squared error we got is 614162.

We are not satisfied with this outcome, so we tuned the hyperparameters for Ridge. We set the alpha for 10 times from 0 to 1, step of 0.1 and it turned out that the optimal alpha for our data set is 0 which means it's better depicted by linear model. Still the MSE is very large at 538179 and the accuracy rate is only 0.55. That's why we did a linear regression to make comparison with Ridge model.

Model 2: Gradient Boosting

The best feature of gradient boosting is that gives a prediction model in the form of an ensemble of weak prediction models. We set the number of boosting stages at 500 and the maximum depth of learning is 3 because our calculation limitation of our machine but we are pretty sure that the more stages we set, the more accurate prediction the model will make.

The accuracy score we got for gradient boosting is 0.71, not bad and the mean squared error of 347081, much smaller than the one we got in Ridge regression.

Model 3: Random Forest

Compared to gradient boosting, random forest is less sensitive to overfitting if the data is noisy and easier to tune with deeper depth.

We set the maximum trees at 100 and the maximum depth at 15. Easily, we get the accuracy score at 0.845, even outperformed than gradient boosting and with the mean squared error at only 185514.

Model Summary:

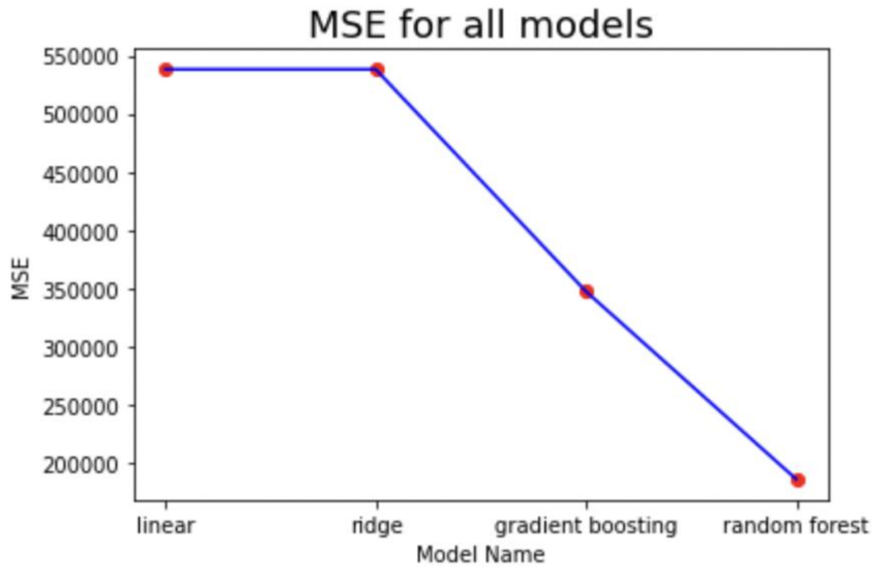
Putting all models together, we get the table with their MSE and accuracy score respectively:

Model Name	linear	ridge	gradient boosting	random forest
Accuracy Score	0.55	0.55	0.71	0.85
MSE	538177.6424612665	538178.84	347081.3990477412	185513.8784201336

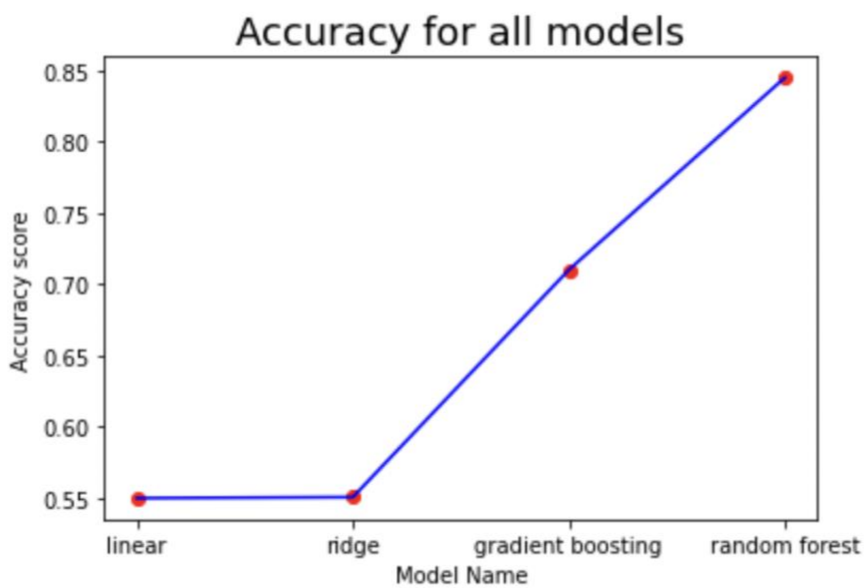
It's evident that random forest makes the best prediction with the smallest MSE and highest accuracy score.

Let's visualize them:

1) The graph for MSE of all models



2) The graph for accuracy rate of all models



Summary:

At last, we choose the random forest to do the prediction for its great performance in our test set. Due to the limited number of variables, we cannot make the best of Ridge regression to penalize parameters for each variable. Technically, gradient boosting would outperform random forest because of its slow learning from the weak models. However, in our case, it could be the reason that there is a lot of noise in our data which needs deeper clean. This might lead to overfitting in gradient boosting. If we tune the gradient boosting model, it might do a better job.

Shifan_Huang_ML_Final

December 19, 2021

[1]:
```
#pip install kaggle; be sure to have kaggle.json(api) in /.kaggle
#pip install tensorflow
import numpy as np
import pandas as pd
import seaborn as sns
from kaggle.api.kaggle_api_extended import KaggleApi
```

```
Warning: Your Kaggle API key is readable by other users on this system! To fix
this, you can run 'chmod 600 /Users/shifan/.kaggle/kaggle.json'
```

[2]:
```
#authenticate API and download the competition file in your root directory
api = KaggleApi()
api.authenticate()
api.competition_download_files('store-sales-time-series-forecasting',path='./')
```

```
Warning: Your Kaggle API key is readable by other users on this system! To fix
this, you can run 'chmod 600 /Users/shifan/.kaggle/kaggle.json'
```

[3]:
```
#open the zip file
import zipfile
with zipfile.ZipFile('./store-sales-time-series-forecasting.zip', 'r') as␣
 ↪zipref:
    zipref.extractall('./')
```

First, we will begin by import data and prep data

[4]:
```
holidays = pd.read_csv('holidays_events.csv')
oil = pd.read_csv('oil.csv')
stores = pd.read_csv('stores.csv')
test = pd.read_csv('test.csv')
train = pd.read_csv('train.csv')
transactions=pd.read_csv('transactions.csv')
```

[5]:
```
#creating another copy of test data which will be used later.
test1=pd.read_csv('test.csv')
```

[6]:
```
#Merging all available datasets to perform exploratory data analysis
m1=pd.merge(holidays,oil)
m2=pd.merge(train,stores)
```

```
m3=pd.merge(m2,transactions)
df=pd.merge(m3,m1,on="date")
```

[7]:
```
df.head()
```

[7]:
```
      id        date  store_nbr      family  sales  onpromotion   city  \
0  73062  2013-02-11          1  AUTOMOTIVE    0.0            0  Quito
1  73063  2013-02-11          1   BABY CARE    0.0            0  Quito
2  73064  2013-02-11          1      BEAUTY    0.0            0  Quito
3  73065  2013-02-11          1   BEVERAGES  172.0            0  Quito
4  73066  2013-02-11          1       BOOKS    0.0            0  Quito

       state type_x  cluster  transactions   type_y    locale locale_name  \
0  Pichincha      D       13           396  Holiday  National     Ecuador
1  Pichincha      D       13           396  Holiday  National     Ecuador
2  Pichincha      D       13           396  Holiday  National     Ecuador
3  Pichincha      D       13           396  Holiday  National     Ecuador
4  Pichincha      D       13           396  Holiday  National     Ecuador

  description  transferred  dcoilwtico
0    Carnaval        False       97.01
1    Carnaval        False       97.01
2    Carnaval        False       97.01
3    Carnaval        False       97.01
4    Carnaval        False       97.01
```

[8]:
```
df.describe()
```

[8]:
```
                 id      store_nbr          sales    onpromotion  \
count  3.220470e+05  322047.000000  322047.000000  322047.000000
mean   1.682979e+06      26.994672     406.383452       3.727136
std    7.862493e+05      15.595174    1246.881240      15.512095
min    5.610000e+02       1.000000       0.000000       0.000000
25%    1.010616e+06      13.000000       1.000000       0.000000
50%    1.842406e+06      27.000000      19.000000       0.000000
75%    2.209556e+06      40.000000     241.260505       1.000000
max    3.000887e+06      54.000000  124717.000000     716.000000

             cluster   transactions     dcoilwtico
count  322047.000000  322047.000000  300003.000000
mean        8.531202    1734.117840      64.077912
std         4.713809    1050.335018      25.147682
min         1.000000      54.000000      27.960000
25%         4.000000    1030.000000      44.660000
50%         9.000000    1409.000000      51.440000
75%        13.000000    2148.000000      94.740000
max        17.000000    8359.000000     107.950000
```

[9]:
```python
#Replacing the null values with the mean daily oil prices
df.loc[(df.dcoilwtico.isnull()),'dcoilwtico']=df.dcoilwtico.mean()
```

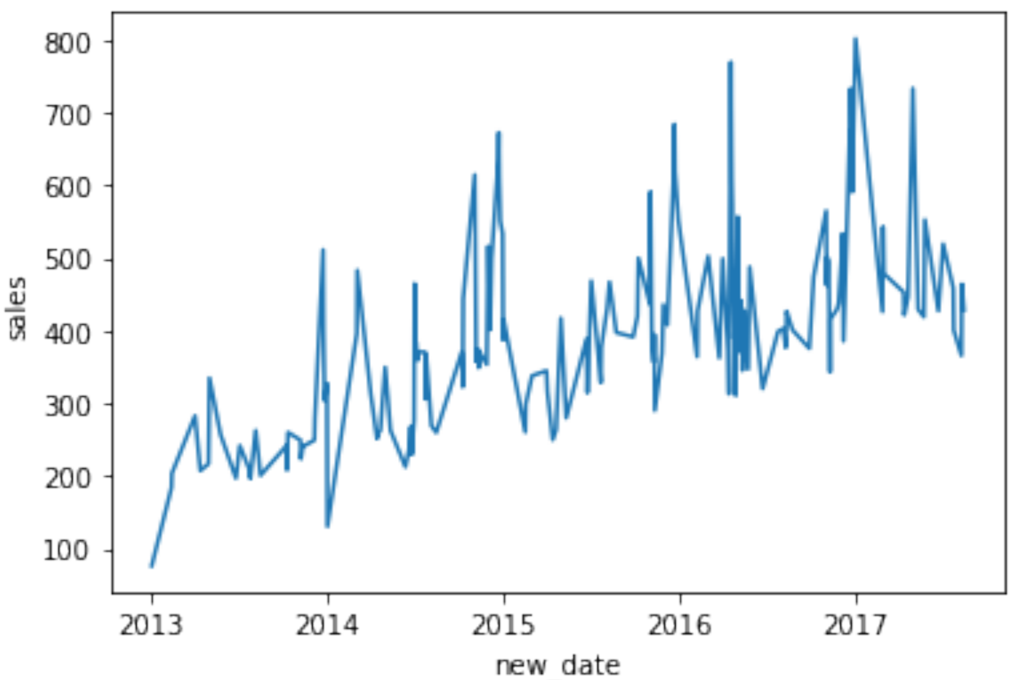
[10]:
```python
#Recheck null values in the dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 322047 entries, 0 to 322046
Data columns (total 17 columns):
 #   Column        Non-Null Count   Dtype
---  ------        --------------   -----
 0   id            322047 non-null  int64
 1   date          322047 non-null  object
 2   store_nbr     322047 non-null  int64
 3   family        322047 non-null  object
 4   sales         322047 non-null  float64
 5   onpromotion   322047 non-null  int64
 6   city          322047 non-null  object
 7   state         322047 non-null  object
 8   type_x        322047 non-null  object
 9   cluster       322047 non-null  int64
 10  transactions  322047 non-null  int64
 11  type_y        322047 non-null  object
 12  locale        322047 non-null  object
 13  locale_name   322047 non-null  object
 14  description   322047 non-null  object
 15  transferred   322047 non-null  bool
 16  dcoilwtico    322047 non-null  float64
dtypes: bool(1), float64(2), int64(5), object(9)
memory usage: 42.1+ MB
```

[11]:
```python
#Converting the date column from string to datetime dtype.
from datetime import datetime
df['new_date']=pd.to_datetime(df['date'],format='%Y-%m-%d',errors='coerce')
```

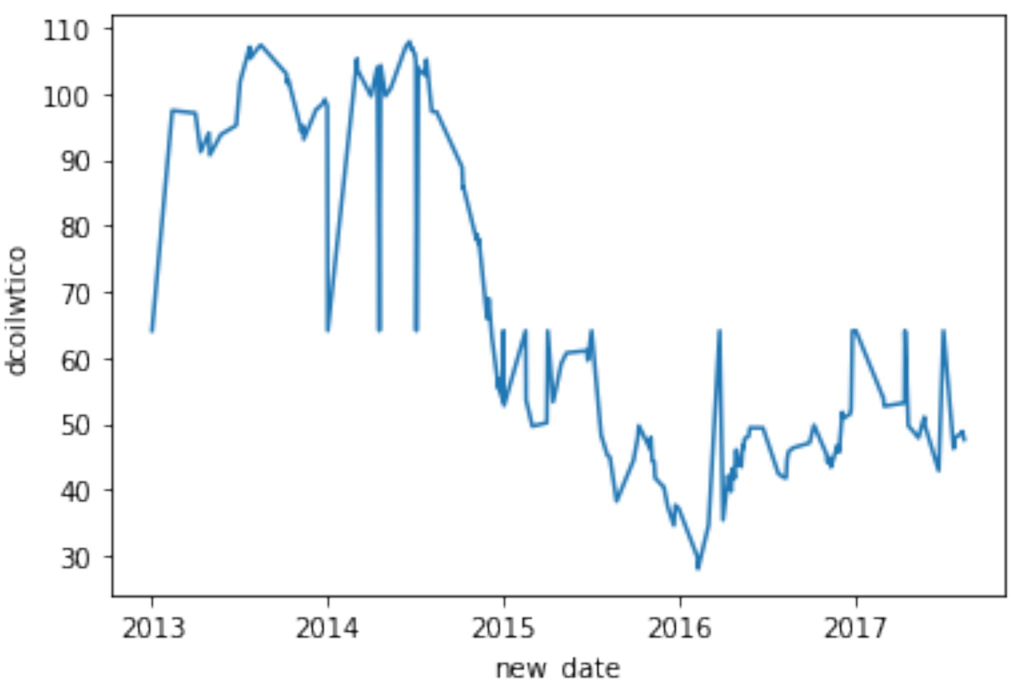
[12]:
```python
#Time Series plot of the sales data
sns.lineplot(x='new_date',y='sales',data=df,ci=None,estimator='mean')
```

[12]:
```
<AxesSubplot:xlabel='new_date', ylabel='sales'>
```




```
[123]: sns.lineplot(x='new_date',y='dcoilwtico',data=df,ci=None,estimator='mean')
```

```
[123]: <AxesSubplot:xlabel='new_date', ylabel='dcoilwtico'>
```



You can see from the line graph that the store sales has a positive trend. Since 2013 to 2017, sales increase in year by year.

Then we will categorize values in family and store_nbr in dataset, train and test set.

[13]:
```
df['family'].replace(['AUTOMOTIVE', 'HARDWARE', 'LAWN AND GARDEN', 'PLAYERS AND␣
↪ELECTRONICS'], 'Tools', inplace = True)
df['family'].replace(['BEAUTY', 'LINGERIE', 'LADIESWEAR','PERSONAL␣
↪CARE','CELEBRATION','MAGAZINES','BOOKS', 'BABY CARE'], 'LifeStyle', inplace␣
↪= True)
df['family'].replace(['HOME APPLIANCES','HOME AND KITCHEN I', 'HOME AND KITCHEN␣
↪II','HOME CARE','SCHOOL AND OFFICE SUPPLIES'], 'Home', inplace=True)
df['family'].replace([ 'GROCERY II', 'PET␣
↪SUPPLIES','SEAFOOD','LIQUOR,WINE,BEER'], 'Food', inplace=True)
df['family'].replace(['DELI', 'EGGS'], 'Daily', inplace=True)
```

[14]:
```
train['family'].replace(['AUTOMOTIVE', 'HARDWARE', 'LAWN AND GARDEN', 'PLAYERS␣
↪AND ELECTRONICS'], 'Tools', inplace = True)
train['family'].replace(['BEAUTY', 'LINGERIE', 'LADIESWEAR','PERSONAL␣
↪CARE','CELEBRATION','MAGAZINES','BOOKS', 'BABY CARE'], 'LifeStyle', inplace␣
↪= True)
train['family'].replace(['HOME APPLIANCES','HOME AND KITCHEN I', 'HOME AND␣
↪KITCHEN II','HOME CARE','SCHOOL AND OFFICE SUPPLIES'], 'Home', inplace=True)
train['family'].replace([ 'GROCERY II', 'PET␣
↪SUPPLIES','SEAFOOD','LIQUOR,WINE,BEER'], 'Food', inplace=True)
train['family'].replace(['DELI', 'EGGS'], 'Daily', inplace=True)
```

[15]:
```
test['family'].replace(['AUTOMOTIVE', 'HARDWARE', 'LAWN AND GARDEN', 'PLAYERS␣
↪AND ELECTRONICS'], 'Tools', inplace = True)
test['family'].replace(['BEAUTY', 'LINGERIE', 'LADIESWEAR','PERSONAL␣
↪CARE','CELEBRATION','MAGAZINES','BOOKS', 'BABY CARE'], 'LifeStyle', inplace␣
↪= True)
test['family'].replace(['HOME APPLIANCES','HOME AND KITCHEN I', 'HOME AND␣
↪KITCHEN II','HOME CARE','SCHOOL AND OFFICE SUPPLIES'], 'Home', inplace=True)
test['family'].replace([ 'GROCERY II', 'PET␣
↪SUPPLIES','SEAFOOD','LIQUOR,WINE,BEER'], 'Food', inplace=True)
test['family'].replace(['DELI', 'EGGS'], 'Daily', inplace=True)
```

Prepare data for modelling, in train and test set

[16]:
```
#splitting the train dataset into train and test and remove id from both sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(train.loc[:, train.columns !
↪= 'sales'], train['sales'], test_size=0.3, random_state=1)
X_train = X_train.drop(['id'], axis = 1)
X_test = X_test.drop(['id'], axis = 1)
```

```
test1 = test1.drop(['id'], axis = 1)
```

[17]:
```
#convert attribute
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder

# one-hot encode the categorical features
cat_attribs = ['date','family']
full_pipeline = ColumnTransformer([('cat',␣
 ↪OneHotEncoder(handle_unknown='ignore'), cat_attribs)],␣
 ↪remainder='passthrough')

encoder = full_pipeline.fit(X_train)
X_train = encoder.transform(X_train)
X_test = encoder.transform(X_test)
test1 = encoder.transform(test1)
```

After we prep data and run descriptive analysis, next we will start predictive model by 3 approahes (1) Ridge regression with cross validation and tuning (2) GradientBoosting (3) Random Forest, then we will compare the efficiency by Mean Squared Error (MSE)

1. Ridge regression and cross validation

[56]:
```
#Ridge regression
from sklearn.linear_model import Ridge
ridg = Ridge(fit_intercept=True, solver='auto', alpha=0.5, normalize=True)
```

[57]:
```
#Cross validation
from numpy import absolute
from numpy import mean
from numpy import std
from sklearn.model_selection import RepeatedKFold
from sklearn.model_selection import cross_val_score

cv_ridge = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)
scores = cross_val_score(ridg, X_train, y_train,␣
 ↪scoring='neg_mean_squared_error', cv=cv_ridge, n_jobs=-1)

# Calculation of mean of MSE for 10 folders
scores = np.absolute(scores)#force it to be positive
print('MSE RidgeCV: %.3f (%.3f)' % (mean(scores), std(scores)))
```

```
MSE RidgeCV: 614161.598 (27957.154)
```

[20]:
```
#Tuning Ridge
from sklearn.model_selection import GridSearchCV
from numpy import arange
grid = dict()
```

```
grid['alpha'] = arange(0,1,0.1)#tune for 10 times with step at 0.1
search = GridSearchCV(ridg, grid, scoring='neg_mean_squared_error',␣
 ↪cv=cv_ridge, n_jobs=-1)
results = search.fit(X_train, y_train)
print('MSE: %.3f' % absolute(results.best_score_))
print('Config: %s' % results.best_params_)
```

```
MSE: 560652.051
Config: {'alpha': 0.0}
```

[58]:
```
#Change alpha to 0
ridg_tun = Ridge(fit_intercept=True, solver='auto', alpha=0, normalize=True)
scores_tun = cross_val_score(ridg_tun, X_train, y_train,␣
 ↪scoring='neg_mean_squared_error', cv=cv_ridge, n_jobs=-1)
scores_tun = np.absolute(scores_tun)
print('MSE Ridge_tun: %.3f (%.3f)' % (mean(scores_tun), std(scores_tun)))
ridg_tun.fit(X_train,y_train)
```

```
MSE Ridge_tun: 560652.051 (26783.795)
```

[58]:
```
Ridge(alpha=0, normalize=True)
```

[ ]:
```
#linear regression
from sklearn.linear_model import LinearRegression
lm = LinearRegression()
lm.fit(X_train,y_train)
lm_score = lm.score(X_test, y_test)
#MSE
lm_yhat = lm.predict(X_test)
MSE_lm = mean_squared_error(y_test, lm_yhat)
```

[120]:
```
ridg_score = round(ridg_tun.score(X_test,y_test),2)
MSE_ridge = round(mean_squared_error(y_test, ridg_tun.predict(X_test)),2)
```

[22]:
```
#Prediction for real test set (the one we want for submission)
model_ridge_pre = ridg_tun.predict(test1)
print('Predicted :', model_ridge_pre)
```

```
Predicted : [ 264.56039119  264.56039119  302.11232613 … 1090.88246029
455.96651735
  286.98281011]
```

2. Gradient Boosting

[68]:
```
from sklearn.ensemble import BaggingRegressor, RandomForestRegressor,␣
 ↪GradientBoostingRegressor
from sklearn.metrics import mean_squared_error
```

[69]:
```
#getting parameters for gradient boosting
gbr_params = {'n_estimators': 500,
          'max_depth': 3,
          'min_samples_split': 5,
          'learning_rate': 0.01,
          'loss': 'ls'}
gbr = GradientBoostingRegressor(**gbr_params)
```

[70]:
```
gbr.fit(X_train, y_train)
```

[70]:
```
GradientBoostingRegressor(learning_rate=0.01, min_samples_split=5,
                          n_estimators=500)
```

[93]:
```
#predict in test set
gbr_yhat= gbr.predict(X_test)
gbr_yhat
```

[93]:
```
array([  38.20118491,   64.33958077,  161.17252815, …,  122.55176769,
        316.51936669, 1266.40414321])
```

[71]:
```
#calculate accuracy score
print("Gradient Boosting Model Accuracy: %.3f" %gbr.score(X_test, y_test))
```

```
Gradient Boosting Model Accuracy: 0.710
```

[94]:
```
MSE_gbr = mean_squared_error(y_test, gbr_yhat)
print("The mean squared error (MSE) on test set: {:.4f}".format(MSE_gbr))
```

```
The mean squared error (MSE) on test set: 347081.3990
```

[29]:
```
#Prediction for real test set (the one for submission)
model_gradient_boosting_pre = gbr.predict(test1)
print('Predicted :', model_gradient_boosting_pre)
```

```
Predicted : [ 122.55176769  122.55176769  340.95697135 … 1319.09630681
511.00615635
  112.8660401 ]
```

3. Random Forest

[20]:
```
#run model in train set
rf = RandomForestRegressor(n_estimators = 100, random_state = 1, max_depth = 15)
```

[21]:
```
#predict in test set
rf.fit(X_train, y_train)
```

[21]:
```
RandomForestRegressor(max_depth=15, random_state=1)
```

[22]:
```
rf_yhat = rf.predict(X_test)
```

[23]:
```python
#calculate accuracy score
print("Random Forest Model Accuracy: %.3f" %rf.score(X_test, y_test))
```

```
Random Forest Model Accuracy: 0.845
```

[24]:
```python
#MSE
MSE_rf = mean_squared_error(y_test, rf_yhat)
print("The mean squared error (MSE) on test set: {:.4f}".format(MSE_rf))
```

```
The mean squared error (MSE) on test set: 185513.8784
```

[25]:
```python
#Prediction for real test set (for submission)
model_random_forest_pre = rf.predict(test1)
print('Predicted :', model_random_forest_pre)
```

```
Predicted : [  29.55321589   29.55321589  143.80742238 … 1479.43859435
410.58829382
   29.55321589]
```

Submission

[29]:
```python
test['prediction'] = model_random_forest_pre
```

[30]:
```python
test
```

[30]:
```
             id        date  store_nbr         family  onpromotion  \
0       3000888  2017-08-16          1          Tools            0
1       3000889  2017-08-16          1      LifeStyle            0
2       3000890  2017-08-16          1      LifeStyle            2
3       3000891  2017-08-16          1      BEVERAGES           20
4       3000892  2017-08-16          1      LifeStyle            0
…           …         …        …              …          …
28507   3029395  2017-08-31          9        POULTRY            1
28508   3029396  2017-08-31          9  PREPARED FOODS           0
28509   3029397  2017-08-31          9        PRODUCE            1
28510   3029398  2017-08-31          9           Home            9
28511   3029399  2017-08-31          9           Food            0

        prediction
0        29.553216
1        29.553216
2       143.807422
3      2080.227761
4        29.553216
…              …
28507   559.107877
28508    42.583192
28509  1479.438594
28510   410.588294
```

```
28511    29.553216

[28512 rows x 6 columns]
```

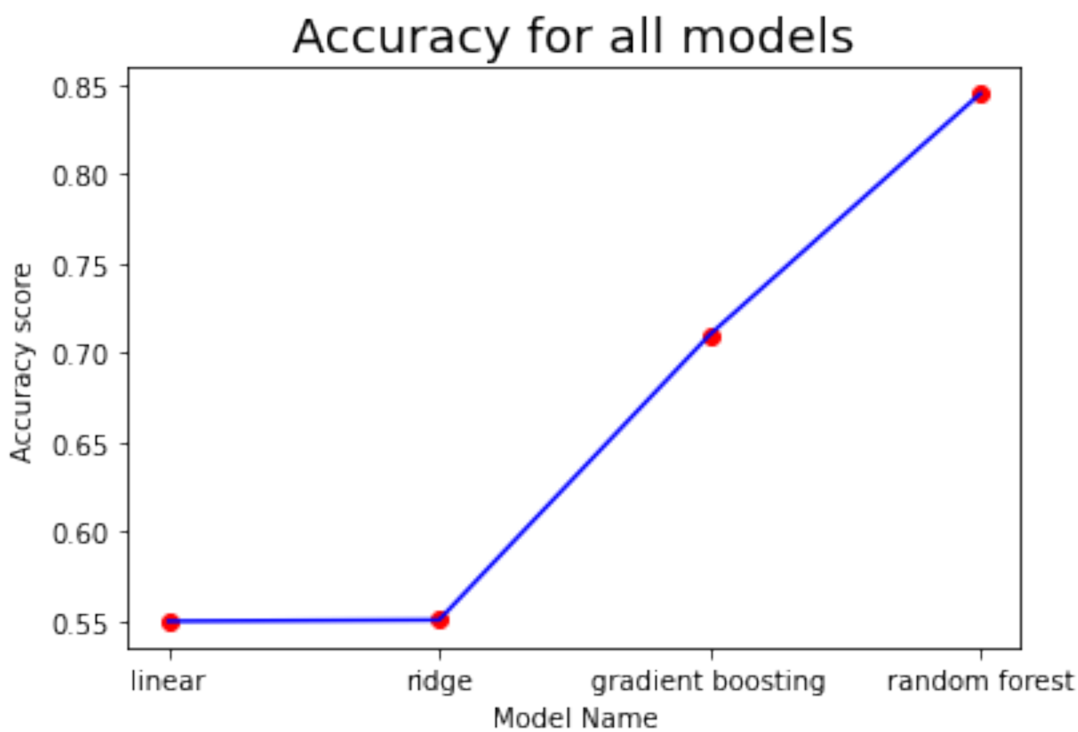
[31]:
```
submission = pd.DataFrame({'Id': test.id, 'Sales': model_random_forest_pre})
submission.to_csv('submission.csv', index=False)
```

Visualization of the outcome

[116]:
```
lm_score = round(lm_score,2)
gbr_score = round(gbr.score(X_test, y_test),2)
rf_score = round(rf.score(X_test, y_test),2)
```

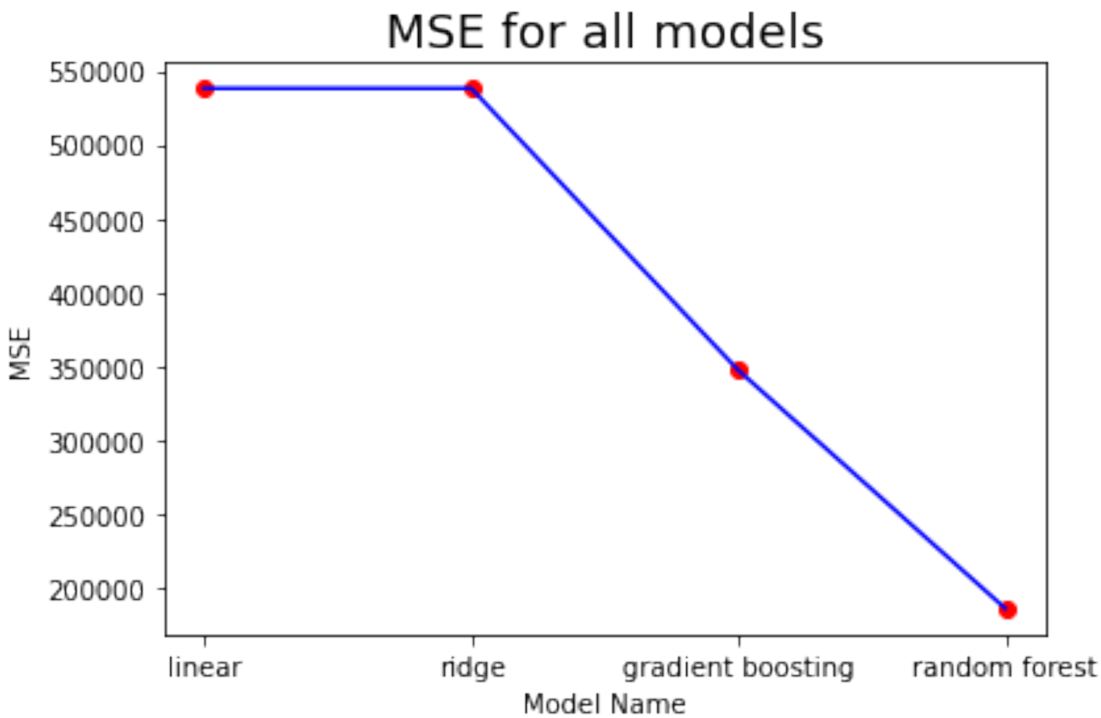
[86]:
```
import matplotlib.pyplot as plt
draw_x = ("linear","ridge","gradient boosting","random forest")
draw_y = (lm_score,ridg_score,gbr_score,rf_score)
plt.scatter(draw_x,draw_y,color = "red")
plt.title("Accuracy for all models",fontsize=18)
plt.xlabel("Model Name",fontsize=10)
plt.ylabel("Accuracy score", fontsize=10)
plt.plot(draw_x,draw_y,color = "blue")
```

[86]:
```
[<matplotlib.lines.Line2D at 0x7fcf476ca5e0>]
```



[99]:
```
draw_x = ("linear","ridge","gradient boosting","random forest")
draw_z = (MSE_lm,MSE_ridge,MSE_gbr,MSE_rf)
plt.scatter(draw_x,draw_z,color = "red")
plt.title("MSE for all models",fontsize=18)
plt.xlabel("Model Name",fontsize=10)
plt.ylabel("MSE", fontsize=10)
plt.plot(draw_x,draw_z,color = "blue")
```

[99]:
```
[<matplotlib.lines.Line2D at 0x7fcf4563bd90>]
```



[135]:
```
from tabulate import tabulate
print(tabulate([["Model Name","linear","ridge","gradient boosting","random␣
↪forest"],
                ["Accuracy Score",lm_score,ridg_score,gbr_score,rf_score],
                ["MSE",MSE_lm,MSE_ridge,MSE_gbr,MSE_rf]]))
```

```
--------------  -----------------  ---------  -----------------
-----------------
Model Name      linear             ridge      gradient boosting  random forest
Accuracy Score  0.55               0.55       0.71               0.85
MSE             538177.6424612665  538178.84  347081.3990477412
185513.8784201336
--------------  -----------------  ---------  -----------------
-----------------
```