Approach

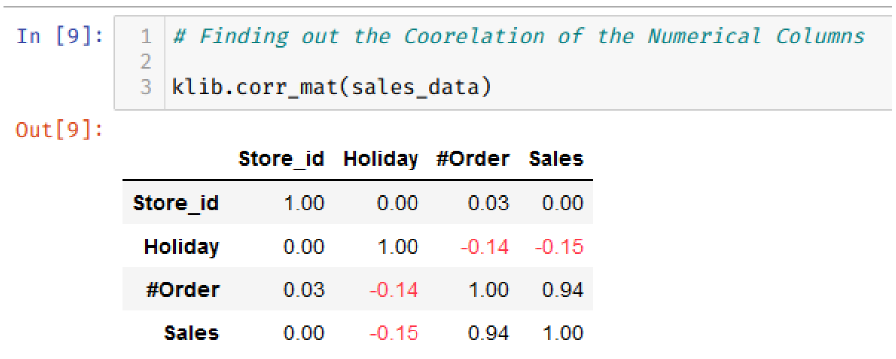
In this problem, the sales dataset has been provided and prediction has to be made for the sales for the next 2 months. The dataset provided for sales contains a set of Categorical Column and Numerical Column. Most of the columns are categorical and two columns contain numerical data i.e. Order and Sales. The target column is the sales. The dataset contains a 'Date' attribute which tells us the number of orders and sales for that particular date. The date column is very important since the holiday column, discount column, Order column and our target column i.e. Sales column are dependent on it. Also, we need to predict the sales for the next 2 months.

How are the holiday column, discount column, Order column, and our target column dependent on the Date column?

- The Holiday column is dependent on the date column because holidays happen upon a particular day.
- Discounts happen due to festivals or special occasions which are dependent on what day of the month it is.
- If there is a festival, there is a discount and the number of orders increases.
- As the number of orders increases, the sales increase.

Initially looking at the problem seemed like the Sales column is the target column. But, for any unseen dataset (which doesn't contain the Sales column), if we knew the number of Orders, we can easily compute approximate sales. But this is not the case and we don't know how many orders there can be in the future and hence we don't know the Sales. So, we have two targets here, the number of orders and the sales, and we need to use Multilabel Prediction. (Although we don't need the number of orders for our output file)

How Each of the Data are Correlated shown below.



```
In [9]:  # Finding out the Coorelation of the Numerical Columns

         klib.corr_mat(sales_data)
Out[9]:
```

	Store_id	Holiday	#Order	Sales
Store_id	1.00	0.00	0.03	0.00
Holiday	0.00	1.00	-0.14	-0.15
#Order	0.03	-0.14	1.00	0.94
Sales	0.00	-0.15	0.94	1.00

Figure 1: Correlation Matrix

Steps Taken

1. First, we need to disintegrate the date column into their corresponding date, month, and year and create their corresponding column. This will help us to categorize dates, months, and years.
2. Drop unnecessary columns like the ID and Store_id in this case. Because this data is not numeric and even it cannot be considered as categorical data. We need to create a generalized model, we don't require ID and Store_id for each Store for that purpose.
3. Standardize and encode all the numerical and categorical attributes correspondingly which will help in converging the solution very fast.
4. Create some Machine Learning Models and try to fit a sample of data to each one of them and then evaluate the performance metrics (using hold out dataset, cross-validation).
5. Once a model performs well, we will fine-tune the model with Hyper Parameter Tuning.

6. Once the model with the best parameter is ready, we will again train this fine-tuned model on the whole dataset.
7. Next, we just need to get some unseen data (TEST_FINAL.csv in this case) and make the sales prediction.

Data Preprocessing/Feature Engineering

1. The date column has been broken into their respective day, month, and year.
2. Two columns have been dropped i.e. the ID and store_id.
3. The categorical and numerical columns are filtered out.
4. OneHotEncoding is performed on the categorical column and standardization is performed on the numerical column.
5. No Imputers were used since the data doesn’t contain null values

How did you discover them?

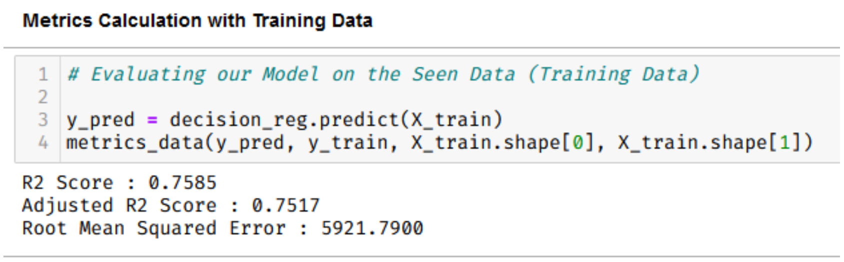
The date attribute is very important and as discussed in the above Approach section that this attribute needs to be broken into categories so we can use them in our prediction. Since we need to predict for the upcoming 61 days (as mentioned in the question).

What does your final model look like?

We selected two models for this purpose. The first one is the Decision Tree and the Second one was Random Forest.

Decision Tree

When using a Decision Tree with no hyper-parameter tuning, we got the following results on a sample of seen/unseen data.



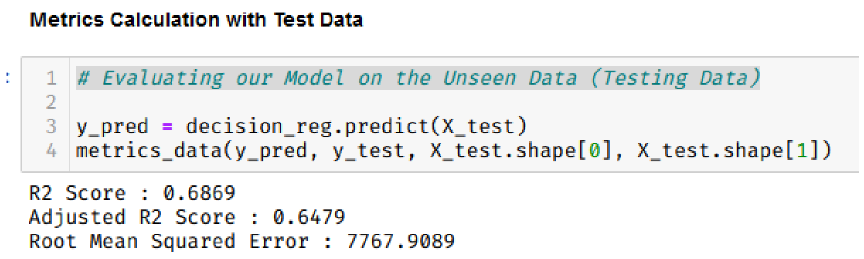
Metrics Calculation with Training Data

```
# Evaluating our Model on the Seen Data (Training Data)

y_pred = decision_reg.predict(X_train)
metrics_data(y_pred, y_train, X_train.shape[0], X_train.shape[1])
```

```
R2 Score : 0.7585
Adjusted R2 Score : 0.7517
Root Mean Squared Error : 5921.7900
```

Figure 2: On Seen Data



Metrics Calculation with Test Data

```
# Evaluating our Model on the Unseen Data (Testing Data)

y_pred = decision_reg.predict(X_test)
metrics_data(y_pred, y_test, X_test.shape[0], X_test.shape[1])
```

```
R2 Score : 0.6869
Adjusted R2 Score : 0.6479
Root Mean Squared Error : 7767.9089
```

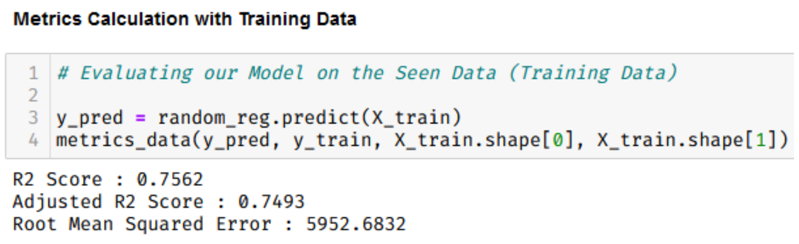
Figure 3: On Unseen Data

Metrics	Performed on Training Data	Performed on Test Data
R2 Score	0.7585	0.6869
Adjusted R2 Score	0.7517	0.6474
Root Mean Squared Error	5921.7900	7767.9089

(All the above values might change if the script is run again)

Random Forest

When using Random Forest with no hyper-parameter tuning, we got the following results on a sample of seen/unseen data.



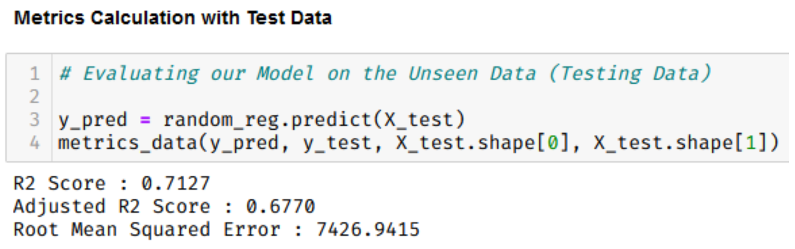
Metrics Calculation with Training Data

```
# Evaluating our Model on the Seen Data (Training Data)

y_pred = random_reg.predict(X_train)
metrics_data(y_pred, y_train, X_train.shape[0], X_train.shape[1])
```

```
R2 Score : 0.7562
Adjusted R2 Score : 0.7493
Root Mean Squared Error : 5952.6832
```

Figure 4: On Seen Data



Metrics Calculation with Test Data

```
# Evaluating our Model on the Unseen Data (Testing Data)

y_pred = random_reg.predict(X_test)
metrics_data(y_pred, y_test, X_test.shape[0], X_test.shape[1])
```

```
R2 Score : 0.7127
Adjusted R2 Score : 0.6770
Root Mean Squared Error : 7426.9415
```

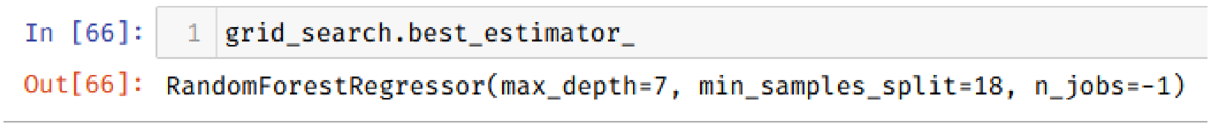
Figure 5: On Unseen Data

Metrics	Performed on Training Data	Performed on Test Data
R2 Score	0.7562	0.7127
Adjusted R2 Score	0.7493	0.6770
Root Mean Squared Error	5952.6832	7426.9415

(All the above values might change if the script is run again)

If we compare the scores we can see that Random Forest is doing better than Decision Trees, since decision trees are very susceptible to overfitting. We will select Random Forest as our model and fine-tune it with Grid Search.

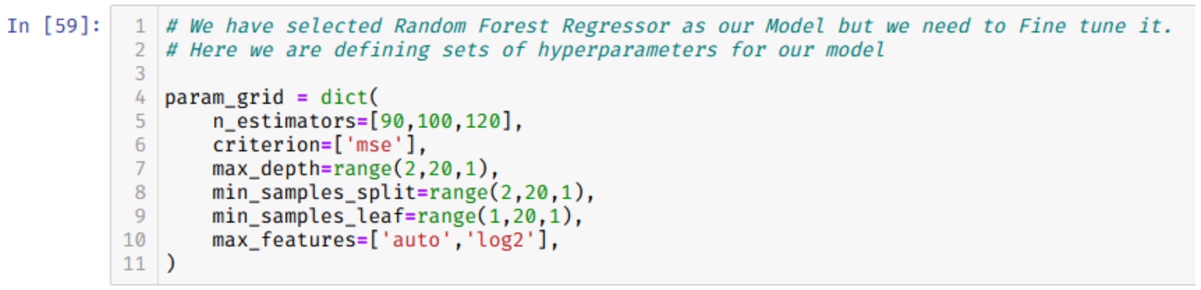
The Final Model is



```
In [66]:  grid_search.best_estimator_
Out[66]: RandomForestRegressor(max_depth=7, min_samples_split=18, n_jobs=-1)
```

How did you reach it?

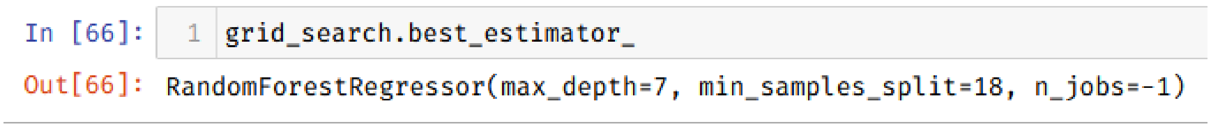
After performing GridSearchCV, with the following parameters,



```
In [59]:  # We have selected Random Forest Regressor as our Model but we need to Fine tune it.
          # Here we are defining sets of hyperparameters for our model

          param_grid = dict(
              n_estimators=[90,100,120],
              criterion=['mse'],
              max_depth=range(2,20,1),
              min_samples_split=range(2,20,1),
              min_samples_leaf=range(1,20,1),
              max_features=['auto','log2'],
          )
```

the best estimator that we find out for Random Forest Regressor is



```
In [66]:  grid_search.best_estimator_
Out[66]: RandomForestRegressor(max_depth=7, min_samples_split=18, n_jobs=-1)
```