Data Science Use Case for Online Retailer

Marcellus Ruben Winastwan

Friday 25 June, 2021

Table of Content

List of Tables and Figures

1. Will a Customer Buy a Product in the Next 3 Months?

The name of this Chapter describes perfectly the use case that needs to be answered: will a customer buy a product in the online store in the next 3 months?

To tackle this problem, there are several feature engineering that should be done in order to capture the buying behavior of each customer. But first thing first, let's take a look at the data.

	Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Customer ID	Country
0	489434	85048	15CM CHRISTMAS GLASS BALL 20 LIGHTS	12	2009-12-01 07:45:00	6.95	13085.0	United Kingdom
1	489434	79323P	PINK CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085.0	United Kingdom
2	489434	79323W	WHITE CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085.0	United Kingdom
3	489434	22041	RECORD FRAME 7" SINGLE SIZE	48	2009-12-01 07:45:00	2.10	13085.0	United Kingdom
4	489434	21232	STRAWBERRY CERAMIC TRINKET BOX	24	2009-12-01 07:45:00	1.25	13085.0	United Kingdom

Figure 1.1: Initial data

As can be seen above, there are several features that will be useful to use to capture the behavior of each customer.

1.1. Data Wrangling and Feature Engineering

Since the goal is to predict if a customer will make a purchase in the next three months, it is necessary to collect the buying behavior of each customer from previous months.

For this specific use case, each customer's buying behavior from the past 18 months is collected to predict their first purchase within the next three months. The chosen cut-off date for this problem is the first of September 2011.

Next, we need to collect the data about the day a customer makes a first purchase after the cutoff date (i.e after 1st of September 2011) and the day a customer makes the last purchase before the cutoff date (i.e before 1st of September 2011)

	CustomerID	MaxPurchaseDate
0	12346.0	2011-01-18 10:17:00
1	12347.0	2011-08-02 08:48:00
2	12348.0	2011-04-05 10:47:00
3	12349.0	2010-10-28 08:23:00
4	12350.0	2011-02-02 16:01:00

	CustomerID	MinPurchaseDate
0	12347.0	2011-10-31 12:25:00
1	12348.0	2011-09-25 13:13:00
2	12349.0	2011-11-21 09:51:00
3	12352.0	2011-09-20 14:34:00
4	12356.0	2011-11-17 08:40:00

Figure 1.2: Customer's last purchase day before cutoff date(left) and customer's first purchase day after cutoff date (right)

1.1.1. Feature Engineering 1: Next Purchase Day

This is the first feature that will be added into the final customer dataframe. The next purchase day can be described as the date interval between a customer's last purchase day before the cutoff date and their first purchase day after the cutoff date.

Let's take a look at the distribution of this feature.

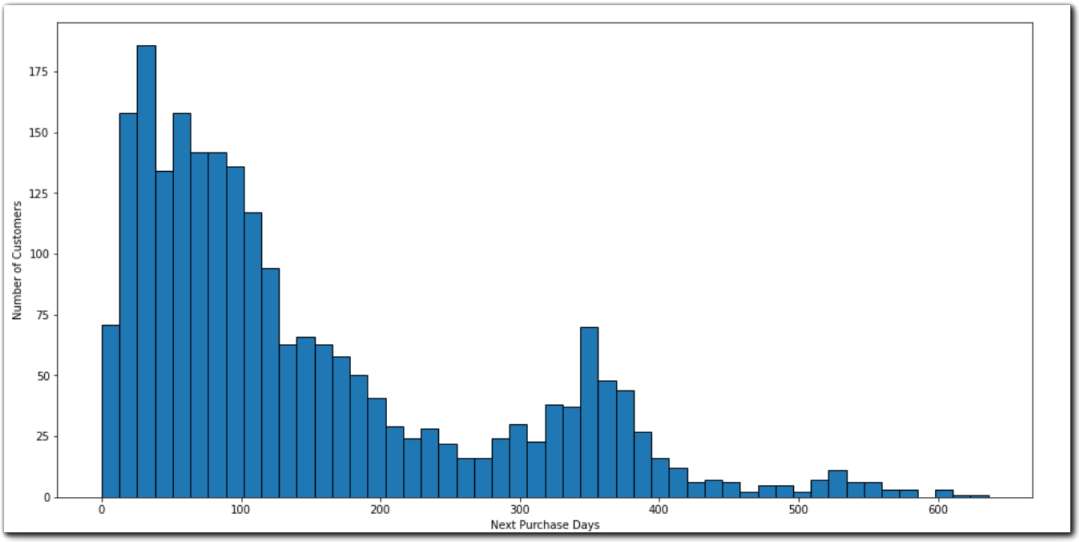


Figure 1.3: Distribution of next purchase day variable

From the distribution above, it can be seen that the day interval between a customer's last purchase before cutoff date and their first purchase after cutoff date is mostly below 250 days.

1.1.2. Feature Engineering 2: Recency and Recency Segment

The next useful feature that will be added to the final customer dataframe is the recency, which measures whether a customer is an active buyer or not. This can be measured by the time interval between the last purchase day of each customer before cutoff date and the last day before cutoff date.

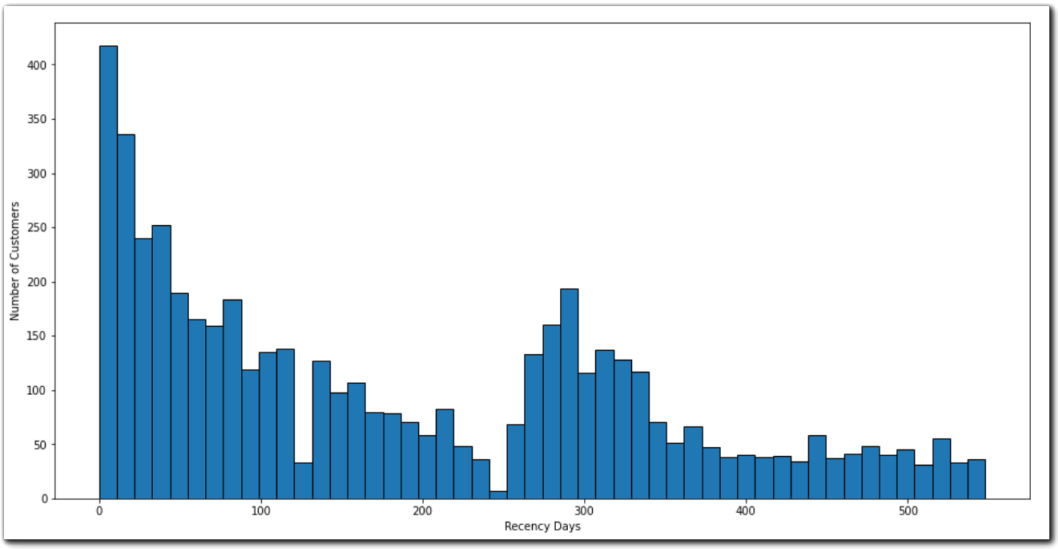


Figure 1.4: Distribution of recency days

As can be seen above, most of the customers have recency below 200 days.

Next, another feature should be created, which is a customer segmentation or customer clustering based on their recency days. This is beneficial to assign the recency score later on. To do clustering, K-means algorithm is implemented. But first,it is important to find out the optimum number of clusters that should be assigned.

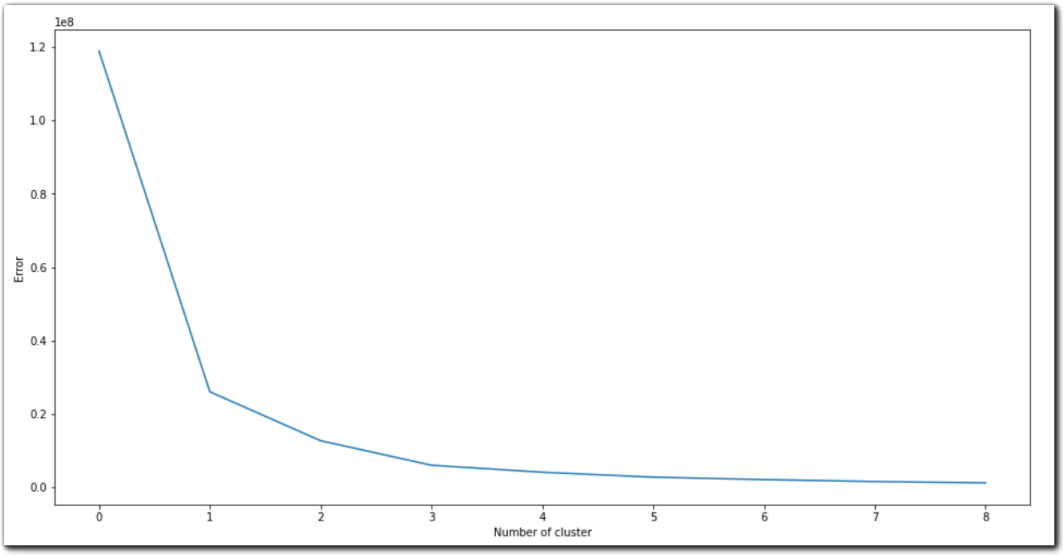


Figure 1.5: Number of clusters vs error

Based on the elbow method, it seems that 3 clusters would be the optimal one. Hence,there will be 3 clusters assigned for the recency.

RecencyCluster	count	mean	std	min	25%	50%	75%	max
0	685.0	456.487591	52.066438	369.0	412.0	455.0	499.0	548.0
1	1625.0	278.490462	51.492754	171.0	251.0	286.0	318.0	367.0
2	2751.0	62.248637	48.782709	0.0	20.0	50.0	99.0	170.0

Figure 1.6: Customer cluster based on recency days

From the table above, it can be concluded that based on the characteristic of the cluster, the most active customers are grouped together in cluster 2, while the least active customers are grouped together in cluster 0.

1.1.3. Feature Engineering 3: Frequency and Frequency Segment

This is the next feature that will be added to the final customer dataframe. Frequency can be described as the total number of days a customer has made a purchase in our online store, or how frequent are they in purchasing an item in our store. Hence, it can be computed by simply aggregating the number of days a customer has made a purchase in the store.

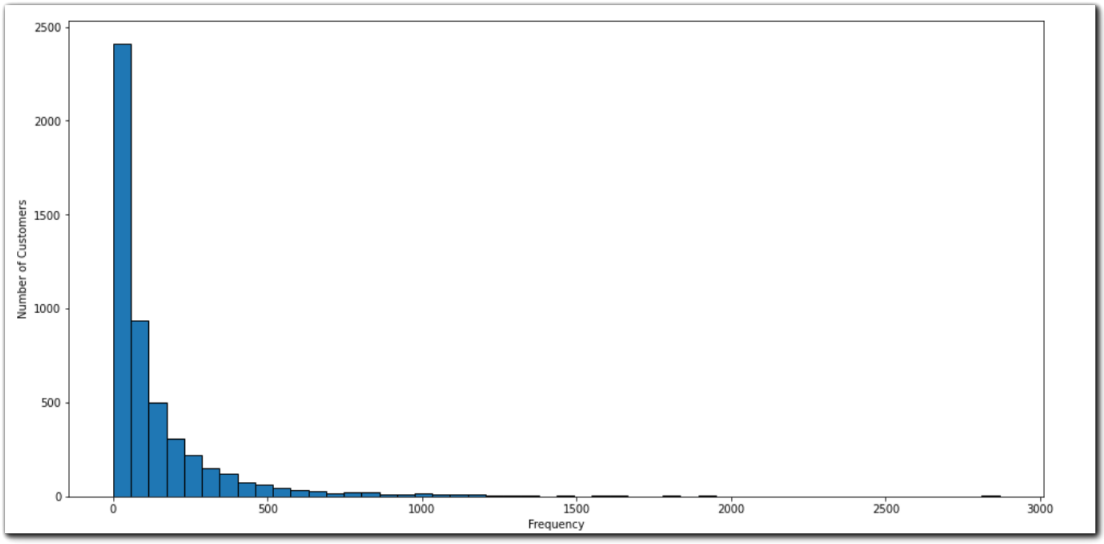


Figure 1.7: Distribution of frequency

It can be seen that most customers have purchasing frequency below 500 days.

Next, same as the steps that has been done previously, K-means clustering is applied to segment the customer based on their purchasing frequency.

FrequencyCluster	count	mean	std	min	25%	50%	75%	max
0	4783.0	101.398495	110.145266	1.0	23.00	58.0	140.5	522.0
1	272.0	940.161765	534.495351	524.0	628.00	784.5	1042.5	4130.0
2	6.0	8095.166667	3468.493645	4717.0	5523.25	7008.0	10536.5	13097.0

Figure 1.8: Customer cluster based on frequency

From the cluster characteristic, it can be concluded that the customers with cluster 2 are the ones who purchase items from the store frequently, while customers with cluster 0 are the least frequent ones.

1.1.4. Feature Engineering 4: Revenue and Revenue Segment

Revenue can be described as the amount of money that has been generated by each customer. To compute revenue, the item's price should be multiplied with the item's quantity that each customer has bought.

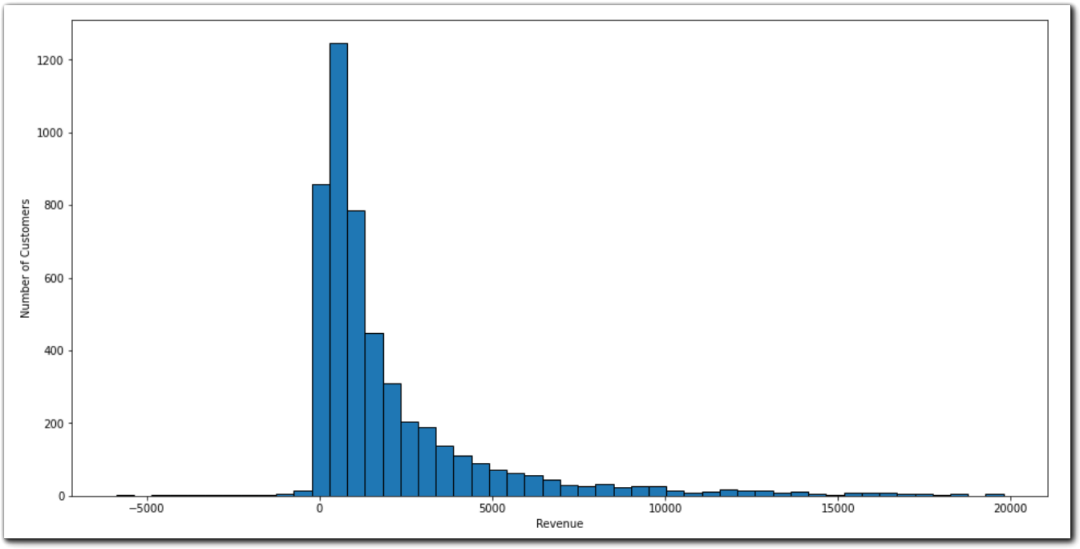


Figure 1.9: Distribution of reveue

Notice that the store actually lose money from few customers. Next, the feature that should be added is the customer segmentation based on the revenue generated by them.

RevenueCluster	count	mean	std	min	25%	50%	75%	max
0	5045.0	2550.584874	5295.087355	-25111.09	390.8600	1016.080	2571.0200	73573.47
1	14.0	155460.684286	68782.104825	84269.38	102475.2400	138946.865	186090.6350	296564.69
2	2.0	560778.645000	52943.312094	523342.07	542060.3575	560778.645	579496.9325	598215.22

Figure 1.10: Customer cluster based on revenue

Same as before, it can be interpreted that the customers in cluster 2 are the customers that generate high revenue to the store, while customers in cluster 0 are the customers that generate the least amount of revenue for the store.

1.1.5. Feature Engineering 5: Overall Score

This will be a very useful feature to be included in the final customer dataframe. Overall score can be computed based on the combination between each customer's recency cluster, frequency cluster, and revenue cluster that have been defined in the previous subsection. The higher the score, the better the combination of recency, frequency, and revenue from a particular customer.

OverallScore	Recency	Frequency	Revenue
0	456.487591	36.642336	501.734264
1	278.730031	74.463158	1113.275110
2	66.312977	139.612294	2922.868911
3	32.890625	882.957031	14659.751383
4	11.000000	4092.166667	124237.026667
5	9.750000	5322.000000	376329.722500

Figure 1.11: Overview of overall score variable

As can be seen above, if a customer has an overall score of 5, this means that he is a 5 star customer. This means he frequently bought items from the store, he's been active, and he generates a lot of revenue for the store. Vice versa for a customer with an overall score of 0.

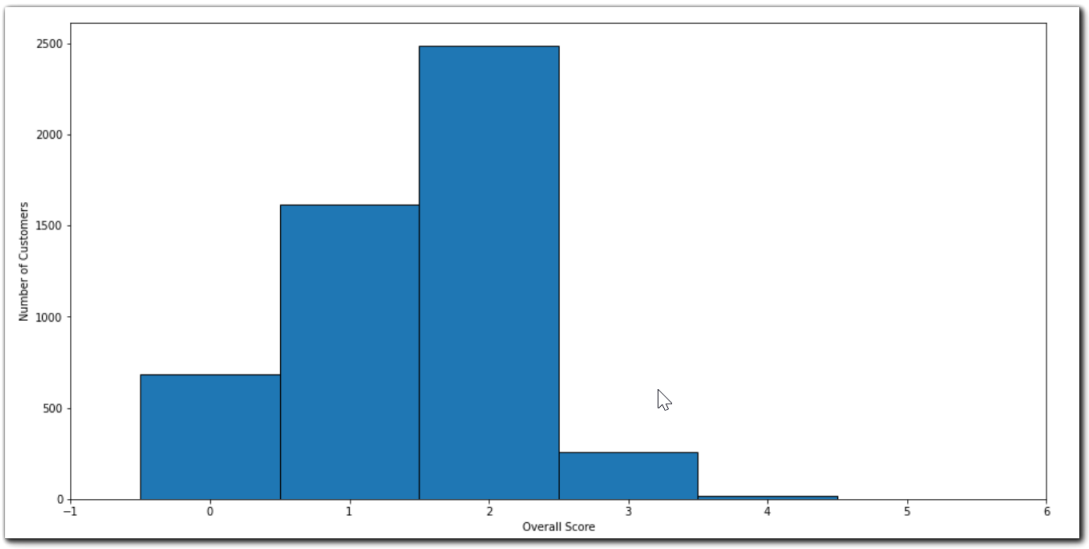


Figure 1.12: Distribution of overall score

We can see that most customers have the overall score of 2.

1.1.6. Feature Engineering 6: Add Days Interval Between Last 3 Purchases

The final features that need to be added are the days interval between the last 3 purchases of each customer before the cutoff date. Next, the mean and standard deviation of the last purchases for each customer can be computed as well. These mean and standard deviation are also added into the final customer dataframe.

Below is the final customer dataframe after adding all of the features listed above:

	CustomerID	NextPurchaseDay	Recency	RecencyCluster	Frequency	FrequencyCluster	Revenue	RevenueCluster	OverallScore	DayDiff	DayDiff2	DayDiff3	DayDiffMean	DayDiffStd
0	14798.0	91.0	44	2	51	0	715.11	0	2	166.0	237.0	388.0	72.000	65.589633
1	16717.0	48.0	44	2	408	0	6309.96	0	2	18.0	28.0	166.0	50.400	50.546348
2	12779.0	152.0	89	2	295	0	6406.76	0	2	2.0	9.0	59.0	30.600	31.422922
3	12598.0	90.0	61	2	211	0	4414.67	0	2	59.0	116.0	123.0	60.875	69.509378
4	14870.0	999.0	161	2	69	0	1794.60	0	2	139.0	152.0	237.0	64.500	57.263426

Figure 1.13: Final customer dataframe

1.1.7. Add Classification Label

This is the final thing that should be done before building a machine learning model. Our goal is to predict whether a given customer will purchase an item from the store within the next 3 months. Hence, the `NextPurchaseDay` feature that we have defined before will be transformed into binary feature. That is, if the next purchase day of a customer is above 90 days or 3 months, then we set the value to 0, otherwise the value will be 1.

Below is the final data that will be fed into machine learning model.

	Recency	RecencyCluster	Frequency	FrequencyCluster	Revenue	RevenueCluster	OverallScore	DayDiff	DayDiff2	DayDiff3	DayDiffMean	DayDiffStd	NextPurchaseDayRange
0	44	2	51	0	715.11	0	2	166.0	237.0	388.0	72.000	65.589633	0
1	44	2	408	0	6309.96	0	2	18.0	28.0	166.0	50.400	50.546348	1
2	89	2	295	0	6406.76	0	2	2.0	9.0	59.0	30.600	31.422922	0
3	61	2	211	0	4414.67	0	2	59.0	116.0	123.0	60.875	69.509378	1
4	161	2	69	0	1794.60	0	2	139.0	152.0	237.0	64.500	57.263426	0

Figure 1.14: Final data to bed into machine learning model

In the above table, the `NextPurchaseDayRange` feature will be the label.

1.2. Building ML Model with PyCaret

To build a machine learning model, PyCaret will be implemented. PyCaret is a wonderful open-source library that we can use for regression or classification task. But why do we need to use PyCaret? Whenever we want to build a machine learning model, often times we don't know which model will yield to the best solution for our problem. We can manually do grid searching and tune the hyperparameter to solve this problem, but it is not that practical. PyCaret gives the perfect solution for this problem.

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
ada	Ada Boost Classifier	0.8460	0.9210	0.8257	0.7826	0.8022	0.6764	0.6787	0.145
lda	Linear Discriminant Analysis	0.8453	0.9209	0.8204	0.7834	0.8007	0.6745	0.6759	0.019
lr	Logistic Regression	0.8447	0.9299	0.8117	0.7863	0.7980	0.6720	0.6731	0.412
ridge	Ridge Classifier	0.8440	0.0000	0.8222	0.7798	0.7996	0.6721	0.6737	0.015
gbc	Gradient Boosting Classifier	0.8440	0.9266	0.8185	0.7833	0.7985	0.6715	0.6743	0.285
rf	Random Forest Classifier	0.8427	0.9236	0.7939	0.7938	0.7920	0.6656	0.6676	0.565
et	Extra Trees Classifier	0.8260	0.9187	0.7500	0.7843	0.7658	0.6276	0.6289	0.504
lightgbm	Light Gradient Boosting Machine	0.8207	0.9120	0.7835	0.7570	0.7677	0.6220	0.6248	0.146
nb	Naive Bayes	0.8153	0.9086	0.6338	0.8394	0.7207	0.5872	0.6011	0.016
dt	Decision Tree Classifier	0.7800	0.7665	0.7111	0.7114	0.7101	0.5330	0.5342	0.020
knn	K Neighbors Classifier	0.7593	0.8235	0.7077	0.6771	0.6904	0.4940	0.4959	0.118
svm	SVM - Linear Kernel	0.7187	0.0000	0.6793	0.7370	0.6220	0.4237	0.4632	0.016
qda	Quadratic Discriminant Analysis	0.6213	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.018

Figure 1.15: Model performance comparison for our use case

PyCaret allows us to compare different models to find the best performing model for our use case in a split second. As can be seen from table above, AdaBoost classifier turned out to be the best performing model for our use case. Hence, AdaBoost is implemented in this use case.

After tuning the hyperparameter, below is the performance of the implemented AdaBoost classifier.

	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.8333	0.9211	0.8596	0.7424	0.7967	0.6568	0.6619
1	0.8467	0.9234	0.8772	0.7576	0.8130	0.6842	0.6895
2	0.8267	0.9004	0.8421	0.7385	0.7869	0.6419	0.6458
3	0.8667	0.9425	0.8772	0.7937	0.8333	0.7227	0.7252
4	0.8733	0.9225	0.8421	0.8276	0.8348	0.7321	0.7322
5	0.8667	0.9351	0.8246	0.8246	0.8246	0.7170	0.7170
6	0.8867	0.9242	0.9123	0.8125	0.8595	0.7651	0.7687
7	0.8467	0.9204	0.7895	0.8036	0.7965	0.6735	0.6735
8	0.8333	0.9229	0.8393	0.7460	0.7899	0.6526	0.6557
9	0.8667	0.9510	0.7679	0.8600	0.8113	0.7087	0.7115
Mean	0.8547	0.9263	0.8432	0.7906	0.8147	0.6955	0.6981
SD	0.0190	0.0131	0.0404	0.0401	0.0222	0.0379	0.0373

Figure 1.16: AdaBoost classifier performance for 10 fold cross-validation

1.3. Predict Test data

Whenever we use PyCaret environment to build machine learning solution, it will automatically allocate around 0.2 portion of the input data into test data. The accuracy of the models that can be seen above were solely based on the training data that is automatically generated by PyCaret. The next step should be using the selected model (in our case AdaBoost classifier) to predict the test data. Below is the accuracy score of the AdaBoost classifier on test data.

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	Ada Boost Classifier	0.8556	0.9366	0.8197	0.8032	0.8114	0.6944	0.6945

Figure 1.17: AdaBoost classifier performance on unseen test data

As it can be seen above, the accuracy of the model on test data is also similar with the training data, at around 85%. This means that the model doesn't suffer from overfitting problem and also, the model can generalize on unseen data relatively well.

The confusion matrix as well as the AUC curve can be observed in the following visualizations:

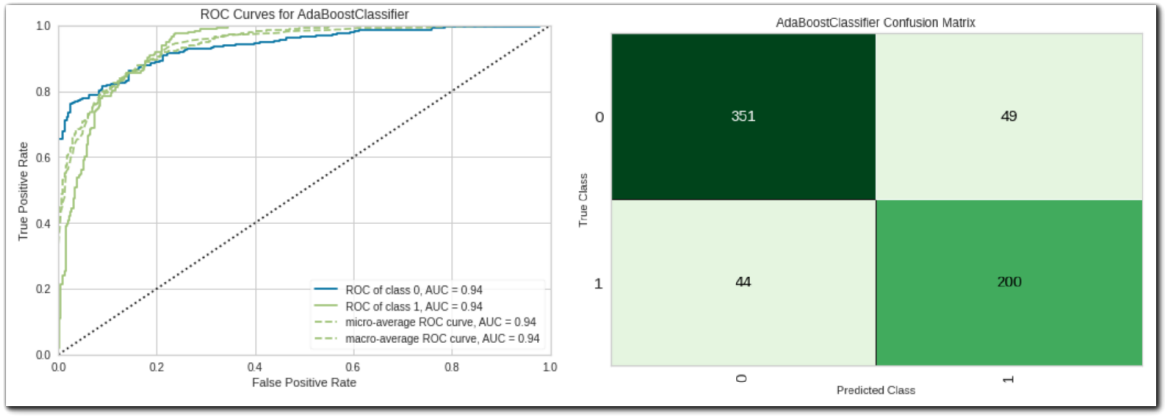


Figure 1.18: AUC and confusion matrix of AdaBoost classifier

The AdaBoost classifier model has a very good AUC score. Also, it performs really well to classify the customer's purchasing behavior as it has a relatively high true positive and true negative values.

The complete code is available in the corresponding Notebook in the following link:Notebook Code

2. Product Recommendation to a Customer

In this Chapter, the main goal that needs to be answered is: what products will most likely land into a customer's basket whenever they're doing their online shopping in our online store?

To answer this question, a recommender system needs to be built. There are two approaches that will be compared: one is products popularity based recommendation system and another is collaborative filtering. The metrics comparison between two approaches will be observed before selecting the approach with the best metrics.

2.1. Data Wrangling and Data Preparation

In order to create a recommender system, the first thing that we need to do is to transform the original data. Before, our data looks like the following:

	Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Customer ID	Country
0	489434	85048	15CM CHRISTMAS GLASS BALL 20 LIGHTS	12	2009-12-01 07:45:00	6.95	13085.0	United Kingdom
1	489434	79323P	PINK CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085.0	United Kingdom
2	489434	79323W	WHITE CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085.0	United Kingdom
3	489434	22041	RECORD FRAME 7" SINGLE SIZE	48	2009-12-01 07:45:00	2.10	13085.0	United Kingdom
4	489434	21232	STRAWBERRY CERAMIC TRINKET BOX	24	2009-12-01 07:45:00	1.25	13085.0	United Kingdom

Figure 2.1: Initial data

The data should be transformed such that the new dataframe only has three important columns:

- `CustomerID`, which is the customer unique ID
- `StockCode`, which is the product unique ID
- `Quantity`, which is the aggregated purchase made by specific customer on specific product.

In the end, the new dataframe will look like this:

	Customer ID	StockCode	Quantity
0	12346	15056BL	1
1	12346	15056N	1
2	12346	15056P	1
3	12346	20679	1
4	12346	20682	1

Figure 2.2: Input dataframe for recommendation system

Next, we can proceed to build the recommendation system. To build the recommendation system, a library called Turicreate will be used. Turicreate is a wonderful open-source library for wide range of machine learning tasks such as object detection, regression, classification, recommendation system, style transfer, etc. Personally, I find this library is very simple and intuitive to use, allowing us to focus on our task rather than the algorithm itself.

The last thing that needs to be done in this data preparation task is splitting the data into training and test set. Splitting the data into training and test data can be done easily with `train_test_split` from sklearn. The training data will be used to train the recommendation system model and then the test data will be used to evaluate the model performance.

2.2. Building Recommendation System

As previously mentioned, there are two different models that will be implemented for our recommendation system: one is product popularity based recommendation system and the other one is collaborative filtering.

2.2.1. Product Popularity Based Recommendation System

As the name suggest, product popularity based recommendation system will recommend the most popular products to each customer. Most popular products can be defined as the products with the highest number of sales across all customers.

Below is the result of model training with product popularity based recommendation system:

Customer ID	StockCode	score	rank
13085	16044	3096.0	1
13085	85220	2424.75	2
13085	37410	2138.9166666666665	3
13085	37352	1545.0	4
13085	21092	910.4285714285714	5
13078	16044	3096.0	1
13078	85220	2424.75	2
13078	37410	2138.9166666666665	3
13078	37352	1545.0	4
13078	21092	910.4285714285714	5
15362	16044	3096.0	1
15362	85220	2424.75	2
15362	37410	2138.9166666666665	3
15362	37352	1545.0	4
15362	21092	910.4285714285714	5
18102	16044	3096.0	1
18102	85220	2424.75	2
18102	37410	2138.9166666666665	3
18102	37352	1545.0	4
18102	21092	910.4285714285714	5

Figure 2.3: Output of product popularity based recommendation system

The above table is the result of the recommendation system. Each customer will get a recommendation of 5 different products, ranked by the products' popularity. As you might notice, the five products being recommended to each customer are the same, i.e there is no personalized nor customized product recommendation to each unique customer. This is because product popularity based recommendation system will only recommend the most purchased products to our customer.

2.2.2. Collaborative Filtering

In contrary with product popularity based recommendation system, collaborative filtering will add some personalized and customized product recommendation system to our customer.

Collaborative filtering is a recommendation system method where the products which will be recommended to a customer will depend on the customer's past purchase as well as how similar a customer's purchasing behavior compared to other customers. To do this, this recommendation system relies on user-item matrix, and then the similarity (purchasing behavior) between customers can be determined by computing the Eucledian distance. Normally, cosine similarity is the go-to algorithm for this purpose.

Below is the result of model training with collaborative filtering:

Customer ID	StockCode	score	rank
13085	21623	2.486994746658537	1
13085	85128	2.4869282643000283	2
13085	23134	2.486895650625229	3
13085	15059A	2.486836122141944	4
13085	84838	2.486836122141944	5
13078	22734	12.260336719575475	1
13078	22137	12.185552321496557	2
13078	23174	12.134347631305944	3
13078	23173	12.103399492678095	4
13078	23175	12.094692036753795	5
15362	21124	0.8878528429911687	1
15362	21988	0.8282309908133286	2
15362	22329	0.8256343878232516	3
15362	84327A	0.8231674020106976	4
15362	21207	0.8214291838499216	5
18102	21477	12.804740055829663	1
18102	21169	12.56091207680823	2
18102	37345	12.526922040725056	3
18102	20913	12.512313367445257	4
18102	23046	12.48281115142605	5

Figure 2.4: Output of collaborative filtering

Now there are different products to be recommended to each customer and they are different between each customer. Also, the score of the product is different between each customer, because they have, of course, different preference. This is expected from collaborative filtering.

2.3. Recommender System Model Evaluation

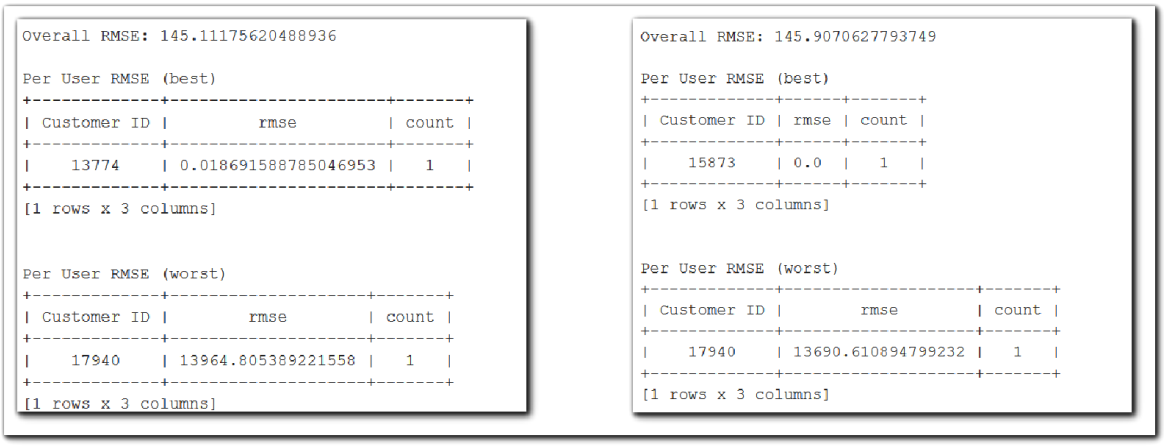
To evaluate the performance of recommendation system model, there are several metrics that one can consider. In our case, 3 types of metrics have been defined, which are the root mean square error (RMSE), precision, and recall. The indicator of how good our recommender system with those metrics can be interpreted as follows:

- The lower the RMSE, the better the recommendation.

- The higher the precision, the better the recommendation. Precision basically tells us: out of all the products that have been recommended to customers, how many do they actually liked?

- The higher the recall, the better the recommendation. Recall basically tells us: what percentage of products that a customer buys are actually being recommended to them?

Hence, all three of the metrics are equally important. Let's start with RMSE.



```
Overall RMSE: 145.11175620488936

Per User RMSE (best)
+-------------+---------------------+-------+
| Customer ID |         rmse        | count |
+-------------+---------------------+-------+
|    13774    | 0.018691588785046953 |   1   |
+-------------+---------------------+-------+
[1 rows x 3 columns]


Per User RMSE (worst)
+-------------+--------------------+-------+
| Customer ID |        rmse        | count |
+-------------+--------------------+-------+
|    17940    | 13964.805389221558 |   1   |
+-------------+--------------------+-------+
[1 rows x 3 columns]
```

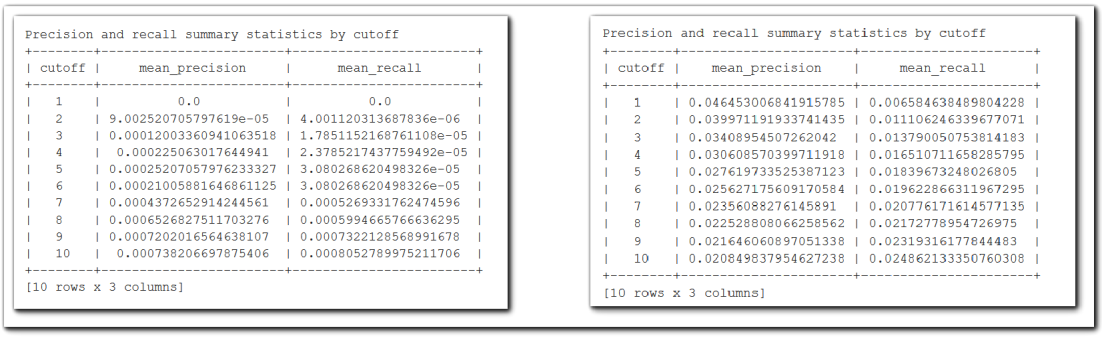
```
Overall RMSE: 145.9070627793749

Per User RMSE (best)
+-------------+------+-------+
| Customer ID | rmse | count |
+-------------+------+-------+
|    15873    | 0.0  |   1   |
+-------------+------+-------+
[1 rows x 3 columns]


Per User RMSE (worst)
+-------------+--------------------+-------+
| Customer ID |        rmse        | count |
+-------------+--------------------+-------+
|    17940    | 13690.610894799232 |   1   |
+-------------+--------------------+-------+
[1 rows x 3 columns]
```

Figure 2.5: RMSE of product popularity based recommender system (left) and collaborative filtering (right)

From the image above, it can be said that product popularity based recommendation system yields to lower overall RMSE. However, collaborative filtering model has better best-and-worst RMSE value in the customer level. Next,let's see the precision and recall of both models.



```
Precision and recall summary statistics by cutoff
+--------+---------------------+---------------------+
| cutoff |    mean_precision   |     mean_recall     |
+--------+---------------------+---------------------+
|   1    |         0.0         |         0.0         |
|   2    | 9.002520705797619e-05 | 4.001120313687836e-06 |
|   3    | 0.0001200336094106351 8 | 1.7851152168761108e-05 |
|   4    | 0.000225063017644941 | 2.3785217437759492e-05 |
|   5    | 0.0002520705797623332 7 | 3.080268620498326e-05 |
|   6    | 0.0002100588164686112 5 | 3.080268620498326e-05 |
|   7    | 0.0004372652914244561 | 0.0005269331762474596 |
|   8    | 0.0006526827511703276 | 0.0005994665766636295 |
|   9    | 0.0007202016564638107 | 0.0007322128568991678 |
|   10   | 0.000738206697875406 | 0.0008052789975211706 |
+--------+---------------------+---------------------+
[10 rows x 3 columns]
```

```
Precision and recall summary statistics by cutoff
+--------+---------------------+---------------------+
| cutoff |    mean_precision   |     mean_recall     |
+--------+---------------------+---------------------+
|   1    | 0.046453006841915785 | 0.006584638489804228 |
|   2    | 0.039971191933741435 | 0.011106246339677071 |
|   3    | 0.034089545072620 42 | 0.013790050753814183 |
|   4    | 0.030608570399711918 | 0.016510711658285795 |
|   5    | 0.027619733525387123 | 0.01839673248026805  |
|   6    | 0.025627175609170584 | 0.019622866311967295 |
|   7    | 0.02356088276145891  | 0.020776171614577135 |
|   8    | 0.022528808066258562 | 0.02172778954726975  |
|   9    | 0.021646060897051338 | 0.02319316177844483  |
|   10   | 0.020849837954627238 | 0.024862133350760308 |
+--------+---------------------+---------------------+
[10 rows x 3 columns]
```

Figure 2.6: Precision and recall product popularity based recommender system (left) and collaborative filtering (right)

From the result above, it can be clearly seen that collaborative filtering model is more superior in terms of precision and recall. This makes sense since product popularity based recommendation doesn't give a personalized recommendation to the customer. Based on the model evaluation, collaborative filtering will be implemented for the final recommendation system model.

2.4. Generating Product Recommendation to a Customer

Now here comes the fun part, where now we can generate product recommendation to a customer. From collaborative filtering model, we supply a unique Customer ID into it. Then, the model will generate 5 products recommendation ranked based on the similarity score with a given Customer ID. The first product will be the most likely product that will land in a customer's basket, the second product will be the second most likely, and so on. Below is the example of top 5 products being recommended to a customer with ID of 12346.

```
Number 1 product to be most likely to be in customer 12346 basket is CREAM HEART CARD HOLDER
Number 2 product to be most likely to be in customer 12346 basket is PINK/BROWN DOTS RUFFLED UMBRELLA
Number 3 product to be most likely to be in customer 12346 basket is CUBIC MUG BLUE POLKA DOT
Number 4 product to be most likely to be in customer 12346 basket is BLACK HEART CARD HOLDER
Number 5 product to be most likely to be in customer 12346 basket is ROSE DU SUD COSMETICS BAG
```

Figure 2.7: Rank of the products being recommended to a particular customer

The complete code is available in the corresponding Notebook in the following link: Notebook Code

3. Predict Weekly Sales

In this Chapter, the main question that needs to be answered is: can we predict the aggregated weekly sales 4 weeks in advance?

Based on the problem statement above, it can be inferred that this is a time series problem. Hence, there are few alternatives that can be tried to tackle this problem. In this Chapter, there are 3 different methods that will be implemented: using Autoregressive Moving Average, using Prophet library, and using LSTM model.

3.1. Data Wrangling and Data Aggregation

First things first, data wrangling and data aggregation need to be performed. As you might have already notice from previous Chapter, the original sales data looks like the following:

	Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Customer ID	Country
0	489434	85048	15CM CHRISTMAS GLASS BALL 20 LIGHTS	12	2009-12-01 07:45:00	6.95	13085.0	United Kingdom
1	489434	79323P	PINK CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085.0	United Kingdom
2	489434	79323W	WHITE CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085.0	United Kingdom
3	489434	22041	RECORD FRAME 7" SINGLE SIZE	48	2009-12-01 07:45:00	2.10	13085.0	United Kingdom
4	489434	21232	STRAWBERRY CERAMIC TRINKET BOX	24	2009-12-01 07:45:00	1.25	13085.0	United Kingdom

Figure 3.1: Initial data

As the interest is in sales with weekly interval, then the `Quantity` variable in the original data needs to be aggregated. For this use case, `Quantity` variable will be summed on weekly basis and it can be done easily with `groupby()` function from Pandas. Here is the final dataframe after data aggregation.

	InvoiceDate	Quantity
0	2009-12-06	136455
1	2009-12-13	111826
2	2009-12-20	133295
3	2009-12-27	38512
4	2010-01-03	0

Figure 3.2: Input dataframe for time series forecasting

One of the most important thing before building a time series model is to check whether the time series data is stationary, or perhaps there might be trend and seasonality in there. For the use case in this project, below is the visualization of the time series data.

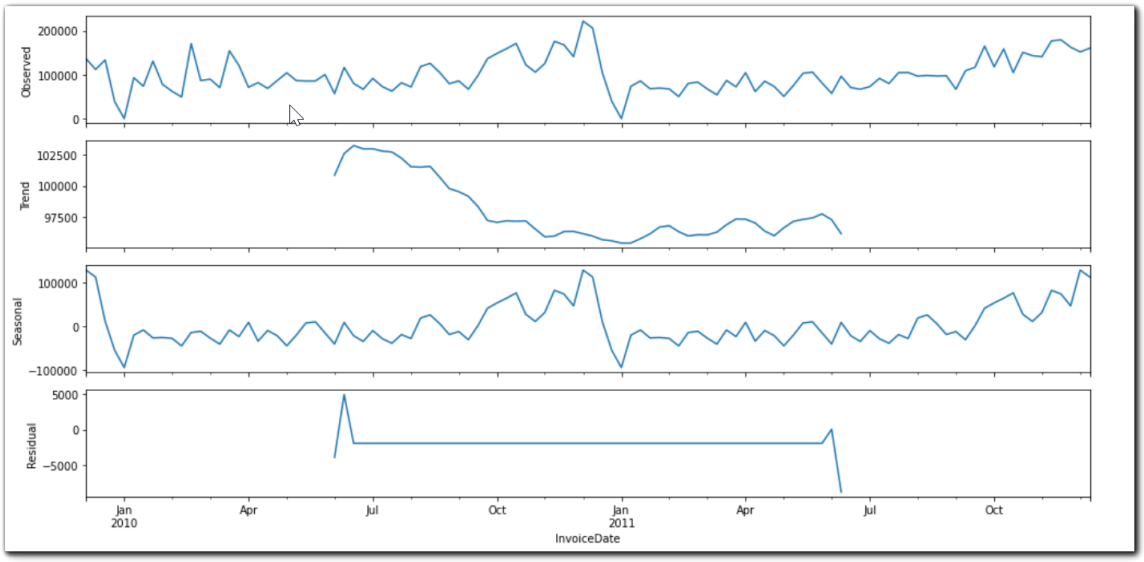


Figure 3.3: Trend, seasonality, and noise from input data

From the visualization above, it can be seen that the data is not stationary and there is seasonality that appears. This means that the amount of sales is relatively low in the beginning of the year, i.e January until around August, and then the sales spike up around September to December. This seasonality has been repeated for the last 2 years of the data, so it is important to make sure that the right time series forecasting algorithm is applied to tackle this problem.

3.2. Building Time Series Forecasting Model with SARIMA

As the data has seasonality in it and this problem can be considered as a univariate time series, then it will be appropriate to use SARIMA. SARIMA stands for Seasonal Autoregressive Integrated Moving Average, which works similarly with ARIMA. However, ARIMA wouldn't be suitable for this problem since ARIMA only takes trend into account, but not seasonality. Meanwhile, SARIMA is a forecasting method which takes both trend and seasonality into account.

The notation of SARIMA then can be written as:

```
SARIMA(p,d,q)(P,D,Q)m
```

where:

- ‘p‘ is the trend autoregressive order.
- ‘d‘ is the trend difference order.
- ‘q‘ is the trend moving average order.
- ‘P‘ is the seasonal autoregressive order.
- ‘D‘ is the seasonal difference order.
- ‘Q‘ is the seasonal moving average order
- ‘m‘ is the time step for single seasonal period.

Below is the example of how SARIMA notation would look like in action:

```
SARIMA(1,1,0)(1,1,0)12
```

There are a lot of parameters to consider in a SARIMA model. The problem is that it is impossible to know the optimum parameters in advance to build the best performing SARIMA model. Hence, a grid search algorithm needs to be implemented and then the Akaike Information Criterion (AIC) of each generated model needs to be observed. The smaller the AIC score, the better the model.

Below is the result example of the grid search:

```
ARIMA(1, 1, 0)x(0, 0, 1, 4) - AIC:2372.6902462510834
ARIMA(1, 1, 0)x(0, 1, 0, 4) - AIC:2438.977023483302
ARIMA(1, 1, 0)x(0, 1, 1, 4) - AIC:2303.89024843193
ARIMA(1, 1, 0)x(1, 0, 0, 4) - AIC:2372.6227125706346
ARIMA(1, 1, 0)x(1, 0, 1, 4) - AIC:2374.5983725520096
ARIMA(1, 1, 0)x(1, 1, 0, 4) - AIC:2313.3340003547855
ARIMA(1, 1, 0)x(1, 1, 1, 4) - AIC:2305.6724211785286
ARIMA(1, 1, 1)x(0, 0, 0, 4) - AIC:2447.8323313368037
ARIMA(1, 1, 1)x(0, 0, 1, 4) - AIC:2338.0898295045404
ARIMA(1, 1, 1)x(0, 1, 0, 4) - AIC:2395.492062907696
ARIMA(1, 1, 1)x(0, 1, 1, 4) - AIC:2273.4054897504484
ARIMA(1, 1, 1)x(1, 0, 0, 4) - AIC:2361.6537863327812
ARIMA(1, 1, 1)x(1, 0, 1, 4) - AIC:2339.8184121200684
ARIMA(1, 1, 1)x(1, 1, 0, 4) - AIC:2301.4720322626063
ARIMA(1, 1, 1)x(1, 1, 1, 4) - AIC:2275.2095320855783
```

Figure 3.4: Grid search to find the best parameter for SARIMA model

From the above result, it can be seen that SARIMA model with parameter (1,1,1)(0,1,1,4) has the smallest AIC score. Hence, this parameter is used for the final SARIMA model. Here the parameter 4 means that a single seasonality period lasts for one month. SARIMA model can be easily built with `statsmodel` library.

Next, it's always a best practice to observe the residual error of the model. Below is the residual plot of SARIMA model.

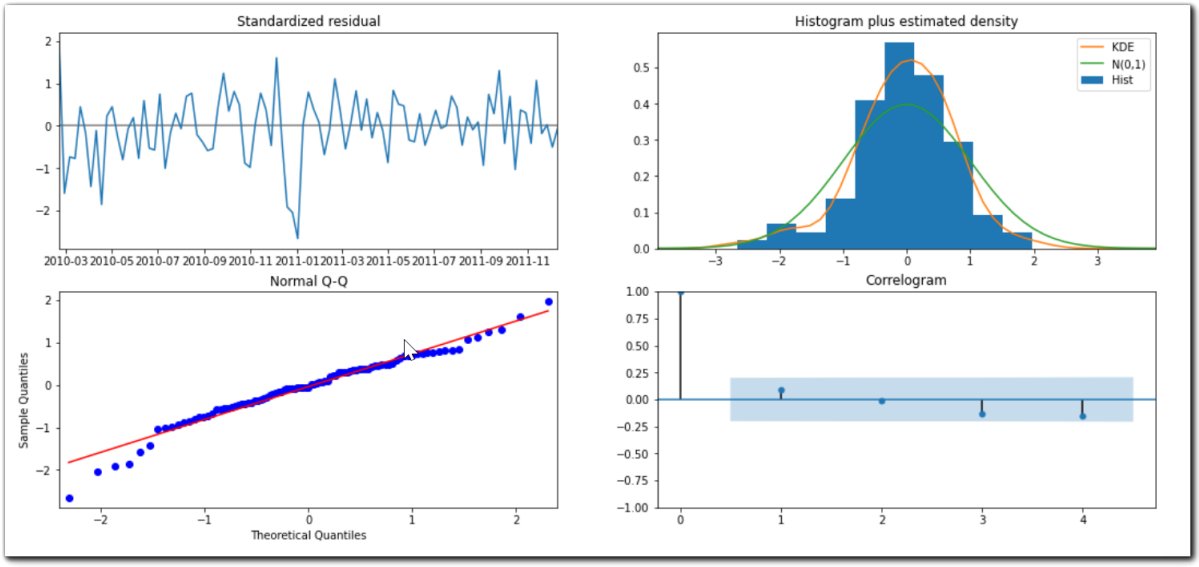


Figure 3.5: Residual plot of SARIMA model

Although not perfect, but it can be seen that the residual error of the model is more or less normally distributed and centered around zero with the exception of few data points, hence the model can be safely used for time series forecasting.

For this use case, the data from July 2011 onward will be treated as the test data that needs to be predicted by SARIMA model. Below is the prediction result.

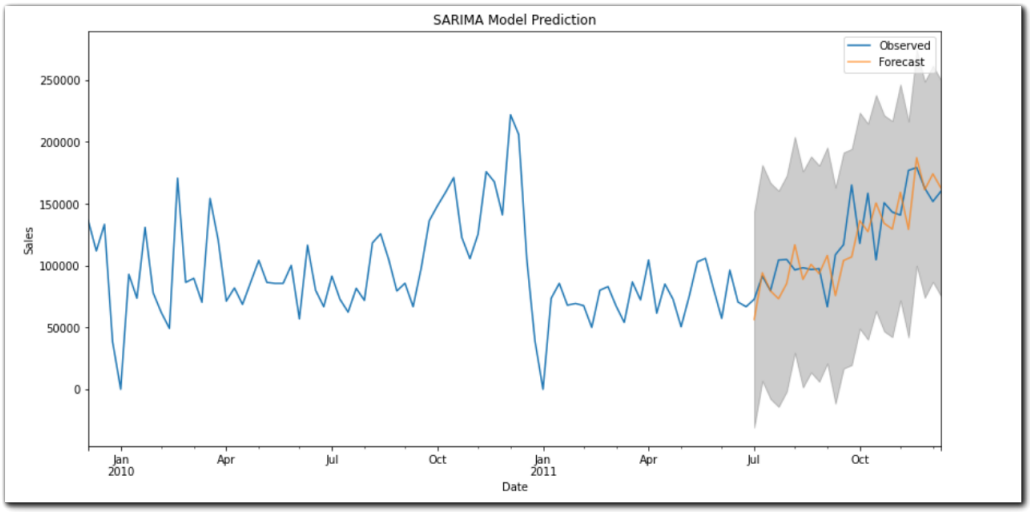


Figure 3.6: SARIMA model prediction

As it can be seen, the SARIMA model can recognize the trend and the seasonality

of the weekly sales, although the performance of this SARIMA model can be greatly improved if there are more data, for example data from the last 5 years instead of just 1.5 years. The more the data, the better the SARIMA model to recognize the seasonality.

3.3. Building Time Series Forecasting Model with Prophet

Aside of using SARIMA, there is also a library specific for time series forecasting called Prophet that can be used to solve this problem. Prophet is an open-source library released by Facebook and this library is a great tool to forecast a univariate time series problem, especially if the time series has a trend and seasonality in it.

First, the Prophet object is called. Next, the aggregated weekly sales data is supplied into it, and the confidence interval of the forecast can be specified as well. Below is the visualization of the forecasting result, prolonged until the next 12 weeks of the last input data.

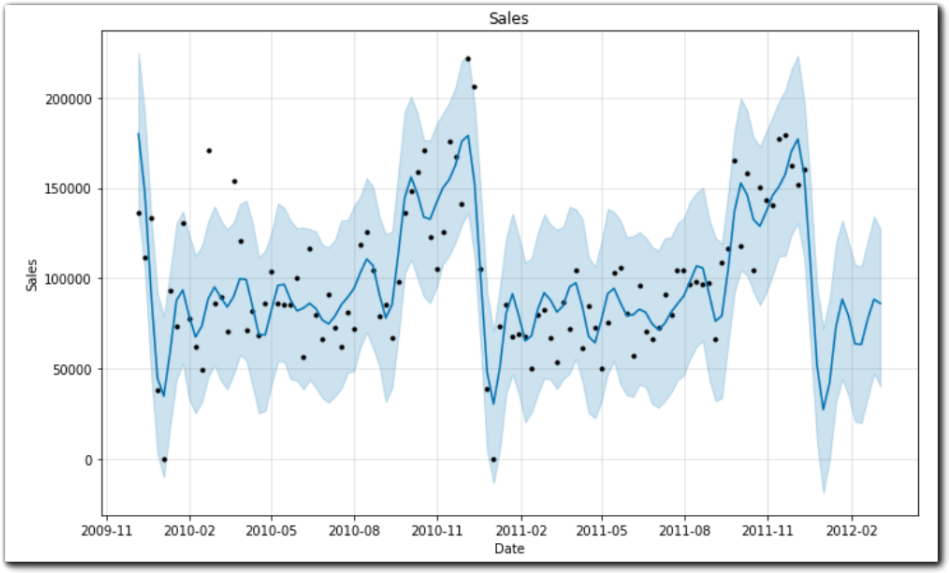


Figure 3.7: Prophet model prediction

It can be seen that the model captures the seasonality of the data perfectly. Like in the transition between December and January, the model predicts the sales' downward trend with very narrow confidence interval, which means that the model is very sure that the sales will drop in January.

The Prophet library could give us an insight about the overall trend of our sales as follows:

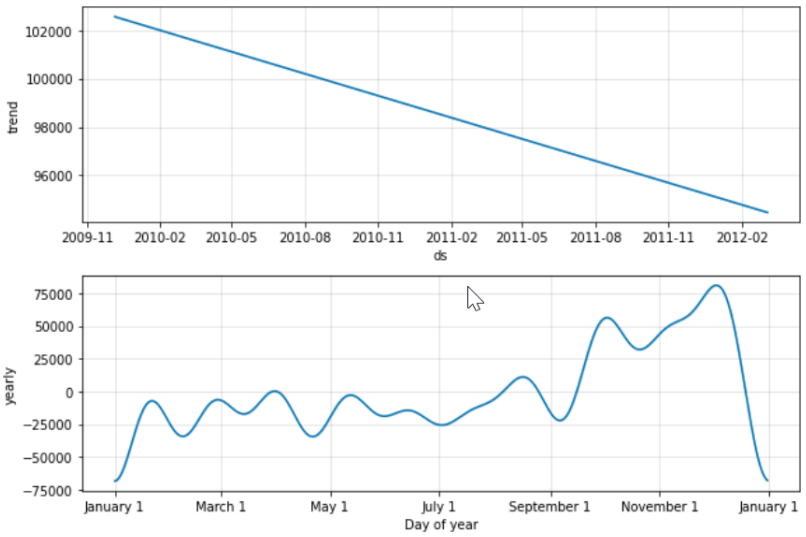


Figure 3.8: Overall sales trend generated by Prophet

From the plot above, it can be seen that overall, the amount of sales is showing downward trend within each month. Hence, a new business strategy is needed in order to counter this trend. Meanwhile, it can also be seen that the amount of sales is very high only in specific months, between October and December.

3.4. Building Time Series Forecasting Model with LSTM

The third approach that can be implemented to forecast weekly aggregated sales is by using deep learning model. In this section, an LSTM model to forecast the weekly sales is utilized.

LSTM stands for Long-Short Term Memory, which is one of Recurrent Neural Network architecture mostly popular for the application of Natural Language Processing. In order to use LSTM as the model, two things need to be done: remove the seasonality of the data and also, transform the time series data as supervised data.

3.4.1. Introducing Difference to Remove Seasonality

From the previous section, it is known that the time series data has seasonality in it. In order for LSTM to work, the data needs to be transformed such that it becomes more stationary. One of the method that can be used to enforce stationarity is difference method, which is the subtraction between the amount of sales from one month to previous month.

Below is the final dataframe after applying difference method.

	index	InvoiceDate	Quantity	PrevSales	Diff
1	1	2009-12-13	111826	136455.0	-24629.0
2	2	2009-12-20	133295	111826.0	21469.0
3	3	2009-12-27	38512	133295.0	-94783.0
4	4	2010-01-03	0	38512.0	-38512.0
5	5	2010-01-10	92971	0.0	92971.0

Figure 3.9: Dataframe after applying difference method

3.4.2. Introducing Lags

Aside from difference method, another approach that can be done to remove seasonality is by applying lags in the time series data. This can be done by shifting the time series data and then compare the shifted time series data with the original one. In our case, we will do 4 different lags.

Below is the final dataframe after applying lags:

	InvoiceDate	Quantity	Diff	lag_1	lag_2	lag_3	lag_4
0	2010-01-10	92971	92971.0	-38512.0	-94783.0	21469.0	-24629.0
1	2010-01-17	73633	-19338.0	92971.0	-38512.0	-94783.0	21469.0
2	2010-01-24	130851	57218.0	-19338.0	92971.0	-38512.0	-94783.0
3	2010-01-31	77908	-52943.0	57218.0	-19338.0	92971.0	-38512.0
4	2010-02-07	62061	-15847.0	-52943.0	57218.0	-19338.0	92971.0

Figure 3.10: dataframe after applying lags

Next, the data needs to separated into training and test set, and the normalize both of training and test set. For the test set, the date from 3rd of July 2011 onward is used.

3.4.3. Building and Generating LSTM Prediction

The applied LSTM model is very simple and there is no need to build a deep model. The LSTM model consists of one hidden layer, which has 16 cells, before the final dense layer which consists of only one neuron. Below is the summary of the LSTM model:

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (1, 16)                   1408
_________________________________________________________________
dense_1 (Dense)              (1, 1)                    17
=================================================================
Total params: 1,425
Trainable params: 1,425
Non-trainable params: 0
_________________________________________________________________
```

Figure 3.11: LSTM model summary

The model is trained on 300 epochs, with Adam optimizer, and utilizing mean square error as the loss function. Below is the prediction generated from the LSTM model on test data.

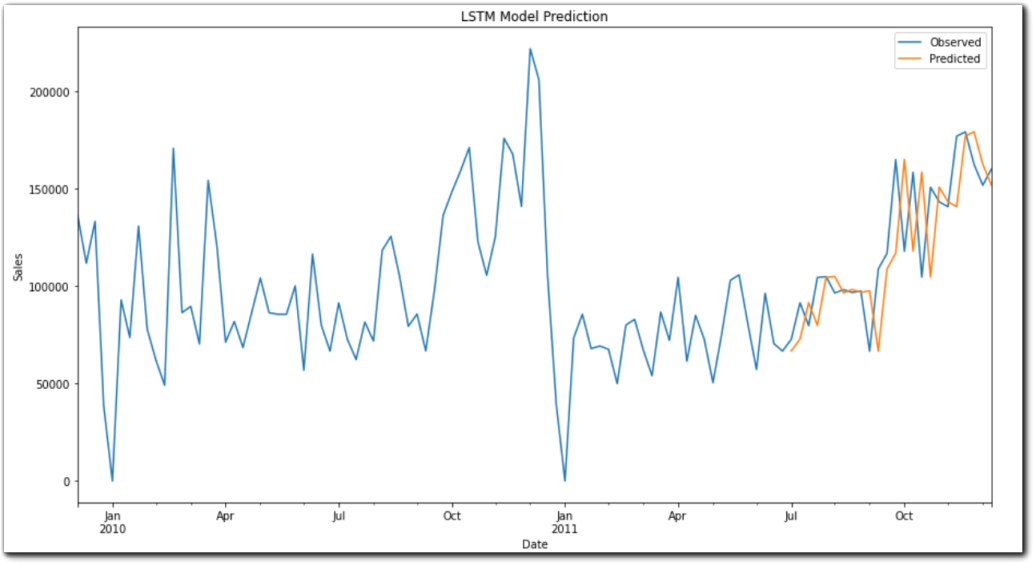


Figure 3.12: LSTM model prediction

As we can see above, the LSTM model is able to capture the trend and the seasonality of the weekly sales time series data with a good performance.

Now we have seen three different approaches to forecast the weekly sales data. Each of the method performs reasonably well on the weekly data.

The complete code is available in the corresponding Notebook in the following link: Notebook Code