15CSE401 - Machine Learning and Data Mining

Case Study - Project Proposal

Team Number: 3

Serial No.	Student Name	Roll Number
1	Dev Mithran J	CB.EN.U4CSE18015
2	Sowmiyanarayan S	CB.EN.U4CSE18053

Project Title: Store Item Demand Forecasting

Introduction:

We have come a long way since the invention of computers. The computer population has skyrocketed in a matter of decades and is deployed in all fields. A lot of modern-day problems require in-depth knowledge of the field and an ability to imbibe the data, interpret it and finally, predict the future outcomes to be able to make the process efficient and profitable to the user. But data is not what it once was. Living amid the data era, we've come to face humongous amounts of data records that can't be made sense of, manually, which is why we have data mining and machine learning to help us in that regard. In this case study, we make use of such techniques to help benefit the storekeepers.

Abstract:

The prosperity of a store lies not only in offering quality products to the customers but also in organizing product supply based on demand. A prudential supply management system would be one designed based on well-substantiated facts. To maximize the benefits in the longer run, one must collect data over a period and mine it to predict the demand trend. The data in the dataset depicts the purchase trend of objects from different stores for 5 years. The objective here is to try and get the sales figures for the next couple of months using a predictive model.

Objective:

The case study aims at maximizing profit as well as minimizing the wastage of resources.

Description:

Many-a-times the shop administrator tends to overbuy items that are purchased rarely and some of them expire which is a blow to sustainability as well as the store profits.

To avoid this, our model studies the recorded product purchase trends, thus understanding the needs of the people, and then goes on to predict the future trends. This is a typical case of a regression problem. The veracity of the model's predictions can be evaluated by taking the deviation between the existing record and the values predicted. The lesser the deviation, the better our results will be.

	store	item	sales
count	913000.000000	913000.000000	913000.000000
mean	5.500000	25.500000	52.250287
std	2.872283	14.430878	28.801144
min	1.000000	1.000000	0.000000
25%	3.000000	13.000000	30.000000
50%	5.500000	25.500000	47.000000
75%	8.000000	38.000000	70.000000
max	10.000000	50.000000	231.000000

- date - holds the dates of the sales.(format: yyyy-mm-dd)
- store - Store ID
- item - Item ID
- sales - Number of items sold at a particular store on a particular date.

Data Collection and Preparation:

For this purpose, we use the demand forecasting dataset from Kaggle which is available at Store Item Demand Forecasting Challenge. The dataset contains 9,13,001 examples each defined by four attributes.

We proceed by reading the CSV file, filling the missing values (if any), and condensing the data by aggregating them into months. We then proceed to scale the data, split the dataset into training and testing datasets, following which we train the model.

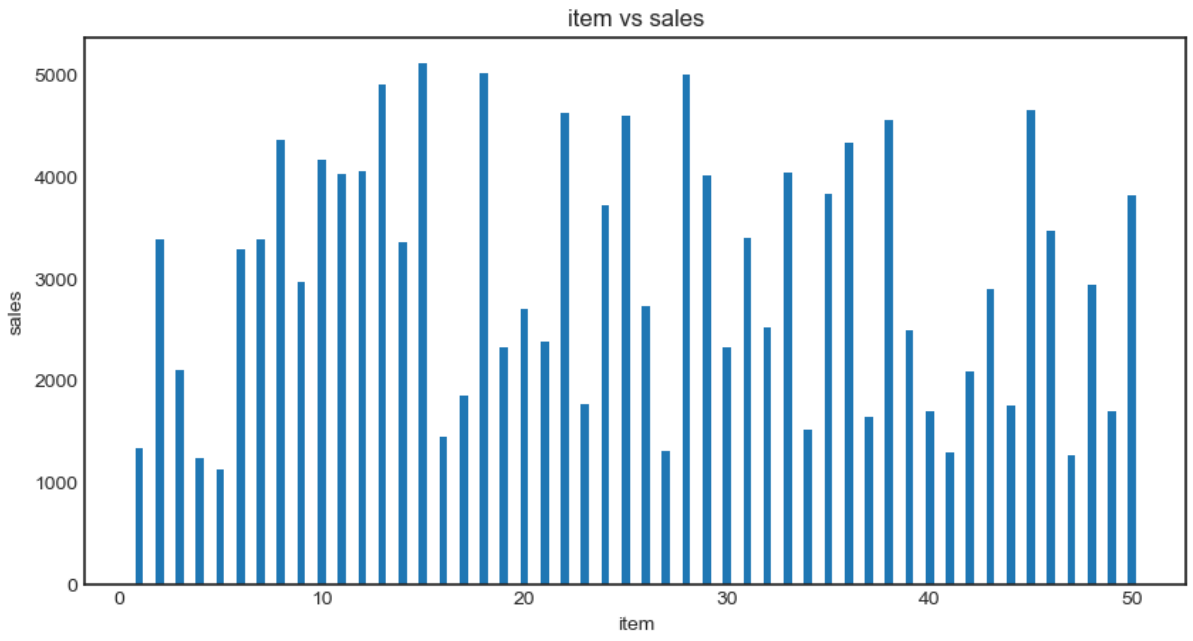
To achieve this, we plan to use Scikit Learn predominantly, followed by other libraries that enable us to use regression models. To better predict the final results, we plan to test the results of multiple prediction models against the ones available and choose the best model before moving on to the forecasts.

Feature Selection / Feature Extraction:

We have to make use of all the attributes offered in the dataset for the best results. Hence, none of the attributes have been removed. On the other hand, we have extracted the month and year out of the date, since we aim to predict how much a shopkeeper must stock up for each month. In doing so, we aggregate the corresponding product sales for each month.

Data Visualization and Hypothesis:

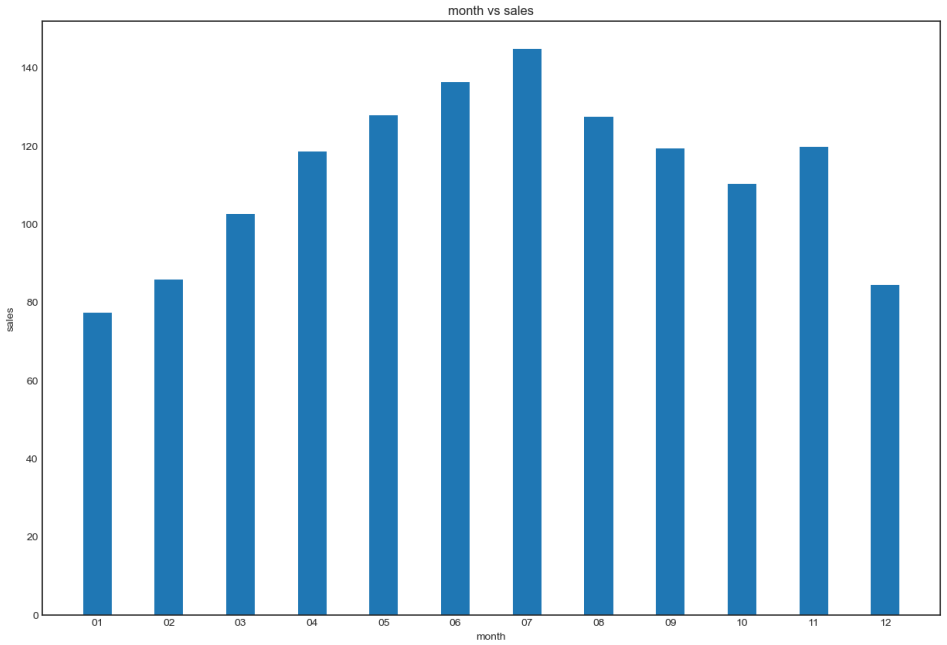
1. **Item vs sales:**
   The items are grouped and their total sales for the cumulative period of sale have been plotted in a bar plot.



Inference:

Here, we could see the preferences of the customer for purchasing a particular product i.e certain products are bought frequently and those that are less preferred.

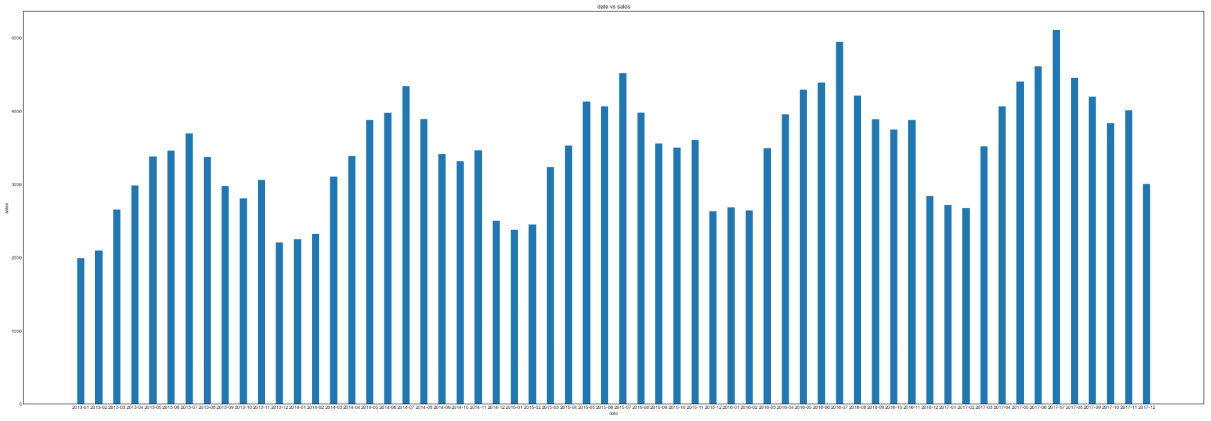
2. **Month vs sales:**
   The sales trends over the years are averaged every month and plotted on a bar graph.



Inference:

Here, we could see that sales figures are at a low at the beginning of the year and gradually rise to reach the highest point in July and, then dip till October and sudden rise in November (maybe due to the holiday season but the dataset does not depend on that) and again dips in December that is towards the end of the year.

3. **Year vs sales:**
   The accumulated sales trends of the years every month and plotted on a bar graph.

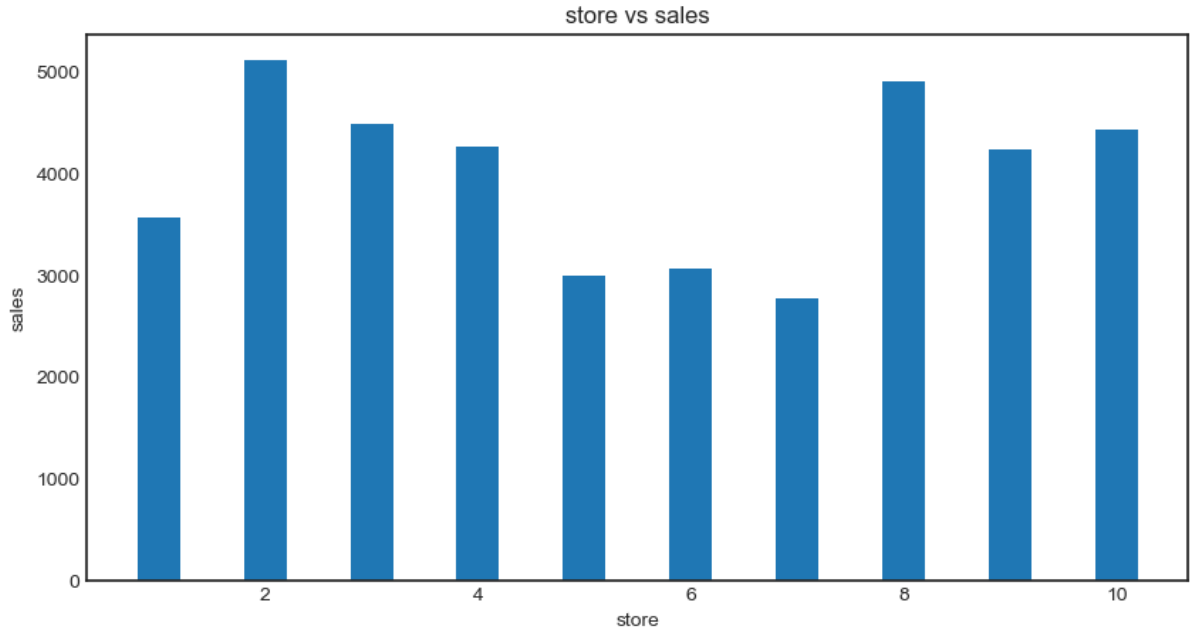


Inference:

Similar to the monthly trends the yearly trends show a similar rise and fall during a particular year but are steadily growing in terms of quantity.

4. **Store vs sales:**

   This plots the sales of different sales on a bar plot.

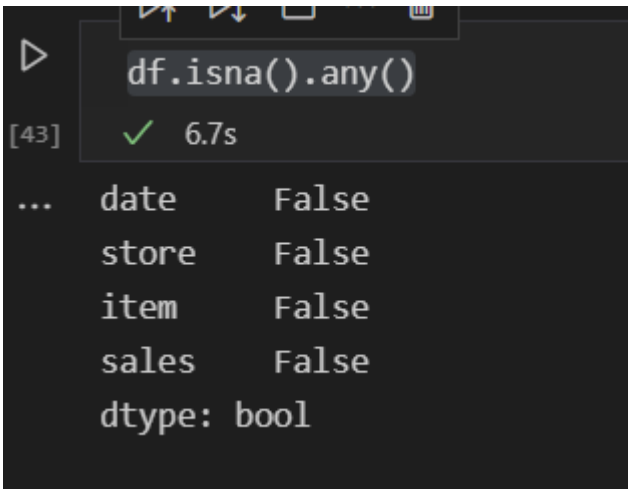


Inference:

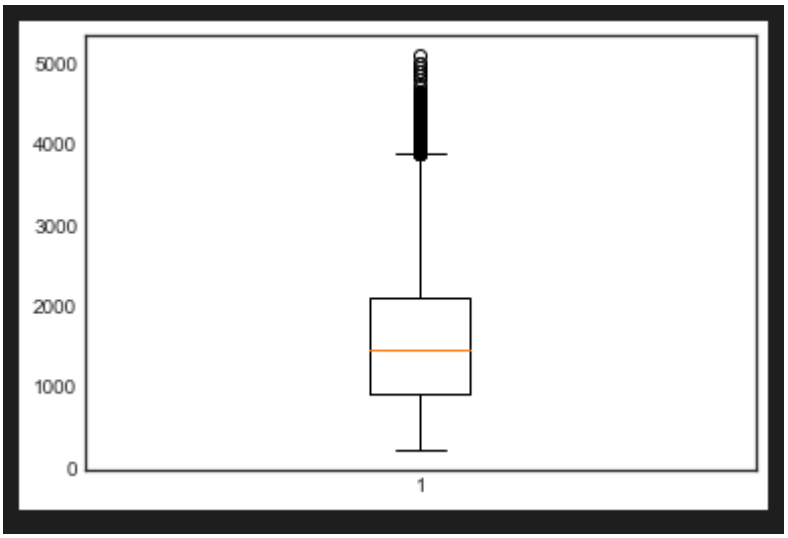
Here, we could see the performance of the stores. Some of the stores like 2 and 8 are high revenue earners while 5,6,7 are low revenue earning stores.

Data Cleaning and Preprocessing:

There were no empty cells in the dataset.



The box plot tells us that rows with sales above 4000 pieces of a product are outliers.



Hence, such rows are removed.

```
df_monthly = df_monthly[df_monthly['sales']<=4000].reset_index().drop('index', axis=1)
```

	date	item	store	sales
0	2013-01	1	1	328
1	2013-01	1	2	486
2	2013-01	1	3	453
3	2013-01	1	4	388
4	2013-01	1	5	294
5	2013-01	1	6	287
6	2013-01	1	7	256
7	2013-01	1	8	431
8	2013-01	1	9	424
9	2013-01	1	10	447

Data Modeling and Inference:

Why XGBoost?

- Yields highly accurate results
- Hardware optimization
- Regularization to avoid overfitting

```
dtrain = xgb.DMatrix(data=x_train, enable_categorical=True, label=y_train)

params = {'objective':'reg:linear',
          'max_depth':2,
          'eta':0.2,
          'silent':1,
          'subsample':1}
num_rounds = 500

xg_depth_20 = xgb.train(params=params, dtrain=dtrain,num_boost_round=num_rounds)
```

[105] ✓ 27.1s Python

```
[19:39:01] WARNING: c:\ci\xgboost-split_1619728435298\work\src\objective\regression_obj.cu:170: reg:linear is now deprecated in favor of
reg:squarederror.
[19:39:01] WARNING: ..\src\learner.cc:541:
Parameters: { silent } might not be used.

  This may not be accurate due to some parameters are only used in language bindings but
  passed down to XGBoost core.  Or some parameters are not used but slip through this
  verification. Please open an issue if you find above cases.
```

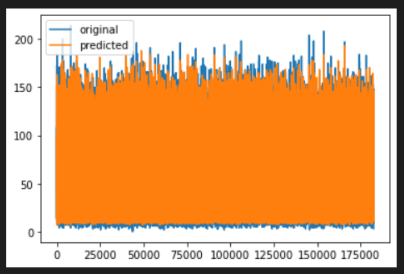
```
dtest = xgb.DMatrix(data=x_test)

train_pred = xg_depth_20.predict(dtest)
print(train_pred)
```

[106] ✓ 0.1s Python

```
[97.74126   39.829147  17.153366  ... 24.339748  28.254974  13.4782095]
```

```
x_ax = range(len(y_test))
plt.plot(x_ax, y_test, label="original")
plt.plot(x_ax, train_pred, label="predicted")
plt.legend()
plt.show()
```

[108] ✓ 1.1s Python



Improvements/ Enhancement of models:

Tried to improve the model by modifying the parameters fed to xgboost.

```
dtrain = xgb.DMatrix(data=x_train, enable_categorical=True, label=y_train)

params = {'objective':'reg:linear',
          'max_depth':3,
          'eta':0.2,
          'silent':1,
          'subsample':1}
num_rounds = 500

xg_depth_20 = xgb.train(params=params, dtrain=dtrain,num_boost_round=num_rounds)
```

[111] ✓ 34.2s Python

```
[19:39:41] WARNING: c:\ci\xgboost-split_1619728435298\work\src\objective\regression_obj.cu:170: reg:linear is now deprecated in favor of
reg:squarederror.
[19:39:41] WARNING: ..\src\learner.cc:541:
Parameters: { silent } might not be used.

  This may not be accurate due to some parameters are only used in language bindings but
  passed down to XGBoost core.  Or some parameters are not used but slip through this
  verification. Please open an issue if you find above cases.
```

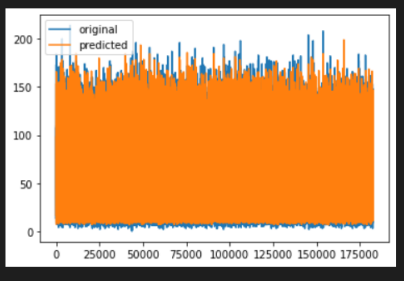
```
dtest = xgb.DMatrix(data=x_test)

train_pred = xg_depth_20.predict(dtest)
print(train_pred)
```

[112] ✓ 0.1s Python

```
[97.7815   40.35874  16.55061  ... 23.902283 29.433537 15.060129]
```

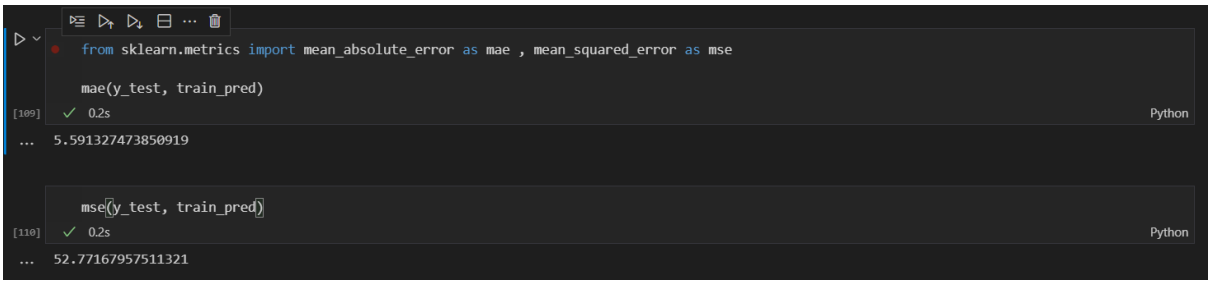
```
x_ax = range(len(y_test))
plt.plot(x_ax, y_test, label="original")
plt.plot(x_ax, train_pred, label="predicted")
plt.legend()
plt.show()
```

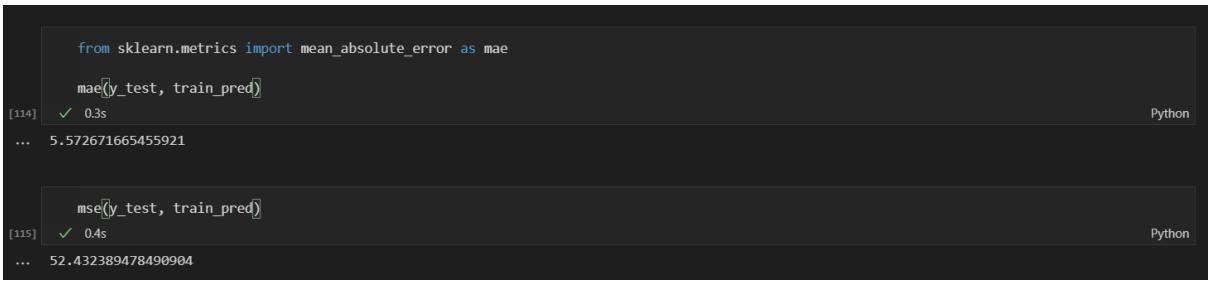
[113] ✓ 1.2s Python



Performance Evaluation and Result Discussion:

Mean Squared Error, Mean Absolute Error





Conclusion and Future Enhancements:

We have successfully planned and implemented the prediction model of our choice, tested the predictions, and used the same to predict the forecast for the next year, thus allowing the user to efficiently stock up goods for every month. This offers the user a two-fold benefit: first in the form of a profit as the excess amount spent in unused stocks can be reduced and next in terms of efficiency by reducing the wastage on a monthly basis.

Going forward, we aim to improve the specificity of the model by including parameters, such as the expiry date of products to make it more consumer-friendly, hence contributing to the store's credibility and goodwill.

Reference:

1. https://scikit-learn.org/dev/tutorial/machine_learning_map/index.html
2. https://mobidev.biz/blog/machine-learning-methods-demand-forecasting-retail
3. https://towardsdatascience.com/fine-tuning-xgboost-in-python-like-a-boss-b4543ed8b1e