Problem statement:

Rossmann operates over 3,000 drug stores in 7 European countries. Currently, Rossmann store managers are tasked with predicting their daily sales for up to six weeks in advance. Store sales are influenced by many factors, including promotions, competition, school and state holidays, seasonality, and locality. With thousands of individual managers predicting sales based on their unique circumstances, the accuracy of results can be quite varied.

1. Import and Describing data

Import data:

We loaded the data from 3 .csv files provided by Kaggle for the competition. We then separated the data into variables, y, which contains the target variable and X, which contains
the explanatory variables. We excluded the rows in training data where store is closed that is the open flag is 0 and also removed rows where sales is less than or equal to zero as they are of no help and will only bias the data while training.

Split the data:

The training set consists of 70 percent of the data while the validation set contains 30 percent of the data. Test data is already provided by Kaggle for evaluation.

Feature Extraction:

Based on existing features calculated some extra empirical features

Parameter Tuning:

We tried using K-Fold nested cross validation in GridSearchCV but since the data is very huge, the models were not able to run even in one day and also gave almost no improvement on the leader board. So, we decided to move forward with holdout validation by splitting the data in training and validation and checking the model's performance on validation data to assess over-fitting or under-fitting. However, for simple models like Linear Regression, Ridge and Lasso, we used GridSearchCV to get best value of parameters.

Evaluation:

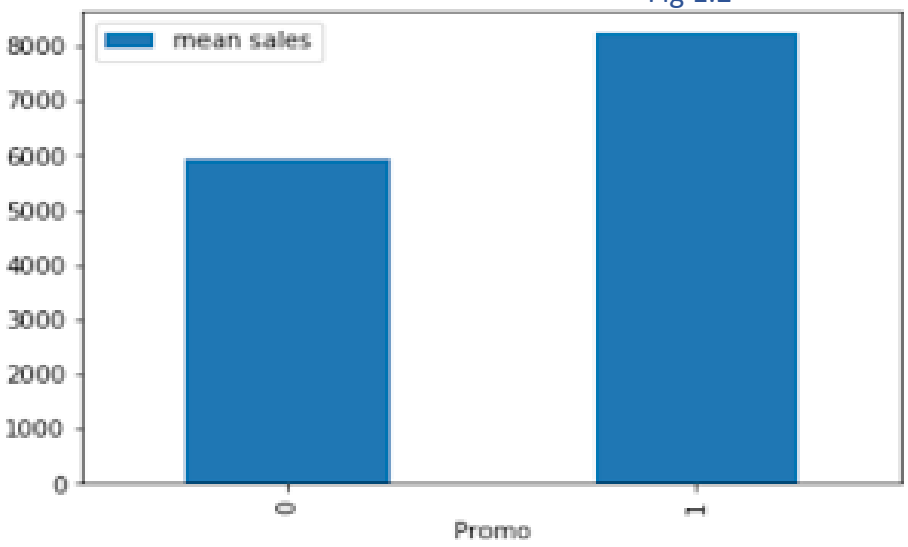
To evaluate different models, we calculated the root mean squared percentage error for each model as this is the criteria that Kaggle was also using to evaluate the model performance. We compared different model using this error metric and selected the one which provided least root mean squared percentage error.

1. Exploratory Data Analysis

Fig 1.1

The above plot explains the average customer count and weekly average sales on each day of the week.

Fig 1.2



The above attached plot tells us the sales got increased when there are promotions on the products, but there is not much difference

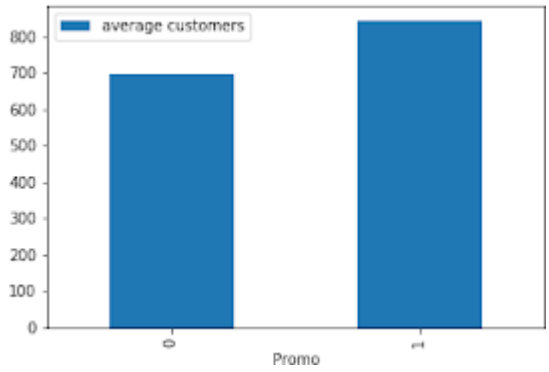


Fig 1.3

This plot shows the proof of customers showing interest to buy the products when there is promos

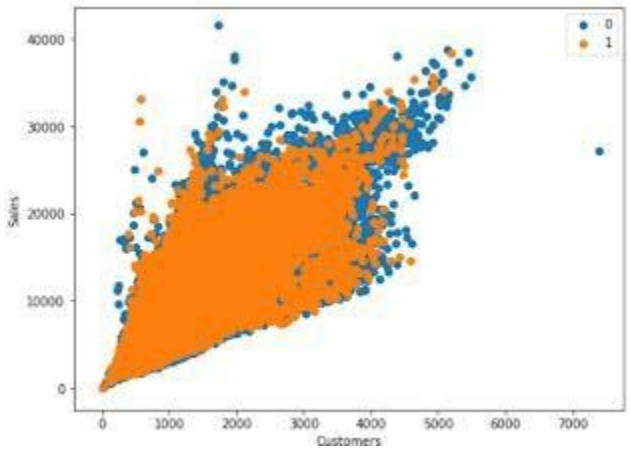


Fig 1.4

In the above scatter plot sales and customers are segregated based on holiday

0 - holiday

1 - not a holiday

By looking the data points distribution, the sales and customers are not getting impacted by the holidays.

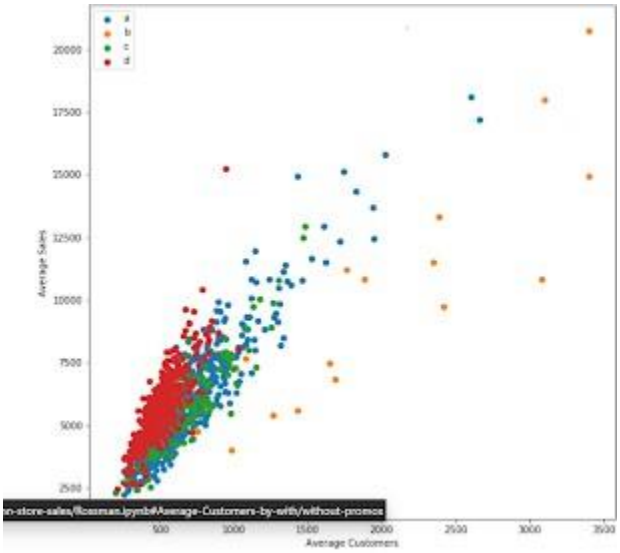


Fig 1.5

In above Figure the data points are less on store type 'b' and from that the conclusion can be made like 'customers spend less' in b, may be some improvements should be made.

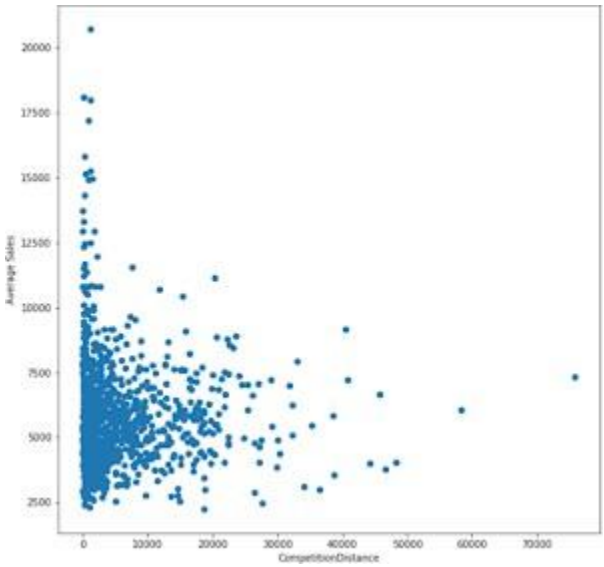


Fig 1.6

Who knew? Closer competition means more profits. (From fig 1.6)

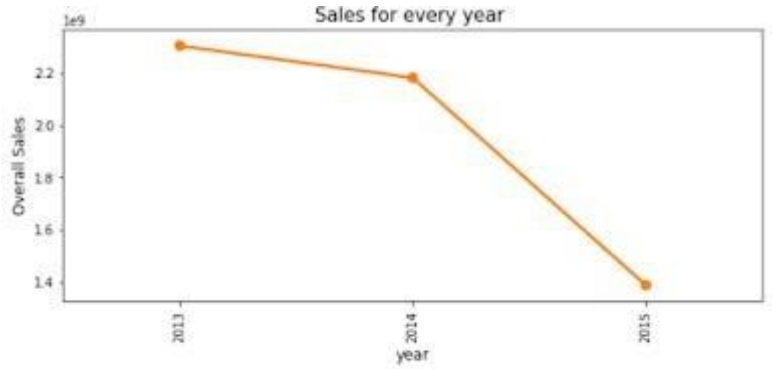


Fig 1.7

The sales are gradually falling down from both of the above trend lines

2. Results:

Linear Regression

The model fitted the data very well with the basic parameters. Below is the following output:

```
: print(f'RMSE for train set with linear regression: {np.sqrt(mean_squared_error(y_train , train_pred))}')
  print(f'RMSE for validation with linear regression: {np.sqrt(mean_squared_error(y_eval , y_pred_linear))}')
  print(f'R-square for validation in linear regression: {r2_score(y_eval, y_pred_linear)}')
  print(f'R-square for train set in linear regression: {r2_score(y_train, train_pred)}')

RMSE for train set with linear regression: 2.004095701657737
RMSE for validation with linear regression: 1.9988355179153638
R-square for validation in linear regression: 0.6348962099922618
R-square for train set in linear regression: 0.6342322751875119
```

Fig 1.8

Decision Tree Model

For this model parameter tuning is done using Grid Search and trained the data. The output screenshot is attached below:

```
: from sklearn.metrics import r2_score , mean_squared_error
  # R-square
  print(f'R-square for train with tree: {r2_score(y_train , train_pred_dt)}')
  print(f'R-square for validation with tree: {r2_score(y_eval , y_pred_dt)}')
  # MSE
  print(f'MSE for train with tree: {np.sqrt(mean_squared_error(y_train , train_pred_dt))}')
  print(f'MSE for validation with tree: {np.sqrt(mean_squared_error(y_eval , y_pred_dt))}')

R-square for train with tree: 0.7515548349088375
R-square for validation with tree: 0.7538006809291129
MSE for train with tree: 1.6516995135774253
MSE for validation with tree: 1.6413919950070008
```

Fig 1.9

Artificial Neural Network

This neural net is black box but can tackle the non-linearity, for this data Linear regression is performing well even though to check how the nets are learning parameters. The performance plots are below
We considered the following options to tune our multi-layer perceptron model:

1. Number of layers
2. Number of neurons in each layer
3. Activation function: ReLu, Sigmoid, Tanh (for classification) and linear (for regression)
4. Kernel initializer = Normal, Uniform
5. Optimizer: SGD, Adam
6. Epoch count and batch size (15 and 10,000 respectively)

After trying many combinations of the above-mentioned parameters, we came up with the final model that have 5 hidden layers with layers having 60, 90, 60, 90, 60 and 1 as output neuron. For model training, we used ReLu activation in all the layers with uniform initialization and adam optimizer optimizing custom function of 'MSE' as loss metric. We also added dropout layers for every hidden layer.
The performance plots are attached below:

The first plot shows the performance on loss based on MSE loss function

The second plot shows the performance on loss-based MAE loss function

```
plt.plot(history.history['mean_squared_error'])
```

```
[<matplotlib.lines.Line2D at 0x15cad638a20>]
```

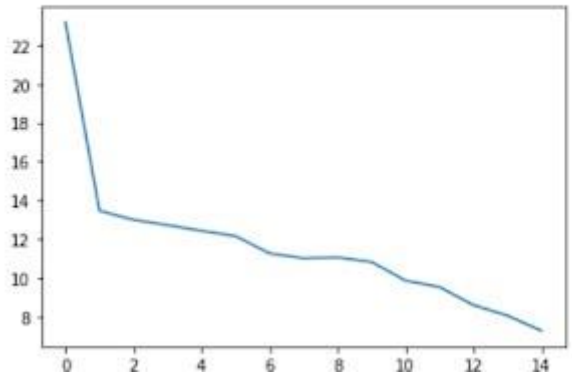


Fig 1.9 (a)

```
: plt.plot(history.history['mean_absolute_error'])
```

```
: [<matplotlib.lines.Line2D at 0x15cad126eb8>]
```

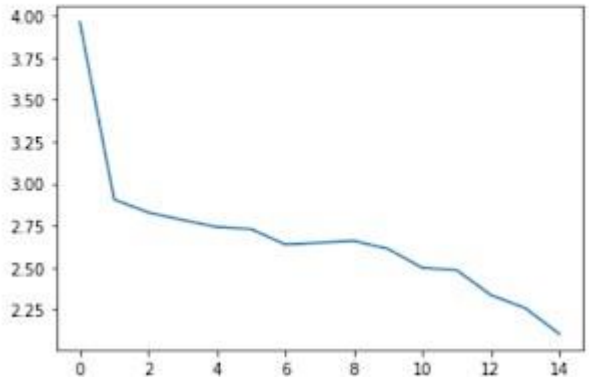


Fig 1.9 (b)