“FUTURE SALES PREDICTION”

A Report

Submitted as special assignment

of

2EI503 MACHINE LEARNING

By
(PRAKHAR AGARWAL)
(21bei040))

Under the Guidance of
Prof. Harsh Kapadia

INSTRUMENTATION AND CONTROL ENGINEERING

INSTITUTE OF TECHNOLOGY

NIRMA UNIVERSITY

Ahmedabad 382 481

NOVEMBER 2023

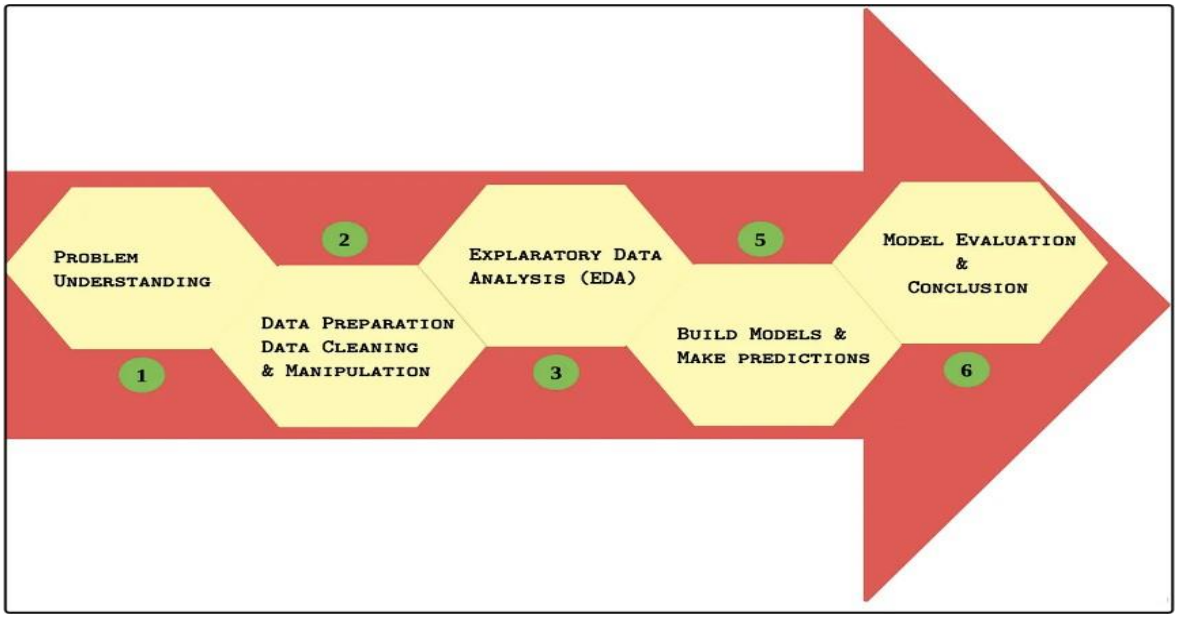
INTRODUCTION

Forecasting future sales of a product offers many advantages. Predicting future sales of a product helps a company manage the cost of manufacturing and marketing the product. In this notebook, I will try to you through the task of future sales prediction with machine learning using Python.

The dataset used in this project is the **Future Sales Prediction dataset** from the Kaggle dataset.

The evaluation process involved the utilization of distinct machine learning algorithms, each with its unique approach to learning patterns and making predictions. The models chosen for evaluation encompassed a diverse array of methodologies, including Random Forest, Logistic Regression, LSTM

BLOCK DIAGRAM



DATASET

We start the project by loading the dataset in our Jupyter notebook. We load the dataset into a Pandas dataframe named df. The first step is to load the data and transform it into a structure that we will then use for each of our models. In its raw form, each row of data represents a single day of sales at one of ten stores. Our goal is to predict monthly sales, so we will first consolidate all stores and days into total monthly sales.

A	B	C	D	E
date	store	item	sales	
01-01-2013	1	1	13	
02-01-2013	1	1	11	
03-01-2013	1	1	14	
04-01-2013	1	1	13	
05-01-2013	1	1	10	
06-01-2013	1	1	12	
07-01-2013	1	1	10	
08-01-2013	1	1	9	
09-01-2013	1	1	12	
10-01-2013	1	1	9	
11-01-2013	1	1	9	
12-01-2013	1	1	7	
13-01-2013	1	1	10	
14-01-2013	1	1	12	
15-01-2013	1	1	5	
16-01-2013	1	1	7	
17-01-2013	1	1	16	
18-01-2013	1	1	7	
19-01-2013	1	1	18	
20-01-2013	1	1	15	
21-01-2013	1	1	8	
22-01-2013	1	1	7	
23-01-2013	1	1	9	
24-01-2013	1	1	8	
25-01-2013	1	1	14	
26-01-2013	1	1	12	
27-01-2013	1	1	12	
28-01-2013	1	1	11	

DATASET VISUALIZATION

First the data set is visualized in various forms using bar graphs and line plots to see the sales per store and the sales per month.

10]:
```
def monthlyORyears_sales(data,time=['monthly','years']):
    data = data.copy()
    if time == "monthly":
        # Drop the day indicator from the date column:
        data.date = data.date.apply(lambda x: str(x)[:-3])
    else:
        data.date = data.date.apply(lambda x: str(x)[:4])

   # Sum sales per month:
    data = data.groupby('date')['sales'].sum().reset_index()
    data.date = pd.to_datetime(data.date)

    return data
m_df = monthlyORyears_sales(df_s,"monthly")

m_df.to_csv('./monthly_data.csv')
m_df.head(10)
```

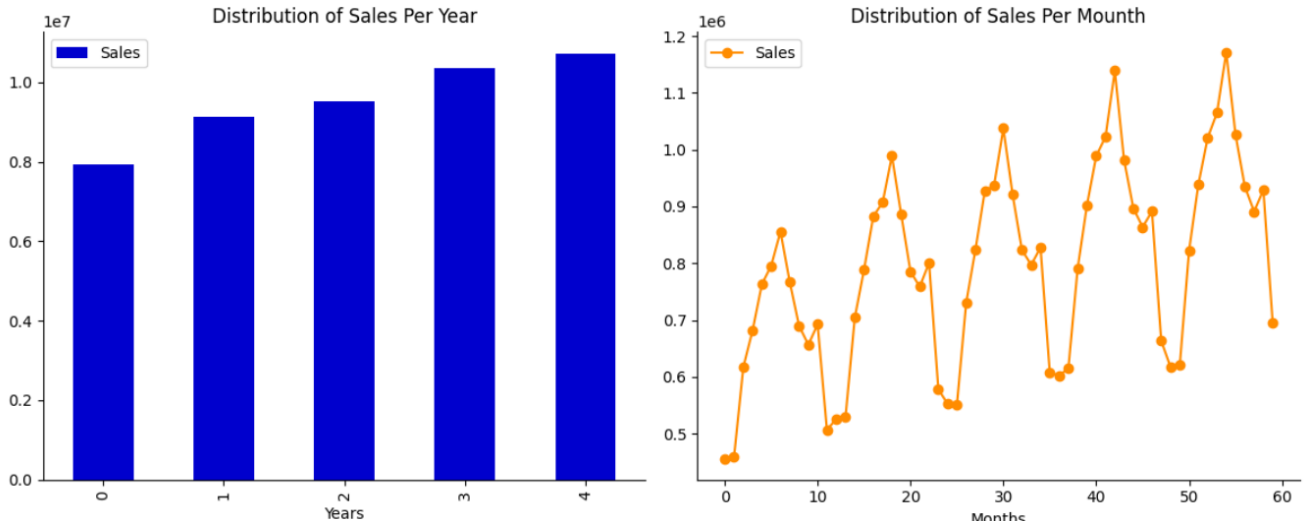
10]:

	date	sales
0	2013-01-01	454904
1	2013-02-01	459417
2	2013-03-01	617382
3	2013-04-01	682274
4	2013-05-01	763242
5	2013-06-01	795597
6	2013-07-01	855922
7	2013-08-01	766761
8	2013-09-01	689907
9	2013-10-01	656587

11]:
```
y_df = monthlyORyears_sales(df_s,"years")
y_df
```

```
layout = (1, 2)

raw = plt.subplot2grid(layout, (0 ,0))
law = plt.subplot2grid(layout, (0 ,1))

years = y_df['sales'].plot(kind = "bar",color = 'mediumblue', label="Sales",ax=raw, figsize=(12,5))
months = m_df['sales'].plot(marker = 'o',color = 'darkorange', label="Sales", ax=law)

years.set(xlabel = "Years",title = "Distribution of Sales Per Year")
months.set(xlabel = "Months", title = "Distribution of Sales Per Mounth")

sns.despine()
plt.tight_layout()

years.legend()
months.legend()
```

```
<matplotlib.legend.Legend at 0x251fcb42fa0>
```



Now sales per store is calculated and displayed using bar Graph.

4]:
```
def sales_per_store(data):
    sales_by_store = data.groupby('store')['sales'].sum().reset_index()

    fig, ax = plt.subplots(figsize=(8,6))
    sns.barplot(x=sales_by_store.store, y= sales_by_store.sales, color='darkred')

    ax.set(xlabel = "Store Id", ylabel = "Sum of Sales", title = "Total Sales Per Store")

    return sales_by_store
sales_per_store(df_s)
```

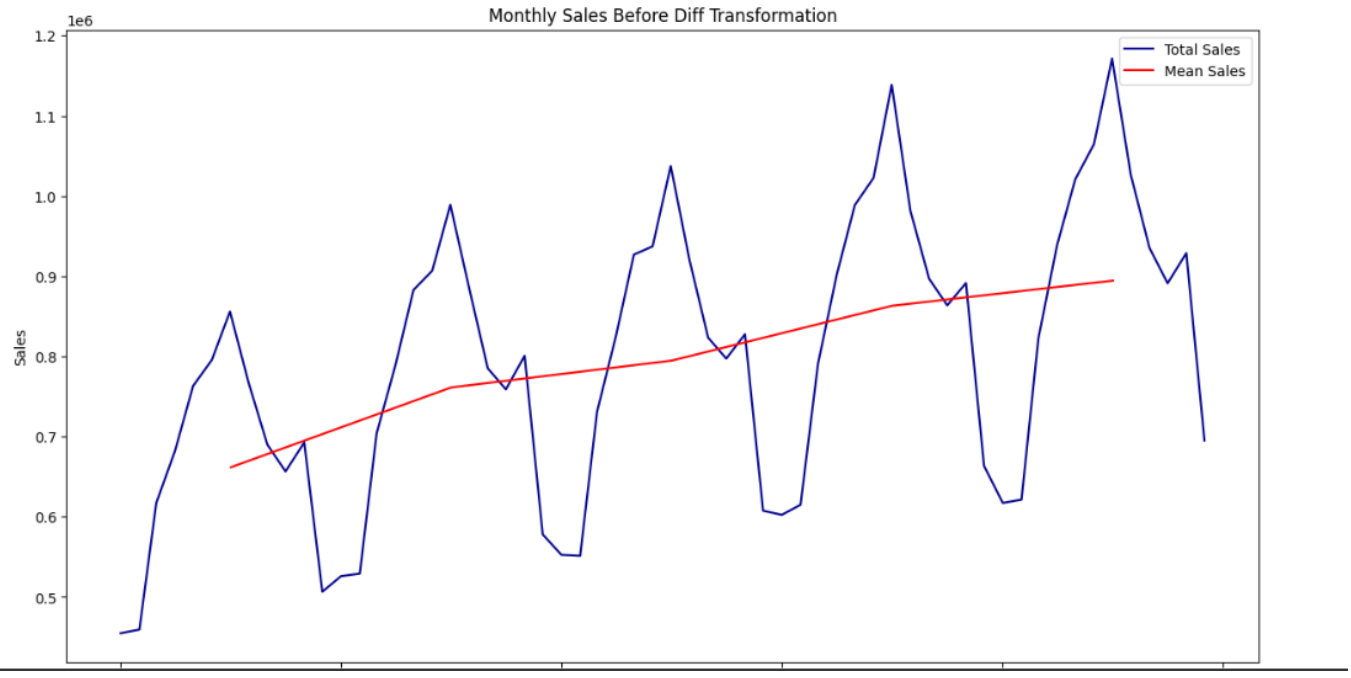
4]:

	store	sales
0	1	4315603
1	2	6120128
2	3	5435144
3	4	5012639
4	5	3631016
5	6	3627670
6	7	3320009
7	8	5856169
8	9	5025976
9	10	5360158

Determining Time Series Stationary

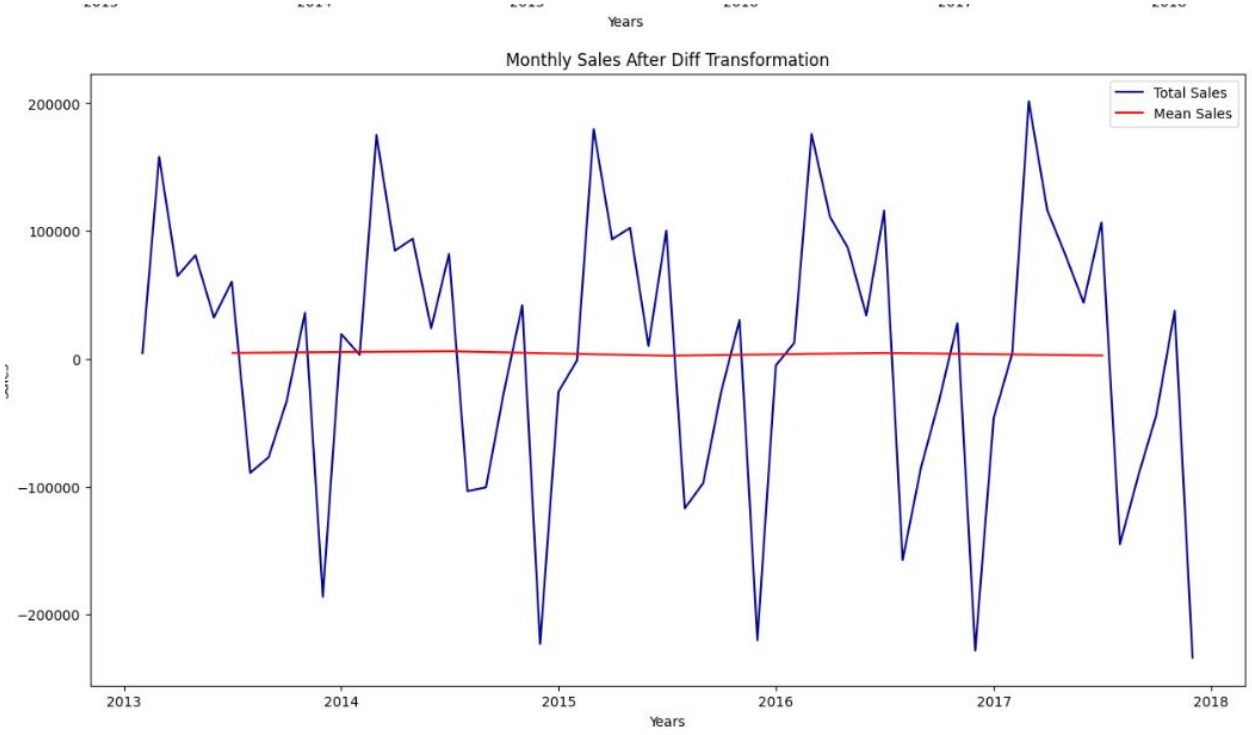
The underlying principle is to model or estimate the trend and seasonality in the series and remove those from the series to get a stationary series. Then statistical forecasting techniques can be implemented in this series. The final step would be to convert the forecasted values into the original scale by applying trend and seasonality constraints back.

```
6]: def time_plot(data, x_col, y_col, title):
        fig, ax = plt.subplots(figsize = (15,8))
        sns.lineplot(x=x_col, y=y_col, data = data, ax = ax, color = 'darkblue', label='Total Sales')

        s_mean = data.groupby(data.date.dt.year)[y_col].mean().reset_index()
        s_mean.date = pd.to_datetime(s_mean.date, format='%Y')
        sns.lineplot(x=(s_mean.date + datetime.timedelta(6*365/12)),y= y_col, data=s_mean, ax=ax, color='red', label='Mean Sales')

        ax.set(xlabel = "Years",
               ylabel = "Sales",
               title = title)
    time_plot(m_df, 'date', 'sales', 'Monthly Sales Before Diff Transformation' )

    def get_diff(data):
        """Calculate the difference in sales month over month:"""

        data['sales_diff'] = data.sales.diff()
        data = data.dropna()

        data.to_csv('./stationary_df.csv')

        return data
    stationary_df = get_diff(m_df)
    time_plot(stationary_df, 'date', 'sales_diff', 'Monthly Sales After Diff Transformation')
```



Differencing

In this method, we compute the difference of consecutive terms in the series. Differencing is typically performed to get rid of the varying mean.



]:
```
def built_supervised(data):
    supervised_df = data.copy()

    # Create column for each lag:
    for i in range(1, 13):
        col_name = 'lag_' + str(i)
        supervised_df[col_name] = supervised_df['sales_diff'].shift(i)

    # Drop null values:
    supervised_df = supervised_df.dropna().reset_index(drop=True)

    supervised_df.to_csv('./model_df.csv', index=False)

    return supervised_df
model_df = built_supervised(stationary_df)
model_df
```

]:

	date	sales	sales_diff	lag_1	lag_2	lag_3	lag_4	lag_5	lag_6	lag_7	lag_8	lag_9	lag_10	lag_11	lag_12
0	2014-02-01	529117	3130.0	19380.0	-186036.0	36056.0	-33320.0	-76854.0	-89161.0	60325.0	32355.0	80968.0	64892.0	157965.0	4513.0
1	2014-03-01	704301	175184.0	3130.0	19380.0	-186036.0	36056.0	-33320.0	-76854.0	-89161.0	60325.0	32355.0	80968.0	64892.0	157965.0
2	2014-04-01	788914	84613.0	175184.0	3130.0	19380.0	-186036.0	36056.0	-33320.0	-76854.0	-89161.0	60325.0	32355.0	80968.0	64892.0
3	2014-05-01	882877	93963.0	84613.0	175184.0	3130.0	19380.0	-186036.0	36056.0	-33320.0	-76854.0	-89161.0	60325.0	32355.0	80968.0
4	2014-06-01	906842	23965.0	93963.0	84613.0	175184.0	3130.0	19380.0	-186036.0	36056.0	-33320.0	-76854.0	-89161.0	60325.0	32355.0
5	2014-07-01	989010	82168.0	23965.0	93963.0	84613.0	175184.0	3130.0	19380.0	-186036.0	36056.0	-33320.0	-76854.0	-89161.0	60325.0
6	2014-08-01	885596	-103414.0	82168.0	23965.0	93963.0	84613.0	175184.0	3130.0	19380.0	-186036.0	36056.0	-33320.0	-76854.0	-89161.0
7	2014-09-01	785124	-100472.0	-103414.0	82168.0	23965.0	93963.0	84613.0	175184.0	3130.0	19380.0	-186036.0	36056.0	-33320.0	-76854.0
8	2014-10-01	758883	-26241.0	-100472.0	-103414.0	82168.0	23965.0	93963.0	84613.0	175184.0	3130.0	19380.0	-186036.0	36056.0	-33320.0
9	2014-11-01	800783	41900.0	-26241.0	-100472.0	-103414.0	82168.0	23965.0	93963.0	84613.0	175184.0	3130.0	19380.0	-186036.0	36056.0
10	2014-12-01	578048	-222735.0	41900.0	-26241.0	-100472.0	-103414.0	82168.0	23965.0	93963.0	84613.0	175184.0	3130.0	19380.0	-186036.0
11	2015-01-01	552513	-25535.0	-222735.0	41900.0	-26241.0	-100472.0	-103414.0	82168.0	23965.0	93963.0	84613.0	175184.0	3130.0	19380.0
12	2015-02-01	551317	-1196.0	-25535.0	-222735.0	41900.0	-26241.0	-100472.0	-103414.0	82168.0	23965.0	93963.0	84613.0	175184.0	3130.0
13	2015-03-01	730951	179634.0	-1196.0	-25535.0	-222735.0	41900.0	-26241.0	-100472.0	-103414.0	82168.0	23965.0	93963.0	84613.0	175184.0
14	2015-04-01	824467	93516.0	179634.0	-1196.0	-25535.0	-222735.0	41900.0	-26241.0	-100472.0	-103414.0	82168.0	23965.0	93963.0	84613.0

DATASET PREPROCESSING

The initial step in the data preprocessing involved loading the dataset from a csv file using pandas. Subsequently, the dataset was divided into feature variables (x) and the target variable (y), to standardize the features, the dataset was split into training and testing sets using a ratio, and the standardscaler from scikit- learn was applied to scale the features. a random forest classifier model was instantiated and trained using the training data. finally, predictions were made on thetest set using the trained model. this data preprocessing pipeline comprised data loading, categorical variable encoding, feature-target split, feature scaling, and the utilization of the random forest classifier as the predictive model for this dataset. overall, these steps formed a structured framework leading up to the model training and prediction phase. Unscaling is performed to visualize the original data as present in the dataset.

```python
def train_test_split(data):
    data = data.drop(['sales','date'], axis=1)
    train , test = data[:-12].values, data[-12:].values

    return train, test

train, test = train_test_split(model_df)
def scale_data(train_set,test_set):
    """Scales data using MinMaxScaler and separates data into X_train, y_train,
    X_test, and y_test."""

    # Apply Min Max Scaler:
    scaler = MinMaxScaler(feature_range=(-1, 1))
    scaler = scaler.fit(train_set)

    # Reshape training set:
    train_set = train_set.reshape(train_set.shape[0],
                                  train_set.shape[1])
    train_set_scaled = scaler.transform(train_set)

    # Reshape test set:
    test_set = test_set.reshape(test_set.shape[0],
                                test_set.shape[1])
    test_set_scaled = scaler.transform(test_set)

    X_train, y_train = train_set_scaled[:, 1:], train_set_scaled[:, 0:1].ravel() # returns the
    X_test, y_test = test_set_scaled[:, 1:], test_set_scaled[:, 0:1].ravel()

    return X_train, y_train, X_test, y_test, scaler


X_train, y_train, X_test, y_test, scaler_object = scale_data(train, test)
def re_scaling(y_pred, x_test, scaler_obj, lstm=False):
    """For visualizing and comparing results, undoes the scaling effect on predictions."""
   # y_pred: model predictions
   # x_test: features from the test set used for predictions
   # scaler_obj: the scaler objects used for min-max scaling
   # lstm: indicate if the model run is the lstm. If True, additional transformation occurs

    # Reshape y_pred:
    y_pred = y_pred.reshape(y_pred.shape[0],
                            1,
                            1)

    if not lstm:
        x_test = x_test.reshape(x_test.shape[0],
                                1,
                                x_test.shape[1])

    # Rebuild test set for inverse transform:
    pred_test_set = []
    for index in range(0, len(y_pred)):
        pred_test_set.append(np.concatenate([y_pred[index],
                                             x_test[index]],
                                             axis=1) )

    # Reshape pred_test_set:
    pred_test_set = np.array(pred_test_set)
    pred_test_set = pred_test_set.reshape(pred_test_set.shape[0],
                                          pred_test_set.shape[2])

    # Inverse transform:
    pred_test_set_inverted = scaler_obj.inverse_transform(pred_test_set)
```

MODEL TRAINING

Now we have our dataset ready for building a machine learning-based classifier. There are several classification models that can be used for this task. In this analysis, we will build three different types of classification models namely Logistic Regression,Decision Tree, LSTM. These are the most popular models usedfor solving classification problems.

```
def prediction_df(unscale_predictions, origin_df):
    """Generates a dataframe that shows the predicted sales for each month
    for plotting results."""

    # unscale_predictions: the model predictions that do not have min-max or other scaling a
    # origin_df: the original monthly sales dataframe

    # Create dataframe that shows the predicted sales:
    result_list = []
    sales_dates = list(origin_df[-13:].date)
    act_sales = list(origin_df[-13:].sales)

    for index in range(0, len(unscale_predictions)):
        result_dict = {}
        result_dict['pred_value'] = int(unscale_predictions[index][0] + act_sales[index])
        result_dict['date'] = sales_dates[index + 1]
        result_list.append(result_dict)

    df_result = pd.DataFrame(result_list)

    return df_result
```

```
model_scores = {}
def get_scores(unscale_df, origin_df, model_name):
    """Prints the root mean squared error, mean absolute error, and r2 scores
    for each model. Saves all results in a model_scores dictionary for
    comparison."""

    rmse = np.sqrt(mean_squared_error(origin_df.sales[-12:],
                                      unscale_df.pred_value[-12:]))

    mae = mean_absolute_error(origin_df.sales[-12:],
                              unscale_df.pred_value[-12:])

    r2 = r2_score(origin_df.sales[-12:],
                  unscale_df.pred_value[-12:])

    model_scores[model_name] = [rmse, mae, r2]

    print(f"RMSE: {rmse}\nMAE: {mae}\nR2 Score: {r2}")
```

```
def plot_results(results, origin_df, model_name):
# results: a dataframe with unscaled predictions

    fig, ax = plt.subplots(figsize=(15,5))
    sns.lineplot(x=origin_df.date, y=origin_df.sales, data=origin_df, ax=ax,
                 label='Original', color='blue')
    sns.lineplot(x=results.date, y=results.pred_value, data=results, ax=ax,
                 label='Predicted', color='red')


    ax.set(xlabel = "Date",
           ylabel = "Sales",
           title = f"{model_name} Sales Forecasting Prediction")

    ax.legend(loc='best')

    filepath = Path('./model_output/{model_name}_forecasting.svg')
    filepath.parent.mkdir(parents=True, exist_ok=True)
    plt.savefig(f'./model_output/{model_name}_forecasting.svg')
def regressive_model(train_data, test_data, model, model_name):
    """Runs regressive models in SKlearn framework. First calls scale_data
```

a) Logistic Regression:

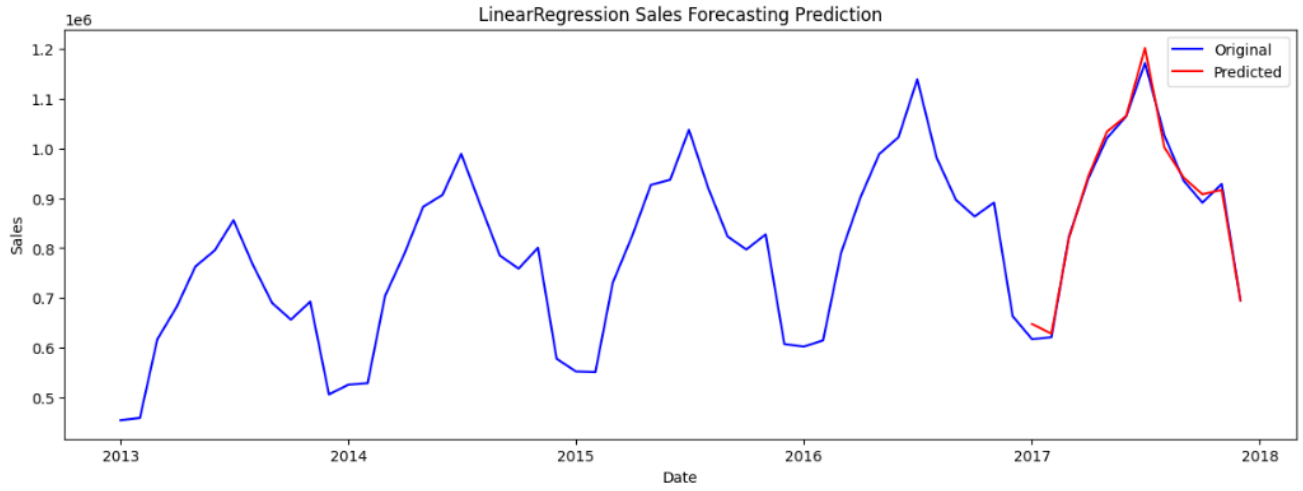
Before implementing logistic regression, we scale the feature variables of our dataset using sklearn's **MinMaxScaler method**. We train the Logistic Regression model with standard parameters using the training dataset. The trained model is saved as logreg.

```
def regressive_model(train_data, test_data, model, model_name):
    """Runs regressive models in Sklearn framework. First calls scale_data
    to split into X and y and scale the data. Then fits and predicts. Finally,
    predictions are unscaled, scores are printed, and results are plotted and
    saved."""

    # Split into X & y and scale data:
    X_train, y_train, X_test, y_test, scaler_object = scale_data(train_data,
                                                                 test_data)

    # Run sklearn models:
    mod = model
    mod.fit(X_train, y_train)
    predictions = mod.predict(X_test) # y_pred=predictions

    # Undo scaling to compare predictions against original data:
    origin_df = m_df
    unscaled = re_scaling(predictions, X_test, scaler_object) # unscaled_prediction
    unscaled_df = prediction_df(unscaled, origin_df)

    # Print scores and plot results:
    get_scores(unscaled_df, origin_df, model_name)
    plot_results(unscaled_df, origin_df, model_name)
```

```
regressive_model(train, test, LinearRegression(), 'LinearRegression')
```

```
RMSE: 16221.040790693221
MAE: 12433.0
R2 Score: 0.9907155879704752
```

we evaluate the performance of our model using the test dataset. we use the metric classification accuracy defined as the fraction of times model prediction matches the value of the target variable. for a detailed evaluation of our model, we look at the confusion matrix. the values in the diagonal of the confusion matrix denote the fraction of correct rejection (first-row first entry) or correct approval (second-row second entry) predictions by our classification model. our logistic regression model has a classification accuracy of 99.07 %.

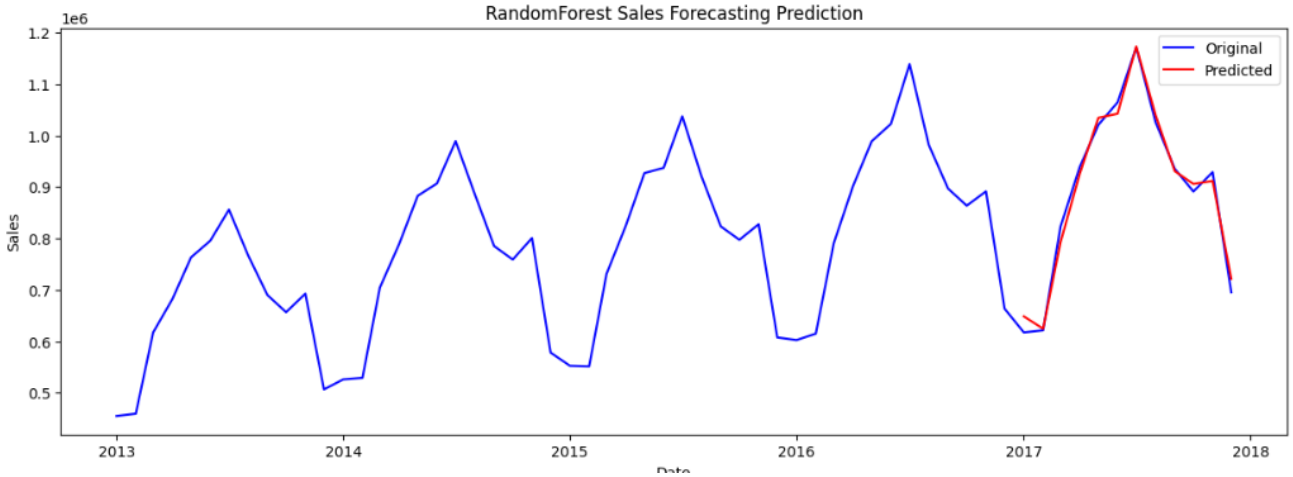


b) Decision Tree

The second model we try for our classification task is the Decision tree model. We have used sklearn's **DecisionTreeClassifier** algorithm to build the model. We find the optimized value of hyperparameter **max_depth** by varying it between 1 and 10 in steps of 1. max_depth value decides the number of times a decision tree is allowed to split. In the plot of Accuracy vs Depth for train and test data, we see for max_depth =3 both train and test accuracy are the same. We choose this value for our model as it avoids a model that is either **overfitted or underfittted. The final test accuracy score of our decision tree model is 98.7%**

```
regressive_model(train, test, RandomForestRegressor(n_estimators=100, max_depth=20),
          'RandomForest')
```

```
RMSE: 18647.048086314717
MAE: 16001.666666666666
R2 Score: 0.9877307743373955
```



C) LSTM

The third model used is lstm that is a kind of RNN model and it gives us a accuracy of 99.35%

```
5]: def lstm_model(train_data, test_data):
        """Runs a long-short-term-memory neural net with 2 dense layers.
        Generates predictions that are then unscaled.
        Scores are printed and the results are plotted and saved."""
        # train_data: dataset used to train the model
        # test_data: dataset used to test the model


        # Split into X & y and scale data:
        X_train, y_train, X_test, y_test, scaler_object = scale_data(train_data, test_data)

        X_train = X_train.reshape(X_train.shape[0], 1, X_train.shape[1])
        X_test = X_test.reshape(X_test.shape[0], 1, X_test.shape[1])


        # Build LSTM:
        model = Sequential()
        model.add(LSTM(4, batch_input_shape=(1, X_train.shape[1], X_train.shape[2]),
                       stateful=True))
        model.add(Dense(1))
        model.add(Dense(1))
        model.compile(loss='mse', optimizer='adam', metrics=['accuracy'])
        model.fit(X_train, y_train, epochs=50, batch_size=1, verbose=1,
                  shuffle=False)
        predictions = model.predict(X_test,batch_size=1)

        # Undo scaling to compare predictions against original data:
        origin_df = m_df
        unscaled = re_scaling(predictions, X_test, scaler_object, lstm=True)
        unscaled_df = prediction_df(unscaled, origin_df)

        get_scores(unscaled_df, origin_df, 'LSTM')

    lstm_model(train,test)
```

```
RMSE: 13556.643313027995
MAE: 11657.25
R2 Score: 0.99351512943978
```

CONCLUSION:

We have tried four different classification models for our credit card approval prediction task. The train and test accuracy of the models is summarized in the Figure below. We have obtained the best test data accuracy (99.35%) from the LSTM classifier. The small difference in train and test accuracy scores indicatesthe absence of overfitting and underfitting.