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.

IMPORT LIBRARIES AND DATASET :

```
import numpy as np
import pandas as pd
import datetime
import matplotlib.pyplot as plt
import seaborn as sns

import sklearn
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import RidgeCV
from sklearn.linear_model import Lasso
from sklearn.linear_model import LassoCV
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
from math import sqrt
from sklearn.metrics import mean_squared_error
from sklearn import metrics
```

Let us first summarize the past observations from PHASE 2 before proceeding with PHASE 3 documentation :

1) A strong positive correlation exists between the amount of Sales and Customers of a store.

2) A positive correlation exists between the stores that had a Promo equal to 1(stores running promotions) and amount of Customers.

3) For observations with Promo2 equal to 1 (that is for consecutive promotion), the number of Customers and Sales has a negative correlation.

4) Sales and Number of customers peak around christmas timeframe.

5) Minimum number of customers are generally around the 24th of the month.

6) Most customers and Sales are around 30th and 1st of the month.

7) Minimum number of customers are generally on Saturday.

8) Most customers and Sales are on Sunday and Monday.

PHASE 3 DOCUMENTATION :

Handling Categorical features :

Feature : StateHoliday

```
set(sales_train_all_df.StateHoliday)
```

{0, '0', 'a', 'b', 'c'}

Let us encode the stateHoliday similarity as follows since only 0 would mean a Non-holiday and rest are holidays :

```
sales_train_all_df["StateHoliday"] = sales_train_all_df["StateHoliday"].map({0: 0, "0": 0, "a": 1, "b": 1, "c": 1})
sales_train_all_df['StateHoliday'] = sales_train_all_df['StateHoliday'].astype(int)
```

```
sales_train_all_df["StateHoliday"].unique()
```

array([0, 1])

Feature : PromoInterval

```
set(sales_train_all_df.PromoInterval)
```

{0, 'Feb,May,Aug,Nov', 'Jan,Apr,Jul,Oct', 'Mar,Jun,Sept,Dec'}

```
sales_train_all_df["PromoInterval"] = sales_train_all_df["PromoInterval"].map({0: 0, "0":0, "Jan,Apr,Jul,Oct":1, "Feb,May,Aug,Nov":2, "Mar,Jun,Sept,Dec":3})
sales_train_all_df['PromoInterval'] = sales_train_all_df['PromoInterval'].astype(int)
```

```
sales_train_all_df["PromoInterval"].unique()
```

array([0, 1, 2, 3])

```
sales_train_all_df_limited = sales_train_all_df.copy()
x_pre = sales_train_all_df_limited.drop(['Sales','Store','Date'] , axis = 1)
```

```
y = sales_train_all_df_limited.Sales
```

```
x = x_pre.copy()
```

Let us get the dummy variables for 'StoreType' and 'Assortment' categorical features before feeding data to machine learning algorithms :

```
x = pd.get_dummies(x)
```

```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.3, random_state=1)
```

Defining the Error functions :

Mean Absolute Error (MAE) :

The MAE is the absolute difference between the data and the model's predictions. As we just use the absolute value of the residual, the MAE does not indicate underperformance or overperformance of the model. A small MAE suggests the model is great at prediction, while a large MAE suggests that the model has trouble in certain areas. A MAE of 0 means that the model is a perfect predictor of the outputs.

```
def MAE(y_actual, y_pred):
  return round(np.mean(abs(y_actual-y_pred)),4)
```

Mean Absolute Percentage Error (MAPE):

MAPE Shows how far the prediction is from the actual value, on average, as a percentage

```
def MAPE(y_actual, y_pred):
    return round(np.mean(np.abs((y_actual - y_pred)/y_actual)),4)
```

Root Mean Squared Error (RMSE) :

RMSE is the square root of the average of squared errors. The effect of each error on RMSE is proportional to the size of the squared error, thus larger errors have a disproportionately large effect on RMSE and aso, RMSE is sensitive to outliers.RMSE is always non-negative, and a value of 0 indicates a perfect fit to the data.

```
def RMSE(y_actual, y_pred):
    return round(np.sqrt(np.mean(np.square((y_actual - y_pred) ))),4)
```

Root Mean Squared Percentage Error (RMSPE):

RMSPE gives an idea of the magnitude of the error in relation to the actual values.

```
def RMSPE(y_actual, y_pred):
    return round((np.sqrt(np.mean(np.square((y_actual - y_pred) / y_actual)))) * 100,4)
```

MACHINE LEARNING MODELS :

1. Average Baseline Model (Model that just calculates the average sales per store) :

```
base_df = sales_train_all_df.copy()

#calculate the average sales value
base_df_avgsales = base_df[['Store','Sales']].groupby("Store").mean().reset_index().ren
ame(columns = {'Sales':'Average_Sales'})

base_all_df = pd.merge(base_df,base_df_avgsales, how = 'left', on = 'Store')

base_df_y_actual = base_all_df['Sales']
base_df_y_pred = base_all_df['Average_Sales']

#get the error metrics of base average model
base_MAE = MAE(base_df_y_actual, base_df_y_pred)
base_MAPE = MAPE(base_df_y_actual, base_df_y_pred)
base_RMSE = RMSE(base_df_y_actual, base_df_y_pred)
base_RMSPE = RMSPE(base_df_y_actual, base_df_y_pred)
```

Base Model MAE : 1456.855
Base Model MAPE : 0.2362
Base Model RMSE : 1957.9119
Base Model RMSPE : 39.3718

2. Linear Regression (Ordinary Least Squares) :

```
lin_model = LinearRegression()
lin_regression = lin_model.fit(x_train, y_train)
# get predictions for Linear Regression
y_pred_LR = lin_regression.predict(x_train)
y_test_pred_LR = lin_regression.predict(x_test)

train_LR_score = lin_regression.score(x_train, y_train)
test_LR_score = lin_regression.score(x_test, y_test)
train_LR_MAE = MAE(y_train, y_pred_LR)
test_LR_MAE = MAE(y_test, y_test_pred_LR)
train_LR_MAPE = MAPE(y_train, y_pred_LR)
test_LR_MAPE = MAPE(y_test, y_test_pred_LR)
train_LR_RMSE = RMSE(y_train, y_pred_LR)
test_LR_RMSE = RMSE(y_test, y_test_pred_LR)
train_LR_RMSPE = RMSPE(y_train, y_pred_LR)
test_LR_RMSPE = RMSPE(y_test, y_test_pred_LR)
```

Linear Regression Model Train Score : 0.826214495661728

Linear Regression Model Test Score : 0.8287774206538482
Train MAE : 938.1422 , Test MAE : 934.7298
Train MAPE : 0.1429 , Test MAPE : 0.1426
Train RMSE : 1292.5497 , Test RMSE : 1287.4622
Training RMSPE : 19.1589 , Testing RMSPE : 19.2301

3. Ridge Regression (Linear least squares with l2 regularization) :

```
#Ridge_l2_regression = RidgeCV(alphas = [0.00001, 0.00003, 0.0001, 0.0003, 0.001, 0
.003, 0.01,0.03, 0.1, 0.3, 1, 3, 10, 30, 100]).fit(x_train, y_train)
#Ridge_l2_regression = Ridge_l2_model.fit(x_train, y_train)
Ridge_l2_model = Ridge(alpha = 1)
Ridge_l2_regression = Ridge_l2_model.fit(x_train, y_train)
# get predictions for Linear Regression with L2 regularization
y_pred_LR2 = Ridge_l2_regression.predict(x_train)
y_test_pred_LR2 = Ridge_l2_regression.predict(x_test)

train_LR2_score = Ridge_l2_regression.score(x_train, y_train)
test_LR2_score = Ridge_l2_regression.score(x_test, y_test)
train_LR2_MAE = MAE(y_train, y_pred_LR2)
test_LR2_MAE = MAE(y_test, y_test_pred_LR2)
train_LR2_MAPE = MAPE(y_train, y_pred_LR2)
test_LR2_MAPE = MAPE(y_test, y_test_pred_LR2)
train_LR2_RMSE = RMSE(y_train, y_pred_LR2)
test_LR2_RMSE = RMSE(y_test, y_test_pred_LR2)
train_LR2_RMSPE = RMSPE(y_train, y_pred_LR2)
test_LR2_RMSPE = RMSPE(y_test, y_test_pred_LR2)
```

Ridge Regression Model Train Score : 0.8259432865513869

Ridge Regression Model Test Score : 0.8285259048331121
Train MAE : 939.6218 , Test MAE : 936.2103
Train MAPE : 0.1432 , Test MAPE : 0.1429
Train RMSE : 1293.5579 , Test RMSE : 1288.4075
Training RMSPE : 19.1795 , Testing RMSPE : 19.2461

4. Lasso Regression (Linear Model trained with L1 prior as regularizer) :

```
#Lasso_l1_regression = LassoCV(alphas = [0.00001, 0.00003, 0.0001, 0.0003, 0.001, 0
.003, 0.01,0.03, 0.1, 0.3, 1, 3, 10, 30, 100]).fit(x_train, y_train)
Lasso_l1_model = Lasso(alpha = 0.01)
Lasso_l1_regression = Lasso_l1_model.fit(x_train, y_train)
# get predictions for Linear Regression with L1 regularization
```

```
y_pred_LR1 = Lasso_l1_regression.predict(x_train)
y_test_pred_LR1 = Lasso_l1_regression.predict(x_test)

train_LR1_score = Lasso_l1_regression.score(x_train, y_train)
test_LR1_score = Lasso_l1_regression.score(x_test, y_test)
train_LR1_MAE = MAE(y_train, y_pred_LR1)
test_LR1_MAE = MAE(y_test, y_test_pred_LR1)
train_LR1_MAPE = MAPE(y_train, y_pred_LR1)
test_LR1_MAPE = MAPE(y_test, y_test_pred_LR1)
train_LR1_RMSE = RMSE(y_train, y_pred_LR1)
test_LR1_RMSE = RMSE(y_test, y_test_pred_LR1)
train_LR1_RMSPE = RMSPE(y_train, y_pred_LR1)
test_LR1_RMSPE = RMSPE(y_test, y_test_pred_LR1)
```

Lasso Regression Model Train Score : 0.8258386024818438

Lasso Regression Model Test Score : 0.8284252702496296
Train MAE : 940.0902 , Test MAE : 936.6873
Train MAPE : 0.1432 , Test MAPE : 0.1429
Train RMSE : 1293.9468 , Test RMSE : 1288.7855
Training RMSPE : 19.1878 , Testing RMSPE : 19.2537

5. Decision Tree :

Before feeding data to Tree based algorithms, let us pre-process the categorical features as before :

```
xt = x_pre.copy()

yt = sales_train_all_df_limited.Sales
```

Let us label the 'StoreType' and 'Assortment' categorical features before feeding data to tree based machine learning algorithms :

```
from sklearn.preprocessing import LabelEncoder
label = LabelEncoder()
xt.StateHoliday = label.fit_transform(xt.StateHoliday)
xt.Assortment = label.fit_transform(xt.Assortment)
xt.StoreType = label.fit_transform(xt.StoreType)

xt_train,xt_test,yt_train,yt_test = train_test_split(xt,yt,test_size=0.3, random_state=1)

dtree_model= DecisionTreeRegressor()
dtree_regression = dtree_model.fit(xt_train, yt_train)

# Create the parameter grid based on the results of random search
params = {
    'max_depth': [2, 3, 5, 10, 20],
```

```
    'min_samples_leaf': [5, 10, 20, 50, 100]
}

# Instantiate the grid search model
grid_search = GridSearchCV(estimator=dtree_model,
                           param_grid=params,
                           cv=4, n_jobs=-1, verbose=1, scoring = "accuracy",return_train_score=
True)

grid_search.fit(xt_train, yt_train)

GridSearchCV(cv=4, estimator=DecisionTreeRegressor(), n_jobs=-1,
             param_grid={'max_depth': [2, 3, 5, 10, 20],
                         'min_samples_leaf': [5, 10, 20, 50, 100]},
             return_train_score=True, scoring='accuracy', verbose=1)

grid_search.best_estimator_

DecisionTreeRegressor(max_depth=2, min_samples_leaf=5)
```

From above grid search, we see that the best decision tree model has max_depth of 2 and min_samples_leaf value of 5 .Hence, using this decision tree model in our case study :

```
dtree_model= DecisionTreeRegressor(max_depth=2, min_samples_leaf=5)

y_pred_DT = dtree_regression.predict(xt_train)
y_test_pred_DT = dtree_regression.predict(xt_test)

train_DT_score = dtree_regression.score(xt_train, yt_train)
test_DT_score = dtree_regression.score(xt_test, yt_test)
train_DT_MAE = MAE(yt_train, y_pred_DT)
test_DT_MAE = MAE(yt_test, y_test_pred_DT)
train_DT_MAPE = MAPE(yt_train, y_pred_DT)
test_DT_MAPE = MAPE(yt_test, y_test_pred_DT)
train_DT_RMSE = RMSE(yt_train, y_pred_DT)
test_DT_RMSE = RMSE(yt_test, y_test_pred_DT)
train_DT_RMSPE = RMSPE(yt_train, y_pred_DT)
test_DT_RMSPE = RMSPE(yt_test, y_test_pred_DT)
```

Decision Tree Model Train Score : 0.9999995196333464

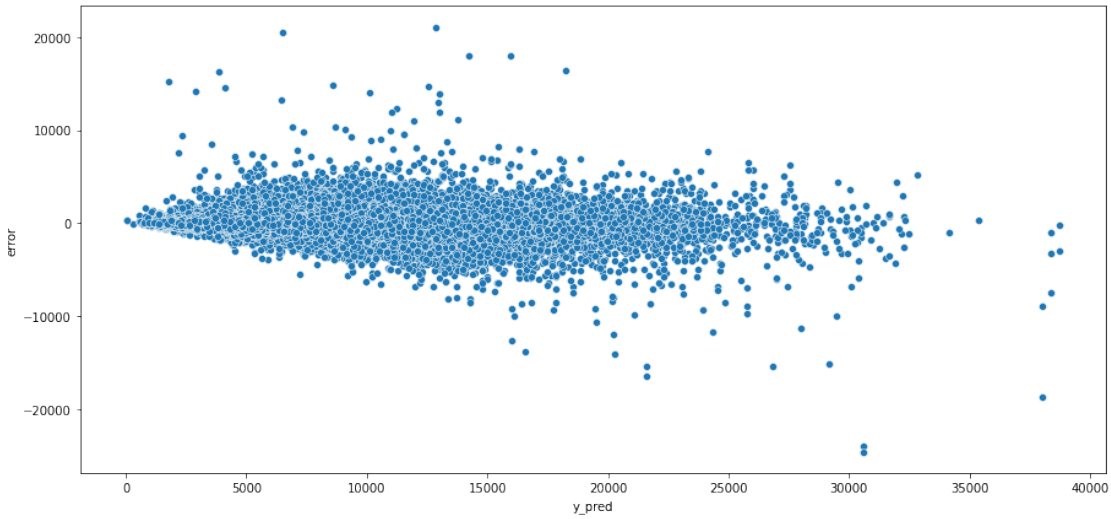
Decision Tree Model Test Score : 0.9477595597051337
Train MAE : 0.0084 , Test MAE : 463.8825
Train MAPE : 0.0 , Test MAPE : 0.0683
Train RMSE : 2.149 , Test RMSE : 711.1437
Training RMSPE : 0.0557 , Testing RMSPE : 9.6098

Error Distribution of Decision Tree Model :

```
error_dist_df = xt_test
error_dist_df['y_actual'] = yt_test
error_dist_df['y_pred'] = y_test_pred_DT

error_dist_df['error'] = error_dist_df['y_actual'] - error_dist_df['y_pred']

plt.figure(figsize=(15, 7))
sns.scatterplot( error_dist_df['y_pred'], error_dist_df['error'] )
```



From the above plot, we can observe that there are few stores with higher error. But majority of the points appear to be in a small horizontal area around zero which indicates that variance in the error is small and centered around zero.

Observations from Phase 3 :

1) It is very important to properly process/map the features 'StateHoliday' and 'PromoInterval' before feeding the data to ML algorithms. Before mapping these features, the model scores were all over the place and only after mapping these features, I was able to get reasonable model performances.

2) Tree algorithms wont even work if we get the dummy variables for 'StateHoliday','Assortment' and 'StoreType' features. Only after label encoding these features, the decision algorith started to work/give scores.

3) After performing grid search hyperparemeter tuning on descision tree algorith, we find that the best decision tree model has max_depth of 2 and min_samples_leaf value of 5.

4) The best Decision tree model has a test R2 score of 94.79%, test MAPE of 6.83% and a test RMSPE of 9.49%.

5) From the Error distribution plot, we observe that there are few stores with higher error. But majority of the other points appear to be in a small horizontal

area around zero which indicates that variance in the error is small and centered around zero.

6) Given below are the comparison of the models used until now :

	Train R2 Score	Test R2 Score	Train MAE Score	\
Linear Regression	**0.826214**	**0.828777**	**938.1422**	
Ridge Regression	**0.825943**	**0.828526**	**939.6218**	
Lasso Regression	**0.825839**	**0.828425**	**940.0902**	
Decision Tree	**1.000000**	**0.947760**	**0.0084**	

	Test MAE Score	Train MAPE Score	Test MAPE Score	\
Linear Regression	**934.7298**	**0.1429**	**0.1426**	
Ridge Regression	**936.2103**	**0.1432**	**0.1429**	
Lasso Regression	**936.6873**	**0.1432**	**0.1429**	
Decision Tree	**463.8825**	**0.0000**	**0.0683**	

	Train RMSE Score	Test RMSE Score	Train RMSPE Score	\
Linear Regression	**1292.5497**	**1287.4622**	**19.1589**	
Ridge Regression	**1293.5579**	**1288.4075**	**19.1795**	
Lasso Regression	**1293.9468**	**1288.7855**	**19.1878**	
Decision Tree	**2.1490**	**711.1437**	**0.0557**	

	Test RMSPE Score
Linear Regression	**19.2301**
Ridge Regression	**19.2461**
Lasso Regression	**19.2537**
Decision Tree	**9.6098**

In the next phase of the project, I will be trying out more complex ML models like XGBoost regressor, Random Forest regressor and if possible, use the state-of-the-art Facebook Prophet model for sales prediction.