**Problem Approach**

To predict the course sales for each course in the test set for the next 60 days, I have built a Light Gradient Boosting Model with K-fold cross validation and Bayesian Optimization for Hyperparameter tuning.

**Steps: -**

* Import Necessary Libraries
* Exploratory Data Analysis and Inferences
* Feature Engineering
* Model Building
* Final Model Selection
* Conclusion

**1.Import all libraries and dataset**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.preprocessing import StandardScaler

from sklearn import preprocessing

from sklearn.linear\_model import LinearRegression, Ridge, Lasso

from sklearn.ensemble import RandomForestRegressor

from xgboost import XGBRegressor

import sklearn

from datetime import date, timedelta

from sklearn.model\_selection import train\_test\_split

import lightgbm as lgb

from bayes\_opt import BayesianOptimization

from sklearn.metrics import mean\_squared\_error

from sklearn.model\_selection import KFold, StratifiedKFold

from sklearn import model\_selection, preprocessing, metrics

import numpy as np # linear algebra

import pandas as pd # data processing, CSV file I/O (e.g. pd.read\_csv)

# Change your location

train = pd.read\_csv('/content/drive/MyDrive/Sales-Forecasting/train.csv')

test = pd.read\_csv('/content/drive/MyDrive/Sales-Forecasting/test\_6hcImfB.csv')

sample = pd.read\_csv('/content/drive/MyDrive/Sales-Forecasting/sample\_submission\_ROrnLUk.csv')

**2.Exploratory Data Analysis**

# Columns of the data set

print(list(train.columns))

print(list(test.columns))

train.dtypes

test.dtypes

train.describe()

train.describe(include='object')

test.describe()

test.describe(include='object')

# checking for null values

train.isnull().sum()

train.ID.nunique()

Univariate Analysis

train[['Sales']].boxplot()

sns.distplot(train['Sales'])

print("Skewness= ", train['Sales'].skew())

print("Kurtosis= ", train['Sales'].kurt())

# plotting on sample of dataset

sampletrain= train.sample(1000)

sns.countplot(train['Course\_Domain'])

sns.countplot(train['Course\_Type'])

sns.countplot(train['Long\_Promotion'])

sns.countplot(train['Short\_Promotion'])

Multivariate Analysis

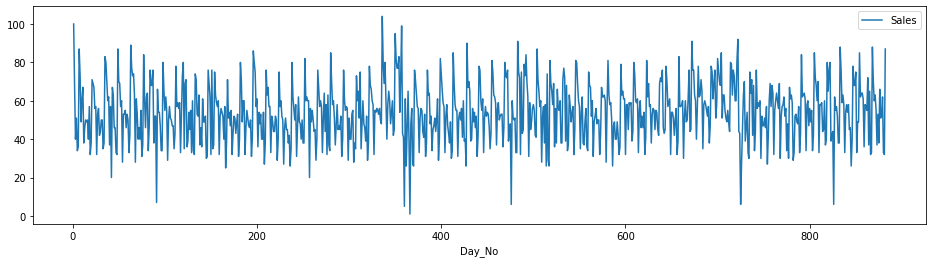
sns.regplot(x='Day\_No',y='Sales',data= sampletrain)

train[['Course\_ID','Sales']].boxplot()

train[['Course\_Domain','Sales']].boxplot()

train[['Short\_Promotion','Sales']].boxplot()

* Let’s look at how sales varying across a particular Course by day.



**Inferences from EDA:**

We find that there are missing values in the Competition\_Metric for both train and test sets. We will be later imputing them.

ID is unique for each data point, So removing it.

From the description of sales, we find that minimum sales is 0 which is not possible.Also there are outliers in the Sales Data.From the distribution plot and the skewness and kurtosis values, we come to know that the Sales data is positively skewed.So we will apply log transformation before feeding it into the model.

There are very few rows which have Business as the Course Domain.

There are also very few rows which have Degree as Course Type

There are almost equal distribution among the Long Promotion category whereas the same thing cannot be said among Short Promotion.

**3.Feature Engineering**

* Converting Day\_No to corresponding date and then date to day, month,year and Week of Year

def day\_to\_date(dataset):

    start = date(2018,12,31)

    dataset['Date'] = dataset['Day\_No'].apply(lambda x: start + timedelta(x))

def day\_month\_year(dataset):

    dataset['Day'] = dataset['Date'].apply(lambda x: x.day)

    dataset['Month'] = dataset['Date'].apply(lambda x: x.month)

    dataset['Year'] = dataset['Date'].apply(lambda x: x.year)

    dataset["Date"] = pd.to\_datetime(dataset["Date"])

    dataset['weekofyear'] = dataset['Date'].dt.weekofyear

day\_to\_date(train)

day\_month\_year(train)

day\_to\_date(test)

day\_month\_year(test)

* Since test data doesn’t contain User Traffic, we are removing it.

train.drop('User\_Traffic',axis =1, inplace = True)

* Imputing Missing values in Competition Metric by replacing it with the medianin each of the corresponding Course group. (choosing median since Competition\_metric data has outliers)

for course\_id in train1.Course\_ID.unique():

        train1[train1['Course\_ID']==course\_id]['Competition\_Metric'].fillna(train1[train1['Course\_ID'==course\_id]]['Competition\_Metric'].median())

for course\_id in train1.Course\_ID.unique():

        test1[test1['Course\_ID']==course\_id]['Competition\_Metric'].fillna(test1[test1['Course\_ID'==course\_id]]['Competition\_Metric'].median())

* One-hot Encoding the Course Domain, Course Type values(Converting categorical to numerical types)

df1=pd.get\_dummies(df,columns=['Course\_Domain','Course\_Type'],drop\_first=True)

* Creating Day of week from Day\_No.

df1['Day\_of\_week']=df1['Day\_No'].apply(lambda x:x%7)

* Correlation heatmap

plt.subplots(figsize=(20,20))

sns.heatmap(df1.corr(),annot=True, vmin=-0.1, vmax=0.1,center=0)

From the above correlation heatmap, we can observe that the newly created features Day, Month, Year have high correlation with Day\_No and Date. So, I am dropping the Day\_No and Date columns.

df1.drop(['Day\_No','Date'],axis = 1,inplace = True)

#splitting train and test from df

train1= df1[df1['Sales'].isnull()!= True]

test1= df1[df1['Sales'].isnull()== True].drop(['Sales'], axis=1)

print(train1.shape)

print(test1.shape)

* This is a function that reduces the memory used by the dataframe in storage.It reduces the memory by assigning the right kind of datatype to the columns in our dataset by analysing the largest value in each column.

def reduce\_mem\_usage(df, verbose=True):

    numerics = ['int8','int16', 'int32', 'int64', 'float16', 'float32', 'float64']

    start\_mem = df.memory\_usage().sum() / 1024\*\*2

    for col in df.columns:

        col\_type = df[col].dtypes

        if col\_type in numerics:

            c\_min = df[col].min()

            c\_max = df[col].max()

            if str(col\_type)[:3] == 'int':

                if c\_min > np.iinfo(np.int8).min and c\_max < np.iinfo(np.int8).max:

                    df[col] = df[col].astype(np.int8)

                elif c\_min > np.iinfo(np.int16).min and c\_max < np.iinfo(np.int16).max:

                    df[col] = df[col].astype(np.int16)

                elif c\_min > np.iinfo(np.int32).min and c\_max < np.iinfo(np.int32).max:

                    df[col] = df[col].astype(np.int32)

                elif c\_min > np.iinfo(np.int64).min and c\_max < np.iinfo(np.int64).max:

                    df[col] = df[col].astype(np.int64)

            else:

                if c\_min > np.finfo(np.float16).min and c\_max < np.finfo(np.float16).max:

                    df[col] = df[col].astype(np.float16)

                elif c\_min > np.finfo(np.float32).min and c\_max < np.finfo(np.float32).max:

                    df[col] = df[col].astype(np.float32)

                else:

                    df[col] = df[col].astype(np.float64)

    end\_mem = df.memory\_usage().sum() / 1024\*\*2

    if verbose: print('Mem. usage decreased to {:5.2f} Mb ({:.1f}% reduction)'.format(end\_mem, 100 \* (start\_mem - end\_mem) / start\_mem))

    return df

%%time

train1= reduce\_mem\_usage(train1)

test1= reduce\_mem\_usage(test1)

**4.Model Building**

* I have experimented with multiple models such as XGBoostRegressor, LGBRegressor and H20AutoML.
* Though LGBMRegressor and H20AutoML performed good on the training set, the RMSLE\*1000 score upon submission were more compared to the model compared to base LGB model.So I proceeded to use that along with Bayesian Optimization for Hyperparameter Tuning.
* The implementations of the XGBoost and H20AutoML models can be found under the section Other Models.
* The reason for choosing Ensemble models is because this is a Regression problem and

What Is Bayesian Optimization and Why Do We Care?

**Bayesian Optimization** is a probabilistic model based approach for finding the minimum of any function that returns a real-value metric. [(source)](https://towardsdatascience.com/an-introductory-example-of-bayesian-optimization-in-python-with-hyperopt-aae40fff4ff0)  
It is very effective with real-world applications in high-dimensional parameter-tuning for complex machine learning algorithms. Bayesian optimization utilizes the Bayesian technique of setting a prior over the objective function and combining it with evidence to get a posterior function. I attached one graph that demonstrates Bayes’ theorem below.

Code for Bayesian Optimization approach to get the best model parameters.

We can pass the range of parameters and the algorithm will find the best set of parameters which will yield the least loss.

%%time

X\_train = train1.drop('Sales',axis=1)

Y\_train = np.log1p(train1.Sales)

def bayes\_parameter\_opt\_lgb(X, y, init\_round=15, opt\_round=25, n\_folds=3, random\_seed=6,n\_estimators=10000, output\_process=False):

    def rmsle(preds, lgb\_train):

        eval\_name = "rmsle"

        eval\_result = np.sqrt(mean\_squared\_log\_error(preds, lgb\_train.get\_label()))

        return (eval\_name, eval\_result\*1000, False)

    # prepare data

    train\_data = lgb.Dataset(data=X, label=y, free\_raw\_data=False)

    # parameters

    def lgb\_eval(learning\_rate,num\_leaves, feature\_fraction, bagging\_fraction, max\_depth, max\_bin, min\_data\_in\_leaf,min\_sum\_hessian\_in\_leaf,reg\_alpha, reg\_lambda,subsample):

        params = {'objective':'gamma', 'metric':'rmse','task':'train'}

        params['learning\_rate'] = max(min(learning\_rate, 1), 0)

        params["num\_leaves"] = int(round(num\_leaves))

        params['feature\_fraction'] = max(min(feature\_fraction, 1), 0)

        params['bagging\_fraction'] = max(min(bagging\_fraction, 1), 0)

        params['max\_depth'] = int(round(max\_depth))

        params['max\_bin'] = int(round(max\_depth))

        params['reg\_lambda'] = float(reg\_lambda)

        params['reg\_alpha'] = float(reg\_alpha)

        params['min\_data\_in\_leaf'] = int(round(min\_data\_in\_leaf))

        params['min\_sum\_hessian\_in\_leaf'] = min\_sum\_hessian\_in\_leaf

        params['subsample'] = max(min(subsample, 1), 0)

        cv\_result = lgb.cv(params, train\_data, nfold=n\_folds, seed=random\_seed,feval=rmsle,stratified=False, verbose\_eval =200, metrics=['rmse'])

        return (-1.0 \* np.array(cv\_result['rmse-mean'])).max()

    lgbBO = BayesianOptimization(lgb\_eval, {'learning\_rate': (0.01, 1.0),

                                            'num\_leaves': (24, 80),

                                            'feature\_fraction': (0.1, 0.9),

                                            'bagging\_fraction': (0.8, 1),

                                            'max\_depth': (5, 30),

                                            'max\_bin':(20,90),

                                            'reg\_lambda': (0.1, 1),

                                            'reg\_alpha': (0.1, 1),

                                            'min\_data\_in\_leaf': (20, 80),

                                            'min\_sum\_hessian\_in\_leaf':(0,100),

                                           'subsample': (0.01, 1.0)}, random\_state=200)

    #n\_iter: How many steps of bayesian optimization you want to perform. The more steps the more likely to find a good maximum you are.

    #init\_points: How many steps of random exploration you want to perform. Random exploration can help by diversifying the exploration space.

    lgbBO.maximize(init\_points=init\_round, n\_iter=opt\_round,acq='ei')

    model\_auc=[]

    for model in range(len( lgbBO.res)):

        model\_auc.append(lgbBO.res[model]['target'])

    # if output\_process==True: lgbBO.points\_to\_csv("bayes\_opt\_result.csv")

    ## return best parameters

    # return lgbBO.res['max']['max\_params']

    # return optimizer.max

    return lgbBO.res[pd.Series(model\_auc).idxmax()]['target'],lgbBO.res[pd.Series(model\_auc).idxmax()]['params']

opt\_params = bayes\_parameter\_opt\_lgb(X\_train, Y\_train, init\_round=10, opt\_round=10, n\_folds=5, random\_seed=6,n\_estimators=10000)

Optimal Parameters found:

{'bagging\_fraction': 0.945328354877452,

'feature\_fraction': 0.6794460921764023,

'learning\_rate': 0.5021558629186151,

'max\_bin': 67,

'max\_depth': 26,

'metric': 'rmse',

'min\_data\_in\_leaf': 34,

'min\_sum\_hessian\_in\_leaf': 24.922537940382195,

'num\_leaves': 66,

'objective': 'gamma',

'reg\_alpha': 0.2794403838240256,

'reg\_lambda': 0.19901597467105386,

'subsample': 0.34814142455066416}

Training the LGBM Model and getting the predictions:

# run\_lgb = True

X\_train=train1.drop('Sales',axis=1)

Y\_train=np.log1p(train1.Sales)

X\_test=test1.copy()

# print('LGB : ', run\_lgb)

# modeling

#--------------------------------------------------------------------------

# if run\_lgb:

import lightgbm as lgb

def rmsle(y, yhat):

    return (np.sqrt(mean\_squared\_log\_error(y, yhat)))\*1000

def rmsle\_xg(yhat, y):

    # print(y.get\_label())

    y = np.expm1(y.get\_label())

    yhat = np.expm1(yhat)

    return ("rmsle", rmsle(y,yhat),False)

def kfold\_lgb\_xgb():

    folds = KFold(n\_splits=5, shuffle=True, random\_state=7)

    oof\_lgb = np.zeros(len(X\_train))

    predictions\_lgb = np.zeros(len(X\_test))

    features\_lgb = list(X\_train.columns)

    feature\_importance\_df\_lgb = pd.DataFrame()

    for fold\_, (trn\_idx, val\_idx) in enumerate(folds.split(X\_train)):

        trn\_data = lgb.Dataset(X\_train.iloc[trn\_idx], label=Y\_train.iloc[trn\_idx])

        val\_data = lgb.Dataset(X\_train.iloc[val\_idx], label=Y\_train.iloc[val\_idx])

        print("LGB " + str(fold\_) + "-" \* 50)

        num\_round = 20000

        clf = lgb.train(opt\_params, trn\_data, num\_round,feval=rmsle\_xg,valid\_sets = [trn\_data, val\_data], verbose\_eval=1000, early\_stopping\_rounds = 100)

        oof\_lgb[val\_idx] = clf.predict(X\_train.iloc[val\_idx], num\_iteration=clf.best\_iteration)

        fold\_importance\_df\_lgb = pd.DataFrame()

        fold\_importance\_df\_lgb["feature"] = features\_lgb

        fold\_importance\_df\_lgb["importance"] = clf.feature\_importance()

        fold\_importance\_df\_lgb["fold"] = fold\_ + 1

        feature\_importance\_df\_lgb = pd.concat([feature\_importance\_df\_lgb, fold\_importance\_df\_lgb], axis=0)

        predictions\_lgb += clf.predict(X\_test, num\_iteration=clf.best\_iteration) / folds.n\_splits

    #lgb.plot\_importance(clf, max\_num\_features=30)

    cols = feature\_importance\_df\_lgb[["feature", "importance"]].groupby("feature").mean().sort\_values(by="importance", ascending=False)[:50].index

    best\_features\_lgb = feature\_importance\_df\_lgb.loc[feature\_importance\_df\_lgb.feature.isin(cols)]

    plt.figure(figsize=(14,10))

    sns.barplot(x="importance", y="feature", data=best\_features\_lgb.sort\_values(by="importance", ascending=False))

    plt.title('LightGBM Features (avg over folds)')

    plt.tight\_layout()

    plt.savefig('lgbm\_importances.png')

    x = []

    for i in oof\_lgb:

        if i < 0:

            x.append(0.0)

        else:

            x.append(i)

    cv\_lgb = mean\_squared\_error(x, Y\_train)\*\*0.5

    cv\_lgb = str(cv\_lgb)

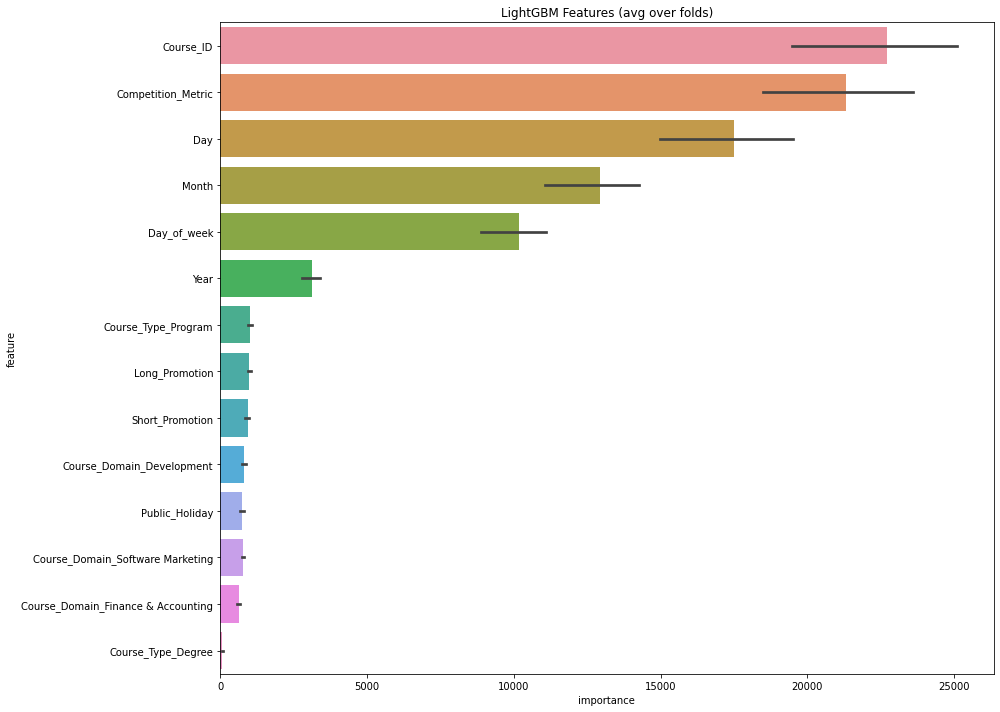
    cv\_lgb = cv\_lgb[:10]

    pd.DataFrame({'preds': x}).to\_csv('lgb\_oof\_' + cv\_lgb + '.csv', index = False)

    print("CV\_LGB : ", cv\_lgb)

    return cv\_lgb, predictions\_lgb

cv\_lgb, lgb\_ans = kfold\_lgb\_xgb()

Feature Importances:

* Saving the predictions to a csv file:

submission = pd.DataFrame()

submission['ID'] = sample['ID']

submission['Sales'] = np.expm1(x)

submission.to\_csv('lgb\_with\_bayesian\_opt.csv', index=False)

submission.head()

Other Models:

* XGB Regressor

#xgb

kf = StratifiedKFold(n\_splits=5,shuffle=True,random\_state=123)

X= train[features]

y= train.Sales

cv\_score =[]

i=1

for train\_index,test\_index in kf.split(X, y):

    print('Fold no. = ', i)

    x\_train, x\_test = X.loc[train\_index], X.loc[test\_index]

    y\_train, y\_test = y.loc[train\_index], y.loc[test\_index]

    #model

    xgb = XGBRegressor(n\_estimators= 500)

    xgb.fit(x\_train, y\_train)

    y\_pred= xgb.predict(x\_test)

    score = rmsle1000(y\_test, y\_pred)

    print('RMSLE score:',score)

    cv\_score.append(score)

    i+=1

#xgb mean rmsle

np.mean(cv\_score)

274.5845448548124

Addition to Feature Engineering:

I have binned the Competition\_metric column into five categories like below: But when I ran XGBoost and LGBMRegressor, the categories had no feature importance.

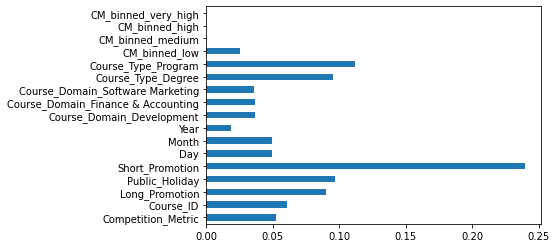
So I removed this binning of the Competition Metric.

df['CM\_binned'] = pd.cut(df['Competition\_Metric'],bins=5, labels=['very\_low','low','medium','high','very\_high'])

df.CM\_binned

# xgb feature importance

feat\_importances = pd.Series(xgb.feature\_importances\_, index=features)

feat\_importances.plot(kind='barh')

* LGBM Regressor

#lgbm

cv\_score =[]

i=1

for train\_index,test\_index in kf.split(X, y):

    print('Fold no. = ', i)

    x\_train, x\_test = X.loc[train\_index], X.loc[test\_index]

    y\_train, y\_test = y.loc[train\_index], y.loc[test\_index]

    #model

    lgbm = LGBMRegressor(n\_estimators= 500 )

    lgbm.fit(x\_train, y\_train)

    y\_pred= lgbm.predict(x\_test)

    score = rmsle1000(y\_test, y\_pred)

    print('RMSLE score:',score)

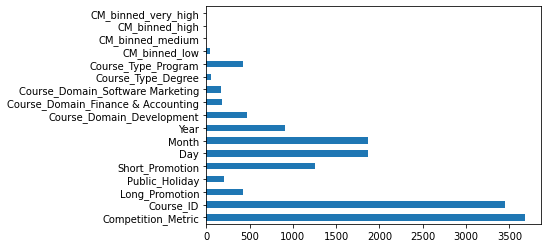
    cv\_score.append(score)

    i+=1

#lgbm mean rmsle

np.mean(cv\_score)

189.300585399091



* H20 AutoML

import h2o

from h2o.automl import H2OAutoML

h2o.init()

train = h2o.H2OFrame(train1)

test = h2o.H2OFrame(test1)

y = "Sales"

x = list(X\_train.columns)

aml = H2OAutoML(max\_models = 30, max\_runtime\_secs=300, seed = 1)

aml.train(x = x, y = y, training\_frame = train)

lb = aml.leaderboard

lb.head()

lb.head(rows=lb.nrows)

preds = aml.predict(test)

water\_preds=h2o.as\_list(preds)

**Conclusion:**

I have done Exploratory Data Analysis(Univariate and Multivariate) and created new features based on the data and successfully built a model that was able to obtain a decent RMSLE\*1000 value on the unknown data. I have explored multiple models for the problem and have finally chosen the LGB model with Bayesian Optimization tuned parameters as my final model.

**Further improvements:**

* The model performance could be enhanced by generating lag features i.e. using past values to generate the future data since the given problem is in a way a time series problem.The code for this looks something like:

def create\_lag\_features(df, sales\_cols, columns\_list, lag\_days):

temp = df.copy()

for i in range(lag\_days, 0, -1):

temp = pd.concat([temp[columns\_list],df[sales\_cols].shift(i)], axis=1)

columns\_list = columns\_list +[sales\_col+'\_t\_'+str(i) for sales\_col in sales\_cols]

temp.columns = columns\_list

return temp

* Also Deep Learning models such as LSTM and Bidirectional LSTM can be used as models for the problem.(I have explored them but they have not given satisfactory results so I am mentioning them so that in the future,they could be enhanced.)
* Further more, the data can be explored to generate some new features suitable to the business problem.