**WOMEN DATA SCIENCE HACKATHON BY BAIN & COMPANY**

**Code and Approach**

**Index :**

* Import all essential libraries and dataset
* Feature Engineering
* Exploratory data analysis
* Basic inferences from EDA
* Model experimentation
* Final model selection
* Insights

1. **Import all essential libraries and dataset**

**Code :**

%matplotlib inline

from sklearn.preprocessing import MinMaxScaler

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

from sklearn import metrics #Additional scklearn functions

from sklearn.model\_selection import GridSearchCV

from sklearn.model\_selection import cross\_validate

from sklearn.model\_selection import cross\_val\_score

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

from sklearn.metrics import confusion\_matrix

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import accuracy\_score

from sklearn import metrics

import matplotlib.pyplot as plt

import lightgbm as lgb

from bayes\_opt import BayesianOptimization

from sklearn.metrics import r2\_score

from sklearn.ensemble import RandomForestRegressor

train = pd.read\_csv("train\_bcg.csv")

test = pd.read\_csv("test\_bcg.csv")

1. **Feature Engineering**
   1. **One hot encoding for categorical features to extract importance all values for a variable**

**Code :**

train.loc[train['Course\_Domain'] == 'Development', 'Development'] = 1

train.loc[train['Course\_Domain'] == 'Software Marketing', 'Software\_Marketing'] = 1

train.loc[train['Course\_Domain'] == 'Finance & Accounting', 'Finance\_Accounting'] = 1

train.loc[train['Course\_Domain'] == 'Business', 'Business'] = 1

train.loc[train['Course\_Type'] == 'Course', 'Course'] = 1

train.loc[train['Course\_Type'] == 'Program', 'Program'] = 1

train.loc[train['Course\_Type'] == 'Degree', 'Degree'] = 1

test.loc[test['Course\_Domain'] == 'Development', 'Development'] = 1

test.loc[test['Course\_Domain'] == 'Software Marketing', 'Software\_Marketing'] = 1

test.loc[test['Course\_Domain'] == 'Finance & Accounting', 'Finance\_Accounting'] = 1

test.loc[test['Course\_Domain'] == 'Business', 'Business'] = 1

test.loc[test['Course\_Type'] == 'Course', 'Course'] = 1

test.loc[test['Course\_Type'] == 'Program', 'Program'] = 1

test.loc[test['Course\_Type'] == 'Degree', 'Degree'] = 1

* 1. **Missing value treatment**

**Code :**

train = train.fillna(0)

test = test.fillna(0)

* 1. **Feature User\_tranffic is absent in test data to it does not make any use to use it in training process**

**Code :**

train=train.drop(['User\_Traffic'],axis=1)

train=train.drop(['Course\_Domain','Course\_Type'],axis=1)

test=test.drop(['Course\_Domain','Course\_Type'],axis=1)

feature = [x for x in train.columns if x not in 'Sales']

y = train[['ID','Sales']]

X = train[feature]

X = X.set\_index('ID')

test = test.set\_index('ID')

y = y.set\_index('ID')

* 1. **Outlier Removal**

**Code :**

upper\_lim = train['Sales'].quantile(.98)

lower\_lim = train['Sales'].quantile(.01)

train.loc[(train['Sales'] > upper\_lim),'Sales'] = upper\_lim

train.loc[(train['Sales'] < lower\_lim),'Sales'] = lower\_lim

feature = [x for x in train.columns if x not in 'Sales']

y = train[['Sales']]

X = train[feature]

1. **Exploratory data analysis:**
   1. **Correlation metrics**

**Code :**

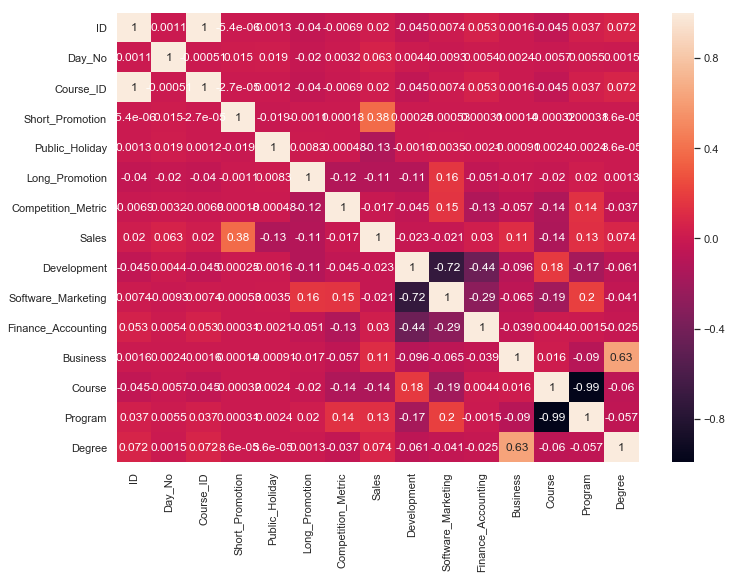
import seaborn as sns

sns.set(rc={'figure.figsize':(11.7,8.27)})

train.corr()

sns.heatmap(train.corr(), annot=True)

plt.show()



**Code :**

# fig, ax = plt.subplots()

# train['Day\_No'].hist(color='#A9C5D3', edgecolor='black',

# grid=False)

# ax.set\_title('Developer Age Histogram', fontsize=12)

# ax.set\_xlabel('Day\_No', fontsize=12)

# ax.set\_ylabel('Frequency', fontsize=12)

# train.plot.scatter(x='Day\_No',

# y='Sales',

# c='DarkBlue')

# train.plot.scatter(x='Course\_ID',

# y='Sales',

# c='DarkBlue')

# train['Day\_No'].max() #882

# train['Day\_No'].min() #1

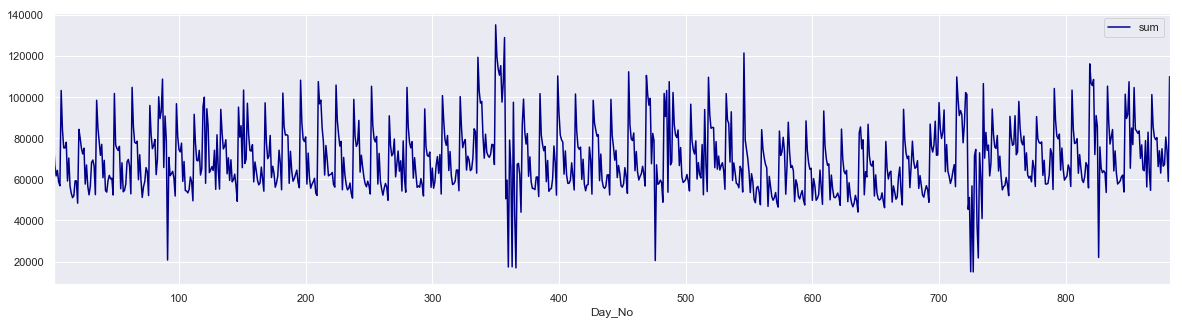
grouped\_sales = train.groupby('Day\_No',as\_index=False).agg({'Sales': ['sum']})

grouped\_sales

grouped\_sales.plot.line(x='Day\_No',

y='Sales',

c='DarkBlue',figsize=(20,5))



* 1. **Multicollinearity check:**

**Code :**

import numpy as np

import pandas as pd

import time

from statsmodels.stats.outliers\_influence import variance\_inflation\_factor

from joblib import Parallel, delayed

# Defining the function that you will run later

def calculate\_vif\_(X, thresh=5.0):

variables = [X.columns[i] for i in range(X.shape[1])]

dropped=True

while dropped:

dropped=False

print(len(variables))

vif = Parallel(n\_jobs=-1,verbose=5)(delayed(variance\_inflation\_factor)(X[variables].values, ix) for ix in range(len(variables)))

maxloc = vif.index(max(vif))

if max(vif) > thresh:

print(time.ctime() + ' dropping \'' + X[variables].columns[maxloc] + '\' at index: ' + str(maxloc))

variables.pop(maxloc)

dropped=True

print('Remaining variables:')

print([variables])

return X[[i for i in variables]]

feature\_list = [x for x in train.columns ]

df = train[feature\_list] # Selecting your data

df2 = calculate\_vif\_(train,5) # Actually running the function

**Dropped multicollinear features :**

X=X.drop(['Development','Program'],axis=1)

test=test.drop(['Development','Program'],axis=1)

1. **Base inferences from EDA**
   1. **The data looks stationary and does not be stationarized**
   2. **There is no seasonality in the data**
   3. **User\_traffic is absent in test data hence omitted from analysis**
   4. **Sales is positively correlated with short promotions and negatively correlated with long promotion in extremes**
   5. **Development Course Domain and Program Course type cause multicollinearity hence omitted**
2. **Choice of model:**
   1. **Seasonal ARIMA**

**Code :**

from pmdarima import auto\_arima

stepwise\_model = auto\_arima(train['Sales'],exogenous= X[['Course\_ID', 'Short\_Promotion', 'Public\_Holiday','Long\_Promotion', 'Competition\_Metric', 'Development','Software\_Marketing', 'Finance\_Accounting', 'Business', 'Course','Program', 'Degree']]

,start\_p=0, start\_q=0,

max\_p=5, max\_q=5, m=1,

start\_P=0, seasonal=False,

d=0, D=1, trace=True,

error\_action='ignore',

suppress\_warnings=True,

stepwise=True)

**Performance :** The algorithm took more than 2hrs+ to tune so I skipped this implementation.

* 1. **Random Forest :** 
     1. Tree based algorithms can handle multicollinearity I
     2. Its easier to tune Random Forest
     3. It’s useful to get feature importance

**Code :**

from sklearn.model\_selection import GridSearchCV

# Create the parameter grid based on the results of random search

param\_grid = {

'bootstrap': [True],

'max\_depth': [80, 90, 100, 110],

'max\_features': [2, 3,5],

'min\_samples\_leaf': [3, 4, 5],

'min\_samples\_split': [8, 10, 12],

'n\_estimators': [100, 200, 300,1000] #will chck later with 1000 iterations

}

**e. Feature importance**

**Code :**

rf.feature\_importances\_

features = [x for x in train.columns if x not in ['ID','Sales']]

importances = rf.feature\_importances\_

indices = np.argsort(importances)

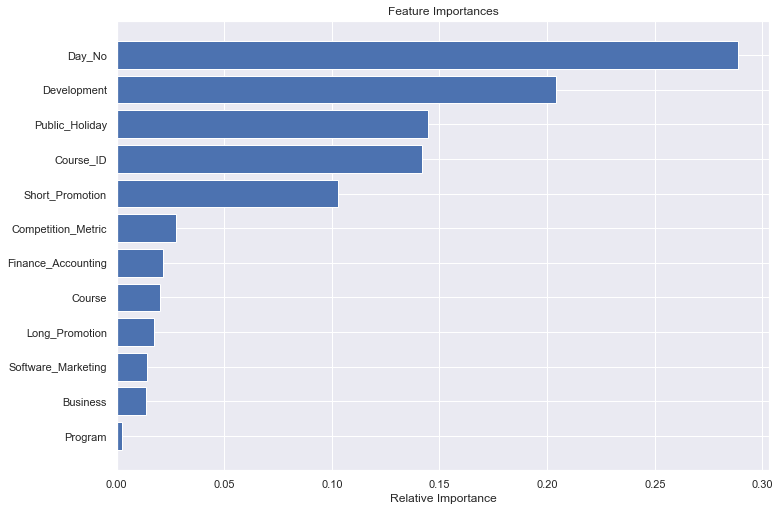
plt.title('Feature Importances')

plt.barh(range(len(indices)), importances[indices], color='b', align='center')

plt.yticks(range(len(indices)), [features[i] for i in indices])

plt.xlabel('Relative Importance')

plt.show()



1. Day\_No should play a role the model hence its omitted

**Code :**

rf = RandomForestRegressor()

# Instantiate the grid search model

grid\_search = GridSearchCV(estimator = rf, param\_grid = param\_grid,

cv = 3, n\_jobs = -1, verbose = 2)

grid\_search.fit(X, y)

grid\_search.best\_params\_

####based on best parameters build a base model

rf.fit(X,y)

* 1. **Light GBM**
     1. To leverage the accuracy of GBM based models and get the results in least possible time Light GBM was leveraged.
     2. It took few hours to tune the model and once the parameters were selected it took few minutes to train the model.
     3. Can handle categorical variables well without the need for one-hot encoding

**Code:**

cate\_vars = ['Course\_ID','Course\_ID', 'Short\_Promotion', 'Public\_Holiday',

'Long\_Promotion', 'Development', 'Software\_Marketing', 'Finance\_Accounting', 'Business', 'Course',

'Program', 'Degree']

X['Competition\_Metric'] = X['Competition\_Metric']\*10

test['Competition\_Metric'] = test['Competition\_Metric']\*10

X.drop(['Day\_No'],axis=1)

test.drop(['Day\_No'],axis=1)

lgb\_train = lgb.Dataset(X, y ,categorical\_feature = cate\_vars)

lgb\_eval = lgb.Dataset(X, y,reference=lgb\_train)

params = {

'task': 'train',

'boosting\_type': 'gbdt',

'objective': 'regression',

'metric': {'rmse'},

'learning\_rate': [0.1,0.2,0.3,0.5],

'num\_leaves': [16,32,64,1024],

'max\_depth' : -1,

'reg\_lambda':0.2,

'min\_child\_samples': [10,20,30,40]

'num\_iteration': 1000, #optimize later

'verbose': 20,

'n\_estimators': [100,200,300,500]

'njobs': -1

}

gbm = lgb.train(params,

lgb\_train,

num\_boost\_round=1000,

valid\_sets=lgb\_eval,

early\_stopping\_rounds=10)

# Instantiate the grid search model

grid\_search = GridSearchCV(estimator = gbm, param\_grid = params,

cv = 3, n\_jobs = -1, verbose = 2)

grid\_search.fit(X, y)

grid\_search.best\_params\_

* 1. **Model fitting**

y\_pred = gbm.predict(test)

result = pd.DataFrame(y\_pred,index = test.index,columns=['Sales'])

result.to\_csv('submission\_rf.csv',index=True)

1. **Final Selection of model** : Light GBM is chosen as the final model with lesser running time and better accuracy on the test data