Advanced Data Analytics

(CSE4029)

Black Friday Sales Analysis
Project Report

(Phase - II)

Varad Vijay Temghare
19BCI7045

Submitted to :

Dr. Nitesh Funde

Introduction:-

In Data Science, we usually deal with datasets that contain multiple labels in one or more than one column. These labels can be in the form of words or numbers. To make the data understandable or in human-readable form, the training data is often labeled in words.

Dataset Link :- [Here]

Dimensions of the above dataset:- We have 5,50,069 rows and 12 columns .

Code Link:- [Here]

Label Encoding refers to converting the labels into a numeric form so as to convert them into the machine-readable form. Machine learning algorithms can then decide in a better way how those labels must be operated. It is an important preprocessing step for the structured dataset in supervised learning.

Encoding the categorical variables

```
[45] from sklearn.preprocessing import LabelEncoder
     lr = LabelEncoder()
```

```
[46] df['Gender'] = lr.fit_transform(df['Gender'])
```

```
[47] df['Age'] = lr.fit_transform(df['Age'])
```

```
[48] df['City_Category'] = lr.fit_transform(df['City_Category'])
```

```
[49] df.head()
```

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Marital_Status	Product_Category_1	Product_Category_2	Product_Category_3	Purchase	Stay_In_Current_City_Years_0	Stay_In_Current_City_Years_
0	1000001	P00069042	0	0	10	0	0	3	NaN	NaN	8370	0	
1	1000001	P00248942	0	0	10	0	0	1	6.0	14.0	15200	0	
2	1000001	P00087842	0	0	10	0	0	12	NaN	NaN	1422	0	
3	1000001	P00085442	0	0	10	0	0	12	14.0	NaN	1057	0	
4	1000002	P00285442	1	6	16	2	0	8	NaN	NaN	7969	0	

```
[50] df['Product_Category_2'] =df['Product_Category_2'].fillna(0).astype('int64')
     df['Product_Category_3'] =df['Product_Category_3'].fillna(0).astype('int64')
```

```
[50] df['Product_Category_2'] =df['Product_Category_2'].fillna(0).astype('int64')
     df['Product_Category_3'] =df['Product_Category_3'].fillna(0).astype('int64')
```

```
[51] df.isnull().sum()
```

```
User_ID                        0
Product_ID                     0
Gender                         0
Age                            0
Occupation                     0
City_Category                  0
Marital_Status                 0
Product_Category_1             0
Product_Category_2             0
Product_Category_3             0
Purchase                       0
Stay_In_Current_City_Years_0   0
Stay_In_Current_City_Years_1   0
Stay_In_Current_City_Years_2   0
Stay_In_Current_City_Years_3   0
Stay_In_Current_City_Years_4+  0
dtype: int64
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 550068 entries, 0 to 550067
Data columns (total 16 columns):
 #   Column                         Non-Null Count   Dtype
---  ------                         --------------   -----
 0   User_ID                        550068 non-null  int64
 1   Product_ID                     550068 non-null  object
 2   Gender                         550068 non-null  int64
 3   Age                            550068 non-null  int64
 4   Occupation                     550068 non-null  int64
 5   City_Category                  550068 non-null  int64
 6   Marital_Status                 550068 non-null  int64
 7   Product_Category_1             550068 non-null  int64
 8   Product_Category_2             550068 non-null  int64
 9   Product_Category_3             550068 non-null  int64
 10  Purchase                       550068 non-null  int64
 11  Stay_In_Current_City_Years_0   550068 non-null  uint8
 12  Stay_In_Current_City_Years_1   550068 non-null  uint8
 13  Stay_In_Current_City_Years_2   550068 non-null  uint8
 14  Stay_In_Current_City_Years_3   550068 non-null  uint8
 15  Stay_In_Current_City_Years_4+  550068 non-null  uint8
dtypes: int64(10), object(1), uint8(5)
memory usage: 48.8+ MB
```

Dropping the irrelevant columns

```
[53] df = df.drop(["User_ID","Product_ID"],axis=1)
```

Splitting data into independent and dependent variables

```
[54] X = df.drop("Purchase",axis=1)
```

```
[55] y=df['Purchase']
```

```
[56] from sklearn.model_selection import train_test_split
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=123)
```

We have split the dataset as ,

TEST DATA -> 30%

TRAIN DATA -> 70%

1. Linear Regression :

Modeling

Linear Regression

```
[57] from sklearn.linear_model import LinearRegression
```

```
[58] lr = LinearRegression()
     lr.fit(X_train,y_train)

LinearRegression()
```

```
[59] lr.intercept_

9536.400764131593
```

```
lr.coef_

array([ 465.82318446,  112.36643445,    5.05508596,  314.06766138,
        -58.23217776, -348.4514785 ,   12.98415047,  143.49190467,
        -20.83796687,    5.4676518 ,   17.68367185,   -3.96751734,
          1.65416056])
```

```
[61] y_pred = lr.predict(X_test)
```

```
[62] from sklearn.metrics import mean_absolute_error,mean_squared_error, r2_score
```

```
[63] mean_absolute_error(y_test, y_pred)

3532.069226165843
```

```
[64] mean_squared_error(y_test, y_pred)

21397853.26940751
```

```
[65] r2_score(y_test, y_pred)

0.15192944521481688
```

```
[66] from math import sqrt
     print("RMSE of Linear Regression Model is ",sqrt(mean_squared_error(y_test, y_pred)))

RMSE of Linear Regression Model is  4625.781368526566
```

2. Decision Tree Regressor :

DecisionTreeRegressor

```
[67] from sklearn.tree import DecisionTreeRegressor

     # create a regressor object
     regressor = DecisionTreeRegressor(random_state = 0)
```

```
[68] regressor.fit(X_train, y_train)
```

```
DecisionTreeRegressor(random_state=0)
```

```
[69] dt_y_pred = regressor.predict(X_test)
```

```
[70] mean_absolute_error(y_test, dt_y_pred)
```

```
2372.0357559134654
```

```
[71] mean_squared_error(y_test, dt_y_pred)
```

```
11300579.466797074
```

```
[72] r2_score(y_test, dt_y_pred)
```

```
0.5521191505924365
```

```
[73] from math import sqrt
     print("RMSE of Linear Regression Model is ",sqrt(mean_squared_error(y_test, dt_y_pred)))
```

```
RMSE of Linear Regression Model is  3361.633452177241
```

3. Random Forest Regressor :

Random Forest Regressor

```
[74] from sklearn.ensemble import RandomForestRegressor

     # create a regressor object
     RFregressor = RandomForestRegressor(random_state = 0)
```

```
[75] RFregressor.fit(X_train, y_train)
```

```
RandomForestRegressor(random_state=0)
```

```
[76] rf_y_pred = RFregressor.predict(X_test)
```

```
[77] mean_absolute_error(y_test, rf_y_pred)
```

```
2222.049109204734
```

```
[78] mean_squared_error(y_test, rf_y_pred)
```

```
9310769.87311957
```

```
[79] r2_score(y_test, rf_y_pred)
```

```
0.6309821516972987
```

```
[80] from math import sqrt
     print("RMSE of Linear Regression Model is ",sqrt(mean_squared_error(y_test, rf_y_pred)))
```

```
RMSE of Linear Regression Model is  3051.35541573242
```

4. XGBoost Regressor :

XGBoost Regressor

```
[81] from xgboost.sklearn import XGBRegressor
```

```
[82] xgb_reg = XGBRegressor(learning_rate=1.0, max_depth=6, min_child_weight=40, seed=0)

     xgb_reg.fit(X_train, y_train)
```

```
[09:40:59] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
XGBRegressor(learning_rate=1.0, max_depth=6, min_child_weight=40, seed=0)
```

```
[83] xgb_y_pred = xgb_reg.predict(X_test)
```

```
[84] mean_absolute_error(y_test, xgb_y_pred)
```

```
2144.8588299087473
```

```
[85] mean_squared_error(y_test, xgb_y_pred)
```

```
8268802.185235631
```

```
[86] r2_score(y_test, xgb_y_pred)
```

```
0.6722789165646108
```

```
[87] from math import sqrt
     print("RMSE of Linear Regression Model is ",sqrt(mean_squared_error(y_test, xgb_y_pred)))
```

```
RMSE of Linear Regression Model is  2875.552500865813
```

The ML algorithm that perform the best was XGBoost Regressor Model with RMSE = 2879