Product sales Prediction using ML algorithm’s

Bhumaraju Mani teja

Date: 06-10-2022

Abstract:

In this I am going to give a detailed report on Product sales prediction using different machine learning algorithms. Dataset is divided in to two files one is training and testing datasets they contain arounds 550069*12 in training data set and 233600*12 in testing dataset. In the both datasets columns are User_ID, Product_ID, Gender, Age, Occupation, City_Category, Stay_In_Current_City_Years, Marital_Status, Product_Category_1, Product_Category_2, Product_Category_3, Purchase these are the 12 columns in our data.

Predictive analytics is the process of using data, regression methods, and machine learning approaches to predict the likelihood of future events based on data. Rather than knowing exactly what has happened, the primary objective is to provide the best prediction of what will happen in the future. We can provide the best income for those who are turning to predictive analytics to boost their bottom line and competitive spirit by using data predictive analytics.

In this report, I established an ideal method for predicting the product for which I want to understand the consumer buy behaviour (particularly, purchase quantity) versus numerous items from various categories. They provided a purchase report of several clients for chosen high volume items from the previous month. Customer demographics (age, gender, marital status, city type, stay in current city), product information (product id and product category), and Total purchase amount from the previous month are also included in the data collection.

Now aim to establish a model to anticipate a customer's purchase amount against various items, which would allow them to generate customized offers for clients against various products.

1.Problem Statement:

That provide a systematic approach to assessing product sales in depth in order to gain information into the biggest indicators for an increase in consumer buy behaviour (particularly, purchase quantity) of a ride, as well as using the concept of machine learning to train the model accordingly and predict when necessary.

2.Market/Customer/Business Need Assessment:

There has been a significant increase in the small business sector, particularly retail shops, and we require specific analysis for their business growth. Our goal was to examine product sales predictions for the system, which will allow small business sales to get a look of fare predictions and plan their prices accordingly.

In this model it consists of different concept of ‘Machine-Learning’ and introducing machine learning model for data analysis and the importance of data produced by the customers in a monthly basis and how this data can be used by the machine learning to tell the business about the exact sales of their business. This provides the business to make better choice of product based on the product predicted by the Machine-Learning model. The proposed system using different machine learning models like Linear Regression, Lasso, Ridge, K-Neighbors Regressor, Decision Tree, RandomForestRegressor, XGBRegressor, CatBoostRegressor, AdaBoostRegressor these are the different machine learning model so that we have different accuracy.

3.Target Specification and characterization:

To understand the customer purchase behaviour (specifically, purchase amount) against various products of different categories

I have to build a model to predict the purchase amount of customer against various products which will help them to create personalized offer for customers against different products using machine learning models.

4.External Search (information sources):

The dataset can be found on the Kaggle. Dataset is divided in to two files one is training and testing datasets they contain arounds 550069*12 in training data set and 233600*12 in testing dataset. In the both datasets columns are User_ID, Product_ID, Gender, Age, Occupation, City_Category, Stay_In_Current_City_Years, Marital_Status, Product_Category_1, Product_Category_2, Product_Category_3, Purchase these are the 12 columns in our data.

The training dataset consists of the following entities:

```
1 df_train = pd.read_csv("train.csv")
2 df_train.head()
```

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Status	Product_Category_1	Product_Category_2	Product_Category_3	Purchase
0	1000001	P00069042	F	0-17	10	A	2	0	3	NaN	NaN	8370
1	1000001	P00248942	F	0-17	10	A	2	0	1	6.0	14.0	15200
2	1000001	P00087842	F	0-17	10	A	2	0	12	NaN	NaN	1422
3	1000001	P00085442	F	0-17	10	A	2	0	12	14.0	NaN	1057
4	1000002	P00285442	M	55+	16	C	4+	0	8	NaN	NaN	7969

```
1 df_train.shape
```

```
(550068, 12)
```

```
1 df_train.columns
```

```
Index(['User_ID', 'Product_ID', 'Gender', 'Age', 'Occupation', 'City_Category',
       'Stay_In_Current_City_Years', 'Marital_Status', 'Product_Category_1',
       'Product_Category_2', 'Product_Category_3', 'Purchase'],
      dtype='object')
```

```
1 df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 550068 entries, 0 to 550067
Data columns (total 12 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  object
 3   Age                         550068 non-null  object
 4   Occupation                  550068 non-null  int64
 5   City_Category               550068 non-null  object
 6   Stay_In_Current_City_Years  550068 non-null  object
 7   Marital_Status              550068 non-null  int64
 8   Product_Category_1          550068 non-null  int64
 9   Product_Category_2          376430 non-null  float64
 10  Product_Category_3          166821 non-null  float64
 11  Purchase                    550068 non-null  int64
dtypes: float64(2), int64(5), object(5)
memory usage: 50.4+ MB
```

```
1 df_train.describe()
```

	User_ID	Occupation	Marital_Status	Product_Category_1	Product_Category_2	Product_Category_3	Purchase
count	5.500680e+05	550068.000000	550068.000000	550068.000000	376430.000000	166821.000000	550068.000000
mean	1.003029e+06	8.076707	0.409653	5.404270	9.842329	12.668243	9263.968713
std	1.727592e+03	6.522660	0.491770	3.936211	5.086590	4.125338	5023.065394
min	1.000001e+06	0.000000	0.000000	1.000000	2.000000	3.000000	12.000000
25%	1.001516e+06	2.000000	0.000000	1.000000	5.000000	9.000000	5823.000000
50%	1.003077e+06	7.000000	0.000000	5.000000	9.000000	14.000000	8047.000000
75%	1.004478e+06	14.000000	1.000000	8.000000	15.000000	16.000000	12054.000000
max	1.006040e+06	20.000000	1.000000	20.000000	18.000000	18.000000	23961.000000

The testing dataset consists of the following entities:

```
df_test.columns
```

```
Index(['User_ID', 'Product_ID', 'Gender', 'Age', 'Occupation', 'City_Category',
       'Stay_In_Current_City_Years', 'Marital_Status', 'Product_Category_1',
       'Product_Category_2', 'Product_Category_3'],
      dtype='object')
```

```
df_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 233599 entries, 0 to 233598
Data columns (total 11 columns):
 #   Column                      Non-Null Count   Dtype  
---  ------                      --------------   -----  
 0   User_ID                     233599 non-null  int64  
 1   Product_ID                  233599 non-null  object 
 2   Gender                      233599 non-null  object 
 3   Age                         233599 non-null  object 
 4   Occupation                  233599 non-null  int64  
 5   City_Category               233599 non-null  object 
 6   Stay_In_Current_City_Years  233599 non-null  object 
 7   Marital_Status              233599 non-null  int64  
 8   Product_Category_1          233599 non-null  int64  
 9   Product_Category_2          161255 non-null  float64
 10  Product_Category_3          71037 non-null   float64
dtypes: float64(2), int64(4), object(5)
memory usage: 19.6+ MB
```

```
df_test.describe()
```

	User_ID	Occupation	Marital_Status	Product_Category_1	Product_Category_2	Product_Category_3
count	2.335990e+05	233599.000000	233599.000000	233599.000000	161255.000000	71037.000000
mean	1.003029e+06	8.085407	0.410070	5.276542	9.849586	12.669454
std	1.726505e+03	6.521146	0.491847	3.736380	5.094943	4.125944
min	1.000001e+06	0.000000	0.000000	1.000000	2.000000	3.000000
25%	1.001527e+06	2.000000	0.000000	1.000000	5.000000	9.000000
50%	1.003070e+06	7.000000	0.000000	5.000000	9.000000	14.000000
75%	1.004477e+06	14.000000	1.000000	8.000000	15.000000	16.000000
max	1.006040e+06	20.000000	1.000000	18.000000	18.000000	18.000000

The complete dataset consists of the following entities:

```
df = df_train.append(df_test)
df.head()
```

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Status	Product_Category_1	Product_Category_2	Product_Category_3	Purchase
0	1000001	P00069042	F	0-17	10	A	2	0	3	NaN	NaN	8370.0
1	1000001	P00248942	F	0-17	10	A	2	0	1	6.0	14.0	15200.0
2	1000001	P00087842	F	0-17	10	A	2	0	12	NaN	NaN	1422.0
3	1000001	P00085442	F	0-17	10	A	2	0	12	14.0	NaN	1057.0
4	1000002	P00285442	M	55+	16	C	4+	0	8	NaN	NaN	7969.0

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 783667 entries, 0 to 233598
Data columns (total 12 columns):
 #   Column                      Non-Null Count   Dtype  
---  ------                      --------------   -----  
 0   User_ID                     783667 non-null  int64  
 1   Product_ID                  783667 non-null  object 
 2   Gender                      783667 non-null  object 
 3   Age                         783667 non-null  object 
 4   Occupation                  783667 non-null  int64  
 5   City_Category               783667 non-null  object 
 6   Stay_In_Current_City_Years  783667 non-null  object 
 7   Marital_Status              783667 non-null  int64  
 8   Product_Category_1          783667 non-null  int64  
 9   Product_Category_2          537685 non-null  float64
 10  Product_Category_3          237858 non-null  float64
 11  Purchase                    550068 non-null  float64
dtypes: float64(3), int64(4), object(5)
memory usage: 77.7+ MB
```

```
df.describe()
```

	User_ID	Occupation	Marital_Status	Product_Category_1	Product_Category_2	Product_Category_3	Purchase
count	7.836670e+05	783667.000000	783667.000000	783667.000000	537685.000000	237858.000000	550068.000000
mean	1.003029e+06	8.079300	0.409777	5.366196	9.844506	12.668605	9263.968713
std	1.727267e+03	6.522206	0.491793	3.878160	5.089093	4.125510	5023.065394
min	1.000001e+06	0.000000	0.000000	1.000000	2.000000	3.000000	12.000000
25%	1.001519e+06	2.000000	0.000000	1.000000	5.000000	9.000000	5823.000000
50%	1.003075e+06	7.000000	0.000000	5.000000	9.000000	14.000000	8047.000000
75%	1.004478e+06	14.000000	1.000000	8.000000	15.000000	16.000000	12054.000000
max	1.006040e+06	20.000000	1.000000	20.000000	18.000000	18.000000	23961.000000

5.Benchmarking:

With EDA

```
df['Age'].unique()
```

```
array(['0-17', '55+', '26-35', '46-50', '51-55', '36-45', '18-25'],
      dtype=object)
```

```
df["Age"] = df["Age"].map({'0-17':1,'18-25':2,'26-35':3,'36-45':4,'46-50':5,'51-55':6,'55+':7})
df.head()
```

	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Status	Product_Category_1	Product_Category_2	Product_Category_3	Purchase
0	P00069042	0	1	10	A	2	0	3	NaN	NaN	8370.0
1	P00248942	0	1	10	A	2	0	1	6.0	14.0	15200.0
2	P00087842	0	1	10	A	2	0	12	NaN	NaN	1422.0
3	P00085442	0	1	10	A	2	0	12	14.0	NaN	1057.0
4	P00285442	1	7	16	C	4+	0	8	NaN	NaN	7969.0

```
df["Age"].value_counts()
```

```
3    313015
4    156724
2    141953
5     65278
6     54784
7     30579
1     21334
Name: Age, dtype: int64
```

```
df["City_Category"].value_counts()
```

```
B    329739
C    243684
A    210244
Name: City_Category, dtype: int64
```

```
df_city=pd.get_dummies(df["City_Category"],drop_first=True)
df_city.head()
```

	B	C
0	0	0
1	0	0
2	0	0
3	0	0
4	0	1

```
df=pd.concat([df,df_city],axis=1)
```

```
df=df.drop(['City_Category'],axis=1)
```

```
df.head()
```

	Product_ID	Gender	Age	Occupation	Stay_In_Current_City_Years	Marital_Status	Product_Category_1	Product_Category_2	Product_Category_3	Purchase	B	C
0	P00069042	0	1	10	2	0	3	NaN	NaN	8370.0	0	0
1	P00248942	0	1	10	2	0	1	6.0	14.0	15200.0	0	0
2	P00087842	0	1	10	2	0	12	NaN	NaN	1422.0	0	0
3	P00085442	0	1	10	2	0	12	14.0	NaN	1057.0	0	0
4	P00285442	1	7	16	4+	0	8	NaN	NaN	7969.0	0	1

```
df.isnull().sum()
```

```
Product_ID                        0
Gender                            0
Age                               0
Occupation                        0
Stay_In_Current_City_Years        0
Marital_Status                    0
Product_Category_1                0
Product_Category_2           245982
Product_Category_3           545809
Purchase                     233599
B                                 0
C                                 0
dtype: int64
```

```
df["Product_Category_2"].unique()
```

```
array([nan,  6., 14.,  2.,  8., 15., 16., 11.,  5.,  3.,  4., 12.,  9.,
       10., 17., 13.,  7., 18.])
```

```
df["Product_Category_2"] = df["Product_Category_2"].fillna(df["Product_Category_2"].mode()[0])
print("Missing values in product category 2 are: - ",df["Product_Category_2"].isnull().sum())
```

```
Missing values in product category 2 are: -  0
```

```
print(df["Product_Category_2"].mode()[0])
```

```
8.0
```

```
df["Product_Category_3"].unique()
```

```
array([nan, 14., 17.,  5.,  4., 16., 15.,  8.,  9., 13.,  6., 12.,  3.,
       18., 11., 10.])
```

```
df["Product_Category_3"].value_counts()
```

```
16.0    46469
15.0    39968
14.0    26283
17.0    23818
5.0     23799
8.0     17861
9.0     16532
12.0    13115
13.0     7849
```

```
df["Product_Category_3"] = df["Product_Category_3"].fillna(df["Product_Category_3"].mode()[0])
print("Missing values in product category 3 are: - ",df["Product_Category_3"].isnull().sum())
```

```
Missing values in product category 3 are: -  0
```

```
print(df["Product_Category_3"].mode()[0])
```

```
16.0
```

```
df.shape
```

```
(783667, 12)
```

```
df.head()
```

	Product_ID	Gender	Age	Occupation	Stay_In_Current_City_Years	Marital_Status	Product_Category_1	Product_Category_2	Product_Category_3	Purchase	B	C
0	P00069042	0	1	10	2	0	3	8.0	16.0	8370.0	0	0
1	P00248942	0	1	10	2	0	1	6.0	14.0	15200.0	0	0
2	P00087842	0	1	10	2	0	12	8.0	16.0	1422.0	0	0
3	P00085442	0	1	10	2	0	12	14.0	16.0	1057.0	0	0
4	P00285442	1	7	16	4+	0	8	8.0	16.0	7969.0	0	1

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 783667 entries, 0 to 233598
Data columns (total 12 columns):
 #   Column                      Non-Null Count   Dtype
---  ------                      --------------   -----
 0   Product_ID                  783667 non-null  object
 1   Gender                      783667 non-null  int64
 2   Age                         783667 non-null  int64
 3   Occupation                  783667 non-null  int64
 4   Stay_In_Current_City_Years  783667 non-null  object
 5   Marital_Status              783667 non-null  int64
 6   Product_Category_1          783667 non-null  int64
 7   Product_Category_2          783667 non-null  float64
 8   Product_Category_3          783667 non-null  float64
 9   Purchase                    550068 non-null  float64
 10  B                           783667 non-null  uint8
 11  C                           783667 non-null  uint8
dtypes: float64(3), int64(5), object(2), uint8(2)
memory usage: 67.3+ MB
```

```
df["Stay_In_Current_City_Years"] = df["Stay_In_Current_City_Years"].map({"0":0,"1":1,"2":2,"3":3,"4+":4})
```

```
df["Stay_In_Current_City_Years"] = df["Stay_In_Current_City_Years"].astype(int)
df.head()
```

	Product_ID	Gender	Age	Occupation	Stay_In_Current_City_Years	Marital_Status	Product_Category_1	Product_Category_2	Product_Category_3	Purchase	B	C
0	P00069042	0	1	10	2	0	3	8.0	16.0	8370.0	0	0
1	P00248942	0	1	10	2	0	1	6.0	14.0	15200.0	0	0
2	P00087842	0	1	10	2	0	12	8.0	16.0	1422.0	0	0
3	P00085442	0	1	10	2	0	12	14.0	16.0	1057.0	0	0
4	P00285442	1	7	16	4	0	8	8.0	16.0	7969.0	0	1

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 783667 entries, 0 to 233598
Data columns (total 12 columns):
 #   Column                      Non-Null Count   Dtype
---  ------                      --------------   -----
 0   Product_ID                  783667 non-null  object
 1   Gender                      783667 non-null  int64
 2   Age                         783667 non-null  int64
 3   Occupation                  783667 non-null  int64
 4   Stay_In_Current_City_Years  783667 non-null  int32
 5   Marital_Status              783667 non-null  int64
 6   Product_Category_1          783667 non-null  int64
 7   Product_Category_2          783667 non-null  float64
 8   Product_Category_3          783667 non-null  float64
 9   Purchase                    550068 non-null  float64
 10  B                           783667 non-null  uint8
 11  C                           783667 non-null  uint8
dtypes: float64(3), int32(1), int64(5), object(1), uint8(2)
memory usage: 64.3+ MB
```

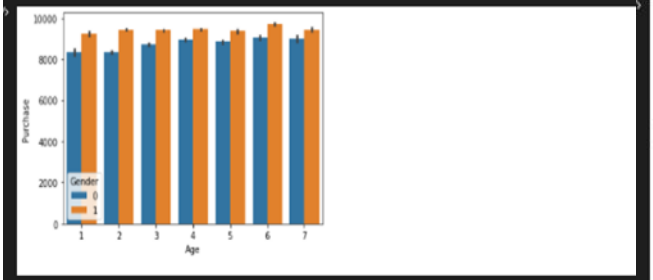
```
df["B"] = df["B"].astype(int)
df.head()
df.describe()
```

	Gender	Age	Occupation	Stay_In_Current_City_Years	Marital_Status	Product_Category_1	Product_Category_2	Product_Category_3	Purchase	B	C
count	783667.00000	783667.000000	783667.000000	783667.000000	783667.000000	783667.000000	783667.000000	783667.000000	550068.000000	783667.000000	783667.000000
mean	0.75291	3.496802	8.079300	1.858247	0.409777	5.366196	9.265541	14.988858	9263.968713	0.420764	0.310954
std	0.43132	1.352736	6.522206	1.288790	0.491793	3.878160	4.301427	2.740792	5023.065394	0.493682	0.462884

Plotting analysis of the data:

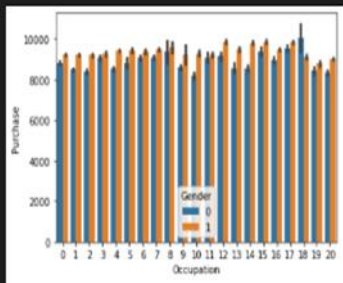
```
## Age vs Purchase
sns.barplot(x="Age",y="Purchase",hue="Gender",data=df)
```

```
<AxesSubplot:xlabel='Age', ylabel='Purchase'>
```

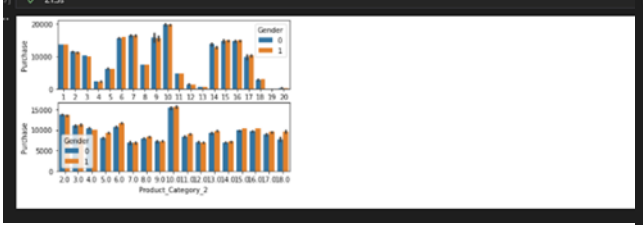



```
sns.barplot(x="Occupation",y="Purchase",hue="Gender",data=df)
```

```
<AxesSubplot:xlabel='Occupation', ylabel='Purchase'>
```




```
plt.subplots(2,1)
plt.subplot(2,1,1)
sns.barplot(x="Product_Category_1",y="Purchase",hue="Gender",data=df)
plt.subplot(2,1,2)
sns.barplot(x="Product_Category_2",y="Purchase",hue="Gender",data=df)
plt.show()
```




```
plt.subplots(2,1)
plt.subplot(2,1,1)
sns.barplot(x="Product_Category_2",y="Purchase",hue="Gender",data=df)
plt.subplot(2,1,2)
sns.barplot(x="Product_Category_3",y="Purchase",hue="Gender",data=df)
plt.show()
```

Trian and test data splitting:

```
from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size=0.33,random_state=42)
```

✓ 1.7s

```
type(X_train)
```

✓ 0.3s

```
pandas.core.frame.DataFrame
```

```
X_train.head()
```

✓ 0.6s

	Gender	Age	Occupation	Stay_In_Current_City_Years	Marital_Status	Product_Category_1	Product_Category_2	Product_Category_3	B	C
396876	1	2	14	3	0	1	2.0	16.0	1	0
433826	1	6	0	0	1	8	16.0	16.0	0	0
516298	1	4	17	0	0	3	4.0	12.0	0	1
193380	1	3	4	1	0	8	16.0	16.0	1	0
273542	0	4	20	3	1	3	4.0	12.0	1	0

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train=scaler.fit_transform(X_train)
X_test=scaler.fit_transform(X_test)
```

✓ 0.5s

The above data gives us a detailed information about the different products of business along with their difference in numbers in the market along with the different sales that the people chosen with this we will do the product sales analysis.

6.Applicable Patents:

To some extent, the current patent may incorporate this patent as insight for the approach in use, as well as EDA analysis.

7.Applicable Regulations:

- Confirm mandatory product regulations and directives.
- Confirm applicable product standards.
- Confirm applicable labelling requirements.
- That country export and import regulations on products.

8.Applicable Constraints:

- Takes much in computation of machine learning model
- As the data is too large so now, we are getting good accuracy that is not up to level so hyper parameter tuning need to done

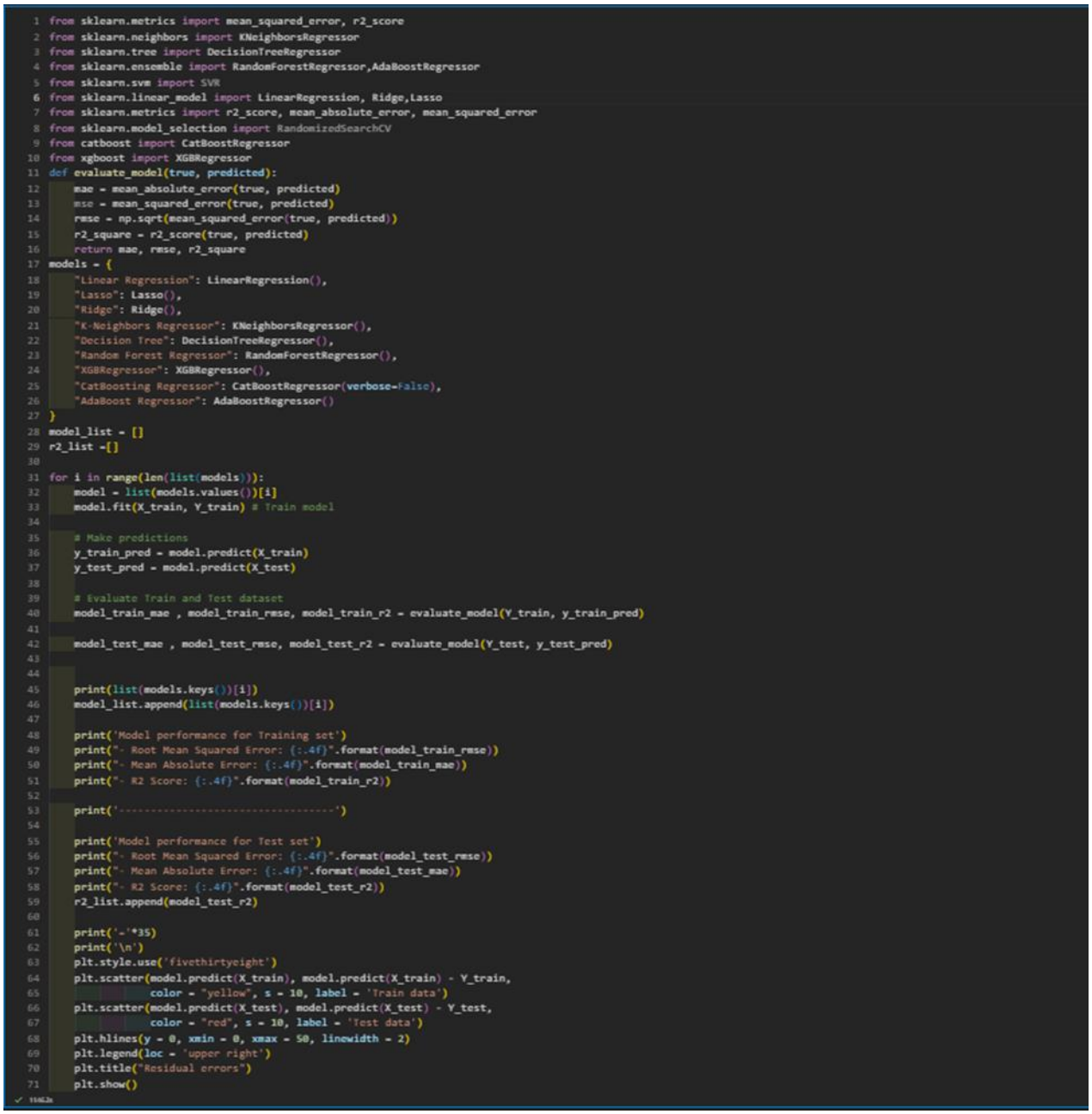
9.Business Opportunity:

After creating web site.so I having a plan to create login authentication so that every user (business owner) will be able to get the details for the sales of a particular product in the visualization of data of the business.so for storing the data I need cloud to store data and retrieve when the user wants to see his sales.

10.Concept Generation:

In order to meet our requirements, this product necessitates the creation of a machine learning model tool from scratch. Fine tuning these models for our purposes is less intimidating than writing entirely new code. A well-trained model can be redeveloped or reconfigured. However, building a model with the resources and data we have is time-consuming but manageable. The customer may prefer to spend as much time as possible giving input data. This accuracy will require some effort to achieve because relying on the Machine Learning algorithm is difficult.

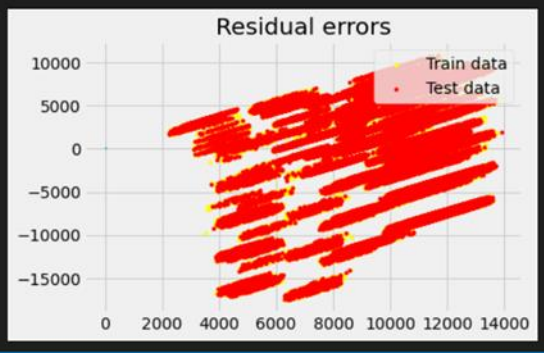
Model with different regression algorithms:



```python
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor,AdaBoostRegressor
from sklearn.svm import SVR
from sklearn.linear_model import LinearRegression, Ridge,Lasso
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
from sklearn.model_selection import RandomizedSearchCV
from catboost import CatBoostRegressor
from xgboost import XGBRegressor
def evaluate_model(true, predicted):
    mae = mean_absolute_error(true, predicted)
    mse = mean_squared_error(true, predicted)
    rmse = np.sqrt(mean_squared_error(true, predicted))
    r2_square = r2_score(true, predicted)
    return mae, rmse, r2_square
models = {
    "Linear Regression": LinearRegression(),
    "Lasso": Lasso(),
    "Ridge": Ridge(),
    "K-Neighbors Regressor": KNeighborsRegressor(),
    "Decision Tree": DecisionTreeRegressor(),
    "Random Forest Regressor": RandomForestRegressor(),
    "XGBRegressor": XGBRegressor(),
    "CatBoosting Regressor": CatBoostRegressor(verbose=False),
    "AdaBoost Regressor": AdaBoostRegressor()
}
model_list = []
r2_list =[]

for i in range(len(list(models))):
    model = list(models.values())[i]
    model.fit(X_train, Y_train) # Train model

    # Make predictions
    y_train_pred = model.predict(X_train)
    y_test_pred = model.predict(X_test)

    # Evaluate Train and Test dataset
    model_train_mae , model_train_rmse, model_train_r2 = evaluate_model(Y_train, y_train_pred)

    model_test_mae , model_test_rmse, model_test_r2 = evaluate_model(Y_test, y_test_pred)


    print(list(models.keys())[i])
    model_list.append(list(models.keys())[i])

    print('Model performance for Training set')
    print("- Root Mean Squared Error: {:.4f}".format(model_train_rmse))
    print("- Mean Absolute Error: {:.4f}".format(model_train_mae))
    print("- R2 Score: {:.4f}".format(model_train_r2))

    print('----------------------------------')

    print('Model performance for Test set')
    print("- Root Mean Squared Error: {:.4f}".format(model_test_rmse))
    print("- Mean Absolute Error: {:.4f}".format(model_test_mae))
    print("- R2 Score: {:.4f}".format(model_test_r2))
    r2_list.append(model_test_r2)

    print('='*35)
    print('\n')
    plt.style.use('fivethirtyeight')
    plt.scatter(model.predict(X_train), model.predict(X_train) - Y_train,
                color = "yellow", s = 10, label = 'Train data')
    plt.scatter(model.predict(X_test), model.predict(X_test) - Y_test,
                color = "red", s = 10, label = 'Test data')
    plt.hlines(y = 0, xmin = 0, xmax = 50, linewidth = 2)
    plt.legend(loc = 'upper right')
    plt.title("Residual errors")
    plt.show()
```

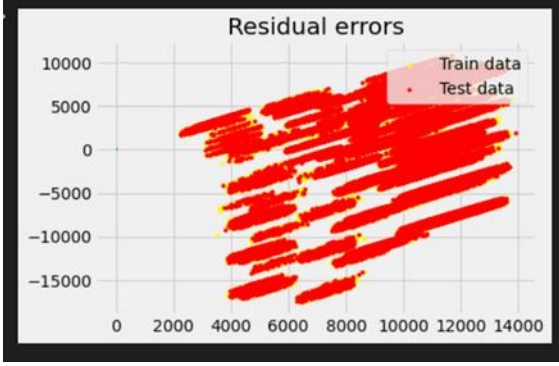
The Accuracy and residual error graph of the different model is given below:

```
Linear Regression
Model performance for Training set
- Root Mean Squared Error: 4680.8269
- Mean Absolute Error: 3577.7367
- R2 Score: 0.1321
---------------------------------
Model performance for Test set
- Root Mean Squared Error: 4683.9207
- Mean Absolute Error: 3577.4986
- R2 Score: 0.1295
=================================
```

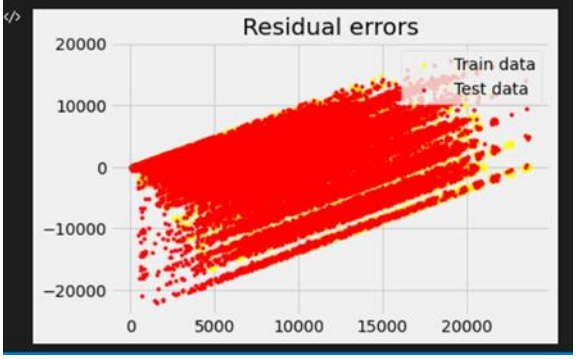



```
Lasso
Model performance for Training set
- Root Mean Squared Error: 4680.8281
- Mean Absolute Error: 3577.8068
- R2 Score: 0.1321
---------------------------------
Model performance for Test set
- Root Mean Squared Error: 4683.9217
- Mean Absolute Error: 3577.5710
- R2 Score: 0.1295
=================================
```

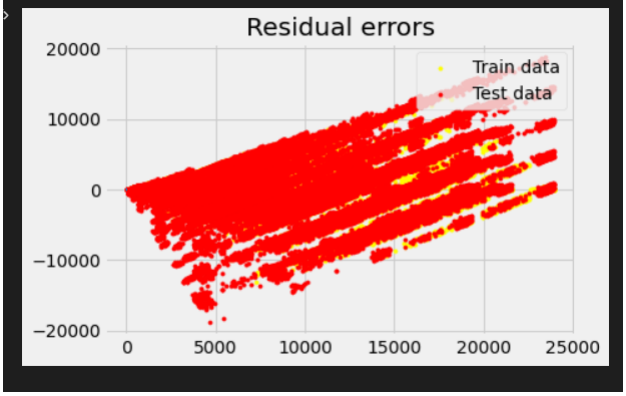

```
Ridge
Model performance for Training set
- Root Mean Squared Error: 4680.8269
- Mean Absolute Error: 3577.7373
- R2 Score: 0.1321
---------------------------------
Model performance for Test set
- Root Mean Squared Error: 4683.9207
- Mean Absolute Error: 3577.4992
- R2 Score: 0.1295
=================================
```



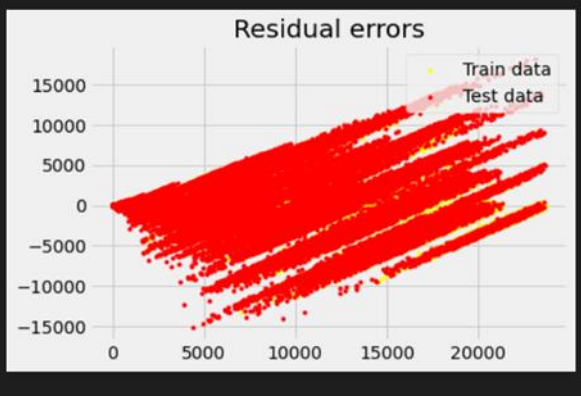

```
K-Neighbors Regressor
Model performance for Training set
- Root Mean Squared Error: 2984.9911
- Mean Absolute Error: 2191.3834
- R2 Score: 0.6471
---------------------------------
Model performance for Test set
- Root Mean Squared Error: 3519.6934
- Mean Absolute Error: 2560.3232
- R2 Score: 0.5085
=================================
```



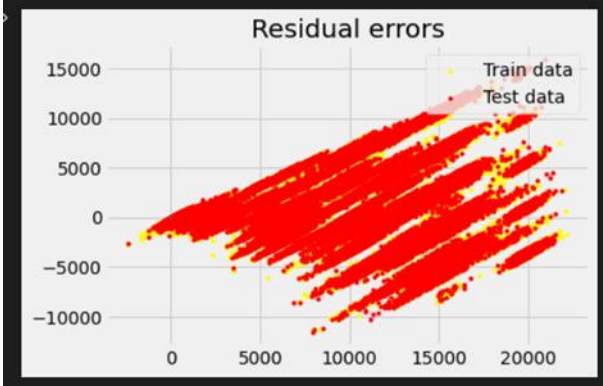
```
Decision Tree
Model performance for Training set
- Root Mean Squared Error: 2256.1816
- Mean Absolute Error: 1512.3720
- R2 Score: 0.7984
----------------------------------
Model performance for Test set
- Root Mean Squared Error: 3334.8826
- Mean Absolute Error: 2361.1494
- R2 Score: 0.5587
==================================
```



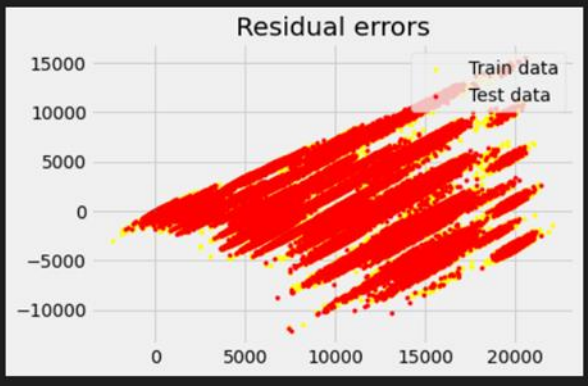

```
Random Forest Regressor
Model performance for Training set
- Root Mean Squared Error: 2315.8510
- Mean Absolute Error: 1667.3946
- R2 Score: 0.7876
----------------------------------
Model performance for Test set
- Root Mean Squared Error: 3056.4650
- Mean Absolute Error: 2226.5392
- R2 Score: 0.6293
==================================
```



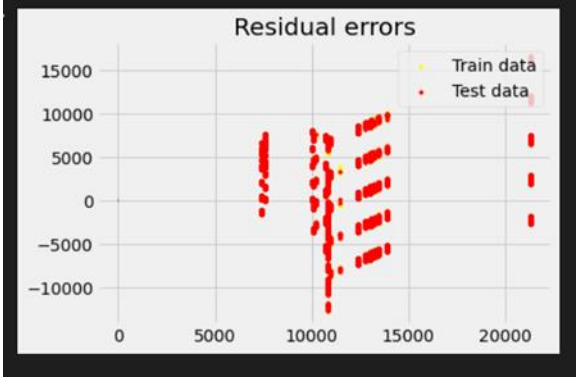

```
XGBRegressor
Model performance for Training set
- Root Mean Squared Error: 2842.9484
- Mean Absolute Error: 2128.8696
- R2 Score: 0.6798
----------------------------------
Model performance for Test set
- Root Mean Squared Error: 2897.1569
- Mean Absolute Error: 2166.3152
- R2 Score: 0.6670
==================================
```




```
CatBoosting Regressor
Model performance for Training set
- Root Mean Squared Error: 2851.5427
- Mean Absolute Error: 2136.6589
- R2 Score: 0.6779
----------------------------------
Model performance for Test set
- Root Mean Squared Error: 2896.4133
- Mean Absolute Error: 2167.5156
- R2 Score: 0.6671
==================================
```



```
AdaBoost Regressor
Model performance for Training set
- Root Mean Squared Error: 3864.5241
- Mean Absolute Error: 3037.5814
- R2 Score: 0.4084
----------------------------------
Model performance for Test set
- Root Mean Squared Error: 3871.9636
- Mean Absolute Error: 3041.2050
- R2 Score: 0.4051
==================================
```




```
pd.DataFrame(list(zip(model_list, r2_list)), columns=['Model Name', 'R2_Score']).sort_values(by=["R2_Score"],ascending=False)
```

	Model Name	R2_Score
7	CatBoosting Regressor	0.667129
6	XGBRegressor	0.666958
5	Random Forest Regressor	0.629325
4	Decision Tree	0.558718
3	K-Neighbors Regressor	0.508454
8	AdaBoost Regressor	0.405136
2	Ridge	0.129490
0	Linear Regression	0.129490
1	Lasso	0.129489

11.Concept Development:

The model can be developed by using Django as framework for its deployment to make webpage so that they can keep the values and get their desired predicted sale report no need of run whole code again and again.

12.Final Report Prototype:

The following functions are required for the product to be perfect and provide a good result.

Back end development

Model Development using Django and need to connect with database for user authentication system This must be completed prior to the release of the service. For export the machine learning model need example to use joblib so that we can deploy to website using Django.

EDA done but need add hyper parameter tuning for Algorithm training and optimization must be performed to reduce model overfitting issues.

Front End development

User interfaces need to keep in both mobile view and desktop view input parameters must be simple to the user in a variety of options and some examples to be given in above the data entry input parameters.

As we taken ml models in some (. extension format) which we will be getting after performing the joblib so no need of running and training and testing the model

13.Product details - How does it work?

An responsive user system will receive product input from the user, after which our model will perform computation using various machine learning algorithms and will be able to produce some analytical graphs for business sales success with a good user interactive UI.

14.References/Source of Information:

1.CatBoost regression in 6 minutes. A brief hands-on introduction to… | by Simon Thiesen | Towards Data Science

2. sklearn.ensemble.AdaBoostRegressor — scikit-learn 1.1.2 documentation

3. Python | Decision Tree Regression using sklearn - GeeksforGeeks

4. Predicting Online Product Sales using Machine Learning – IJERT

5. What is Hyper Parameter Tuning in Machine Learning? - TechVenture(need to be done)

15.Github link: Manitejabhumaraju/feyyn: internship (github.com)

.