Online Vehicle Booking Market:

Predicting the city that can generate early foot in the market and revenue.
Using Machine Learning

-Abhishekh Kumar Singh

Rajiv Gandhi Institute Of Petroleum Technology (RGIPT-Jais)

22.09.23

Introduction

In the rapidly evolving landscape of the Indian vehicle booking market, our endeavor is to conduct a comprehensive Segmentation Analysis to identify strategic entry points. This analysis aims to scrutinize diverse customer, vehicle, and B2B segments, drawing insights from Geographic, Demographic, Psychographic, and Behavioral facets. By harnessing the power of data, we seek to redefine segmentation categories based on the availability of rich datasets. Our ultimate goal is to formulate a robust and practical market strategy that capitalizes on the segments with the highest profit potential. This dynamic approach recognizes that market segmentation is not a static concept; it evolves in tandem with the ever-changing data landscape.

Data Source:

Indian Cities Data: *Top 500 Indian Cities | Kaggle* (bit.ly/3rsdNWX)
Road Data : data.gov.in (https://bit.ly/3ZwVr3M)

Data Preprocessing:

Analysis began with the collection of two datasets from data.gov.in & KAGGLE.

The first dataset contains detailed city information, including attributes such as population, literacy rates, and gender demographics.

The second dataset focuses on the road network within each city, providing insights into the state's infrastructure.

These datasets were initially separate but were harmonized to create a unified dataset for analysis.

1. Data Collection:
    - I obtained two datasets: one comprising city details and the other representing road network information.
    - The city dataset includes features like population, literacy rates, and sex ratio.
    - The road network dataset provides insights into the city's transportation infrastructure.
2. Data Alignment:
    - To ensure compatibility, we removed cities and states that were not present in both datasets.
    - I corrected state names to ensure uniformity across both datasets.

3. Data Cleaning:
    - Performed data cleaning by removing extra spaces and ensuring data consistency.
    - The Fuzzy-Wuzzy algorithm was used to match and combine city information from both datasets. (when matched at least 80%)
    -
```
from fuzzywuzzy import fuzz
from fuzzywuzzy import process

# Function to perform fuzzy string matching and find the best match
def fuzzy_match_state_name(state_name, choices):
    match, score = process.extractOne(state_name, choices)
    return match if score >= 80 else None
```
4. Data Encoding:
    - Non-numerical values were encoded using one-hot encoding to prepare the data for machine learning.
5. Data Splitting:
    - The dataset was split into training and testing sets to facilitate model development and evaluation.

-These preprocessing steps were essential to create a consistent and well-structured dataset for our segmentation and analysis tasks. The resulting dataset is ready for further exploration and model development.

Custom Scaling:

- In our data preprocessing pipeline, we applied custom scaling techniques to certain features with different scales. This custom scaling approach allowed us to give specific importance to selected features while keeping others within a defined range. The primary goal was to normalize and control the impact of these features on our segmentation and analysis tasks.

- Two features, "0-6_population_total" and "effective_literacy_rate_total," were identified for custom scaling to increase their influence on the analysis. These features were considered crucial for our strategy formulation, and we wanted to ensure they had a more significant impact on the final results.

- Applied the Min-Max scaling technique with a custom feature range of 5 to 10 for these two features, effectively scaling them within this predefined range. This custom scaling approach allowed us to fine-tune the importance of specific features according to their relevance in our analysis.

    - By custom scaling these features, we ensured that they would play a more prominent role in our segmentation and analysis, ultimately leading to more meaningful insights and a more effective strategy formulation for entering the vehicle booking market in India.

```
from sklearn.preprocessing import MinMaxScaler
import numpy as np

# Create a copy of your original DataFrame
X_train_custom_scaled = X_train.copy()
X_test_custom_scaled = X_test.copy()

# Specify the columns you want to give more weight
columns_to_scale = ['0-6_population_total', 'effective_literacy_rate_total']

# Apply Min-Max scaling to the specified columns
scaler = MinMaxScaler(feature_range=(5, 10))

# Fit and transform the training data
X_train_custom_scaled[columns_to_scale] = scaler.fit_transform(X_train_custom_scaled[columns_to_scale])

# Transform the test data using the same scaler
X_test_custom_scaled[columns_to_scale] = scaler.transform(X_test_custom_scaled[columns_to_scale])
```

Judging Criteria

❖ Covering cities of 26 indian states

```
array(['Punjab', 'Maharashtra', 'Andhra pradesh', 'Tripura',
       'Uttar pradesh', 'Gujarat', 'Mizoram', 'Rajasthan', 'Tamil nadu',
       'Kerala', 'Haryana', 'Chhattisgarh', 'Jammu & kashmir', 'Bihar',
       'West bengal', 'Karnataka', 'Orissa', 'Madhya pradesh',
       'Chandigarh', 'Uttarakhand', 'Assam', 'Nagaland', 'Manipur',
       'Andaman & nicobar islands', 'Meghalaya', 'Himachal pradesh'],
      dtype=object))
```

❖ Weights are assigned to these

- Lower the value of state weight means higher the chance of success:

Name of the States	state_weight
Andhra Pradesh	8
Assam	19
Bihar	25
Chhattisgarh	11
Gujarat	3
Haryana	5
Himachal Pradesh	16
Jammu & Kashmir	14
Jharkhand	17
Karnataka	2
Kerala	7
Madhya Pradesh	12
Maharashtra	1
Manipur	20
Meghalaya	21
Mizoram	22
Nagaland	23
Orissa	18
Punjab	6
Rajasthan	9
Tamil Nadu	4
Tripura	23
Uttar Pradesh	15
Uttarakhand	10
West Bengal	13
Andaman & nicoba	27
Chandigarh	26

❖ Final Dataset looks like this:

	name_of_city	state_name_x	population_total	0-6_population_tot	sex_ratio	effective_literacy_r	state_we	National	State Hig	District R	Rural Po	Urban ro	Project F	Total roa	Total Are	Road De	Road De	Road De	Road De	Road De	Road Density per 10	Road Density per 1000 Sq. Km - Project Roads
2	Abohar	punjab	145238	15870	890	79.86	6	3274	1133	6808	104565	18887	13195	147862	50362	2936	65	22.5	135	2076.3	375.02	262
3	Achalpur	maharashtra	112293	11810	928	91.99	1	17757	32005	108419	426327	30386	21993	636887	307713	2069.7	57.7	104	352	1385.5	98.75	71.47
4	Adilabad	andhra pradesh	117388	13103	982	80.51	8	10709	15649	58083	180833	22724	28907	316906	275045	1152.2	38.9	56.9	211	657.47	82.62	105.1
5	Adityapur	jharkhand	173988	23042	902	83.46	17	3367	1232	10969	44317	7433	13928	81245	79716	1019.2	42.2	15.5	138	555.93	93.24	174.73
6	Adoni	andhra pradesh	166537	18406	1013	68.38	8	10709	15649	58083	180833	22724	28907	316906	275045	1152.2	38.9	56.9	211	657.47	82.62	105.1
7	Agartala	tripura	339688	33635	1002	93.88	23	854	1057	461	40236	1073	1439	45120	10486	4302.9	81.4	100.8	44	3837.1	102.36	137.2
8	Agra	uttar pradesh	1574542	186516	853	63.44	15	11737	7427	55816	255576	70178	42173	442907	240928	1838.3	48.7	30.8	232	1060.8	291.28	175.04
9	Ahmadabad	gujarat	5570585	589076	897	89.62	3	6635	16746	31262	132467	30920	31343	249373	196244	1270.7	33.8	85.3	159	675.01	157.56	159.71
10	Ahmadnagar	maharashtra	350905	36712	952	91.49	1	17757	32005	108419	426327	30386	21993	636887	307713	2069.7	57.7	104	352	1385.5	98.75	71.47
11	Aizawl	mizoram	291822	35147	1029	98.8	22	1423	170	1714	11849	11	1082	16250	21081	770.82	67.5	8.1	81	562.09	0.52	51.34
12	Ajmer	rajasthan	542580	59437	946	87.53	9	10342	15061	24448	220402	32772	10444	313469	342239	915.94	30.2	44	71	644	95.76	30.52
13	Akbarpur	uttar pradesh	111594	14037	939	76.94	15	11737	7427	55816	255576	70178	42173	442907	240928	1838.3	48.7	30.8	232	1060.8	291.28	175.04
14	Akola	maharashtra	427146	46500	958	91.34	1	17757	32005	108419	426327	30386	21993	636887	307713	2069.7	57.7	104	352	1385.5	98.75	71.47
15	Alandur	tamil nadu	164162	14486	997	95.15	4	6742	11169	48236	170246	24693	10052	271137	130060	2084.7	51.8	85.9	371	1309	189.86	77.29
16	Alappuzha	kerala	174164	15434	1076	96.56	7	1782	4342	27470	188517	28785	9036	259332	38852	6690.3	45.9	111.7	707	4852.2	740.89	232.56
17	Aligarh	uttar pradesh	872575	113658	884	70.36	15	11737	7427	55816	255576	70178	42173	442907	240928	1838.3	48.7	30.8	232	1060.8	291.28	175.04
18	Allahabad	uttar pradesh	1117094	102556	858	86.5	15	11737	7427	55816	255576	70178	42173	442907	240928	1838.3	48.7	30.8	232	1060.8	291.28	175.04
19	Alwar	rajasthan	315310	34576	889	86.78	9	10342	15061	24448	220402	32772	10444	313469	342239	915.94	30.2	44	71	644	95.76	30.52
20	Ambala	haryana	196216	19645	895	88.23	5	3166	1602	22051	7027	14853	1594	50292	44212	1137.5	71.6	36.2	499	158.93	335.95	36.06
21	Ambala Sadar	haryana	104268	9646	922	87.97	5	3166	1602	22051	7027	14853	1594	50292	44212	1137.5	71.6	36.2	499	158.93	335.95	36.06
22	Ambarnath	maharashtra	254003	27465	910	89.04	1	17757	32005	108419	426327	30386	21993	636887	307713	2069.7	57.7	104	352	1385.5	98.75	71.47
23	Ambattur	tamil nadu	478134	45216	982	92.69	4	6742	11169	48236	170246	24693	10052	271137	130060	2084.7	51.8	85.9	371	1309	189.86	77.29
24	Ambikapur	chhattisgarh	114575	12395	920	88.2	11	3606	4176	12221	56575	12824	15672	105074	135192	777.22	26.7	30.9	90	418.48	94.86	115.92
25	Ambur	tamil nadu	113856	12150	1031	86.83	4	6742	11169	48236	170246	24693	10052	271137	130060	2084.7	51.8	85.9	371	1309	189.86	77.29
26	Amravati	maharashtra	646801	62497	957	93.03	1	17757	32005	108419	426327	30386	21993	636887	307713	2069.7	57.7	104	352	1385.5	98.75	71.47
27	Amreli	gujarat	105980	9885	921	83.47	3	6635	16746	31262	132467	30920	31343	249373	196244	1270.7	33.8	85.3	159	675.01	157.56	159.71
28	Amritsar	punjab	1132761	109540	879	85.27	6	3274	1133	6808	104565	18887	13195	147862	50362	2936	65	22.5	135	2076.3	375.02	262
29	Amroha	uttar pradesh	197135	26855	916	63.88	15	11737	7427	55816	255576	70178	42173	442907	240928	1838.3	48.7	30.8	232	1060.8	291.28	175.04
30	Anand	gujarat	197351	19893	928	93.71	3	6635	16746	31262	132467	30920	31343	249373	196244	1270.7	33.8	85.3	159	675.01	157.56	159.71
31	Anantapur	andhra pradesh	262340	23630	995	81.88	8	10709	15649	58083	180833	22724	28907	316906	275045	1152.2	38.9	56.9	211	657.47	82.62	105.1
32	Anantnag	jammu & kashmir	108505	18056	937	78.95	14	2423	310	19695	75684	2096	19826	120034	222236	540.12	10.9	1.4	89	340.56	3.43	89.21
33	Arrah	bihar	261099	34413	874	83.41	25	5358	4006	12797	253507	14308	2230	298205	94163	3166.9	56.9	42.5	136	2755.9	151.95	23.68
34	Asansol	west bengal	564431	56014	922	84.82	13	3664	3612	10449	157497	95065	13576	283865	88752	3198.4	41.3	40.7	118	1774.6	1071.13	152.97
35	Ashoknagar Kalya	west bengal	123906	8885	981	92.45	13	3664	3612	10449	157497	95065	13576	283865	88752	3198.4	41.3	40.7	118	1774.6	1071.13	152.97
36	Aurangabad	bihar	101520	14292	904	86.89	25	5358	4006	12797	253507	14308	2230	298205	94163	3166.9	56.9	42.5	136	2755.9	151.95	23.68
37	Aurangabad	maharashtra	1171330	151827	919	89.13	1	17757	32005	108419	426327	30386	21993	636887	307713	2069.7	57.7	104	352	1385.5	98.75	71.47
38	Avadi	tamil nadu	344701	33414	969	88.63	4	6742	11169	48236	170246	24693	10052	271137	130060	2084.7	51.8	85.9	371	1309	189.86	77.29

Model Selection and Evaluation:

❖ Problem Type: Regression

Algorithms Tested:

- Linear Regression

```
Linear Regression : Mean Squared Error: 0.3237447701611982
```

```
model = LinearRegression()

# Train
model.fit(X_train_custom_scaled, y_train)

y_pred = model.predict(X_test_custom_scaled)

mse = mean_squared_error(y_test, y_pred)
print(f'Mean Squared Error: {mse}')
```

```
Mean Squared Error: 0.31007198687164195
```

- Ridge Lasso Regression

Using Ridge Lasso Regression: Mean Squared Error: 8.008902311029392

```
[ ] from sklearn.linear_model import Ridge, Lasso

    # Ridge Regression
    ridge_model = Ridge(alpha=1.0)
    ridge_model.fit(X_train_custom_scaled, y_train)

    # Lasso Regression
    lasso_model = Lasso(alpha=1.0)
    lasso_model.fit(X_train_custom_scaled, y_train)

    y_pred =lasso_model.predict(X_test_custom_scaled)

    mse = mean_squared_error(y_test, y_pred)
    print(f'Mean Squared Error: {mse}')

    Mean Squared Error: 4.979988748326029
```

- Decision Tree & RandomForest

Using Decision Tree & RandomForest : Mean Squared Error: 0.07004042553191495

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor

# Decision Tree Regression
dt_model = DecisionTreeRegressor()
dt_model.fit(X_train_custom_scaled, y_train)

# Random Forest Regression
rf_model = RandomForestRegressor(n_estimators=100)
rf_model.fit(X_train_custom_scaled, y_train)

# predictions
y_pred =rf_model.predict(X_test_custom_scaled)

mse = mean_squared_error(y_test, y_pred)
print(f'Mean Squared Error: {mse}')

Mean Squared Error: 0.06222571428571434
```

- Support vector Regression

Using Support vector Regression: Mean Squared Error: 0.20459407050178582

```
from sklearn.svm import SVR

svr_model = SVR(kernel='polynimial')
svr_model.fit(X_train_custom_scaled, y_train)

y_pred =svr_model.predict(X_test_custom_scaled)

mse = mean_squared_error(y_test, y_pred)
print(f'Mean Squared Error: {mse}')
```

- GradientBoostingRegressor

BEST: Using GradientBoostingRegressor : Mean Squared Error: 0.008281055716779595

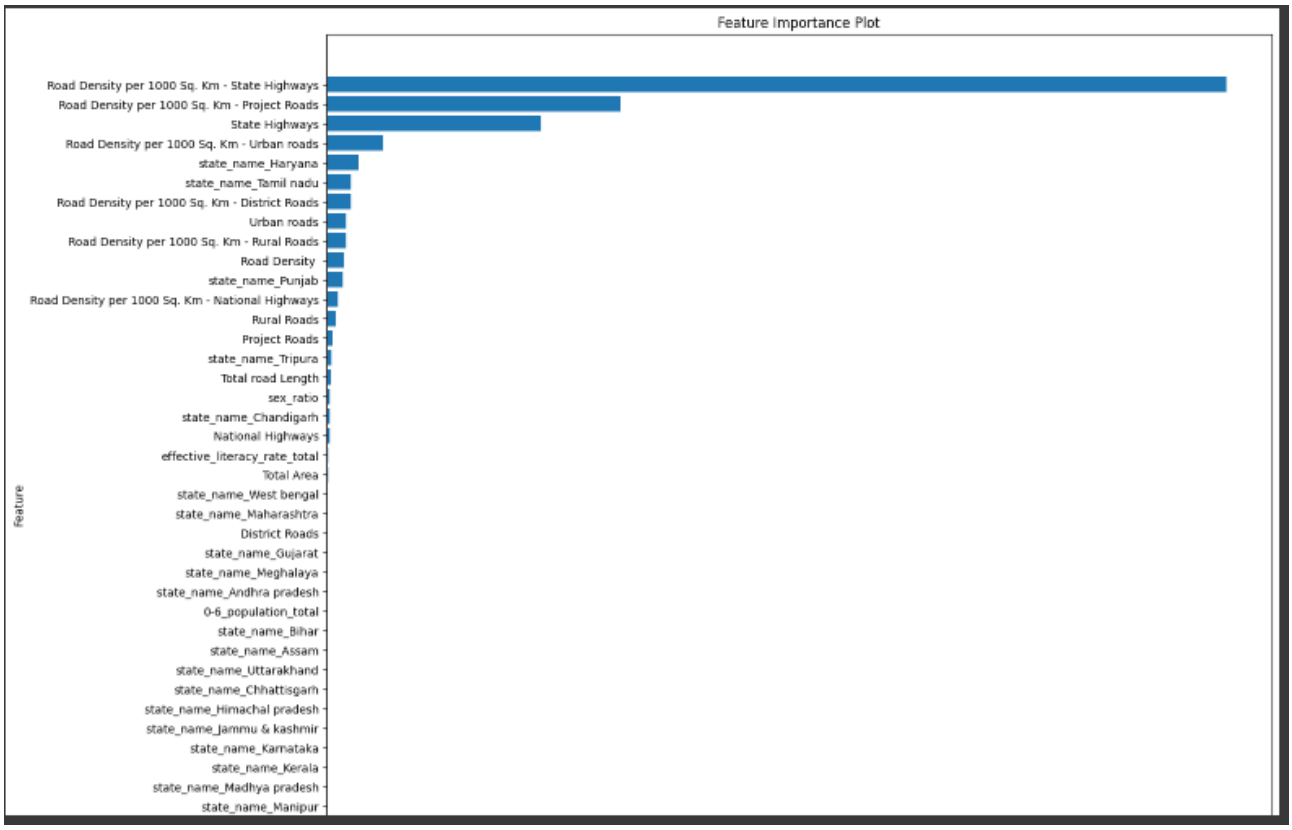
```
[ ]  from sklearn.ensemble import GradientBoostingRegressor

     # Gradient Boosting Regression
     gb_model = GradientBoostingRegressor(n_estimators=100, learning_rate=0.01)
     gb_model.fit(X_train_custom_scaled, y_train)


     # Make predictions on the test set
     y_pred =gb_model.predict(X_test_custom_scaled)

     # Evaluate the model using Mean Squared Error (MSE) as an example
     mse = mean_squared_error(y_test, y_pred)
     print(f'Mean Squared Error: {mse}')
```

- Result: GradientBoostingRegressor performs best regression with Mean Squared Error: 0.008281055716779595
- Predicted Array :

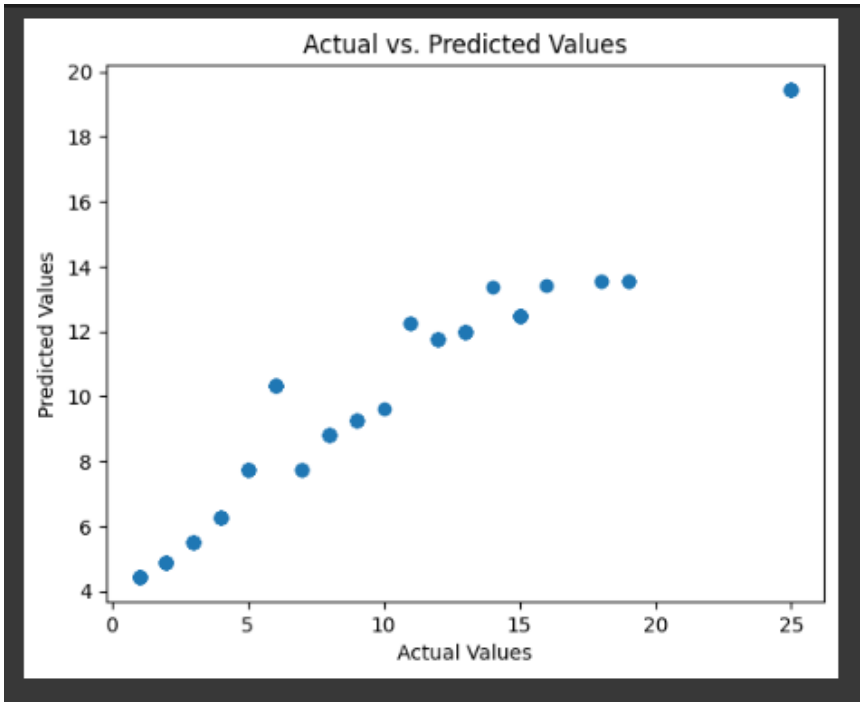
```
[25] y_pred

array([11.9846838 , 11.9846838 , 19.4299747 ,  8.80126826, 19.4299747 ,
       12.47489196,  7.72106386,  9.2588698 ,  4.43361716,  8.80126826,
       19.4299747 , 11.7767804 , 19.4299747 , 12.26666786,  4.43361716,
       12.47489196,  4.43361716, 12.47489196,  5.52253879, 11.9846838 ,
       10.31706522,  4.43361716,  4.43361716,  7.72106386,  4.8864563 ,
        6.2689586 ,  9.2588698 , 11.7767804 , 19.4299747 ,  7.73212128,
        9.2588698 ,  9.63027463,  4.43361716, 13.56205466,  7.72106386,
        4.8864563 ,  5.52253879,  4.8864563 ,  4.8864563 ,  7.73212128,
        5.52253879,  4.8864563 ,  5.52253879, 19.4299747 ,  7.72106386,
        8.80126826, 10.31706522,  8.80126826, 12.47489196, 11.7767804 ,
        5.52253879,  8.80126826, 11.9846838 , 11.7767804 ,  7.72106386,
       11.7767804 ,  9.2588698 ,  6.2689586 , 13.42101905,  4.8864563 ,
        5.52253879, 12.47489196,  5.52253879,  8.80126826, 10.31706522,
       13.56205466, 11.7767804 ,  6.2689586 ,  5.52253879, 11.9846838 ,
       12.47489196,  6.2689586 , 11.9846838 ,  4.43361716, 12.26666786,
       19.4299747 , 12.47489196,  6.2689586 , 11.9846838 ,  4.8864563 ,
       11.9846838 ,  4.43361716,  8.80126826, 10.31706522, 12.47489196,
       10.31706522, 19.4299747 , 12.47489196,  8.80126826,  4.43361716,
       11.9846838 , 12.26666786, 19.4299747 , 13.56205466, 11.7767804 ,
        6.2689586 , 19.4299747 , 11.9846838 ,  7.72106386,  6.2689586 ,
       12.47489196, 13.56205466, 11.9846838 ,  5.52253879,  4.43361716,
        5.52253879,  7.72106386, 10.31706522, 11.7767804 ,  9.2588698 ,
       11.9846838 ,  8.80126826,  4.43361716,  8.80126826,  4.43361716,
        4.43361716,  4.8864563 ,  4.43361716, 12.47489196,  4.8864563 ,
        8.80126826, 13.56205466,  6.2689586 , 12.47489196, 11.9846838 ,
       19.4299747 , 12.47489196, 11.7767804 , 12.47489196,  9.2588698 ,
        7.73212128,  4.43361716,  8.80126826,  8.80126826, 10.31706522,
       12.47489196, 11.7767804 , 11.9846838 , 13.35383056, 10.31706522])
```

VISUALS.

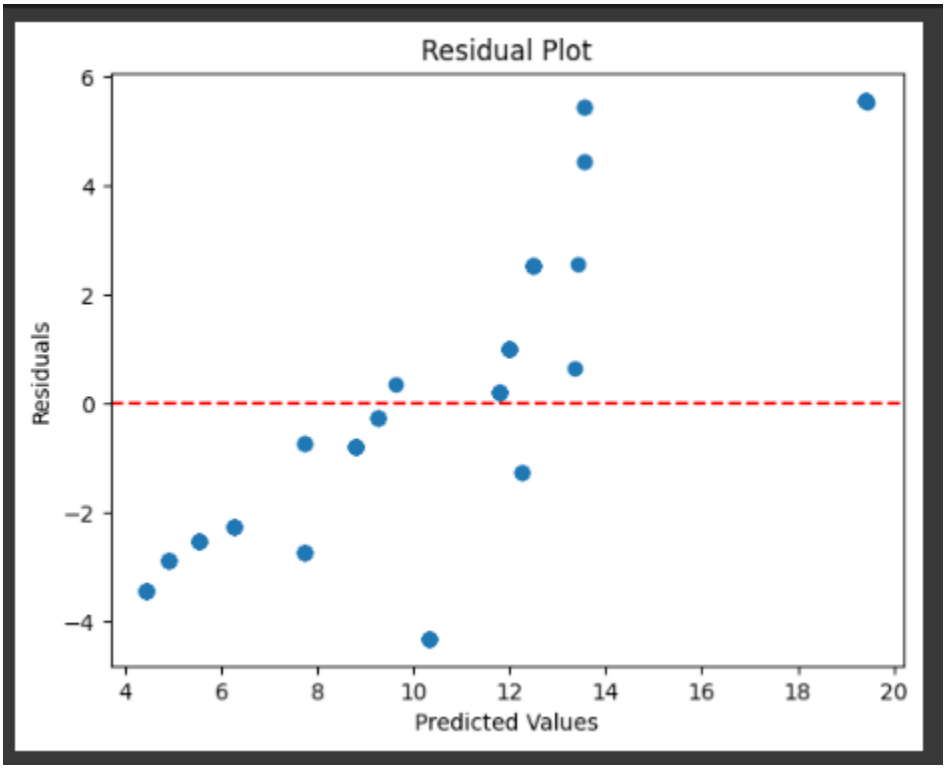
- Feature Importance Plot



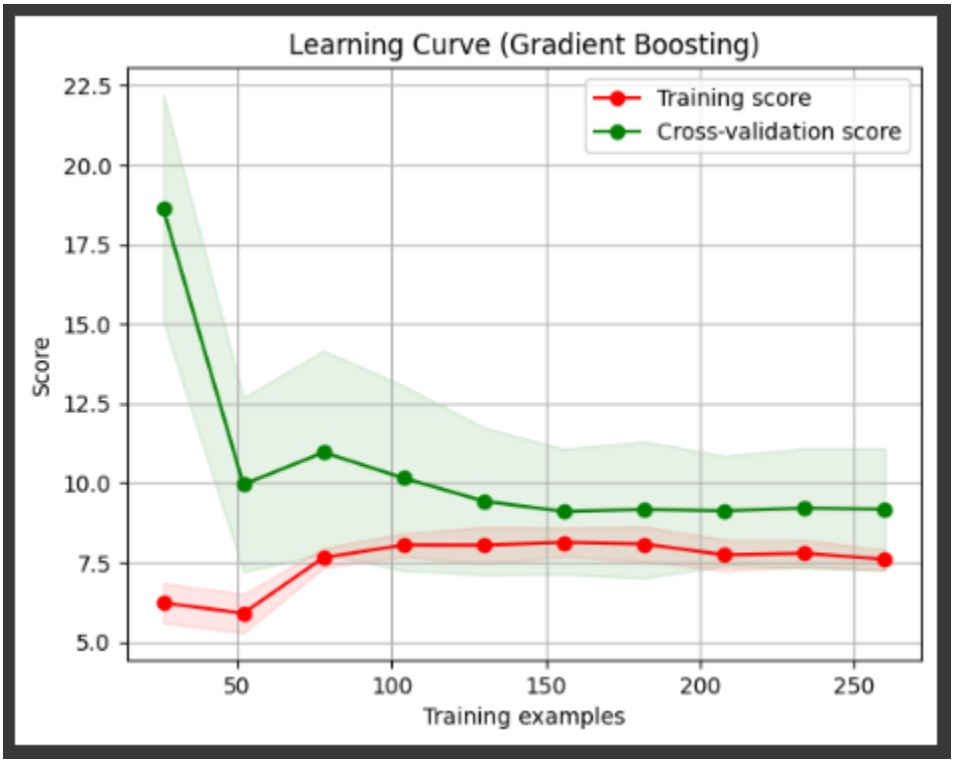
- Actual Vs Predicted:



- Predicted & Residual values:



- Learning Curve:



CODE IMPEMENTAON:

GITHUB LINK

Colab Link