In [2]:

```
import pandas as ps
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Task:-1.Data Preprocessing:

Data preprocessing is a critical step in the data analysis and machine learning pipeline. It involves cleaning, organizing, and transforming raw data into a format suitable for analysis or model training.

In [186]:

```
car=pd.read_csv("D:\extract file of 8 may 2023 fast track batch\car data (1) assienment 7.csv"
```

In [187]:

```
car.head(101)
```

Out[187]:

	Car_Name	Year	Selling_Price	Present_Price	Kms_Driven	Fuel_Type	Seller_Type	Transmissi
0	ritz	2014	3.35	5.59	27000	Petrol	Dealer	Manu
1	sx4	2013	4.75	9.54	43000	Diesel	Dealer	Manu
2	ciaz	2017	7.25	9.85	6900	Petrol	Dealer	Manu
3	wagon r	2011	2.85	4.15	5200	Petrol	Dealer	Manu
4	swift	2014	4.60	6.87	42450	Diesel	Dealer	Manu
...	...	...	...	...	...	...	...	
96	innova	2016	20.75	25.39	29000	Diesel	Dealer	Automa
97	corolla altis	2017	17.00	18.64	8700	Petrol	Dealer	Manu
98	corolla altis	2013	7.05	18.61	45000	Petrol	Dealer	Manu
99	fortuner	2010	9.65	20.45	50024	Diesel	Dealer	Manu
100	Royal Enfield Thunder 500	2016	1.75	1.90	3000	Petrol	Individual	Manu

101 rows × 9 columns

use less column = owner

In [188]:

```
car.drop('Owner',axis=1,inplace=True)
```

after analysis the data owner column there is no requirement so i am going to delete it.

In [189]:

```
car
```

Out[189]:

	Car_Name	Year	Selling_Price	Present_Price	Kms_Driven	Fuel_Type	Seller_Type	Transmissi
0	ritz	2014	3.35	5.59	27000	Petrol	Dealer	Manu
1	sx4	2013	4.75	9.54	43000	Diesel	Dealer	Manu
2	ciaz	2017	7.25	9.85	6900	Petrol	Dealer	Manu
3	wagon r	2011	2.85	4.15	5200	Petrol	Dealer	Manu
4	swift	2014	4.60	6.87	42450	Diesel	Dealer	Manu
...	...	...	...	...	...	...	...	
296	city	2016	9.50	11.60	33988	Diesel	Dealer	Manu
297	brio	2015	4.00	5.90	60000	Petrol	Dealer	Manu
298	city	2009	3.35	11.00	87934	Petrol	Dealer	Manu
299	city	2017	11.50	12.50	9000	Diesel	Dealer	Manu
300	brio	2016	5.30	5.90	5464	Petrol	Dealer	Manu

301 rows × 8 columns

since computer is not working for categorical data i.e i am goinng to canvet the data into numeric with the help of one hot encoding

In [190]:

```
categorical_columns = ['Car_Name', 'Fuel_Type', 'Seller_Type', 'Transmission']
car_encoded = pd.get_dummies(car, columns=categorical_columns,dtype=int)
car_encoded
```

Out[190]:

	Year	Selling_Price	Present_Price	Kms_Driven	Car_Name_800	Car_Name_Activa 3g	Car_Name_A
0	2014	3.35	5.59	27000	0	0	
1	2013	4.75	9.54	43000	0	0	
2	2017	7.25	9.85	6900	0	0	
3	2011	2.85	4.15	5200	0	0	
4	2014	4.60	6.87	42450	0	0	
...	...	...	...	...	...	...	
296	2016	9.50	11.60	33988	0	0	
297	2015	4.00	5.90	60000	0	0	
298	2009	3.35	11.00	87934	0	0	
299	2017	11.50	12.50	9000	0	0	
300	2016	5.30	5.90	5464	0	0	

301 rows × 109 columns

now as i can see here the column name lenght is to long i.e i am goint to short it.

In [191]:

```
column_mapping={
    'Car_Name_800':'800',
    'Car_Name_Activa 3g':'Activa 3g',
    'Car_Name_Activa 4g':'Activa 4g',
    'Car_Name_Bajaj ct 100':'Bajaj ct 100',
    'Car_Name_Bajaj Avenger 150':'Bajaj Avenger 150',
    'Car_Name_Bajaj Avenger 150 street':'Avenger 150 street',
    'Car_Name_vitara brezza':'vitara brezza',
    'Car_Name_wagonr':'wagonr',
    'Car_Name_xcent':'xcent',
    }
car_encoded.rename(columns=column_mapping,inplace=True)
```

In [192]:

```
car_encoded
```

Out[192]:

	Year	Selling_Price	Present_Price	Kms_Driven	800	Activa 3g	Activa 4g	Car_Name_Bajaj ct 100	Bajaj Avenger 150
0	2014	3.35	5.59	27000	0	0	0	0	0
1	2013	4.75	9.54	43000	0	0	0	0	0
2	2017	7.25	9.85	6900	0	0	0	0	0
3	2011	2.85	4.15	5200	0	0	0	0	0
4	2014	4.60	6.87	42450	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...
296	2016	9.50	11.60	33988	0	0	0	0	0
297	2015	4.00	5.90	60000	0	0	0	0	0
298	2009	3.35	11.00	87934	0	0	0	0	0
299	2017	11.50	12.50	9000	0	0	0	0	0
300	2016	5.30	5.90	5464	0	0	0	0	0

301 rows × 109 columns

there is now any name of car 800 thats way i am going to remove it

In [193]:

```
car_encoded.drop('800',axis=1,inplace=True)
```

In [194]:

```
car_encoded
```

Out[194]:

	Year	Selling_Price	Present_Price	Kms_Driven	Activa 3g	Activa 4g	Car_Name_Bajaj ct 100	Bajaj Avenger 150	Ave st
0	2014	3.35	5.59	27000	0	0	0	0	
1	2013	4.75	9.54	43000	0	0	0	0	
2	2017	7.25	9.85	6900	0	0	0	0	
3	2011	2.85	4.15	5200	0	0	0	0	
4	2014	4.60	6.87	42450	0	0	0	0	
...	...	...	...	...	...	...	...	...	
296	2016	9.50	11.60	33988	0	0	0	0	
297	2015	4.00	5.90	60000	0	0	0	0	
298	2009	3.35	11.00	87934	0	0	0	0	
299	2017	11.50	12.50	9000	0	0	0	0	
300	2016	5.30	5.90	5464	0	0	0	0	

301 rows × 108 columns

since car_encoded name id so long that way i am rename it

In [195]:

```
car=car_encoded.copy()
```

In [196]:

```
car
```

Out[196]:

	Year	Selling_Price	Present_Price	Kms_Driven	Activa 3g	Activa 4g	Car_Name_Bajaj ct 100	Bajaj Avenger 150	Ave st
0	2014	3.35	5.59	27000	0	0	0	0	
1	2013	4.75	9.54	43000	0	0	0	0	
2	2017	7.25	9.85	6900	0	0	0	0	
3	2011	2.85	4.15	5200	0	0	0	0	
4	2014	4.60	6.87	42450	0	0	0	0	
...	...	...	...	...	...	...	...	...	
296	2016	9.50	11.60	33988	0	0	0	0	
297	2015	4.00	5.90	60000	0	0	0	0	
298	2009	3.35	11.00	87934	0	0	0	0	
299	2017	11.50	12.50	9000	0	0	0	0	
300	2016	5.30	5.90	5464	0	0	0	0	

301 rows × 108 columns

In [197]:

```
car.shape
```

Out[197]:

```
(301, 108)
```

In [198]:

```
car.columns
```

Out[198]:

```
Index(['Year', 'Selling_Price', 'Present_Price', 'Kms_Driven', 'Activa 3g',
       'Activa 4g', 'Car_Name_Bajaj  ct 100', 'Bajaj Avenger 150',
       'Avenger 150 street', 'Car_Name_Bajaj Avenger 220',
       ...
       'vitara brezza', 'Car_Name_wagon r', 'xcent', 'Fuel_Type_CNG',
       'Fuel_Type_Diesel', 'Fuel_Type_Petrol', 'Seller_Type_Dealer',
       'Seller_Type_Individual', 'Transmission_Automatic',
       'Transmission_Manual'],
      dtype='object', length=108)
```

In [199]:

```
car.describe().transpose()
```

Out[199]:

	count	mean	std	min	25%	50%	75%	
Year	301.0	2013.627907	2.891554	2003.00	2012.0	2014.0	2016.0	2
Selling_Price	301.0	4.661296	5.082812	0.10	0.9	3.6	6.0	
Present_Price	301.0	7.628472	8.644115	0.32	1.2	6.4	9.9	
Kms_Driven	301.0	36947.205980	38886.883882	500.00	15000.0	32000.0	48767.0	500
Activa 3g	301.0	0.006645	0.081378	0.00	0.0	0.0	0.0	
...	...	...	...	...	...	...	...	
Fuel_Type_Petrol	301.0	0.794020	0.405089	0.00	1.0	1.0	1.0	
Seller_Type_Dealer	301.0	0.647841	0.478439	0.00	0.0	1.0	1.0	
Seller_Type_Individual	301.0	0.352159	0.478439	0.00	0.0	0.0	1.0	
Transmission_Automatic	301.0	0.132890	0.340021	0.00	0.0	0.0	0.0	
Transmission_Manual	301.0	0.867110	0.340021	0.00	1.0	1.0	1.0	

108 rows × 8 columns

In [200]:

```
car.dtypes
```

Out[200]:

```
Year                      int64
Selling_Price           float64
Present_Price           float64
Kms_Driven                int64
Activa 3g                 int32
                          ...
Fuel_Type_Petrol          int32
Seller_Type_Dealer        int32
Seller_Type_Individual    int32
Transmission_Automatic    int32
Transmission_Manual       int32
Length: 108, dtype: object
```

missing value

1.Check for Missing Values in Entire DataFrame:

In [201]:

```
car.isnull().sum()
```

Out[201]:

```
Year                     0
Selling_Price            0
Present_Price            0
Kms_Driven               0
Activa 3g                0
                        ..
Fuel_Type_Petrol         0
Seller_Type_Dealer       0
Seller_Type_Individual   0
Transmission_Automatic   0
Transmission_Manual      0
Length: 108, dtype: int64
```

In [202]:

```
mis_any_value=car.isna().any().any()
```

In [203]:

```
mis_any_value
```

Out[203]:

```
False
```

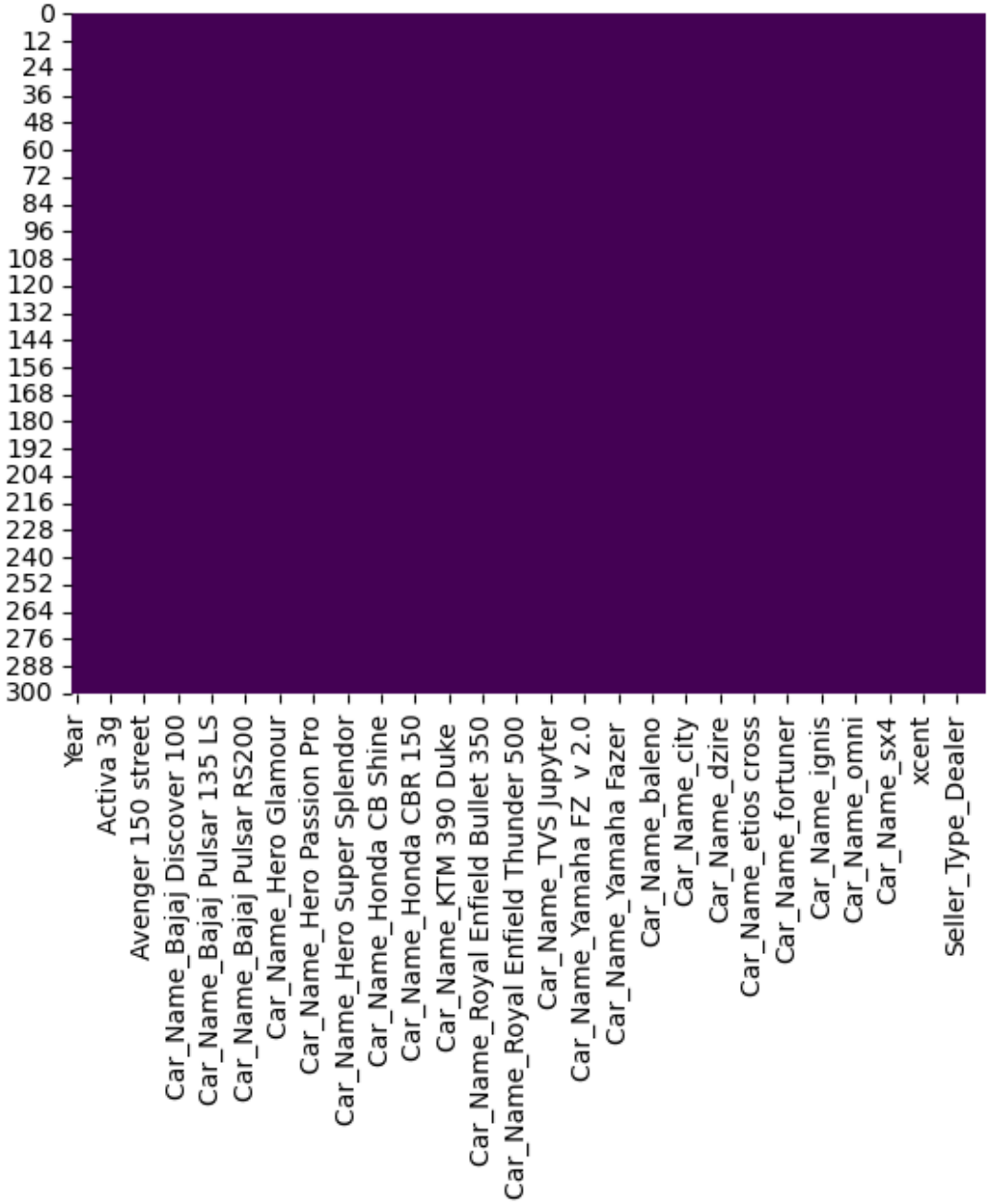
4.Visualize Missing Data:

In [204]:

```
import  matplotlib.pyplot as plt
import seaborn as sns
```

In [205]:

```
sns.heatmap(car.isna(),cbar=False,cmap='viridis')
plt.show()
```



In [206]:

```
# count the category element
d_types=dict(car.dtypes)
for name,type_ in d_types.items():
    if str(type_)=="int32":
        print(f"<===========(name)=========>")
        print(car[name].value_counts())
        print()
```

```
1      4
Name: count, dtype: int64

<===========(name)=========>
Car_Name_s cross
0    300
1      1
Name: count, dtype: int64

<===========(name)=========>
Car_Name_swift
0    296
1      5
Name: count, dtype: int64

<===========(name)=========>
Car_Name_sx4
0    295
1      6
Name: count, dtype: int64
```

after label encoding

In [207]:

```
car.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 301 entries, 0 to 300
Columns: 108 entries, Year to Transmission_Manual
dtypes: float64(2), int32(104), int64(2)
memory usage: 131.8 KB
```

In [208]:

```
car
```

Out[208]:

	Year	Selling_Price	Present_Price	Kms_Driven	Activa 3g	Activa 4g	Car_Name_Bajaj ct 100	Bajaj Avenger 150	Ave st
0	2014	3.35	5.59	27000	0	0	0	0	
1	2013	4.75	9.54	43000	0	0	0	0	
2	2017	7.25	9.85	6900	0	0	0	0	
3	2011	2.85	4.15	5200	0	0	0	0	
4	2014	4.60	6.87	42450	0	0	0	0	
...	...	...	...	...	...	...	...	...	
296	2016	9.50	11.60	33988	0	0	0	0	
297	2015	4.00	5.90	60000	0	0	0	0	
298	2009	3.35	11.00	87934	0	0	0	0	
299	2017	11.50	12.50	9000	0	0	0	0	
300	2016	5.30	5.90	5464	0	0	0	0	

301 rows × 108 columns

Task 2:- Feature selection and Engineering

Feature selection and feature engineering are essential steps in the process of preparing data for machine learning models. They both aim to improve the performance of machine learning models by selecting relevant features and creating new features from the existing data.

feature selection:

Feature selection is the process of choosing a subset of the most relevant features (variables or columns) from the original set of features in your dataset. The goal is to remove irrelevant, redundant, or noisy features that do not contribute significantly to the predictive power of the model. Feature selection has several advantages:

Simplifies the Model: Fewer features mean simpler models, which are easier to understand and interpret.

Reduces Overfitting: Removing irrelevant or redundant features can help prevent overfitting, where the model performs well on the training data but poorly on unseen data.

Improves Training Speed: Smaller feature sets reduce the computational cost and training time of machine learning models

comman techniques selection include:

Filter Methods: These methods use statistical measures to rank and select features based on their individual relevance to the target variable. Examples include correlation-based feature selection and chi-squared tests.

Wrapper Methods: These methods evaluate feature subsets by training and testing the model on different combinations of features. Examples include forward selection and backward elimination.

Embedded Methods: These methods incorporate feature selection as part of the model training process. For

feature Engineering

Feature engineering involves creating new features or transforming existing features to make them more informative for the machine learning model. It aims to capture hidden patterns or relationships in the data that the model might not learn from the raw features alone. Feature engineering is a creative and domain-specific process and can significantly impact the model's performance.

Common techniques and examples of feature engineering include:

Polynomial Features: Creating new features by raising existing features to a power. For instance, squaring a feature to capture quadratic relationships.

Binning and Bucketing: Grouping continuous numerical features into bins or categories. For example, converting age into age groups (e.g., "child," "adult," "senior").

One-Hot Encoding: Converting categorical variables into binary (0/1) indicator variables, making them suitable for models that require numerical input.

Feature Scaling: Scaling numerical features to have a similar range, often using techniques like Min-Max scaling or Z-score normalization.

Text Processing: For natural language processing (NLP) tasks, converting text data into numerical representations, such as TF-IDF or word embeddings.

Date and Time Features: Extracting information from date and time variables, such as day of the week, month, or year, which can be useful for time-series analysis.

Domain-Specific Transformations: Applying transformations specific to the problem domain. For example, calculating ratios, averages, or percentages based on domain knowledge.

one hot Encoding:

here in this data cars are using differnt types of fuel so we can use one hot encoding

In [164]:

```
car.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 301 entries, 0 to 300
Columns: 108 entries, Year to Transmission_Manual
dtypes: float64(2), int32(104), int64(2)
memory usage: 131.8 KB
```

In [240]:

```
columns_to_exclude = [
    'Selling_Price',
    'Activa 3g',
    'Activa 4g',
    'Car_Name_Bajaj ct 100',
    'Bajaj Avenger 150',
    'Avenger 150 street',
    'Car_Name_Bajaj Avenger 220',
    'Car_Name_wagon r',
    'xcent'
]

# Select columns that are NOT in the columns_to_exclude list
x = car.loc[:, ~car.columns.isin(columns_to_exclude)]
```

In [241]:

```
x
```

Out[241]:

	Year	Present_Price	Kms_Driven	Car_Name_Bajaj ct 100	Car_Name_Bajaj Avenger 220 dtsi	Car_Name_Bajaj Avenger Street 220	Car_Nam Disco
0	2014	5.59	27000	0	0	0	
1	2013	9.54	43000	0	0	0	
2	2017	9.85	6900	0	0	0	
3	2011	4.15	5200	0	0	0	
4	2014	6.87	42450	0	0	0	
...	...	...	...	...	...	...	
296	2016	11.60	33988	0	0	0	
297	2015	5.90	60000	0	0	0	
298	2009	11.00	87934	0	0	0	
299	2017	12.50	9000	0	0	0	
300	2016	5.90	5464	0	0	0	

301 rows × 100 columns

In [237]:

```
car.columns
```

Out[237]:

```
Index(['Year', 'Selling_Price', 'Present_Price', 'Kms_Driven', 'Activa 3g',
       'Activa 4g', 'Car_Name_Bajaj  ct 100', 'Bajaj Avenger 150',
       'Avenger 150 street', 'Car_Name_Bajaj Avenger 220',
       ...
       'vitara brezza', 'Car_Name_wagon r', 'xcent', 'Fuel_Type_CNG',
       'Fuel_Type_Diesel', 'Fuel_Type_Petrol', 'Seller_Type_Dealer',
       'Seller_Type_Individual', 'Transmission_Automatic',
       'Transmission_Manual'],
      dtype='object', length=108)
```

In [242]:

```
x.head()
```

Out[242]:

	Year	Present_Price	Kms_Driven	Car_Name_Bajaj ct 100	Car_Name_Bajaj Avenger 220 dtsi	Car_Name_Bajaj Avenger Street 220	Car_Name_ Discove
0	2014	5.59	27000	0	0	0	
1	2013	9.54	43000	0	0	0	
2	2017	9.85	6900	0	0	0	
3	2011	4.15	5200	0	0	0	
4	2014	6.87	42450	0	0	0	

5 rows × 100 columns

now i am goint to anlysis my data for Activa 3g

In [260]:

```
x=car.iloc[:,1:5]
```

In [261]:

```
x
```

Out[261]:

	Selling_Price	Present_Price	Kms_Driven	Activa 3g
0	3.35	5.59	27000	0
1	4.75	9.54	43000	0
2	7.25	9.85	6900	0
3	2.85	4.15	5200	0
4	4.60	6.87	42450	0
...	...	...	...	...
296	9.50	11.60	33988	0
297	4.00	5.90	60000	0
298	3.35	11.00	87934	0
299	11.50	12.50	9000	0
300	5.30	5.90	5464	0

301 rows × 4 columns

In [262]:

```
y=car.iloc[:,1:2]
```

In [263]:

```
y.head()
```

Out[263]:

	Selling_Price
0	3.35
1	4.75
2	7.25
3	2.85
4	4.60

In [215]:

```
x.shape,y.shape
```

Out[215]:

```
((301, 107), (301, 1))
```

Model selction

In [264]:

```
from sklearn.model_selection import train_test_split
```

In [265]:

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=4)
```

In [266]:

```
x_train.head(10)
```

Out[266]:

	Selling_Price	Present_Price	Kms_Driven	Activa 3g
293	3.25	9.900	38000	0
93	23.00	30.610	40000	0
47	1.05	4.150	65000	0
175	0.38	0.787	75000	0
84	3.49	13.460	197176	0
16	7.25	10.790	41678	0
83	12.50	13.460	38000	0
292	6.40	8.400	12000	0
42	1.95	7.150	58000	0
106	1.35	3.450	16500	0

task 2:-StandardScaler and feature scaling

StandardScaler is a preprocessing technique in scikit-learn that standardizes the features by removing the mean and scaling to unit variance. This is important for many machine learning algorithms as it helps ensure that all features have the same scale, which can improve the model's convergence and performance.

In [267]:

```
from sklearn.preprocessing import StandardScaler
```

In [268]:

```
scaler=StandardScaler()
x_train=scaler.fit_transform(x_train)
x_test=scaler.transform(x_test)
```

In [269]:

```
x_train.shape,x_test.shape
```

Out[269]:

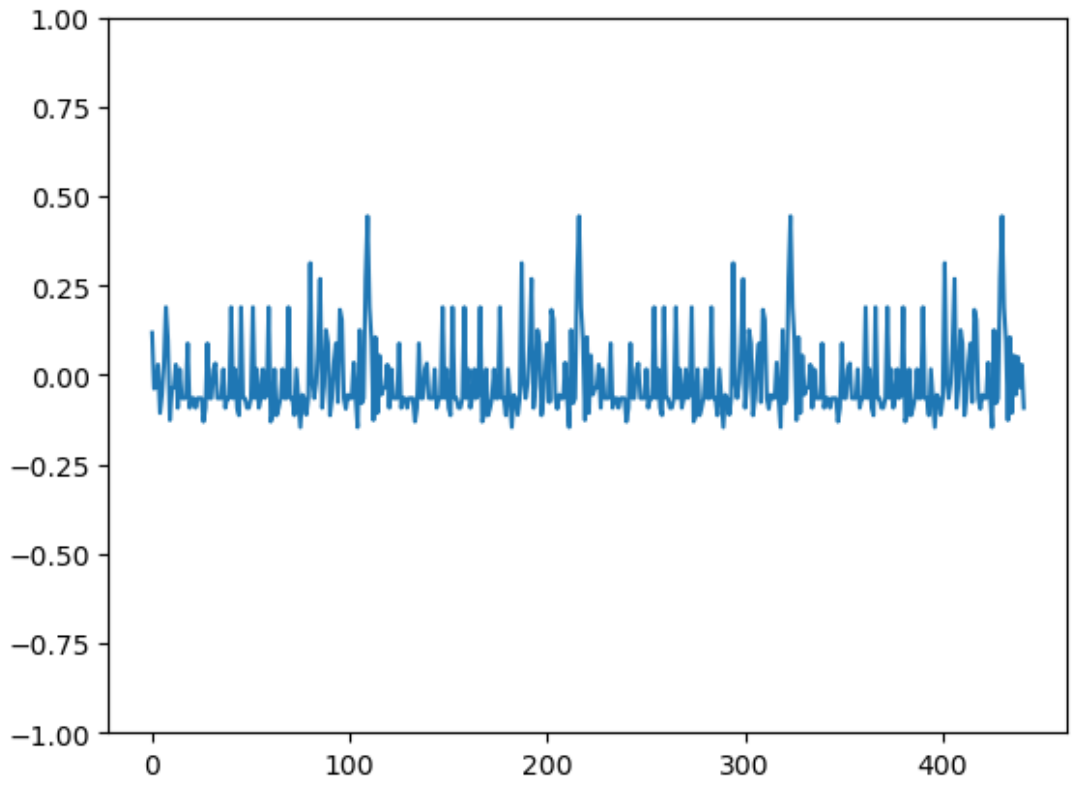
```
((240, 4), (61, 4))
```

In [270]:

```
plt.ylim(-1,1)
for i in range(x_test.shape[1]):
    means.append(np.mean(x_test[:, i]))
plt.plot(means,scaley=False)
```

Out[270]:

```
[<matplotlib.lines.Line2D at 0x1d4976a4f10>]
```



explanation

plt.ylim(-1,1): This line sets the y-axis (vertical axis) limits of the plot. It specifies that the y-axis should range from -1 to 1. This is often used to adjust the range of the y-axis to focus on a specific interval.

for i in range(x_test.shape[1]):: This is a loop that iterates through the columns of the x_test dataset. x_test.shape[1] returns the number of columns in x_test, and the loop runs from 0 to one less than this number.

means.append(np.mean(x_test[:, i])): Inside the loop, this line calculates the mean (average) value of the data in each column of x_test. x_test[:, i] extracts the values of the i-th column, and np.mean() computes the mean of those values. The mean value of each column is then appended to the means list.

plt.plot(means, scaley=False): After the loop, this line plots the means against their corresponding column indices (0, 1, 2, etc.). The plt.plot() function is used to create the line plot. means contains the y-values (means of the columns), and the x-values are implicitly generated as the indices of the means list.

scaley=False is an argument that tells Matplotlib not to automatically scale the y-axis to fit the data. Since you've already set the y-axis limits with plt.ylim(-1, 1), this prevents Matplotlib from changing the y-axis scaling. In summary, this code calculates the means of the columns in x_test and then creates a line plot showing how the means vary across the columns. The y-axis is limited to the range -1 to 1 for better visualization, and the scaley=False argument ensures that the y-axis limits are not automatically adjusted by Matplotlib.

In [271]:

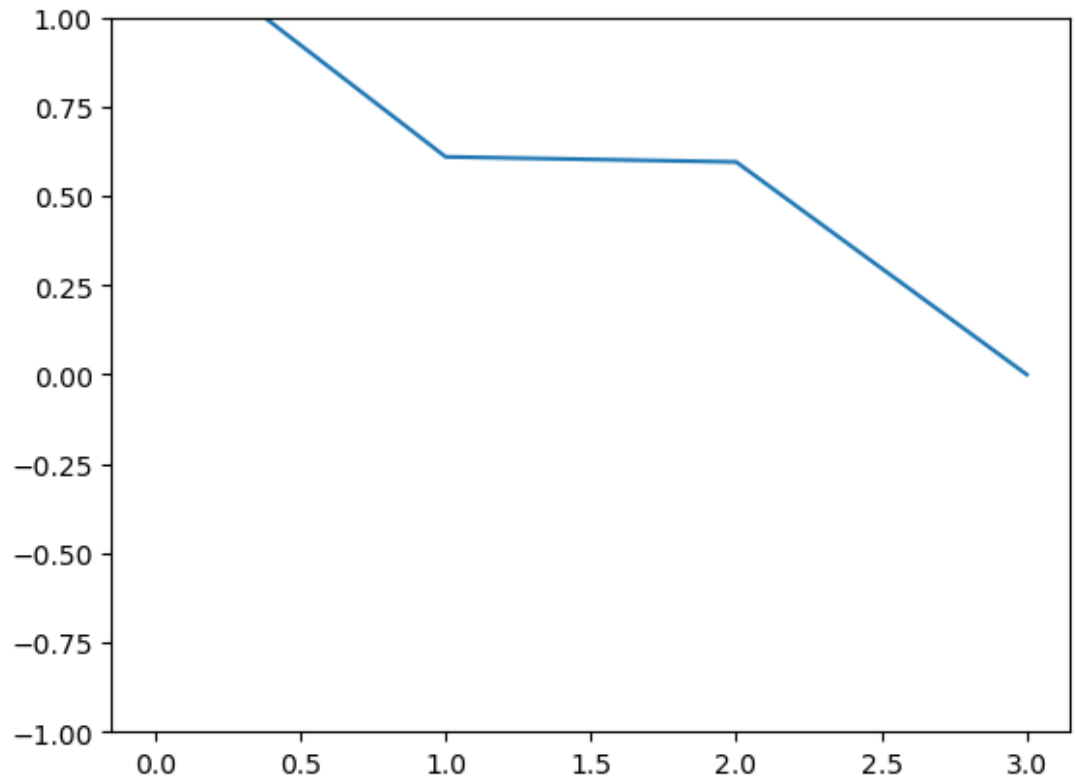
```
# chack variance
#STD

plt.ylim(-1,1)
vars=[]

for i in range(x.shape[1]):
    vars.append(np.var(x_test[:,i]))
plt.plot(vars)
```

Out[271]:

```
[<matplotlib.lines.Line2D at 0x1d497a572e0>]
```



plt.ylim(-1, 1): This line sets the y-axis (vertical axis) limits of the plot. It specifies that the y-axis should range from -1 to 1. This is often used to adjust the range of the y-axis to focus on a specific interval.

vars = []: This line initializes an empty list named vars to store the variances of the columns.

for i in range(x.shape[1]):: This is a loop that iterates through the columns of the x_test dataset. x.shape[1] returns the number of columns in x_test, and the loop runs from 0 to one less than this number.

vars.append(np.var(x_test[:, i])): Inside the loop, this line calculates the variance of the data in each column of x_test. x_test[:, i] extracts the values of the i-th column, and np.var() computes the variance of those values. The variance of each column is then appended to the vars list.

plt.plot(vars): After the loop, this line plots the variances against their corresponding column indices (0, 1, 2, etc.). The plt.plot() function is used to create the line plot. vars contains the y-values (variances of the columns), and the x-values are implicitly generated as the indices of the vars list.

In summary, this code calculates the variances of the columns in x_test and then creates a line plot showing how the variances vary across the columns. The y-axis is limited to the range -1 to 1 for better visualization, and the variances provide information about the spread or dispersion of data in each column.

Task3:- Different types of Algorithms for continuos data

Supervised learning is a type of machine learning where the algorithm learns from labeled data, making predictions or classifications based on input features. There are various algorithms used in supervised learning, each with its own strengths and weaknesses. Here are some common types of supervised learning algorithms:

(i) linear regression

Linear regression is a statistical method used in machine learning and statistics to model the relationship between a dependent variable and one or more independent variables by fitting a linear equation to the observed data. The goal is to find the best-fitting line (or hyperplane in the case of multiple independent variables) that minimizes the sum of the squared differences between the observed and predicted values.

In [272]:

```
from sklearn.linear_model import LinearRegression
model=LinearRegression()
model.fit(x_train,y_train)
```

Out[272]:

```
LinearRegression()
```

In [273]:

```
# predict the  test set results
y_pred=model.predict(x_test)
y_pred
```

Out[273]:

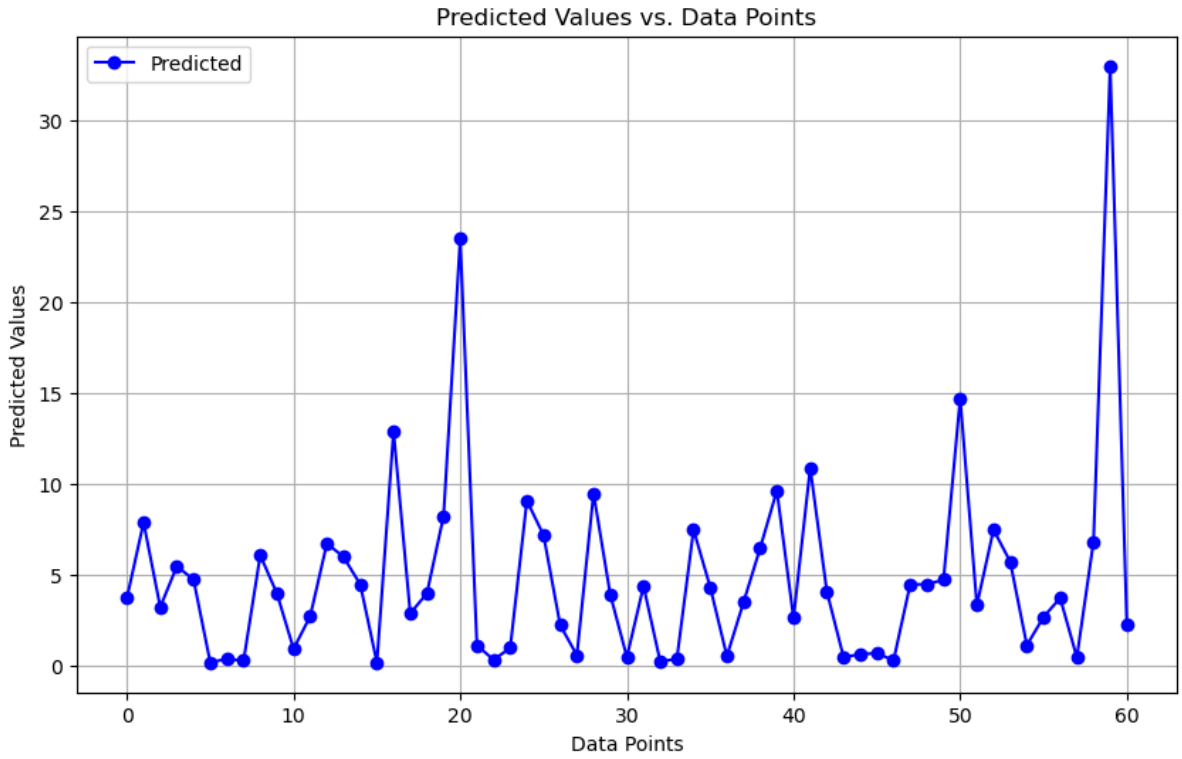
```
array([[ 3.75],
       [ 7.9 ],
       [ 3.25],
       [ 5.5 ],
       [ 4.75],
       [ 0.2 ],
       [ 0.4 ],
       [ 0.3 ],
       [ 6.1 ],
       [ 4.  ],
       [ 0.95],
       [ 2.75],
       [ 6.75],
       [ 6.  ],
       [ 4.5 ],
       [ 0.2 ],
       [12.9 ],
       [ 2.9 ],
       [ 4.  ],
       [ 8.25],
       [23.5 ],
       [ 1.15],
       [ 0.35],
       [ 1.  ],
       [ 9.1 ],
       [ 7.2 ],
       [ 2.25],
       [ 0.6 ],
       [ 9.5 ],
       [ 3.95],
       [ 0.5 ],
       [ 4.4 ],
       [ 0.25],
       [ 0.4 ],
       [ 7.5 ],
       [ 4.35],
       [ 0.55],
       [ 3.51],
       [ 6.5 ],
       [ 9.65],
       [ 2.65],
       [10.9 ],
       [ 4.1 ],
       [ 0.5 ],
       [ 0.65],
       [ 0.72],
       [ 0.31],
       [ 4.5 ],
       [ 4.5 ],
       [ 4.75],
       [14.73],
       [ 3.35],
       [ 7.5 ],
       [ 5.75],
       [ 1.15],
       [ 2.7 ],
       [ 3.75],
       [ 0.48],
       [ 6.85],
       [33.  ],
       [ 2.25]])
```

In [276]:

```
# Assuming you have already predicted y_pred using your model

# Create an array of indices for the data points
indices = range(len(y_pred))

# Create a line plot to visualize y_pred
plt.figure(figsize=(10, 6))  # Adjust the figure size as needed
plt.plot(indices, y_pred, marker='o', linestyle='-', color='b', label='Predicted')
plt.xlabel('Data Points')
plt.ylabel('Predicted Values')
plt.title('Predicted Values vs. Data Points')
plt.legend()
plt.grid(True)
plt.show()
```



(ii) Logistic Regression

Logistic regression is used for binary classification tasks. It models the probability that an input belongs to a particular class and is particularly useful for problems like spam detection or medical diagnosis.

(iii) Decision trees

Decision trees are versatile and can be used for both classification and regression tasks. They recursively split the data into subsets based on feature values, making decisions at each node.

In [279]:

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
```

(iv) random forest

Random Forest is an ensemble method that combines multiple decision trees to improve predictive accuracy and reduce overfitting. It's effective for both classification and regression.

(v) Support vector Machines(svm)

SVM is used for binary classification. It finds a hyperplane that best separates data into different classes while maximizing the margin between them.

(vi) K-Nearest Neighbours(K-NN)

K-NN is a simple algorithm that classifies data points based on the majority class among their k-nearest neighbors in feature space. It's used for both classification and regression.

(vii) Navie Bayes:

Naive Bayes is a probabilistic classifier based on Bayes' theorem. It's particularly useful for text classification tasks like spam filtering and sentiment analysis.

(viii) Neural Networks

Deep learning models, such as artificial neural networks, including feedforward, convolutional, and recurrent neural networks (RNNs), have become popular for various supervised learning tasks, including image classification, natural language processing, and speech recognition.

(ix) Gradient Boosting

Gradient Boosting algorithms like Gradient Boosted Trees and XGBoost create an ensemble of weak learners (usually decision trees) to form a strong predictive model. They are highly effective and often used in various competitions and real-world applications.

(x) Linear Discriminant analysis (LDA)

LDA is used for dimensionality reduction and classification. It finds a linear combination of features that best separates classes in the data.

(xi) Ensemble Methods

Besides Random Forest and Gradient Boosting, other ensemble methods like AdaBoost and Bagging (Bootstrap Aggregating) can be used to combine multiple base models to improve overall performance.

(xii) Deep Learning Architectures

For complex tasks like image recognition, speech recognition, and natural language understanding, deep learning architectures like Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) are commonly employed.

(xiii) Time series Algorithums

For time-series prediction tasks, algorithms like ARIMA (AutoRegressive Integrated Moving Average) and LSTM (Long Short-Term Memory) networks are used.

(xiv) Nearest centroid classifier

This algorithm classifies data points by finding the centroid of each class and assigning new data points to the class with the nearest centroid.

(xv) Gaussion Processes

Gaussian Processes are a probabilistic approach used for regression tasks, particularly when dealing with uncertainty estimation.

Task 4 :- Traning ,Evaluate & Selection

(i) Linear Regrestion evaluation

In [274]:

```
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
```

In [275]:

```
# Evaluate the model
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

print(f"Mean Absolute Error (MAE): {mae}")
print(f"Mean Squared Error (MSE): {mse}")
print(f"Root Mean Squared Error (RMSE): {rmse}")
print(f"R-squared (R²) Score: {r2}")
```

```
Mean Absolute Error (MAE): 9.955606466720666e-16
Mean Squared Error (MSE): 2.823249120837084e-30
Root Mean Squared Error (RMSE): 1.6802526955303728e-15
R-squared (R²) Score: 1.0
```

In [ ]:

```

```