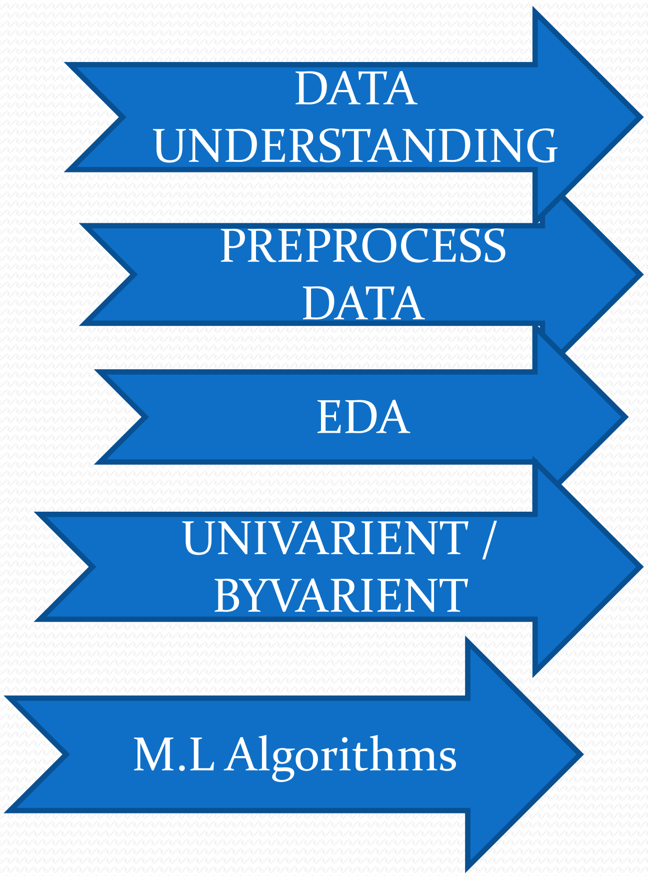
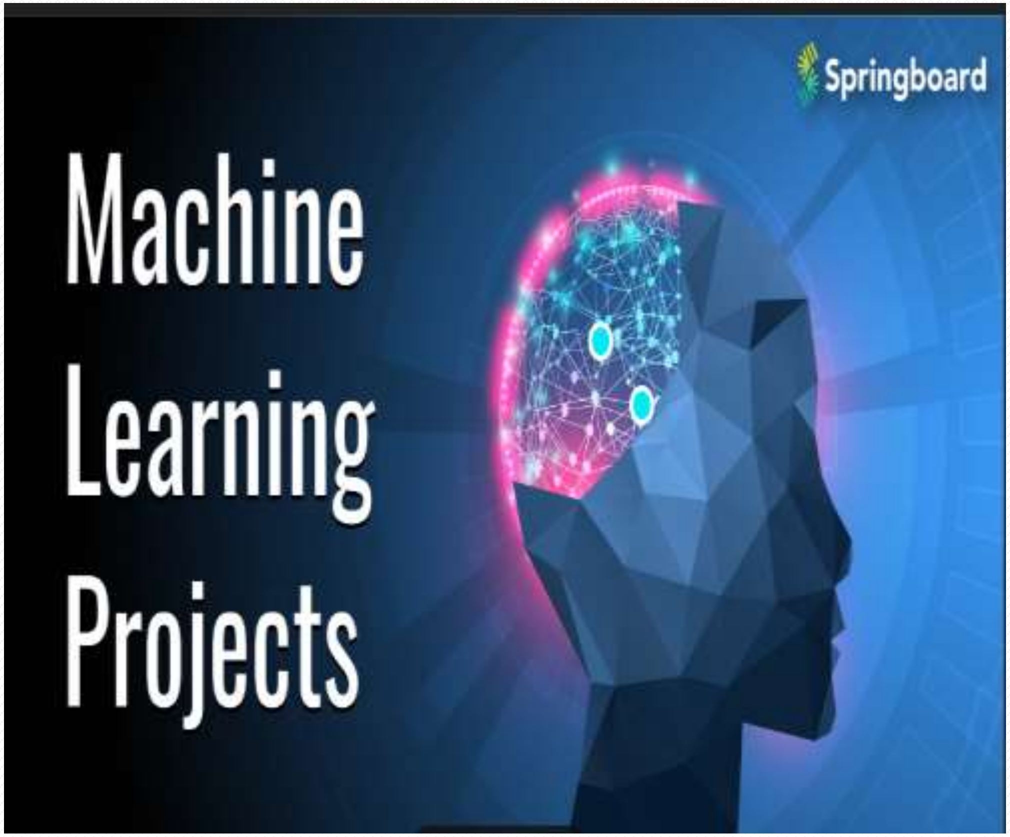
20-09-2024

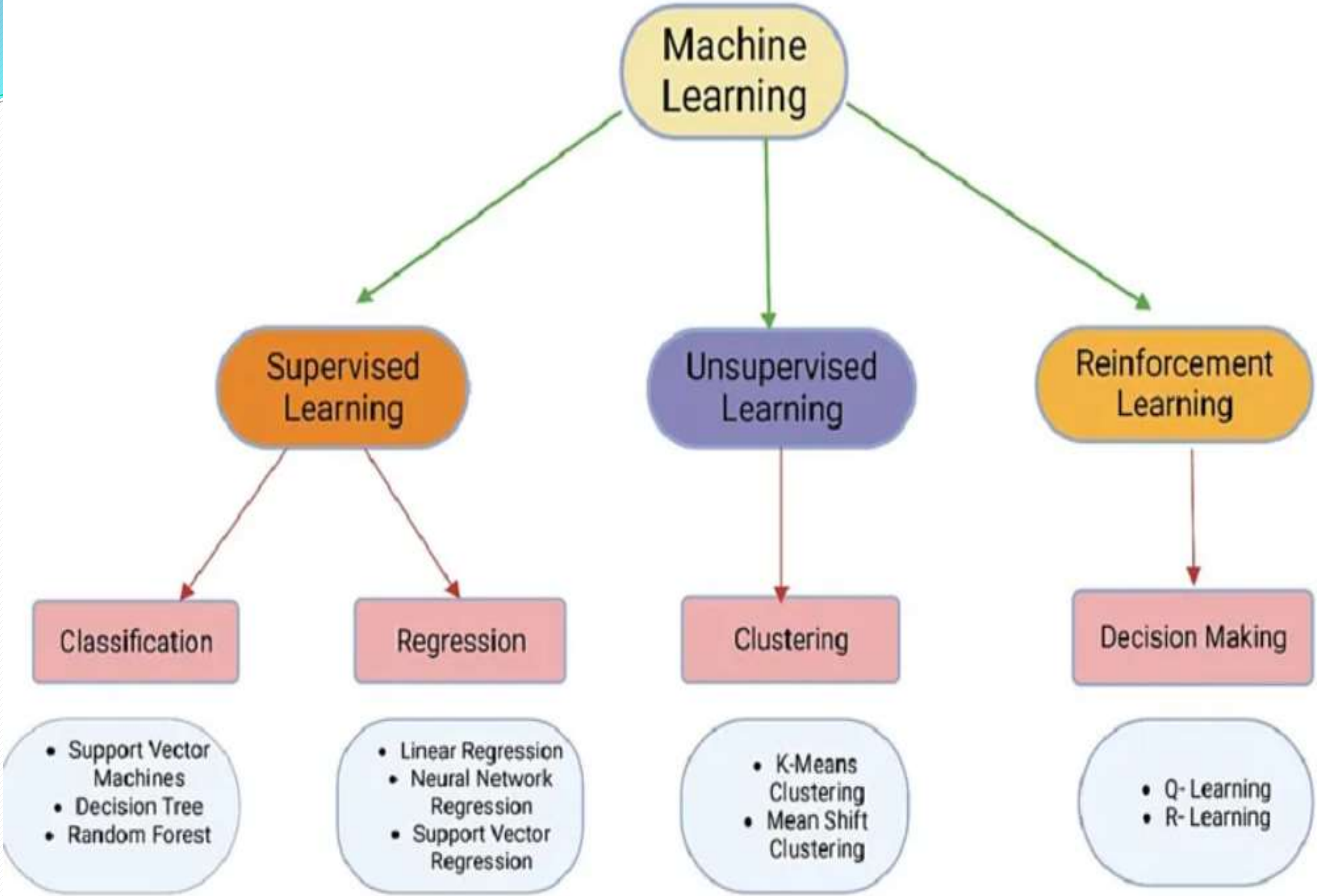
Vijayabharathi M

Data Analytics and Science,Milestone-(2)

<u>HEART DATASET</u>






Machine Learning
Supervised Learning
Unsupervised Learning
Reinforcement Learning
Classification
Regression
Clustering
Decision Making
• Support Vector Machines
• Decision Tree
• Random Forest
• Linear Regression
• Neural Network Regression
• Support Vector Regression
• K-Means Clustering
• Mean Shift Clustering
• Q- Learning
• R- Learning

MACHINE LEARNING

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import mean_squared_error,r2_score,confusion_matrix,classification_report,accuracy_score
from sklearn.preprocessing import StandardScaler
```

Exploring the HEART Dataset with different (CLASSIFICATION ALGORITHMS)

We have a data which classified if patients have heart disease or not according to features in it. We will try to use this data to create a model which tries predict if a patient has this disease or not. We will use classification algorithms.

```
df = pd.read_csv("C:/kgisl class/MILESTONE - 2/heart.csv")
df
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	52	1	0	125	212	0	1	168	0	1.0	2	2	3	0
1	53	1	0	140	203	1	0	155	1	3.1	0	0	3	0
2	70	1	0	145	174	0	1	125	1	2.6	0	0	3	0
3	61	1	0	148	203	0	1	161	0	0.0	2	1	3	0
4	62	0	0	138	294	1	1	106	0	1.9	1	3	2	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1020	59	1	1	140	221	0	1	164	1	0.0	2	0	2	1
1021	60	1	0	125	258	0	0	141	1	2.8	1	1	3	0
1022	47	1	0	110	275	0	0	118	1	1.0	1	1	2	0
1023	50	0	0	110	254	0	0	159	0	0.0	2	0	2	1

Exploring the dataset

```
df.shape
```

```
(1025, 14)
```

```
df.dtypes
```

```
age         int64
sex         int64
cp          int64
trestbps    int64
chol        int64
fbs         int64
```

```
df.head()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	52	1	0	125	212	0	1	168	0	1.0	2	2	3	0
1	53	1	0	140	203	1	0	155	1	3.1	0	0	3	0
2	70	1	0	145	174	0	1	125	1	2.6	0	0	3	0
3	61	1	0	148	203	0	1	161	0	0.0	2	1	3	0
4	62	0	0	138	294	1	1	106	0	1.9	1	3	2	0

```
df.tail()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
1020	59	1	1	140	221	0	1	164	1	0.0	2	0	2	1

```
[7]: df.info()
```

```
<class 'pandas.core.frame.Da
RangeIndex: 1025 entries, 0
Data columns (total 14 colum
 #   Column    Non-Null Coun
---  ------    -------------
 0   age       1025 non-null
 1   sex       1025 non-null
 2   cp        1025 non-null
 3   trestbps  1025 non-null
 4   chol      1025 non-null
 5   fbs       1025 non-null
 6   restecg   1025 non-null
```

```
df.describe()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach
count	1025.000000	1025.000000	1025.000000	1025.000000	1025.00000	1025.000000	1025.000000	1025.00000
mean	54.434146	0.695610	0.942439	131.611707	246.00000	0.149268	0.529756	149.11414
std	9.072290	0.460373	1.029641	17.516718	51.59251	0.356527	0.527878	23.00572
min	29.000000	0.000000	0.000000	94.000000	126.00000	0.000000	0.000000	71.00000
25%	48.000000	0.000000	0.000000	120.000000	211.00000	0.000000	0.000000	132.00000
50%	56.000000	1.000000	1.000000	130.000000	240.00000	0.000000	1.000000	152.00000
75%	61.000000	1.000000	2.000000	140.000000	275.00000	0.000000	1.000000	166.00000
max	77.000000	1.000000	3.000000	200.000000	564.00000	1.000000	2.000000	202.00000

```
df.columns
```

```
Index(['age', 'sex', 'c
       'exang', 'oldpea
      dtype='object')
```

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

```
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
ca          0
thal        0
target      0
dtype: int64
```

```
df['target'].unique()                                # this is Nu
```

```
array([0, 1], dtype=int64)
```

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

```
array([1. , 3.1, 2.6, 0. , 1.9, 4.4, 0.8, 3.2, 1.6, 3. , 0.7, 4.2, 1
       2.2, 1.1, 0.3, 0.4, 0.6, 3.4, 2.8, 1.2, 2.9, 3.6, 1.4, 0.2, 2
       5.6, 0.9, 1.8, 6.2, 4. , 2.5, 0.5, 0.1, 2.1, 2.4, 3.8, 2.3, 1
       3.5])
```

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

```
array([52, 53, 70, 61, 62, 58, 55, 46, 54, 71, 43, 34, 51, 50, 60, 6
       63, 42, 44, 56, 57, 59, 64, 65, 41, 66, 38, 49, 48, 29, 37, 4
       76, 40, 39, 77, 69, 35, 74], dtype=int64)
```

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

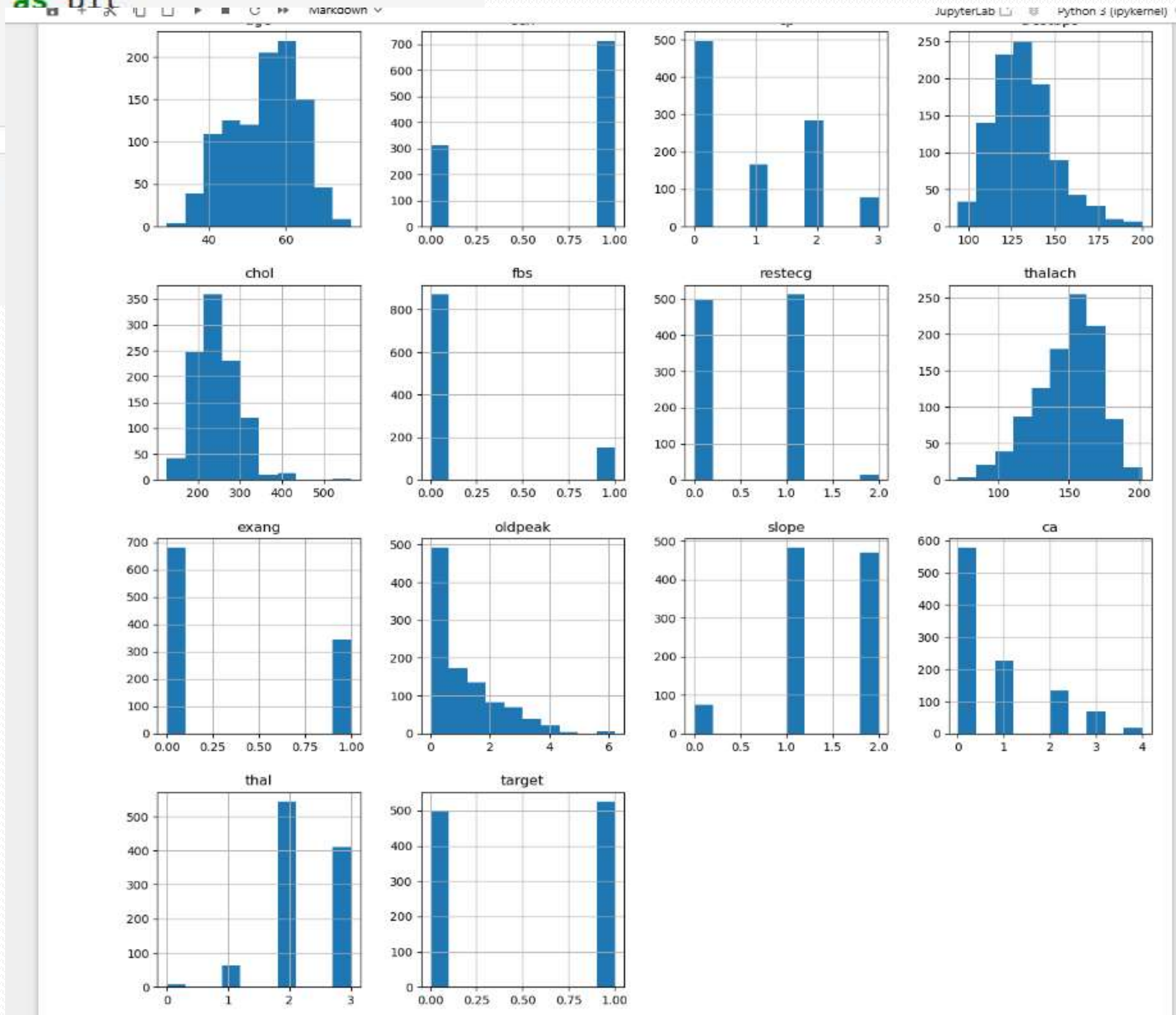
```
array([1, 0], dtype=int64)
```

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

Data Visualization ¶

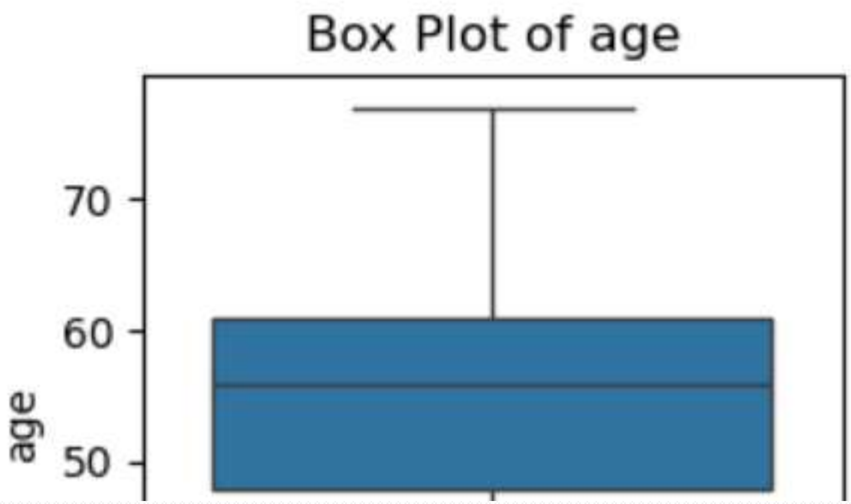
```
# Importing essential libraries
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

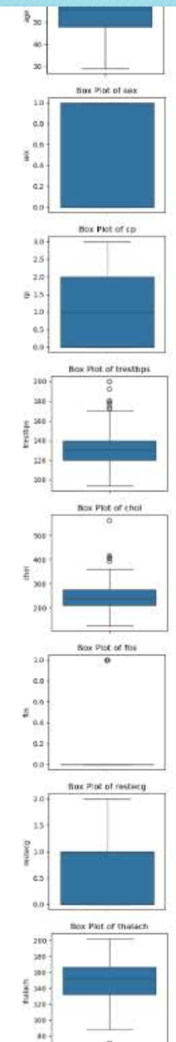
```
fig = plt.figure(figsize
ax = fig.gca()
g = df.hist(ax=ax)
```

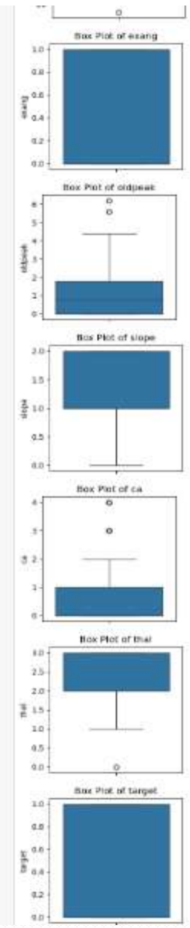


outlayer detection

```python
import seaborn as sns
#outlier detection
for col in df:
    plt.figure(figsize=(3, 3))
    sns.boxplot(y=df[col])
    plt.title(f'Box Plot of {col}')
    plt.show()
```







[31]:
```python
q1=r.quantile(0.25)
q1
```

[31]:
```
age          48.0
sex           0.0
cp            0.0
trestbps    120.0
chol        211.0
fbs           0.0
restecg       0.0
thalach     132.0
exang         0.0
oldpeak       0.0
slope         1.0
ca            0.0
thal          2.0
target        0.0
Name: 0.25, dtype: float64
```

[32]:
```python
q3=r.quantile(0.75)
q3
```

[32]:
```
age          61.0
sex           1.0
cp            2.0
trestbps    140.0
chol        275.0
fbs           0.0
restecg       1.0
thalach     166.0
exang         1.0
oldpeak       1.8
slope         2.0
ca            1.0
thal          3.0
```

```python
IQR=q3-q1
IQR
```

```
age         13.0
sex          1.0
cp           2.0
trestbps    20.0
chol        64.0
fbs          0.0
restecg      1.0
thalach     34.0
exang        1.0
oldpeak      1.8
slope        1.0
ca           1.0
thal         1.0
target       1.0
dtype: float64
```

```python
a=((r<q1-1.5*IQR)|(r>q1+1.5*IQR))
a
```

]:
```python
df1=df[~((r<q1-1.5*IQR)|(r>q1+1.5*IQR)).any(axis=1)]
df1
```

]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
3	61	1	0	148	203	0	1	161	0	0.0	2	1	3	0
5	58	0	0	100	248	0	0	122	0	1.0	1	0	2	1
8	46	1	0	120	249	0	0	144	0	0.8	2	0	3	0
17	54	1	0	124	266	0	0	109	1	2.2	1	1	3	0
18	50	0	1	120	244	0	1	162	0	1.1	2	0	2	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1019	47	1	0	112	204	0	1	143	0	0.1	2	0	2	1
1020	59	1	1	140	221	0	1	164	1	0.0	2	0	2	1
1022	47	1	0	110	275	0	0	118	1	1.0	1	1	2	0
1023	50	0	0	110	254	0	0	159	0	0.0	2	0	2	1
1024	54	1	0	120	188	0	1	113	0	1.4	1	1	3	0

univarient analysis

[37]:
```
df1.groupby(['target']).count()
```

[37]:

target	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal
0	173	173	173	173	173	173	173	173	173	173	173	173	173
1	301	301	301	301	301	301	301	301	301	301	301	301	301

[38]:
```
a=df1.groupby(['target']).size().reset_index(name='count')
a
```

[38]:

	target	count
0	0	173
1	1	301

```
x=filter['target']
plt.hist(x,bins=2,color="skyblue")
plt.title("Histogram --- clarity  of target")
plt.xlabel("target")                                          #  Histogram
plt.ylabel("count")
plt.show()
```

Bivariate Analysis ¶

```
p=sns.pairplot(df1)
```

```
plt.figure(figsize=(10,10))
sns.heatmap(df1.corr(),annot=True,cmap='coolwarm')
plt.title('Correlation Matrix Heatmap')
plt.show()
```

```python
x=df.iloc[:,:-1]
x
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal
0	52	1	0	125	212	0	1	168	0	1.0	2	2	3
1	53	1	0	140	203	1	0	155	1	3.1	0	0	3
2	70	1	0	145	174	0	1	125	1	2.6	0	0	3
3	61	1	0	148	203	0	1	161	0	0.0	2	1	3
4	62	0	0	138	294	1	1	106	0	1.9	1	3	2
...	...	...	...	...	...	...	...	...	...	...	...	...	...
1020	59	1	1	140	221	0	1	164	1	0.0	2	0	2
1021	60	1	0	125	258	0	0	141	1	2.8	1	1	3
1022	47	1	0	110	275	0	0	118	1	1.0	1	1	2
1023	50	0	0	110	254	0	0	159	0	0.0	2	0	2
1024	54	1	0	120	188	0	1	113	0	1.4	1	1	3

1025 rows × 13 columns

```python
y=df.iloc[:,-1]
y
```

```
0       0
1       0
2       0
3       0
4       0
       ..
1020    1
1021    0
1022    0
1023    1
1024    0
25, dtype: ir
```

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

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

```python
x_train.shape
```

```
(820, 13)
```

LogisticRegression

```
from sklearn.model_selection import train_test_split

# Define your feature columns and target column
x = df.drop('target', axis=1)
y = df['target']

# Split the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report

# Initialize and train the model
logistic_model = LogisticRegression()
logistic_model.fit(x_train, y_train)

# Make predictions
y_pred_logistic = logistic_model.predict(x_test)

# Evaluate the model
accuracy_logistic = accuracy_score(y_test, y_pred_logistic)
report_logistic = classification_report(y_test, y_pred_logistic)

print("Logistic Regression Accuracy:", accuracy_logistic)
print("Logistic Regression Classification Report:\n", report_logistic)
```

```
Logistic Regression Accuracy: 0.7853658536585366
Logistic Regression Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.70      0.76       102
           1       0.74      0.87      0.80       103

    accuracy                           0.79       205
   macro avg       0.79      0.78      0.78       205
weighted avg       0.79      0.79      0.78       205
```

WITH HP

```python
from sklearn.model_selection import GridSearchCV

# Define parameter grid
param_grid_logistic = {
    'C': [0.01, 0.1, 1, 10, 100],
    'penalty': ['l1', 'l2']
}

# Initialize and train the model with GridSearch
grid_search_logistic = GridSearchCV(LogisticRegression(max_iter=1000), param_grid_logistic, cv=5, n_jobs=-1)
grid_search_logistic.fit(x_train, y_train)

# Best parameters and best score
print("Best Parameters for Logistic Regression:", grid_search_logistic.best_params_)
print("Best Score for Logistic Regression:", grid_search_logistic.best_score_)

# Make predictions
y_pred_logistic_tuned = grid_search_logistic.predict(x_test)

# Evaluate the model
accuracy_logistic_tuned = accuracy_score
report_logistic_tuned = classification_r

print("Logistic Regression Accuracy (With
print("Logistic Regression Classificatio
```

```
warnings.warn(
Best Parameters for Logistic Regression: {'C': 1, 'penalty': 'l2'}
Best Score for Logistic Regression: 0.85
Logistic Regression Accuracy (With Tuning): 0.7951219512195122
Logistic Regression Classification Report (With Tuning):
              precision    recall  f1-score   support

           0       0.85      0.72      0.78       102
           1       0.76      0.87      0.81       103

    accuracy                           0.80       205
   macro avg       0.80      0.79      0.79       205
weighted avg       0.80      0.80      0.79       205
```

RandomForestClassifier

```
]:  from sklearn.ensemble import RandomForestClassifier

    # Initialize and train the model
    rf_model = RandomForestClassifier()
    rf_model.fit(x_train, y_train)

    # Make predictions
    y_pred_rf = rf_model.predict(x_test)

    # Evaluate the model
    accuracy_rf = accuracy_score(y_test, y_pred_rf)
    report_rf = classification_report(y_test, y_pred_rf)

    print("Random Forest Accuracy:", accuracy_rf)
    print("Random Forest Classification Report:\n", report_rf)
```

```
Random Forest Accuracy: 0.9853658536585366
Random Forest Classification Report:
              precision    recall  f1-score   support

           0       0.97      1.00      0.99       102
           1       1.00      0.97      0.99       103

    accuracy                           0.99       205
   macro avg       0.99      0.99      0.99       205
weighted avg       0.99      0.99      0.99       205
```

WITH HP

```python
from sklearn.model_selection import GridSearchCV

# Define parameter grid
param_grid_svm = {
    'C': [0.1, 1, 10],
    'kernel': ['linear'],

}

# Initialize and train the model with GridSearch
grid_search_svm = GridSearchCV(SVC(), param_grid_svm, cv=5, n_jobs=-1)
grid_search_svm.fit(x_train, y_train)

# Best parameters and best score
print("Best Parameters for SVM:", grid_search_svm.best_params_)
print("Best Score for SVM:", grid_search_svm.best_score_)

# Make predictions
y_pred_svm_tuned = grid_search_svm.predict(x_test)

# Evaluate the model
accuracy_svm_tuned = accuracy_score(y_test, y_pred_svm_tuned)
report_svm_tuned = classification_report(y_test, y_pred_svm_tuned)

print("SVM Accuracy (With Tuning):", accuracy_svm_tuned)
print("SVM Classification Report (With Tuning):\n", report_svm_tuned)
```

```
Best Parameters for SVM: {'C': 0.1, 'kernel': 'linear'}
Best Score for SVM: 0.8512195121951219
SVM Accuracy (With Tuning): 0.7951219512195122
SVM Classification Report (With Tuning):
              precision    recall  f1-score   support

           0       0.88      0.68      0.77       102
           1       0.74      0.91      0.82       103

    accuracy                           0.80       205
   macro avg       0.81      0.79      0.79       205
weighted avg       0.81      0.80      0.79       205
```

KNN Classifier

```
from sklearn.neighbors import KNeighborsClassifier

# Initialize and train the model
knn_model = KNeighborsClassifier()
knn_model.fit(x_train, y_train)

# Make predictions
y_pred_knn = knn_model.predict(x_test)

# Evaluate the model
accuracy_knn = accuracy_score(y_test, y_pred_knn)
report_knn = classification_report(y_test, y_pred_knn)

print("KNN Accuracy:", accuracy_knn)
print("KNN Classification Report:\n", report_knn)
```

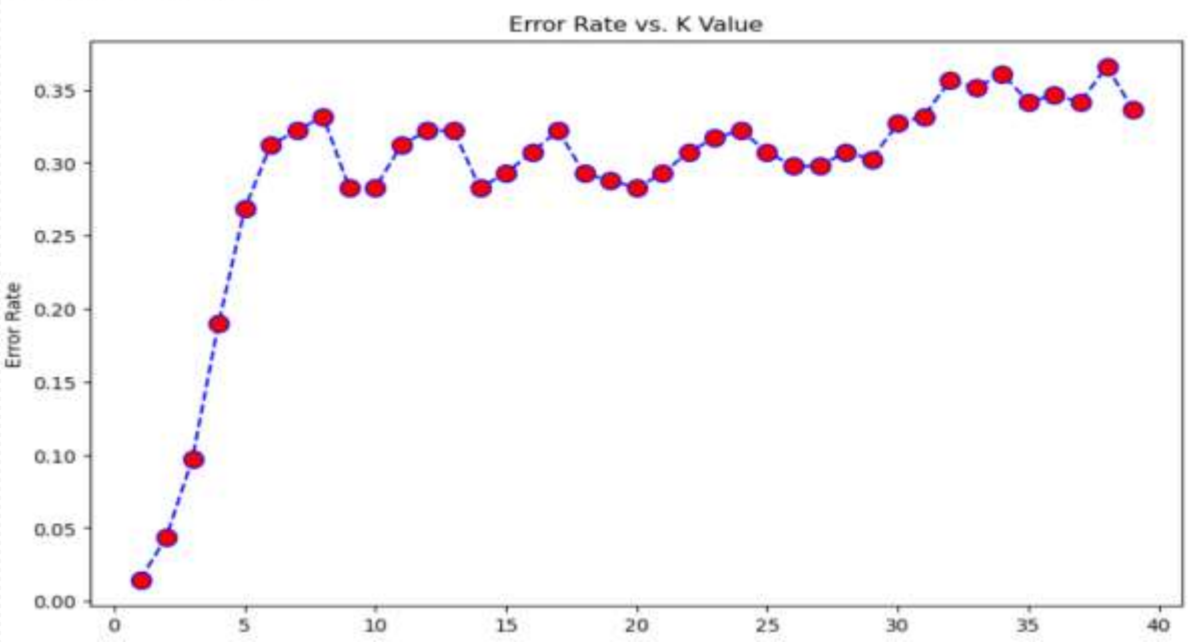
```
KNN Accuracy: 0.7317073170731707
KNN Classification Report:
               precision    recall  f1-score   support

           0       0.73      0.73      0.73       102
           1       0.73      0.74      0.73       103

    accuracy                           0.73       205
   macro avg       0.73      0.73      0.73       205
weighted avg       0.73      0.73      0.73       205
```

With HyperParametric Tuning

```
[51]: error_rate = []
for  i in range(1,40):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(x_train,y_train)
    pred_i=knn.predict(x_test)
    error_rate.append(np.mean(pred_i !=y_test))
```

```
[52]: plt.figure(figsize=(10,6))
plt.plot(range(1,40),error_rate,color='blue',linestyle='dashed',marker='o',
         markerfacecolor='red',markersize=10)
plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
```

Text(0, 0.5, 'Error Rate')



53]:

```python
from sklearn.metrics import confusion_matrix
knn= KNeighborsClassifier(n_neighbors=3)

knn.fit(x_train,y_train)
pred = knn.predict(x_test)

print('With K=3')
print('\n')
print(confusion_matrix(y_test,pred))
print(classification_report(y_test,pred))
```

```
With K=3



[[91 11]
 [ 9 94]]
              precision    recall  f1-score   support

           0       0.91      0.89      0.90       102
           1       0.90      0.91      0.90       103

    accuracy                           0.90       205
   macro avg       0.90      0.90      0.90       205
weighted avg       0.90      0.90      0.90       205
```

Exploring the CAR Dataset with different (REGRESSION ALGORITHMS)

We have a data which is car or not according to features in it. We will try to use this data to create a model which tries predict types of car and price. We will use regression algorithms.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
df=pd.read_csv("C:/kgisl class/MILESTONE - 2/car details v4.csv")
df
```

	Make	Model	Price	Y
0	Honda	Amaze 1.2 VX i-VTEC	505000	2
1	Maruti Suzuki	Swift DZire VDI	450000	2

```python
df.describe()
```

	Price	Year	Kilometer	Length	Width	Height
count	2.059000e+03	2059.000000	2.059000e+03	1995.000000	1995.000000	1995.000000
mean	1.702992e+06	2016.425449	5.422471e+04	4280.860652	1767.991980	1591.735338
std	2.419881e+06	3.363564	5.736172e+04	442.458507	135.265825	136.073956
min	4.900000e+04	1988.000000	0.000000e+00	3099.000000	1475.000000	1165.000000
25%	4.849990e+05	2014.000000	2.900000e+04	3985.000000	1695.000000	1485.000000
50%	8.250000e+05	2017.000000	5.000000e+04	4370.000000	1770.000000	1545.000000
75%	1.925000e+06	2019.000000	7.200000e+04	4629.000000	1831.50000	[illegible]
max	3.500000e+07	2022.000000	2.000000e+06	5569.000000	2220.00000	[illegible]

```python
df=df.drop(columns=['Length','Width','Height','Fuel Tank Capacity
df
```

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

```
Make                  0
Model                 0
Price                 0
Year                  0
Kilometer             0
Fuel Type             0
Transmission          0
Color                 0
Owner                 0
Seller Type           0
Engine               80
Max Power            80
Max Torque           80
Drivetrain          136
Seating Capacity     64
dtype: int64
```

```python
categorical=['Engine','Max Power','Max Torque','Drivetrain']
```

```python
for i in categorical:
    df[i].fillna(df[i].mode()[0], inplace=True)
```

```python
df['Seating Capacity']=df['Seating Capacity'].fillna(df['Seating Capacity'].median())
```

```python
df.describe()
```

	Price	Year	Kilometer	Seating Capacity
count	2.059000e+03	2059.000000	2.059000e+03	2059.000000
mean	1.702992e+06	2016.425449	5.422471e+04	5.296746
std	2.419881e+06	3.363564	5.736172e+04	0.811029
min	4.900000e+04	1988.000000	0.000000e+00	2.000000
25%	4.849990e+05	2014.000000	2.900000e+04	5.000000
50%	8.250000e+05	2017.000000	5.000000e+04	5.000000
75%	1.925000e+06	2019.000000	7.200000e+04	5.000000
max	3.500000e+07	2022.000000	2.000000e+06	8.000000

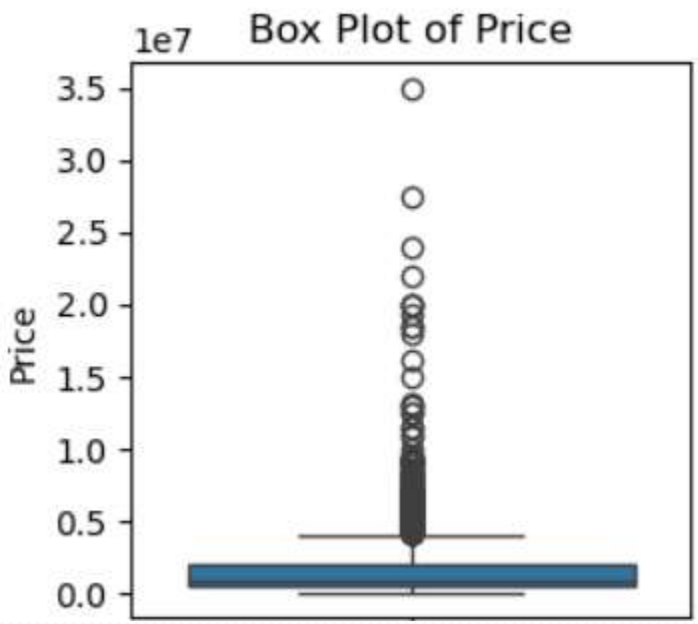
```python
df.isnull().sum()
```

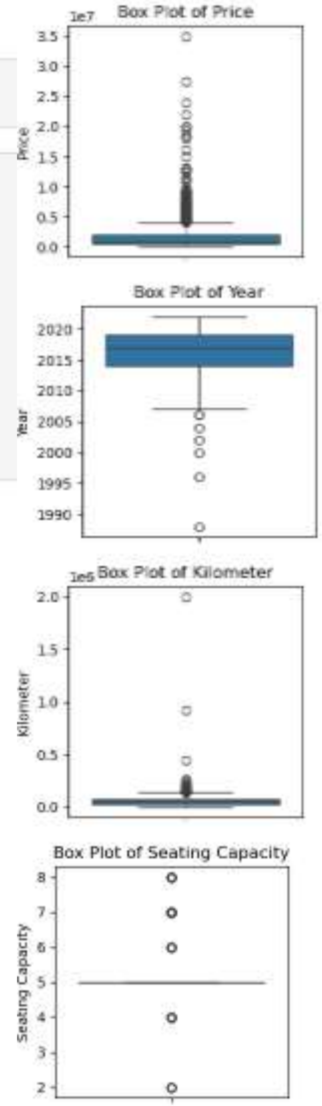
```
Make                0
Model               0
Price               0
Year                0
Kilometer           0
Fuel Type           0
Transmission        0
Color               0
Owner               0
Seller Type         0
Engine              0
Max Power           0
Max Torque          0
Drivetrain          0
Seating Capacity    0
dtype: int64
```

Outlier Detection

```
numerical_columns=['Price','Year','Kilometer','Seating Capacity']
```

```
import seaborn as sns
#outlier detection
for col in numerical_columns:
    plt.figure(figsize=(3, 3))
    sns.boxplot(y=df[col])
    plt.title(f'Box Plot of {col}')
    plt.show()
```





```python
# Calculate Q1 (25th percentile) and Q3 (75th percentile) for numerical columns
Q1 = df[numerical_columns].quantile(0.25)
Q3 = df[numerical_columns].quantile(0.75)
IQR = Q3 - Q1
IQR
```

```
Price               1440001.0
Year                      5.0
Kilometer             43000.0
Seating Capacity          0.0
dtype: float64
```

```python
outlier_mask = ((df[numerical_columns] < (Q1 - 1.5 * IQR)) | (df[numerical_columns] > (Q3 + 1.5 * IQR)))

# Check which rows contain outliers
outliers = outlier_mask.any(axis=1)
print(f"Number of rows with outliers: {outliers.sum()}")
```

```
Number of rows with outliers: 563
```

```python
df_cleaned = df[~outliers]

# Check the shape of the cleaned data
print(f"Original data shape: {df.shape
print(f"Cleaned data shape: {df_cleanec
```

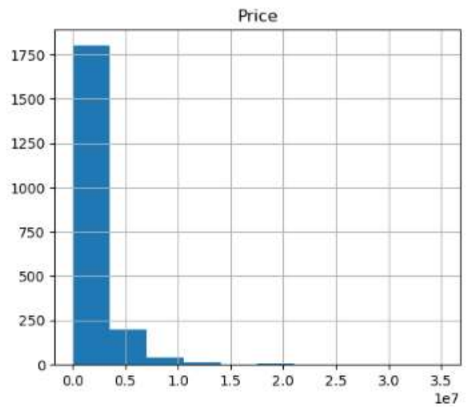
```
Original data shape: (2059, 15)
Cleaned data shape: (1496, 15)
```

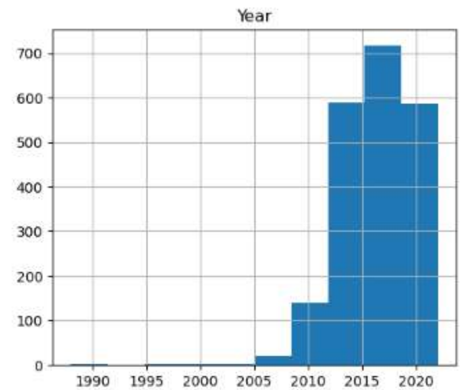
```python
df_cleaned
```

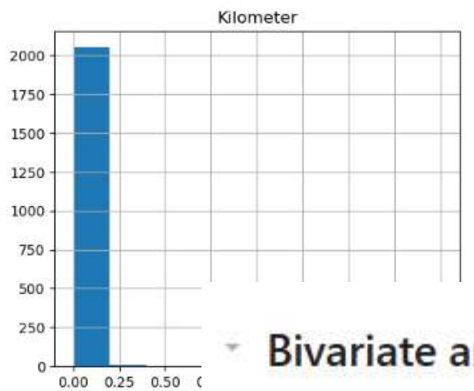
	Make	Model	Price	Year	Kilometer	Fuel Type	Transmission	Color	Owner	Seller Type	Engine	Max Power	Max Torque	Drivetrain	Seating Capacity
0	Honda	Amaze 1.2 VX i-VTEC	505000	2017	87150	Petrol	Manual	Grey	First	Corporate	1198 cc	87 bhp @ 6000 rpm	109 Nm @ 4500 rpm	FWD	5.0
1	Maruti Suzuki	Swift DZire VDI	450000	2014	75000	Diesel	Manual	White	Second	Individual	1248 cc	74 bhp @ 4000 rpm	190 Nm @ 2000 rpm	FWD	5.0
2	Hyundai	i10 Magna 1.2 Kappa2	220000	2011	67000	Petrol	Manual	Maroon	First	Individual	1197 cc	79 bhp @ 6000 rpm	112.7619 Nm @ 4000 rpm	FWD	5.0
3	Toyota	Glanza G	799000	2019	37500	Petrol	Manual	Red	First	Individual	1197 cc	82 bhp @ 6000 rpm	113 Nm @ 4200 rpm	FWD	5.0
5	Maruti Suzuki	Ciaz ZXi	675000	2017	73315	Petrol	Manual	Grey	First	Individual	1373 cc	91 bhp @ 6000 rpm	130 Nm @ 4000 rpm	FWD	5.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2051	Maruti Suzuki	Vitara Brezza VXi	925000	2021	48000	Petrol	Manual	White	First	Individual	1462 cc	103 bhp @ 6000 rpm	138 Nm @ 4400 rpm	FWD	5.0
2052	Hyundai	i20 Sportz 1.4 CRDI	409999	2014	68000	Diesel	Manual	Silver	First	Individual	1396 cc	90@4000	220@1750	FWD	5.0
2053	Maruti Suzuki	Ritz Vxi (ABS) BS-IV	245000	2014	79000	Petrol	Manual	White	Second	Individual	1197 cc	85 bhp @ 6000 rpm	113 Nm @ 4500 rpm	FWD	5.0

Univariate Analysis

[18]:
```python
# Histograms
df[['Price', 'Year', 'Kilometer', 'Engine', 'Max Power', 'Max Torque']].hist(figsize=(12, 10))
plt.show()
```







Bivariate analysis

[ ]:

[19]:
```python
# Function to extract numeric values from strings
def extract_numeric(value):
    try:
        return float(value.split()[0].replace(',', ''))
    except:
        return None

# Apply the function to convert columns
df_cleaned['Engine'] = df_cleaned['Engine'].apply(extract_numeric)
df_cleaned['Max Power'] = df_cleaned['Max Power'].apply(extract_numeric)
df_cleaned['Max Torque'] = df_cleaned['Max Torque'].apply(extract_numeric)
```

]:
```python
df_cleaned.isnull().sum()
```

]:
```
Make                0
Model               0
Price               0
Year                0
Kilometer           0
Fuel Type           0
Transmission        0
Color               0
Owner               0
Seller Type         0
Engine              0
Max Power          99
Max Torque         99
Drivetrain          0
Seating Capacity    0
dtype: int64
```

```python
df_cleaned['Max Power'].fillna(df_cleaned['Max Power'].median(), inplace=True)
df_cleaned['Max Torque'].fillna(df_cleaned['Max Torque'].median(), inplace=True)
```

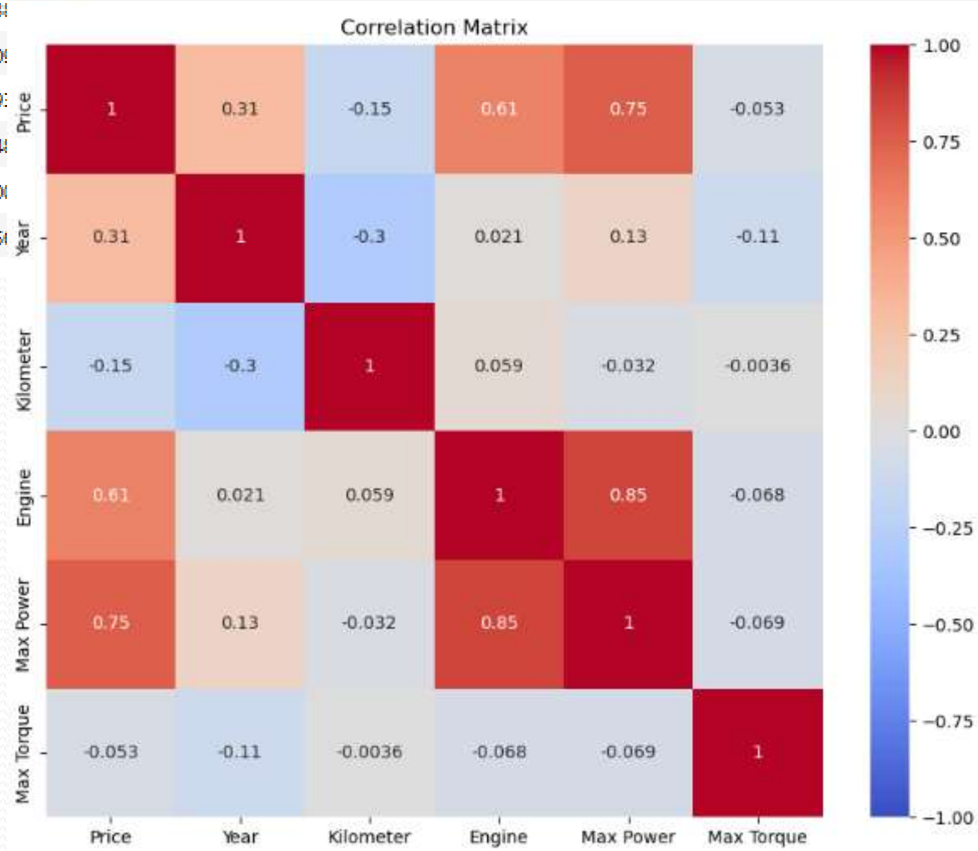
```python
df_cleaned = df_cleaned.dropna()
```

```python
# Remove non-numeric characters and convert to numeric
df['Engine'] = df['Engine'].replace('[^\d]', '', regex=True).astype(float)
df['Max Power'] = df['Max Power'].replace('[^\d]', '', regex=True).astype(float)
df['Max Torque'] = df['Max Torque'].replace('[^\d]', '', regex=True).astype(float)
```

```python
corr_matrix = df[['Price', 'Year', 'Kilometer', 'Engine', 'Max Power', 'Max Torque']].corr()
corr_matrix
```

	Price	Year	Kilometer	Engine	Max Powe
Price	1.000000	0.311400	-0.150825	0.608255	0.75333
Year	0.311400	1.000000	-0.296547	0.021308	0.12670
Kilometer	-0.150825	-0.296547	1.000000	0.058900	-0.03239
Engine	0.608255	0.021308	0.058900	1.000000	0.84824
Max Power	0.753338	0.126709	-0.032393	0.848248	1.00000
Max Torque	-0.053103	-0.112548	-0.003623	-0.068478	-0.06855

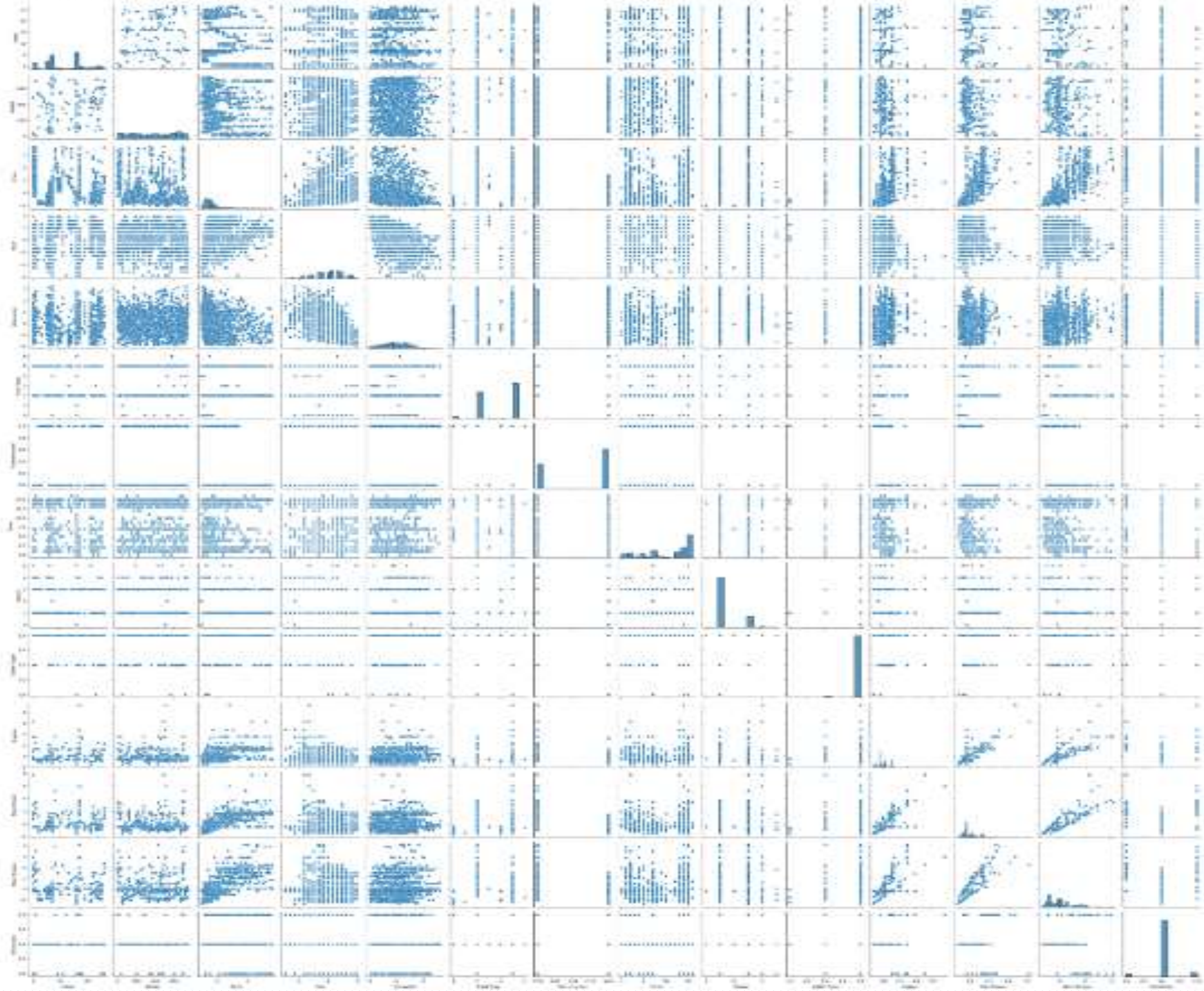
```python
# Heatmap of correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', vmin=-1, vmax=1)
plt.title('Correlation Matrix')
plt.show()
```



[37]:

```
sns.pairplot(df_cleaned)
```

[37]: <seaborn.axisgrid.PairGrid at 0x22f2a746e10>



```python
x = df_cleaned.drop(columns=['Price'])
y = df_cleaned['Price']
```

```python
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, r2_score
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
from sklearn.metrics import classification_report
```

```python
linear=LinearRegression()
linear.fit(x_train,y_train)
y_predict=linear.predict(x_test)
mse1 = mean_squared_error(y_test,y_predict)
r2_sq = np.sqrt(mse1)

print("MSE of LR: ",mse1)
print('R2 of LR: ', r2_sq)
```

```
MSE of LR:  0.24322150109243054
R2 of LR:  0.4931749193667806
```

```python
from sklearn.linear_model import Ridge,Lasso
from sklearn.model_selection import GridSearchCV

# Define parameter grid for Ridge regression
param_grid_ridge = {'alpha': [0.1, 1, 10, 100]}

# Create a Ridge regression model
ridge = Ridge()

# Perform grid search with cross-validation
grid_search_ridge = GridSearchCV(ridge, param_grid_ridge, cv=5, scoring='neg_mean_squared_error')
grid_search_ridge.fit(x_train, y_train)

# Best parameters and score
best_params_ridge = grid_search_ridge.best_params_
best_score_ridge = -grid_search_ridge.best_score_

print("Best parameters for Ridge:", best_params_ridge)
print("Best score (MSE) for Ridge:", best_score_ridge)
```

```
Best parameters for Ridge: {'alpha': 10}
Best score (MSE) for Ridge: 0.24550850761586768
```

```
best_ridge_model = grid_search_ridge.best_estimator_
y_pred_ridge = best_ridge_model.predict(x_test)
mse_ridge = mean_squared_error(y_test, y_pred_ridge)
print("Test MSE for Ridge:", mse_ridge)
```

```
Test MSE for Ridge: 0.2427443602402604
```

```
# Define parameter grid for Lasso regression
param_grid_lasso = {'alpha': [0.1, 1, 10, 100]}

# Create a Lasso regression model
lasso = Lasso()

# Perform grid search with cross-validation
grid_search_lasso = GridSearchCV(lasso, param_grid_lasso, cv=5, scoring='neg_mean_squared_error')
grid_search_lasso.fit(x_train, y_train)

# Best parameters and score
best_params_lasso = grid_search_lasso.best_params_
best_score_lasso = -grid_search_lasso.best_score_

print("Best parameters for Lasso:", best_params_lasso)
print("Best score (MSE) for Lasso:", best_score_lasso)

# Evaluate on test set
best_lasso_model = grid_search_lasso.best_estimator_
y_pred_lasso = best_lasso_model.predict(x_test)
mse_lasso = mean_squared_error(y_test, y_pred_lasso)
print("Test MSE for Lasso:", mse_lasso)
```

```
Best parameters for Lasso: {'alpha': 0.1}
Best score (MSE) for Lasso: 0.3024644206576163 6
Test MSE for Lasso: 0.272531288061480 5
```

Applying Random Forest Regressor

```
from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
```

```
rfr=RandomForestRegressor(random_state=42)
```

```
rfr.fit(x_train, y_train)
```

```
▾ RandomForestRegressor
RandomForestRegressor(random_state=42)
```

```
y_pred = rfr.predict(x_test)

mse= mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Without Hyperparameter Tuning:")
print("Mean Squared Error:", mse)
print("R-squared:", r2)
```

```
Without Hyperparameter Tuning:
Mean Squared Error: 0.08856394845372201
R-squared: 0.912306620755473
```

```
# with hyperparametric tuning
```

```
param_grid={
    'n_estimators':[50,100,200],
    'max_features':['auto','sqrt'],
    'max_depth':[10,20,30],
    'min_samples_split':[2,5,10],
    'min_samples_leaf':[1,2,4]
}
```

```
grid_search =GridSearchCV(estimator=rfr,param_grid=param_grid , cv=5, scoring='neg_mean_squared_error', n_jobs=-1,verbose=2)
```

```
grid_search.fit(x_train,y_train)
```

```
Fitting 5 folds for each of 162 candidates, totalling 810 fits
▸ GridSearchCV
▸ estimator: RandomForestRegressor
  ▸ RandomForestRegressor
```

```
best_params = grid_search.best_params_
print(best_params)
best_rfr= grid_search.best_estimator_
print(best_rfr)
```

```
{'max_depth': 20, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200}
RandomForestRegressor(max_depth=20, max_features='sqrt', n_estimators=200,
                      random_state=42)
```

```
y_pred=best_rfr.predict(x_test)
```

```
mse=mean_squared_error(y_test,y_pred)
mse
```

```
0.0860003557006667
```

```
r2_score= r2_score(y_test,y_pred)
r2_score
```

```
0.9148450138087103
```

```
from sklearn.svm import SVR

# SVR without tuning
svr_model = SVR()
svr_model.fit(x_train, y_train)
y_pred_svr = svr_model.predict(x_test)
```

```
from sklearn.svm import SVR

# SVR without tuning
svr_model = SVR()
svr_model.fit(x_train, y_train)
y_pred_svr = svr_model.predict(x_test)
```

```
from sklearn.metrics import r2_score

# Evaluate SVR
r2_svr = r2_score(y_test, y_pred_svr)
mse_svr = mean_squared_error(y_test, y_pred_svr)
print(f"SVR R2: {r2_svr}")
print(f"SVR MSE: {mse_svr}")
```

```
SVR R2: -0.09972620190197734
SVR MSE: 1.1106436483291606
```

```
# SVR with tuning
param_grid_svr = {'C': [0.1, 1, 10], 'epsilon': [0.1, 0.01]}
svr_cv = GridSearchCV(SVR(), param_grid_svr, cv=5)
svr_cv.fit(x_train, y_train)
y_pred_svr_tuned = svr_cv.predict(x_test)
```

```
# Evaluate tuned SVR
r2_svr_tuned = r2_score(y_test, y_pred_svr_tuned)
mse_svr_tuned = mean_squared_error(y_test, y_pred_svr_tuned)
print(f"Tuned SVR R2: {r2_svr_tuned}")
print(f"Tuned SVR MSE: {mse_svr_tuned}")
```

```
Tuned SVR R2: 0.17398036676004547
Tuned SVR MSE: 0.8342198789721214
```

Thank you