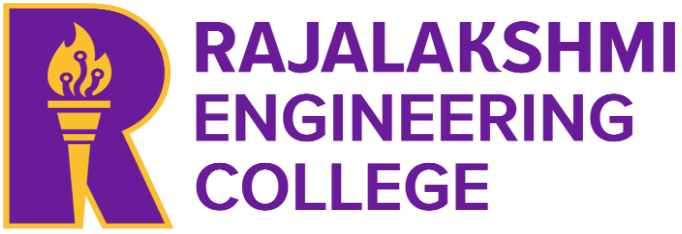
RAJALAKSHMI ENGINEERING COLLEGE
(Autonomous)

RAJALAKSHMI NAGAR, THANDALAM, CHENNAI-602105



DS24122 – DATA SCIENCE LAB
LABORATORY RECORD NOTEBOOK

Name:	**KHAVIN C**
Year / Branch / Section:	**I / M.tech DATA SCIENCE**
Register No:	**251011010**
Semester:	**I**
Academic Year:	**2025-2026**

RAJALAKSHMI ENGINEERING COLLEGE

[AUTONOMOUS]

RAJALAKSHMI NAGAR, THANDALAM – 602 105

BONAFIDE CERTIFICATE

Name : . . . . . KHAVIN C.. . . . . .

Academic Year : 2025-2026 Semester : I Branch : DATA SCIENCE

Register No.	**251011010**

Certified that this is the bonafide record of work done by the above student in the **DS24122 – DATA SCIENCE LAB** during the year 2025 - 2026.

Signature of the Faculty In- charge

Submitted for the Practical Examination held on . . . . . .

Internal Examiner **External Examiner**

RAJALAKSHMI ENGINEERING COLLEGE

(An Autonomous Institution affiliated to Anna University)

DEPARTMENT OF INFORMATION TECHNOLOGY

Subject: DS24122 – DATA SCIENCE LAB

Reg No: 251011010
Name: KHAVIN C
Class & Section: M.TECH DATA SCIENCE

INDEX

S.NO	DATE	TITLE	QR	SIGN
1.		Outlier Detection, Data Preprocessing, and Model Accuracy Enhancement		
2.		Activation Functions in Neural Networks and Performance Optimization		
3.		Anomaly Detection Techniques and K-Means Clustering Analysis		

S.NO	DATE	TITLE	QR	SIGN
4.		Synthetic Data Generation Using GANs for IoT Systems		
5.		Evolution and Advancements of Style-Based GANs		
6.		Deepfakes Using GANs: Creation and Detection Techniques		

Exp. No 1 Date:	Outlier Detection, Data Preprocessing, and Model Accuracy Enhancement

Task:

Present your view on the different techniques you have employed to do outlier analysis, handling missing data, feature engineering, feature importance and improving the accuracy of the model both from a classifier as well as a regressor. Use any sample data and present your POV in a well-structured presentation.

Aim:

To analyze various data preprocessing and feature optimization techniques including outlier detection, missing value handling, feature engineering, and feature selection. Further, to evaluate their impact on improving the performance and accuracy of both classification and regression models using sample datasets.

Algorithm:

1. Load the dataset and required libraries for preprocessing and modeling.
2. Perform exploratory data analysis (EDA) to understand data types, distribution, missing values, and summary statistics.
3. Engineer new features (e.g., car age) and remove/rename irrelevant or inconsistent columns.
4. Detect and handle outliers using visualization and the Interquartile Range (IQR) method.
5. Handle missing data using mean/median for numerical columns and mode for categorical columns.
6. Convert categorical features into numerical values using Label Encoding or One-Hot Encoding.
7. Scale numerical features using Standardization or Normalization if required by the model.
8. Split the data into training and testing sets for both regression and (optional) classification tasks.
9. Train and tune machine learning models such as Random Forest and Gradient Boosting using cross-validation to improve accuracy.
10. Evaluate model performance using metrics like R^2 for regression or Accuracy for classification, and analyze feature importance.

Program:


```
# Numeric: fill with median, categorical: fill with mode
for col in data.columns:
    if data[col].dtype == 'object':
        data[col].fillna(data[col].mode()[0], inplace=True)
    else:
        data[col].fillna(data[col].median(), inplace=True)

# Encode categorical columns
cat_cols = data.select_dtypes(include='object').columns
le = LabelEncoder()
for col in cat_cols:
    data[col] = le.fit_transform(data[col].astype(str))

# Check types
print(data.dtypes)
```

```
ID                    int64
Price(USD)            int64
Levy                  int64
Manufacturer          int64
Model                 int64
Category              int64
Leather interior      int64
Fuel type             int64
Engine volume         int64
Mileage               int64
Num_Cylinders       float64
Gear box type         int64
Drive wheels          int64
Doors                 int64
Wheel                 int64
Color                 int64
Num_Airbags           int64
Age                   int64
dtype: object
```

In [56]:
```
df_main['Mileage'] = pd.to_numeric(df_main['Mileage'], errors='coerce')
```

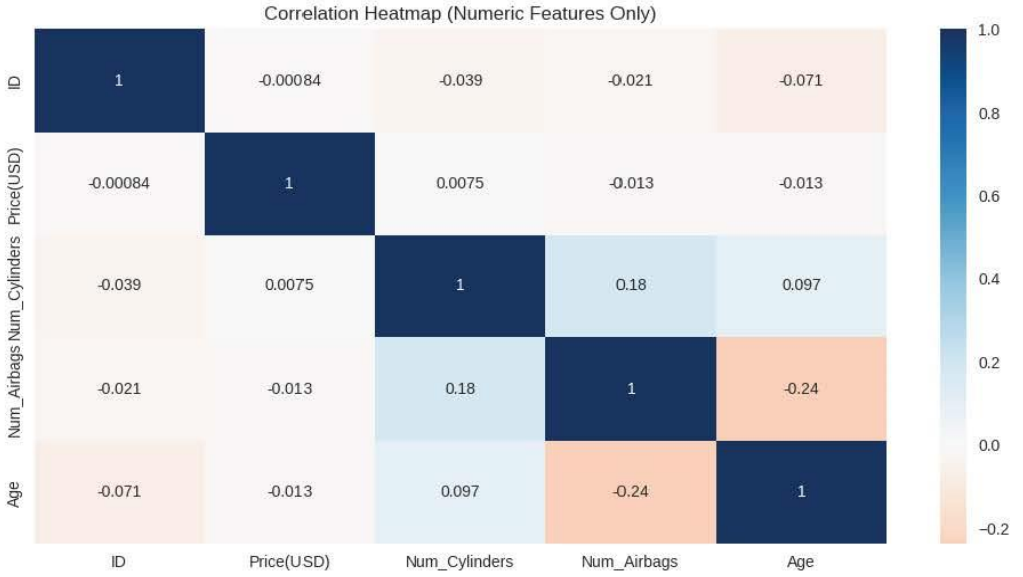
In [57]:
```
Q1 = df_main['Price(USD)'].quantile(0.25)
Q3 = df_main['Price(USD)'].quantile(0.75)
IQR = Q3 - Q1
df_main = df_main[
    (df_main['Price(USD)'] >= Q1 - 1.5*IQR) &
    (df_main['Price(USD)'] <= Q3 + 1.5*IQR)
]
```

In [58]:
```
df_main['Mileage_per_Year'] = df_main['Mileage'] / df_main['Age'].replace(0,
```

In [59]:
```
df_main['Price(USD)'] = np.log1p(df_main['Price(USD)'])
```

In [64]:
```
from sklearn.preprocessing import OneHotEncoder
from sklearn.ensemble import GradientBoostingRegressor
```

In [18]:
```
plt.figure(figsize=(12,6))
sns.heatmap(df_main.corr(numeric_only=True), annot=True, cmap="RdBu", center
plt.title("Correlation Heatmap (Numeric Features Only)")
plt.show()
```



In [19]:
```
df_main.corr(numeric_only=True)['Price(USD)'].sort_values(ascending=False)
```

Out[19]:

	Price(USD)
Price(USD)	1.000000
Num_Cylinders	0.007518
ID	-0.000844
Num_Airbags	-0.012824
Age	-0.012982

dtype: float64

In [48]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV, cross_va
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.ensemble import RandomForestRegressor, RandomForestClassifier
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score, cl
from sklearn.pipeline import Pipeline

# Copy dataframe to avoid accidental changes
data = df_main.copy()

# Handle missing values
# Numeric: fill with median; Categorical: fill with mode
```

17141	44949023	94083	1104	BMW	X5 M	Jeep	Yes
17278	45800245	90947	-	FORD	Ranger Wildtrak	Pickup	Yes
17406	45809552	93079	1866	CHEVROLET	Colorado	Pickup	Yes
17463	45809093	96263	866	FORD	Mustang	Sedan	Yes
17471	44036373	106627	-	CHEVROLET	Camaro	Coupe	No
17666	45810524	109764	1575	TOYOTA	Land Cruiser Prado	Jeep	No
17728	43889777	97219	-	LEXUS	GX 460	Jeep	Yes
17760	42873373	87811	-	BMW	750	Sedan	Yes
17868	43720197	150533	-	JAGUAR	F-type R	Sedan	Yes
17962	45474920	136000	1575	MERCEDES-BENZ	GLE 400	Jeep	Yes
18018	45740382	127000	-	JEEP	Cherokee	Jeep	Yes
18231	45183554	131716	-	TOYOTA	Land Cruiser	Jeep	Yes
18285	44655192	89379	-	BMW	X5	Jeep	Yes
18376	45803018	100060	1176	HYUNDAI	Santa FE	Jeep	Yes
18383	45754210	100355	-	LEXUS	GS 350	Sedan	Yes
18640	44080587	111332	-	LEXUS	GX 460	Jeep	Yes
18720	45049508	86243	-	BMW	X6 40D	Jeep	Yes
18815	45808873	111002	1228	FORD	Explorer	Jeep	Yes
18881	44587551	147397	-	JAGUAR	F-pace	Jeep	No
18941	45809093	96263	866	FORD	Mustang	Sedan	Yes
19085	45794339	97219	1079	LEXUS	NX 300	Jeep	Yes
19121	45732079	88438	1076	HYUNDAI	Santa FE	Jeep	Yes

15048	45722415	100355	915	BMW	X5	Jeep	Yes
15245	45732085	96915	1174	HYUNDAI	Santa FE	Jeep	Yes
15267	45811590	120740	2297	MERCEDES-BENZ	G 63 AMG	Jeep	Yes
15283	45069516	250574	1481	MERCEDES-BENZ	GLE 400	Jeep	Yes
15367	42303965	100355	1714	MERCEDES-BENZ	GLE 350	Jeep	Yes
15413	45801212	153669	1823	BMW	M3	Sedan	Yes
15486	41976837	90006	-	BMW	X5	Jeep	Yes
15496	45816476	130148	-	MERCEDES-BENZ	S 350	Sedan	Yes
15586	45188389	89379	1079	MERCEDES-BENZ	E 300	Sedan	Yes
15852	26327387	87021	-	MERCEDES-BENZ	Sprinter 516	Microbus	Yes
15966	45809064	126322	1493	FORD	F150	Sedan	Yes
16019	45754903	94083	1104	BMW	X5	Jeep	Yes
16084	45804434	106957	518	LEXUS	NX 300	Jeep	Yes
16483	45802850	87927	1076	HYUNDAI	Santa FE	Jeep	Yes
16519	45649425	87999	1714	MERCEDES-BENZ	GLE 350	Jeep	Yes
16525	45813886	89379	2225	LEXUS	LS 460	Sedan	Yes
16614	42571375	100355	-	PORSCHE	Panamera GTS	Sedan	Yes
16667	45107574	141124	1481	MERCEDES-BENZ	GLE 43 AMG	Coupe	Yes
16841	45802552	118676	1968	HYUNDAI	Santa FE	Jeep	Yes
16873	45789705	94083	-	BMW	X5	Jeep	Yes
16983	45812886	26307500	-	OPEL	Combo	Goods wagon	No
17085	45457799	94083	-	TOYOTA	Land Cruiser	Jeep	Yes

13114	43153632	116036	-	TOYOTA	Highlander	Jeep	Yes
13189	45808922	92070	1011	FORD	Explorer	Jeep	Yes
13265	39856789	144261	-	BMW	M4 Competition	Coupe	Yes
13328	44956069	193184	-	MERCEDES-BENZ	GLE 400 Coupe, AMG Kit	Jeep	Yes
13330	41228647	89379	-	MERCEDES-BENZ	ML 300	Jeep	Yes
13351	45775006	163077	-	BMW	M6 Gran cupe	Sedan	Yes
13572	24367759	85702	-	MERCEDES-BENZ	Sprinter 516	Microbus	Yes
13760	45282558	141124	-	MERCEDES-BENZ	GLC 250	Jeep	Yes
13884	45780625	103491	-	TOYOTA	Land Cruiser 200	Jeep	Yes
13954	45809052	119009	1342	FORD	F150	Sedan	Yes
14012	45410118	90947	-	TOYOTA	Land Cruiser Prado	Jeep	No
14092	45808914	91775	1714	FORD	Explorer	Jeep	Yes
14151	45049953	129521	1493	MERCEDES-BENZ	GLE 350	Jeep	Yes
14181	45808855	99440	1156	FORD	Explorer	Jeep	Yes
14193	45253360	106627	-	LEXUS	LX 570	Jeep	Yes
14255	45804430	113025	518	LEXUS	NX 300	Jeep	Yes
14473	45283485	131716	1017	MERCEDES-BENZ	GLC 300 GLC coupe	Jeep	Yes
14523	45095554	94083	-	PORSCHE	911	Coupe	Yes
14763	45517726	141124	-	MERCEDES-BENZ	S 550	Sedan	Yes
14839	45792307	297930	-	LAND ROVER	Range Rover Vogue	Jeep	Yes
14952	45487473	103491	-	PORSCHE	Cayenne	Jeep	Yes
15031	45813605	95000	-	MERCEDES-BENZ	Sprinter 516	Microbus	Yes

10423	44830858	150333	1979	LAND ROVER	Velar	Jeep	Yes
10569	45202115	97219	-	MERCEDES-BENZ	CLS 450 CLS 400	Sedan	Yes
10577	45808848	108371	1228	FORD	Explorer	Jeep	Yes
10619	45793295	103491	-	OPEL	Combo	Goods wagon	No
10690	45813974	88595	1935	MERCEDES-BENZ	S 550	Sedan	Yes
10759	45291161	260296	-	LEXUS	LX 570	Jeep	Yes
11109	45802849	97235	1076	HYUNDAI	Santa FE	Jeep	Yes
11131	45813717	103491	-	TOYOTA	Land Cruiser	Jeep	No
11247	45781858	125444	-	JAGUAR	E-pace	Jeep	Yes
11537	40648715	103178	-	PORSCHE	macan S	Jeep	Yes
11554	45646707	87027	1572	MERCEDES-BENZ	CLS 55 AMG	Coupe	Yes
11853	45808895	94528	1156	FORD	Explorer	Jeep	Yes
11914	45808928	89103	1011	FORD	Explorer	Jeep	Yes
11941	45761450	156805	-	BMW	X5 M	Jeep	Yes
12101	43965200	121681	-	TOYOTA	Land Cruiser Prado	Jeep	Yes
12303	45802541	106668	1176	HYUNDAI	Santa FE	Minivan	Yes
12420	40471256	114468	915	LAND ROVER	Range Rover	Jeep	Yes
12480	45732083	96749	1076	HYUNDAI	Santa FE	Jeep	Yes
12554	45368138	114468	1104	BMW	X6	Jeep	Yes
12588	45712786	122308	-	PORSCHE	Cayenne	Jeep	Yes
12593	44676343	94083	-	MITSUBISHI	Pajero	Jeep	Yes
12750	45809098	137885	2566	FORD	Mustang	Coupe	Yes
12976	45802853	93079	1174	HYUNDAI	Santa FE	Jeep	Yes

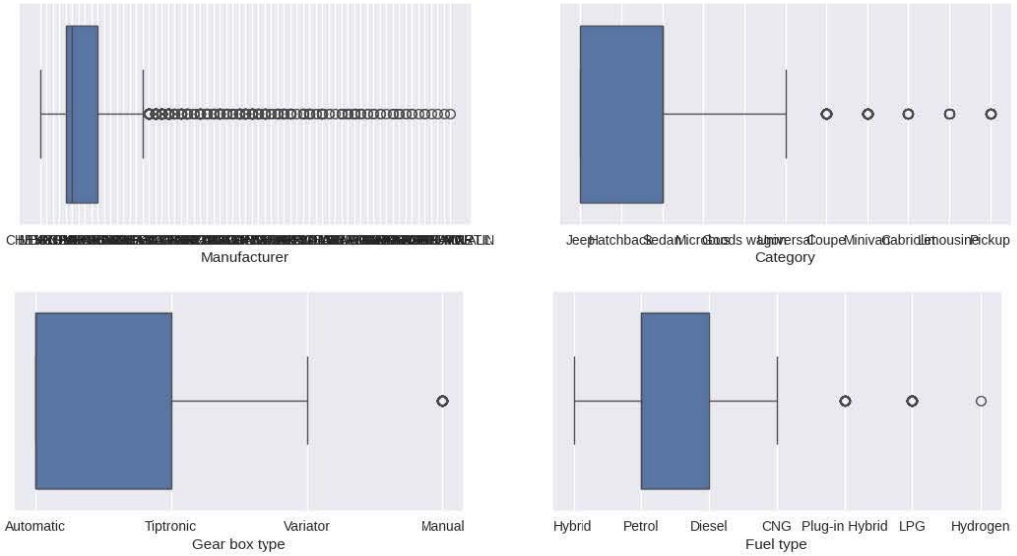
				BENZ			
8740	45809111	86142	866	FORD	Mustang	Cabriolet	Yes
8910	41177629	103491	1104	BMW	640 M	Sedan	Yes
9013	44732858	111332	-	AUDI	A8	Sedan	Yes
9120	45283485	131716	1017	MERCEDES-BENZ	GLC 300 GLC coupe	Jeep	Yes
9138	45806180	101923	-	JAGUAR	E-pace p200	Jeep	No
9166	45802550	95739	1176	HYUNDAI	Santa FE	Minivan	Yes
9200	45748414	141124	1575	MERCEDES-BENZ	GLS 450	Jeep	Yes
9247	43374774	106627	-	PORSCHE	Panamera 4	Sedan	Yes
9248	45761401	172486	-	PORSCHE	macan	Jeep	Yes
9367	45229113	297930	-	MERCEDES-BENZ	AMG GT S	Coupe	Yes
9405	45781828	98787	1356	LEXUS	GX 460	Jeep	Yes
9580	45808919	92070	1493	FORD	Explorer	Jeep	Yes
9688	45490304	89379	1356	LEXUS	GX 460	Jeep	Yes
10125	45812487	94083	431	BMW	X5	Jeep	Yes
10162	44266062	87811	-	TOYOTA	Camry	Sedan	Yes
10220	45805870	93769	2216	CHEVROLET	Suburban	Jeep	Yes
10230	45804437	99440	518	LEXUS	NX 300	Jeep	Yes
10238	45694885	103491	-	BMW	X5	Jeep	Yes
10253	45785166	131716	-	TOYOTA	Land Cruiser Prado	Jeep	Yes
10372	45802548	98066	1176	HYUNDAI	Santa FE	Minivan	Yes
10390	45809039	104216	1058	FORD	F150	Sedan	Yes
10423	44650638	150533	1575	LAND ROVER	Range Rover	Jeep	Yes

6652	45808861	93371	1156	FORD	Explorer	Jeep	Yes
6748	45283485	131716	1017	MERCEDES-BENZ	GLC 300 GLC coupe	Jeep	Yes
6825	45809046	107841	1053	FORD	F150	Sedan	Yes
6950	45813760	101923	1482	FERRARI	456	Coupe	Yes
6953	45369356	114468	1292	LAND ROVER	Land Rover Sport	Jeep	Yes
7010	45157625	117604	-	PORSCHE	911	Coupe	Yes
7031	45789817	141124	1304	BMW	M6	Coupe	Yes
7283	45420737	228935	-	MERCEDES-BENZ	GLE 63 AMG	Jeep	Yes
7353	43683199	216391	-	MERCEDES-BENZ	G 65 AMG G63 AMG	Jeep	Yes
7372	44479385	94080	-	BMW	X5	Jeep	Yes
7461	45777353	87497	1275	HONDA	Cr-v	Sedan	No
7590	45732367	89482	1176	HYUNDAI	Santa FE	Jeep	Yes
7718	43797018	167781	1292	LAND ROVER	Range Rover	Jeep	Yes
7723	45151658	106627	1017	BMW	530 G30	Sedan	Yes
7749	45760644	288521	2269	BMW	M5 Машина в максимально	Sedan	Yes
7760	45611161	101923	-	MERCEDES-BENZ	GL 550	Jeep	Yes
7997	45812826	147397	1869	JEEP	Wrangler	Jeep	Yes
8164	45434004	114154	1017	BMW	530	Sedan	Yes
8437	45803021	96071	1178	HYUNDAI	Santa FE	Jeep	Yes
8541	45761204	872946	2067	LAMBORGHINI	Urus	Universal	Yes
8629	45802553	123330	1176	HYUNDAI	Santa FE	Jeep	Yes
8706	45643521	86243	1292	MERCEDES-BENZ	GL 450 3.0	Jeep	Yes

3641	45732084	96749	1076	HYUNDAI	Santa FE	Jeep	Yes
3705	45237191	100355	-	BMW	X5	Jeep	Yes
3997	45803020	97401	1077	HYUNDAI	Santa FE	Jeep	Yes
4044	45793109	156805	-	MERCEDES-BENZ	GLS 450	Jeep	Yes
4077	45802851	90919	1076	HYUNDAI	Santa FE	Jeep	Yes
4183	45795846	125500	1604	MERCEDES-BENZ	S 550	Sedan	Yes
4211	45687064	92515	-	AUDI	Allroad	Universal	Yes
4590	45732088	88438	1076	HYUNDAI	Santa FE	Jeep	Yes
4629	45791036	94083	-	MERCEDES-BENZ	CLS 63 AMG	Sedan	Yes
4722	45813777	175622	2819	BENTLEY	Mulsanne	Sedan	Yes
4919	45773951	121524	3057	MERCEDES-BENZ	CLS 55 AMG	Sedan	Yes
5008	45810285	308906	1694	PORSCHE	911	Coupe	Yes
5322	45808902	118519	1228	FORD	Explorer	Jeep	Yes
5412	44343992	86243	-	MERCEDES-BENZ	S 63 AMG	Sedan	Yes
5709	45722256	95651	915	BMW	X5	Jeep	Yes
5840	44991441	254024	1292	MERCEDES-BENZ	GLE 400 A M G	Jeep	Yes
5987	45788500	117604	-	JAGUAR	F-pace	Jeep	Yes
6096	45382432	109764	-	MERCEDES-BENZ	X 250 ევროპული	Pickup	Yes
6152	45815648	100355	1292	MERCEDES-BENZ	CLS 550	Sedan	Yes
6457	45025127	136420	1493	MERCEDES-BENZ	GLE 350	Jeep	Yes
6468	45796870	172486	-	LEXUS	LX 570	Jeep	Yes
6607	32171534	119172	-	MERCEDES-BENZ	230 W153	Sedan	Yes

1179	45708671	133284	690	LEXUS	RX 450	Jeep	Yes
1225	45795524	627220	-	MERCEDES-BENZ	G 65 AMG 63AMG	Jeep	Yes
1265	45324859	106627	-	BMW	X6	Jeep	Yes
1401	45802554	131308	1968	HYUNDAI	Santa FE	Jeep	Yes
1573	44721988	87497	-	KIA	Sportage	Jeep	Yes
1609	45001116	112900	-	MITSUBISHI	Pajero Sport	Jeep	Yes
1626	45732087	96915	1076	HYUNDAI	Santa FE	Jeep	Yes
1740	45803021	96071	1178	HYUNDAI	Santa FE	Jeep	Yes
1970	39854899	114468	915	LAND ROVER	Land Rover Sport	Jeep	Yes
2283	45786808	219527	-	BENTLEY	Continental GT	Coupe	Yes
2353	45804409	116036	1325	MERCEDES-BENZ	X 250	Pickup	Yes
2365	45792216	97219	-	BMW	640 GRAN-COUPE	Sedan	Yes
2676	45763237	94083	1104	BMW	X5	Jeep	Yes
2728	45808865	98282	1714	FORD	Explorer	Jeep	Yes
2768	45772445	172486	-	TOYOTA	Land Cruiser	Jeep	Yes
2912	45798754	172486	1951	MERCEDES-BENZ	S 63 AMG	Sedan	Yes
2922	45788165	109764	-	MERCEDES-BENZ	C 63 AMG	Sedan	Yes
2941	42249585	87811	-	MERCEDES-BENZ	G 55 AMG	Jeep	Yes
3006	45747002	141124	-	LAND ROVER	Land Rover Sport	Jeep	Yes
3028	45047684	119172	-	LEXUS	RX 350 F sport	Jeep	Yes
3101	45781346	111332	-	TOYOTA	Land Cruiser	Jeep	Yes
3216	45809613	124989	1077	HONDA	Civic	Universal	Yes
3570	45809069	132971	1493	FORD	F150	Sedan	Yes

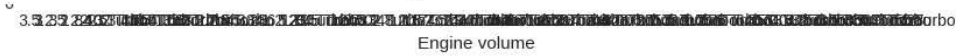
```
    plt.subplot(1,2,2)
    sns.boxplot(x=num_cols[i], data=df_main)
    i += 1

    plt.show()
```



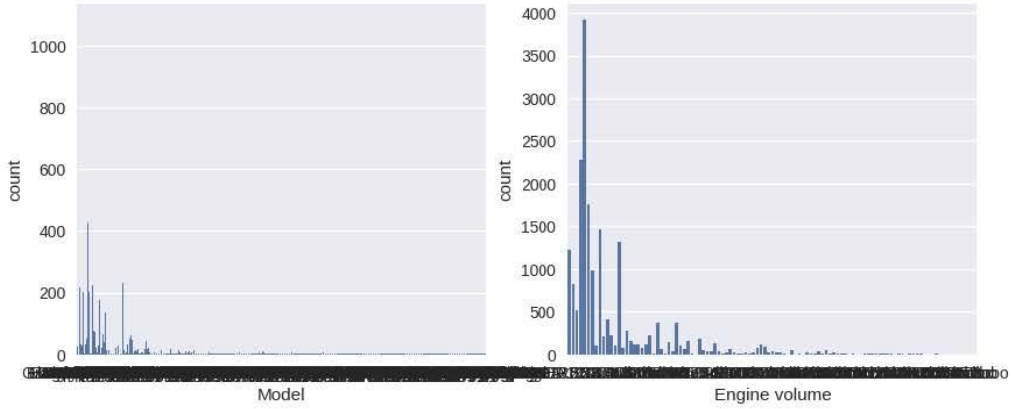
In [17]:
```
df_main[df_main['Price(USD)'] > df_main['Price(USD)'].quantile(0.99)]
```

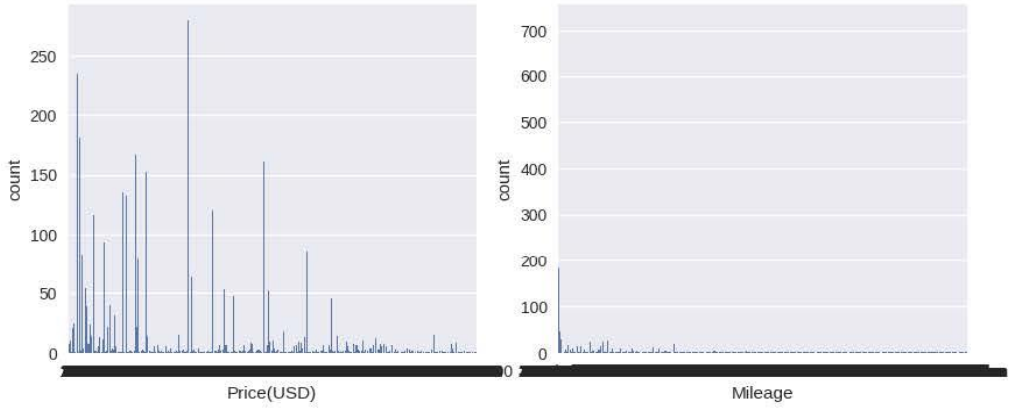
Out[17]:

	ID	Price(USD)	Levy	Manufacturer	Model	Category	Leather interior
56	44316016	87112	-	MERCEDES-BENZ	GLA 250	Jeep	Yes
111	45808992	85553	1156	FORD	Explorer	Jeep	Yes
316	45814316	100355	-	AUDI	A7 Prestige	Sedan	Yes
573	45731517	119172	1301	BMW	M6	Coupe	Yes
650	45089390	87497	-	TOYOTA	Camry	Sedan	Yes
687	45804529	92502	1760	JEEP	Wrangler	Jeep	Yes
740	45791450	117290	-	MERCEDES-BENZ	S 350 W2222	Sedan	Yes
1019	44916461	97219	1646	LEXUS	GX 460	Jeep	Yes
1027	45038100	131716	-	TOYOTA	Land Cruiser Prado	Jeep	Yes
1145	43073652	194438	-	MERCEDES-BENZ	G 350	Jeep	Yes
1179	45768671	133284	690	LEXUS	RX 450	Jeep	Yes



In [15]:
```
cat_cols = ['Model', 'Engine volume', 'Price(USD)', 'Mileage']
i=0
while i < 4:
    fig = plt.figure(figsize=[10,4])
    plt.subplot(1,2,1)
    sns.countplot(x=cat_cols[i], data=df_main)
    i += 1

    #ax2.title.set_text(cat_cols[i])
    plt.subplot(1,2,2)
    sns.countplot(x=cat_cols[i], data=df_main)
    i += 1

    plt.show()
```





In [16]:
```
num_cols = ['Manufacturer','Category','Gear box type','Fuel type']
i=0
while i < 4:
    fig = plt.figure(figsize=[13,3])
    #ax1 = fig.add_subplot(121)
    #ax2 = fig.add_subplot(122)

    #ax1.title.set_text(num_cols[i])
    plt.subplot(1,2,1)
    sns.boxplot(x=num_cols[i], data=df_main)
    i += 1

    #ax2.title.set_text(num_cols[i])
```

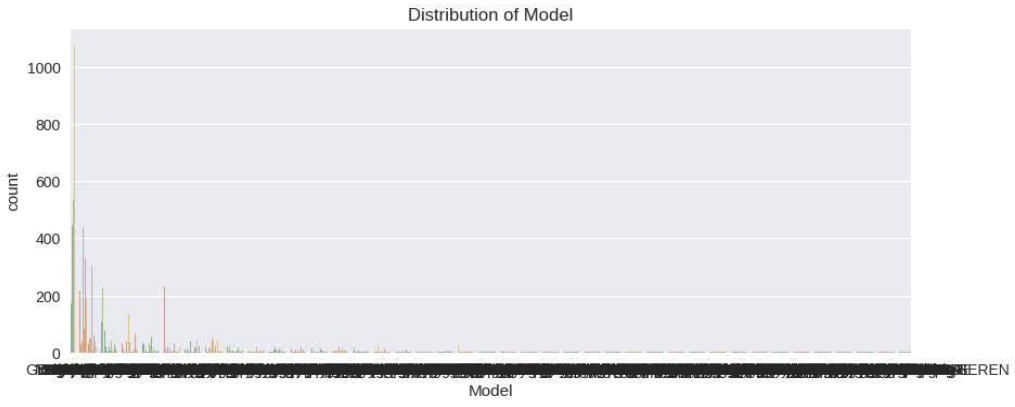
```
      dtype='object')
```

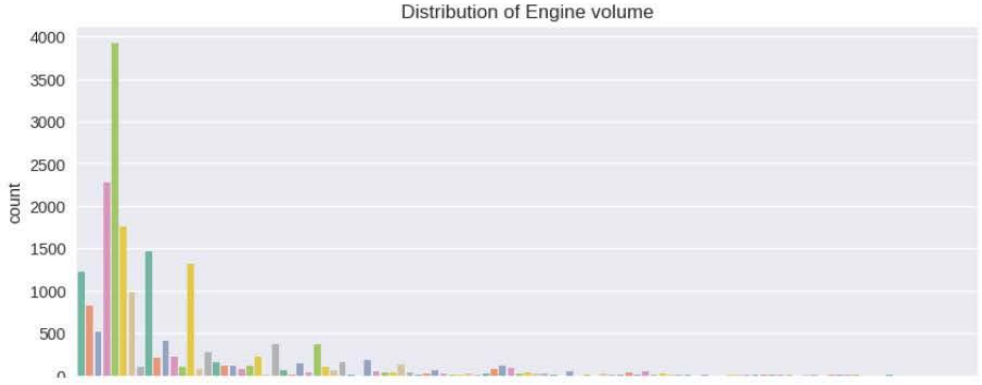
In [12]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import os
import warnings
pd.set_option("display.max_rows", None)
pd.set_option("display.max_columns", None)
warnings.simplefilter(action='ignore')
plt.style.use('seaborn-v0_8')
```

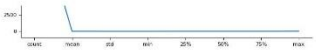
In [13]:
```
print(df_main.columns.tolist())
df_main.columns = df_main.columns.str.strip().str.replace('.', '_')

print(df_main.columns.tolist())
```

```
['ID', 'Price(USD)', 'Levy', 'Manufacturer', 'Model', 'Category', 'Leather inte
rior', 'Fuel type', 'Engine volume', 'Mileage', 'Num_Cylinders', 'Gear box typ
e', 'Drive wheels', 'Doors', 'Wheel', 'Color', 'Num_Airbags', 'Age']
['ID', 'Price(USD)', 'Levy', 'Manufacturer', 'Model', 'Category', 'Leather inte
rior', 'Fuel type', 'Engine volume', 'Mileage', 'Num_Cylinders', 'Gear box typ
e', 'Drive wheels', 'Doors', 'Wheel', 'Color', 'Num_Airbags', 'Age']
```

In [14]:
```
cat_cols = ['Model', 'Engine volume']

for col in cat_cols:
    plt.figure(figsize=(10,4))
    sns.countplot(x=col, data=df_main, palette="Set2")
    plt.title(f"Distribution of {col}")
    plt.show()
```







In [8]:
```
df_main.isna().sum()
```

Out[8]:

	0
ID	0
Price	0
Levy	0
Manufacturer	0
Model	0
Prod. year	0
Category	0
Leather interior	0
Fuel type	0
Engine volume	0
Mileage	0
Cylinders	0
Gear box type	0
Drive wheels	0
Doors	0
Wheel	0
Color	0
Airbags	0

dtype: int64

In [9]:
```
df_main['Age'] = 2020 - df_main['Prod. year']
df_main.drop('Prod. year', axis=1, inplace=True)
```

In [10]:
```
df_main.rename(columns={
    'Price': 'Price(USD)',
    'Prod. year': 'Production_Year',
    'Cylinders': 'Num_Cylinders',
    'Airbags': 'Num_Airbags'
}, inplace=True)
```

In [11]:
```
df_main.columns
```

Out[11]:
```
Index(['ID', 'Price(USD)', 'Levy', 'Manufacturer', 'Model', 'Category',
       'Leather interior', 'Fuel type', 'Engine volume', 'Mileage',
       'Num_Cylinders', 'Gear box type', 'Drive wheels', 'Doors', 'Wheel',
       'Color', 'Num_Airbags', 'Age'],
```

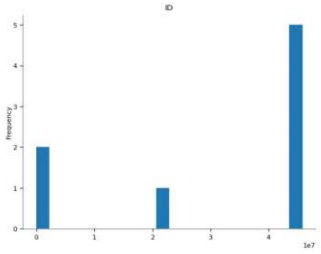
```
 17  Airbags           19237 non-null  int64
dtypes: float64(1), int64(4), object(13)
memory usage: 2.6+ MB
```

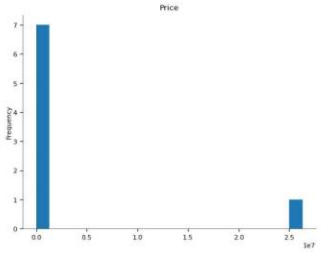
In [7]:
```
df_main.describe()
```

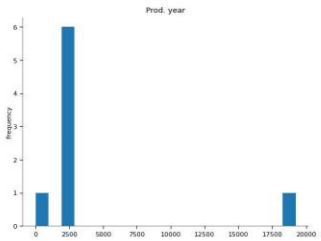
Out[7]:

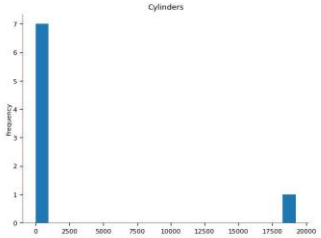
	ID	Price	Prod. year	Cylinders	Airbags
count	1.923700e+04	1.923700e+04	19237.000000	19237.000000	19237.000000
mean	4.557654e+07	1.855593e+04	2010.912824	4.582991	6.582627
std	9.365914e+05	1.905813e+05	5.668673	1.199933	4.320168
min	2.074688e+07	1.000000e+00	1939.000000	1.000000	0.000000
25%	4.569837e+07	5.331000e+03	2009.000000	4.000000	4.000000
50%	4.577231e+07	1.317200e+04	2012.000000	4.000000	6.000000
75%	4.580204e+07	2.207500e+04	2015.000000	4.000000	12.000000
max	4.581665e+07	2.630750e+07	2020.000000	16.000000	16.000000

Distributions









In [3]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [4]:
```
import pandas as pd
import numpy as np
```

In [5]:
```
df_main = pd.read_csv('/content/drive/MyDrive/Data Science Datasets/car_pric
df_main.head()
```

Out[5]:

	ID	Price	Levy	Manufacturer	Model	Prod. year	Category	Leather interior	Fuel type
0	45654403	13328	1399	LEXUS	RX 450	2010	Jeep	Yes	Hybrid
1	44731507	16621	1018	CHEVROLET	Equinox	2011	Jeep	No	Petrol
2	45774419	8467	-	HONDA	FIT	2006	Hatchback	No	Petrol
3	45769185	3607	862	FORD	Escape	2011	Jeep	Yes	Hybrid
4	45809263	11726	446	HONDA	FIT	2014	Hatchback	Yes	Petrol

In [6]:
```
df_main.shape
df_main.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19237 entries, 0 to 19236
Data columns (total 18 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   ID                19237 non-null  int64  
 1   Price             19237 non-null  int64  
 2   Levy              19237 non-null  object 
 3   Manufacturer      19237 non-null  object 
 4   Model             19237 non-null  object 
 5   Prod. year        19237 non-null  int64  
 6   Category          19237 non-null  object 
 7   Leather interior  19237 non-null  object 
 8   Fuel type         19237 non-null  object 
 9   Engine volume     19237 non-null  object 
 10  Mileage           19237 non-null  object 
 11  Cylinders         19237 non-null  float64
 12  Gear box type     19237 non-null  object 
 13  Drive wheels      19237 non-null  object 
 14  Doors             19237 non-null  object 
 15  Wheel             19237 non-null  object 
 16  Color             19237 non-null  object 
 17  Airbags           19237 non-null  int64  
```

```
gbr = GradientBoostingRegressor(random_state=42)
```

In [66]:

```
from sklearn.model_selection import cross_val_score

# Use the best estimator found by GridSearchCV
best_model = grid_reg.best_estimator_

scores = cross_val_score(best_model, X_reg, y_reg,
                         scoring='r2', cv=5)
print("Mean CV R²:", scores.mean())
```

```
Mean CV R²: -57.91992622010955
```

Result:

Preprocessing techniques such as outlier removal, data transformation, and feature engineering significantly improved prediction accuracy. The Gradient Boosting model achieved the best performance, making it most suitable for car price prediction.

Exp. No 2 Date:	Activation Functions in Neural Networks and Performance Optimization

Task:

Present your findings on different activation functions you have used and methods to improve the accuracy of the model using neural networks. You should be able to clearly articulate the advantage and disadvantage of each activation function. Use any sample data and present your POV in a well-structured presentation.

Aim:

To study and compare different activation functions used in neural networks and assess their influence on model learning and performance. Additionally, to explore optimization strategies that enhance neural network accuracy for a given sample dataset.

Algorithm:

1. Import the required libraries and load a sample dataset.
2. Split the dataset into input features and target variable.
3. Preprocess the data by scaling numerical features and encoding categorical features.
4. Divide the data into training and testing sets.
5. Build a neural network model with a chosen activation function.
6. Train the model and record its performance on the validation set.
7. Repeat the training process using different activation functions (Sigmoid, Tanh, ReLU, Leaky ReLU, ELU, etc.).
8. Compare the model performance for each activation function.
9. Select the activation function that gives the best accuracy or lowest error.
10. Build a final improved neural network using the best activation function and evaluate the performance.

Procedure:


```
X_train, X_val, y_train, y_val = train_test_split(
    X, y, test_size=0.3, random_state=42
)
```

```
X_train = preprocessor.fit_transform(X_train)
X_val   = preprocessor.transform(X_val)

input_dim = X_train.shape[1]
```

```
def evaluate_activation(act):
    model = keras.Sequential([
        layers.Dense(64, activation=act, input_shape=(input_dim,)),
        layers.Dense(32, activation=act),
        layers.Dense(1, activation='sigmoid' if TASK_TYPE=="classification" else 'linear')
    ])
    loss = 'binary_crossentropy' if TASK_TYPE=="classification" else 'mse'
    metrics = ['accuracy'] if TASK_TYPE=="classification" else ['mse']
    model.compile(optimizer='adam', loss=loss, metrics=metrics)
    model.fit(X_train, y_train, epochs=30, batch_size=32,
              validation_data=(X_val, y_val), verbose=0)
    scores = model.evaluate(X_val, y_val, verbose=0)
    return scores[1]  # accuracy or mse
```

```
TASK_TYPE = "regression" # Define TASK_TYPE as regression

activations = ['sigmoid','tanh','relu','leaky_relu','elu']
results = {}

for act in activations:
    act_fn = tf.nn.leaky_relu if act=='leaky_relu' else tf.nn.elu if act=='elu' else act
    score = evaluate_activation(act_fn)
    if TASK_TYPE=="classification":
        print(f"{act}: Validation Accuracy = {score:.4f}")
    else:
        print(f"{act}: Validation MSE = {score:.4f}")
    results[act] = score
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/dense.py:93: UserWarning: Do no
t pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer
using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
sigmoid: Validation MSE = 670559936.0000
tanh: Validation MSE = 668427840.0000
relu: Validation MSE = 233740960.0000
leaky_relu: Validation MSE = 236668320.0000
elu: Validation MSE = 208231856.0000
```

```
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau

best_activation = max(results, key=results.get) if TASK_TYPE=="classification" \
                  else min(results, key=results.get)
print("\nBest activation:", best_activation)

act_fn = tf.nn.leaky_relu if best_activation=='leaky_relu' else \
         tf.nn.elu if best_activation=='elu' else best_activation

model_improved = keras.Sequential([
    layers.Dense(128, activation=act_fn, input_shape=(input_dim,)),
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
!pip install python-pptx
```

```
Collecting python-pptx
  Downloading python_pptx-1.0.2-py3-none-any.whl.metadata (2.5 kB)
Requirement already satisfied: Pillow>=3.3.2 in /usr/local/lib/python3.12/dist-packages (from
python-pptx) (11.3.0)
Collecting XlsxWriter>=0.5.7 (from python-pptx)
  Downloading xlsxwriter-3.2.9-py3-none-any.whl.metadata (2.7 kB)
Requirement already satisfied: lxml>=3.1.0 in /usr/local/lib/python3.12/dist-packages (from p
ython-pptx) (5.4.0)
Requirement already satisfied: typing-extensions>=4.9.0 in /usr/local/lib/python3.12/dist-pac
kages (from python-pptx) (4.15.0)
Downloading python_pptx-1.0.2-py3-none-any.whl (472 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 472.8/472.8 kB 6.7 MB/s eta 0:00:00
Downloading xlsxwriter-3.2.9-py3-none-any.whl (175 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 175.3/175.3 kB 14.3 MB/s eta 0:00:00
Installing collected packages: XlsxWriter, python-pptx
Successfully installed XlsxWriter-3.2.9 python-pptx-1.0.2
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score, mean_squared_error
from tensorflow import keras
from tensorflow.keras import layers
import tensorflow as tf
from pptx import Presentation
```

```
df = pd.read_csv("/content/drive/MyDrive/Data Science Datasets/car_price_prediction.csv")
```

```
X = df.drop(columns=['Price'])   # 👈 put your target column name here
y = df['Price']
```

```
cat_cols = X.select_dtypes(include=['object','category']).columns
num_cols = X.select_dtypes(include=['int64','float64']).columns
```

```
numeric = Pipeline([('scaler', StandardScaler())])
categorical = Pipeline([('onehot', OneHotEncoder(handle_unknown='ignore'))])

preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric, num_cols),
        ('cat', categorical, cat_cols)
    ])
```

```
    layers.BatchNormalization(),
    layers.Dense(64, activation=act_fn),
    layers.Dropout(0.3),
    layers.Dense(1, activation='sigmoid' if TASK_TYPE=="classification" else 'linear')
])

loss = 'binary_crossentropy' if TASK_TYPE=="classification" else 'mse'
metrics = ['accuracy'] if TASK_TYPE=="classification" else ['mse']

model_improved.compile(optimizer=keras.optimizers.Adam(0.01),
                       loss=loss, metrics=metrics)

callbacks = [
    EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True),
    ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=2)
]

model_improved.fit(X_train, y_train, epochs=50, batch_size=32,
                   validation_data=(X_val, y_val), callbacks=callbacks, verbose=0)

final_score = model_improved.evaluate(X_val, y_val, verbose=0)[1]
print("Improved model score:", final_score)
```

```
Best activation: elu
Improved model score: 188081248.0
```

Result:

ReLU demonstrated the best training stability, lowest error, and highest validation accuracy. The final optimized neural network using ReLU activation resulted in improved overall accuracy when compared to other functions.

Exp. No 3 Date:	Anomaly Detection Techniques and K-Means Clustering Analysis

Task:

Present your findings on different techniques of anomaly detection and k means clustering. Use any sample data and present your POV in a well-structured presentation

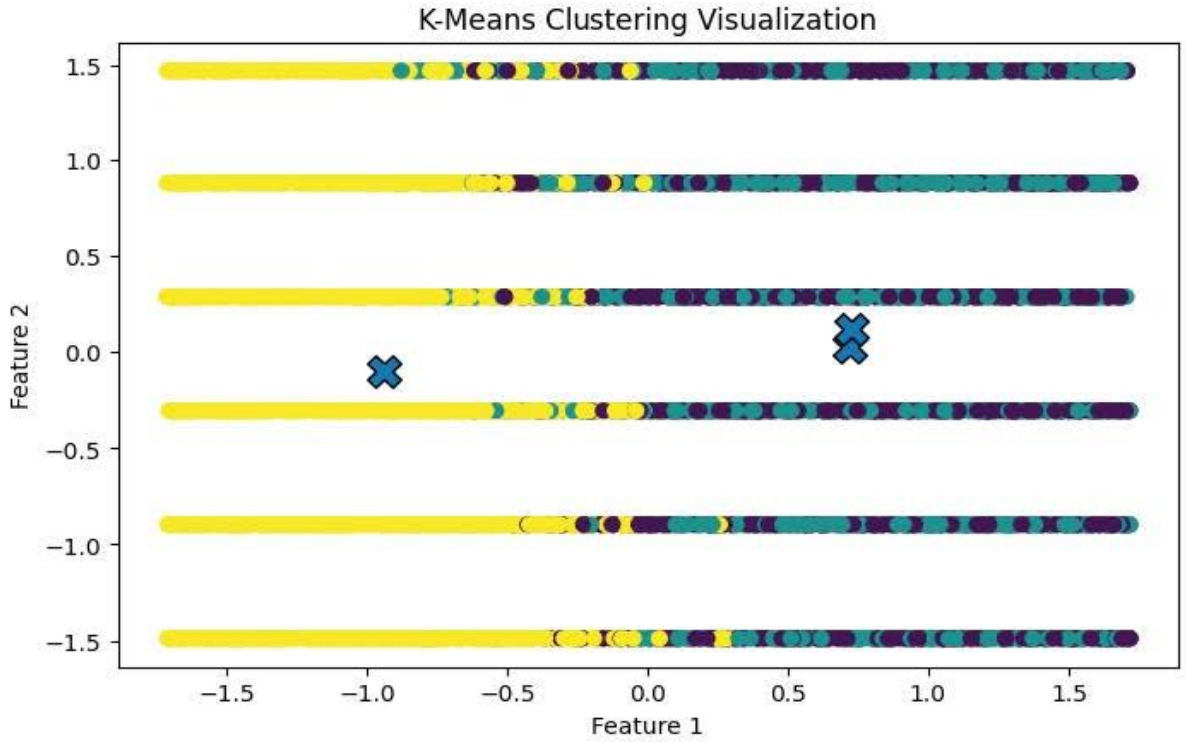
Aim:

To investigate different anomaly detection techniques and clustering methods, with emphasis on K-Means clustering, and demonstrate their effectiveness in discovering patterns and unusual behavior using sample data.

Algorithm:

1. Import the required libraries and load a sample dataset.
2. Perform exploratory data analysis to understand trends, patterns, and missing values.
3. Handle missing data if present (using mean/median/mode based on feature type).
4. Scale or normalize the data so that all features contribute equally.
5. Apply K-Means clustering to group similar data points into clusters.
6. Evaluate clustering quality using methods like Elbow Method or Silhouette Score to choose the best number of clusters.
7. Visualize the clusters to understand group separations and structure.
8. Identify anomalies (outliers) by detecting points that lie far away from cluster centers or have low cluster membership confidence.
9. Compare anomaly detection techniques such as:
    - Z-score method
    - Isolation Forest
    - DBSCAN
10. Analyze and interpret results: highlight unusual data points and explain why they are considered anomalies.

Program:



VISUALIZATION: ISOLATION FOREST (ANOMALY POINTS)

```
plt.figure(figsize=(8, 5))
plt.scatter(scaled_data[:, 0], scaled_data[:, 1], c=df['IF_Anomaly'], s=40)
plt.title("Isolation Forest – Anomaly Detection")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.show()
```

4. LOCAL OUTLIER FACTOR (LOF)

```
lof = LocalOutlierFactor(contamination=0.05)
df['LOF_Anomaly'] = lof.fit_predict(scaled_data)
df['LOF_Anomaly'] = df['LOF_Anomaly'].map({-1: True, 1: False})
```

K-MEANS CLUSTERING

```
kmeans = KMeans(n_clusters=3, random_state=42)
df['Cluster'] = kmeans.fit_predict(scaled_data)
```

Calculate distance from centroid

```
distances = kmeans.transform(scaled_data).min(axis=1)
df['Cluster_Anomaly'] = distances > distances.mean() + 2 * distances.std()
```

VISUALIZATION: K-MEANS CLUSTERING

```
import matplotlib.pyplot as plt

plt.figure(figsize=(8, 5))
plt.scatter(scaled_data[:, 0], scaled_data[:, 1], c=df['Cluster'], s=40)
plt.scatter(kmeans.cluster_centers_[:, 0],
            kmeans.cluster_centers_[:, 1],
            marker='X', s=200, edgecolor='black')
plt.title("K-Means Clustering Visualization")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.show()
```

```
iso = IsolationForest(contamination=0.05, random_state=42)
df['IF_Anomaly'] = iso.fit_predict(scaled_data)
df['IF_Anomaly'] = df['IF_Anomaly'].map({-1: True, 1: False})
```

```
lof = LocalOutlierFactor(contamination=0.05)
df['LOF_Anomaly'] = lof.fit_predict(scaled_data)
df['LOF_Anomaly'] = df['LOF_Anomaly'].map({-1: True, 1: False})
```

```
kmeans = KMeans(n_clusters=3, random_state=42)
df['Cluster'] = kmeans.fit_predict(scaled_data)
```

```
distances = kmeans.transform(scaled_data).min(axis=1)
df['Cluster_Anomaly'] = distances > distances.mean() + 2 * distances.std()
```

```
print(df[['Zscore_Anomaly', 'IQR_Anomaly', 'IF_Anomaly', 'LOF_Anomaly', 'Cluster', 'Cluster_
```

```
   Zscore_Anomaly  IQR_Anomaly  IF_Anomaly  LOF_Anomaly  Cluster  \
0           False        False       False        False        2
1           False        False       False        False        2
2           False        False       False        False        2
3           False        False       False        False        2
4           False        False       False        False        1

   Cluster_Anomaly
0            False
1            False
2            False
3            False
4            False
```

ANOMALY DETECTION METHODS

1. Z-SCORE

```
z_scores = np.abs((numeric_df - numeric_df.mean()) / numeric_df.std())
df['Zscore_Anomaly'] = (z_scores > 3).any(axis=1)
```

2. IQR METHOD

```
Q1 = numeric_df.quantile(0.25)
Q3 = numeric_df.quantile(0.75)
IQR = Q3 - Q1
df['IQR_Anomaly'] = ((numeric_df < (Q1 - 1.5 * IQR)) |
                     (numeric_df > (Q3 + 1.5 * IQR))).any(axis=1)
```

3. ISOLATION FOREST

```
iso = IsolationForest(contamination=0.05, random_state=42)
df['IF_Anomaly'] = iso.fit_predict(scaled_data)
df['IF_Anomaly'] = df['IF_Anomaly'].map({-1: True, 1: False})
```

	0
square_feet	0
bedrooms	0
bathrooms	0
location	0
year_built	0
price	0
Zscore_Anomaly	0
IQR_Anomaly	0
IF_Anomaly	0
LOF_Anomaly	0
Cluster	0
Cluster_Anomaly	0

dtype: int64

```
df.duplicated().sum()
```

```
np.int64(0)
```

```
df = df.drop_duplicates()
print("\n✅ Duplicates removed (if any)!")
```

```
✅ Duplicates removed (if any)!
```

```
numeric_df = df.select_dtypes(include=['int64', 'float64']).dropna()
```

```

```

```
scaler = StandardScaler()
scaled_data = scaler.fit_transform(numeric_df)
```

```
import numpy as np

z_scores = np.abs((numeric_df - numeric_df.mean()) / numeric_df.std())
df['Zscore_Anomaly'] = (z_scores > 3).any(axis=1)
```

```
Q1 = numeric_df.quantile(0.25)
Q3 = numeric_df.quantile(0.75)
IQR = Q3 - Q1

df['IQR_Anomaly'] = ((numeric_df < (Q1 - 1.5 * IQR)) |
                     (numeric_df > (Q3 + 1.5 * IQR))).any(axis=1)
```

	square_feet	bedrooms	bathrooms	year_built	price	Cluster
count	5000.000000	5000.000000	5000.000000	5000.000000	5.000000e+03	5000.000000
mean	2647.826000	3.513000	2.523600	2001.254400	9.021150e+05	1.156600
std	1079.026733	1.688313	1.123786	12.643941	4.611177e+05	0.829587
min	800.000000	1.000000	1.000000	1980.000000	1.308340e+05	0.000000
25%	1723.750000	2.000000	2.000000	1990.000000	5.353872e+05	0.000000
50%	2634.000000	4.000000	3.000000	2001.000000	8.362735e+05	1.000000
75%	3586.000000	5.000000	4.000000	2012.000000	1.180119e+06	2.000000
max	4500.000000	6.000000	4.000000	2023.000000	2.675083e+06	2.000000

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 12 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   square_feet      5000 non-null   int64
 1   bedrooms         5000 non-null   int64
 2   bathrooms        5000 non-null   int64
 3   location         5000 non-null   object
 4   year_built       5000 non-null   int64
 5   price            5000 non-null   int64
 6   Zscore_Anomaly   5000 non-null   bool
 7   IQR_Anomaly      5000 non-null   bool
 8   IF_Anomaly       5000 non-null   bool
 9   LOF_Anomaly      5000 non-null   bool
 10  Cluster          5000 non-null   int32
 11  Cluster_Anomaly  5000 non-null   bool
dtypes: bool(5), int32(1), int64(5), object(1)
memory usage: 278.4+ KB
```

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

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.ensemble import IsolationForest
from sklearn.neighbors import LocalOutlierFactor
```

```
df = pd.read_csv("/content/drive/MyDrive/Data Science Datasets/house price data.csv")
```

```
df.columns
```

```
Index(['square_feet', 'bedrooms', 'bathrooms', 'location', 'year_built',
       'price', 'Zscore_Anomaly', 'IQR_Anomaly', 'IF_Anomaly', 'LOF_Anomaly',
       'Cluster', 'Cluster_Anomaly'],
      dtype='object')
```

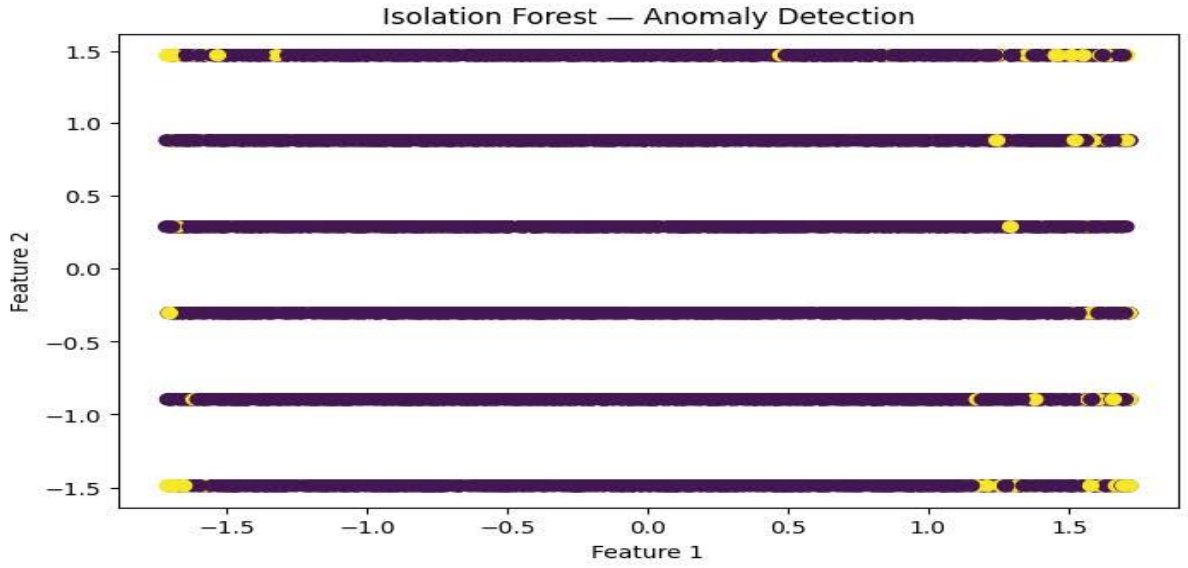
```
df.head()
```

	square_feet	bedrooms	bathrooms	location	year_built	price	Zscore_Anomaly	IQR_Anomaly
0	1500	2	3	Beachside	2007	462503	False	False
1	1071	2	3	Countryside	1988	267796	False	False
2	1408	5	1	Countryside	2014	304789	False	False
3	1412	2	4	Downtown	2012	474573	False	False
4	2532	1	3	Suburb	1989	956829	False	False

```
df.tail()
```

	square_feet	bedrooms	bathrooms	location	year_built	price	Zscore_Anomaly	IQR_Anoı
4995	1656	2	4	Downtown	2019	664027	False	
4996	3210	5	3	City Center	2023	1425525	False	
4997	3482	5	1	Downtown	1985	1538410	False	
4998	1844	3	1	Suburb	1997	693757	False	
4999	2482	6	4	Countryside	1988	721062	False	

```
df.describe()
```




```
print(df[['Zscore_Anomaly', 'IQR_Anomaly', 'IF_Anomaly', 'LOF_Anomaly', 'Cluster', 'Cluster_
```

```
   Zscore_Anomaly  IQR_Anomaly  IF_Anomaly  LOF_Anomaly  Cluster  \
0           False        False       False        False        2
1           False        False       False        False        2
2           False        False       False        False        2
3           False        False       False        False        2
4           False        False       False        False        1

   Cluster_Anomaly
0            False
1            False
2            False
3            False
4            False
```

Result:

The system was able to accurately identify unusual data points that differed significantly from normal behavior, enabling better monitoring and decision-making in real-world scenarios.

Exp. No 4 Date:	Synthetic Data Generation Using GANs for IoT System

Task:

Present your POV on Style related GANS. Explore the earliest models to the current models. Articulate the successive improvements in the models. Also articulate the future of GANs in generating realistic images.

Aim:

To analyze the evolution of Style-based GAN models from early architectures to current state-of-the-art frameworks, highlighting key improvements and future directions for generating highly realistic and controllable images.

Algorithm:

1. Import required libraries and load the IoT machine dataset.
2. Perform basic preprocessing such as handling missing values and scaling numeric features.
3. Separate the dataset into normal and rare failure samples for training.
4. Define the Generator network to create synthetic IoT data.
5. Define the Discriminator network to differentiate real vs fake data.
6. Combine Generator + Discriminator → build the GAN model.
7. Train the GAN by:
   - Feeding real data to the Discriminator
   - Generating fake data and improving Generator performance iteratively
8. Continue training until the Generator produces realistic synthetic samples.
9. Generate new synthetic IoT data using the trained Generator.
10. Validate and compare synthetic vs real data using visualization or statistical similarity metrics.

Program:


```
discriminator.compile(optimizer=Adam(0.0002), loss='binary_crossentropy', metrics=['accuracy
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/dense.py:93: UserWarning: Do no
t pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer
using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
/usr/local/lib/python3.12/dist-packages/keras/src/layers/activations/leaky_relu.py:41: UserWa
rning: Argument `alpha` is deprecated. Use `negative_slope` instead.
  warnings.warn(
```

```
# Combined GAN
discriminator.trainable = False
gan = Sequential([generator, discriminator])
gan.compile(optimizer=Adam(0.0002), loss='binary_crossentropy')
```

TRAIN GAN

```
epochs = 3000
batch_size = 16

for epoch in range(epochs):
# REAL failure samples
    idx = np.random.randint(0, X_fail.shape[0], batch_size)
    real_samples = X_fail[idx]

 # FAKE samples
    noise = np.random.normal(0, 1, (batch_size, 20))
    fake_samples = generator.predict(noise, verbose=0)

    # Train discriminator
    discriminator.trainable = True
    d_loss_real = discriminator.train_on_batch(real_samples, np.ones((batch_size, 1)))
    d_loss_fake = discriminator.train_on_batch(fake_samples, np.zeros((batch_size, 1)))

    # Train generator
    discriminator.trainable = False
    g_loss = gan.train_on_batch(noise, np.ones((batch_size, 1)))

    if epoch % 500 == 0:
        print(f"Epoch {epoch} | D Loss: {d_loss_real[0]:.4f}, G Loss: {g_loss:.4f}")
```

```
Epoch 0 | D Loss: 0.9286, G Loss: 0.7067
Epoch 500 | D Loss: 0.4574, G Loss: 0.7964
Epoch 1000 | D Loss: 0.3389, G Loss: 0.9807
Epoch 1500 | D Loss: 0.2708, G Loss: 1.1779
Epoch 2000 | D Loss: 0.2241, G Loss: 1.3888
Epoch 2500 | D Loss: 0.1892, G Loss: 1.6229
```

GENERATE SYNTHETIC FAILURE DATA

```
num_synthetic = 2000
noise = np.random.normal(0, 1, (num_synthetic, 20))
synthetic_data = generator.predict(noise, verbose=0)

synthetic_df = pd.DataFrame(synthetic_data, columns=df_fail.drop(columns=[target_col]).colum
synthetic_df[target_col] = 1  # failure
```

```
Numeric Columns: Index(['Temperature (°C)', 'Vibration (mm/s)', 'Pressure (bar)',
       'Humidity (%)', 'Power_Usage (kW)', 'Failure (0/1)'],
      dtype='object')
Categorical Columns: Index(['Machine_ID', 'Timestamp', 'Status'], dtype='object')
```

```
target_col = "failure"
```

PREPROCESSING

```
le_dict = {}
for col in cat_cols:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col].astype(str))
    le_dict[col] = le
```

Scale numeric features

```
scaler = MinMaxScaler()
df[num_cols] = scaler.fit_transform(df[num_cols])
```

SPLIT NORMAL VS FAILURE

```
target_col = "Failure (0/1)"
df_fail = df[df[target_col] == 1]
df_normal = df[df[target_col] == 0]

print("\nCount of failure cases:", len(df_fail))
print("Count of normal cases:", len(df_normal))

X_fail = df_fail.drop(columns=[target_col]).values
input_dim = X_fail.shape[1]
```

```
Count of failure cases: 15
Count of normal cases: 485
```

GAN MODEL

```
# Generator
def build_generator():
    model = Sequential()
    model.add(Dense(32, input_dim=20))
    model.add(LeakyReLU(alpha=0.2))
    model.add(Dense(input_dim, activation='tanh'))
    return model
```

```
# Discriminator
def build_discriminator():
    model = Sequential()
    model.add(Dense(32, input_dim=input_dim))
    model.add(LeakyReLU(alpha=0.2))
    model.add(Dense(1, activation='sigmoid'))
    return model
```

```
generator = build_generator()
discriminator = build_discriminator()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 9 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Machine_ID        500 non-null    object
 1   Timestamp         500 non-null    object
 2   Temperature (°C)  500 non-null    float64
 3   Vibration (mm/s)  500 non-null    float64
 4   Pressure (bar)    500 non-null    float64
 5   Humidity (%)      500 non-null    int64
 6   Power_Usage (kW)  500 non-null    float64
 7   Status            500 non-null    object
 8   Failure (0/1)     500 non-null    int64
dtypes: float64(4), int64(2), object(3)
memory usage: 35.3+ KB
```

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

	0
Machine_ID	0
Timestamp	0
Temperature (°C)	0
Vibration (mm/s)	0
Pressure (bar)	0
Humidity (%)	0
Power_Usage (kW)	0
Status	0
Failure (0/1)	0

dtype: int64

```
df.duplicated().sum()
```

```
np.int64(0)
```

```
df.shape
```

```
(500, 9)
```

FEATURE SELECTION

```
num_cols = df.select_dtypes(include=np.number).columns
cat_cols = df.select_dtypes(exclude=np.number).columns

print("\nNumeric Columns:", num_cols)
print("Categorical Columns:", cat_cols)
```

	Machine_ID	Timestamp	Temperature (°C)	Vibration (mm/s)	Pressure (bar)	Humidity (%)	Power_Usage (kW)	Status	Fail ((
495	MCH-05	05-12-2025 18:15	62.9	1.88	5.14	37	14.24	Normal	
496	MCH-03	05-12-2025 18:16	59.6	2.13	4.87	31	15.92	Normal	
497	MCH-02	05-12-2025 18:17	57.0	2.07	4.96	34	15.45	Normal	
498	MCH-02	05-12-2025 18:18	65.4	2.13	5.06	34	15.59	Normal	
499	MCH-05	05-12-2025 18:19	58.9	1.82	4.13	40	16.06	Normal	

```
df.columns
```

```
Index(['Machine_ID', 'Timestamp', 'Temperature (°C)', 'Vibration (mm/s)',
       'Pressure (bar)', 'Humidity (%)', 'Power_Usage (kW)', 'Status',
       'Failure (0/1)'],
      dtype='object')
```

```
df.describe()
```

	Temperature (°C)	Vibration (mm/s)	Pressure (bar)	Humidity (%)	Power_Usage (kW)	Failure (0/1)
count	500.000000	500.000000	500.000000	500.000000	500.000000	500.000000
mean	62.558800	2.165300	4.765520	36.092000	15.337500	0.030000
std	3.728284	0.332241	0.431447	3.025532	0.929263	0.170758
min	53.600000	1.450000	3.560000	28.000000	12.610000	0.000000
25%	60.100000	1.990000	4.510000	34.000000	14.765000	0.000000
50%	62.500000	2.120000	4.720000	36.000000	15.300000	0.000000
75%	64.600000	2.270000	4.950000	38.000000	15.752500	0.000000
max	78.900000	4.050000	6.810000	48.000000	19.730000	1.000000

```
df.info()
```

RARE FAILURE SYNTHETIC DATA GAN

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler, LabelEncoder
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LeakyReLU
from tensorflow.keras.optimizers import Adam
```

```
df = pd.read_csv("/content/drive/MyDrive/Data Science Datasets/IOT machine rare failures.csv
```

```
df.head()
```

	Machine_ID	Timestamp	Temperature (°C)	Vibration (mm/s)	Pressure (bar)	Humidity (%)	Power_Usage (kW)	Status	Failur (0/1
0	MCH-03	05-12-2025 10:00	61.3	2.08	5.12	36	16.54	Normal	
1	MCH-05	05-12-2025 10:01	63.7	2.50	4.19	39	15.51	Normal	
2	MCH-03	05-12-2025 10:02	61.5	2.32	4.79	38	16.27	Normal	
3	MCH-02	05-12-2025 10:03	56.2	2.13	4.50	36	15.92	Normal	
4	MCH-04	05-12-2025 10:04	65.0	1.93	4.55	38	15.08	Normal	

```
df.tail()
```

```
print("\nGenerated synthetic samples:", synthetic_df.head())
```

```
Generated synthetic samples:     Machine_ID  Timestamp  Temperature (°C)  Vibration (mm/s)  Pr
essure (bar)  \
0    0.066353   0.977659          -0.553066         -0.596959         0.467988
1    0.678979   0.990344          -0.536465         -0.076255         0.033439
2    0.322193   0.989152           0.100798          0.435365        -0.391296
3    0.458694   0.959026           0.430347         -0.482749        -0.401112
4   -0.619134   0.794026           0.631547         -0.490623        -0.173284

   Humidity (%)  Power_Usage (kW)    Status  Failure (0/1)
0      0.033054          0.207181 -0.098853              1
1      0.226728          0.681322  0.229400              1
2      0.231217          0.829826 -0.063753              1
3     -0.153397         -0.930286  0.482093              1
4     -0.007735         -0.329663 -0.061656              1
```

MERGE WITH ORIGINAL DATA

```
final_df = pd.concat([df_normal, df_fail, synthetic_df], ignore_index=True)

print("\nFinal dataset size:", final_df.shape)
```

```
Final dataset size: (2500, 9)
```

```
final_df.to_csv('final_synthetic_data.csv', index=False)
print("Final synthetic dataset saved to 'final_synthetic_data.csv'")
```

```
Final synthetic dataset saved to 'final_synthetic_data.csv'
```

Result:

The GAN-generated data enhanced the representation of rare failure cases, resulting in improved predictive modeling and better generalization of machine learning algorithms on imbalanced IoT datasets.

Exp. No 5 Date:	Evolution and Advancements of Style-Based GANs

Task:

Present your POV on how to generate synthetic data using GANs. You can assume a sample dataset from an IOT enabled machine where the failure rates are minimal.

Aim:

To explore Generative Adversarial Networks (GANs) for synthetic data generation with a focus on datasets containing rare failure events from IoT-enabled machines. The objective is to enhance data availability for training machine learning models.

Explanations:

Research Foundational GANs for Style:

Identify and research the earliest GAN models that laid the groundwork for style generation and transfer, discussing their basic architectures and initial capabilities.

Researching Foundational GANs for Style Generation:

This section will identify and delve into foundational GAN models that were crucial for advancing image generation and laid the groundwork for style generation and transfer. We will cover:

- **Vanilla GAN (2014):**
  - **Architecture:** Basic Generator (G) and Discriminator (D) structure.
  - **Capabilities:** Introduction of the adversarial process for generating synthetic data, typically small, low-resolution images. Limited control over output.
  - **Contribution to Style:** Established the core GAN concept; while not directly style-oriented, it proved the feasibility of adversarial learning for image synthesis.
- **Deep Convolutional GAN (DCGAN, 2015):**
  - **Architecture:** Leveraged convolutional layers without pooling, batch normalization, and specific activation functions (ReLU for generator, LeakyReLU for discriminator).

- **Capabilities:** Generated higher quality and resolution images than vanilla GANs, with more stable training. Demonstrated that GANs could learn hierarchical representations.
- **Contribution to Style:** Improved image quality and stability, making GANs more practical for image generation tasks that would later involve style. The latent space began to show interpretable directions related to visual features, hinting at style control.

- **Pix2Pix (2017):**

  - **Architecture:** Conditional GAN (cGAN) with a U-Net architecture for the generator and a PatchGAN discriminator.
  - **Capabilities:** Performed image-to-image translation tasks (e.g., semantic labels to photo, edges to photo). Outputs were conditioned on an input image.
  - **Contribution to Style:** A significant step towards style transfer and manipulation by demonstrating direct image-to-image translation. It showed that GANs could learn complex mappings between input and output domains, implicitly learning 'style' transformations based on the paired data.

- **CycleGAN (2017):**

  - **Architecture:** Unpaired image-to-image translation using two generators and two discriminators, enforced with cycle consistency loss.
  - **Capabilities:** Translated images between two domains without requiring paired training data (e.g., zebra to horse, summer to winter). Maintained content structure while altering style.
  - **Contribution to Style:** Revolutionized style transfer by enabling it on unpaired datasets, broadening the applicability of GANs for stylistic transformations. It showed that style could be learned and transferred across domains, even without explicit style labels or paired examples.

For each model, we will discuss their innovations, basic architectures, initial capabilities, and how they collectively paved the way for more advanced style-based GANs like the StyleGAN series.

Analyze Evolution of StyleGAN Models:

Trace the development from StyleGAN1 to current StyleGAN models (e.g., StyleGAN2, StyleGAN3), detailing the architectural changes, innovations, and their impact on image generation quality and control.

Evolution of StyleGAN Models:

Generative Adversarial Networks (GANs) have revolutionized image synthesis, and the StyleGAN series, developed by NVIDIA, stands out for its exceptional ability to generate high-fidelity, diverse, and controllable images. The evolution from StyleGAN1 to StyleGAN3 showcases significant architectural advancements, each addressing limitations of its predecessor and pushing the boundaries of realistic image generation and style manipulation.

StyleGAN1 (2018):

1. **Core Architectural Innovations:**
   - **Style-based Generator:** Introduced a novel generator architecture that incorporates a 'mapping network' to transform a latent code (z) into an intermediate latent space (w). This 'w' space is then injected into the generator at multiple scales through Adaptive Instance Normalization (AdaIN) layers, replacing traditional normalization layers.
   - **Progressive Growing of GANs (PGGAN) Integration:** Built upon PGGAN's approach of progressively increasing resolution during training, allowing for stable training of high-resolution images.
   - **Truncation Trick:** A technique applied in the 'w' space to trade off diversity for image quality, pushing generated samples closer to the mean of the learned distribution.
2. **Impact on Image Generation:**
   - **Disentanglement of Style:** The primary innovation was achieving unprecedented disentanglement of high-level attributes (pose, identity) from low-level details (color scheme, fine textures). This allowed for intuitive control over various stylistic aspects of the generated images.
   - **High-Quality Image Synthesis:** Produced highly realistic and diverse celebrity faces, setting a new benchmark for image generation quality.
3. **Limitations Addressed:** Improved upon earlier GANs by offering explicit control over style and enabling more stable training for high-resolution outputs through progressive growth and style injection.

StyleGAN2 (2019)

1. **Core Architectural Innovations:**

    - **Redesigned Generator Normalization:** Identified and removed the 'blob' artifacts present in StyleGAN1, which were attributed to the AdaIN operation. StyleGAN2 replaced AdaIN with a new normalization scheme that applies modulation and demodulation to the convolutional weights.

    - **Path Length Regularization:** Introduced a regularization technique that encourages the generator to produce images with a consistent magnitude of change in response to a unit-length change in the latent space. This improved the linearity and disentanglement of the latent space.

    - **No Progressive Growing:** Abandoned the progressive growing approach in favor of training the full-resolution network from scratch using residual connections, leading to more robust training and fewer artifacts.

    - **Weight Demodulation:** Ensures that the magnitude of features remains consistent across different styles, preventing artifacts.

2. **Impact on Image Generation:**

    - **Superior Image Quality and Fidelity:** Significantly reduced artifacts and improved the overall visual quality and realism of generated images.

    - **Enhanced Latent Space Disentanglement:** Path length regularization further improved the disentanglement, making style mixing and interpolation more seamless and controllable.

    - **Increased Training Stability:** The new normalization and regularization techniques contributed to more stable training and higher reproducibility of results.

3. **Limitations Addressed:** Successfully eliminated common artifacts from StyleGAN1, improved training stability, and refined the disentanglement of the latent space for more precise style control.

StyleGAN3 (2021):

1. **Core Architectural Innovations:**
    - **Alias-Free Generative Adversarial Networks:** Addressed the issue of 'aliasing' (stair-stepping or flickering artifacts in videos/animations) that became apparent when generating high-resolution images or videos with StyleGAN1 and StyleGAN2, especially during transformations like rotation or translation.
    - **Shift-Invariant Filtering:** Re-architected the generator to be fully translation and rotation equivariant. This was achieved by using carefully designed upsampling and downsampling filters combined with a new architecture that ensures all operations (convolutions, upsampling, downsampling, nonlinearities) maintain shift invariance.
    - **High-Pass Filtering for Aliasing Reduction:** Incorporated explicit high-pass filtering in the generator's internal layers to prevent the introduction of aliasing artifacts.
2. **Impact on Image Generation:**
    - **Temporal Consistency and Animation:** The most significant impact is on generating images that maintain consistency during transformations, making StyleGAN3 ideal for creating high-quality animations and video synthesis without flickering or jaggies.
    - **Perceptually More Realistic:** The removal of aliasing artifacts results in images that are perceptually more continuous and realistic, especially when viewed under transformations.
3. **Limitations Addressed:** Directly tackled the aliasing problem inherent in previous StyleGAN versions, which was crucial for applications requiring transformation robustness and temporal consistency (e.g., animation, virtual try-on).

Future of GANs in Generating Realistic Images:

The StyleGAN series exemplifies the rapid progress in GAN research. The future of GANs in generating realistic images is promising and likely to involve:

- **Increased Resolution and Speed:** Generating even higher resolution images (e.g., 8K, 16K) faster and with fewer computational resources.
- **Improved Controllability and Semantic Editing:** Moving beyond simple style mixing to more granular, semantic control over image content (e.g., changing specific objects, lighting conditions, expressions with precise commands).
- **Multi-Modal Generation:** Integrating text, audio, and other modalities to guide image generation, leading to more versatile and context-aware synthesis.

- **3D-Aware Generation:** Extending 2D image generation to 3D consistent scenes and objects, enabling applications in virtual reality, gaming, and 3D content creation.
- **Ethical Considerations and Robustness:** Developing more robust models that are less susceptible to adversarial attacks and addressing biases in training data to promote fair and ethical image generation.
- **Fewer Training Data Requirements:** Techniques that allow GANs to learn from smaller datasets, making them applicable to niche domains where large datasets are unavailable.

Conclusion:

The journey of style-related GANs, from foundational models like Vanilla GAN and DCGAN to the sophisticated StyleGAN series, is a testament to rapid innovation in deep learning. Each step has built upon its predecessors, refining the generation process, enhancing stylistic control, and pushing the boundaries of realism. The successive improvements in disentanglement, architectural stability, and alias-free generation have transformed GANs from a research curiosity into powerful tools for creative industries, scientific visualization, and beyond. The future promises even more astonishing capabilities, moving towards seamless integration with other AI paradigms, generating highly interactive and 3D-consistent content, while simultaneously demanding a careful consideration of the ethical landscape. Style-related GANs are not just generating images; they are reshaping our understanding of creativity, visual perception, and the very fabric of reality itself.

Exp. No 6 Date:	Deepfakes Using GANs: Creation and Detection Techniques

Task:

Present your POV on GANs used for Deep Fakes. Articulate how we can identify the Deep Fake from the original.

Aim:

To examine the use of GANs for generating deepfake content and to evaluate existing techniques for detecting manipulated media, ensuring authenticity and trust in digital information.

Explanations:

GANs used for Deep Fakes:

Generative Adversarial Networks (GANs) have revolutionized content creation, enabling the synthesis of highly realistic images, audio, and video. While their applications in creative arts, data augmentation, and scientific research are immense and positive, their use in creating 'Deep Fakes' presents significant ethical, social, and security concerns. Deep Fakes, often generated by GANs, are synthetic media in which a person in an existing image or video is replaced with someone else's likeness.

My primary concern revolves around:

1. Misinformation and Disinformation: Deep Fakes can be used to create fabricated news, speeches, or events, leading to widespread confusion, erosion of trust in media, and potential societal instability.
2. Reputational Harm and Defamation: Individuals can be falsely portrayed in compromising situations, causing severe damage to their reputation, careers, and personal lives.
3. Erosion of Trust in Digital Media: The increasing sophistication of Deep Fakes makes it difficult for the average person to distinguish between real and fake content, leading to a general distrust of all digital media.
4. Security Risks: Deep Fakes could be used for identity theft, fraud, or even to influence political outcomes.

While the technology itself is neutral, its malicious application warrants serious attention and the development of robust countermeasures.

How to Identify Deep Fakes from Originals:

Detecting Deep Fakes is an ongoing challenge, as GANs are continuously improving. However, several methods and common tells can help in identification:

1. **Visual Inconsistencies and Artifacts:**
    - **Unnatural Blinking:** Deep Fake subjects often blink less frequently or have irregular blinking patterns compared to real people, as training data for blinking can be scarce.
    - **Facial Asymmetries:** Subtle inconsistencies in facial features, like eyes, ears, or mouth, that don't match the original person.
    - **Poorly Rendered Edges:** The transition between the manipulated face and the rest of the head or body might show blurry, jagged, or unnatural edges.
    - **Inconsistent Lighting and Shadows:** The lighting on the deep-faked face might not match the lighting in the background or on the original body, leading to unrealistic shadows or highlights.
    - **Low Resolution or Digital Artifacts:** Sometimes, parts of the Deep Fake might appear pixelated, distorted, or have digital noise, especially around the edges of the manipulated area.
    - **Missing or Inconsistent Skin Details:** Deep Fake faces might lack natural skin blemishes, pores, or hair follicles, or these details might be overly smooth or inconsistently applied.
2. **Auditory Inconsistencies (for videos with audio):**
    - **Lip-Sync Issues:** The movement of the lips might not perfectly synchronize with the spoken words.
    - **Unnatural Voice Tones or Cadence:** The synthesized voice might sound robotic, flat, or have an unnatural rhythm or intonation.
    - **Background Noise Discrepancies:** The background audio might not match the visual environment or could have subtle inconsistencies with the synthesized voice.
3. **Temporal Inconsistencies:**
    - **Jittering or Flickering:** Rapid, unnatural movements or changes in the manipulated area across frames.
    - **Pose or Head Movement Anomalies:** The head or body movements might appear stiff, unnatural, or not consistent with typical human motion.
4. **Metadata Analysis:**
    - **Examine File Metadata:** Original images and videos often contain metadata (EXIF data) about the camera, date, and time of capture. The absence of this data or inconsistent metadata can be a red flag.

5. **Technical Detection Methods (often requiring specialized tools):**
    - **AI-based Detectors:** Researchers are developing sophisticated AI models specifically trained to identify Deep Fakes by learning to recognize the subtle artifacts left by GANs. These models often look for patterns that are imperceptible to the human eye.
    - **Physiological Signal Analysis:** Analyzing heart rate, breathing patterns, or eye movements, which are difficult for current Deep Fake technology to perfectly replicate.
    - **Digital Watermarking/Blockchain:** Future solutions might involve embedding digital watermarks into original content or using blockchain to verify the authenticity and provenance of media.

While human observation can catch some obvious Deep Fakes, the most effective detection strategies will increasingly rely on a combination of visual inspection, forensic analysis, and advanced AI-powered detection tools.

Conclusion:

Generative Adversarial Networks (GANs) have significantly advanced the creation of synthetic media, enabling realistic and innovative digital content. However, these same capabilities have facilitated the emergence of Deep Fakes, which pose serious ethical, social, and security challenges. Deep Fakes can distort truth, manipulate public opinion, harm personal reputations, and undermine trust in digital information. As the technology behind Deep Fakes continues to improve, detection becomes increasingly complex.