Introduction

This dataset is designed for **regression analysis** and is mainly used for **machine learning practice**. It contains **numerical features only**, which makes it suitable for applying differen regression models without heavy preprocessing.

The dataset consists of **10,000 rows** and **7 columns**, where **6 columns are input features 1 column is the target variable**. There are **no missing values**, which helps in faster and cleaner model training.

Import libarays

In [6]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from IPython.display import HTML
import warnings
warnings.filterwarnings("ignore")

plt.rcParams["figure.figsize"] = (10, 6)
print('All libraries imported successfully')
```

```
All libraries imported successfully
```

Loading the Dataset

In [7]:
```python
data = pd.read_csv("practice_dataset.csv")
data.head()
```

Out[7]:

	a	b	c	d	e	f	target
0	3.745401	7.472816	2.299983	6.743301	-4.021759	2.541710	-23.804962
1	9.507143	6.658242	-3.154880	5.133632	-8.103644	1.483551	-158.554897
2	7.319939	3.523078	-1.533603	9.680487	-7.472816	0.586397	-69.630480
3	5.986585	12.145333	1.632806	2.970806	-6.386577	2.209925	-77.556230
4	1.560186	9.532483	-0.179107	6.290708	-5.926933	1.256034	-93.973154

Exploratory Data Analysis(EDA)

In [8]:
```python
# ------------------------------------------------------------
print("*"*70)
print("\033[1m" + "Columns Names".center(70) + "\033[0m")
print("*"*70)
practice  = pd.DataFrame(data.columns, columns=["Columns"])
display(practice)
# -------------------------------------------------------------
print("*"*70)
#print("")
print("\033[1m" + "Information Dataset ".center(70) + "\033[0m")
print("*"*70)
print(" ")
# -------------------------------------------------------------
data.info()
print("*"*70)
print("\033[1m" + "Data Types from Dataset".center(70) + "\033[0m")
print("*"*70)
display(data.dtypes)
# -------------------------------------------------------------
print("*"*70)
print("\033[1m" + "Describetion".center(70) + "\033[0m")
print("*"*70)
display(data.describe())
# ---------------------------------------------------------------
print("*"*70)
print("\033[1m" + "Missing Values".center(70) + "\033[0m")
print("*"*70)
```

```
display(data.isnull().sum())
# ---------------------------------------------------------------
print("*"*70)
print("\033[1m" + "Duplicated Values".center(70) + "\033[0m")
print("*"*70)
display(data.duplicated())
# ---------------------------------------------------------------
print("*"*70)
print("\033[1m" + "Shape of the Dataset ".center(70) + "\033[0m")
print("*"*70)
var_1 = pd.DataFrame(data.shape, columns = ["Shape"])
var_1
# ---------------------------------------------------------------
```

```
**********************************************************************
                            Columns Names
**********************************************************************
```

	Columns
0	a
1	b
2	c
3	d
4	e
5	f
6	target

```
**********************************************************************
                         Information Dataset
**********************************************************************

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 7 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   a       10000 non-null  float64
 1   b       10000 non-null  float64
 2   c       10000 non-null  float64
 3   d       10000 non-null  float64
 4   e       10000 non-null  float64
 5   f       10000 non-null  float64
 6   target  10000 non-null  float64
dtypes: float64(7)
memory usage: 547.0 KB
**********************************************************************
                       Data Types from Dataset
**********************************************************************
a         float64
b         float64
c         float64
d         float64
e         float64
f         float64
target    float64
dtype: object
```

```
**********************************************************************
                              Describetion
**********************************************************************
```

	a	b	c	d	e	f
count	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000
mean	4.941596	10.090598	0.000504	5.488420	-0.062724	1.509434
std	2.876301	5.785891	2.867738	2.600951	5.785684	0.865024
min	0.000116	0.003155	-4.999519	1.000050	-9.999665	0.000025
25%	2.463289	5.078916	-2.462521	3.221715	-5.114556	0.770869
50%	4.925286	10.117936	0.020681	5.500272	-0.125656	1.518271
75%	7.400063	15.129584	2.446739	7.724046	5.009541	2.260338
max	9.997177	19.998497	4.999010	9.998104	9.999443	2.999819

```
**********************************************************************
                             Missing Values
**********************************************************************
a         0
b         0
c         0
d         0
e         0
f         0
target    0
dtype: int64
**********************************************************************
                           Duplicated Values
**********************************************************************
0       False
1       False
2       False
3       False
4       False
        ...
9995    False
9996    False
9997    False
9998    False
9999    False
Length: 10000, dtype: bool
**********************************************************************
                         Shape of the Dataset
**********************************************************************
```

Out[8]:

	Shape
0	10000
1	7

✅ EDA Completed

The **Exploratory Data Analysis (EDA)** has been completed successfully. The dataset do **not contain any null or missing values**, and **no duplicate records** were found.

This confirms that the dataset is **clean and well-prepared** for further analysis. We will no proceed to the **Data Visualization** step to better understand patterns and relationships within the data.

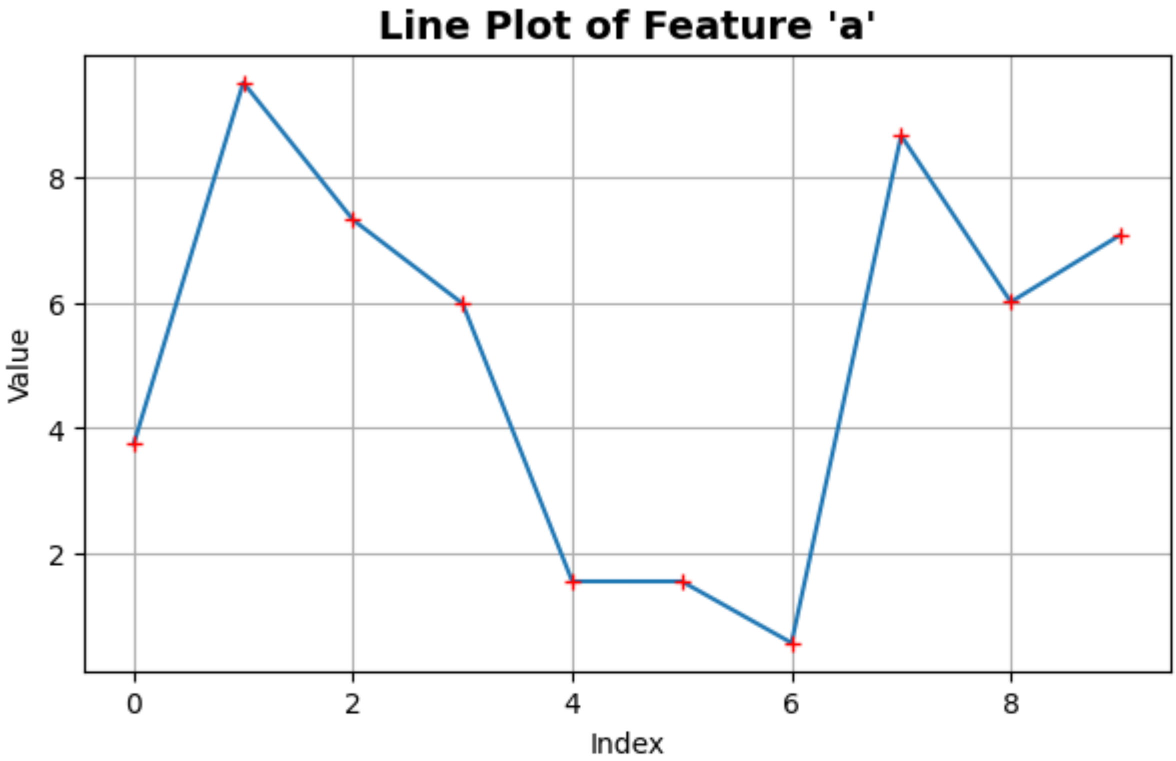
📊 Start Data Visualization

In this step, we begin the **data visualization** process to explore patterns, trends, and relationships within the dataset.

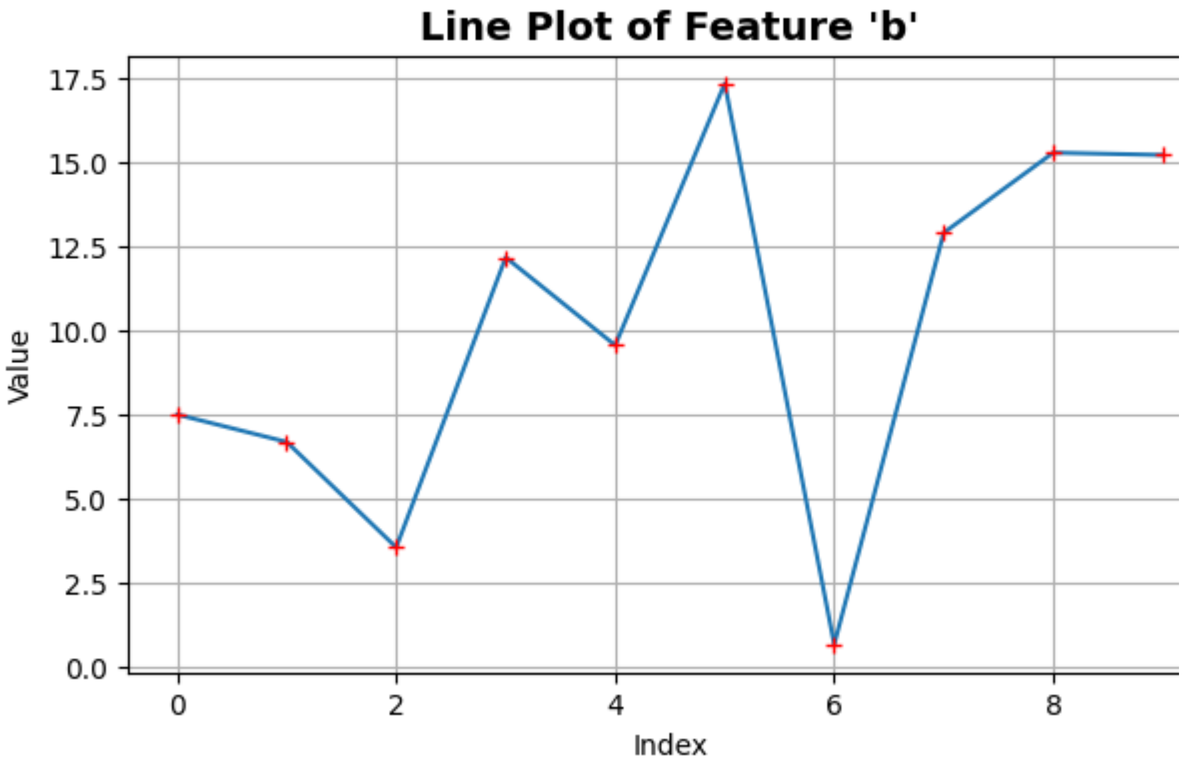
In [9]:

```
#==========================================================================
#                                         Line plot
# ========================================================================
features = ['a', 'b', 'c', 'd', 'e', 'f']
# Loop through each feature
for feature in features:
    plt.figure(figsize=(7,4))
    plt.plot(data[feature].head(10), marker='+', markeredgecolor='red', line
    # Title and labels
    plt.title(f"Line Plot of Feature '{feature}'", fontsize=14, fontweight='
    plt.xlabel("Index")
    plt.ylabel("Value")
    plt.grid(True)  # optional: adds grid
    plt.show()
#==========================================================================
#                                      Bar Plot from dataset
#==========================================================================

print("   ")
print("\033[1m" + "Bar Plot ".center(70) + "\033[0m")
print("   ")

#==========================================================================
for feature in features:
    plt.figure(figsize=(7,4))
    values = data[feature].head(10)
    x = range(len(values))
    plt.bar(x, values)
    # Title and labels
    plt.title(f"Line Plot of Feature '{feature}'", fontsize=14, fontweight='
    plt.xlabel("Index")
    plt.ylabel("Feature")
    plt.grid(True)  # optional: adds grid
    plt.show()
#==========================================================================
#                                              Histogram
#==========================================================================
print("   ")
print("\033[1m" + "Histogram".center(70) + "\033[0m")
print("   ")
plt.figure(figsize=(8,5))
plt.hist(data['a'].head(19), bins=10, color='skyblue', edgecolor='black')  #
```

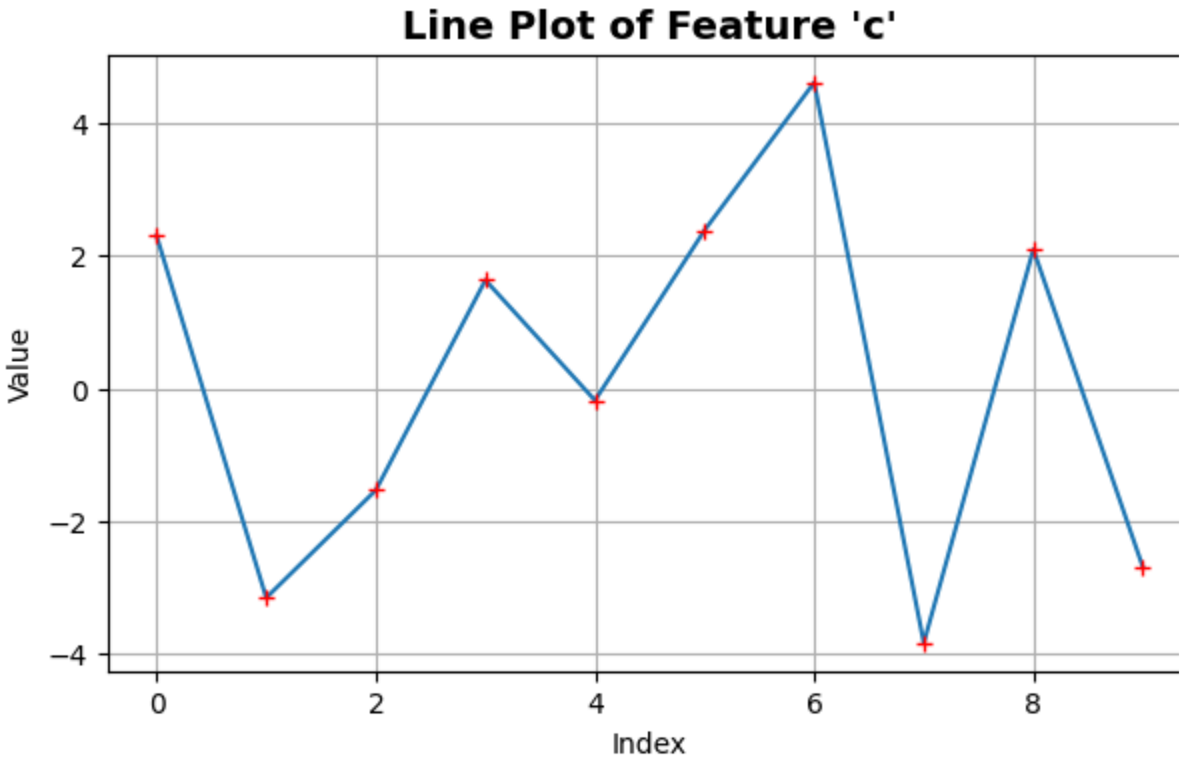
```
plt.title("Histogram of Feature 'a'", fontsize=14, fontweight='bold')
plt.xlabel("Value")
plt.ylabel("Frequency")
plt.grid(axis='y')
plt.show()
#===========================================================================

print("   ")
print("\033[1m" + "Box Plot".center(70) + "\033[0m")
print("   ")

#===========================================================================
#                                         Box Plot
#===========================================================================
plt.figure(figsize=(10,6))
# Boxplot
plt.boxplot([data[feature] for feature in features], labels=features, patch_
            boxprops=dict(facecolor='skyblue', color='black'),
            whiskerprops=dict(color='black'),
            capprops=dict(color='black'),
            medianprops=dict(color='red', linewidth=2),
            flierprops=dict(marker='o', markerfacecolor='red', markersize=6,
plt.title("Box Plot of Features 'a' to 'f'", fontweight='bold')
plt.xlabel("Features", fontsize=12)
plt.ylabel("Value", fontsize=12)
plt.grid(axis='y', alpha=0.75)
plt.show()
#===========================================================================

print("   ")
print("\033[1m" + "Heatmap ".center(70) + "\033[0m")
print("   ")

#===========================================================================
#                                     Heatmap from dataset
#===========================================================================
corr_matrix = data[features].corr()

# Plot heatmap
plt.figure(figsize=(8,6))
sns.heatmap(corr_matrix, annot=True, cmap="coolwarm", linewidths=1, linecolo
plt.title("Heatmap of Feature Correlations ('a' to 'f')", fontsize=14, fontw
plt.show()
#===========================================================================
#                                     Pair Plot /Kde plot
#===========================================================================
plt.figure(figsize=(8,6))
sns.kdeplot(data["a"], fill="green")
plt.show()
```

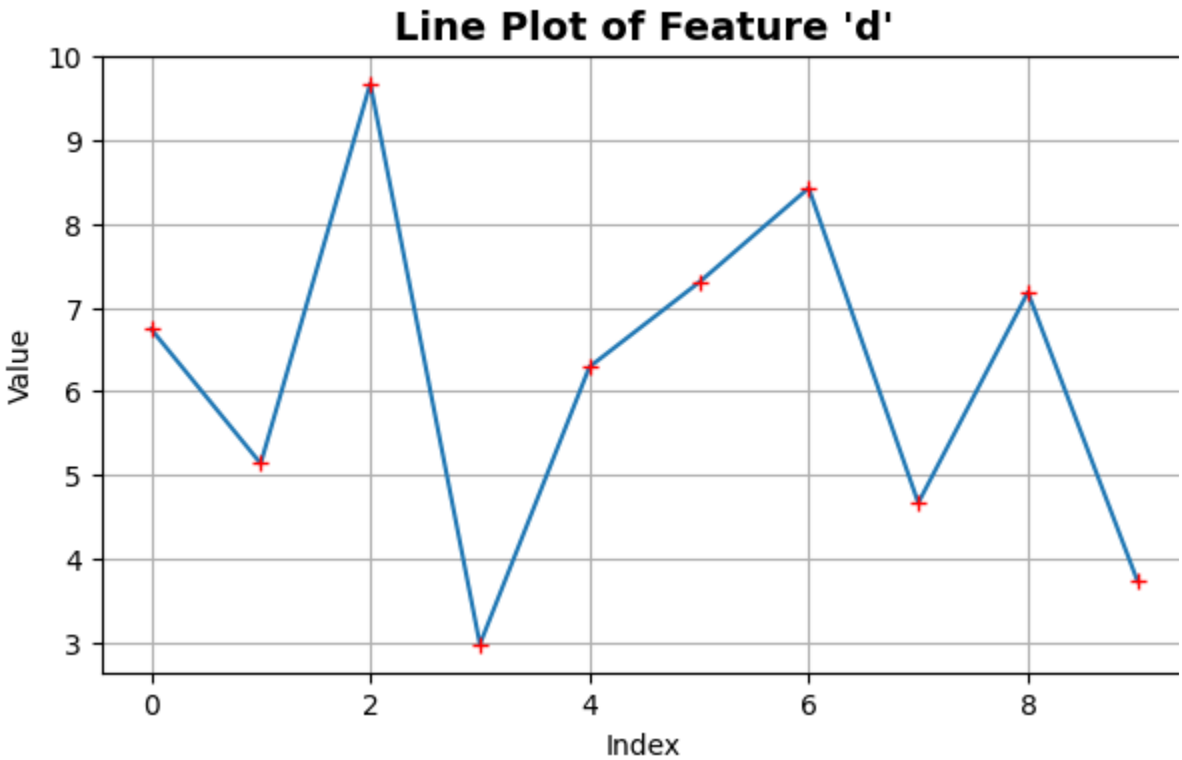


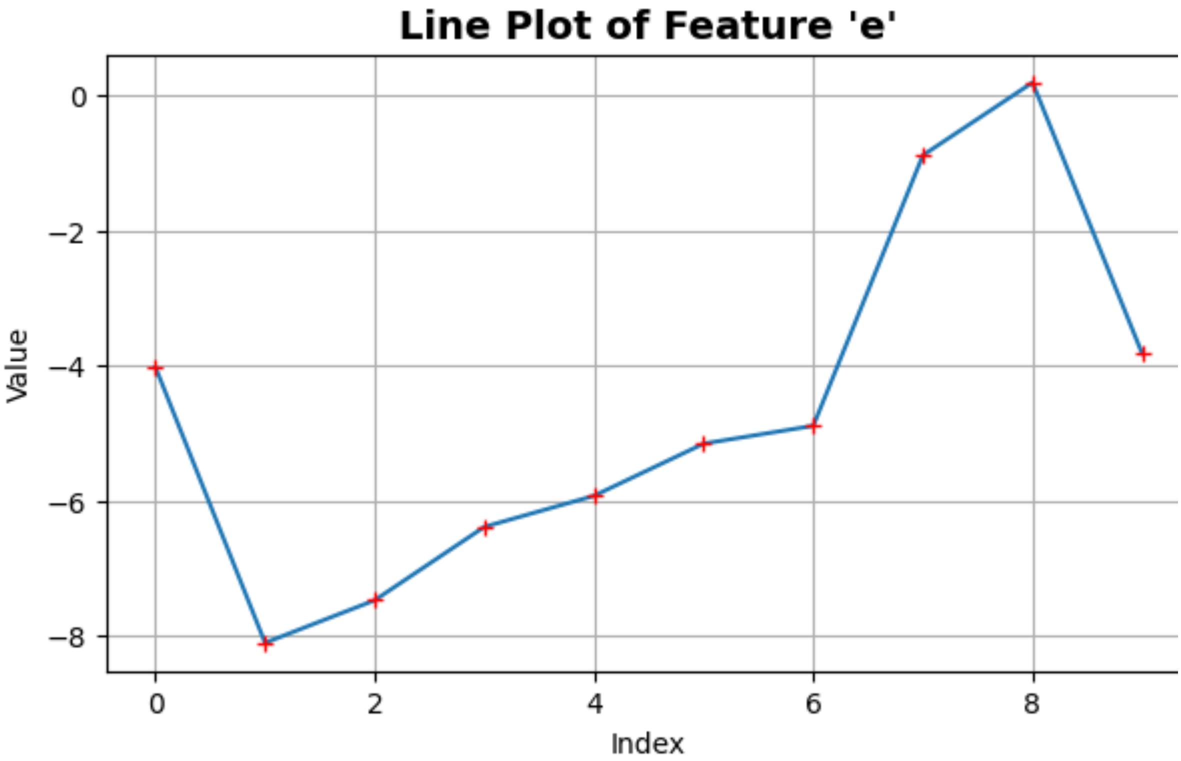
Line Plot of Feature 'a'
Value
Index



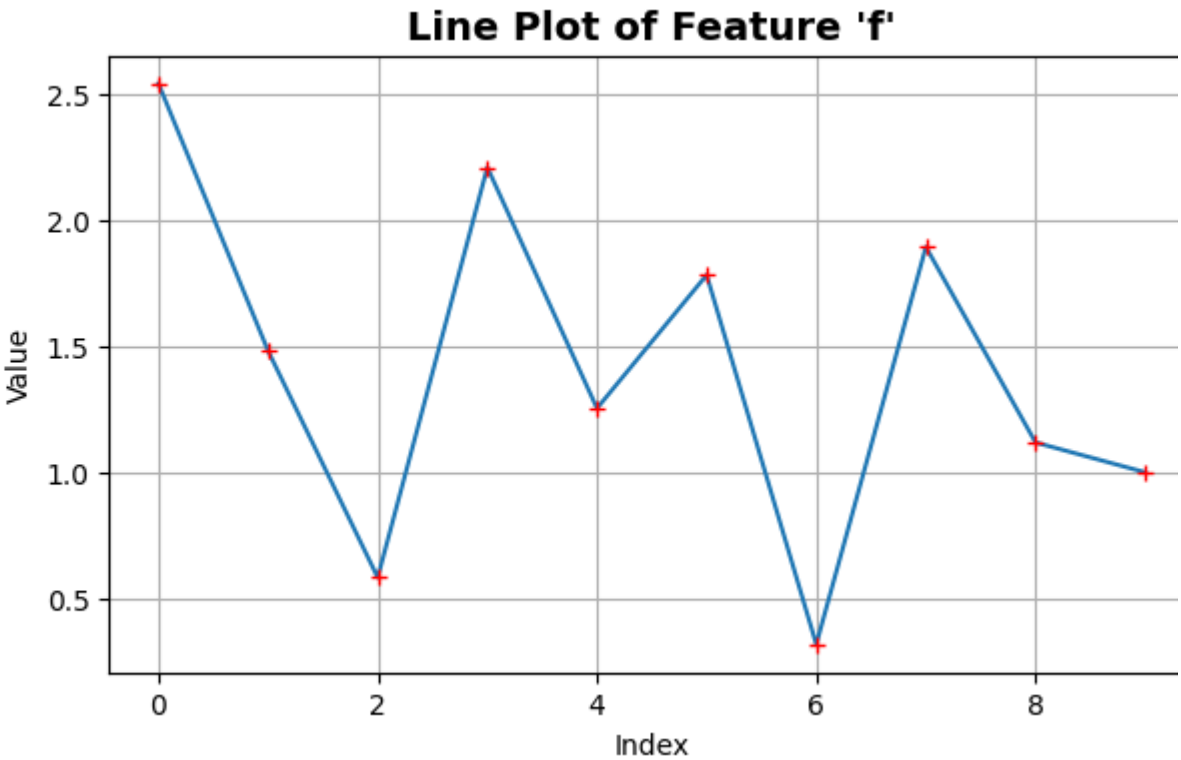
Line Plot of Feature 'b'
Value
Index


Line Plot of Feature 'c'
Value
Index
4
2
0
−2
−4
0
2
4
6
8


Line Plot of Feature 'd'
Value
Index
10
9
8
7
6
5
4
3
0
2
4
6
8

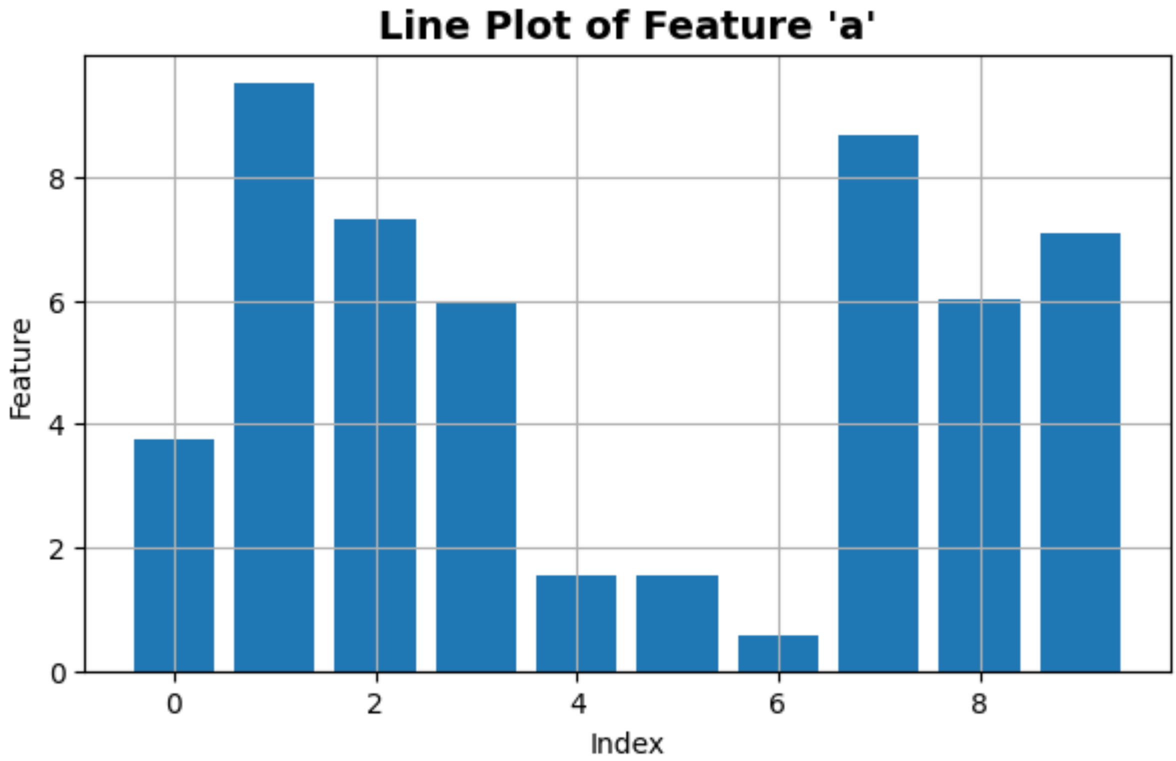


Line Plot of Feature 'e'
0
−2
−4
−6
−8
Value
0
2
4
6
8
Index

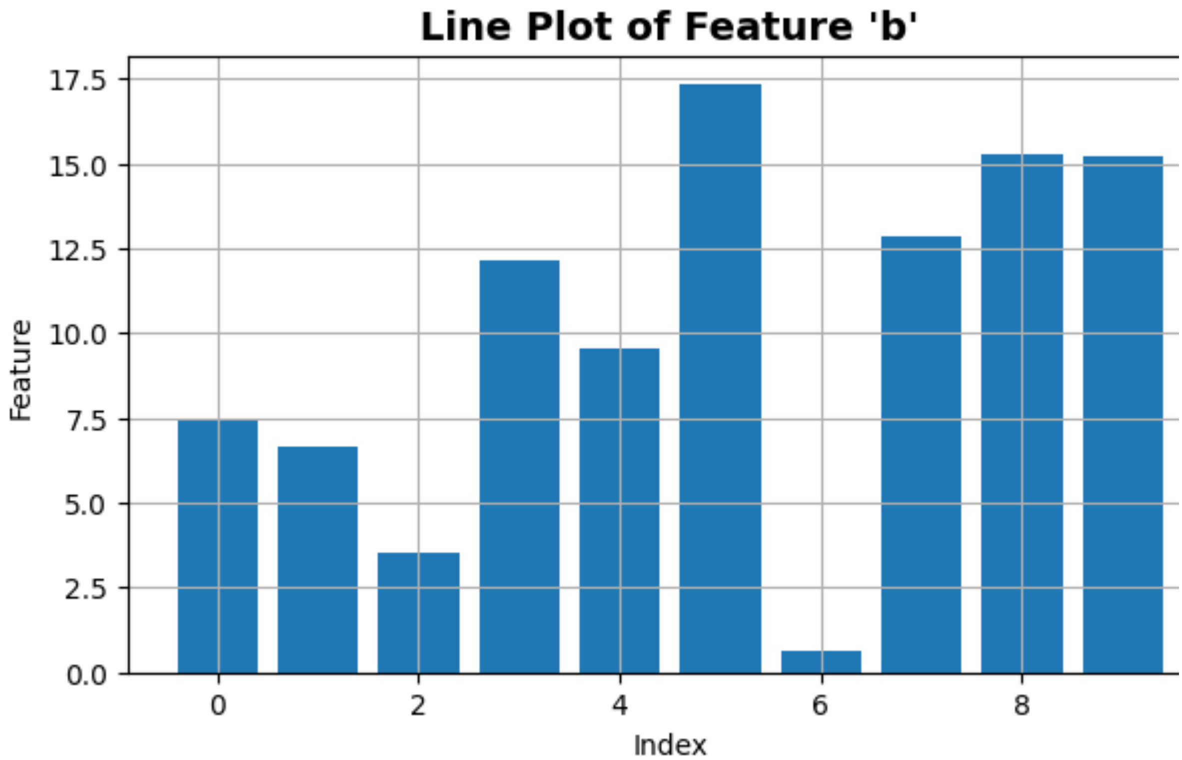


Line Plot of Feature 'f'
2.5
2.0
1.5
1.0
0.5
Value
0
2
4
6
8
Index

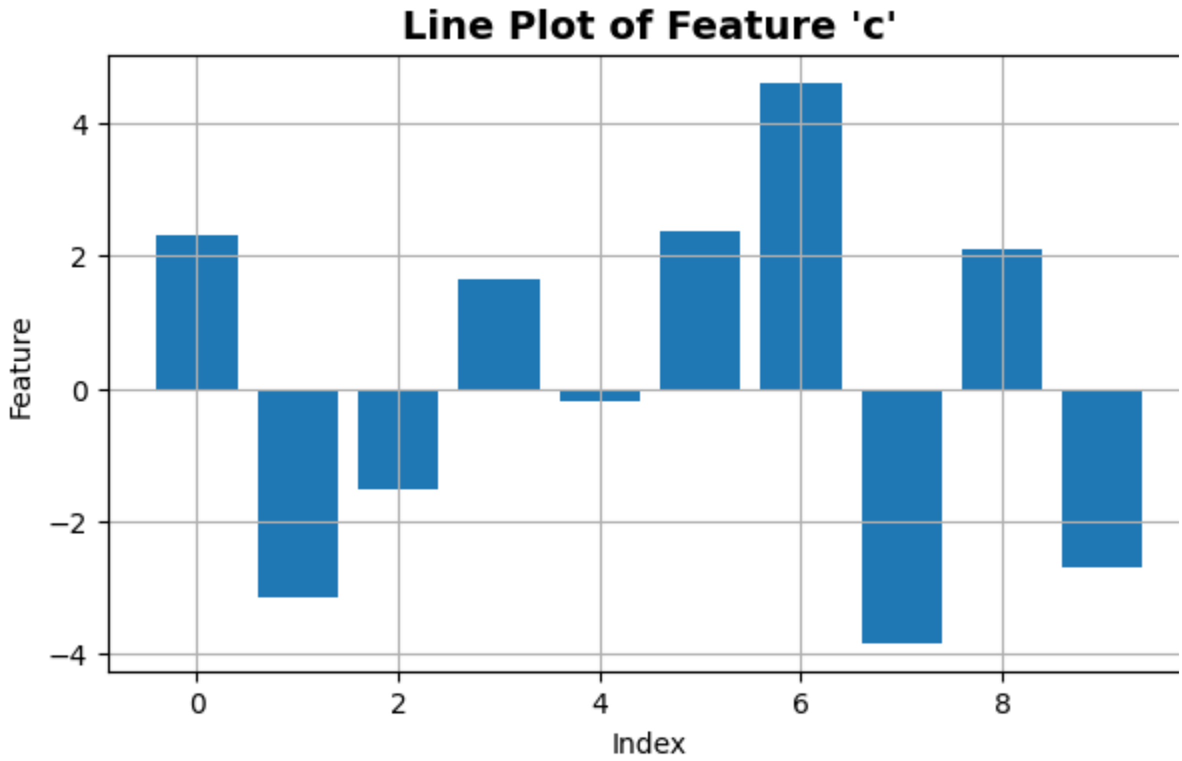
Bar Plot



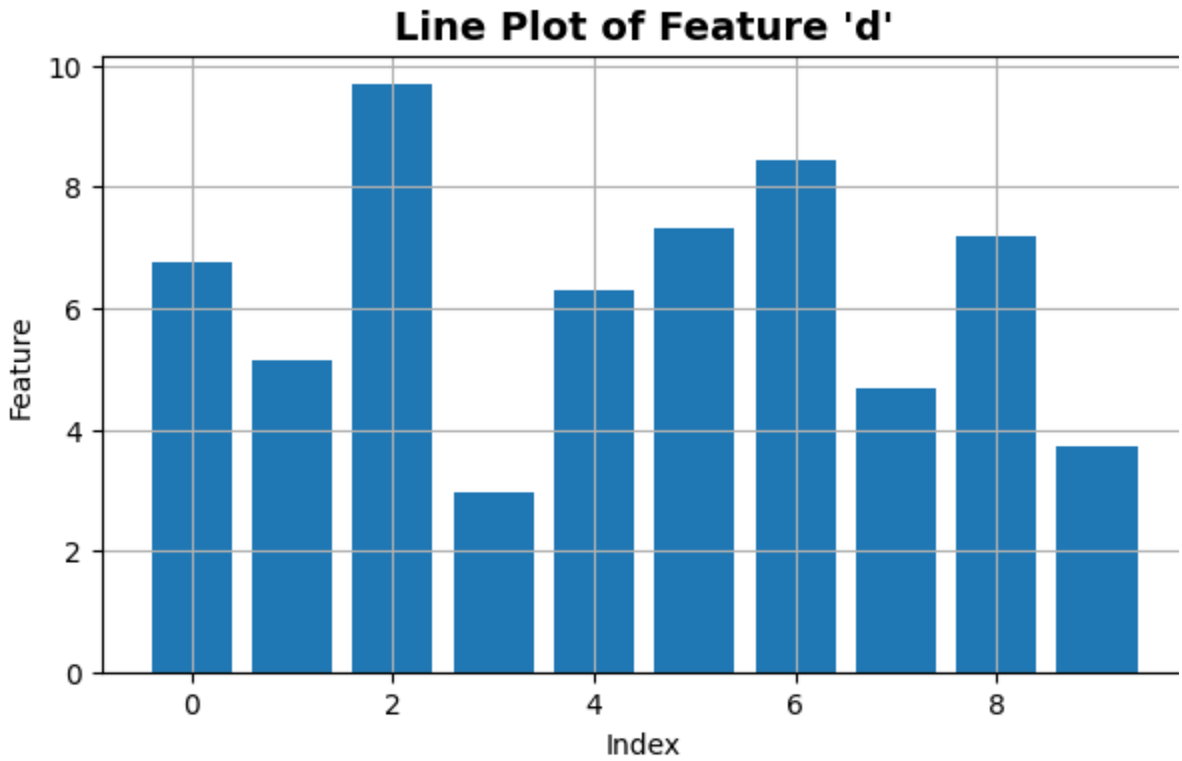
Line Plot of Feature 'a'
Feature
Index
0
2
4
6
8



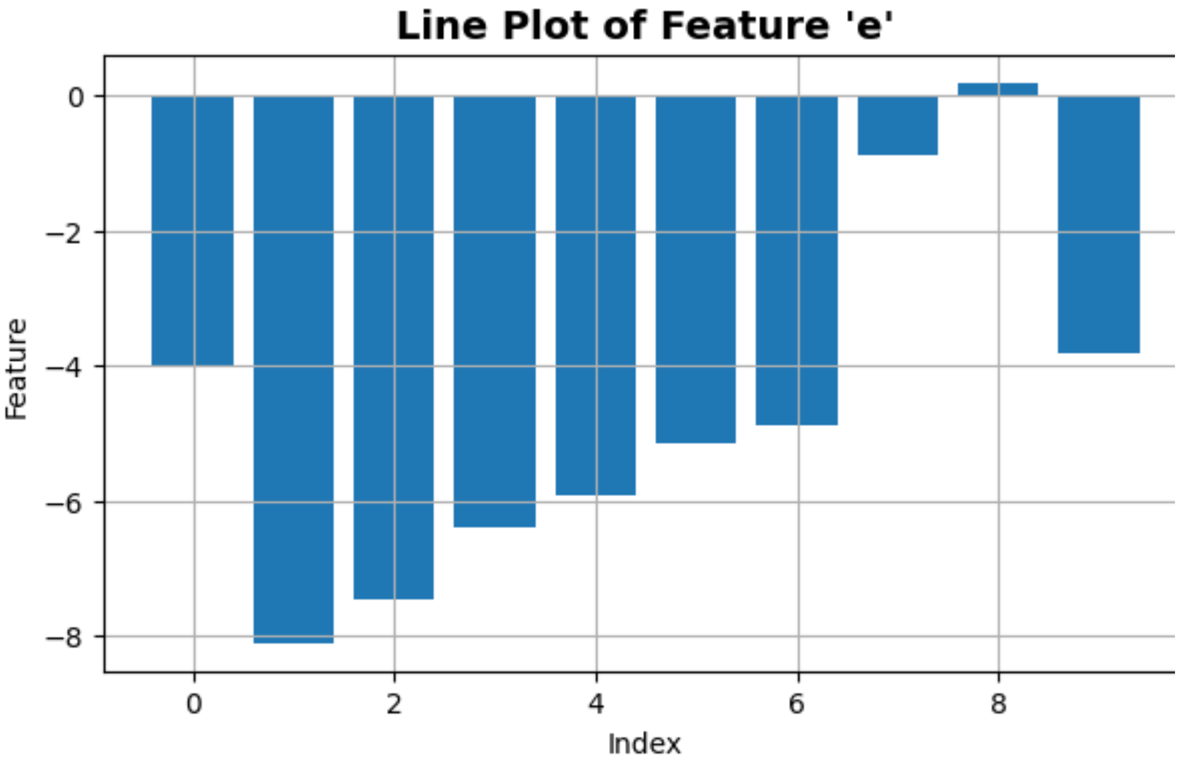
Line Plot of Feature 'b'
Feature
Index
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
0
2
4
6
8

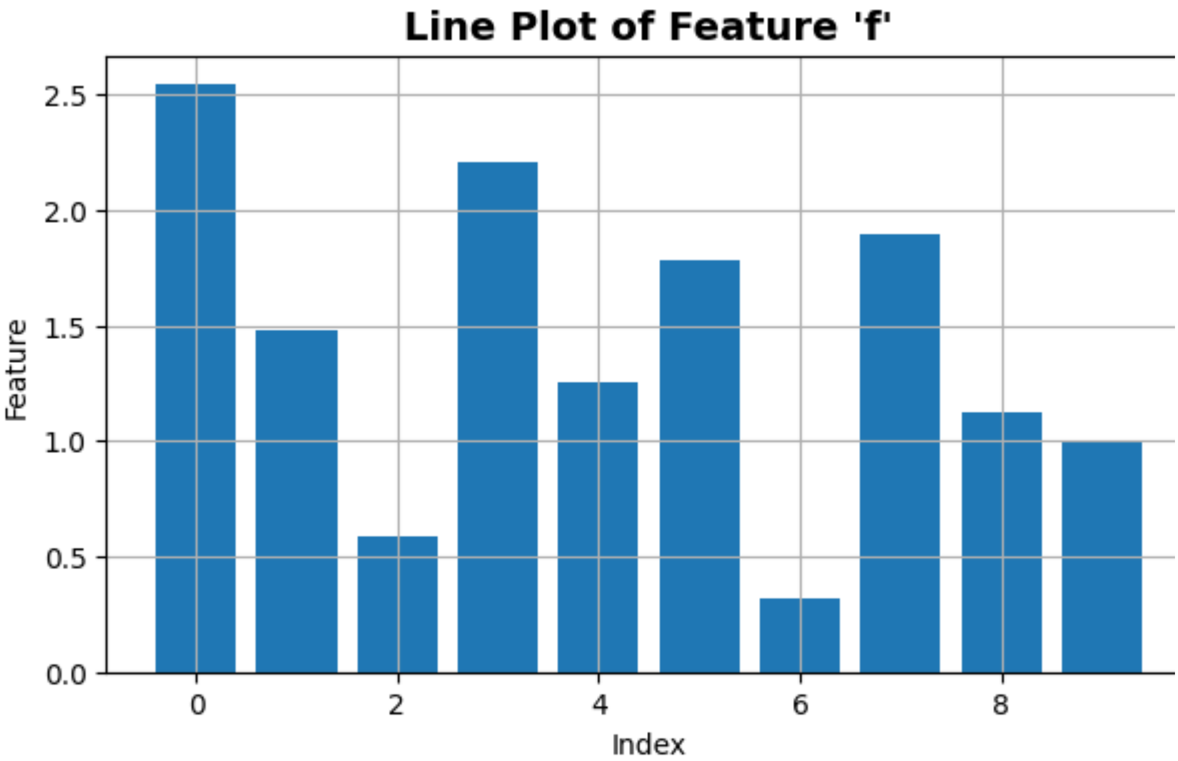


Line Plot of Feature 'c'
Feature
Index
4
2
0
−2
−4
0
2
4
6
8

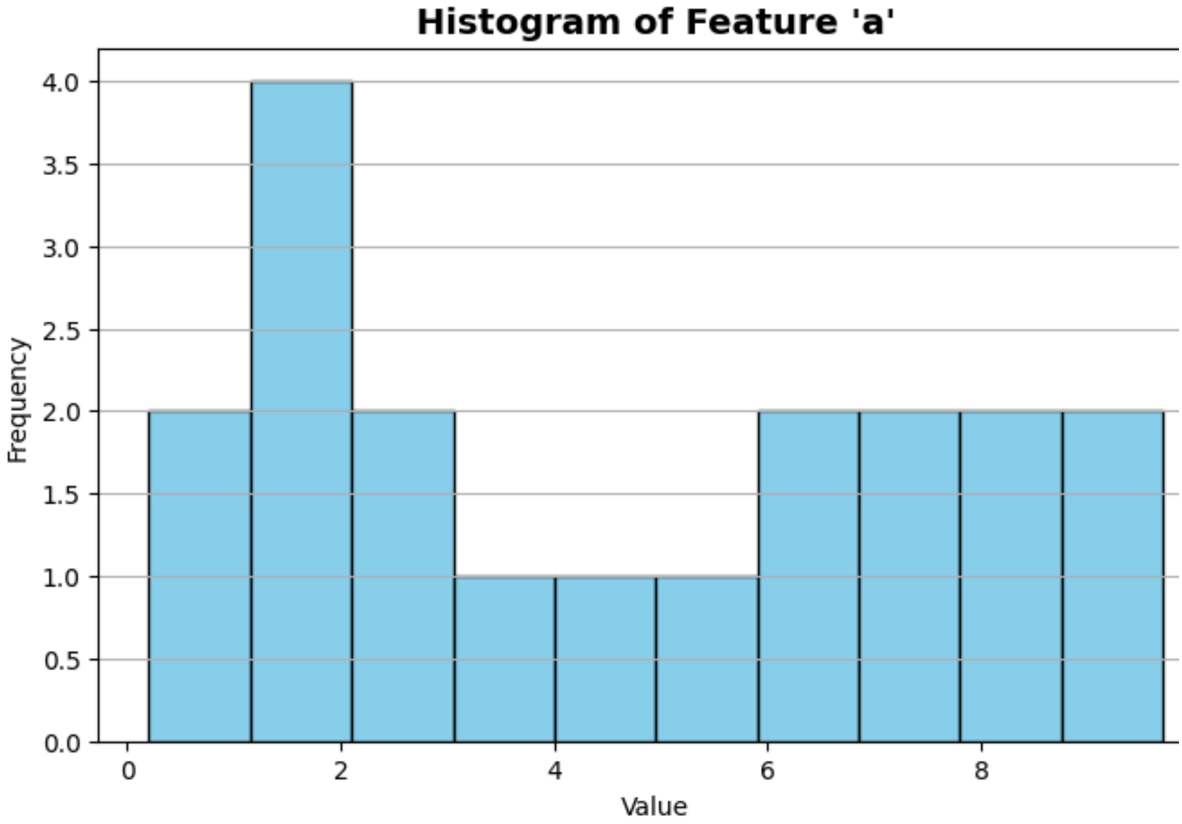


Line Plot of Feature 'd'
Feature
Index
10
8
6
4
2
0
0
2
4
6
8

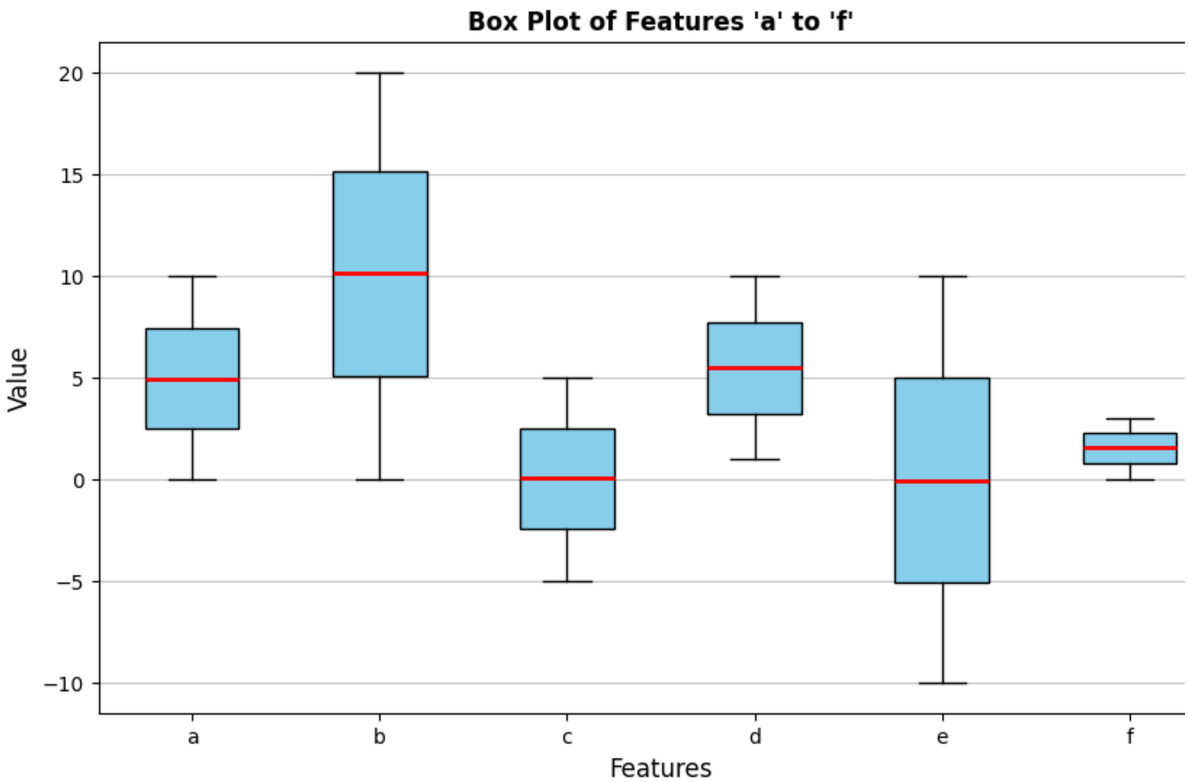

Line Plot of Feature 'e'
Feature
Index
0
−2
−4
−6
−8
0
2
4
6
8


Line Plot of Feature 'f'
Feature
Index
2.5
2.0
1.5
1.0
0.5
0.0
0
2
4
6
8

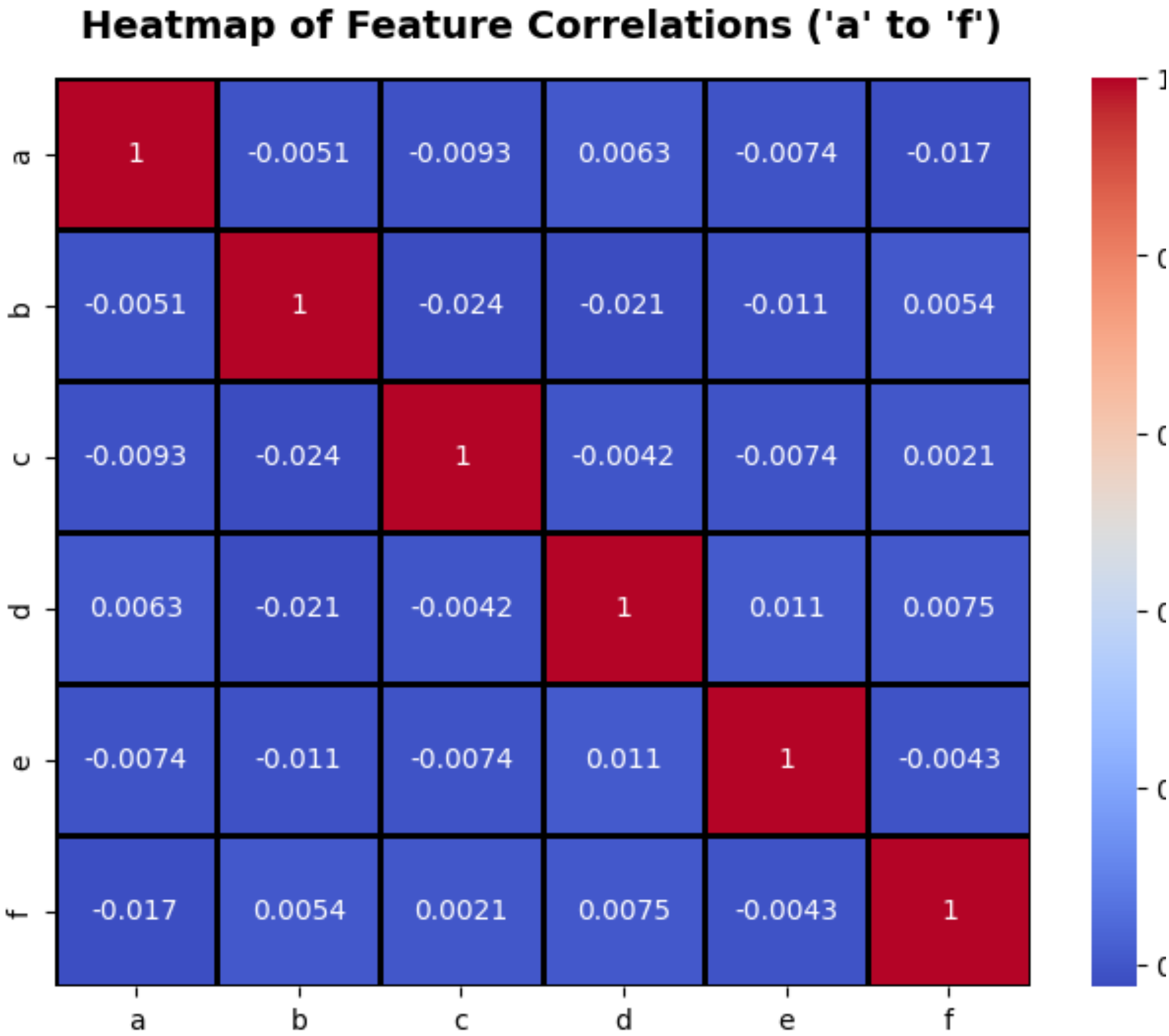
Histogram

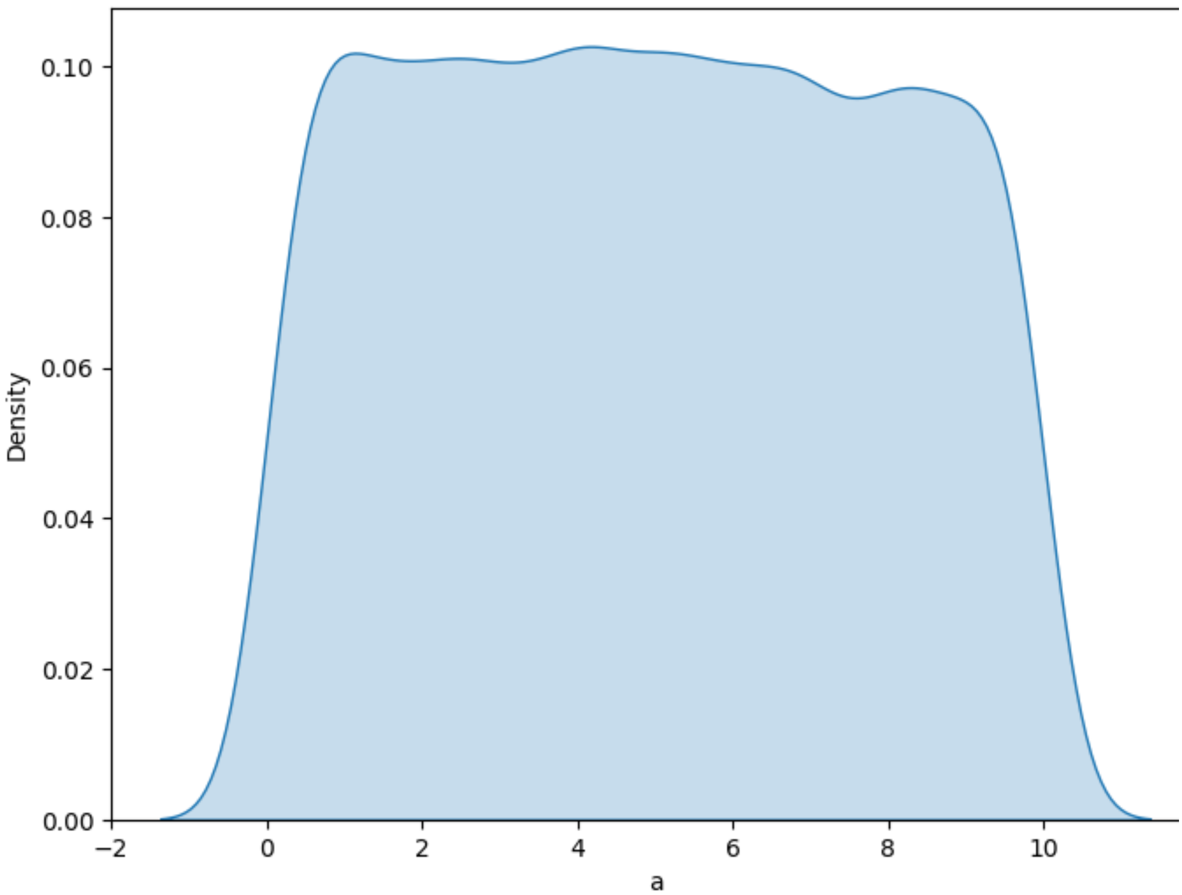


Box Plot



Heatmap


Heatmap of Feature Correlations ('a' to 'f')
a
b
c
d
e
f
1 -0.0051 -0.0093 0.0063 -0.0074 -0.017
-0.0051 1 -0.024 -0.021 -0.011 0.0054
-0.0093 -0.024 1 -0.0042 -0.0074 0.0021
0.0063 -0.021 -0.0042 1 0.011 0.0075
-0.0074 -0.011 -0.0074 0.011 1 -0.0043
-0.017 0.0054 0.0021 0.0075 -0.0043 1


Density
0.00
0.02
0.04
0.06
0.08
0.10
−2
0
2
4
6
8
10
a

Insight

Feature a shows a moderate spread with a few outliers. Feature d has a narrow distribution indicating consistent values. Features b and f have higher variability and multiple outliers, whic need further investigation.

Feature Engreening

1. Train Test split

2. Standrad Scaler

3. LinearRegression

4. DecisionTreeRegressor

5. RandomForestRegressor

6. Pipeline

In [14]:

```
#============================================================================
#                                  Train Test Split
#============================================================================
from sklearn.model_selection import train_test_split
X = data.iloc[:, :-1]
y = data.iloc[:, -1]

X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    random_state = 42,
    test_size = 0.2
)
#============================================================================
#                                 X_train_describe
#============================================================================
print("*"*80)
print("\033[1m" + "Before Describe mean/std".center(70) + "\033[0m")
print("*"*80)
display(X_train.describe().round(2))
#============================================================================
#                                 Standrad_Scalar
#============================================================================
print("*"*80)
print("\033[1m" + "After Describe Mean/Standard derivation".center(70) + "\0
print("\033[1m" + "Used Standard Scaler".center(70) + "\033[0m")
print("*"*80)
                    #===================================
                              #Import Model
                    #===================================
from sklearn.preprocessing import StandardScaler
Scaler = StandardScaler()
                    #===================================
                           #Fit/Transfrom model
                    #===================================
Scaler.fit(X_train)
```

```
X_train_scaled = Scaler.transform(X_train)
X_test_scaled = Scaler.transform(X_test)
X_train_Scaler = pd.DataFrame(X_train_scaled, columns = X_train.columns)
X_test_Scaler = pd.DataFrame(X_test_scaled, columns = X_train.columns)
                    #===================================
                                # Describe
                    #===================================
X_test_Scaler.describe()
print("The StandardScaler model is working correctly because the mean is 0 a
```

```
********************************************************************************
                           Before Describe mean/std
********************************************************************************
```

	a	b	c	d	e	f
count	8000.00	8000.00	8000.00	8000.00	8000.00	8000.00
mean	4.93	10.07	0.00	5.49	-0.07	1.51
std	2.88	5.80	2.87	2.60	5.78	0.86
min	0.00	0.00	-5.00	1.00	-10.00	0.00
25%	2.44	5.03	-2.46	3.23	-5.12	0.77
50%	4.92	10.06	0.01	5.52	-0.12	1.52
75%	7.38	15.17	2.44	7.73	4.99	2.26
max	10.00	20.00	5.00	10.00	10.00	3.00

```
********************************************************************************
                   After Describe Mean/Standard derivation
                             Used Standard Scaler
********************************************************************************
The StandardScaler model is working correctly because the mean is 0 and the standa
deviation is 1.
```

In [44]:

```
#=========================================================================
#                            Linear Regression
#=========================================================================
                   #======================================
                             #Import model
                   #======================================
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_squared_error
                   #======================================
                         # Fit/transfrom model
                   #======================================
print("*"*50)
print("That is linear Regression Score R2:")
print("*"*50)
lr_model = LinearRegression()
lr_model.fit(X_train_scaled, y_train)
y_pred = lr_model.predict(X_test_scaled)
r2 = r2_score(y_test, y_pred)
rmse = mean_squared_error(y_test, y_pred)

print("R2 Score:", r2)
print("RMSE:", rmse)
#======================================================================
                        #DecisionTreeRegressor
#======================================================================
from sklearn.tree import DecisionTreeRegressor
```

```python
print(" ")
print("*"*50)
print("That is DecisionTreeRegressor Score R2:")
print("*"*50)
for depth in [10]:
    model = DecisionTreeRegressor(max_depth=depth, random_state=42)
                  #=================================
                        #Fit/Predict model
                  #=================================
    model.fit(X_train, y_train)
    pred = model.predict(X_test)
    print("R2:", r2_score(y_test, pred))

#===========================================================
                  # RandomForestRegressor
#===========================================================
from sklearn.ensemble import RandomForestRegressor

rf = RandomForestRegressor(
    n_estimators=200, ## 200 trees
    max_depth=10,  # maximum height 10 levels
    min_samples_split=5,     # model is stable
    min_samples_leaf=2,       # Overfitting control
    random_state=42        # Data randomly split
)
#===========================================================
                  # Fit/Predict Model
#===========================================================
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)
print(" ")
print("*"*50)
print("That is RandomForestRegressor Score:")
print("*"*50)
print("R2:", r2_score(y_test, y_pred))
print("RMSE:", np.sqrt(mean_squared_error(y_test, y_pred)))
```

```
**************************************************
That is linear Regression Score R2:
**************************************************
R2 Score: 0.5281134635297018
RMSE: 2509.0934429103254

**************************************************
That is DecisionTreeRegressor Score R2:
**************************************************
R2: 0.9046040055320388

**************************************************
That is RandomForestRegressor Score:
**************************************************
R2: 0.9628245221688397
RMSE: 14.0594361629072
```

from IPython.display import display, HTML

display(HTML("""

Model Performance Conclusion

Linear Regression

- **R^2 Score:** 0.53
- **RMSE:** 2509.09

Linear Regression shows **poor performance** on this dataset. This indicates that the relationship between features and target is **not purely linear**. The high RMSE value also suggests large prediction errors.

Decision Tree Regressor

- **R^2 Score:** 0.90

Decision Tree Regressor significantly improves performance by capturing **non-linear patterns** in the data. This model provides a strong fit with much better explanatory power than Linear Regression.

Random Forest Regressor ☆ (Best Model)

- **R^2 Score:** 0.96
- **RMSE:** 14.05

Random Forest Regressor delivers the **best performance** among all models. With the highest R^2 score and the lowest RMSE, it demonstrates excellent generalization and accuracy. The ensemble approach reduces overfitting and captures complex feature interactions effectively.

☑ Final Conclusion

Based on the evaluation metrics, **Random Forest Regressor** is the most suita model for this dataset. It outperforms Linear Regression and Decision Tree by a significant margin and is recommended for **real-world deployment** and **future predictions**.


```
In [ ]:
```