Exploratory Data Analysis (EDA) in Python

Exploratory Data Analysis (EDA) is an approach to analyzing datasets to summarize their main characteristics, often with visual methods. EDA helps identify patterns, detect anomalies, test hypotheses, and check assumptions.

Here's how you can perform EDA in Python using libraries like **Pandas**, **Matplotlib**, **Seaborn**, and **Plotly**.

---

1. Import Required Libraries

bash
Copy code
```bash
pip install pandas numpy matplotlib seaborn plotly
```

python
Copy code
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
```

---

2. Load the Data

python
Copy code
```python
# Load data from a CSV file
data = pd.read_csv('data.csv')

# View first few rows of the dataset
print(data.head())

# Get a summary of the dataframe
print(data.info())

# Check for missing values
print(data.isnull().sum())
```

---

3. Data Cleaning

Before beginning the analysis, you may need to clean your data:

Handling missing values:
python
Copy code
```
# Fill missing values with mean/median/mode or drop
data['Column'] = data['Column'].fillna(data['Column'].mean())
data = data.dropna()  # Drop rows with any missing values
```

-

Remove duplicates:
python
Copy code
```
data = data.drop_duplicates()
```

-

Outlier detection:
python
Copy code
```
# Using IQR to detect and remove outliers
Q1 = data['Column'].quantile(0.25)
Q3 = data['Column'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
data = data[(data['Column'] >= lower_bound) & (data['Column'] <=
upper_bound)]
```

-

---

4. Descriptive Statistics

Get an overview of the central tendency, spread, and shape of the dataset's distribution.

python
Copy code
```
# Summary statistics
print(data.describe())

# Mean, median, mode
mean = data['Column'].mean()
median = data['Column'].median()
```

```
mode = data['Column'].mode()[0]
print("Mean:", mean, "Median:", median, "Mode:", mode)

# Variance and standard deviation
variance = data['Column'].var()
std_dev = data['Column'].std()
print("Variance:", variance, "Standard Deviation:", std_dev)
```

---

5. Univariate Analysis (Analyzing a Single Variable)

Visualize Distributions

Histogram:
python
Copy code
```
data['Column'].hist(bins=20, edgecolor='black')
plt.title('Histogram of Column')
plt.xlabel('Values')
plt.ylabel('Frequency')
plt.show()
```

1.

Boxplot:
python
Copy code
```
sns.boxplot(x=data['Column'])
plt.title('Boxplot of Column')
plt.show()
```

2.

Density Plot (Kernel Density Estimation - KDE):
python
Copy code
```
sns.kdeplot(data['Column'], shade=True)
plt.title('KDE of Column')
plt.show()
```

3.

Categorical Variables Analysis
python
Copy code
```
# Count plot for categorical data
```

```
sns.countplot(x='CategoryColumn', data=data)
plt.title('Count Plot of CategoryColumn')
plt.show()
```

---

6. Bivariate Analysis (Analyzing Two Variables)

Correlation Analysis

Pearson Correlation:
python
Copy code
```
correlation = data.corr()
print(correlation)
```

-

Heatmap:
python
Copy code
```
sns.heatmap(correlation, annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```

-

Scatter Plot (For continuous variables):
python
Copy code
```
plt.scatter(data['Column1'], data['Column2'])
plt.title('Scatter Plot of Column1 vs Column2')
plt.xlabel('Column1')
plt.ylabel('Column2')
plt.show()
```

Pairplot (For exploring multiple relationships):
python
Copy code
```
sns.pairplot(data[['Column1', 'Column2', 'Column3', 'Column4']])
plt.show()
```

---

7. Multivariate Analysis (Analyzing More Than Two Variables)

Facet Grid (Multiple plots based on categorical variables):

python
Copy code

```
sns.FacetGrid(data, col='CategoryColumn').map(plt.hist, 'Column')
plt.show()
```

Heatmap of Correlation Matrix

python
Copy code

```
plt.figure(figsize=(12, 8))
sns.heatmap(data.corr(), annot=True, cmap='coolwarm',
linewidths=0.5)
plt.title('Correlation Matrix Heatmap')
plt.show()
```

Violin Plot (For comparing distributions across categories):

python
Copy code

```
sns.violinplot(x='CategoryColumn', y='Column', data=data)
plt.title('Violin Plot of Column by CategoryColumn')
plt.show()
```

---

8. Time Series Analysis (If Applicable)

Plot Time Series Data

python
Copy code

```
data['Date'] = pd.to_datetime(data['Date'])  # Convert to datetime
data.set_index('Date', inplace=True)

data['Column'].plot(figsize=(10, 5))
plt.title('Time Series Plot of Column')
plt.show()
```

Rolling Mean

python
Copy code

```
data['RollingMean'] = data['Column'].rolling(window=7).mean()
data[['Column', 'RollingMean']].plot(figsize=(10, 5))
plt.title('Rolling Mean of Column')
```

```
plt.show()
```

---

9. Advanced Visualizations

Interactive Visualizations with Plotly

python
Copy code

```
fig = px.scatter(data, x='Column1', y='Column2',
color='CategoryColumn', title="Interactive Scatter Plot")
fig.show()

fig = px.histogram(data, x='Column1', title="Interactive Histogram")
fig.show()
```

---

10. Identifying and Handling Categorical Variables

For categorical variables, you can:

Convert categorical variables to numeric form using **Label Encoding** or **One-Hot Encoding**:
python
Copy code

```
# Label Encoding
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
data['CategoryColumn'] =
label_encoder.fit_transform(data['CategoryColumn'])

# One-Hot Encoding
data = pd.get_dummies(data, columns=['CategoryColumn'],
drop_first=True)
```

-

---

11. Feature Engineering

You can create new features based on existing ones:

python
Copy code

```
# Create a new feature
```

```
data['NewFeature'] = data['Column1'] * data['Column2']

# Convert date column to year, month, and day
data['Year'] = data['Date'].dt.year
data['Month'] = data['Date'].dt.month
data['Day'] = data['Date'].dt.day
```

---

12. Outlier Detection

You can use statistical methods to identify and remove outliers:

python
Copy code

```
# Using Z-Score to detect outliers
from scipy.stats import zscore
z_scores = np.abs(zscore(data[['Column1', 'Column2']]))
outliers = (z_scores > 3).all(axis=1)
data_no_outliers = data[~outliers]
```

---

13. Summary of Insights

After conducting EDA, summarize key insights such as:

- Important relationships between variables.
- Presence of missing values and their treatment.
- Distribution of variables (normal, skewed, etc.).
- Outliers and their impact on the analysis.
- Correlations among features and possible multicollinearity.