Exploratory Data Analysis (EDA)

1. Types of Exploratory Data Analysis:

- 1.1 Univariate Analysis:
- 1.2 Bivariate Analysis:
- 1.3 Multivariate Analysis:
- 1.4 Continuous and Categorical Variables Analysis:

(1.4) Key Concepts:

- Distinguishing Variable Types: Continuous (Quantitative) vs. Categorical (Qualitative).
- Visualization Techniques: Different plots for each variable type.

(1.4) Key Points:

- Recognizing the nature of variables and selecting appropriate visualizations.
- Tailoring analysis based on variable types.

2. Steps of Exploratory Data Analysis:

2.1 Data Collection:

Key Points:

- Gathering data from diverse sources, ensuring completeness and accuracy.

2.2 Data Cleaning:

Key Concepts:

- Handling Missing Data: Imputation or removal.
- Duplicate Removal: Ensuring unique observations.
- Error Correction: Identifying and fixing inconsistencies.

2.2.1. Identifying Missing Values:

```
import pandas as pd

# Load your dataset
df = pd.read_csv('your_dataset.csv')

# Check for missing values in each column
missing_values = df.isnull().sum()
print(missing_values)
```

2.2.2. Imputation:

```
# Replace missing values with the mean
df.fillna(df.mean(), inplace=True)

# Replace missing values in a specific column with the medi
an
df['column_name'].fillna(df['column_name'].median(), inplac
e=True)
```

2.2.3. Deletion:

```
# Remove rows with any missing values
df.dropna(inplace=True)

# Remove columns with any missing values
df.dropna(axis=1, inplace=True)
```

2.2.4. Interpolation:

```
# Linear interpolation for missing values
df.interpolate(method='linear', inplace=True)
```

2.2.5. Predictive Modeling:

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split

# Assuming 'target_column' is the column with missing value
s
train_data = df.dropna(subset=['target_column'])
test_data = df[df['target_column'].isnull()]

X = train_data.drop('target_column', axis=1)
y = train_data['target_column']

model = RandomForestRegressor()
model.fit(X, y)

# Predict missing values
predicted_values = model.predict(test_data.drop('target_col
umn', axis=1))

# Fill missing values with predictions
df.loc[df['target_column'].isnull(), 'target_column'] = pre
dicted_values
```

2.2.6. Create Missingness Indicator:

```
# Add a binary column indicating missing values in 'column_
name'
df['column_name_missing'] = df['column_name'].isnull().asty
pe(int)
```

These are basic examples, and you should adapt them based on your specific dataset and requirements. Always consider the context and nature of your data when choosing a method.

Key Points:

- Creating a clean, reliable dataset for analysis.

2.3 Descriptive Statistics:

Key Concepts:

- Central Tendency Measures: Mean, Median, Mode.
- Variability Measures: Range, Variance, Standard Deviation.

Descriptive statistics help summarize and describe the main features of a dataset. Here are some codes in Python using the pandas library for basic descriptive statistics in Exploratory Data Analysis (EDA):

2.3.1. Summary Statistics:

```
# Display basic summary statistics for numerical columns
summary_stats = df.describe()
print(summary_stats)
```

2.3.2. Mean, Median, Mode:

```
# Mean of a column
mean_value = df['numerical_column'].mean()

# Median of a column
median_value = df['numerical_column'].median()

# Mode of a column
mode_value = df['categorical_column'].mode()[0]

print(f"Mean: {mean_value}")
print(f"Median: {median_value}")
print(f"Mode: {mode_value}")
```

2.3.3. Variance and Standard Deviation:

```
# Variance of a column
variance_value = df['numerical_column'].var()

# Standard deviation of a column
std_deviation_value = df['numerical_column'].std()

print(f"Variance: {variance_value}")
print(f"Standard Deviation: {std_deviation_value}")
```

2.3.4. Skewness and Kurtosis:

```
# Skewness of a column
skewness_value = df['numerical_column'].skew()

# Kurtosis of a column
kurtosis_value = df['numerical_column'].kurt()

print(f"Skewness: {skewness_value}")
print(f"Kurtosis: {kurtosis_value}")
```

2.3.5. Count of Unique Values:

```
# Count of unique values in a column
unique_values_count = df['categorical_column'].nunique()

print(f"Number of unique values: {unique_values_count}")
```

These codes provide a basic overview of descriptive statistics that you can use to understand the characteristics of your dataset. Depending on your data, you may want to explore additional statistics and visualizations to gain deeper insights.

Key Points:

- Summarizing and understanding the basic statistical properties of the data.

2.4 Univariate Analysis:

Key Concepts:

- Histograms: Visualizing the distribution of a single variable.
- Box Plots: Identifying outliers and understanding quartiles.

Univariate analysis focuses on analyzing a single variable at a time to understand its distribution and characteristics. Here are some codes in Python using the pandas and matplotlib/seaborn libraries for univariate analysis in Exploratory Data Analysis (EDA):

2.4.1. Histogram:

```
import matplotlib.pyplot as plt

# Plot a histogram for a numerical column
plt.hist(df['numerical_column'], bins=20, color='skyblue',
edgecolor='black')
plt.title('Histogram of Numerical Column')
plt.xlabel('Values')
plt.ylabel('Frequency')
plt.show()
```

2.4.2. Box Plot:

```
import seaborn as sns

# Create a box plot for a numerical column
sns.boxplot(x=df['categorical_column'], y=df['numerical_col
umn'])
plt.title('Box Plot of Numerical Column by Category')
plt.show()
```

2.4.3. Count Plot (Bar Plot for Categorical Variables):

```
# Plot a count plot for a categorical column
sns.countplot(x='categorical_column', data=df, palette='Set
3')
plt.title('Count Plot of Categorical Column')
plt.show()
```

2.4.4. Kernel Density Estimate (KDE) Plot:

```
# Plot a KDE plot for a numerical column
sns.kdeplot(df['numerical_column'], fill=True, color='skybl
ue')
plt.title('Kernel Density Estimate (KDE) Plot')
plt.xlabel('Values')
plt.ylabel('Density')
plt.show()
```

2.4.5. Pie Chart (for small categorical variables):

```
# Plot a pie chart for a categorical column
plt.pie(df['categorical_column'].value_counts(), labels=df
['categorical_column'].unique(), autopct='%1.1f%%', startan
gle=90)
plt.title('Pie Chart of Categorical Column')
plt.show()
```

These are basic examples of univariate analysis. Depending on your dataset and the type of variable you are analyzing, you may want to explore other visualizations and summary statistics to gain a deeper understanding of the distribution and characteristics of individual variables.

Key Points:

- Revealing patterns and characteristics of individual variables.

2.5 Bivariate Analysis:

Key Concepts:

- Correlation Coefficient: Quantifying the strength and direction of a relationship.
- Scatter Plots: Visualizing relationships between two variables.

Bivariate analysis involves exploring the relationship between two variables. Here are some codes in Python using the pandas, matplotlib, and seaborn libraries for bivariate analysis in Exploratory Data Analysis (EDA):

2.5.1. Scatter Plot:

```
import matplotlib.pyplot as plt

# Create a scatter plot for two numerical columns
plt.scatter(df['numerical_column_1'], df['numerical_column_
2'], alpha=0.5)
plt.title('Scatter Plot of Numerical Column 1 vs Numerical
Column 2')
plt.xlabel('Numerical Column 1')
plt.ylabel('Numerical Column 2')
plt.show()
```

2.5.2. Line Plot:

```
# Create a line plot for two numerical columns
plt.plot(df['numerical_column_1'], df['numerical_column_
2'])
plt.title('Line Plot of Numerical Column 1 vs Numerical Col
umn 2')
plt.xlabel('Numerical Column 1')
plt.ylabel('Numerical Column 2')
plt.show()
```

2.5.3. Heatmap (Correlation Matrix):

```
import seaborn as sns

# Calculate the correlation matrix
correlation_matrix = df.corr()

# Plot a heatmap of the correlation matrix
sns.heatmap(correlation_matrix, annot=True, cmap='coolwar
m', fmt=".2f")
plt.title('Correlation Matrix Heatmap')
plt.show()
```

2.5.4. Pair Plot:

```
# Create a pair plot for multiple numerical columns
sns.pairplot(df[['numerical_column_1', 'numerical_column_
2', 'numerical_column_3']])
plt.suptitle('Pair Plot of Numerical Columns', y=1.02)
plt.show()
```

2.5.5. Box Plot:

```
import seaborn as sns

# Create a box plot for a numerical column grouped by a cat
egorical column
sns.boxplot(x=df['categorical_column'], y=df['numerical_col
umn'])
plt.title('Box Plot of Numerical Column by Category')
plt.show()
```

2.5.6. Violin Plot:

```
import seaborn as sns

# Create a violin plot for a numerical column grouped by a
categorical column
sns.violinplot(x=df['categorical_column'], y=df['numerical_
column'])
plt.title('Violin Plot of Numerical Column by Category')
plt.show()
```

These examples demonstrate various ways to explore the relationship between two variables in your dataset. Customize the visualizations based on your specific needs and the nature of your data.

Key Points:

- Identifying associations between pairs of variables.

2.6 Multivariate Analysis:

Key Concepts:

- Heatmaps: Visualizing correlations among multiple variables.
- Pair Plots: Understanding joint distributions.

Multivariate analysis involves examining relationships among three or more variables simultaneously. Here are some codes in Python using the pandas, matplotlib, and seaborn libraries for multivariate analysis in Exploratory Data Analysis (EDA):

1. Pair Plot with Hue (Color by a Categorical Variable):

```
import seaborn as sns

# Create a pair plot with hue for a categorical column
sns.pairplot(df, hue='categorical_column', palette='Set1')
plt.suptitle('Pair Plot with Hue', y=1.02)
plt.show()
```

2. 3D Scatter Plot:

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Create a 3D scatter plot for three numerical columns
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(df['numerical_column_1'], df['numerical_column_
2'], df['numerical_column_3'], c='skyblue', s=50)
ax.set_xlabel('Numerical Column 1')
ax.set_ylabel('Numerical Column 2')
ax.set_zlabel('Numerical Column 3')
plt.title('3D Scatter Plot of Numerical Columns')
plt.show()
```

3. Parallel Coordinates Plot:

```
from pandas.plotting import parallel_coordinates

# Create a parallel coordinates plot for selected columns
selected_columns = ['numerical_column_1', 'numerical_column
_2', 'numerical_column_3', 'categorical_column']
parallel_coordinates(df[selected_columns], 'categorical_col
umn', colormap='viridis')
plt.title('Parallel Coordinates Plot')
plt.show()
```

4. Bubble Chart:

```
# Create a bubble chart with three numerical columns and si
ze representing a fourth numerical column
plt.scatter(df['numerical_column_1'], df['numerical_column_
2'], s=df['numerical_column_3']*10, c='skyblue', alpha=0.5)
plt.title('Bubble Chart of Numerical Columns')
plt.xlabel('Numerical Column 1')
plt.ylabel('Numerical Column 2')
plt.show()
```

5. 3D Surface Plot:

```
import plotly.express as px

# Create a 3D surface plot for three numerical columns
fig = px.scatter_3d(df, x='numerical_column_1', y='numerica
l_column_2', z='numerical_column_3', color='categorical_col
umn')
fig.update_layout(scene=dict(zaxis=dict(range=[df['numerica
l_column_3'].min(), df['numerical_column_3'].max()])))
fig.show()
```

These codes provide examples of multivariate analysis using various visualization techniques. Customize them based on your specific dataset and analysis goals. Note that some visualizations, such as 3D plots, may require additional libraries like `plotly` for interactive plots.

Key Points:

- Investigating interactions and dependencies among multiple variables.

2.7 Feature Engineering:

Key Concepts:

- Creating New Features: Enhancing predictive power.
- Handling Categorical Variables: Encoding or creating dummy variables.

Feature engineering involves creating new features or transforming existing ones to improve the performance of machine learning models. Here are some common feature engineering techniques with example codes in Python using the pandas library:

1. Handling Missing Values:

```
# Filling missing values with the mean of a numerical colum
n
df['numerical_column'].fillna(df['numerical_column'].mean
(), inplace=True)

# Filling missing values in a categorical column with the m
ode
df['categorical_column'].fillna(df['categorical_column'].mo
de()[0], inplace=True)
```

2. Creating Dummy Variables (One-Hot Encoding):

```
# Convert categorical variables into dummy/indicator variab
les
df = pd.get_dummies(df, columns=['categorical_column'], dro
p_first=True)
```

3. Binning (Discretization):

```
# Create bins for a numerical column
bins = [0, 25, 50, 75, 100]
labels = ['Low', 'Medium', 'High', 'Very High']
df['binned_column'] = pd.cut(df['numerical_column'], bins=b
ins, labels=labels, include_lowest=True)
```

4. Feature Scaling (Standardization or Min-Max Scaling):

```
from sklearn.preprocessing import StandardScaler, MinMaxSca
ler

# Standardization
scaler = StandardScaler()
df[['numerical_column_1', 'numerical_column_2']] = scaler.f
it_transform(df[['numerical_column_1', 'numerical_column_
2']])

# Min-Max Scaling
minmax_scaler = MinMaxScaler()
df['numerical_column'] = minmax_scaler.fit_transform(df[['n
umerical_column']])
```

5. Log Transformation:

```
# Apply log transformation to a numerical column
df['log_transformed_column'] = np.log1p(df['numerical_colum
n'])
```

6. Creating Interaction Terms:

```
# Create an interaction term between two numerical columns
df['interaction_term'] = df['numerical_column_1'] * df['num
erical_column_2']
```

7. Feature Extraction (Principal Component Analysis - PCA):

```
from sklearn.decomposition import PCA

# Apply PCA to reduce dimensionality
pca = PCA(n_components=2)
pca_result = pca.fit_transform(df[['numerical_column_1', 'n
umerical_column_2']])
df['pca_component_1'] = pca_result[:, 0]
df['pca_component_2'] = pca_result[:, 1]
```

8. Time-Based Features:

```
# Extract time-based features from a datetime column
df['year'] = df['datetime_column'].dt.year
df['month'] = df['datetime_column'].dt.month
df['day'] = df['datetime_column'].dt.day
df['weekday'] = df['datetime_column'].dt.weekday
```

These are just a few examples of feature engineering techniques. The specific techniques you choose will depend on the nature of your data and the goals of your machine learning task. Always validate the impact of feature engineering on your model's performance through careful evaluation.

Key Points:

- Improving the dataset for better predictive modeling.

2.8 Outlier Detection:

Key Concepts:

- Identifying Outliers: Using statistical measures or visualization techniques.

Outliers are data points that significantly differ from the majority of the data. Detecting and handling outliers is crucial in data preprocessing. Here are some common techniques for outlier detection with example codes in Python:

1. Z-Score Method:

```
from scipy import stats

# Calculate the z-scores for a numerical column
z_scores = stats.zscore(df['numerical_column'])

# Set a threshold for considering data points as outliers
(e.g., z-score > 3 or < -3)
threshold = 3
outliers = (np.abs(z_scores) > threshold)

# Identify and print outlier values
outlier_values = df['numerical_column'][outliers]
print("Outliers:", outlier_values)
```

2. IQR (Interquartile Range) Method:

```
# Calculate the IQR for a numerical column
Q1 = df['numerical_column'].quantile(0.25)
Q3 = df['numerical_column'].quantile(0.75)
IQR = Q3 - Q1

# Set a threshold for considering data points as outliers
(e.g., Q1 - 1.5 * IQR or Q3 + 1.5 * IQR)
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Identify and print outlier values
outliers = ((df['numerical_column'] < lower_bound) | (df['n
umerical_column'] > upper_bound))
outlier_values = df['numerical_column'][outliers]
print("Outliers:", outlier_values)
```

3. Isolation Forest:

```
from sklearn.ensemble import IsolationForest

# Fit Isolation Forest to a numerical column
isolation_forest = IsolationForest(contamination=0.05)
df['outlier_flag'] = isolation_forest.fit_predict(df[['nume
rical_column']])

# Identify and print outlier values
outliers = df[df['outlier_flag'] == -1]['numerical_column']
print("Outliers:", outliers)
```

4. Visualizing Box Plots:

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

# Create a box plot to visualize potential outliers
sns.boxplot(x=df['numerical_column'])
plt.title('Box Plot of Numerical Column')
plt.show()
```