Data Pre-processing and Feature Engineering

Data Pre-processing and Feature Engineering are two crucial steps in the data analysis pipeline.

These steps ensure that raw data is meticulously prepared and transformed into a format suitable for analysis. Feature Engineering, in particular, involves the art of crafting new features to enhance a model's capabilities and help it reach its full potential. 💡

Step-by-Step Exploratory Data Analysis (EDA) 📊:

Step 1: Import Python Libraries 🐍

```
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Step 2: Reading Dataset 📄

```
# Load the dataset into a Pandas DataFrame
df = pd.read_csv('your_dataset.csv')

# Read Excel file
df_excel = pd.read_excel('your_file.xlsx', sheet_name='Sheet1')

# Read JSON file
df_json = pd.read_json('your_file.json')

# Read SQL File
from sqlalchemy import create_engine
engine = create_engine('sqlite:///your_database.db')    #### Create a SQLite database engine
df_sql = pd.read_sql('SELECT * FROM your_table', engine)     #### Read data from a SQL table
```

```
# Read Parquet file
df_parquet = pd.read_parquet('your_file.parquet')

# Read HDF5 file
df_hdf5 = pd.read_hdf('your_file.h5', key='your_key')

# Read Feather file
df_feather = pd.read_feather('your_file.feather')


# Read fixed-width file
column_widths = [10, 15, 20]        #### Define column widths
df_fixed_width = pd.read_fwf('your_file.txt', widths=column_widths)

# Read data from the clipboard
df_clipboard = pd.read_clipboard()

# Read data from a URL
df_url = pd.read_csv('https://example.com/your_data.csv')
```

Step 3: Data Reduction 📉

Data reduction involves handling missing values and removing unnecessary columns.

```
1. Handling Missing Values:
# Remove rows with missing values
df = df.dropna()

# Impute missing values with mean or median
df['numeric_column'].fillna(df['numeric_column'].mean(), inplace=True)
df['numeric_column'].fillna(df['numeric_column'].median(),
inplace=True)

2. Removing Unnecessary Columns:
# Drop columns with a high percentage of missing values (e.g., 70%
threshold)
df = df.dropna(thresh=len(df) * 0.7, axis=1)

## Drop columns with low variance:
```

```
from sklearn.feature_selection import VarianceThreshold
threshold = 0.1    ### Set a threshold for variance
selector = VarianceThreshold(threshold)
df_reduced = selector.fit_transform(df)

# Drop duplicate columns
df = df.T.drop_duplicates().T

# Remove columns with constant values
df = df.loc[:, df.nunique() != 1]
```

Step 4: Feature Engineering 🚀

Feature engineering aims to create new features or modify existing ones to improve model performance.

```
1. Creating New Features:
# Create a new feature by adding two existing features
df['new_feature_sum'] = df['feature1'] + df['feature2']

# Create a new feature by multiplying two existing features
df['new_feature_product'] = df['feature1'] * df['feature2']

# Create a new feature by calculating the mean of a group
df['mean_feature_by_category'] = df.groupby('category')
['numeric_feature'].transform('mean')

# Binning/Discretization
bins = [0, 25, 50, 75, 100]
labels = ['Group1', 'Group2', 'Group3', 'Group4']
df['binned_feature'] = pd.cut(df['numeric_feature'], bins=bins,
labels=labels)

2. Transforming Existing Features:
#  Log-transform a numeric feature
df['log_transformed_feature'] = np.log1p(df['numeric_feature'])

#  Min-Max scaling
df['scaled_feature'] = (df['numeric_feature'] -
df['numeric_feature'].min()) / (df['numeric_feature'].max() -
```

```
df['numeric_feature'].min())

# One-hot encoding for a categorical feature
df_encoded = pd.get_dummies(df, columns=['categorical_feature'])
```

Step 5: Creating Features 🎨

Generate additional features to provide more insights into the data.

```
1. Extracting Information from Date Columns:
#  Extract year, month, and day from a date column
df['year'] = pd.to_datetime(df['date_column']).dt.year
df['month'] = pd.to_datetime(df['date_column']).dt.month
df['day'] = pd.to_datetime(df['date_column']).dt.day

#  Extract day of the week from a date column
df['day_of_week'] = pd.to_datetime(df['date_column']).dt.dayofweek

2. Creating Interaction Features:
#  Create a new feature by multiplying two existing features
df['interaction_feature'] = df['feature1'] * df['feature2']

#  Create a new feature by dividing two existing features
df['ratio_feature'] = df['feature1'] / df['feature2']

3. Text Data:
#  Count the number of words in a text column
df['word_count'] = df['text_column'].apply(lambda x:
len(str(x).split()))

4. Creating Indicator Features:
#  Create a binary indicator for a specific condition
df['high_value_indicator'] = np.where(df['numeric_column'] > 100, 1, 0)

5. Feature Scaling:
#  Min-Max scaling
df['scaled_feature'] = (df['numeric_feature'] -
df['numeric_feature'].min()) / (df['numeric_feature'].max() -
df['numeric_feature'].min())
```

Step 6: Data Cleaning/Wrangling 🧹

Clean and preprocess the data to handle outliers and anomalies.

```
1. Handling Outliers using Interquartile Range (IQR):
# Calculate the Interquartile Range (IQR)
Q1 = df['numeric_column'].quantile(0.25)
Q3 = df['numeric_column'].quantile(0.75)
IQR = Q3 - Q1

# Remove outliers based on IQR
df = df[(df['numeric_column'] >= Q1 - 1.5 * IQR) &
(df['numeric_column'] <= Q3 + 1.5 * IQR)]

2. Handling Missing Values:
# Impute missing values using mean or median
df['numeric_column'].fillna(df['numeric_column'].mean(), inplace=True)
df['numeric_column'].fillna(df['numeric_column'].median(),
inplace=True)

3. Handling Categorical Data:
# One-hot encoding for a categorical variable
df_encoded = pd.get_dummies(df, columns=['categorical_variable'])

4. Handling Text Data:
#Text cleaning for NLP:
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

nltk.download('stopwords')


nltk.download('punkt')

# Define a function for text cleaning
def clean_text(text):
    stop_words = set(stopwords.words('english'))
    tokens = word_tokenize(text)
    tokens = [word.lower() for word in tokens if word.isalpha() and
word.lower() not in stop_words]
```

```
    return ' '.join(tokens)

# Apply text cleaning to a text column
df['cleaned_text'] = df['text_column'].apply(clean_text)

5. Handling Inconsistent Data:
# Standardize categorical labels
df['categorical_column'] =
df['categorical_column'].replace({'categoryA': 'Category_A',
'categoryB': 'Category_B'})
```

Step 7: EDA Exploratory Data Analysis 📈

```
1. Visualizing Data Distributions:
# Create a histogram
sns.histplot(df['numeric_column'], kde=True)
plt.title('Distribution of Numeric Column')
plt.show()

2. Statistical Summary:
# Display summary statistics
summary_stats = df.describe()
print(summary_stats)

3. Visualizing Categorical Data:
# Create a bar plot for a categorical variable
sns.countplot(x='categorical_column', data=df)
plt.title('Distribution of Categorical Column')
plt.show()

4. Box Plots for Outlier Detection:
# Create a box plot for a numeric variable
sns.boxplot(x='categorical_column', y='numeric_column', data=df)
plt.title('Box Plot of Numeric Column by Category')
plt.show()

5. Pair Plots for Multivariate Analysis:
# Create a pair plot for multiple numeric variables
sns.pairplot(df[['numeric_column1', 'numeric_column2',
'numeric_column3']])
```

```
plt.title('Pair Plot of Numeric Columns')
plt.show()

6. Correlation Heatmap:
# Create a correlation heatmap
correlation_matrix = df.corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Heatmap')
plt.show()

7. Distribution of Categorical Variables across Numerical Categories:
# Create a bar plot for a categorical variable vs. a numeric variable
sns.barplot(x='categorical_column', y='numeric_column', data=df)
plt.title('Bar Plot of Numeric Column by Category')
plt.show()

8. Violin Plots:
# Create a violin plot for a categorical variable vs. a numeric
variable
sns.violinplot(x='categorical_column', y='numeric_column', data=df)
plt.title('Violin Plot of Numeric Column by Category')
plt.show()

9. Scatter Plots:
# Create a scatter plot for numeric-numeric relationships
sns.scatterplot(x='numeric_column1', y='numeric_column2', data=df)
plt.title('Scatter Plot of Numeric Column1 vs. Numeric Column2')
plt.xlabel('Numeric Column1')
plt.ylabel('Numeric Column2')
plt.show()
```

Key Takeaways 🚀:

- **Data Cleaning:**
  - Handle missing values through imputation or removal.
  - Detect and address outliers to maintain data integrity.
- **Univariate Analysis:**
  - Explore individual variables using statistical measures and visualizations.
  - Utilize histograms, box plots, and summary statistics.

- **Bivariate and Multivariate Analysis:**
  - Analyze relationships between pairs of variables and multiple variables.
  - Use scatter plots, pair plots, and correlation matrices for insights.
- **Feature Engineering:**
  - Create or transform features for more meaningful insights.
  - Normalize or scale features for consistent numerical representation.
- **Statistical Testing:**
  - Conduct statistical tests to validate hypotheses.
  - Examples include t-tests, chi-square tests, and ANOVA.
- **Data Visualization:**
  - Enhance understanding with visualizations using Matplotlib, Seaborn, and Plotly.
  - Create interactive dashboards for dynamic exploration.
- **Data Distribution:**
  - Understand data distribution and assess its fit with known distributions.
  - Use probability plots and statistical tests for distribution analysis.
- **Time Series Analysis:**
  - Consider seasonality, trends, and autocorrelation for time-series data.
  - Use time series plots and decomposition to analyze temporal patterns.
- **Interactive Dashboards:**
  - Build interactive dashboards with tools like Streamlit or Dash.
  - Allow users to interact and customize analyses.
- **Documentation:**
  - Document findings, insights, and EDA steps.
  - Provide clear explanations for collaboration and knowledge sharing.
- **Handling Categorical Data:**
  - Analyze frequency distributions and proportions for categorical variables.
  - Use bar charts and pie charts for visualizing categorical data.
- **Robustness and Reproducibility:**
  - Script analysis steps for robust and reproducible EDA.
  - Use Jupyter Notebooks or scripts to document and reproduce analyses.
- **Interactive Widgets:**

  - Implement widgets for parameter tuning and exploration in Jupyter Notebooks.
  - Tools like `ipywidgets` can enhance interactivity.
- **Domain Knowledge:**
  - Combine statistical analysis with domain knowledge.
  - Consult domain experts to validate interpretations and insights.
- **Continuous Iteration:**
  - EDA is an iterative process; revisit analyses for deeper insights.
  - Be open to adjusting approaches based on new findings.
