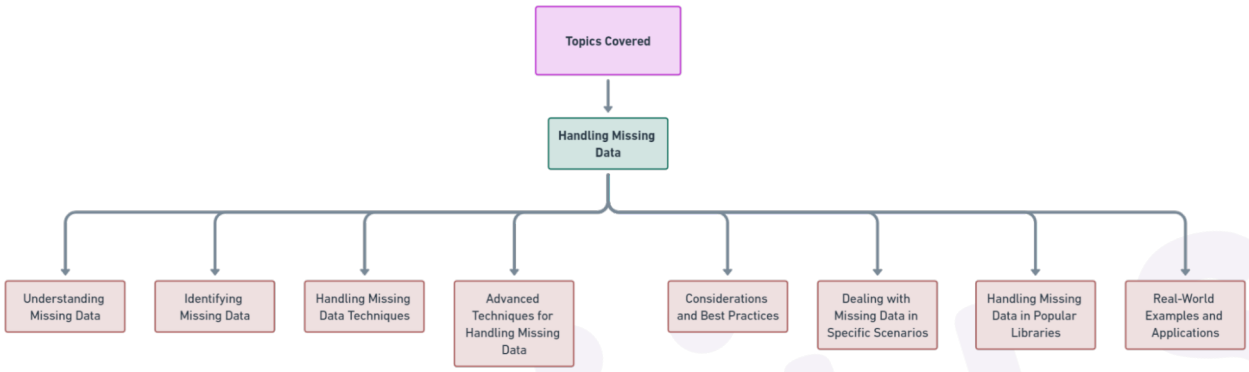
PW SKILLS

Lesson Plan

Handling Missing Data

Topic to covered:

- Understanding Missing Data
- Identifying Missing Data
- Handling Missing Data Techniques
- Advanced Techniques for Handling Missing Data
- Considerations and Best Practices
- Dealing with Missing Data in Specific Scenarios
- Handling Missing Data in Popular Libraries
- Real-World Examples and Applications



Understanding Missing Data

- Missing data refers to the absence of values in a dataset. It can occur due to various reasons such as human error, sensor malfunction, or data corruption during collection.
- Understanding missing data is crucial because it can affect the performance of machine learning models.

Code

```
import pandas as pd
import numpy as np

# Create a sample dataset with missing values
data = {
    'A': [1, 2, np.nan, 4, 5],
    'B': [3, np.nan, 7, 8, 9],
    'C': [np.nan, 12, 13, 14, 15],
    'D': [16, 17, 18, np.nan, 20]
}

# Convert the dictionary to a Pandas DataFrame
df = pd.DataFrame(data)
print("Original DataFrame:")
print(df)
```

Output::

```
Original DataFrame:
     A    B     C     D
0  1.0  3.0   NaN  16.0
1  2.0  NaN  12.0  17.0
2  NaN  7.0  13.0  18.0
3  4.0  8.0  14.0   NaN
4  5.0  9.0  15.0  20.0
```

Identifying Missing Data

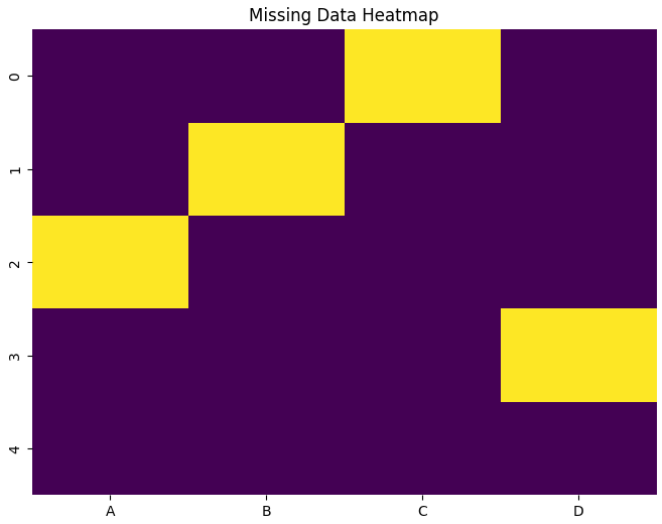
- Identifying missing data involves recognizing and quantifying the presence of missing values within a dataset.
- Visualization tools like heatmaps or summary statistics help in understanding the patterns of missingness.

Code

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

# Visualize missing data using a heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(df.isnull(), cmap='viridis', cbar=False)
plt.title('Missing Data Heatmap')
plt.show()
```

Output::



Handling Missing Data Techniques

- Handling missing data involves strategies to deal with missing values in a dataset.
- Common techniques include removal of rows/columns with missing values or imputation methods such as mean, median, or mode replacement.

Code

```
# Impute missing values with mean
df_imputed = df.fillna(df.mean())
print("Imputed DataFrame (using mean):")
print(df_imputed)
```

Output::

```
Imputed DataFrame (using mean):
     A     B     C      D
0  1.0  3.00  13.5  16.00
1  2.0  6.75  12.0  17.00
2  3.0  7.00  13.0  18.00
3  4.0  8.00  14.0  17.75
4  5.0  9.00  15.0  20.00
```

Advanced Techniques for Handling Missing Data

- Advanced techniques consider missing data mechanisms (MCAR, MAR, MNAR) and employ more sophisticated methods like K-Nearest Neighbors (KNN) imputation or multiple imputation techniques (e.g., MICE).

Code

```
from sklearn.impute import KNNImputer

# Initialize KNN imputer with k neighbors
imputer = KNNImputer(n_neighbors=2)

# Apply KNN imputation to the dataset
df_knn_imputed = pd.DataFrame(imputer.fit_transform(df),
columns=df.columns)
print("Imputed DataFrame (using KNN imputation):")
print(df_knn_imputed)
```

Output::

```
Imputed DataFrame (using KNN imputation):
     A    B     C     D
0  1.0  3.0  12.5  16.0
1  2.0  5.0  12.0  17.0
2  3.0  7.0  13.0  18.0
3  4.0  8.0  14.0  19.0
4  5.0  9.0  15.0  20.0
```

Considerations and Best Practices

- Considerations and best practices involve understanding the impact of missing data on model performance and choosing appropriate strategies to handle missing values without introducing bias or affecting the overall quality of the dataset.

Code

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# Split data into train and test sets
X = df_imputed.drop('D', axis=1)  # Assuming 'D' is the target variable
y = df_imputed['D']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Fit a model (Example: Linear Regression) and evaluate
model = LinearRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

# Calculate Mean Squared Error (MSE)
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error (after handling missing data):", mse)
```

Output::

```
Mean Squared Error (after handling missing data): 0.042694172900057246
```

Dealing with Missing Data in Specific Scenarios

- Different scenarios, such as time series data or categorical data, may require specialized techniques for handling missing values.
- For instance, time series data might involve forward or backward filling, while categorical data might require mode imputation.

Code

```
# Assuming 'A' is time-related and needs forward fill
df_time_series = df.fillna(method='ffill')
print("Time Series Data with Forward Fill:")
print(df_time_series)
```

Output::

```
Time Series Data with Forward Fill:
     A    B     C     D
0  1.0  3.0   NaN  16.0
1  2.0  3.0  12.0  17.0
2  2.0  7.0  13.0  18.0
3  4.0  8.0  14.0  18.0
4  5.0  9.0  15.0  20.0
```

Handling Missing Data in Popular Libraries

- Popular libraries like Pandas provide efficient methods to handle missing data, such as fillna(), dropna(), and specialized functions for imputation.

Code

```
# Drop rows with any missing values
df_dropped = df.dropna()
print("DataFrame after dropping rows with missing values:")
print(df_dropped)
```

Output::

```
DataFrame after dropping rows with missing values:
     A    B     C     D
4  5.0  9.0  15.0  20.0
```

Real-World Examples and Applications

Example: Medical Data Analysis

- Consider a medical dataset where patient information is collected for analysis.
- Missing data in this scenario could significantly impact diagnostic or predictive models.
- Handling missing values becomes critical to ensure accurate analysis and decision-making in healthcare.

Code

```
import pandas as pd
import numpy as np

# Creating a sample medical dataset with missing values
data = {
    'Age': [32, np.nan, 45, 28, 60],
    'Blood_Pressure': [120, 130, np.nan, 140, 115],
    'BMI': [25.5, 29.8, 27.2, np.nan, 26.1],
    'Cholesterol': [200, 220, 190, np.nan, 210],
    'Diabetes': ['Yes', 'No', 'Yes', 'No', np.nan]
}

medical_df = pd.DataFrame(data)
print("Medical DataFrame:")
print(medical_df)
```

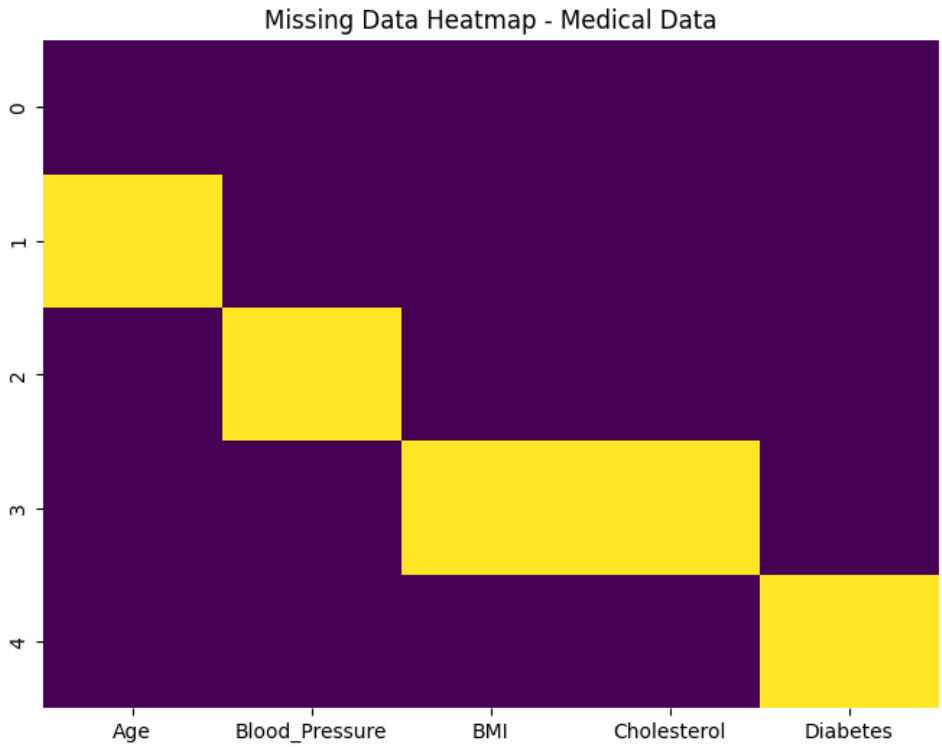
Output::

```
Medical DataFrame:
    Age  Blood_Pressure   BMI  Cholesterol Diabetes
0  32.0           120.0  25.5        200.0      Yes
1   NaN           130.0  29.8        220.0       No
2  45.0             NaN  27.2        190.0      Yes
3  28.0           140.0   NaN          NaN       No
4  60.0           115.0  26.1        210.0      NaN
```

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

# Visualize missing data Before imputation
plt.figure(figsize=(8, 6))
sns.heatmap(medical_df.isnull(), cmap='viridis', cbar=False)
plt.title('Missing Data Heatmap - Medical Data')
plt.show()
```

Output::




```
# Impute missing values with mean and mode
medical_df['Age'].fillna(medical_df['Age'].mean(), inplace=True)
medical_df['Blood_Pressure'].fillna(medical_df['Blood_Pressure'].mean(),
inplace=True)
medical_df['BMI'].fillna(medical_df['BMI'].mean(), inplace=True)
medical_df['Cholesterol'].fillna(medical_df['Cholesterol'].mean(),
inplace=True)
medical_df['Diabetes'].fillna(medical_df['Diabetes'].mode()[0],
inplace=True)

print("Imputed Medical DataFrame (using mean and mode):")
print(medical_df)
```

Output::

```
Imputed Medical DataFrame (using mean and mode):
     Age  Blood_Pressure    BMI  Cholesterol Diabetes
0  32.00          120.00  25.50        200.0      Yes
1  41.25          130.00  29.80        220.0       No
2  45.00          126.25  27.20        190.0      Yes
3  28.00          140.00  27.15        205.0       No
4  60.00          115.00  26.10        210.0       No
```

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

# Visualize missing data after imputation
plt.figure(figsize=(8, 6))
sns.heatmap(medical_df.isnull(), cmap='viridis', cbar=False)
plt.title('Missing Data Heatmap - Medical Data')
plt.show()

# Perform further analysis or model building using the imputed dataset
```

Output::