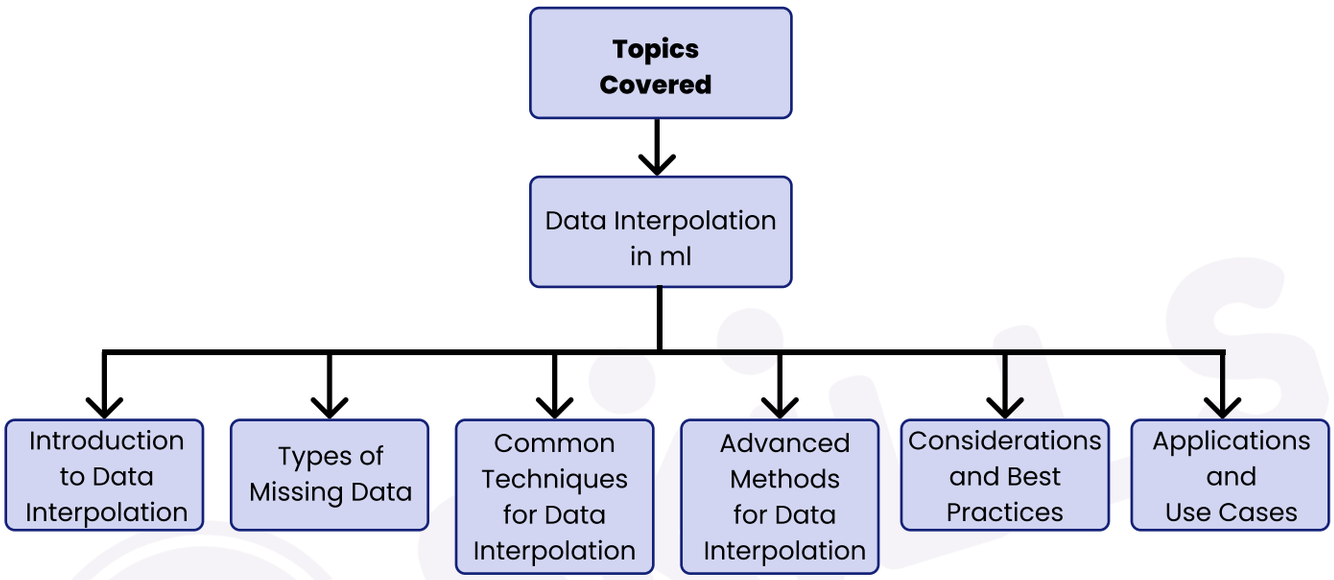
PW SKILLS

Lesson Plan

Data Interpolation

Topic to covered:

- Introduction to Data Interpolation
- Types of Missing Data
- Common Techniques for Data Interpolation
- Advanced Methods for Data Interpolation
- Considerations and Best Practices
- Applications and Use Cases



Introduction to Data Interpolation

- Data Interpolation refers to the technique of estimating values for missing or incomplete data points within a dataset.
- In the realm of Machine Learning, dealing with missing data is a crucial step before building models, as most algorithms cannot handle missing values.
- Understanding and addressing missing data is essential to ensure the accuracy and reliability of machine learning models

Types of Missing Data

1. Missing Completely at Random (MCAR):

- MCAR occurs when the missingness of data points is entirely random and unrelated to any other variables in the dataset.
- There's no systematic reason for the absence of data.
- Example: A weather station malfunction causing random data loss in recorded temperature.

2. Missing at Random (MAR):

- MAR refers to the missingness pattern that depends on observed variables but not on the missing values themselves.
- The missingness is related to other observed variables in the dataset.
- Example: Missing income information in a survey based on the age of participants.

3. Missing Not at Random (MNAR):

- MNAR occurs when the missingness is related to the missing values themselves, and it's not random or systematically related to observed or unobserved variables.
- Example: People with high income are less likely to disclose their income in a survey.

Common Techniques Of Data Interpretation

Code

```
import numpy as np
import pandas as pd

# Creating a sample DataFrame with missing values
data = {
    'A': [1, 2, np.nan, 4, 5],
    'B': [1, np.nan, 3, 4, 5],
    'C': [np.nan, 2, 3, np.nan, 5],
    'D': [1, 2, 3, 4, np.nan]
}
df = pd.DataFrame(data)
print("Original DataFrame:")
print(df)
```

Output:

```
Original DataFrame:
     A    B    C    D
0  1.0  1.0  NaN  1.0
1  2.0  NaN  2.0  2.0
2  NaN  3.0  3.0  3.0
3  4.0  4.0  NaN  4.0
4  5.0  5.0  5.0  NaN
```

Mean/Median/Mode Imputation

- Mean Imputation: This involves replacing missing values with the mean of the available data in that particular column.

Code:

```
# Common techniques for data interpolation

# Mean Imputation
df_mean = df.fillna(df.mean())
print("\nAfter Mean Imputation:")
print(df_mean)
```

Output:

```
After Mean Imputation:
     A     B         C    D
0  1.0  1.00  3.333333  1.0
1  2.0  3.25  2.000000  2.0
2  3.0  3.00  3.000000  3.0
3  4.0  4.00  3.333333  4.0
4  5.0  5.00  5.000000  2.5
```

- **Median Imputation:** Similar to mean imputation, but uses the median value instead of the mean.

Code:

```
median_values = df.median()
# median Imputation
df_median = df.fillna(median_values)  # Fill missing values with respective column medians

print("DataFrame with Median Imputation:\n", df_median)
```

Output:

```
DataFrame with Median Imputation:

      A    B   C    D
0  1.0  1.0  3.0  1.0
1  2.0  3.5  2.0  2.0
2  3.0  3.0  3.0  3.0
3  4.0  4.0  3.0  4.0
4  5.0  5.0  5.0  2.5
```

- **Mode Imputation:** Replaces missing values with the mode (most frequently occurring value) of the respective column.

Code:

```
# Imputing missing values with mode
mode_values = df.mode().iloc[0]

df_mode = df.fillna(mode_values)  # Fill missing values with respective column modes

print("DataFrame with Mode Imputation:\n", df_mode)
```

Output:

```
DataFrame with Mode Imputation:
      A    B    C    D
0  1.0  1.0  2.0  1.0
1  2.0  1.0  2.0  2.0
2  1.0  3.0  3.0  3.0
3  4.0  4.0  2.0  4.0
4  5.0  5.0  5.0  1.0
```

Forward Filling and Backward Filling

- **Forward Filling:** Propagates the last observed non-missing value forward to fill missing values.

Code:

```
# Forward filling missing values
df_ffill = df.ffill()
print("Forward Fill:\n", df_ffill)
```

Output:

```
 Forward Fill:
      A    B    C    D
0  1.0  1.0  NaN  1.0
1  2.0  1.0  2.0  2.0
2  2.0  3.0  3.0  3.0
3  4.0  4.0  3.0  4.0
4  5.0  5.0  5.0  4.0
```

- **Forward Filling:** Carries the next observed non-missing value backward to fill missing values.

Code:

```
# Backward filling missing values
df_bfill = df.bfill()
print("\nBackward Fill:\n", df_bfill)
```

```
Backward Fill:
     A    B    C    D
0  1.0  1.0  2.0  1.0
1  2.0  3.0  2.0  2.0
2  4.0  3.0  3.0  3.0
3  4.0  4.0  5.0  4.0
4  5.0  5.0  5.0  NaN
```

Linear Interpolation

- Estimates missing values by creating a linear relationship between adjacent observed values.
- Applicable to ordered data, time-series, or datasets with continuous variables.

Code:

```
# Linear Interpolation
df_linear = df.interpolate(method='linear')
print("\nAfter Linear Interpolation:")
print(df_linear)
```

Output:

```
After Linear Interpolation:
     A    B    C    D
0  1.0  1.0  NaN  1.0
1  2.0  2.0  2.0  2.0
2  3.0  3.0  3.0  3.0
3  4.0  4.0  4.0  4.0
4  5.0  5.0  5.0  4.0
```

Multiple Imputation by Chained Equations (MICE)

- MICE is an iterative imputation technique that generates multiple sets of imputed values by cycling through each variable, filling in missing values based on observed values in other variables.
- MICE operates by predicting missing values in a variable using observed values from other variables iteratively, updating missing values for each variable multiple times based on the imputed values of the other variables.

Code:

```
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer

# Sample data with missing values
data = np.array([[1, 2, np.nan],
                 [4, np.nan, 6],
                 [7, 8, 9]])

imputer = IterativeImputer(max_iter=10, random_state=0)  # Initialize MICE imputer
filled_data = imputer.fit_transform(data)  # Impute missing values
print("Imputed Data using MICE:")
print(filled_data)
```

Code:

```
Imputed Data using MICE:
[[1.         2.         3.00203274]
 [4.         4.99796925 6.        ]
 [7.         8.         9.        ]]
```

Considerations and Best Practices

When dealing with missing data, there are several considerations and best practices to keep in mind:

1. Understanding the Nature of Missing Data:

- Analyze the patterns of missing data (MCAR, MAR, MNAR) to choose appropriate imputation methods.

2. Pros and Cons of Interpolation Techniques:

- Each interpolation method has its strengths and weaknesses; understanding these helps in choosing the most suitable technique for the dataset.

3. Impact on Model Performance:

- Evaluate the impact of imputation on the performance of machine learning models; imputing data may introduce biases or affect model accuracy.

Applications and Use Cases

- Data interpolation finds applications in various domains like healthcare (patient data), finance (stock market data), and IoT (sensor data).
- It's crucial in scenarios where missing data can't be ignored but needs to be accurately filled for analysis or modeling purposes.

Real-Time Code

```
import pandas as pd
import numpy as np
from fancyimpute import KNN

# Generating a synthetic healthcare dataset with missing values
np.random.seed(42)
num_patients = 100

# Generating arrays with NaN values to simulate missing data
age = np.random.randint(20, 70, num_patients).astype(float)
blood_pressure = np.random.randint(80, 180, num_patients).astype(float)
cholesterol = np.random.randint(120, 300, num_patients).astype(float)

# Define indices to introduce missing values
missing_age_indices = np.random.choice(num_patients, size=5, replace=False)
missing_blood_pressure_indices = np.random.choice(num_patients, size=5, replace=False)
missing_cholesterol_indices = np.random.choice(num_patients, size=5, replace=False)

# Introduce missing values by setting selected indices to NaN
age[missing_age_indices] = np.nan
blood_pressure[missing_blood_pressure_indices] = np.nan

# Creating a DataFrame for healthcare data with missing values

health_data = pd.DataFrame({
    'age': age,
    'blood_pressure': blood_pressure,
    'cholesterol': cholesterol
})

# Impute missing values using KNN imputation
imputer = KNN(k=5)
health_data_imputed = pd.DataFrame(imputer.fit_transform(health_data),
columns=health_data.columns)

print("Healthcare data before imputation:")
print(health_data.head(10))

print("\nHealthcare data after KNN imputation:")
print(health_data_imputed.head(10))
```

```
Imputing row 1/100 with 0 missing, elapsed time: 0.005
Healthcare data before imputation:
    age  blood_pressure  cholesterol
0  58.0            88.0        215.0
1  48.0           167.0        299.0
2  34.0            80.0        232.0
3  62.0            87.0        181.0
4   NaN           167.0        171.0
5  40.0           142.0          NaN
6  58.0            90.0        158.0
7  38.0           160.0        249.0
8  42.0            87.0        250.0
9  30.0           114.0        232.0

Healthcare data after KNN imputation:
         age  blood_pressure  cholesterol
0  58.000000            88.0   215.000000
1  48.000000           167.0   299.000000
2  34.000000            80.0   232.000000
3  62.000000            87.0   181.000000
4  39.867528           167.0   171.000000
5  40.000000           142.0   290.999936
6  58.000000            90.0   158.000000
7  38.000000           160.0   249.000000
8  42.000000            87.0   250.000000
9  30.000000           114.0   232.000000
```

PW SKILLS

THANK YOU !