# Lab Experiment 4 : Data Mining

Apply different missing values handing methods namely

* Ignore the tuple,
* Use a global constant to fill in the missing value,
* Use a measure of central tendency for the attribute (e.g., the mean or median) to fill in the missing value,
* Use the attribute mean or median for all samples belonging to the same class as the given tuple,
* Use the most probable value to fill in the missing value on your datasets.
* Further, address the issue of noisy data points still pertaining in the datasets even after handling the missing values using
  + Binning and
  + Regression methods.
* Analyze the effect of different techniques on dataset in terms of statistical parameters such as central tendency and dispersion.

### Importing Libraries

import pandas as pd  
import numpy as np

### Load the dataset

# url = "https://archive.ics.uci.edu/ml/machine-learning-databases/heart-disease/processed.cleveland.data"  
# column\_names = ['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach', 'exang', 'oldpeak', 'slope', 'ca', 'thal', 'target']  
# heart\_df = pd.read\_csv(url, header=None, names=column\_names, na\_values='?')  
  
url = "../heart/heart.csv"  
heart\_df = pd.read\_csv(url)  
  
# Display the first few rows and summary statistics  
heart\_df

age sex cp trestbps chol fbs restecg thalach exang oldpeak \  
0 52 1 0 125 212 0 1 168 0 1.0   
1 53 1 0 140 203 1 0 155 1 3.1   
2 70 1 0 145 174 0 1 125 1 2.6   
3 61 1 0 148 203 0 1 161 0 0.0   
4 62 0 0 138 294 1 1 106 0 1.9   
... ... ... .. ... ... ... ... ... ... ...   
1020 59 1 1 140 221 0 1 164 1 0.0   
1021 60 1 0 125 258 0 0 141 1 2.8   
1022 47 1 0 110 275 0 0 118 1 1.0   
1023 50 0 0 110 254 0 0 159 0 0.0   
1024 54 1 0 120 188 0 1 113 0 1.4   
  
 slope ca thal target   
0 2 2 3 0   
1 0 0 3 0   
2 0 0 3 0   
3 2 1 3 0   
4 1 3 2 0   
... ... .. ... ...   
1020 2 0 2 1   
1021 1 1 3 0   
1022 1 1 2 0   
1023 2 0 2 1   
1024 1 1 3 0   
  
[1025 rows x 14 columns]

### Remove rows having null

# Ignore the tuples with missing values  
heart\_df\_ignored = heart\_df.dropna()  
print("Dataset after ignoring tuples with missing values:")  
print(heart\_df\_ignored.info())

Dataset after ignoring tuples with missing values:  
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 1025 entries, 0 to 1024  
Data columns (total 14 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 age 1025 non-null int64   
 1 sex 1025 non-null int64   
 2 cp 1025 non-null int64   
 3 trestbps 1025 non-null int64   
 4 chol 1025 non-null int64   
 5 fbs 1025 non-null int64   
 6 restecg 1025 non-null int64   
 7 thalach 1025 non-null int64   
 8 exang 1025 non-null int64   
 9 oldpeak 1025 non-null float64  
 10 slope 1025 non-null int64   
 11 ca 1025 non-null int64   
 12 thal 1025 non-null int64   
 13 target 1025 non-null int64   
dtypes: float64(1), int64(13)  
memory usage: 112.2 KB  
None

### Fill Missing with -1

heart\_df\_global\_constant = heart\_df.fillna(-1)  
  
print("Dataset after using global constant to fill missing values:")  
print(heart\_df\_global\_constant.info())

Dataset after using global constant to fill missing values:  
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 1025 entries, 0 to 1024  
Data columns (total 14 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 age 1025 non-null int64   
 1 sex 1025 non-null int64   
 2 cp 1025 non-null int64   
 3 trestbps 1025 non-null int64   
 4 chol 1025 non-null int64   
 5 fbs 1025 non-null int64   
 6 restecg 1025 non-null int64   
 7 thalach 1025 non-null int64   
 8 exang 1025 non-null int64   
 9 oldpeak 1025 non-null float64  
 10 slope 1025 non-null int64   
 11 ca 1025 non-null int64   
 12 thal 1025 non-null int64   
 13 target 1025 non-null int64   
dtypes: float64(1), int64(13)  
memory usage: 112.2 KB  
None

### Use a Measure of Central Tendency (Mean/Median) to fill missing values

# Fill missing values with the mean  
heart\_df\_mean = heart\_df.fillna(heart\_df.mean())  
  
# Fill missing values with the median  
heart\_df\_median = heart\_df.fillna(heart\_df.median())  
  
print("Dataset after using mean to fill missing values:")  
print(heart\_df\_mean.info())  
  
print("Dataset after using median to fill missing values:")  
print(heart\_df\_median.info())

Dataset after using mean to fill missing values:  
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 1025 entries, 0 to 1024  
Data columns (total 14 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 age 1025 non-null int64   
 1 sex 1025 non-null int64   
 2 cp 1025 non-null int64   
 3 trestbps 1025 non-null int64   
 4 chol 1025 non-null int64   
 5 fbs 1025 non-null int64   
 6 restecg 1025 non-null int64   
 7 thalach 1025 non-null int64   
 8 exang 1025 non-null int64   
 9 oldpeak 1025 non-null float64  
 10 slope 1025 non-null int64   
 11 ca 1025 non-null int64   
 12 thal 1025 non-null int64   
 13 target 1025 non-null int64   
dtypes: float64(1), int64(13)  
memory usage: 112.2 KB  
None  
Dataset after using median to fill missing values:  
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 1025 entries, 0 to 1024  
Data columns (total 14 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 age 1025 non-null int64   
 1 sex 1025 non-null int64   
 2 cp 1025 non-null int64   
 3 trestbps 1025 non-null int64   
 4 chol 1025 non-null int64   
 5 fbs 1025 non-null int64   
 6 restecg 1025 non-null int64   
 7 thalach 1025 non-null int64   
 8 exang 1025 non-null int64   
 9 oldpeak 1025 non-null float64  
 10 slope 1025 non-null int64   
 11 ca 1025 non-null int64   
 12 thal 1025 non-null int64   
 13 target 1025 non-null int64   
dtypes: float64(1), int64(13)  
memory usage: 112.2 KB  
None

### Fill missing values using the mean or median for samples belonging to the same class (e.g., the same target class).

# Fill missing values using the mean of the same class  
heart\_df\_class\_mean = heart\_df.copy()  
for column in heart\_df.columns:  
 heart\_df\_class\_mean[column].fillna(heart\_df.groupby('target')[column].transform('mean'), inplace=True)  
  
# Fill missing values using the median of the same class  
heart\_df\_class\_median = heart\_df.copy()  
for column in heart\_df.columns:  
 heart\_df\_class\_median[column].fillna(heart\_df.groupby('target')[column].transform('median'), inplace=True)  
  
print("Dataset after using class mean to fill missing values:")  
print(heart\_df\_class\_mean.info())  
  
print("Dataset after using class median to fill missing values:")  
print(heart\_df\_class\_median.info())

Dataset after using class mean to fill missing values:  
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 1025 entries, 0 to 1024  
Data columns (total 14 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 age 1025 non-null int64   
 1 sex 1025 non-null int64   
 2 cp 1025 non-null int64   
 3 trestbps 1025 non-null int64   
 4 chol 1025 non-null int64   
 5 fbs 1025 non-null int64   
 6 restecg 1025 non-null int64   
 7 thalach 1025 non-null int64   
 8 exang 1025 non-null int64   
 9 oldpeak 1025 non-null float64  
 10 slope 1025 non-null int64   
 11 ca 1025 non-null int64   
 12 thal 1025 non-null int64   
 13 target 1025 non-null int64   
dtypes: float64(1), int64(13)  
memory usage: 112.2 KB  
None  
Dataset after using class median to fill missing values:  
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 1025 entries, 0 to 1024  
Data columns (total 14 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 age 1025 non-null int64   
 1 sex 1025 non-null int64   
 2 cp 1025 non-null int64   
 3 trestbps 1025 non-null int64   
 4 chol 1025 non-null int64   
 5 fbs 1025 non-null int64   
 6 restecg 1025 non-null int64   
 7 thalach 1025 non-null int64   
 8 exang 1025 non-null int64   
 9 oldpeak 1025 non-null float64  
 10 slope 1025 non-null int64   
 11 ca 1025 non-null int64   
 12 thal 1025 non-null int64   
 13 target 1025 non-null int64   
dtypes: float64(1), int64(13)  
memory usage: 112.2 KB  
None

C:\Users\debat\AppData\Local\Temp\ipykernel\_15200\535031063.py:4: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.  
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.  
  
For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.  
  
  
 heart\_df\_class\_mean[column].fillna(heart\_df.groupby('target')[column].transform('mean'), inplace=True)  
C:\Users\debat\AppData\Local\Temp\ipykernel\_15200\535031063.py:9: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.  
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.  
  
For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.  
  
  
 heart\_df\_class\_median[column].fillna(heart\_df.groupby('target')[column].transform('median'), inplace=True)

### Fill missing values with the most probable value, which could be inferred through methods like regression, k-NN, or similar techniques.

# Fill missing values with the most probable value (mode)  
heart\_df\_mode = heart\_df.apply(lambda x: x.fillna(x.mode()[0]))  
  
print("Dataset after using mode to fill missing values:")  
print(heart\_df\_mode.info())

Dataset after using mode to fill missing values:  
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 1025 entries, 0 to 1024  
Data columns (total 14 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 age 1025 non-null int64   
 1 sex 1025 non-null int64   
 2 cp 1025 non-null int64   
 3 trestbps 1025 non-null int64   
 4 chol 1025 non-null int64   
 5 fbs 1025 non-null int64   
 6 restecg 1025 non-null int64   
 7 thalach 1025 non-null int64   
 8 exang 1025 non-null int64   
 9 oldpeak 1025 non-null float64  
 10 slope 1025 non-null int64   
 11 ca 1025 non-null int64   
 12 thal 1025 non-null int64   
 13 target 1025 non-null int64   
dtypes: float64(1), int64(13)  
memory usage: 112.2 KB  
None

## Handling Noisy Data Points

### Binning Method

Binning is a simple technique that smooths noisy data by grouping it into bins and then replacing the values within each bin with a representative value (such as the mean, median, or boundaries).

Steps for Binning:

* Equal-width Binning: Divides the range of the data into equal-sized intervals.
* Equal-frequency Binning: Divides the data into intervals that each contain approximately the same number of data points.

We'll apply equal-width binning to smooth the chol (cholesterol) attribute, which might have noisy data.

# Apply equal-width binning on 'chol' attribute  
heart\_df\_binned = heart\_df\_mean.copy()  
  
# Define the number of bins  
num\_bins = 4  
  
# Binning using pandas cut function  
heart\_df\_binned['chol\_binned'] = pd.cut(heart\_df\_binned['chol'], bins=num\_bins, labels=False)  
  
# Replace original 'chol' values with bin means  
bin\_means = heart\_df\_binned.groupby('chol\_binned')['chol'].mean()  
heart\_df\_binned['chol'] = heart\_df\_binned['chol\_binned'].map(bin\_means)  
  
print("Dataset after applying equal-width binning on 'chol' attribute:")  
print(heart\_df\_binned[['chol', 'chol\_binned']].head())

Dataset after applying equal-width binning on 'chol' attribute:  
 chol chol\_binned  
0 204.80083 0  
1 204.80083 0  
2 204.80083 0  
3 204.80083 0  
4 276.40619 1

### Regression Method

Regression can be used to predict and smooth out noisy data by fitting a regression model to the data. We'll use linear regression to predict the trestbps (resting blood pressure) attribute based on other attributes and replace its values with the predicted ones to smooth the data.

from sklearn.linear\_model import LinearRegression  
  
# Prepare data for regression  
regression\_df = heart\_df\_mean.dropna(subset=['trestbps'])  
X = regression\_df.drop(['trestbps', 'target'], axis=1)  
y = regression\_df['trestbps']  
  
# Fit a linear regression model  
regressor = LinearRegression()  
regressor.fit(X, y)  
  
# Predict 'trestbps' values  
heart\_df\_regression = heart\_df\_mean.copy()  
predicted\_trestbps = regressor.predict(heart\_df\_regression.drop(['trestbps', 'target'], axis=1))  
heart\_df\_regression['trestbps'] = predicted\_trestbps  
  
print("Dataset after applying regression on 'trestbps' attribute:")  
print(heart\_df\_regression[['trestbps']].head())

Dataset after applying regression on 'trestbps' attribute:  
 trestbps  
0 127.606252  
1 143.082032  
2 137.413752  
3 128.398309  
4 138.927470

## Analyze the Effect of Different Techniques

def analyze\_statistics(df, attribute):  
 """Calculate and display central tendency and dispersion statistics for a given attribute."""  
 mean = df[attribute].mean()  
 median = df[attribute].median()  
 mode = df[attribute].mode()[0]  
 range\_val = df[attribute].max() - df[attribute].min()  
 variance = df[attribute].var()  
 std\_dev = df[attribute].std()  
   
 print(f"Statistics for {attribute}:")  
 print(f"Mean: {mean}, Median: {median}, Mode: {mode}")  
 print(f"Range: {range\_val}, Variance: {variance}, Standard Deviation: {std\_dev}\n")  
  
# Analyze the 'chol' and 'trestbps' attributes across different techniques  
print("After Ignoring Tuples:")  
analyze\_statistics(heart\_df\_ignored, 'chol')  
analyze\_statistics(heart\_df\_ignored, 'trestbps')  
  
print("After Filling with Global Constant:")  
analyze\_statistics(heart\_df\_global\_constant, 'chol')  
analyze\_statistics(heart\_df\_global\_constant, 'trestbps')  
  
print("After Filling with Mean:")  
analyze\_statistics(heart\_df\_mean, 'chol')  
analyze\_statistics(heart\_df\_mean, 'trestbps')  
  
print("After Binning (chol):")  
analyze\_statistics(heart\_df\_binned, 'chol')  
  
print("After Regression (trestbps):")  
analyze\_statistics(heart\_df\_regression, 'trestbps')

After Ignoring Tuples:  
Statistics for chol:  
Mean: 246.0, Median: 240.0, Mode: 204  
Range: 438, Variance: 2661.787109375, Standard Deviation: 51.59251020618206  
  
Statistics for trestbps:  
Mean: 131.61170731707318, Median: 130.0, Mode: 120  
Range: 106, Variance: 306.835409679878, Standard Deviation: 17.516718005376408  
  
After Filling with Global Constant:  
Statistics for chol:  
Mean: 246.0, Median: 240.0, Mode: 204  
Range: 438, Variance: 2661.787109375, Standard Deviation: 51.59251020618206  
  
Statistics for trestbps:  
Mean: 131.61170731707318, Median: 130.0, Mode: 120  
Range: 106, Variance: 306.835409679878, Standard Deviation: 17.516718005376408  
  
After Filling with Mean:  
Statistics for chol:  
Mean: 246.0, Median: 240.0, Mode: 204  
Range: 438, Variance: 2661.787109375, Standard Deviation: 51.59251020618206  
  
Statistics for trestbps:  
Mean: 131.61170731707318, Median: 130.0, Mode: 120  
Range: 106, Variance: 306.835409679878, Standard Deviation: 17.516718005376408  
  
After Binning (chol):  
Statistics for chol:  
Mean: 246.0, Median: 276.4061895551257, Mode: 276.4061895551257  
Range: 359.1991701244813, Variance: 1992.448015590442, Standard Deviation: 44.636845941334634  
  
After Regression (trestbps):  
Statistics for trestbps:  
Mean: 131.61170731707318, Median: 131.40491906597907, Mode: 120.60533396201558  
Range: 32.82357606343098, Variance: 44.10907254282059, Standard Deviation: 6.641466144069439

### Observation and Conclusion

Based on the results, we can draw conclusions about how different missing value handling and noise reduction techniques affect the dataset in terms of central tendency (mean, median, mode) and dispersion (range, variance, standard deviation).