ksml4.com/mindmap

classical missing data strategies

imputation (filling in values)

Mean/Median/Mode Imputation

- Description: Replace missing values with the mean, median, or mode of the non-missing values in a column.

- Best for: Numeric data without strong skew.

```
import pandas as pd
from sklearn.impute import SimpleImputer

# Assuming df is a pandas DataFrame
mean_imputer = SimpleImputer(strategy="mean")  # For mean imputation
median_imputer = SimpleImputer(strategy="median")  # For median imputation
mode_imputer = SimpleImputer(strategy="most_frequent")  # For mode imputation

df['column_name'] = mean_imputer.fit_transform(df[['column_name']])
```

K-Nearest Neighbors (KNN) Imputation

Uses the values of the k-nearest neighbors to impute missing values.

- Best for: Small to medium datasets where similar instances are expected to have similar values.

```
from sklearn.impute import KNNImputer

knn_imputer = KNNImputer(n_neighbors=5)
df_imputed = knn_imputer.fit_transform(df)
```

Constant Imputation

- Description: Replace missing values with a constant value, like 0 or "Unknown" for categorical data.

- Best for: Data where missing values represent a meaningful category.

```
constant_imputer = SimpleImputer(strategy="constant", fill_value=0)
df['column_name'] = constant_imputer.fit_transform(df[['column_name']])
```

Multivariate Imputation by Chained Equations (MICE)

Iteratively imputes missing values by modeling each variable with missing values as a function of other variables.

- Best for: Data with a complex correlation structure.

```
import pandas as pd
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer

iterative_imputer = IterativeImputer()
df_imputed = iterative_imputer.fit_transform(df)
```

Regression Imputation

Regresses missing values based on other available features.

- Best for: Data where the relationship between variables can be accurately modeled.

```
from sklearn.linear_model import LinearRegression

# Example for single column imputation
df_non_missing = df[df['column_name'].notnull()]
df_missing = df[df['column_name'].isnull()]

X_train = df_non_missing.drop('column_name', axis=1)
y_train = df_non_missing['column_name']
X_missing = df_missing.drop('column_name', axis=1)

regressor = LinearRegression()
regressor.fit(X_train, y_train)
df.loc[df['column_name'].isnull(), 'column_name'] = regressor.predict(X_missing)
```

Use a model that can handle missing data

Decision Tree

Handle missing values by considering available data during splits

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Sample data with missing values
X = np.array([[1, 2], [np.nan, 4], [3, 5]])
y = np.array([0, 1, 0])

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train the Decision Tree
clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)

# Predict
y_pred = clf.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
```

Random Forest

handles missing values by using surrogate splits. This means it finds alternative splitting rules when a value is missing.

```
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Sample data with missing values
X = np.array([[1, 2], [np.nan, 4], [3, 5], [6, np.nan], [8, 7], [np.nan, 10]])
y = np.array([0, 1, 0, 1, 0, 1])

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train the Random Forest
clf = RandomForestClassifier()
clf.fit(X_train, y_train)

# Predict
y_pred = clf.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
```

XGBoost

Handles missing values natively, learning the best direction during training

```
import xgboost as xgb
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Sample data with missing values
X = np.array([[1, 2], [np.nan, 4], [3, 5]])
y = np.array([0, 1, 0])

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train XGBoost
model = xgb.XGBClassifier()
model.fit(X_train, y_train)

# Predict
y_pred = model.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
```