Data Cleaning Techniques

Urban Data Analysis Course

Week- 4

Handling Missing Values and Outliers

1.1 Understanding Missing Data

Missing data occurs when no value is stored for a feature in an observation. This can be due to various reasons:

- Data entry errors
- Incomplete data collection
- Data corruption
- Unavailable information

Handling Missing Values and Outliers

1.2 Techniques for Handling Missing Data

Deletion Methods

- **Listwise deletion (Complete Case Analysis):** Remove rows where any value is missing.
- **Pairwise deletion:** Use available data when performing specific analyses instead of dropping entire rows.

Imputation Methods

- **Mean, Median, Mode Imputation:** Replace missing values with the mean, median, or mode of the column.
- **Forward or Backward Fill:** Fill missing values using previous or next values.
- **Interpolation:** Estimate missing values using linear, polynomial, or time-series interpolation.
- **K-Nearest Neighbors (KNN) Imputation:** Predict missing values using the nearest neighbors.
- **Multiple Imputation:** Generate multiple datasets with different plausible values for missing data and aggregate results.

Handling Missing Values and Outliers

1.3 Identifying and Handling Outliers

Outliers are extreme values that differ significantly from other observations. They can result from measurement errors, data entry errors, or genuine anomalies.

Detection Methods:

Z-score: Outliers are values beyond 3 standard deviations from the mean.

Interquartile Range (IQR): Outliers are values beyond 1.5 times the IQR.

Boxplots & Scatterplots: Visual methods to identify extreme values.

Isolation Forests & DBSCAN: Machine learning-based outlier detection.

Handling Outliers:

Win-sorizing: Replace extreme values with a defined percentile value.

Trimming: Remove extreme outliers.

Transformation: Apply log, square root, or Box-Cox transformation to reduce outlier impact.

Grouping and Filtering Data

Using Pandas

Grouping is useful when analyzing subsets of data or performing aggregations.

Aggregation Functions: mean(), sum(), count(), min(), max(), std(), etc.

Custom aggregation functions using apply().

Grouping and Aggregating

- groupby('category_column'): Groups data by unique values in category_column.
- agg({'numeric_column': 'mean'}): Computes the mean of numeric_column for each group.

```
df.groupby('category_column').agg({'numeric_column': 'mean'})
```

✓ 0.0s Python

Filtering Data

- **df[df['column'] > 100]**
- df['column'] > 100: Creates a boolean mask where values greater than100 are True.
- df[...]: Filters the DataFrame based on this condition.

- **df[(df['column1'] > 100) & (df['column2'] < 50)]**
- &: Logical "AND" operator ensuring both conditions are met.

Cleaning Categorical and Numerical Data

Standardizing Categorical Data

```
df['category'] = df['category'].str.lower().str.strip()
```

- .str.lower(): Converts text to lowercase.
- .str.strip(): Removes leading/trailing whitespace.

Fixing Typos

```
df['category'] = df['category'].replace({'catgory': 'category'})
```

- .replace(): Corrects misspellings.

Cleaning Numerical Data

```
df['column'] = df['column'].abs()
```

- .abs(): Converts negative values to positive.

```
df['price'] = df['price'].str.replace('$', '').astype(float)
```

- .str.replace('$', ''): Removes $ symbols.
- .astype(float): Converts the column to numeric.

Handling Duplicate Data

```
df.duplicated().sum()
```

- .duplicated(): Identifies duplicate rows.
- .sum(): Counts the number of duplicates.

```
df.drop_duplicates(inplace=True)
```

- .drop_duplicates(): Removes duplicate rows from the DataFrame.