Data preparation is a crucial step in the machine learning pipeline that involves cleaning, transforming and organizing the dataset to make it suitable for training a model. The housing data that we have is not of the data type - DataFrame, so we will be convert it into 'DataFrame'.

**Creating DataFrame**

We can create a dataframe type of data after loading the California Housing dataset using 'pd.DataFrame(data, columns)', which creates a DataFrame named dataset using the data and feature names. The columns parameter in the pd.DataFrame constructor is used to specify the column names.

```
[8] dataset = pd.DataFrame(data=housing.data, columns=housing.feature_names)
    dataset
```

| | MedInc | HouseAge | AveRooms | AveBedrms | Population | AveOccup | Latitude | Longitude |
|---|---|---|---|---|---|---|---|---|
| 0 | 8.3252 | 41.0 | 6.984127 | 1.023810 | 322.0 | 2.555556 | 37.88 | -122.23 |
| 1 | 8.3014 | 21.0 | 6.238137 | 0.971880 | 2401.0 | 2.109842 | 37.86 | -122.22 |
| 2 | 7.2574 | 52.0 | 8.288136 | 1.073446 | 496.0 | 2.802260 | 37.85 | -122.24 |
| 3 | 5.6431 | 52.0 | 5.817352 | 1.073059 | 558.0 | 2.547945 | 37.85 | -122.25 |
| 4 | 3.8462 | 52.0 | 6.281853 | 1.081081 | 565.0 | 2.181467 | 37.85 | -122.25 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... |

**Adding a new column to the dataframe**

The code dataset['Price'] = housing.target is adding a new column named 'Price' to the DataFrame dataset and populating it with the values from the 'target' variable of the housing dataset.

- **dataset['Price']:** This creates a new column called 'Price' in the DataFrame dataset.
- **housing.target:** This is assumed to be the target variable from the California Housing dataset, which likely represents the median house value.
- This is often done to consolidate the target variable with the features, making it more convenient for further analysis or model training.

| | MedInc | HouseAge | AveRooms | AveBedrms | Population | AveOccup | Latitude | Longitude | Price |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 8.3252 | 41.0 | 6.984127 | 1.023810 | 322.0 | 2.555556 | 37.88 | -122.23 | 4.526 |
| 1 | 8.3014 | 21.0 | 6.238137 | 0.971880 | 2401.0 | 2.109842 | 37.86 | -122.22 | 3.585 |
| 2 | 7.2574 | 52.0 | 8.288136 | 1.073446 | 496.0 | 2.802260 | 37.85 | -122.24 | 3.521 |
| 3 | 5.6431 | 52.0 | 5.817352 | 1.073059 | 558.0 | 2.547945 | 37.85 | -122.25 | 3.413 |
| 4 | 3.8462 | 52.0 | 6.281853 | 1.081081 | 565.0 | 2.181467 | 37.85 | -122.25 | 3.422 |

**Getting summary of the dataframe**

The dataset.info() method in pandas is used to print a concise summary of a DataFrame, including information about the data types, non-null values and memory usage.

The output of dataset.info() will include:

- The total number of entries (rows).
- The number of non-null values for each column.
- The data type of each column.
- The memory usage of the DataFrame.

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 9 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   MedInc      20640 non-null  float64
 1   HouseAge    20640 non-null  float64
 2   AveRooms    20640 non-null  float64
 3   AveBedrms   20640 non-null  float64
 4   Population  20640 non-null  float64
 5   AveOccup    20640 non-null  float64
 6   Latitude    20640 non-null  float64
```

**Generating Descriptive Statistics**

The **dataset.describe()** method in pandas is used to generate descriptive statistics.

The output of dataset.describe() will include:

- **Count:** Number of non-null values
- **Mean:** Average value
- **Std:** Standard deviation, a measure of the amount of variation or dispersion
- **Min:** Minimum value
- **25%:** First quartile (25th percentile)
- **50%:** Median (50th percentile)
- **75%:** Third quartile (75th percentile)
- **Max:** Maximum value

This summary helps you quickly grasp the distribution of each numerical variable in your dataset, providing insights into the central tendency and spread of the data. It's a useful tool for the initial exploration of your data during the data preparation phase.

```
[15] dataset.describe()
```

| | MedInc | HouseAge | AveRooms | AveBedrms | Population |
|---|---|---|---|---|---|
| count | 20640.000000 | 20640.000000 | 20640.000000 | 20640.000000 | 20640.000000 |
| mean | 3.870671 | 28.639486 | 5.429000 | 1.096675 | 1425.476744 |
| std | 1.899822 | 12.585558 | 2.474173 | 0.473911 | 1132.462122 |
| min | 0.499900 | 1.000000 | 0.846154 | 0.333333 | 3.000000 |
| 25% | 2.563400 | 18.000000 | 4.440716 | 1.006079 | 787.000000 |
| 50% | 3.534800 | 29.000000 | 5.229129 | 1.048780 | 1166.000000 |
| 75% | 4.743250 | 37.000000 | 6.052381 | 1.099526 | 1725.000000 |
| max | 15.000100 | 52.000000 | 141.909091 | 34.066667 | 35682.000000 |

**Check for missing values**

The **dataset.isnull()** method in pandas is used to check for missing values in the DataFrame. It returns a DataFrame of the same shape as the original, where each element is either True if the corresponding element in the original DataFrame is NaN (null or missing) or False otherwise.

*# Check for missing values in the entire dataset*

missing_values = dataset.isnull()

*# Display the DataFrame indicating True for missing values and False otherwise*

print(missing_values)

To get the total count of missing values for each column, you can use **dataset.isnull().sum()**

```python
missing_count_per_column = dataset.isnull().sum()

# Display the count of missing values
print(missing_count_per_column)
```

```
MedInc        0
HouseAge      0
AveRooms      0
AveBedrms     0
Population    0
AveOccup      0
Latitude      0
Longitude     0
Price         0
dtype: int64
```

**Other Important Functions**

**dataset.dropna()**

Drops rows with missing values.

# Drop rows with missing values

dataset_without_missing = dataset.dropna()

**dataset.fillna(value)**

Fills missing values with a specified value.

# Fill missing values with the mean of each column

dataset_filled = dataset.fillna(dataset.mean())

**dataset.drop(columns=['column_name'])**

Drops columns with missing values.

# Drop columns with missing values

dataset_no_missing_columns = dataset.drop(columns=['column_with_missing_values'])

These functions are essential for data cleaning and preparation, ensuring that missing values are appropriately handled before further analysis or model training.