missing_values

July 28, 2022

1 Missing Values Function Example

1.1 Dependency Package

[ ]:
```
import numpy as np
import pandas as pd
```

[ ]:
```
# read file to DataFrame to show example
df=pd.read_csv('../../data/data.csv')
```

1.2 missing_val_info

[ ]:
```
def missing_val_info(df):

    """
    Show the DataFrame with the column has missing value as index and the␣
 ↪missing counts
    and missing percent of each column.

    Parameters
    ----------
    df : DataFrame
        The DataFrame to report missing values.

    Returns
    -------
    res : DataFrame
        The DataFrame that reports missing values.
    """

    res = pd.DataFrame(np.sum(df.isnull())[np.sum(df.isnull())!=0],␣
 ↪columns=["Missing"])
    res['Missing Percent %'] = res['Missing']/df.shape[0]
    return res
```

[ ]:
```
missing_val_info(df)
```

[ ]:
```
                    Missing  Missing Percent %
entity_country           26           0.000013
entity_exch_code         40           0.000020
entity_industry          26           0.000013
entity_region            26           0.000013
entity_sector            22           0.000011
```

1.3 handle_missing

[ ]:
```python
def handle_missing(df, method="drop"):

    """
    Handle the missing value in the DataFrame with the method indicated.

    Parameters
    ----------
    df : DataFrame
        The DataFrame contains NaN valuess
    method : str
        The method to handle NaN values. The set of potential methods is:
        'drop' : drop all the rows that contains NaN value.
        'forward' : replace NaN value with the last value in the column.
        'backward' : replace NaN value with the next value in the column.

    Returns
    -------
     : DataFrame
        The DataFrame with all NaN values handled.
    """

    if method=="drop":
        return df.dropna()
    elif method=="forward":
        return df.fillna(method='ffill')
    elif method=="backward":
        return df.fillna(method='bfill')
```

[ ]:
```python
df.iloc[5,3] = np.NaN
```

[ ]:
```python
df.iloc[0:9,3]
```

[ ]:
```
0                     NYSE
1                     NYSE
2                     NYSE
3    London Stock Exchange
4    London Stock Exchange
5                      NaN
```

```
6                     NYSE
7                     NYSE
8                     NYSE
Name: entity_exchange, dtype: object
```

[ ]:
```
handle_missing(df, method="drop").iloc[0:9,3]
```

[ ]:
```
0                     NYSE
1                     NYSE
2                     NYSE
3    London Stock Exchange
4    London Stock Exchange
6                     NYSE
7                     NYSE
8                     NYSE
9                     NYSE
Name: entity_exchange, dtype: object
```

[ ]:
```
handle_missing(df, method="forward").iloc[0:9,3]
```

[ ]:
```
0                     NYSE
1                     NYSE
2                     NYSE
3    London Stock Exchange
4    London Stock Exchange
5    London Stock Exchange
6                     NYSE
7                     NYSE
8                     NYSE
Name: entity_exchange, dtype: object
```

[ ]:
```
handle_missing(df, method="backward").iloc[0:9,3]
```

[ ]:
```
0                     NYSE
1                     NYSE
2                     NYSE
3    London Stock Exchange
4    London Stock Exchange
5                     NYSE
6                     NYSE
7                     NYSE
8                     NYSE
Name: entity_exchange, dtype: object
```

1.4 impute

[ ]:
```
def impute(df, column, method="mean"):

    """
    Given a numeric column from a data frame, impute all the NaN value in the␣
 ↪column with the indicated method.

    Parameters
    ----------
    df : DataFrame
        The DataFrame contains numeric column with NaN values.
    column : str
        The column name of the numerical column to impute.
    method : str
        The method to impute the NaN value to. The set of potential methods is:
        'mean' : Replace all the NaN value with the mean value of the column.
        'median' : Replace all the NaN value with the median value of the␣
 ↪column.
        'mood' : Replace all the NaN value with the mood value of the column.

    Returns
    -------
     : Series
        The Series with all the NaN values imputed.
    """

    if method=="mean":
        return df[column].replace(np.nan, np.nanmean(df[column]))
    elif method=="median":
        return df[column].replace(np.nan, np.nanmedian(df[column]))
    elif method=="mood":
        return df[column].replace(np.nan, df[column].value_counts().index[0])
```

[ ]:
```
df.loc[1,"entity_relevance"] = np.NaN
```

[ ]:
```
df["entity_relevance"]
```

[ ]:
```
0          100.0
1            NaN
2          100.0
3           90.0
4           90.0
           ...
2038199    100.0
2038200    100.0
2038201    100.0
```

```
2038202     90.0
2038203    100.0
Name: entity_relevance, Length: 2038204, dtype: float64
```

```
[ ]: impute(df, "entity_relevance", method="mean")
```

```
[ ]: 0          100.000000
     1           90.665062
     2          100.000000
     3           90.000000
     4           90.000000
                   …
     2038199    100.000000
     2038200    100.000000
     2038201    100.000000
     2038202     90.000000
     2038203    100.000000
     Name: entity_relevance, Length: 2038204, dtype: float64
```

```
[ ]: impute(df, "entity_relevance", method="median")
```

```
[ ]: 0          100.0
     1          100.0
     2          100.0
     3           90.0
     4           90.0
                 …
     2038199    100.0
     2038200    100.0
     2038201    100.0
     2038202     90.0
     2038203    100.0
     Name: entity_relevance, Length: 2038204, dtype: float64
```

```
[ ]: impute(df, "entity_relevance", method="mood")
```

```
[ ]: 0          100.0
     1          100.0
     2          100.0
     3           90.0
     4           90.0
                 …
     2038199    100.0
     2038200    100.0
     2038201    100.0
     2038202     90.0
     2038203    100.0
```

```
Name: entity_relevance, Length: 2038204, dtype: float64
```

1.5 rolling_impute

[ ]:
```python
def rolling_impute(df, column, method="mean"):

    """
    Given a numeric column from a data frame, impute all the NaN value in the␣
↪column with the indicated method.

    Parameters
    ----------
    df : DataFrame
        The DataFrame contains numeric column with NaN values.
    column : str
        The column name of the numerical column to impute.
    method : str
        The method to impute the NaN value to. The set of potential methods is:
        'mean' : Replace all the NaN value with the mean value of all the␣
↪values prior to the NaN value.
        'median' : Replace all the NaN value with the median value of all the␣
↪values prior to the NaN value.
        'mood' : Replace all the NaN value with the mood value of all the␣
↪values prior to the NaN value.

    Returns
    -------
     : Series
        The Series with all the NaN values imputed.
    """
    indexes = np.where(df[column].isnull())[0]
    if method=="mean":
        indexes = np.where(df[column].isnull())[0]
        tmps = [np.nanmean(df.loc[0:index-1, column]) for index in indexes]
        temp_df = df.copy()
        temp_df.loc[indexes, column] = tmps
        return  temp_df[column]
    elif method=="median":
        indexes = np.where(df[column].isnull())[0]
        tmps = [np.nanmedian(df.loc[0:index-1, column]) for index in indexes]
        temp_df = df.copy()
        temp_df.loc[indexes, column] = tmps
        return  temp_df[column]
    elif method=="mood":
        indexes = np.where(df[column].isnull())[0]
        tmps = [df.loc[0:index, column].value_counts().index[0] for index in␣
↪indexes]
```

```
        temp_df = df.copy()
        temp_df.loc[indexes, column] = tmps
        return  temp_df[column]
```

[ ]:
```
df.loc[2, 'entity_relevance'] = np.NaN
df.loc[4, 'entity_relevance'] = np.NaN
df.loc[7, 'entity_relevance'] = np.NaN
df.loc[0:10, 'entity_relevance']
```

[ ]:
```
0     100.0
1     100.0
2       NaN
3      90.0
4       NaN
5      90.0
6      90.0
7       NaN
8      90.0
9      45.0
10     45.0
Name: entity_relevance, dtype: float64
```

[ ]:
```
rolling_impute(df, 'entity_relevance', 'mean')[0:10]
```

[ ]:
```
0    100.000000
1    100.000000
2    100.000000
3     90.000000
4     96.666667
5     90.000000
6     90.000000
7     94.000000
8     90.000000
9     45.000000
Name: entity_relevance, dtype: float64
```

[ ]:
```
rolling_impute(df, 'entity_relevance', 'median')[0:10]
```

[ ]:
```
0    100.0
1    100.0
2    100.0
3     90.0
4    100.0
5     90.0
6     90.0
7     90.0
8     90.0
```

```
9     45.0
Name: entity_relevance, dtype: float64
```

[ ]:
```
rolling_impute(df, 'entity_relevance', 'mood')[0:10]
```

[ ]:
```
0    100.0
1    100.0
2    100.0
3     90.0
4    100.0
5     90.0
6     90.0
7     90.0
8     90.0
9     45.0
Name: entity_relevance, dtype: float64
```

[ ]:
```

```