missing-values-22mcb1002

March 18, 2023

1 Missing Data Handling

[1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

[2]:
```
df = pd.read_csv('/content/drive/MyDrive/Data Analytics/heart.csv', na_values =␣
↪['??','???'])
df.head(5)
```

[2]:

	id	age	sex	cp	trestbps	chol	restecg	thalch	\
0	1	63	Male	typical angina	145.0	233.0	lv hypertrophy	150.0	
1	2	67	Male	asymptomatic	160.0	286.0	lv hypertrophy	108.0	
2	3	67	Male	asymptomatic	120.0	229.0	lv hypertrophy	129.0	
3	4	37	Male	non-anginal	130.0	250.0	normal	187.0	
4	5	41	Female	atypical angina	130.0	204.0	lv hypertrophy	172.0	

	exang	oldpeak	slope	ca
0	False	2.3	downsloping	0.0
1	True	1.5	flat	3.0
2	True	2.6	flat	2.0
3	False	3.5	downsloping	0.0
4	False	1.4	upsloping	0.0

[39]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 920 entries, 0 to 919
Data columns (total 12 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   id        920 non-null    int64
 1   age       920 non-null    int64
 2   sex       920 non-null    object
 3   cp        913 non-null    object
 4   trestbps  861 non-null    float64
 5   chol      890 non-null    float64
```

```
 6   restecg   918 non-null    object
 7   thalch    864 non-null    float64
 8   exang     865 non-null    object
 9   oldpeak   858 non-null    float64
 10  slope     611 non-null    object
 11  ca        309 non-null    float64
dtypes: float64(5), int64(2), object(5)
memory usage: 86.4+ KB
```

[40]:
```
df.describe()
```

[40]:

	id	age	trestbps	chol	thalch	oldpeak	ca
count	920.000000	920.000000	861.000000	890.000000	864.000000	858.000000	309.000000
mean	460.500000	53.510870	132.132404	199.130337	137.538194	0.878788	0.676375
std	265.725422	9.424685	19.066070	110.780810	25.940362	1.091226	0.935653
min	1.000000	28.000000	0.000000	0.000000	60.000000	-2.600000	0.000000
25%	230.750000	47.000000	120.000000	175.000000	120.000000	0.000000	0.000000
50%	460.500000	54.000000	130.000000	223.000000	140.000000	0.500000	0.000000
75%	690.250000	60.000000	140.000000	268.000000	157.000000	1.500000	1.000000
max	920.000000	77.000000	200.000000	603.000000	202.000000	6.200000	3.000000

[21]:
```
df.shape
```

[21]:
```
(920, 12)
```

[23]:
```
df_na = df.dropna()
df_na.shape

# In the original dataset we had 920 rows containing data. But after dropping
# rows containing NaN or NULL values we are left with 302 rows.
```

[23]:
```
(302, 12)
```

[42]:
```
df_index = df.set_index('id')
df_index.head(5)

# Earlier we had default index starting from 0. By setting the index we had
```

```
# converted our column (id) as an index.
```

[42]:
```
    age     sex               cp  trestbps   chol         restecg  thalch  \
id
1    63    Male   typical angina     145.0  233.0  lv hypertrophy   150.0
2    67    Male     asymptomatic     160.0  286.0  lv hypertrophy   108.0
3    67    Male     asymptomatic     120.0  229.0  lv hypertrophy   129.0
4    37    Male      non-anginal     130.0  250.0          normal   187.0
5    41  Female  atypical angina     130.0  204.0  lv hypertrophy   172.0

    exang  oldpeak        slope   ca
id
1   False      2.3  downsloping  0.0
2    True      1.5         flat  3.0
3    True      2.6         flat  2.0
4   False      3.5  downsloping  0.0
5   False      1.4    upsloping  0.0
```

[45]:
```
df_reset = df_index.reset_index('id')
df_reset.head(5)

# Resets the index to its default form.
```

[45]:
```
   id  age     sex               cp  trestbps   chol         restecg  thalch  \
0   1   63    Male   typical angina     145.0  233.0  lv hypertrophy   150.0
1   2   67    Male     asymptomatic     160.0  286.0  lv hypertrophy   108.0
2   3   67    Male     asymptomatic     120.0  229.0  lv hypertrophy   129.0
3   4   37    Male      non-anginal     130.0  250.0          normal   187.0
4   5   41  Female  atypical angina     130.0  204.0  lv hypertrophy   172.0

   exang  oldpeak        slope   ca
0  False      2.3  downsloping  0.0
1   True      1.5         flat  3.0
2   True      2.6         flat  2.0
3  False      3.5  downsloping  0.0
4  False      1.4    upsloping  0.0
```

[23]:
```
df.isnull().any()

# This function checks if any attribute contains NaN values. If there is a
# presence of even a single NaN value it marks that attribute with boolean value
# 'True' and if there is no such value then it is marked with false. Based on
# our dataset id, age and sex are the columns with no NaN value.
```

[23]:
```
id          False
age         False
sex         False
```

```
cp           True
trestbps     True
chol         True
restecg      True
thalch       True
exang        True
oldpeak      True
slope        True
ca           True
dtype: bool
```

[24]:
```
df.isnull().sum()

# It counts the total NaN and NULL values present in each attribute. The oldpeak
# attribute contains the maximum number of NULL values.
```

[24]:
```
id            0
age           0
sex           0
cp            7
trestbps     59
chol         30
restecg       2
thalch       56
exang        55
oldpeak      62
slope       309
ca          611
dtype: int64
```

[20]:
```
df1 = df.dropna(how = 'all')
df1.shape

# It checks and drops that row in which the whole row contains NULL values.
# Based on the output there were no such rows containing all NULL values as the
# number of rows are same as original dataframe.
```

[20]:
```
(920, 12)
```

[34]:
```
df2 = df.dropna(axis = 1)
df2.shape

# It checks and removes that column which contains at least one missing value.
# If we go through the output we observe there are only three columns left
# that means rest all column possessed at least one missing value. Since each
# attribute plays a major role in data analysis it important to remove the
# particular data of the whole dataset and hence maintaining its integrity.
```

[34]:
```
(920, 3)
```

2 Detection of missing/invalid values

[35]:
```
df['cp'].unique()

# It checks all the unique values present in a particular attribute . The cp
# attribute has four unique values and five if we count NaN value as well.
```

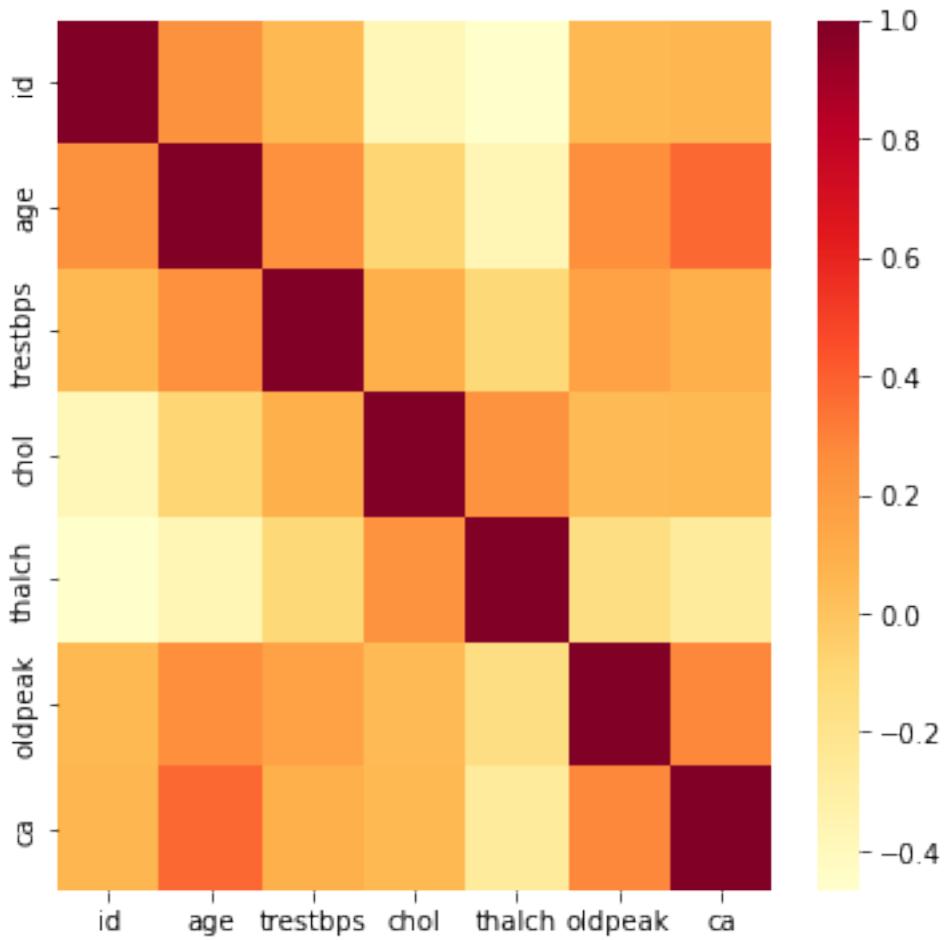
[35]:
```
array(['typical angina', 'asymptomatic', 'non-anginal', 'atypical angina',
       nan], dtype=object)
```

[36]:
```
df['cp'].value_counts()

# It counts the frequency of unique value present in an attribute.
```

[36]:
```
asymptomatic       491
non-anginal        204
atypical angina    172
typical angina      46
Name: cp, dtype: int64
```

[5]:
```
df['restecg'].replace('??', np.nan, inplace = True)

# It replaces ?? value with NaN value present in "restecg" column. The
# inplace arguement validates the replace command in the original dataset rather
# than creating a new dataset.
```

[52]:
```
cormat = df.corr()
top_cor_features = cormat.index
plt.figure(figsize = (6,6))
g = sns.heatmap(df[top_cor_features].corr(), cmap = "YlOrRd")

# The heatmap represents the coorelation among the attributes. The darker region
# shows that the corresponding attribute is strongly correlated while the
# lightest region are least coorelated. So those data which are least␣
 ↪coorelated
# could either be removed or should be given less priority for further analysis.
```



[13]:
```
missing = df[df.isnull().any(axis = 1)]
missing.head()

# It creates a dataframe of all columns containing NaN or missing values. Here
# row 20, 28, 154, 166 and so on contains NaN or NULL values.
```

[13]:

	id	age	sex	cp	trestbps	chol	restecg	thalch	exang	oldpeak	slope	ca
20	21	64	Male	typical angina	110.0	211.0	lv hypertrophy	NaN	True	1.8	flat	0.0
28	29	43	Male	NaN	150.0	247.0	normal	171.0	False	1.5	upsloping	0.0
154	155	64	Male	NaN	120.0	246.0	lv hypertrophy	96.0	True	2.2	downsloping	1.0
166	167	52	Male	non-anginal	138.0	223.0	normal	169.0				
192	193	43	Male	asymptomatic	132.0	247.0	lv hypertrophy	143.0				

```
166  False      0.0    upsloping  NaN
192   True      0.1         flat  NaN
```

[26]:
```
df4 = df_na.astype({'thalch':int, 'chol':int})
df4.info()

# Earlier the attribute "thalch" and "chol" was of 'float' type. After
# typecasting the attributes were changed to 'int' type.
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 302 entries, 0 to 748
Data columns (total 12 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   id        302 non-null    int64
 1   age       302 non-null    int64
 2   sex       302 non-null    object
 3   cp        302 non-null    object
 4   trestbps  302 non-null    float64
 5   chol      302 non-null    int64
 6   restecg   302 non-null    object
 7   thalch    302 non-null    int64
 8   exang     302 non-null    object
 9   oldpeak   302 non-null    float64
 10  slope     302 non-null    object
 11  ca        302 non-null    float64
dtypes: float64(3), int64(4), object(5)
memory usage: 30.7+ KB
```

3 Filling missing values

[29]:
```
df5 = df.fillna(value = 0)
df5.head()

# It fills all the NaN values with 0. This is done during preporcessing or data
# cleaning in order to perform operations like regression.
```

[29]:

	id	age	sex	cp	trestbps	chol	restecg	thalch	exang	oldpeak	slope	ca
0	1	63	Male	typical angina	145.0	233.0	lv hypertrophy	150.0	False	2.3	downsloping	0.0
1	2	67	Male	asymptomatic	160.0	286.0	lv hypertrophy	108.0				
2	3	67	Male	asymptomatic	120.0	229.0	lv hypertrophy	129.0				
3	4	37	Male	non-anginal	130.0	250.0	normal	187.0				
4	5	41	Female	atypical angina	130.0	204.0	lv hypertrophy	172.0				

```
1   True      1.5         flat  3.0
2   True      2.6         flat  2.0
3  False      3.5  downsloping  0.0
4  False      1.4    upsloping  0.0
```

[30]:
```
df5 = df.fillna(method = 'pad')
df5.head()

# It is a forward filling method. It replaces the NaN value with the most recent
# non null value present in the same column.
```

[30]:
```
   id  age     sex               cp  trestbps   chol         restecg  thalch  \
0   1   63    Male   typical angina     145.0  233.0  lv hypertrophy   150.0
1   2   67    Male     asymptomatic     160.0  286.0  lv hypertrophy   108.0
2   3   67    Male     asymptomatic     120.0  229.0  lv hypertrophy   129.0
3   4   37    Male      non-anginal     130.0  250.0          normal   187.0
4   5   41  Female  atypical angina     130.0  204.0  lv hypertrophy   172.0

   exang  oldpeak        slope   ca
0  False      2.3  downsloping  0.0
1   True      1.5         flat  3.0
2   True      2.6         flat  2.0
3  False      3.5  downsloping  0.0
4  False      1.4    upsloping  0.0
```

[38]:
```
df5 = df.fillna(method = 'bfill')
df5.head()

# It is a backward filling method. It replaces the NaN value with the next
# non null value present in the same column.
```

[38]:
```
   id  age     sex               cp  trestbps   chol         restecg  thalch  \
0   1   63    Male   typical angina     145.0  233.0  lv hypertrophy   150.0
1   2   67    Male     asymptomatic     160.0  286.0  lv hypertrophy   108.0
2   3   67    Male     asymptomatic     120.0  229.0  lv hypertrophy   129.0
3   4   37    Male      non-anginal     130.0  250.0          normal   187.0
4   5   41  Female  atypical angina     130.0  204.0  lv hypertrophy   172.0

   exang  oldpeak        slope   ca
0  False      2.3  downsloping  0.0
1   True      1.5         flat  3.0
2   True      2.6         flat  2.0
3  False      3.5  downsloping  0.0
4  False      1.4    upsloping  0.0
```

[54]:
```
df.interpolate()
df.head()
```

```
# It replaces the missing values using linear interpolation. This is done by
# observing the neighboring value present in that column.
```

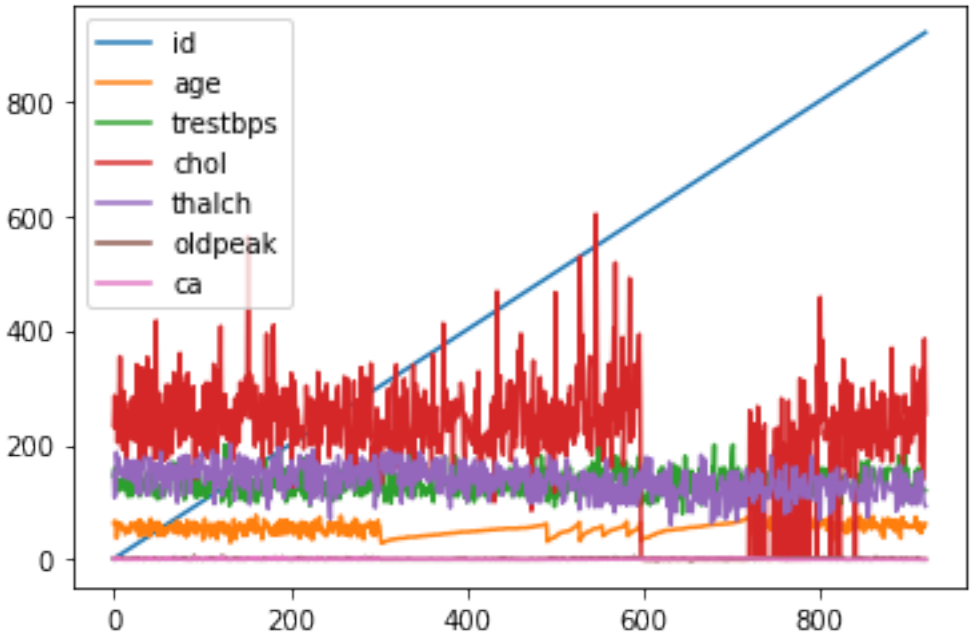
[54]:
```
   id  age     sex               cp  trestbps   chol         restecg  thalch  \
0   1   63    Male   typical angina     145.0  233.0  lv hypertrophy   150.0
1   2   67    Male     asymptomatic     160.0  286.0  lv hypertrophy   108.0
2   3   67    Male     asymptomatic     120.0  229.0  lv hypertrophy   129.0
3   4   37    Male      non-anginal     130.0  250.0          normal   187.0
4   5   41  Female  atypical angina     130.0  204.0  lv hypertrophy   172.0

   exang  oldpeak        slope   ca
0  False      2.3  downsloping  0.0
1   True      1.5         flat  3.0
2   True      2.6         flat  2.0
3  False      3.5  downsloping  0.0
4  False      1.4    upsloping  0.0
```

[36]:
```
df.interpolate().count()

# After the interpolate() method fills the missing or null values in df, the
# count() method is used to count the number of non-null values in each column
# of the resulting DataFrame.
```

[36]:
```
id          920
age         920
sex         920
cp          913
trestbps    920
chol        920
restecg     918
thalch      920
exang       865
oldpeak     920
slope       611
ca          920
dtype: int64
```

[37]:
```
df.interpolate().plot()

# After the interpolate() method fills the missing or null values in df, the
# plot() method is used to plot the non-null values in each column
# of the resulting DataFrame.
```

[37]:
```
<Axes: >
```


id
age
trestbps
chol
thalch
oldpeak
ca
800
600
400
200
0
0
200
400
600
800