Feature Engineering Missing Value Types

July 17, 2023

Missing Complety At Random Value A variable is missing completely at random (MCAR) if the probability of being missing is the same for all the observations. When data is MCAR, there is absolutely no relationship between the data missing and any other values, observed or missing, within the dataset.

```
[36]: import pandas as pd
      import numpy as np
      import matplotlib.pyplot as plt
      %matplotlib inline
```

```
[2]: df = pd.read_csv("C:\\Users\\ssart\\Downloads\\train.csv")
```

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

```
[3]: PassengerId      0
     Survived         0
     Pclass           0
     Name             0
     Sex              0
     Age            177
     SibSp            0
     Parch            0
     Ticket           0
     Fare             0
     Cabin          687
     Embarked         2
     dtype: int64
```

```
[4]: df[df['Embarked'].isnull()]
```

```
[4]:      PassengerId  Survived  Pclass                                       Name  \
     61            62         1       1                        Icard, Miss. Amelie
     829          830         1       1  Stone, Mrs. George Nelson (Martha Evelyn)

             Sex   Age  SibSp  Parch  Ticket  Fare Cabin Embarked
     61   female  38.0      0      0  113572  80.0   B28      NaN
     829  female  62.0      0      0  113572  80.0   B28      NaN
```

2.Missing Data Not At Random(MNAR): Systematic missing Values There is absolutely some relationship between the data missing and any other values, observed or missing, within the dataset.

```
[5]: df['Cabin_null']= np.where(df['Cabin'].isnull(),1,0)
```

```
[6]: df['Cabin_null']
```

```
[6]: 0      1
     1      0
     2      1
     3      0
     4      1
           ..
     886    1
     887    0
     888    1
     889    0
     890    1
     Name: Cabin_null, Length: 891, dtype: int32
```

```
[7]: df['Cabin']
```

```
[7]: 0        NaN
     1        C85
     2        NaN
     3       C123
     4        NaN
             …
     886      NaN
     887      B42
     888      NaN
     889     C148
     890      NaN
     Name: Cabin, Length: 891, dtype: object
```

```
[8]: # find the percentage of null values
```

```
[9]: df['Cabin_null'].mean()
```

```
[9]: 0.7710437710437711
```

```
[11]: df.columns
```

```
[11]: Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
             'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked', 'Cabin_null'],
            dtype='object')
```

[14]:
```
df.groupby(['Survived'])['Cabin_null'].mean()*100
```

[14]:
```
Survived
0    87.613843
1    60.233918
Name: Cabin_null, dtype: float64
```

0.1 Missing At Random (MAR)

0.1.1 All the techniques of handling ,issing values

1. Mean/ Median/Mode replacement
2. Random Sample Imputation
3. Capturing NAN values with a new feature
4. End of Distribution imputation
5. Arbitrary imputation
6. Frequent categories imputation

0.2 Mean/ MEdian /Mode imputation

When should we apply? Mean/median imputation has the assumption that the data are missing completely at random(MCAR). We solve this by replacing the NAN with the most frequent occurance of the variables

[16]:
```
df= pd.read_csv("C:\\Users\\ssart\\Downloads\\train.csv" , usecols =␣
 ↪['Age','Fare','Survived'])
```

[17]:
```
df
```

[17]:
```
     Survived   Age     Fare
0           0  22.0   7.2500
1           1  38.0  71.2833
2           1  26.0   7.9250
3           1  35.0  53.1000
4           0  35.0   8.0500
..        …    …       …
886         0  27.0  13.0000
887         1  19.0  30.0000
888         0   NaN  23.4500
889         1  26.0  30.0000
890         0  32.0   7.7500

[891 rows x 3 columns]
```

[22]:
```
df.isnull().mean()*100
```

[22]:
```
Survived     0.00000
Age         19.86532
Fare         0.00000
```

```
dtype: float64
```

[32]:
```
def impute_nan(df,variable,median):
    df[variable+ "_median"] = df[variable].fillna(median)
```

[33]:
```
median = df.Age.median()
median
```

[33]:
```
28.0
```

[34]:
```
impute_nan(df,'Age',median)
df.sample(5)
```

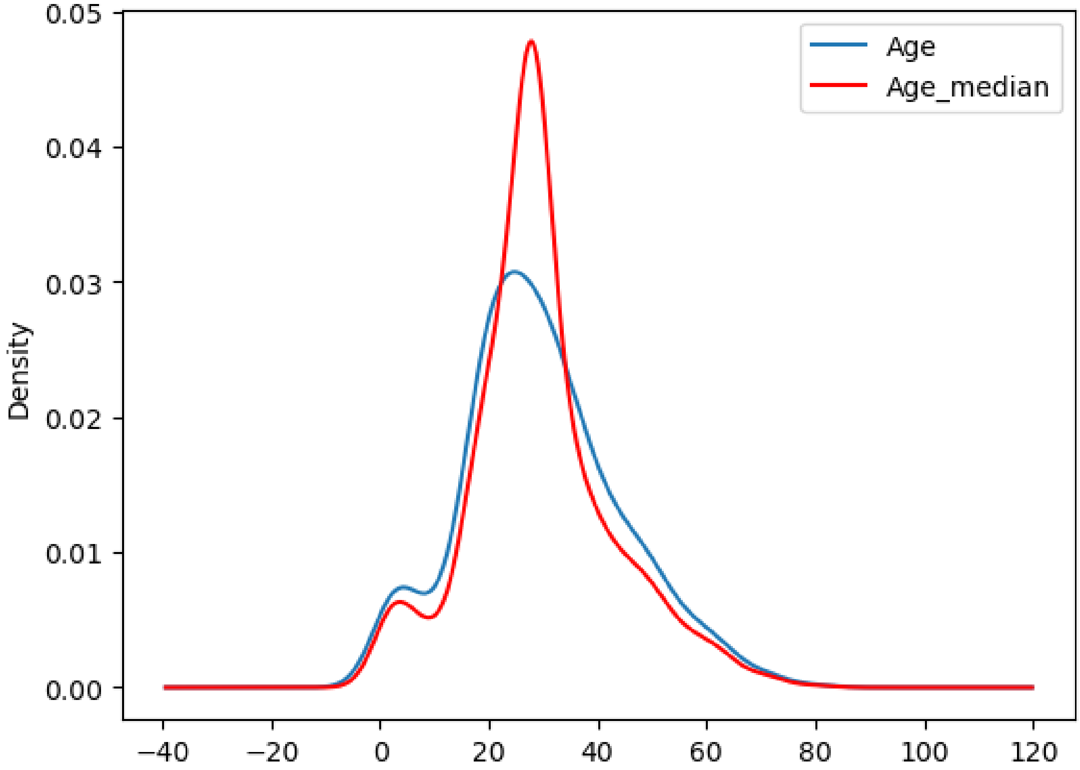
[34]:
```
     Survived   Age   Fare  Age_median
577         1  39.0  55.90        39.0
250         0   NaN   7.25        28.0
84          1  17.0  10.50        17.0
492         0  55.0  30.50        55.0
588         0  22.0   8.05        22.0
```

[35]:
```
print(df['Age'].std())
print(df['Age_median'].std())
```

```
14.526497332334044
13.019696550973194
```

[37]:
```
fig = plt.figure()
ax = fig.add_subplot(111)
df['Age'].plot(kind='kde', ax=ax)
df.Age_median.plot(kind='kde', ax=ax, color='red')
lines, labels = ax.get_legend_handles_labels()
ax.legend(lines, labels, loc='best')
```

[37]:
```
<matplotlib.legend.Legend at 0x251bd525a60>
```



0.3 Advantages And Disadvantages of Mean/Median Imputation

0.3.1 Advantages

Easy to implement(Robust to outliers) Faster way to obtain the complete dataset ### Disadvantages Change or Distortion in the original variance Impacts Correlation

[ ]:

```

```