4.1-Data_Cleaning-Categorical_Variables

KeytoDataScience.com

Data cleaning, or cleansing, is the process of correcting and deleting inaccurate records from a database or table.

It mainly consists of identifying and replacing incomplete, inaccurate, irrelevant, or otherwise problematic ('dirty') data and records.

Table of Contents

1 Issues in Datasets:

- Missing Values
- Irrelevant data
- Duplicated records
- Outliers
- Noise Values
- ...

The errors in the data are primarily due to source of the data.

1.1 Handling Missing Data

Missing data can arise in the dataset due to multiple reasons:

- the data for the specific field was not added by the user/data collection application,
- data was lost while transferring manually,
- a programming error, etc.

It is sometimes essential to understand the cause because this will influence how you deal with such data.

In [1]:
```
import pandas as pd
import numpy as np
```

In [2]:
```
# Creating a pandas series
data = pd.Series([0, 1, 2, 3, 4, 5, np.nan, 6, 7, 8])

# To check if and what index in the dataset contains null value
data.isnull()
```

Out[2]:
```
0    False
1    False
2    False
3    False
4    False
5    False
6     True
7    False
8    False
9    False
dtype: bool
```

1.1.1 Drop Missing Values

We can use the **`dropna()`** function to filter out missing data and to remove the null (missing) value and see only the non-null values. However, the NaN value is not really deleted and can still be found in the original dataset.

In [3]:
```
# Will not show the index 6 cause it contains null (NaN) value
data.dropna()
```

Out[3]:
```
0    0.0
1    1.0
2    2.0
3    3.0
4    4.0
5    5.0
7    6.0
8    7.0
9    8.0
dtype: float64
```

Example

In [4]:
```
# Creating a dataframe with 4 rows and 4 columns (4*4 matrix)
data_dim = pd.DataFrame([[1,2,3,np.nan],[4,5,np.nan,np.nan],[7,np.nan,np.nan,np.nan]])
data_dim
```

Out[4]:

	0	1	2	3
0	1	2.0	3.0	NaN
1	4	5.0	NaN	NaN

	0	1	2	3
2	7	NaN	NaN	NaN

In [5]:

```
# Drop all columns that have atleast 1 NaN value
data_dim.dropna(how = 'any',axis=1)
```

Out[5]:

	0
0	1
1	4
2	7

In [6]:

```
# Drop all columns that have all NaN values
data_dim.dropna(how = 'all',axis=1)
```

Out[6]:

	0	1	2
0	1	2.0	3.0
1	4	5.0	NaN
2	7	NaN	NaN

In [7]:

```
# Fill the NaN values with 0
data_dim_fill = data_dim.fillna(0)
data_dim_fill
```

Out[7]:

	0	1	2	3
0	1	2.0	3.0	0.0
1	4	5.0	0.0	0.0
2	7	0.0	0.0	0.0

1.1.2 Fill Missing Values

With some understanding of the data and your use-case, we can use the **`fillna()`** function in many other ways than simply filling it with numbers.

We could fill it up using the `mean` value using the mean() or the `median` value median() as well.

In [8]:

```
# Fill the NaN value with mean values in the corresponding column
data_dim_fill = data_dim.fillna(data_dim.mean())
data_dim_fill
```

Out[8]:

	0	1	2	3
0	1	2.0	3.0	NaN

	0	1	2	3
1	4	5.0	3.0	NaN
2	7	3.5	3.0	NaN

↥ back to top

2 Dealing with Categorical Variables

Categorical features can only take on a limited, and usually fixed, number of possible values.

For example,

- if a dataset is about information related to users, then you will typically find features like country, gender etc.
- alternatively, if the data you're working with is related to products, you will find features like product type, manufacturer, seller and so on.

There are two types of categorical features:

1. Nominal features: The categories are labeled without any order of precedence. For example, gender, etc.

2. Ordinal features: Categories have some order associated with them. For example, a feature like economic status, with three categories: low, medium and high, which have an order associated with them.

In [9]:

```
from sklearn.preprocessing import LabelEncoder
import seaborn as sns
from sklearn.impute import SimpleImputer
```

In [10]:

```
# Sample Data
data = pd.DataFrame(
       [['female', 'New York', 'low', 84], ['female', 'London', 'medium', 37], ['male',
       columns=['Gender', 'City', 'Temperature', 'Score'])
```

In [11]:

```
data
```

Out[11]:

	Gender	City	Temperature	Score
0	female	New York	low	84
1	female	London	medium	37
2	male	New Delhi	high	92

Since, there is a meaning behind the order of levels(low < medium < high), column Temperature is Oridal.

In [31]:

```
# Load Flights Dataset
```

```
df_flights = pd.read_csv(r'flights.txt')

#drop few columns for ease of understanding
# df_flights = df_flights.drop(['ARR_DELAY','DIVERTED','CANCELLED'],axis=1)
df_flights.head()
```

Out[31]:

	year	month	day	dep_time	dep_delay	arr_time	arr_delay	carrier	tailnum	flight	origin	dest
0	2014	1	1	1.0	96.0	235.0	70.0	AS	N508AS	145	PDX	ANC
1	2014	1	1	4.0	-6.0	738.0	-23.0	US	N195UW	1830	SEA	CLT
2	2014	1	1	8.0	13.0	548.0	-4.0	UA	N37422	1609	PDX	IAH
3	2014	1	1	28.0	-2.0	800.0	-23.0	US	N547UW	466	PDX	CLT
4	2014	1	1	34.0	44.0	325.0	43.0	AS	N762AS	121	SEA	ANC

In [32]:

```
print(df_flights.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 162049 entries, 0 to 162048
Data columns (total 16 columns):
 #   Column     Non-Null Count   Dtype  
---  ------     --------------   -----  
 0   year       162049 non-null  int64  
 1   month      162049 non-null  int64  
 2   day        162049 non-null  int64  
 3   dep_time   161192 non-null  float64
 4   dep_delay  161192 non-null  float64
 5   arr_time   161061 non-null  float64
 6   arr_delay  160748 non-null  float64
 7   carrier    162049 non-null  object 
 8   tailnum    161801 non-null  object 
 9   flight     162049 non-null  int64  
 10  origin     162049 non-null  object 
 11  dest       162049 non-null  object 
 12  air_time   160748 non-null  float64
 13  distance   162049 non-null  int64  
 14  hour       161192 non-null  float64
 15  minute     161192 non-null  float64
dtypes: float64(7), int64(5), object(4)
memory usage: 19.8+ MB
None
```

In [33]:

```
# print first 5 rows of data
df_flights.head()
```

Out[33]:

	year	month	day	dep_time	dep_delay	arr_time	arr_delay	carrier	tailnum	flight	origin	dest
0	2014	1	1	1.0	96.0	235.0	70.0	AS	N508AS	145	PDX	ANC
1	2014	1	1	4.0	-6.0	738.0	-23.0	US	N195UW	1830	SEA	CLT
2	2014	1	1	8.0	13.0	548.0	-4.0	UA	N37422	1609	PDX	IAH
3	2014	1	1	28.0	-2.0	800.0	-23.0	US	N547UW	466	PDX	CLT

	year	month	day	dep_time	dep_delay	arr_time	arr_delay	carrier	tailnum	flight	origin	dest
4	2014	1	1	34.0	44.0	325.0	43.0	AS	N762AS	121	SEA	ANC

In [34]:

```
# Check null values in the dataset
df_flights.isna().sum()
```

Out[34]:

```
year            0
month           0
day             0
dep_time      857
dep_delay     857
arr_time      988
arr_delay    1301
carrier         0
tailnum       248
flight          0
origin          0
dest            0
air_time     1301
distance        0
hour          857
minute        857
dtype: int64
```

Select columns with object data type

In [35]:

```
# Select columns with object data type
cat_df_flights = df_flights.select_dtypes(include=['object']).copy()
```

In [36]:

```
cat_df_flights.head()
```

Out[36]:

	carrier	tailnum	origin	dest
0	AS	N508AS	PDX	ANC
1	US	N195UW	SEA	CLT
2	UA	N37422	PDX	IAH
3	US	N547UW	PDX	CLT
4	AS	N762AS	SEA	ANC

Another Exploratory Data Analysis (EDA) step that you might want to do on categorical features is the frequency distribution of categories within the feature, which can be done with the **.value_counts()** method as described earlier.

In [38]:

```
cat_df_flights['carrier'].value_counts()
```

Out[38]:

```
AS    62460
WN    23355
OO    18710
DL    16716
```

```
UA    16671
AA     7586
US     5946
B6     3540
VX     3272
F9     2698
HA     1095
Name: carrier, dtype: int64
```

Check null values

In [39]:

```
cat_df_flights.isnull().sum()
```

Out[39]:

```
carrier      0
tailnum    248
origin       0
dest         0
dtype: int64
```

It seems that only the `tailnum` column has null values.

We can do a mode imputation for those null values. The function `fillna()` is handy for such operations.

In [40]:

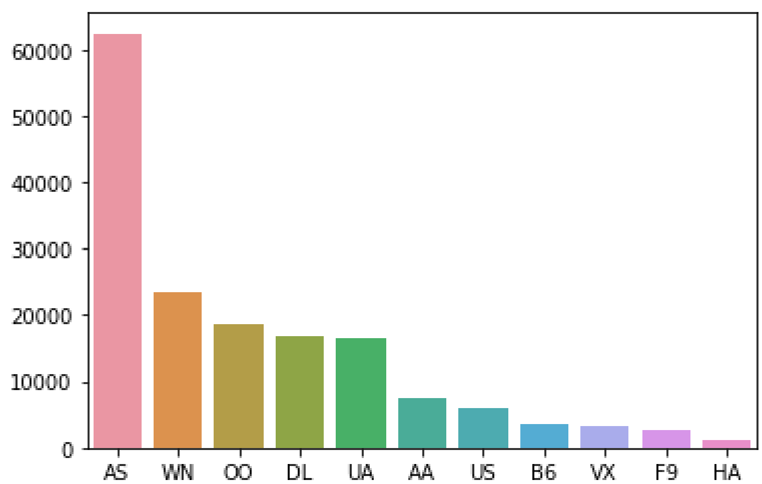
```
cat_df_flights = cat_df_flights.fillna(cat_df_flights['tailnum'].value_counts().index[0
```

In [44]:

```
carrier_count = df_flights['carrier'].value_counts()
# sns.set(style="darkgrid")
sns.barplot(x=carrier_count.index, y=carrier_count.values)
```

Out[44]:

```
<AxesSubplot:>
```



Many machine learning models, such as regression or SVM, are **algebraic**.

This means that their input must be **numerical**.

To use these models, categories must be transformed into numbers first, before you can apply the learning algorithm on them.

2.1 Ways of Encoding

1. Replace Values
2. Encoding labels
3. One Hot Encoding

2.1.1 Replacing values

This can be achieved with the help of the replace() function in pandas. The idea is that we have the liberty to choose whatever numbers we want to assign to the categories according to the business use case.

In [45]:
```
cat_df_flights.carrier.nunique()
```

Out[45]:
```
11
```

In [46]:
```
replace_map = {'carrier': {'AA': 1, 'AS': 2, 'B6': 3, 'DL': 4,
                                  'F9': 5, 'HA': 6, 'OO': 7 , 'UA': 8 , 'US': 9,'VX': 1
```

In [47]:
```
cat_df_flights_replace = cat_df_flights.copy()
```

In [48]:
```
cat_df_flights_replace.replace(replace_map, inplace=True)

print(cat_df_flights_replace.head())
```

```
   carrier tailnum origin dest
0        2  N508AS    PDX  ANC
1        9  N195UW    SEA  CLT
2        8  N37422    PDX  IAH
3        9  N547UW    PDX  CLT
4        2  N762AS    SEA  ANC
```

As we can observe, we have encoded the categories with the mapped numbers in our DataFrame.

we can also check the dtype of the newly encoded column, which is now converted to integers.

In [49]:
```
print(cat_df_flights_replace['carrier'].dtypes)
```

```
int64
```

2.1.2 Label Encoding

Another approach is to encode categorical values with a technique called "label encoding", which allows you to convert each value in a column to a number. Numerical labels are always between 0 and n_categories-1.

In [50]:
```
from sklearn.preprocessing import LabelEncoder

lb_make = LabelEncoder()
cat_df_flights['carrier_code_le'] = lb_make.fit_transform(cat_df_flights['carrier'])

cat_df_flights.head()
```

Out[50]:

	carrier	tailnum	origin	dest	carrier_code_le
0	AS	N508AS	PDX	ANC	1
1	US	N195UW	SEA	CLT	8
2	UA	N37422	PDX	IAH	7
3	US	N547UW	PDX	CLT	8
4	AS	N762AS	SEA	ANC	1

Label encoding is pretty much intuitive and straight-forward and may give you a good performance from your learning algorithm, but it has as disadvantage that the numerical values can be misinterpreted by the algorithm.

2.1.3 One Hot Encoding

The basic strategy is to convert each category value into a new column and assign a 1 or 0 (True/False) value to the column. This has the benefit of not weighting a value improperly.

There are many libraries out there that support one-hot encoding but the simplest one is using pandas' .get_dummies() method.

In [53]:

```
cat_df_flights_onehot = cat_df_flights.copy()
cat_df_flights_onehot = pd.get_dummies(cat_df_flights_onehot, columns=['carrier'], pref

cat_df_flights_onehot.head()
```

Out[53]:

	tailnum	origin	dest	carrier_code_le	c_AA	c_AS	c_B6	c_DL	c_F9	c_HA	c_OO	c_UA	c_US	c_V
0	N508AS	PDX	ANC	1	0	1	0	0	0	0	0	0	0	
1	N195UW	SEA	CLT	8	0	0	0	0	0	0	0	0	1	
2	N37422	PDX	IAH	7	0	0	0	0	0	0	0	1	0	
3	N547UW	PDX	CLT	8	0	0	0	0	0	0	0	0	1	
4	N762AS	SEA	ANC	1	0	1	0	0	0	0	0	0	0	

As you can see, the column `c_AS` gets value 1 at the 0th and 4th observation points as those points had the AS category labeled in the original DataFrame. Likewise for other columns also.

Let's merge this new column with the original dataframe

Note that this **`cat_df_flights_onehot`** resulted in a new DataFrame with only the one hot encodings for the feature carrier.

This needs to be concatenated back with the original DataFrame, which can be done via pandas' .concat() method. The axis argument is set to 1 as you want to merge on columns

In [54]:

```
result_df = pd.concat([cat_df_flights,cat_df_flights_onehot ], axis=1)

result_df.head()
```

Out[54]:

	carrier	tailnum	origin	dest	carrier_code_le	tailnum	origin	dest	carrier_code_le	c_AA	c_AS
0	AS	N508AS	PDX	ANC	1	N508AS	PDX	ANC	1	0	1
1	US	N195UW	SEA	CLT	8	N195UW	SEA	CLT	8	0	0
2	UA	N37422	PDX	IAH	7	N37422	PDX	IAH	7	0	0
3	US	N547UW	PDX	CLT	8	N547UW	PDX	CLT	8	0	0
4	AS	N762AS	SEA	ANC	1	N762AS	SEA	ANC	1	0	1

↥ back to top

Great Job!