data_wrangling

February 8, 2023

Data Wrangling

[ ]:
```
# import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

[ ]:
```
# read data file
df = pd.read_csv ('cars.csv')
```

[ ]:
```
# get details of the dataframe
df.shape
```

[ ]:
```
# get the head
df.head(5)
```

[ ]:
```
# get the tail
df.tail(5)
```

[ ]:
```
# replace ? with NaN
df.replace ('?', np.nan, inplace = True)
```

[ ]:
```
# check replacement
df.head(5)
```

Detect Missing Data

- isnull()
- notnull()

[ ]:
```
# find the missing data
missing_data = df.isnull()
missing_data.head(10)
```

[ ]:
```
# count missing values in each column
for column in missing_data.columns.values.tolist():
  print (column)
  print (missing_data[column].value_counts())
  print (' ')
```

In the 205 rows, missing data occurs in * normalized-losses: 41 * num-of-doors: 2 * bore: 4 * stroke: 4 * horsepower: 2 * peak-rpm: 2 * price: 4

How to Deal with Missing Data

- Drop data - row or column
- Replace data with mean, median, mode, or some other function

Replace by Mean or Median

- normalized-losses
- stroke
- bore
- horsepower
- peak-rpm

Replace by Mode

- num-of-doors replace missing data with "four"
  Most sedans have four doors

Drop Row

- price: delete row with missing data

[ ]:
```
# compute mean and median for normalized-losses
avg_norm_loss = df["normalized-losses"].astype("float").mean(axis=0)
print("Average of normalized-losses:", avg_norm_loss)
median_norm_loss = df["normalized-losses"].astype("float").median(axis=0)
print("Median of normalized-losses:", median_norm_loss)
```

[ ]:
```
# replace missing normalized-losses with mean
df["normalized-losses"].replace(np.nan, avg_norm_loss, inplace=True)
```

[ ]:
```
# compute mean and median for stroke
avg_stroke = df["stroke"].astype("float").mean(axis=0)
print("Average of stroke:", avg_stroke)
median_stroke = df["stroke"].astype("float").median(axis=0)
print("Median of stroke:", median_stroke)
```

[ ]:
```
# replace missing stroke with mean
df["stroke"].replace(np.nan, avg_stroke, inplace=True)
```

[ ]:
```
# compute mean and median for bore
avg_bore = df["bore"].astype("float").mean(axis=0)
print("Average of bore:", avg_bore)
median_bore = df["bore"].astype("float").median(axis=0)
print("Median of bore:", median_bore)
```

[ ]:
```
# replace missing bore with mean
df["bore"].replace(np.nan, avg_bore, inplace=True)
```

[ ]:
```
# compute mean and median for horsepower
avg_hp = df["horsepower"].astype("float").mean(axis=0)
print("Average of horsepower:", avg_hp)
median_hp = df["horsepower"].astype("float").median(axis=0)
print("Median of horsepower:", median_hp)
```

[ ]:
```
# replace missing horsepower with mean
df["horsepower"].replace(np.nan, avg_hp, inplace=True)
```

[ ]:
```
# compute mean and median for peak-rpm
avg_rpm = df["peak-rpm"].astype("float").mean(axis=0)
print("Average of peak-rpm:", avg_rpm)
median_rpm = df["peak-rpm"].astype("float").median(axis=0)
print("Median of peak-rpm:", median_rpm)
```

[ ]:
```
# replace missing peak-rpm with mean
df["peak-rpm"].replace(np.nan, avg_rpm, inplace=True)
```

[ ]:
```
# find the frequency of values for num-of-doors
df['num-of-doors'].value_counts()
```

[ ]:
```
# find the most frequent value
df['num-of-doors'].value_counts().idxmax()
```

[ ]:
```
#replace the missing 'num-of-doors' values by the most frequent
df["num-of-doors"].replace(np.nan, "four", inplace=True)
```

[ ]:
```
# simply drop whole row with NaN in "price" column
df.dropna(subset=["price"], axis=0, inplace=True)

# reset index, because we droped two rows
df.reset_index(drop=True, inplace=True)
```

[ ]:
```
# let us check the data
df.head(10)
```

Check Correct Data Format

[ ]:
```
# check dtata types
df.dtypes
```

[ ]:
```
# convert data types to correct format
df[["bore", "stroke"]] = df[["bore", "stroke"]].astype("float")
df[["normalized-losses"]] = df[["normalized-losses"]].astype("int")
```

```
df[["price"]] = df[["price"]].astype("int")
df[["peak-rpm"]] = df[["peak-rpm"]].astype("float")
```

[ ]:
```
# check types again
df.dtypes
```

Check for Duplicates

[ ]:
```
# check for duplicates
pd.options.display.max_rows = 1000
print (df.duplicated())
```

[ ]:
```
# drop duplicates
df_nodup = df.drop_duplicates()
```

[ ]:
```
# check size
df.shape
```

[ ]:
```
df_nodup.shape
```