Lecture 20 - Data Cleanup

Pandas for data cleanup really needs a way to match when data is not “clean”. In our data example there is a column that has a unit in it - that is “MPG” - but there are also values that are have blanks in them. Blanks will make something not a *number* so we will need to find and fix these values.

Machine learning works on all numbers - so some of our data is in the form of text values. We will have to find and fix these also.

Let’s take a look at the “data”.

The first step is to read the data into Pandas.

Normally when I read in data I immediately want to print out some of it to verify that it is in the program. Pandas has a number of nice “tools” to do this. “head()” is the first.

```
#!/Users/philip/opt/anaconda3/bin/python

import numpy as np
import pandas as pd
import re

dataset_path = "./train-data.csv"

column_names = ['Ind', 'Name', 'Location', 'Year', 'Kilometers_Driven',
    'Fuel_Type', 'Transmission', 'Owner_Type', 'Mileage', 'Engine',
    'Power', 'Seats', 'New_Price', 'Price']
raw_dataset = pd.read_csv(dataset_path, names=column_names,
    na_values = "?", comment='\t', skiprows=1, sep=",",
    skipinitialspace=True)

dataset = raw_dataset.copy()
print ( dataset.head() )

# https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.to_csv.html

dataset = dataset.drop(columns=['Ind', 'Name', 'Location', 'Seats', 'New_Price'])
print ( dataset.head() )

# To see a good description of the dataset

print ( dataset.describe() )

```

Also “describe()”

```
#!/Users/philip/opt/anaconda3/bin/python

import numpy as np
import pandas as pd
import re

dataset_path = "./train-data.csv"

column_names = ['Ind', 'Name', 'Location', 'Year', 'Kilometers_Driven',
    'Fuel_Type', 'Transmission', 'Owner_Type', 'Mileage', 'Engine',
    'Power', 'Seats', 'New_Price', 'Price']
raw_dataset = pd.read_csv(dataset_path, names=column_names,
    na_values = "?", comment='\t', skiprows=1, sep=",",
    skipinitialspace=True)

dataset = raw_dataset.copy()
print ( dataset.head() )

# https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.to_csv.html

dataset = dataset.drop(columns=['Ind', 'Name', 'Location', 'Seats', 'New_Price'])
print ( dataset.head() )

# To see a good description of the dataset

print ( dataset.describe() )

# Cleaning the data
# The dataset contains a few unknown values. Let’s find them and drop them.

dataset.isna().sum()
dataset = dataset.dropna()
dataset = dataset.reset_index(drop=True)

print ( dataset.head() )

```

Now let’s take care of the column with “ MPG” in it.

We have a problem - how to find values that match a “pattern” instead of a single value.

29 MPG

And

31 MPG

```
dataset['Mileage'] = pd.Series([re.sub('[^.0-9]', '', str(val)) for val in dataset['Mil
```

Part 1 - Regular Expressions

Python (and most computer languages) have a pattern-matching language in it called regular expressions.

First regular expressions can match letters.

```
a
```

Matches the letter ‘a’.

```
ab
```

Matches the letter ‘a’ followed by the letter ‘b’.

Now Let’s look at a pattern.

```
[abcc]
```

Matches any characters a, b, c, or d.

```
[a-d]
```

Also Matches any characters a, b, c, or d.

So…

```
[a-z]
```

Mach’s any lower case letter.

and

```
[A-Z]
```

Matches any upper case letter.

```
a*
```

Matches 0 or more 'a's

```
ab*
```

Matches an 'a' followed by 0 or more b's.

Let's combine the character class `[a-z]` with or more `*`

```
[a-z]*
```

Matches any string of letters.

```
[0-9]
```

Matches a digit.

```
[0-9][0-9]*
```

Matches numbers!

Now we an match our erant column with numbers followed by MPG.

```
[0-9][0-9]* M[pP][Gg]
```

Finally! We are getting someplace.

Suppose that we just want the number at the beginning.

We can match all the characters that are *NOT* in a character-class.

```
[^ab]
```

Matches things that are *NOT* a or b.

So it will match “xy” and “123” but it will not match “a”.

So if we want to match “ MPG” and “ Mpg” and “MPG” we can use:

```
[^0-9]
```

And if we want to take care of the case where we have a floating point number.

```
[^.0-9]
```

So we can use a regular expressions function called “sub” that allows us to substitute values. It takes 3 values, a patter, a replacement string and the original value to work on.

```
re.sub('Pattern',"Replace","data to work on")
```

```

import re

s = "128 MPG"
t = re.sub('[^.0-9]', '', str(s))

print ( "s=-->{}<--   t=-->{}<--".format(s,t))

n = 128
t = re.sub('[^.0-9]', '', str(n))

print ( "n=-->{}<--   t=-->{}<--".format(n,t))
```

Part 2 - list comprehension

List comprehensions are a powerful way to manipulate lists in Python.

Let’s start with a list.

```
fruits = [ "apple", "cherry", "kiwi", "mango", "duraian" ]
stinks = [ "no", "no", "no", "no", "yes" ]
```

we can build a list to pick out values from this

```
newlist = [x for x in fruits if "a" in x]
print ( "newlist={}".format(newlist) );
```

the syntax is:

```
newlist = [expression for item in iterable if condition == True]
```

Let’s pick out just the stinky fruit:

or Let’s convert our fruit to upper case:

```
yellingFruit = [ x.upper() for x in fruits ]
print ( yellingFruit )
```