Statistics and Counts

This lesson discusses principal data processing and data analytics techniques using Pandas.

WE'LL COVER THE FOLLOWING

In the last chapter, while studying how to read data from CSV files, we used Pandas. Now, we will look deeper at Pandas to process data.

To do so, we will be using the same data set we used when learning about how to read in CSV data. Here are the first 5 rows to refresh your memory.

```
import pandas as pd
names = ['age', 'workclass', 'fnlwgt', 'education', 'educationnum', 'maritalstatus', 'occupat
        'sex', 'capitalgain', 'capitalloss', 'hoursperweek', 'nativecountry', 'label']
train_df = pd.read_csv("adult.data", header=None, names=names)
print(train_df.head())
```

Gathering statistics on data

A good place to start is just looking at your data using some pandas functions to better understand what issues there might be. `Describe` will give you counts and some statistics for continuous variables.

```
import pandas as pd
names = ['age', 'workclass', 'fnlwgt', 'education', 'educationnum', 'maritalstatus', 'occupat
        'sex', 'capitalgain', 'capitalloss', 'hoursperweek', 'nativecountry', 'label']
train_df = pd.read_csv("adult.data", header=None, names=names)
print(train_df.describe())
```

For all of our numeric values, we now have the `mean`, the `std`, the `min`, the `max`, and a few different `percentiles`.

> **Note**: It is good to remember that the mean value will be influenced more by outliers than the median. Also, you can always square the standard deviation to get the variance.

You may have noticed that some columns are missing. The only columns `describe()` function fetched for us are the ones that hold numeric data.

Finding the data types

To see what types of data we have, we can use the `info()` function:

```
import pandas as pd

# Read in data as explained in reading CSV lesson
names = ['age', 'workclass', 'fnlwgt', 'education', 'educationnum', 'maritalstatus', 'occupat
        'sex', 'capitalgain', 'capitalloss', 'hoursperweek', 'nativecountry', 'label']
train_df = pd.read_csv("adult.data", header=None, names=names)

print(train_df.info())  # Use the info() function on the dataframe
```

In our dataframe, we have two data types, `object` and `int64`. You can think of an `object` as a string value and `int64` as an integer.

Converting data types

If a column doesn't seem to have the correct type, it is easy to convert it to different types using `.to_()` functions:

- `to_numeric()`

- `to_numeric()`
- `to_datetime()`
- `to_string()`

For example:

```
df['numeric_column'] = pd.to_numeric(df['string_column'])
```

You also get the count of non-null values per column and the memory usage of your dataframe from the `info` command used above.

Finding unique values

Another useful step is to look at unique values for columns. Here is an example for the `relationship` column:

```
import pandas as pd
names = ['age', 'workclass', 'fnlwgt', 'education', 'educationnum', 'maritalstatus', 'occupat
        'sex', 'capitalgain', 'capitalloss', 'hoursperweek', 'nativecountry', 'label']
train_df = pd.read_csv("adult.data", header=None, names=names)
print(train_df['relationship'].unique())
```

Now we can see all the unique values above and can check the counts of unique values for `relationships`:

```
import pandas as pd
names = ['age', 'workclass', 'fnlwgt', 'education', 'educationnum', 'maritalstatus', 'occupat
        'sex', 'capitalgain', 'capitalloss', 'hoursperweek', 'nativecountry', 'label']
train_df = pd.read_csv("adult.data", header=None, names=names)
print(train_df['relationship'].value_counts())
```

This shows us all the unique values for relationships as well as their counts. So, we have a lot of values as **husband** in the `relationship` column, but less **Other-relative**.

Grouping the data

We can also do these types of counts by specific groups by using the `groupby()`

function. This function takes a list of columns by which you would like to

group your dataframe. It then performs the requested calculations on each group individually and returns the results by group. Here is an example:

```
import pandas as pd
names = ['age', 'workclass', 'fnlwgt', 'education', 'educationnum', 'maritalstatus', 'occupat
        'sex', 'capitalgain', 'capitalloss', 'hoursperweek', 'nativecountry', 'label']
train_df = pd.read_csv("adult.data", header=None, names=names)

# Group by relationship and then get the value counts of label with normalization
print(train_df.groupby('relationship')['label'].value_counts(normalize=True))
```

What we did above was group by the variable, `relationship`, and then perform value counts on the variable `label`. For these data, label is whether you make more than **50k**. We can see above that **55%** of *husbands* make more than **50k**. We received *percentages* because we used the `normalize=True` parameter.

You can do many types of calculations on groups using Pandas. For example, here we can see the mean hours worked per week by `workclass`.

```
import pandas as pd
names = ['age', 'workclass', 'fnlwgt', 'education', 'educationnum', 'maritalstatus', 'occupat
        'sex', 'capitalgain', 'capitalloss', 'hoursperweek', 'nativecountry', 'label']
train_df = pd.read_csv("adult.data", header=None, names=names)
print(train_df.groupby(['workclass'])['hoursperweek'].mean())
```

From the above, it looks like Federal government workers work more than local workers on average. **Never-worked** average about 28 hours.

Finding the correlation

Another useful statistic is the correlation. If you need a refresher on correlation, please check out Wikipedia. You can calculate all the pair-wise correlations in your dataframe by using the `corr` function.

```
import pandas as pd
names = ['age', 'workclass', 'fnlwgt', 'education', 'educationnum', 'maritalstatus', 'occupat
        'sex', 'capitalgain', 'capitalloss', 'hoursperweek', 'nativecountry', 'label']

train_df = pd.read_csv("adult.data", header=None, names=names)

# Calculate correlations
print(train_df.corr())
```

We can quickly see that compared to all of the correlations, there is a higher correlation between "hours per week" and "education num", but it is not very high. You will notice, though since our label is an object, it isn't included here. Knowing how variables correlate with our label would be useful, so let's take care of that:

```
import pandas as pd
names = ['age', 'workclass', 'fnlwgt', 'education', 'educationnum', 'maritalstatus', 'occupat
        'sex', 'capitalgain', 'capitalloss', 'hoursperweek', 'nativecountry', 'label']
train_df = pd.read_csv("adult.data", header=None, names=names)

# Convert the string label into a value of 1 when >= 50k and 0 otherwise
train_df['label_int'] = train_df.label.apply(lambda x: ">" in x)
print(train_df.corr())
```

There seems to be some decent correlation with the label and education num. One thing to note, though, is that our label is categorical, so correlation doesn't really apply, our groupby frequencies are probably a better method.

> **Note**: Categorical variables are variables with categories with no intrinsic order. For example, gender.

Also, keep in mind, these are just univariate correlations (between one variable) and don't account for multi-variate effects (between multiple variables). You can also calculate the correlation using the `scipy` package which has the added benefit of p-values. This was discussed in the "Scipy an External Library" lesson.

Generating percentiles

Lastly, the describe function of Pandas gives some percentiles, but it is easy to add more:

```
import pandas as pd
names = ['age', 'workclass', 'fnlwgt', 'education', 'educationnum', 'maritalstatus', 'occupat
        'sex', 'capitalgain', 'capitalloss', 'hoursperweek', 'nativecountry', 'label']
train_df = pd.read_csv("adult.data", header=None, names=names)

# Use the describe function to calculate the percentiles specified
print(train_df.describe(percentiles=[.01,.05,.95,.99]))
```

A percentile is the value below which a given percent of the data falls.

We pass the percentile values we want using the `percentiles` parameter shown above.

In the next lesson, we will take a look at various ways to reshape our dataframe.