Metrics

Use pandas to obtain statistical metrics for data.

Chapter Goals:

- Understand the common metrics used to summarize numeric data
- Learn how to describe categorical data using histograms

A. Numeric metrics

When working with numeric features, we usually want to calculate metrics such as mean, standard deviation, etc. These metrics give us more insight into the type of data we're working with, which benefits our overall analysis of the dataset.

Rather than calculating several different metrics separately, pandas provides the `describe (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.describe.html)` function to obtain a summary of a DataFrame's numeric data.

The metrics included in the output summary of `describe` are

Metric	Description
count	The number of rows in the DataFrame
mean	The mean value for a feature
std	The standard deviation for a feature
min	The minimum value in a feature

Metric	Description
25%	The 25th percentile of a feature
50%	The 50th percentile of a feature. Note that this is identical to the median
75%	The 75th percentile of a feature
max	The maximum value in a feature

The code below shows how to use the `describe` function.

```
# df is predefined
print('{}\n'.format(df))

metrics1 = df.describe()
print('{}\n'.format(metrics1))

hr_rbi = df[['HR','RBI']]
metrics2 = hr_rbi.describe()
print('{}\n'.format(metrics2))
```

Output 1.057s

```
    HR  RBI   playerID teamID  yearID
0  39  119   cruzne02    SEA    2017
1   7   62  pedrodu01    BOS    2017
2  43  105   cruzne02    SEA    2016
3  12   42  pedrodu01    BOS    2015
4  33   72  troutmi01    LAA    2017
5  15   74  pedrodu01    BOS    2016

               HR         RBI       yearID
```

Using `describe` with a DataFrame will return a summary of metrics for each of the DataFrame's numeric features. In our example, `df` had three features with numerical values: `yearID`, `HR`, and `RBI`.

Since we normally treat `yearID` as a categorical feature, the second time we used `describe` was with the `hr_rbi` DataFrame, which only included the `HR` and `RBI` features.

To have `describe` return specific percentiles, we can use the `percentiles` keyword argument. The `percentiles` argument takes in a list of decimal percentages, representing the percentiles we want returned in the summary.

```
metrics1 = hr_rbi.describe(percentiles=[.5])
print('{}\n'.format(metrics1))

metrics2 = hr_rbi.describe(percentiles=[.1])
print('{}\n'.format(metrics2))

metrics3 = hr_rbi.describe(percentiles=[.2,.8])
print('{}\n'.format(metrics3))
```

Output 0.668s

```
               HR         RBI
count   6.000000    6.000000
mean   24.833333   79.000000
std    15.341664   28.312541
min     7.000000   42.000000
50%    24.000000   73.000000
max    43.000000  119.000000

               HR         RBI
```

Note that the 50th percentile, i.e. the median, is always returned. The values specified in the `percentiles` list will replace the default 25th and 75th percentiles.

B. Categorical features

With categorical features, we don't calculate metrics like mean, standard deviation, etc. Instead, we use *frequency counts* to describe a categorical feature.

The frequency count for a specific category of a feature refers to how many times that category appears in the dataset. In pandas, we use the `value_counts (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.value_counts.html)` function to obtain the frequency counts for each category in a column feature.

The code below uses the `value_counts` function to get frequency counts of the `'playerID'` feature.

```
p_ids = df['playerID']
print('{}\n'.format(p_ids.value_counts()))

print('{}\n'.format(p_ids.value_counts(normalize=True)))

print('{}\n'.format(p_ids.value_counts(ascending=True)))
```

Output 0.621s

```
pedrodu01    3
cruzne02     2
troutmi01    1
Name: playerID, dtype: int64

pedrodu01    0.500000
cruzne02     0.333333
troutmi01    0.166667
Name: playerID, dtype: float64
```

Using `value_counts` without any keyword arguments will return the frequency counts for each category, sorted in descending order.

Setting `normalize=True` returns the frequency proportions, rather than counts, for each category (note that the sum of all the proportions is 1). We can also set `ascending=True` to get the frequencies sorted in ascending order.

If we just want the names of each unique category in a column, rather than the frequencies, we use the `unique` (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.unique.html) function.

```
unique_players = df['playerID'].unique()
print('{}\n'.format(repr(unique_players)))

unique_teams = df['teamID'].unique()
print('{}\n'.format(repr(unique_teams)))
```

Output 0.670s

```
array(['cruzne02', 'pedrodu01', 'troutmi01'], dtype=object)

array(['SEA', 'BOS', 'LAA'], dtype=object)
```

So far we've focused on categorical features with string values. However, categorical features can also have integer values. For example, we can use `yearID` as a categorical feature with each unique year as a separate category.

```
y_ids = df['yearID']
print('{}\n'.format(y_ids))

print('{}\n'.format(repr(y_ids.unique())))
print('{}\n'.format(y_ids.value_counts()))
```

Output 1.125s

```
0    2017
1    2017
2    2016
3    2015
4    2017
5    2016
Name: yearID, dtype: int64

array([2017, 2016, 2015])
```

Time to Code!

The coding exercises for this chapter involve getting metrics from a DataFrame of MLB players, `player_df`.

First, we'll get a summary of all the statistics in `player_df`.

Set `summary_all` equal to `player_df.describe` with no arguments.

```
# CODE HERE
```

Solution

```
summary_all = player_df.describe()


```

Next, we want to get summaries specifically for the home run totals. The first summary will contain the default metrics from `describe`, while the second summary will contain the 10th and 90th percentiles.

Set `hr_df` equal to `player_df[]` directly indexed with `'HR'`.

Set `summary_hr` equal to `hr_df.describe` with no arguments.

Set `low_high_10` equal to `hr_df.describe` with `[.1,.9]` as the `percentiles` keyword argument.

```
# CODE HERE
```

Solution

```
hr_df = player_df['HR']
summary_hr = hr_df.describe()
low_high_10 = hr_df.describe(percentiles=[.1, .9])

```

Finally, we'll treat the `'HR'` feature as a categorical variable, with each unique home run total as a separate category. We then get the frequency counts for each category.

Set `hr_counts` equal to `hr_df.value_counts` with no arguments.

```
# CODE HERE
```

Solution

```
hr_counts = hr_df.value_counts()

```

Back
Sorting

Next
Plotting

Mark as Completed