Wprowadzenie do ggplot2

`ggplot2` jest jednym z najczęściej wykorzystywanych pakietów do tworzenia grafiki w R. Pozwala tworzyć w łatwy sposób dość skomplikowane wykresy, posiada również dość rozsądne ustawienia standardowe, dzieki którym możemy uzyskać estetycznie wyglądające ilustracje małym nakładem pracy.

Nie jest on dołączy do podstawowej dystrybucji R i należy zainstalować go za pomocą polecenia `install.packages`.

```
# install.packages('ggplot2') # instalacja ggplot2
library(ggplot2)
```

Praca z `ggplot2` różni się od pracy z grafiką w standardowym R. Podstawowe różnice to:

1. Preferowana jest praca z ramkami danych, raczej nie zajdziemy za daleko pracując na pojedynczych wektorach.
2. Przestaje obowiązywać metafora płótna. Rysując w standardowym R mogliśmy podchodzić do tego jak do nakładania kolejnych linii, punktów, segmentów itp. za pomocą funkcji niskiego poziomu. W przypadku `ggplot2` raczej definiujemy wykres, nie przejmujemy się najczęściej niskopoziomowymi szczegółami.
3. Opiera się na teoretycznej koncepcji *grammar of graphics* i stara się ją implementować.
4. Wykresy tworzymy dodając (`+` - `ggplot2` przeciąża operator dodawania) do siebie obiekty reprezentujące "warstwy" albo "mapowania" w specyficznej "gramatyce grafiki" implementowanej przez `ggplot2`.
5. Inaczej zapisujemy grafikę - używamy funkcji `ggsave`.

Co będziemy omawiać:

1. Zarys możliwości pakietu.
2. Tworzenie wykresów za pomocą `qplot` oraz gramatyki grafiki (mapowanie).
3. Różne rodzaje wykresów dla jednej lub dwóch zmiennych nominalnych i liczbowych.
4. Kilkupanelowe wykresy.

Co nie będzie omawiane, ale jest ważną czescią pakietu `ggplot2`:

1. Funkcje niskiego poziomu do rysowania kształtów.
2. Osie, skale.
3. Podpisy, napisy, przypisy (elementy tekstowe).
4. Drobnoziarnistwa kontrola.

Innymi słowy - w `ggplot2` możemy zmieniać detale tworzonych przez nas wykresów, ale nie będziemy omawiać każdej służącej do tego funkcji i jej argumentu. Informacje można łatwo znaleźć w Internecie, dokumentacja rozmaitych funkcji bogata jest też w przykłady.

`qplot` - funkcja do szybkiego tworzenia wykresów (*quickplot*)

`qplot` jest *wrapperem* (można myśleć: skrótem) na inne funkcje `ggplot2`, który pozwala nam bardzo szybko i bez potrzeby żmudnego definiowania mapowan tworzyć proste wykresy. Z założenia funkcja `qplot` nie ma generować bardzo złożonych wykresów, lecz estetycznie wyglądające bardzo podstawowe typy wykresów. Przypomnijmy sobie jak wyglądały dane ze zbioru `beaver1` - będziemy na nich pracować.

```
head(beaver1)
```

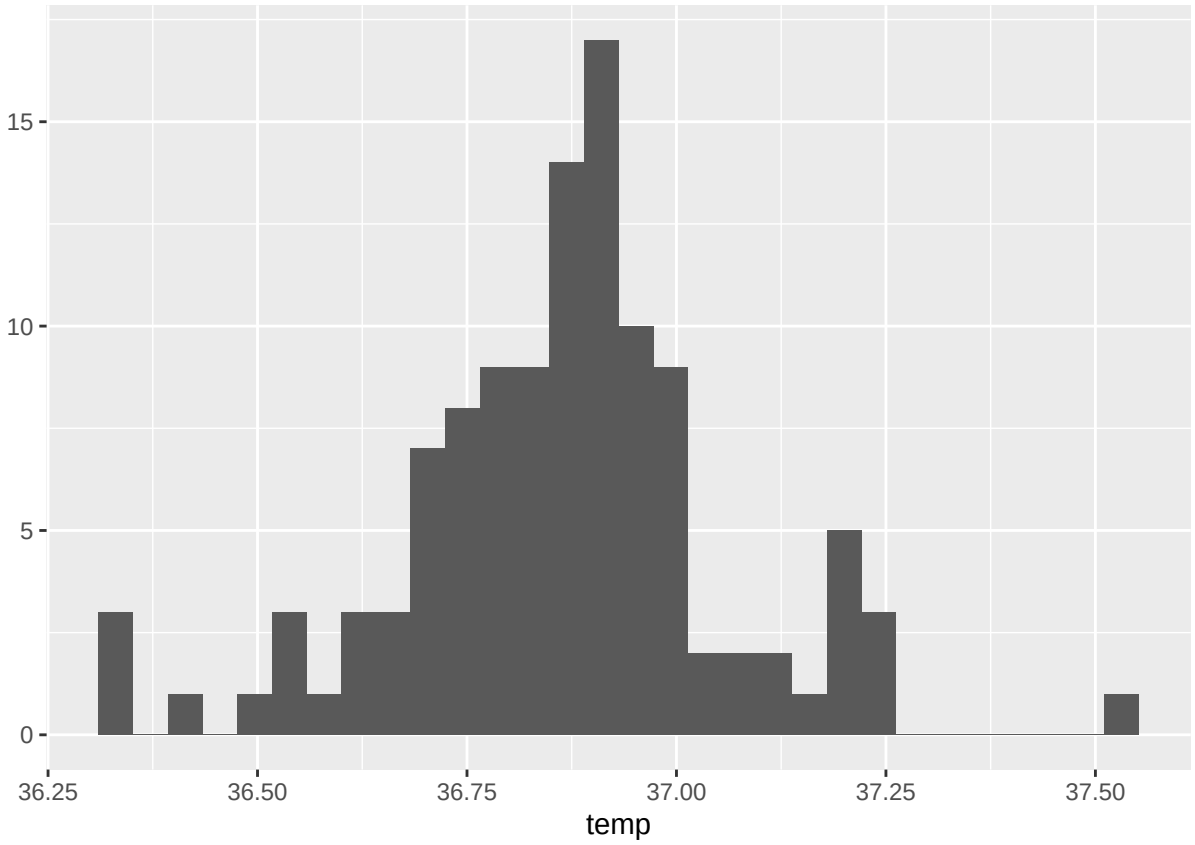
```
##   day time  temp activ
```

```
## 1 346  840 36.33     0
## 2 346  850 36.34     0
## 3 346  900 36.35     0
## 4 346  910 36.42     0
## 5 346  920 36.55     0
## 6 346  930 36.69     0
```

W przypadku, gdy w funkcji `qplot` podamy tylko jedną zmienną (`x`), `ggplot2` wyprodukuje domyślnie histogram. Zmienne przekazujemy **bez cudzysłowiów** i są nimi **nazwy kolumn** w naszej ramce danych. Podajemy również argument `data`, w którym przekazujemy zmienną, w której znajduje się ramka danych, której chcemy używać.

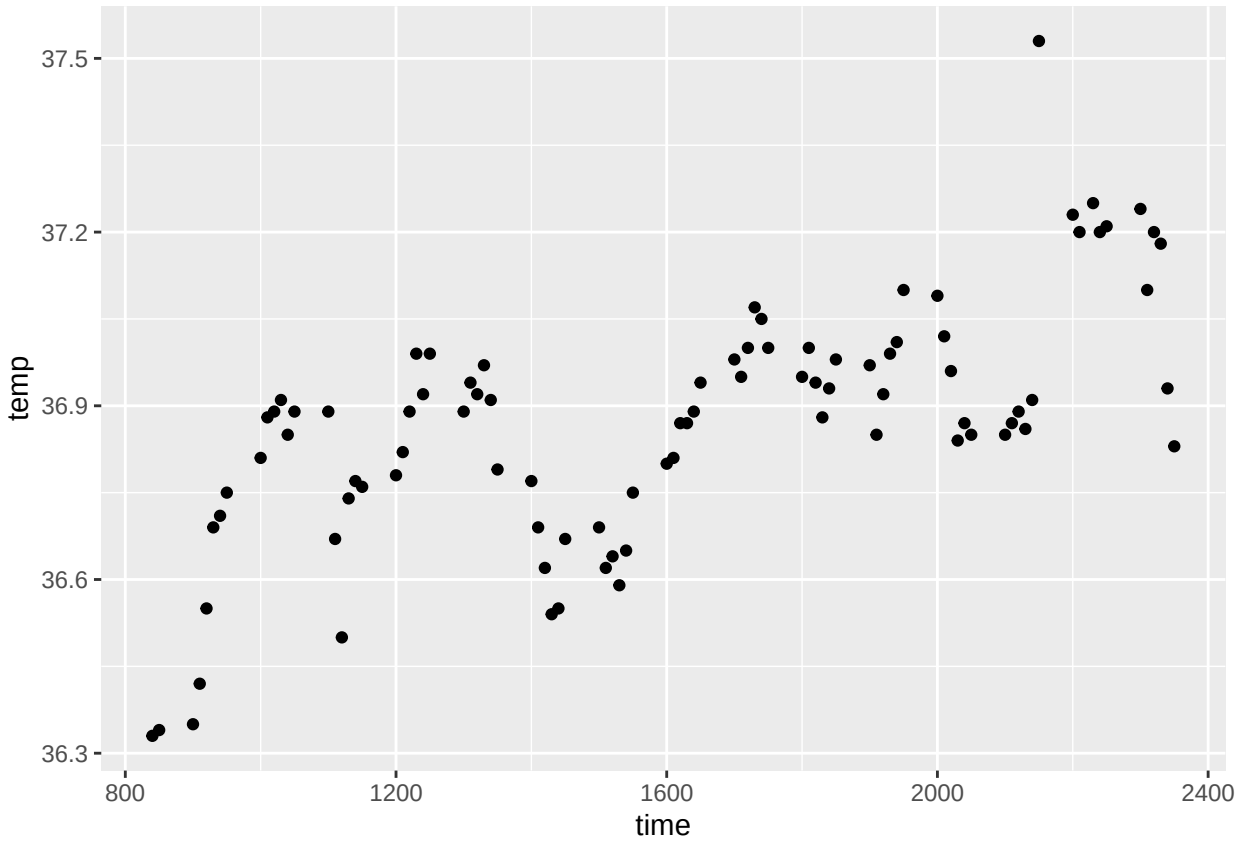
```
qplot(x = temp, data = beaver1)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



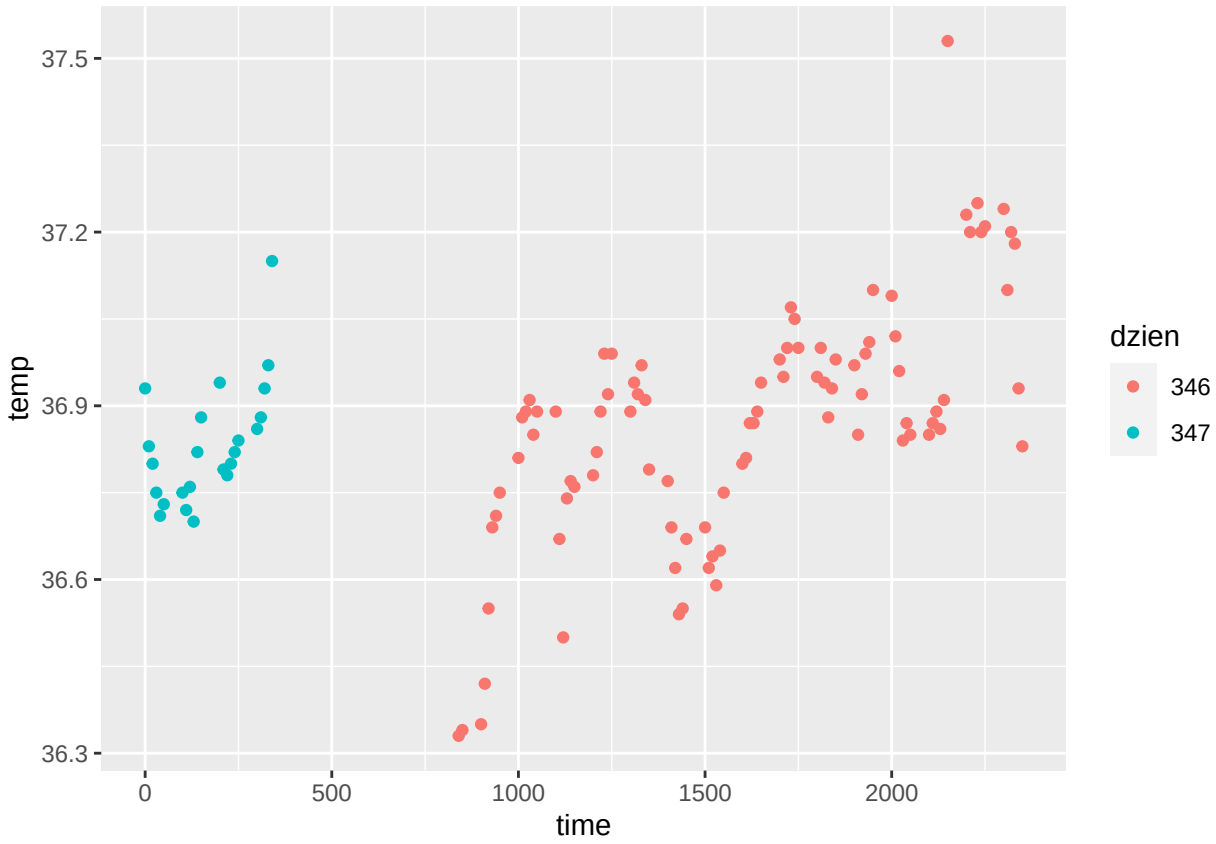
Gdy podamy dwie zmienne - `x` oraz `y` - domyślnym rodzajem wykresu będzie wykres punktowy (*scatterplot*).

```
qplot(x = time, y = temp, data = beaver1[beaver1$day == 346,])
```



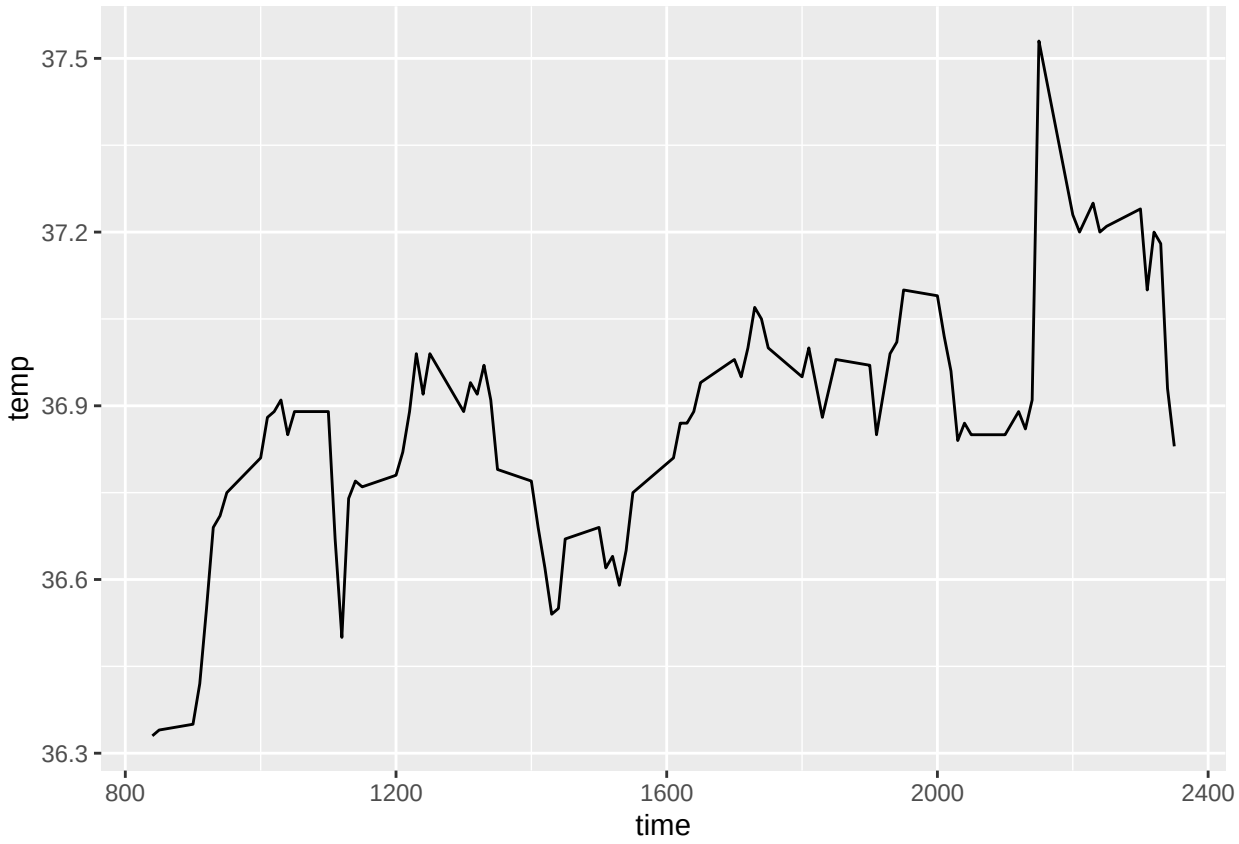
Przekazując kolumnę z wektorem typu `factor` jako argument `color` możemy ustalić inne kolory ze względu na wartość tej kolumny. W poniższym przypadku punkty dotyczące obserwacji w obu dniach oznaczone są innymi kolorami. Warto zauważyć, że `ggplot2` automatycznie stworzył legendę (czego nie robią funkcje wbudowane w R!).

```
beaver1$dzien <- as.factor(beaver1$day)
qplot(x = time, y = temp, color = dzien, data = beaver1)
```



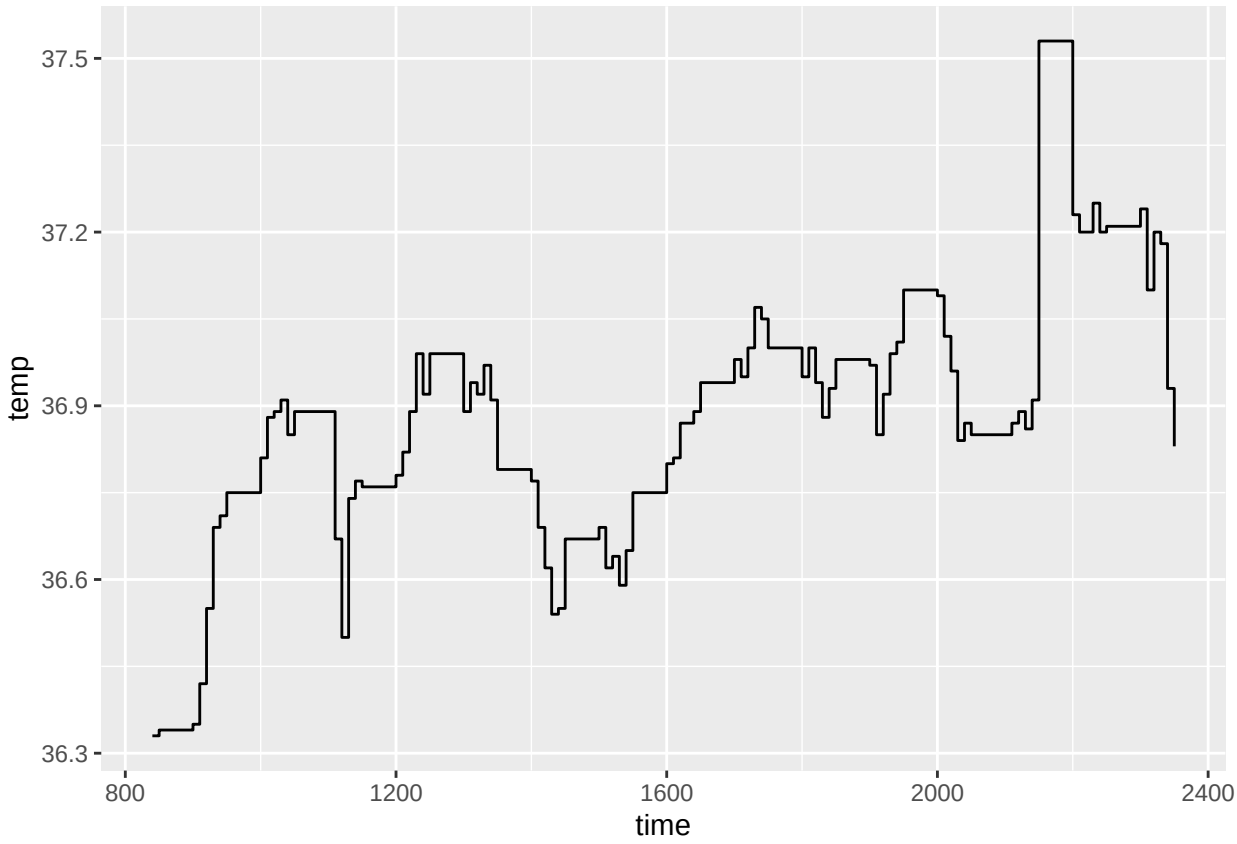
Za pomocą argumentu `geom` możemy zmienić typ wykresu. Listę wszystkich możliwości z łatwością znajdą Państwo w internecie. Pokażemy sobie jednak kilka z nich. Ustawiając argument `geom` na `"line"` możemy stworzyć wykres liniowy.

```
qplot(x = time, y = temp, geom = 'line', data = beaver1[beaver1$day == 346,])
```



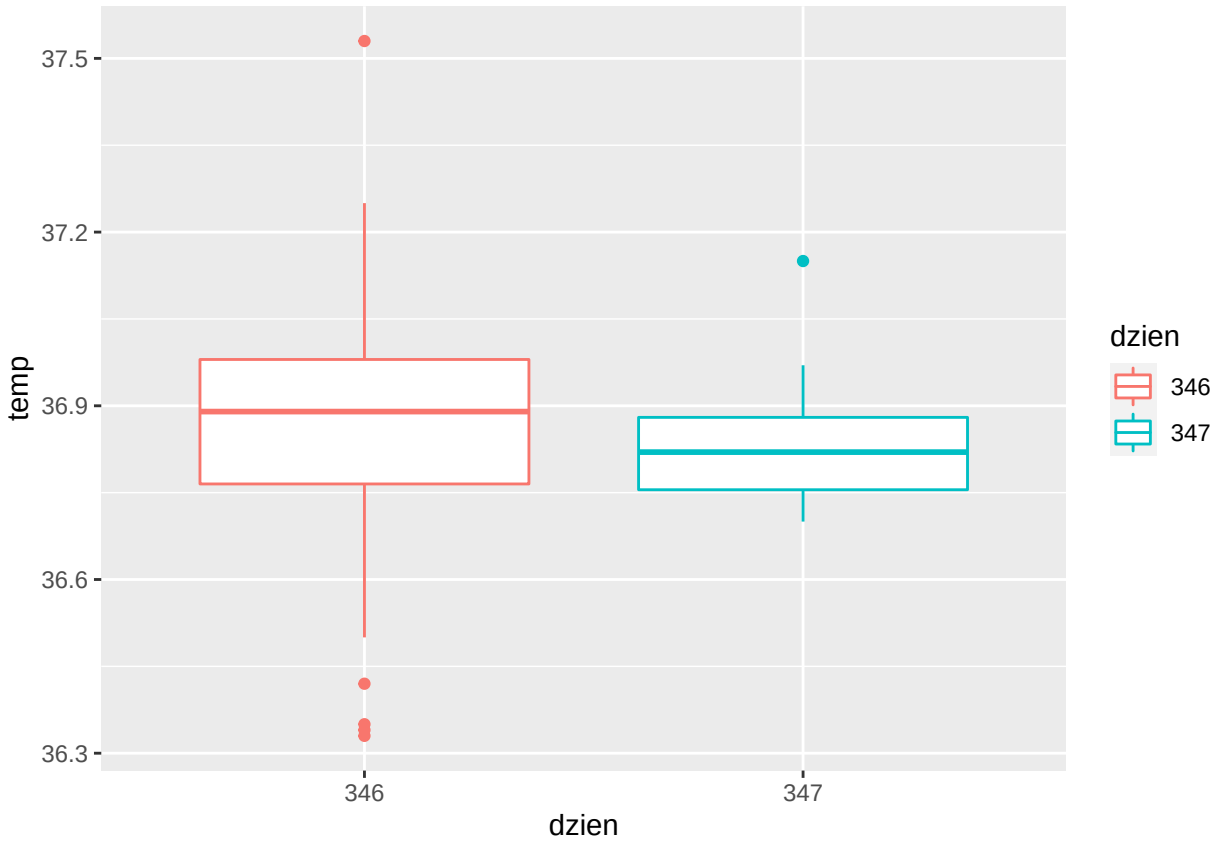
`"step"` stworzy wykres schodkowy.

```
qplot(x = time, y = temp, geom = 'step', data = beaver1[beaver1$day == 346,])
```



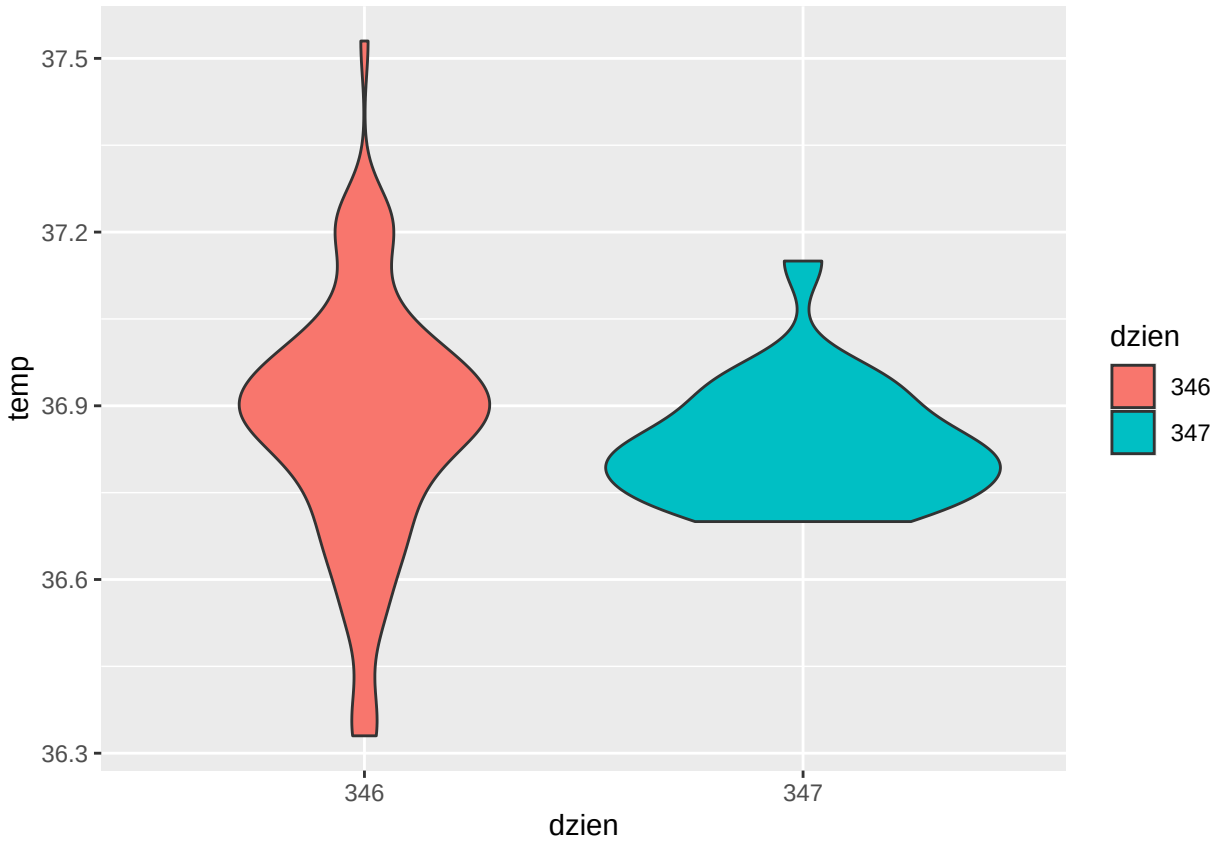
Jeśli do funkcji `plot` przekażemy dwie zmienne, z których jedna będzie kategorialna (`factor`), to `ggplot2` stworzy jeden ze znanych Państwu typów wykresów - na przykład wykres budełkowy. Jak jednak widzimy w przypadku wykresu pudełkowego argument `color` nie zadziałał tak, jak byśmy chcieli - pokolorował tylko linie na wykresie.

```
qplot(x=dzien, y = temp, color = dzien, geom = 'boxplot', data = beaver1)
```



Jeżeli chcielibyśmy pokolorować również wypełnienie, musielibyśmy użyć argumentu `fill`.

```
qplot(x=dzien, y = temp, fill = dzien, geom = 'violin', data = beaver1)
```



Tworzenie wykresów w języku grafiki `ggplot2`

`qplot` przydaje się przede wszystkim wtedy, kiedy chcemy stworzyć wykres *szybko* lub *nie jest skomplikowany*. Nie pozwala jednak wykorzystać wszystkich możliwości, jakie ma ten pakiet. Spróbujmy więc stworzyć wykres za pomocą "gramatyki grafiki" jaki implementuje `ggplot2`.

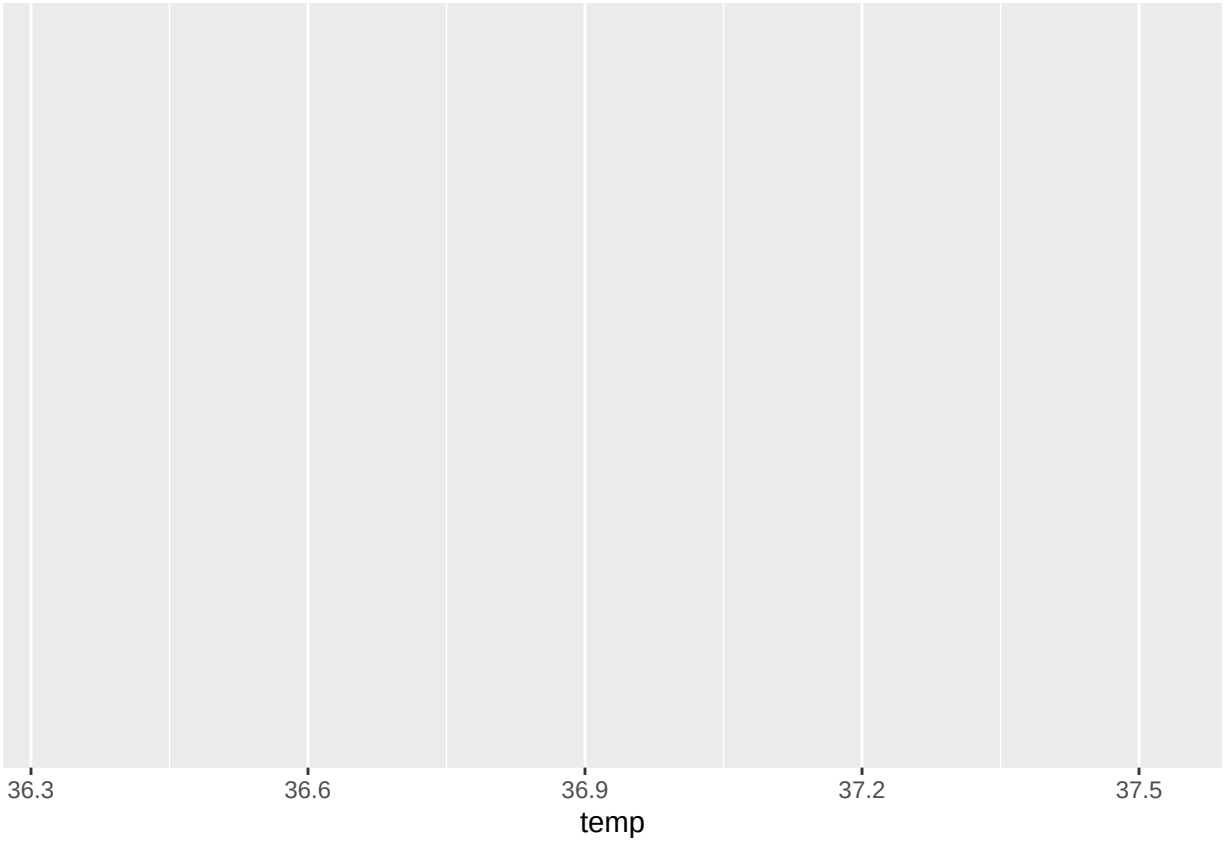
Jedna zmienna (lub dwie zmienne i jedna dyskretna)

Spróbujmy na początku stworzyć obiekt reprezentujący nasz wykres. Robimy to za pomocą funkcji `ggplot`.

```
ggplot(data = beaver1)
```

Otrzymaliśmy puste płótno. Dlaczego? Oczywiście nie ustaliliśmy co ma być na osi rzędnych, co na odciętych. Stwórzmy więc obiekt, w którym przekażemy nasze mapowanie. Odpowiada za to argument `mapping` (będziemy go dalej pomijać i przekazywać wartości pozycyjnie), w którym przekazujemy obiekt stworzomy za pomocą funkcji `aes`.

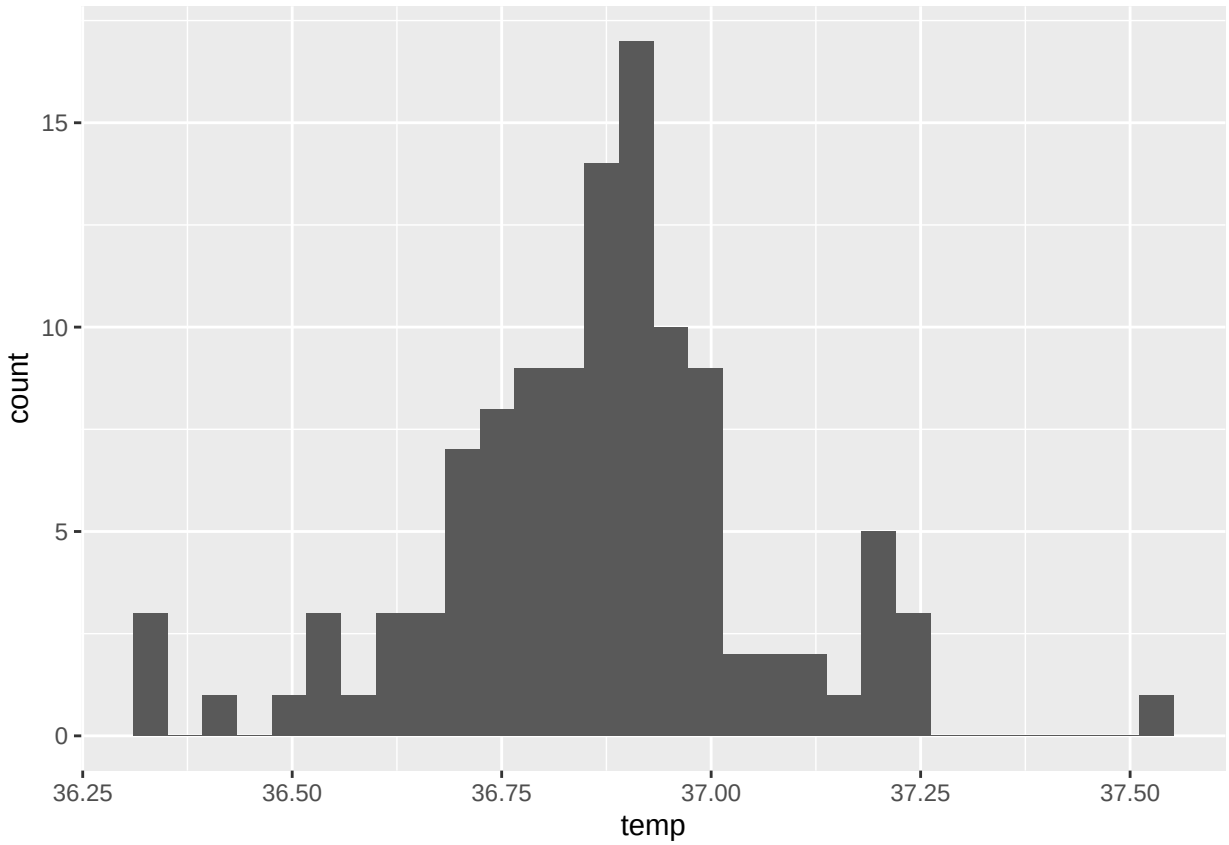
```
ggplot(data = beaver1, mapping = aes(x = temp))
```



Mamy już wykres, mamy już mapowanie - nawet dostaliśmy odpowiednie oznaczenia na osi. Nie mamy jednak na nim żadnego kształtu! Spróbujmy dodać do naszego wykresu jakiś kształt. Robimy to za pomocą funkcji `geom_rodzaj_kształtu`, którą "dodajemy" do już istniejącego obiektu.

```
ggplot(data = beaver1, aes(x = temp)) +
  geom_histogram()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

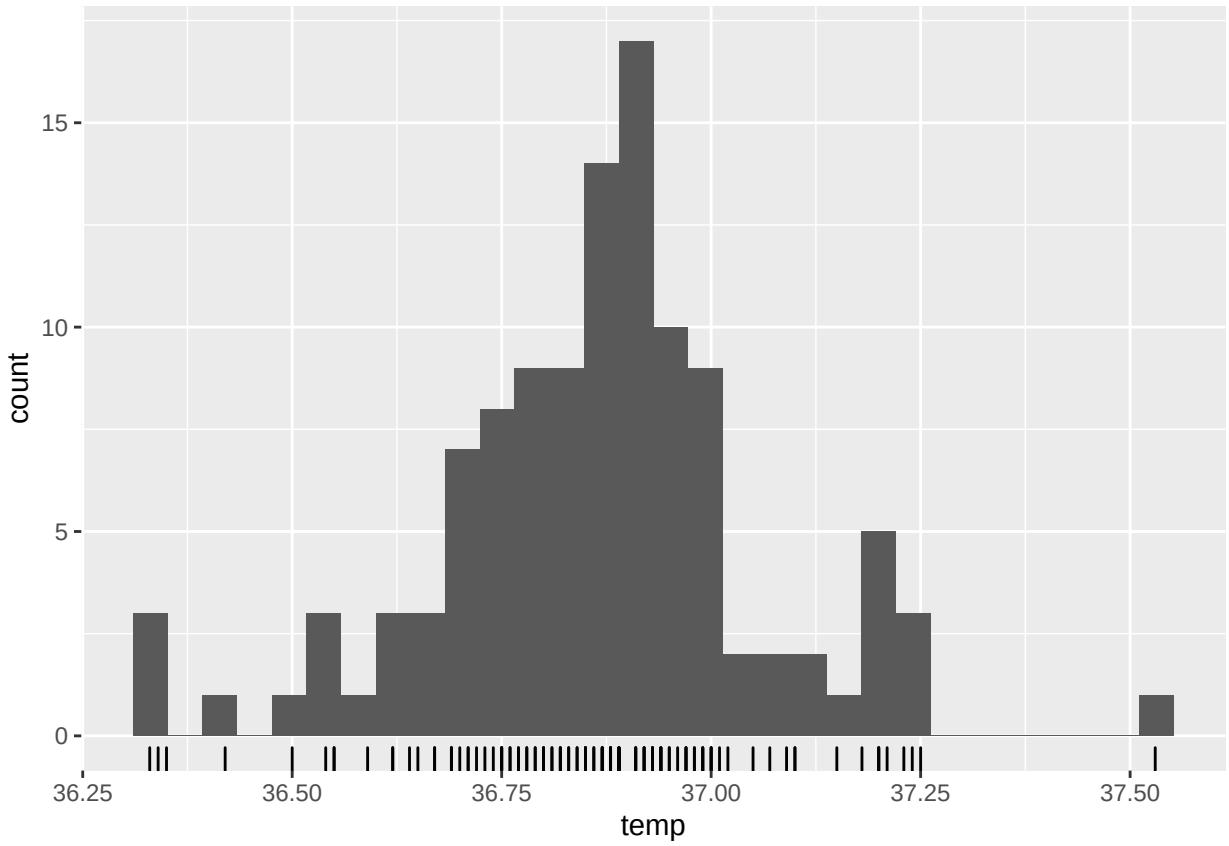


Stworzyliśmy histogram! Jak Państwo widzą wygląda chyba troszkę ładniej niż standardowy histogram w R.

No dobrze, ale czy możemy dodać jakiś inny kształt? Spróbujmy dodać małe kreseczki reprezentujące poszczególne obserwacje za pomocą `geom_rug`.

```
ggplot(data = beaver1, aes(x = temp)) +
  geom_histogram() +
  geom_rug()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



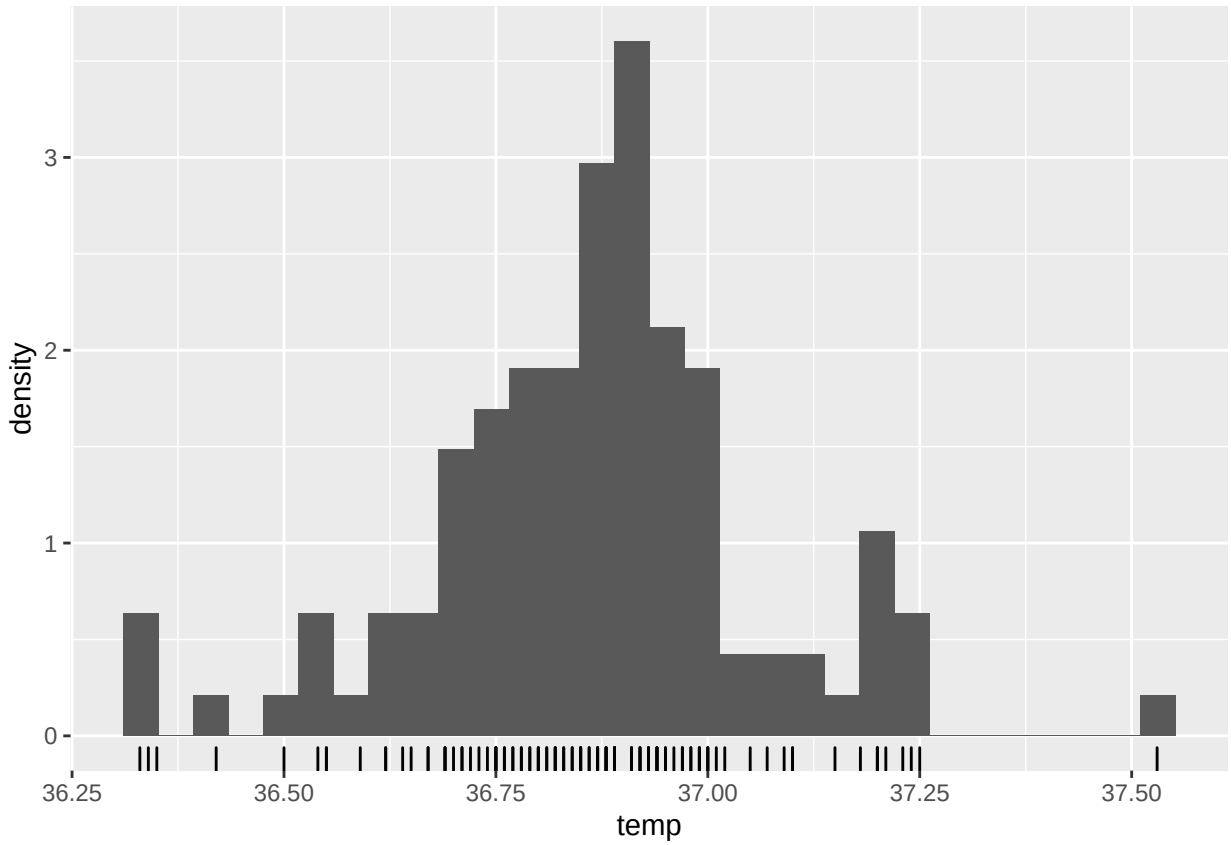
Wygląda to całkiem nieźle!

Mogą się Państwo zastanawiać skąd `ggplot2` wie, co i gdzie powinien narysować wówczas, gdy "dodajemy" kolejne kształty? `ggplot2` uwzględnia dorysowując do wykresu kolejne kształte mapowanie (`aes`) odziedziczone z głównego wykresu (= obiektu stworzonego przez funckję `ggplot`). Dlatego właśnie możemy po prostu dodawać kolejne kształty.

Nie musimy godzić się na mapowanie "odziedziczone" z głównego wykresu. Dla każdego kształtu (*geoma*) możemy mapowanie zmienić lub uzupełnić o dodatkowe elementy. W tym celu używamy funkcji `aes`. W poniższym przykładzie użyliśmy wbudowanego w `ggplot2` słowa kluczowego `..density..` za pomocą którego zmieniliśmy wartości na osi y.

```
ggplot(data = beaver1, aes(x = temp)) +
  geom_histogram(aes(y = ..density..)) +
  geom_rug()
```

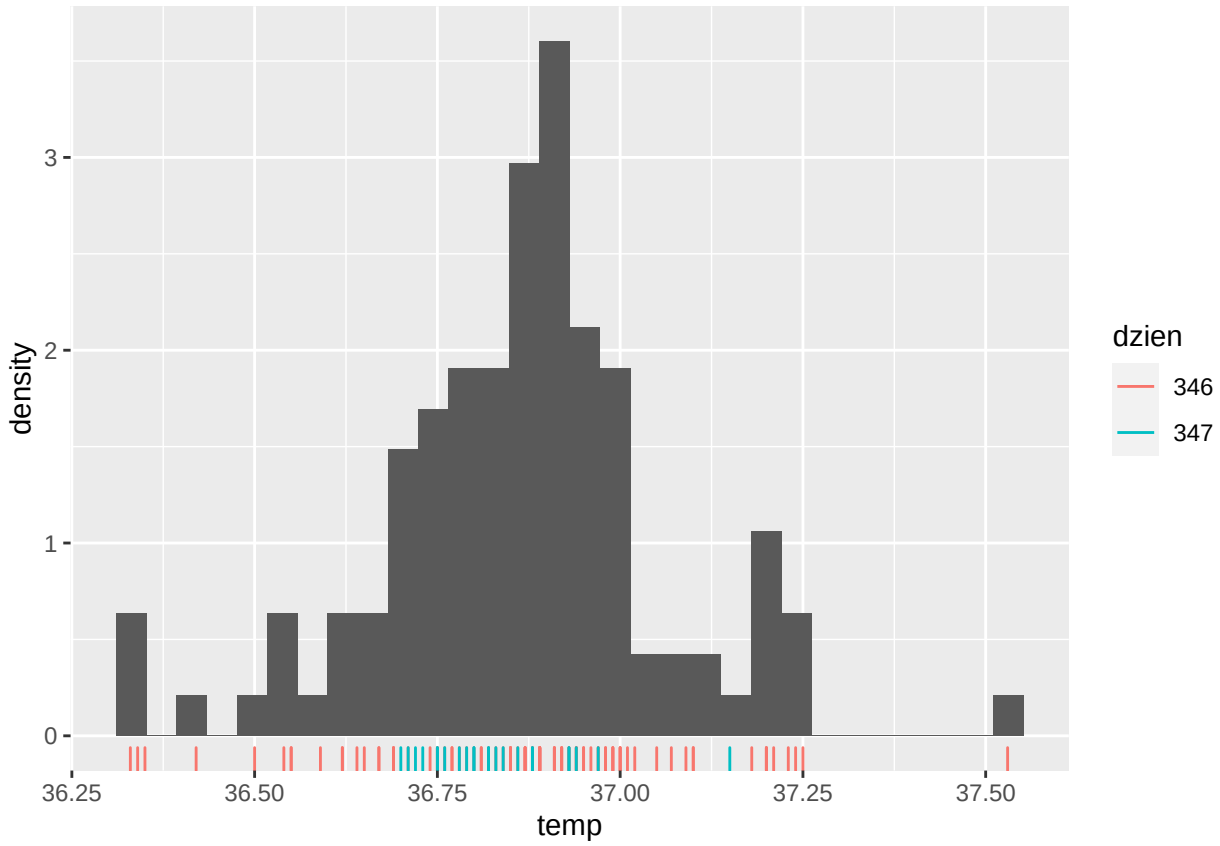
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Chociaż kolejne geomy dziedziczą mapowanie po pierwszym obiekcie, to możemy je oczywiście nadpisać. W tym przypadku podobnie jak w poprzednich dodaliśmy do naszego histogramu kreseczki za pomocą `geom_rug`, tym razem jednak dodaliśmy dodatkowe mapowanie - zmienna `dzien` ma być mapowana na kolor kreseczek.

```
ggplot(data = beaver1, aes(x = temp)) +
  geom_histogram(aes(y = ..density..)) +
  geom_rug(aes(color = dzien))
```

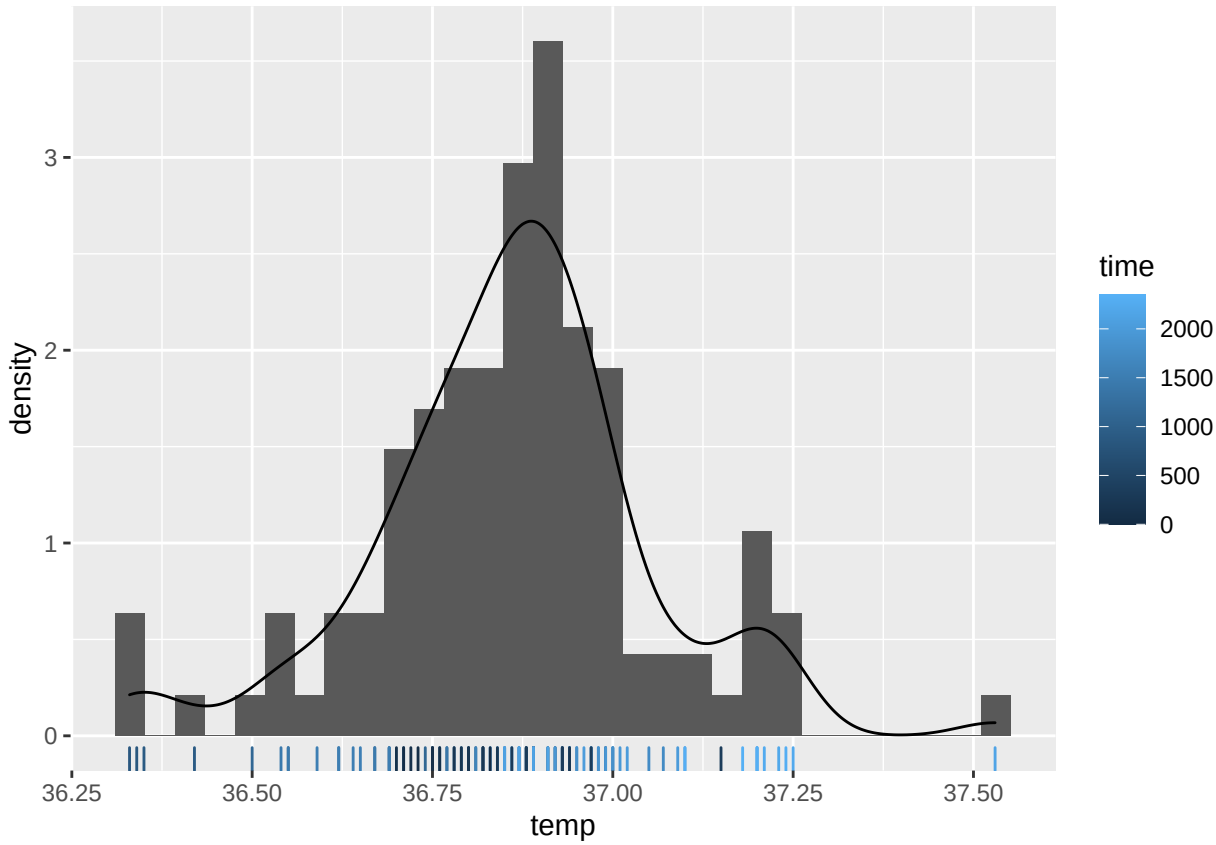
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Bez trudu możemy dodać jądrowy estymator gęstości dodając po prostu kolejny *geom* - tym razem `geom_density`. Proszę zwrócić uwagę, że zmieniliśmy jeszcze mapowanie kreseczek w `geom_rug`. `time` jest zmienną ciągłą (w miarę), więc `ggplot2` automatycznie dobrał ciągłą skalę kolorów i stworzył odpowiednią legendę.

```
ggplot(data = beaver1, aes(x = temp)) +
  geom_histogram(aes(y = ..density..)) +
  geom_rug(aes(color = time)) +
  geom_density()
```

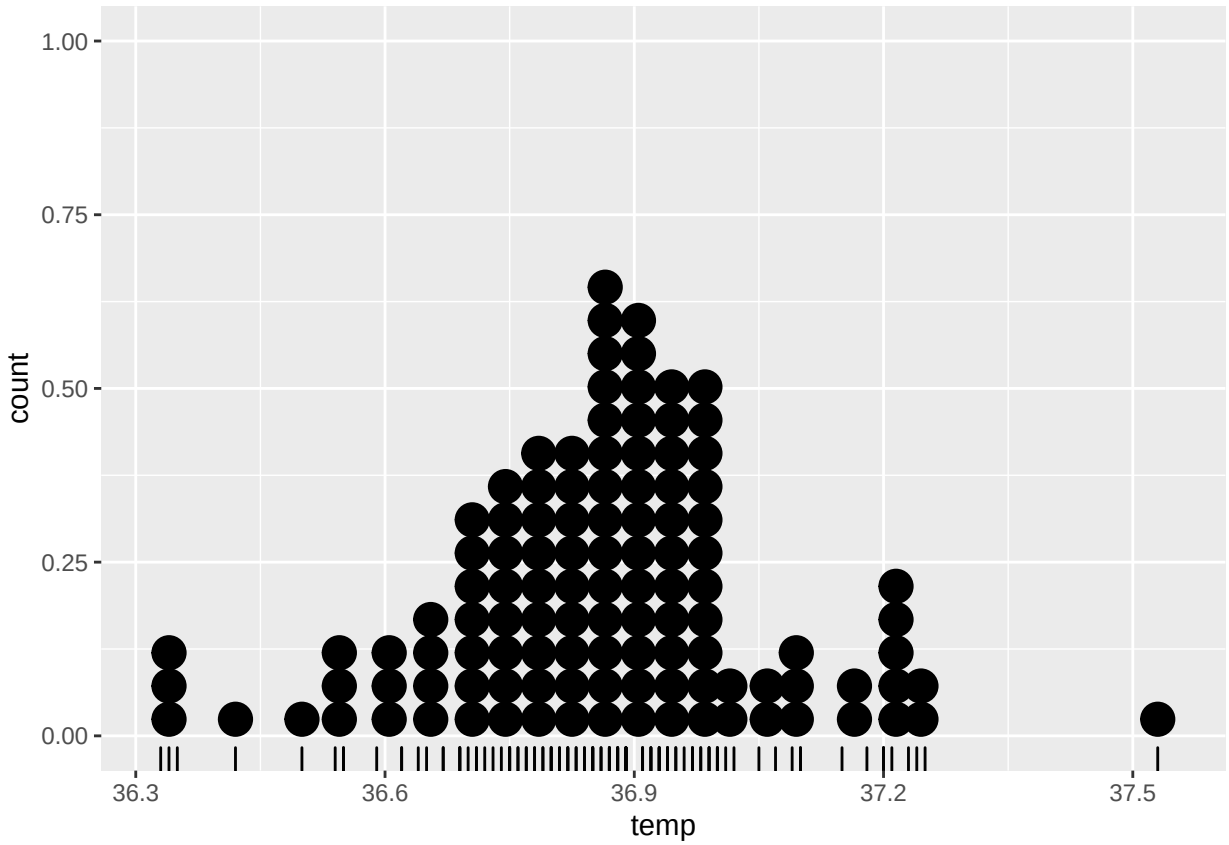
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



`ggplot2` udostępnia oczywiście również rodzaje geomów bardziej wymyślne niż zwykłe histogramy. Na poniższym wykresie mają Państwo inny geom - `geom_dotplot`, który zamiast prostokątów rysuje kropki.

```
ggplot(data = beaver1, aes(x = temp)) +
  geom_dotplot() +
  geom_rug()
```

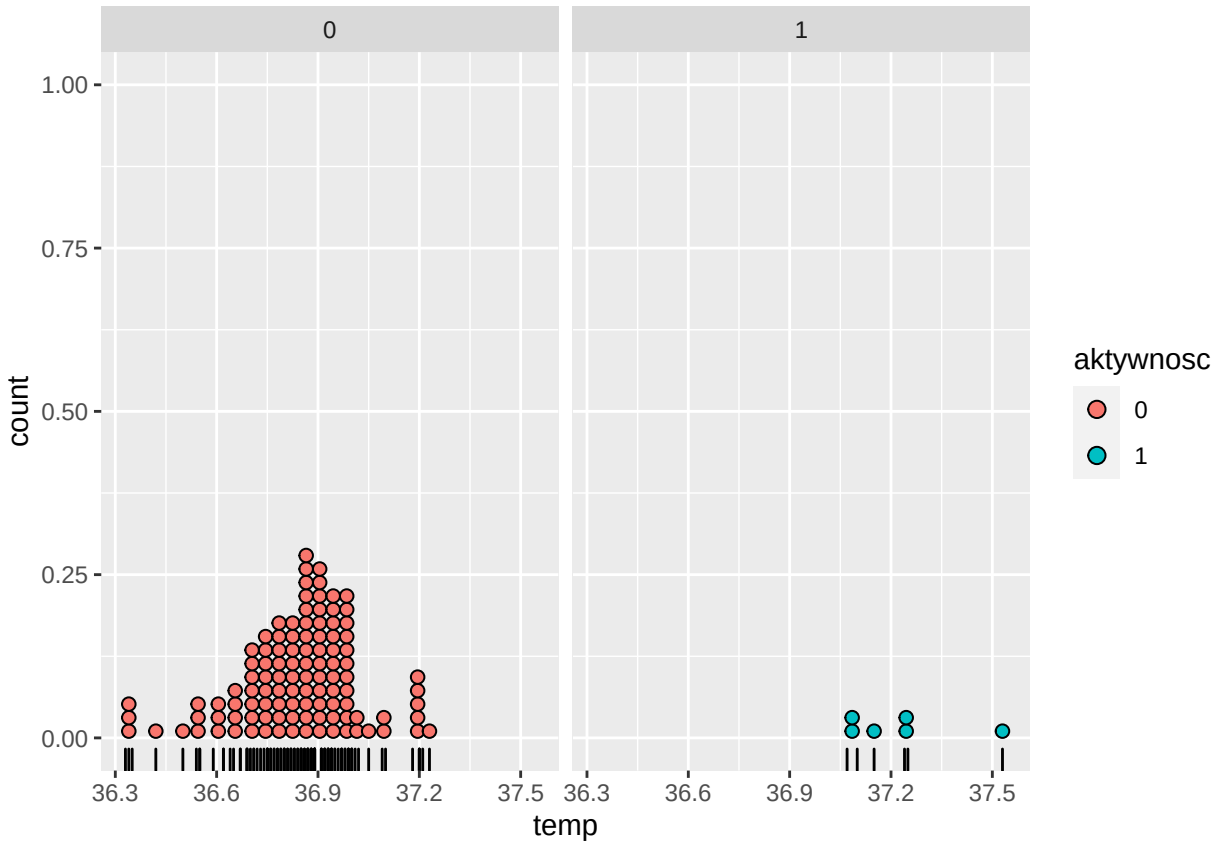
```
## Bin width defaults to 1/30 of the range of the data. Pick better value with `binwidth`.
```



Zwróćmy teraz uwagę na to, że w naszym zbiorze danych mamy dane z okresów aktywności i braku aktywności. Chcielibyśmy je rozdzielić, żeby lepiej zobrazować różnicę między nimi. Użyjemy do tego funkcji `facet_grid`, która w sposób automatyczny dzieli nasz wykres na panele według zadanego czynnika.

```
beaver1['aktywnosc'] <- as.factor(beaver1$'activ')
ggplot(data = beaver1, aes(x = temp, fill = aktywnosc)) +
  geom_dotplot() +
  geom_rug() +
  facet_grid(.~aktywnosc) # możemy wybrać dwie zmienne, składnia to `zmienna1 ~ zmienna2`
```

```
## Bin width defaults to 1/30 of the range of the data. Pick better value with `binwidth`.
```

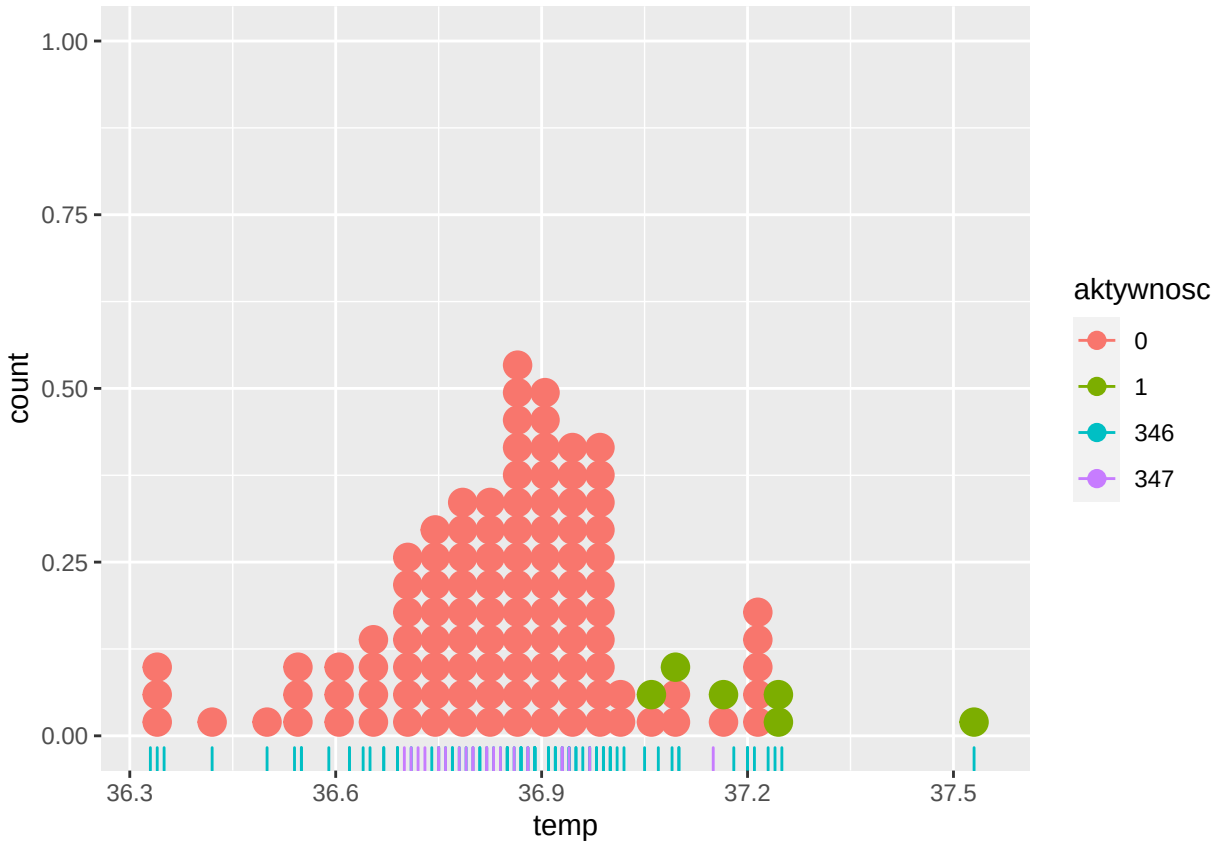


Możemy próbować uatrakcyjnić nasz wykres. Proszę zauważyć, że każdy *geom* ma tutaj inne mapowanie - wysokosc słupków kropek reprezentuje liczbe zliczeń w danym "koszyku" hisgoramu, kolor kropek - aktywność bobra, kreseczki - dokładna wartość każdej obserwacji. Z kolei kolor kreseczek reprezentuje dzień, w którym ich dokonywaliśmy. Każda z tych rzeczy była osobną "warstwą", którą definiowaliśmy w gramatyce grafiki ggplot2.

```
ggplot(data = beaver1, aes(x = temp)) +
  geom_dotplot(aes(fill = aktywnosc, color = aktywnosc),
               binpositions="all", stackgroups = TRUE) +
  geom_rug(aes(color=dzien, fill = dzien))
```

```
## Warning: Ignoring unknown aesthetics: fill
```

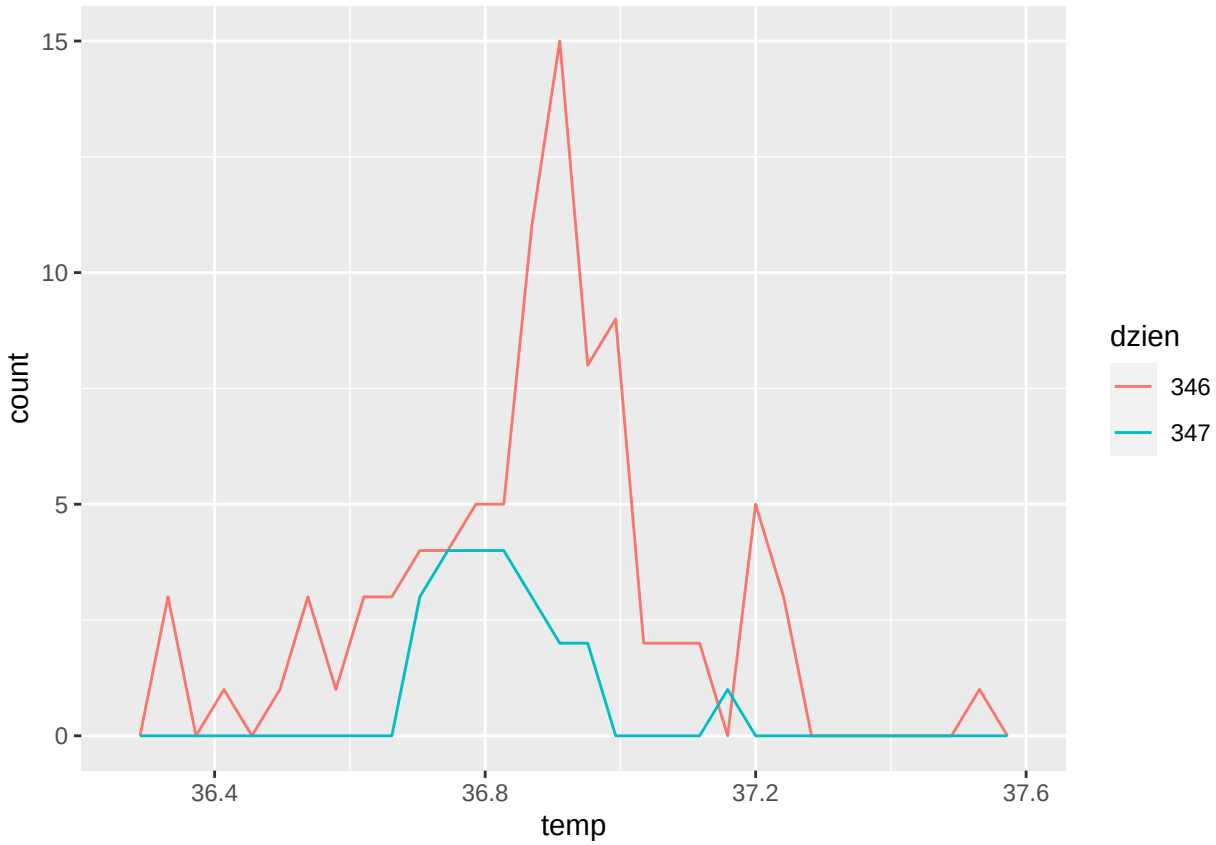
```
## Bin width defaults to 1/30 of the range of the data. Pick better value with `binwidth`.
```



Jeżeli chodzi o inne *geomy* w stylu histogramów to mamy jeszcze `geom_freqpoly`.

```
ggplot(data = beaver1, aes(x = temp, fill = dzien, color = dzien)) +
  geom_freqpoly()
```

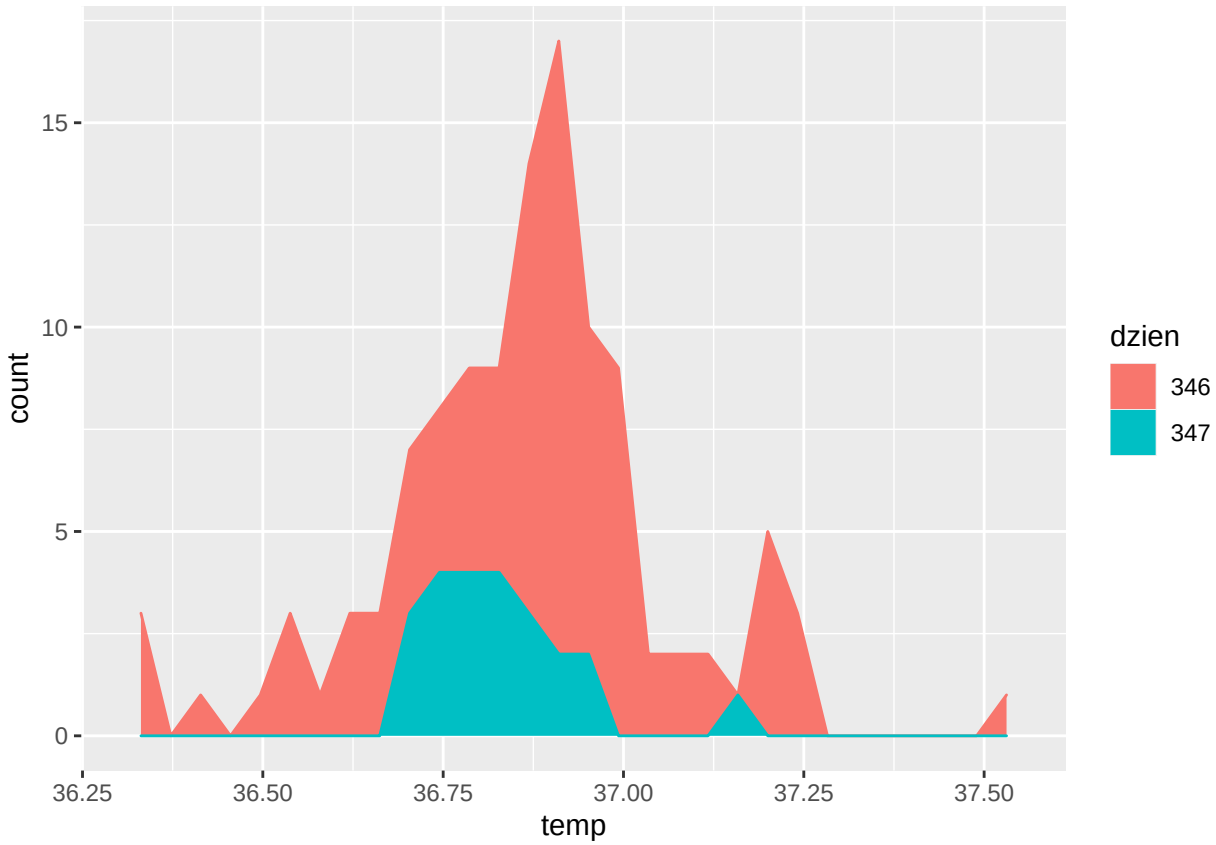
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Oraz `geom_area` (ten jest szczególnie efektowny).

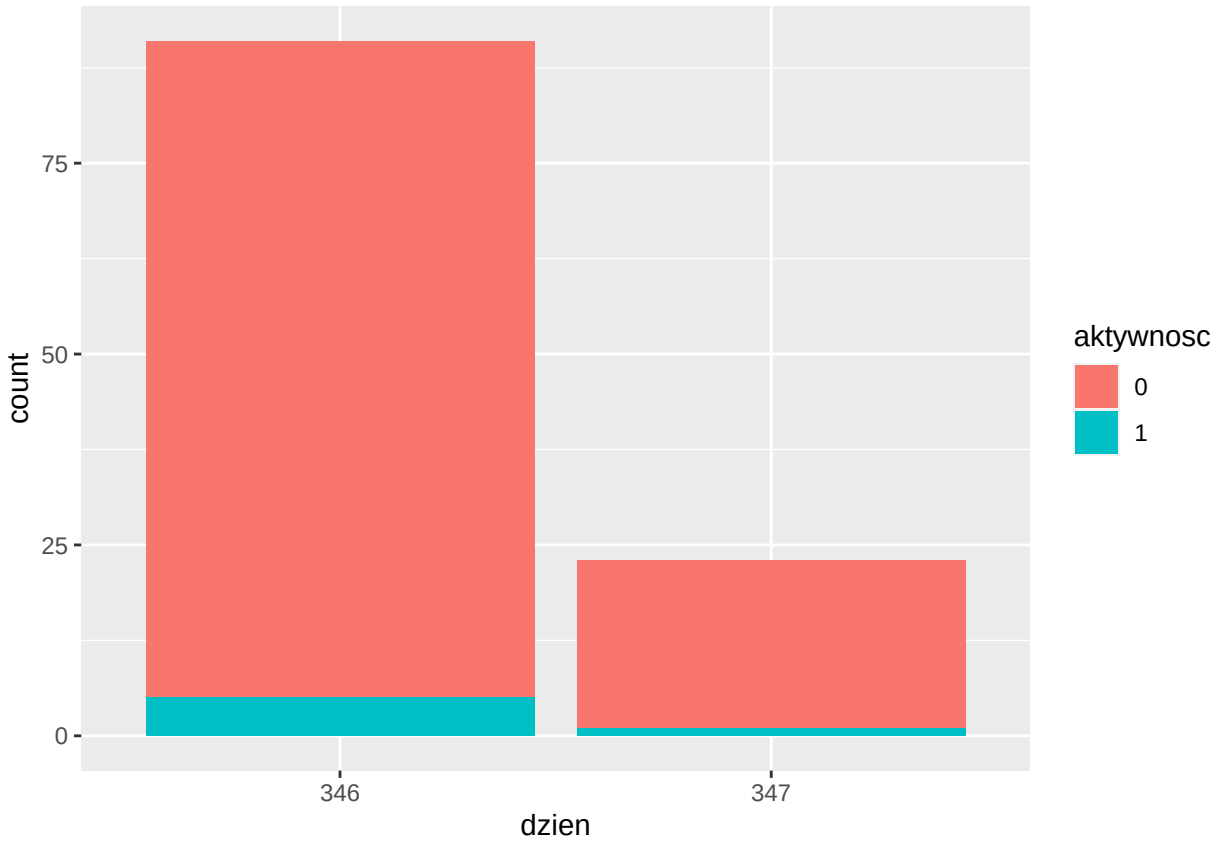
```
ggplot(data = beaver1, aes(x = temp, fill = dzien, color = dzien)) +
  geom_area(stat='bin')
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



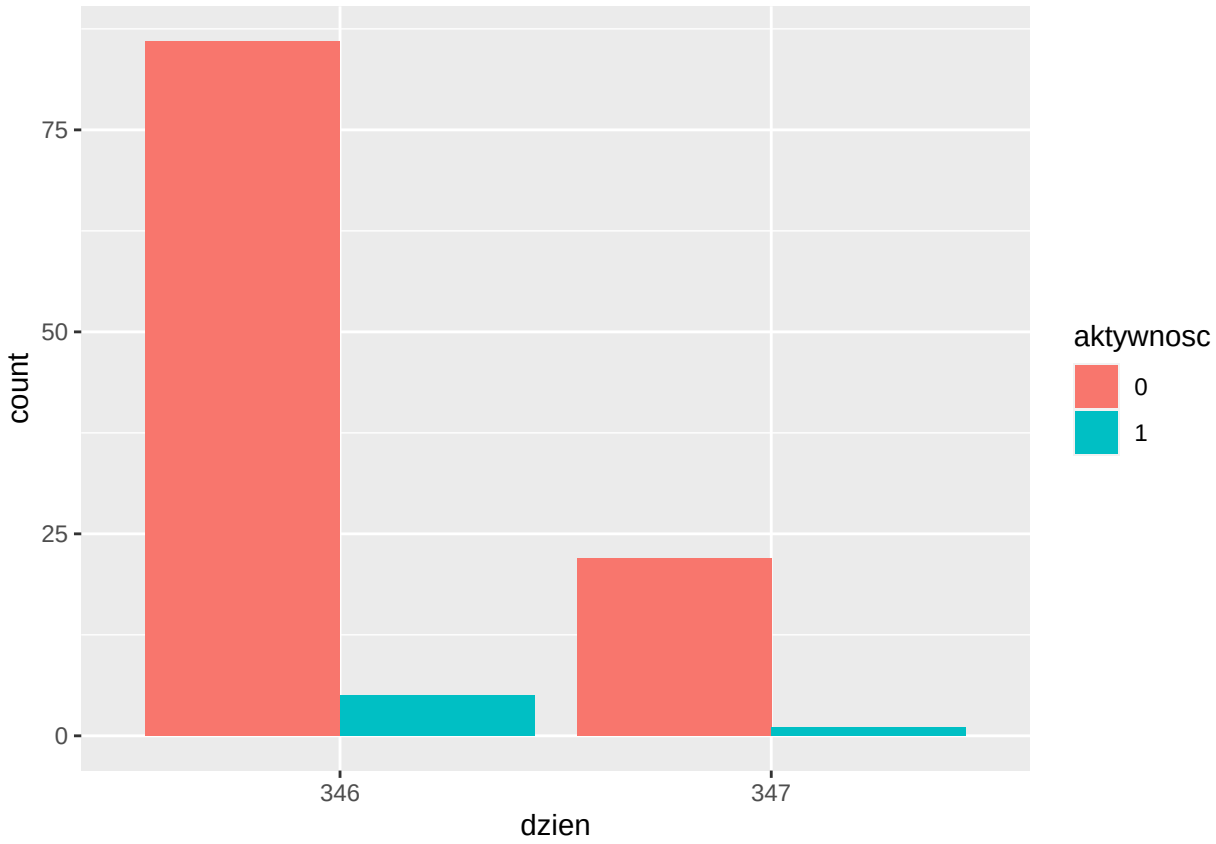
Jeżeli chodzi o wykresy słupkowe, to podobnie jak w standardowym R sami musimy policzyć wartości mające odpowiadać wysokości słupków. Jeśli mają to być zliczenia postąpimy więc tak:

```
ggplot(data = beaver1, aes(x = dzien, fill = aktywnosc)) +
  geom_bar()
```



Domyślnym zachowaniem jest ustawianie słupków na sobie, ale możemy je zmienić tak, by ustawiane były obok siebie.

```
ggplot(data = beaver1, aes(x = dzien, fill = aktywnosc)) +
  geom_bar(position= 'dodge')
```

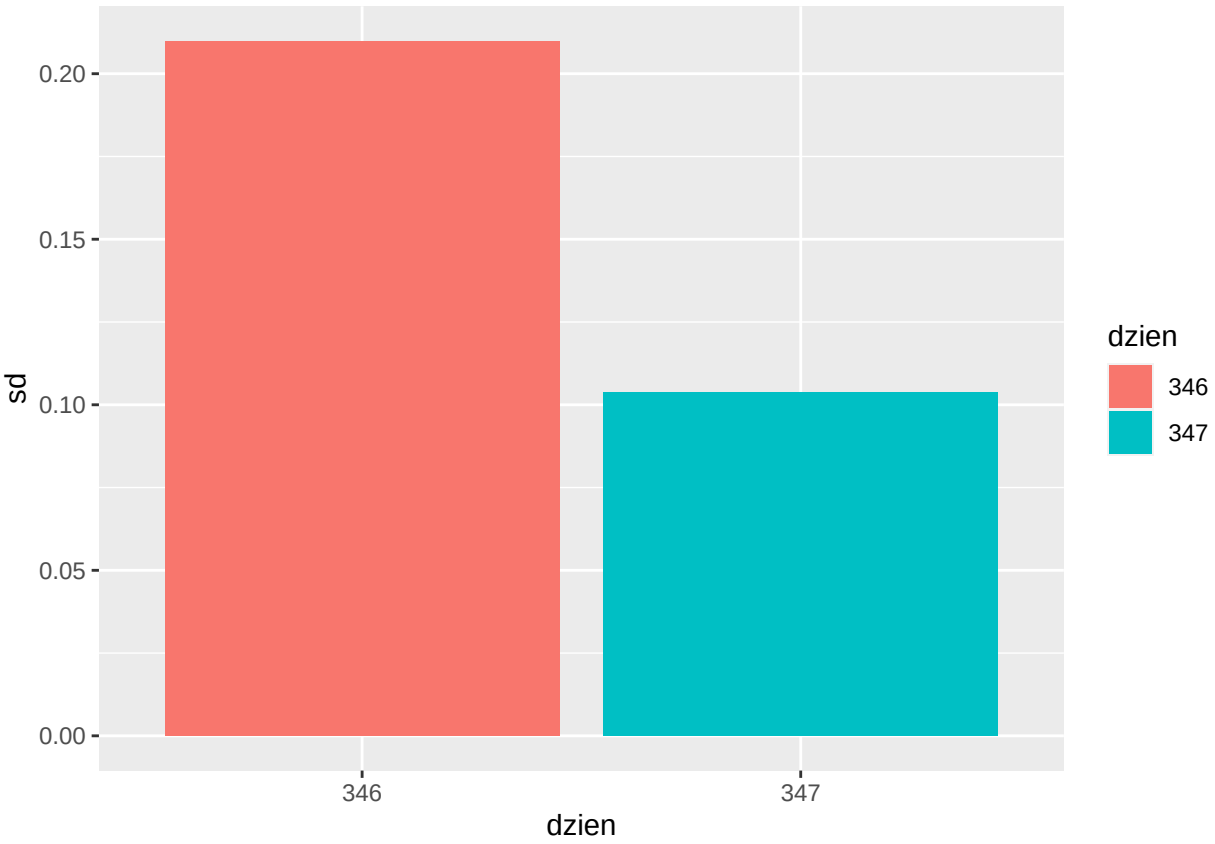


Czasami jednak nie chcemy reprezentować na wykresie słupkowym liczby zliczeń czegoś, ale np. średnią, odchylenie standardowe, medianę. Wówczas chyba najprościej jest stworzyć sobie nową ramkę danych, w której będą obliczone te wartości i przekazać jako argument `stat` funkcji `geom_bar` wartość `"identity"`.

```
df <- data.frame(sd = tapply(beaver1$temp, beaver1$dzien, sd), dzien = levels(beaver1$dzien))
df
```

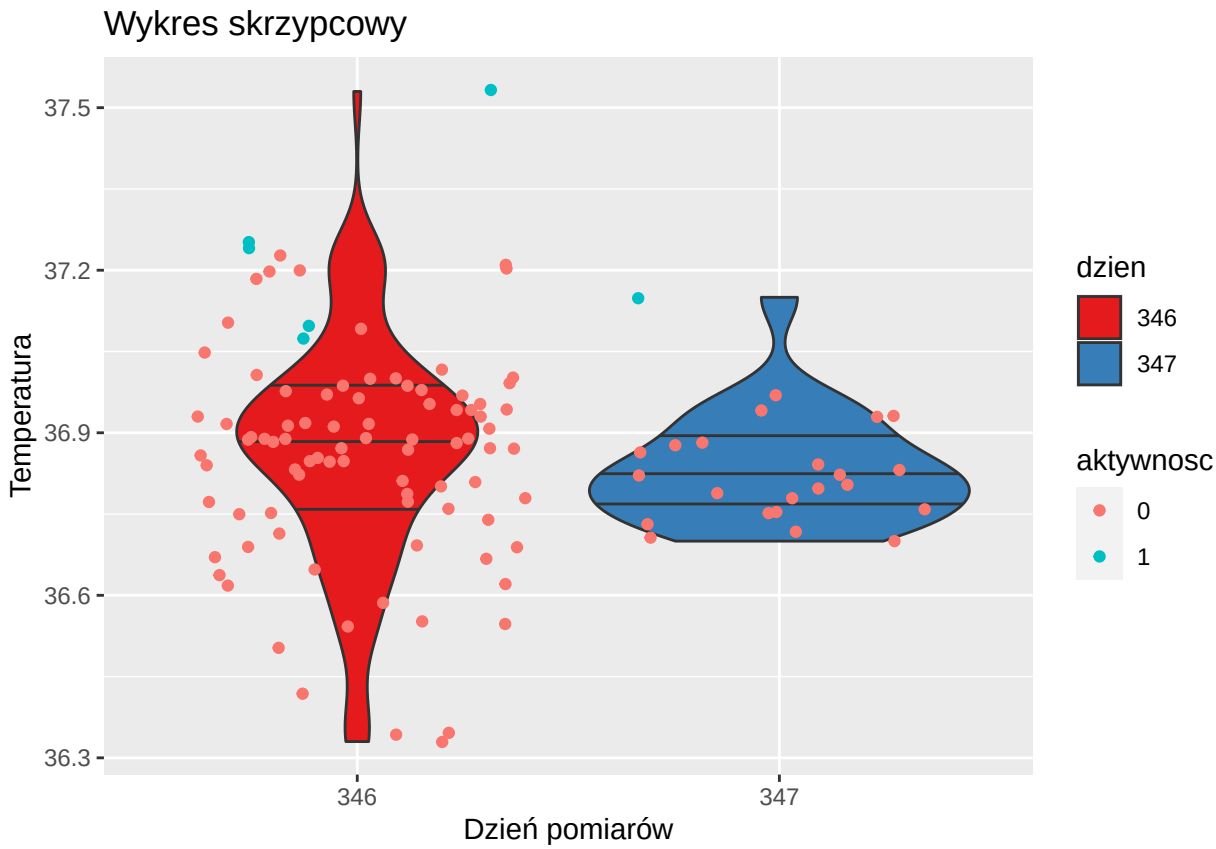
```
##            sd dzien
## 346 0.2098782   346
## 347 0.1038336   347
```

```
ggplot(data = df, aes(y = sd, x = dzien, fill = dzien)) +
  geom_bar(stat = 'identity')
```



Biblioteka `ggplot2` daje możliwość łączenia ze sobą kilka rodzajów wykresów. Na poniższym rysunku połączyliśmy wykres punktowy (z *jitterm* dla większej przejrzystosci) z wykresem skrzypcowym. Dodaliśmy również podpisy pod osiami oraz tytuł wykresu, zmieniliśmy też domyślne kolory (żeby mniej się zlewały).

```
ggplot(data = beaver1) +
  geom_violin(aes(x = dzien, y = temp, fill = dzien), draw_quantiles = c(0.25, 0.5, 0.75)) +
  geom_jitter(aes(x = dzien, y = temp, color = aktywnosc)) +
  scale_fill_brewer(palette='Set1') +
  ggtitle('Wykres skrzypcowy') +
  labs(x='Dzień pomiarów', y='Temperatura')
```



Dwie zmienne ciągłe.

W przypadku sytuacji, w której mamy dwie zmienne ciągłe postępujemy dokładnie tak samo. Do ilustracji posłużymy się zbiorem `iris`, który dotyczy kwiatków.

```
head(iris)
```

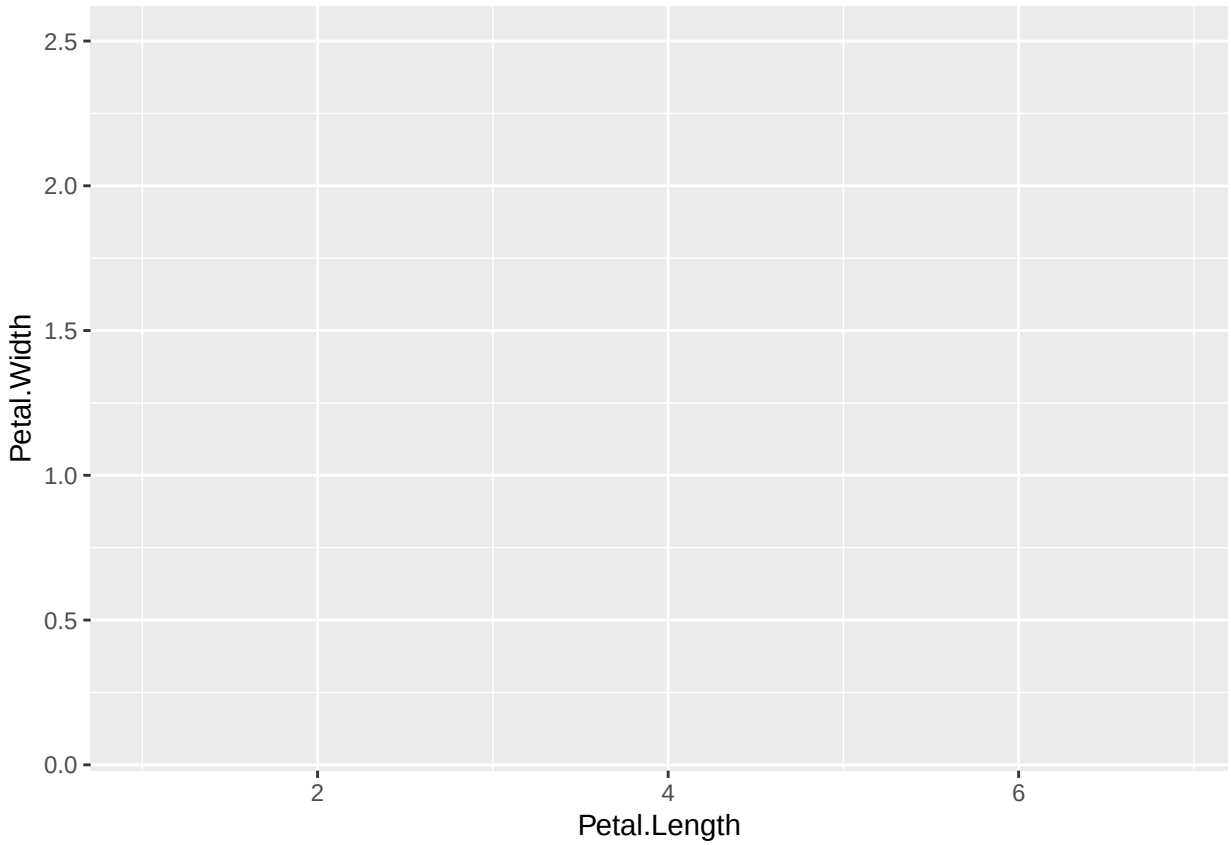
```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

Znów, samo wywołanie funkcji `ggplot` nic nie daje.

```
ggplot(data = iris)
```

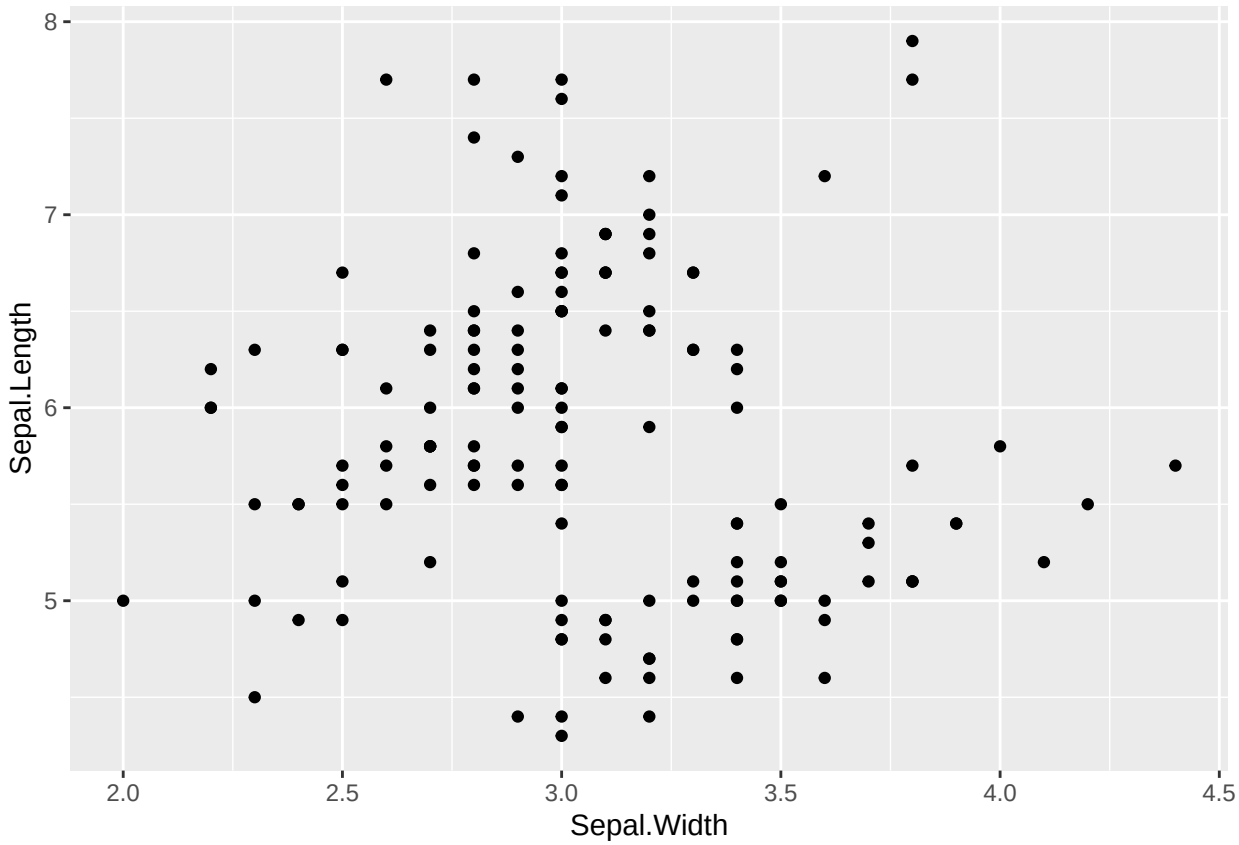
Nawet ustawienie mapowania nic nie daje, chociaż ustala nam osie i siatkę.

```
ggplot(data = iris) +
  aes(x = Petal.Length, y = Petal.Width)
```



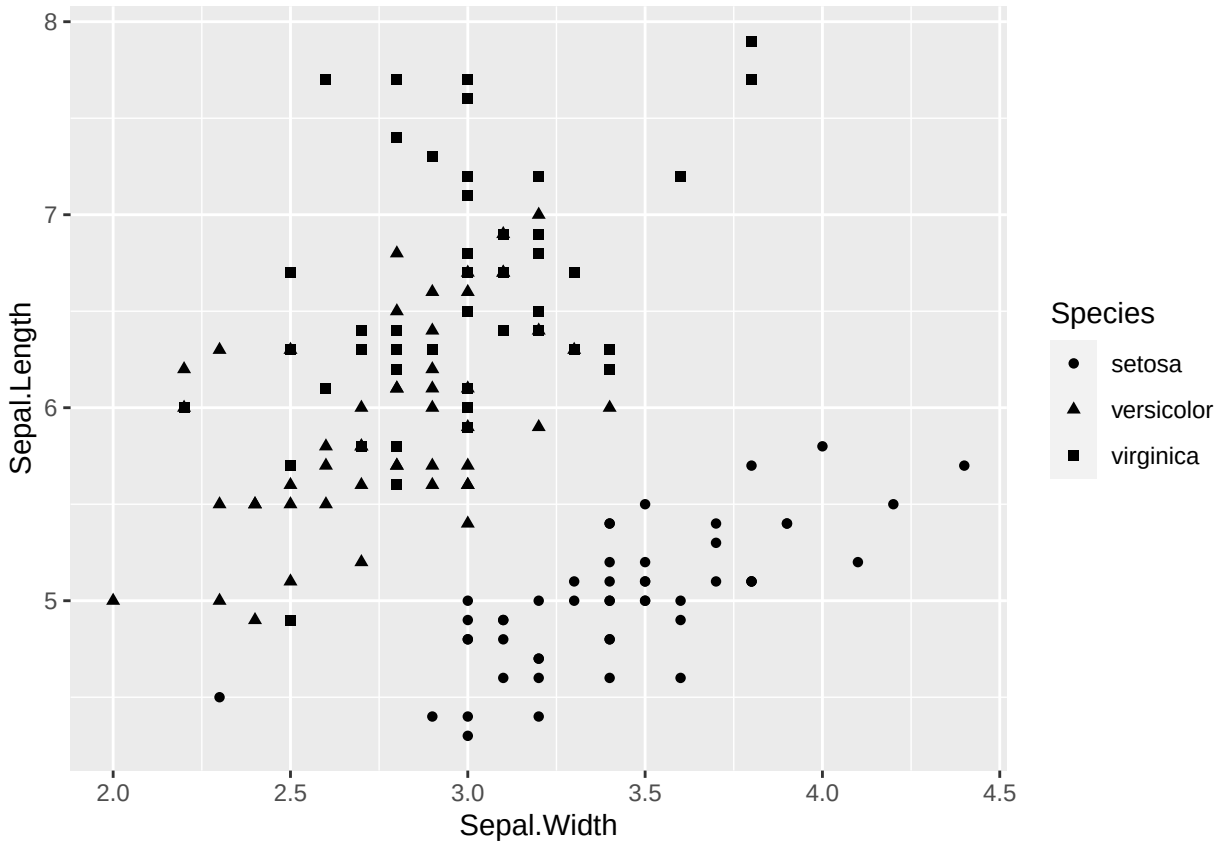
Pierwszy *geom* jaki omowimy to `geom_point` odpowiadający znanemu już nam wykresowy punktowemu.

```
ggplot(data = iris) +
  geom_point(aes(x = Sepal.Width, y = Sepal.Length))
```



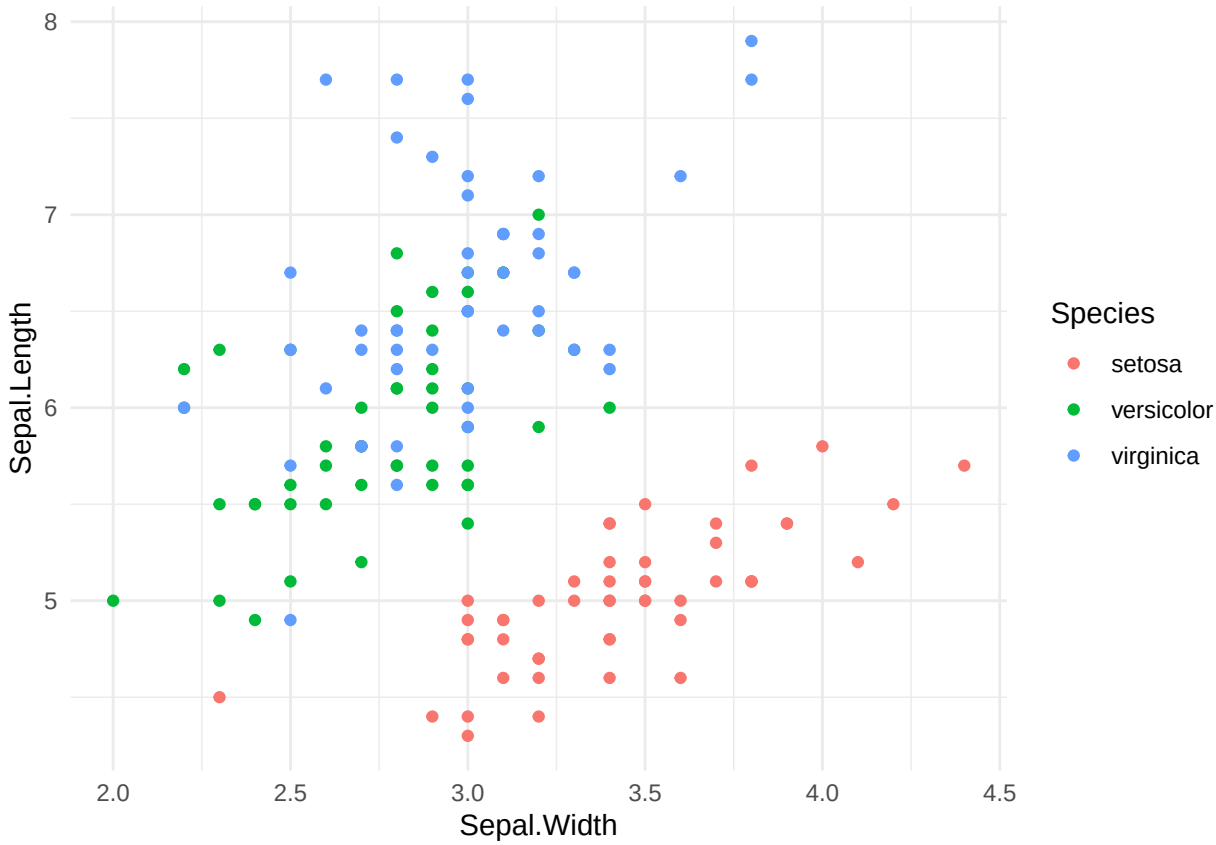
Kstzałt punktów jest jedną z możliwościa mapowania zmiennej dyskretnej.

```
ggplot(data = iris) +
  geom_point(aes(x = Sepal.Width, y = Sepal.Length, shape = Species))
```



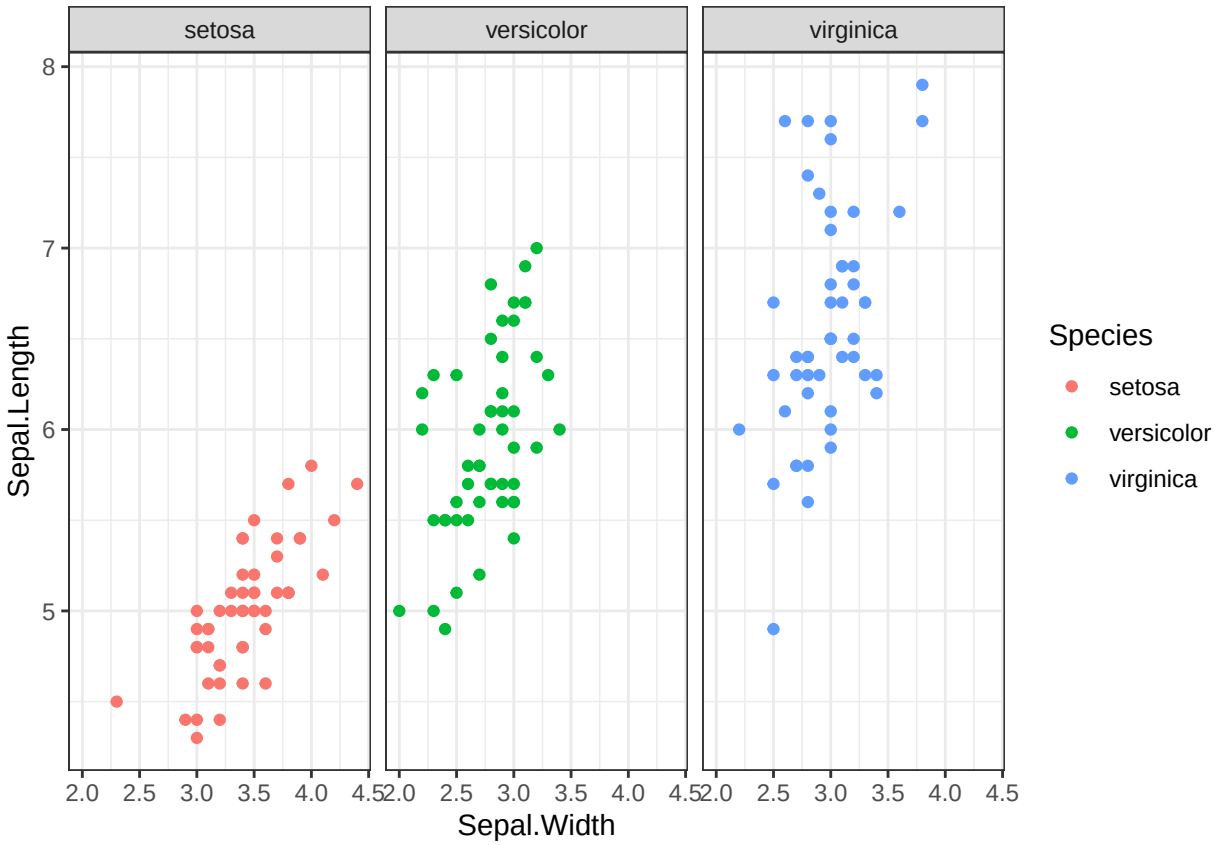
Podobnie jak poprzednio możemy dodać dodatkowe mapowanie, które trzecią zmienną - gatunek - mapuje jako kolor. Dodatkowo zmieniliśmy wygląd całego wykresu dodając jeden z kilku wbudowanych motywów (`theme_minimal`)

```
ggplot(data = iris) +
  geom_point(aes(x = Sepal.Width, y = Sepal.Length, col = Species)) +
  theme_minimal()
```



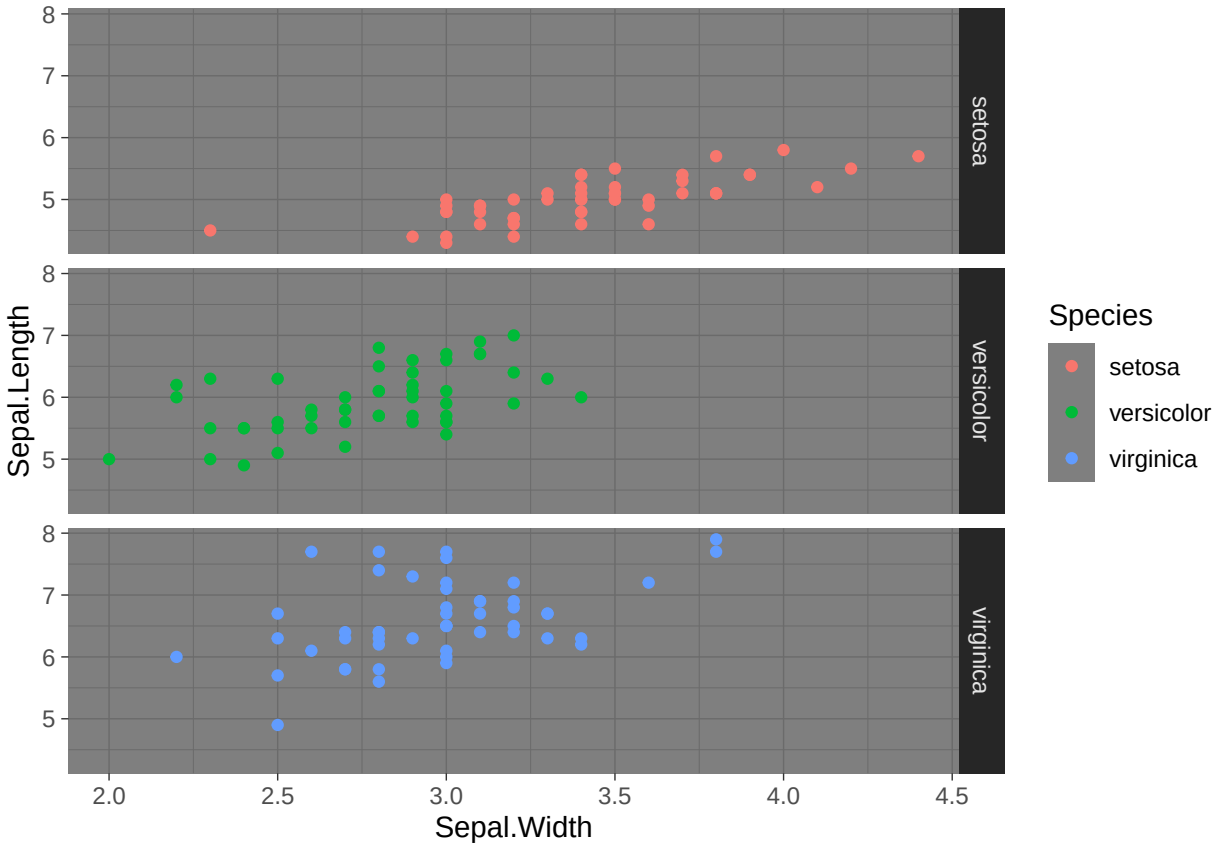
Możemy oczywiście umieścić nasze punkty na trzech osobnych panelach w znany nam już sposób.

```
ggplot(data = iris) +
  geom_point(aes(x = Sepal.Width, y = Sepal.Length, col = Species)) +
  facet_grid(. ~ Species) +
theme_bw()
```



Panele możemy umieścić w innej orientacji zmieniając kolejnośc zmiennych w funkcji `facet_grid`. Warto zwrócić uwagę, że podpisy pod kategoriami są prawidłowo w pozycji poziomej. Pokazuję również inny motyw (`theme_dark`)

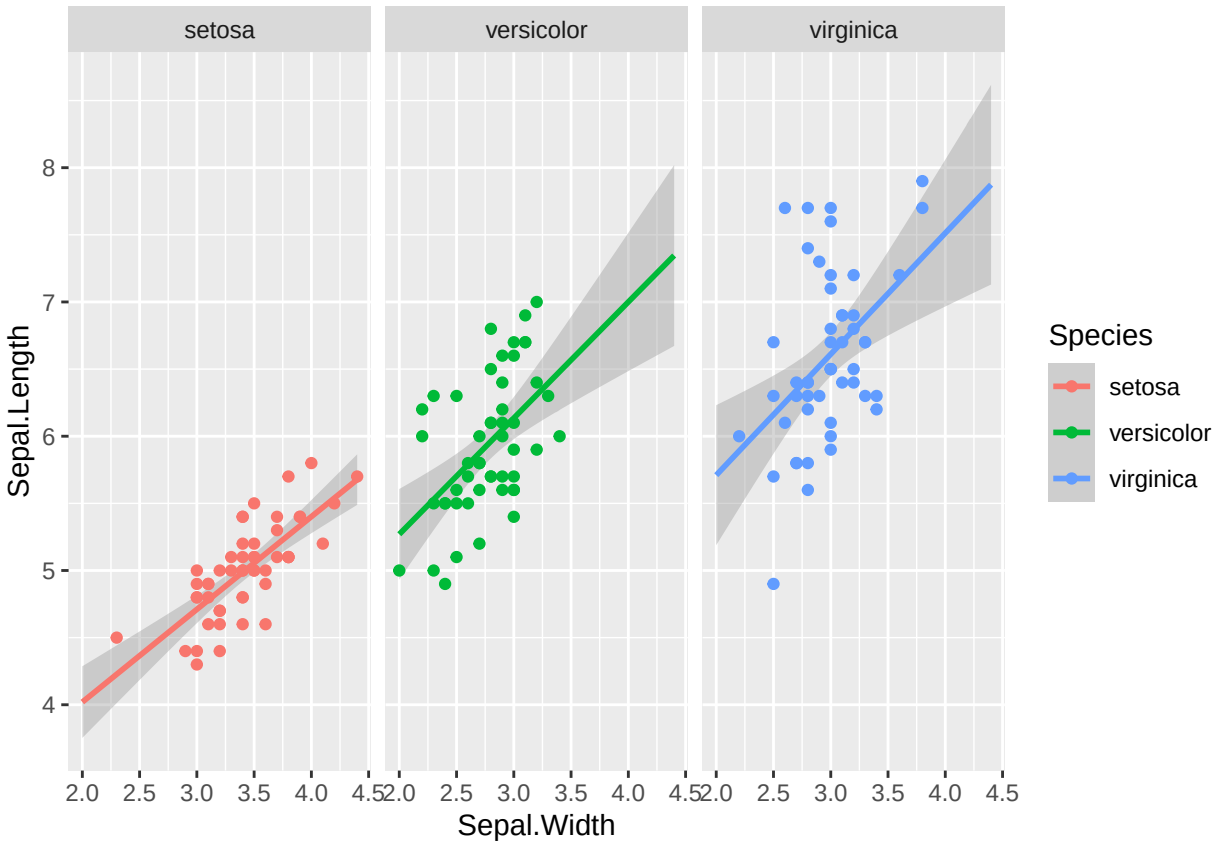
```
ggplot(data = iris) +
  geom_point(aes(x = Sepal.Width, y = Sepal.Length, col = Species)) +
  facet_grid(Species~.) +
  theme_dark()
```



Bardzo przydatne może być dla Państwa dodawanie linii regresji do wykresu punktowego. Pokazuje więc jak zrobić to za pomocą `stat_smooth`. Dopasowuje ona model liniowy do danych. Zacienione pole do przedział ufności (domyślnie 95%) dla linii regresji.

```
ggplot(data = iris) +
  facet_grid(.~Species) +
  stat_smooth(aes(x = Sepal.Width, y = Sepal.Length, col = Species), method = "lm", fullrange= TRUE)+
  geom_point(aes(x = Sepal.Width, y = Sepal.Length, col = Species))
```

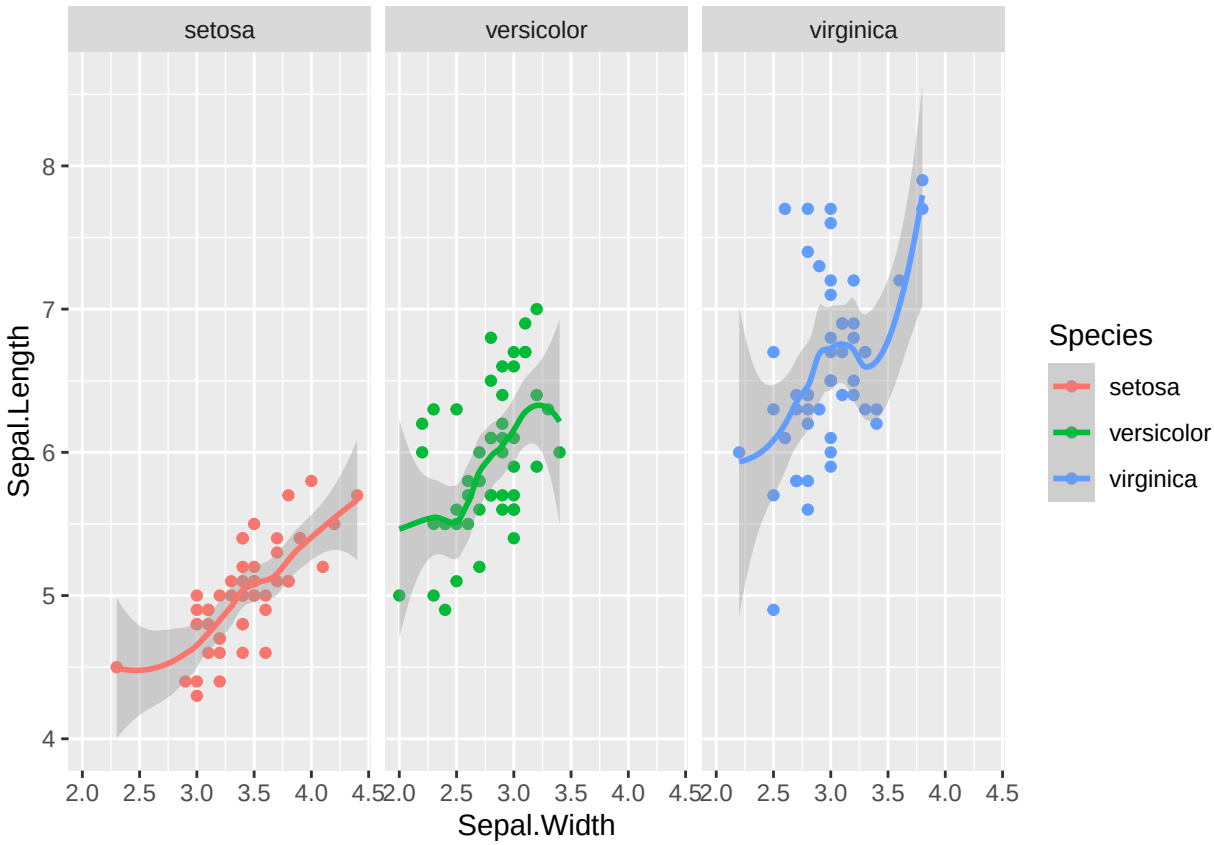
```
## `geom_smooth()` using formula 'y ~ x'
```



Należy jednak pamiętać, że domyślnie dopasowywane jest nie model liniowy, ale ten: https://en.wikipedia.org/wiki/Local_regression Linia jest więc pogięta, podobnie przedziały ufności.

```
ggplot(data = iris, aes(x = Sepal.Width, y = Sepal.Length, col = Species)) +
  geom_point() +
  geom_smooth() +
  facet_grid(.~Species)
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



Oczywiście nic nie stoi na przeszkodzie dodawać znane już nam geomy takie jak `geom_rug`.

```
ggplot(data = iris, aes(x = Sepal.Width, y = Sepal.Length, col = Species)) +
  geom_point() +
  geom_smooth() +
  geom_rug() +
  facet_grid(.~Species)
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```

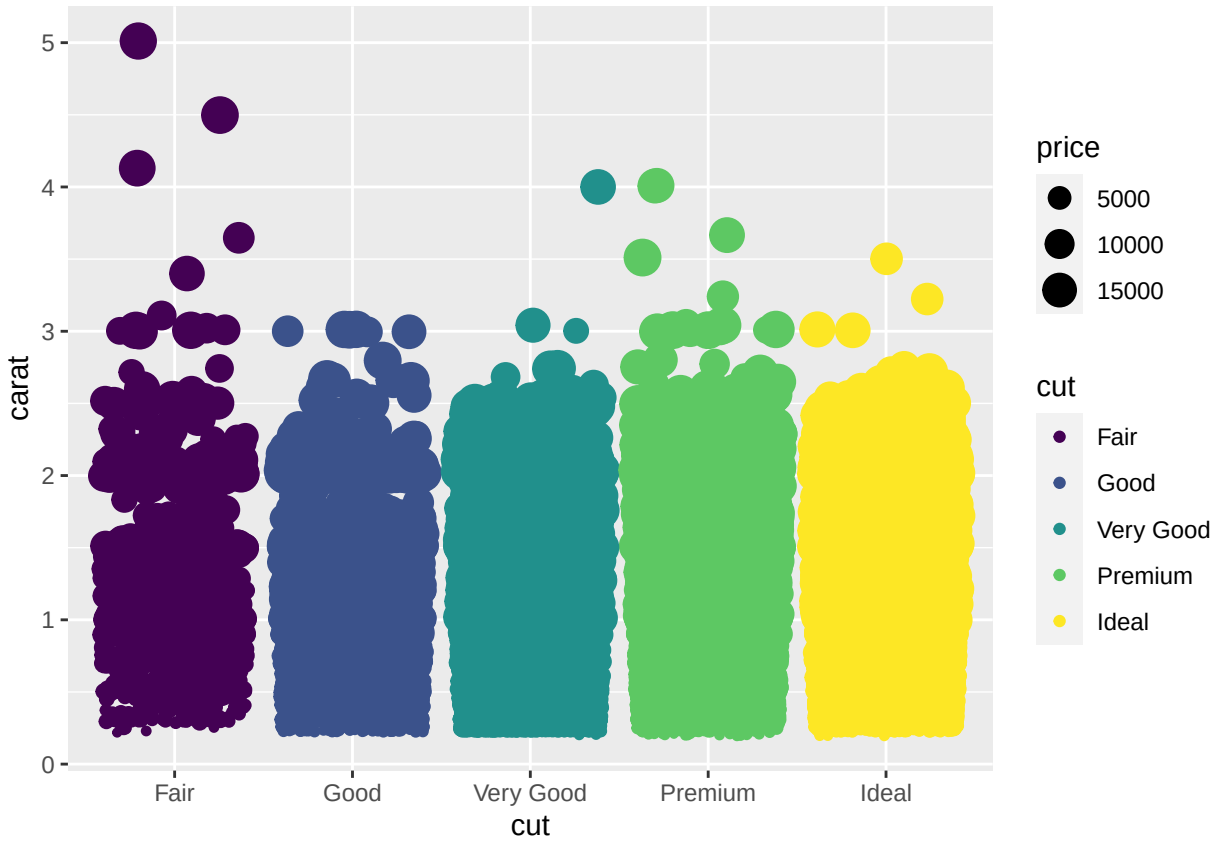
Zakończenie - bardziej skomplikowany przykład.

W kolejnym przykładzie tworzymy bardziej skomplikowany wykres dotyczący rozmaitych parametrów diamentów. Dane znajdują się w zbiorze `diamonds`, który wbudowany jest w R.

```
head(diamonds)
```

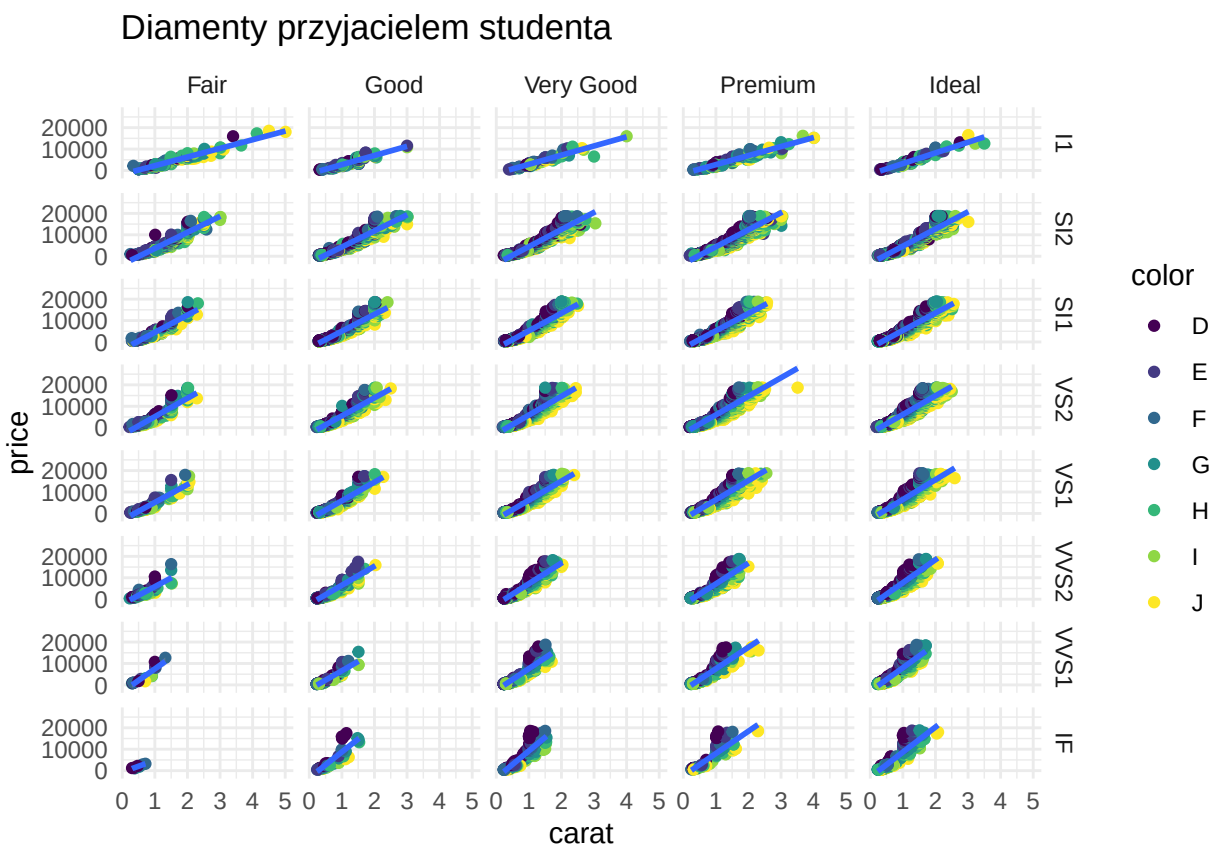
```
## # A tibble: 6 x 10
##   carat cut       color clarity depth table price     x     y     z
##   <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
## 1  0.23 Ideal     E     SI2      61.5    55   326  3.95  3.98  2.43
## 2  0.21 Premium   E     SI1      59.8    61   326  3.89  3.84  2.31
## 3  0.23 Good      E     VS1      56.9    65   327  4.05  4.07  2.31
## 4  0.29 Premium   I     VS2      62.4    58   334  4.2   4.23  2.63
## 5  0.31 Good      J     SI2      63.3    58   335  4.34  4.35  2.75
## 6  0.24 Very Good J     VVS2     62.8    57   336  3.94  3.96  2.48
```

```
ggplot(data = diamonds, aes(y=carat, x = cut, size = price, color = cut)) +
  geom_jitter()
```




```
ggplot(data = diamonds, aes(x=carat, y = price)) +  # podstawowe mapowanie
  geom_point(aes(color = color)) +  # mapowanie dla punktów: kolor diamentów -> kolor punktów
  facet_grid(clarity~cut) + # dzielimy na panele ze względu na przejrzystość i szlif
  stat_smooth(method = 'lm') + # dopasowujemy do danych linię regresji
  ggtitle('Diamenty przyjacielem studenta') +
  theme_minimal()
```

```
## `geom_smooth()` using formula 'y ~ x'
```



Uwagi:

1. Proszę dobierać rodzaj wykresu do celu - inny będzie w specjalistycznym czasopiśmie, inny na prezentacji, inny na plakacie, inny w artykule prasowym, inny dla siebie do pracy.
2. Nie wszystkie mapowania są sobie równe - kwestie poznawcze.
3. Nie należy przesadzać, należy dobrze zaplanować co i dlaczego chcemy pokazać.
4. Ale nie warto bać się eksperymentowania.

Przydatne linki:

1. Plotly + `ggplot2` = wygryw: https://plot.ly/ggplot2/
2. Dokumentacja https://ggplot2.tidyverse.org/
3. Dodatki: https://exts.ggplot2.tidyverse.org/
4. Galeria grafik wg koncepcji Tufte'a - niektóre z wykorzystaniem `ggplot2`: http://motioninsocial.com/tufte/