Modeling mass vs. length cubed

```
perch["length_cm_cubed"] = perch["length_cm"] ** 3

mdl_perch = ols("mass_g ~ length_cm_cubed", data=perch).fit()
mdl_perch.params
```

```
Intercept         -0.117478
length_cm_cubed    0.016796
dtype: float64
```

Coefficient of Determination

- Sometimes called "r-squared" or "R-squared".
- The proportion of the variance in the response variable that is predictable from the explanatory variable
  - 1 means a perfect
  - 0 means the worst possible

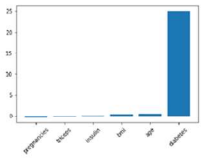
Lasso regression in scikit-learn

```
from sklearn.linear_model import Lasso

diabetes_df = pd.read_csv('diabetes.csv', index_col = 0)
diabetes_df = diabetes_df[diabetes_df['bmi'] != 0]
diabetes_df = diabetes_df[diabetes_df['glucose'] != 0]
X = diabetes_df.drop('glucose', axis=1).values
y = diabetes_df['glucose'].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
scores = []
for alpha in [0.01,1.0,10.0, 20.0, 50.0]:
    lasso = Lasso(alpha=alpha)
    lasso.fit(X_train, y_train)
    lasso_pred = lasso.predict(X_test)
    scores.append(lasso.score(X_test, y_test))
print(scores)
```

Lasso regression for feature selection

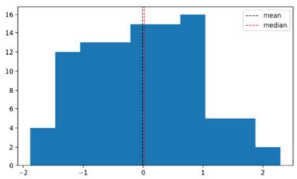
- Lasso can select important features of a dataset
- Shrinks the coefficients of less important features to zero
- Features not shrunk to zero are selected by lasso

```
diabetes_df = pd.read_csv('diabetes.csv', index_col = 0)
diabetes_df = diabetes_df[diabetes_df['bmi'] != 0]
diabetes_df = diabetes_df[diabetes_df['glucose'] != 0]
X = diabetes_df.drop('glucose', axis=1).values
y = diabetes_df['glucose'].values
```



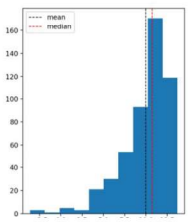

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

names = diabetes_df.drop('glucose', axis=1).columns
lasso = Lasso(alpha=0.1)
lasso_coef = lasso.fit(X,y).coef_
plt.bar(names, lasso_coef)
plt.xticks(rotation=45)
```

Which measure (mean vs. median) to use?

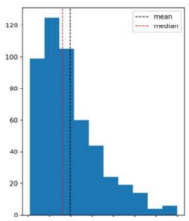


When data is skewed, the mean and median are different. The mean is pulled in the direction of the skew, so it's lower than the median on the left-skewed data, and higher than the median on the right-skewed data. Because the mean is pulled around by the extreme values, it's better to use the median since it's less affected by outliers.

Left-skewed



Right-skewed



Measures of spread

- What is spread?
  - Spread is just what it sounds like - it describes how spread apart or close together the data points are. Just like measures of center, there are a few different measures of spread.
- Variance - measures the average distance from each data point to the data's mean.

Standard deviation

- The standard deviation is another measure of spread, calculated by taking the square root of the variance. It can be calculated using np.std. Just like np.var, we need to set ddof to 1. The nice thing about standard deviation is that the units are usually easier to understand since they're not squared. It's easier to wrap your head around 4 and a half hours than 19-point-8 hours squared.

Quantiles

- Quantiles, also called percentiles, split up the data into some number of equal parts.
- Here, we call np.quantile, passing in the column of interest, followed by point-5. This gives us 10.1 hours, so 50% of mammals in the dataset sleep less than 10.1 hours a day, and the other 50% sleep more than 10.1 hours, so this is exactly the same as the median.

Quartiles

- We can also pass in a list of numbers to get multiple quantiles at once. Here, we split the data into 4 equal parts. These are also called quartiles. This means that 25% of the data is between 1.9 and 7.85, another 25% is between 7.85 and 10.10, and so on.

```
# Calculate total co2_emission per country: emissions_by_country
emissions_by_country = food.groupby('country')['co2_emission'].sum()
# Compute the first and third quantiles and IQR of emissions_by_country
q1 = np.quantile(emissions_by_country, 0.25)
q3 = np.quantile(emissions_by_country, 0.75)
iqr = q3 - q1
# Calculate the lower and upper cutoffs for outliers
lower = q1 - 1.5 * iqr
upper = q3 + 1.5 * iqr
# Subset emissions_by_country to find outliers
outliers = emissions_by_country[(emissions_by_country < lower) | (emissions_by_country > upper)]
print(outliers)
```

```
country
Argentina    2172.4
Name: co2_emission, dtype: float64
```

Outliers

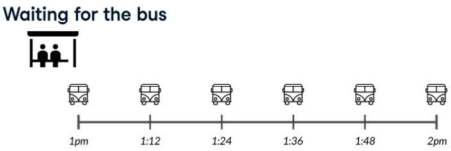
Outlier: data point that is substantially different from the others

How do we know what a substantial difference is? A data point is an outlier if:

- $\text{data} < Q1 - 1.5 \times IQR$ or
- $\text{data} > Q3 + 1.5 \times IQR$

Continuous distributions

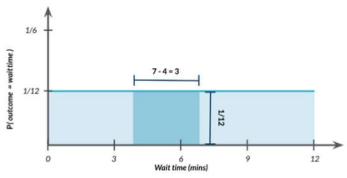
- We can use discrete distributions to model situations that involve discrete or countable variables, but how can we model continuous variables?
- Let's start with an example. The city bus arrives once every twelve minutes, so if you show up at a random time, you could wait anywhere from 0 minutes if you just arrive as the bus pulls in, up to 12 minutes if you arrive just as the bus leaves.



Probability still = area

- Now that we have our distribution, let's figure out what the probability is that we'll wait between 4 and 7 minutes. Just like with discrete distributions, we can take the area from 4 to 7 to calculate probability.
- The width of this rectangle is 7 minus 4 which is 3. The height is one twelfth.
- Multiplying those together to get area, we get 3/12 or 25%.

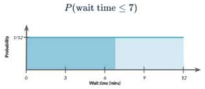
$$P(4 \leq \text{wait time} \leq 7) = 3 \times 1/12 = 3/12$$



Uniform distribution in Python

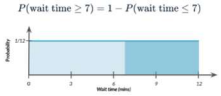
- Let's use the uniform distribution in Python to calculate the probability of waiting 7 minutes or less. We need to import uniform from scipy.stats. We can call uniform.cdf and pass it 7, followed by the lower and upper limits, which in our case is 0 and 12. The probability of waiting less than 7 minutes is about 58%.
- If we want the probability of waiting more than 7 minutes, we need to take 1 minus the probability of waiting less than 7 minutes.

$P(\text{wait time} \leq 7)$



```
.stats import uniform
f(7, 0, 12)
```

$P(\text{wait time} \geq 7) = 1 - P(\text{wait time} \leq 7)$



```
from scipy.stats import uniform
1 - uniform.cdf(7, 0, 12)
```

```
0.4166667
```