Practice – Data back-ups

To model how long Amir will wait for a back-up using a continuous uniform distribution, save his lowest possible wait time as min_time and his longest possible wait time as max_time. Remember that back-ups happen every 30 minutes.

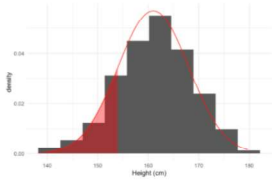
Import uniform from scipy.stats and calculate the probability that Amir has to wait less than 5 minutes, and store in a variable called prob_less_than_5.

Calculate the probability that Amir has to wait more than 5 minutes, and store in a variable called prob_greater_than_5.

Calculate the probability that Amir has to wait between 10 and 20 minutes, and store in a variable called prob_between_10_and_20.

```
min_time = 0
max_time = 30

# Import uniform from scipy.stats
from scipy.stats import uniform

# Calculate probability of waiting more than 5 mins
prob_greater_than_5 = 1-uniform.cdf(5,0,30)
print(prob_greater_than_5)

# Calculate probability of waiting 10-20 mins
prob_between_10_and_20 =
uniform.cdf(20,0,30)-uniform.cdf(10,0,30)
print(prob_between_10_and_20)
```

What percent of women are shorter than 154 cm?

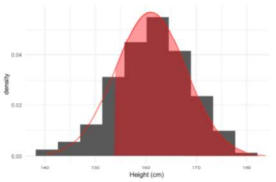


```
from scipy.stats import norm
norm.cdf(154, 161, 7)
```

```
0.158655
```

16% of women in the survey are shorter than 154 cm

What percent of women are taller than 154 cm?



```
from scipy.stats import norm
1 - norm.cdf(154, 161, 7)
```

```
0.841345
```

Simulating wait times

- To give Amir a better idea of how long he'll have to wait, you'll simulate Amir waiting 1000 times and create a histogram to show him what he should expect. Recall from the last exercise that his minimum wait time is 0 minutes and his maximum wait time is 30 minutes.
- Set the random seed to 334.
- Generate 1000 wait times from the continuous uniform distribution that models Amir's wait time. Save this as wait_times.
- Create a histogram of the simulated wait times and show the plot.

```
# Set random seed to 334
np.random.seed(334)

# Import uniform
from scipy.stats import uniform

# Generate 1000 wait times between 0 and 30 mins
wait_times = uniform.rvs(0, 30, size=1000)

# Create a histogram of simulated times and show plot
plt.hist(wait_times)
plt.show()
```

The central limit theorem

Rolling the dice 5 times

```
die = pd.Series([1, 2, 3, 4, 5, 6])
# Roll 5 times
samp_5 = die.sample(5, replace=True)
print(samp_5)
```

```
array([3, 1, 4, 1, 1])
```

```
np.mean(samp_5)
```

```
2.0
```

```
# Roll 5 times and take mean
samp_5 = die.sample(5, replace=True)
np.mean(samp_5)
```

```
4.4
```

```
samp_5 = die.sample(5, replace=True)
np.mean(samp_5)
```

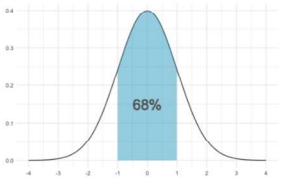
```
3.8
```

This phenomenon is known as the central limit theorem, which states that a sampling distribution will approach a normal distribution as the number of trials increases. In our example, the sampling distribution became closer to the normal distribution as we took more and more sample means. It's important to note that the central limit theorem only applies when samples are taken randomly and are independent, for example, randomly picking sales deals with replacement.

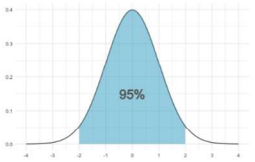
Normal distribution

- It's one of the most important probability distributions you'll learn about since a countless number of statistical methods rely on it, and it applies to more real-world situations than the distributions we've covered so far.
- The normal distribution looks like a "bell curve".
  - First, it's symmetrical, so the left side is a mirror image of the right.
  - Second, just like any continuous distribution, the area beneath the curve is 1.
  - The probability never hits 0, even if it looks like it does at the tail ends. Only 0-point-006% of its area is contained beyond the edges of this graph.

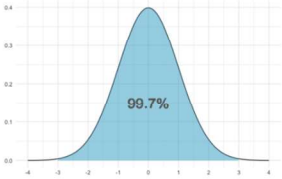
68% falls within 1 standard deviation



95% falls within 2 standard deviations

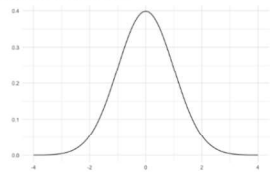


99.7% falls within 3 standard deviations

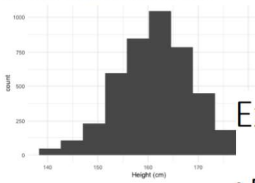


This is sometimes called the 68-95-99-point-7 rule.

Normal distribution



Women's heights from NHANES



Mean: 161 cm *Standard deviation: 7 cm*

There's lots of real-world data shaped like the normal distribution. For example, here is a histogram of the heights of women that participated in the National Health and Nutrition Examination Survey. The mean height is around 161 centimeters, and the standard deviation is about 7 centimeters.

Python packages for regression

- statsmodels
  - Optimized for insight
- scikit-learn
  - Optimized for prediction

Linear regression and logistic regression

Linear regression

- The response variable is numeric.

Logistic regression

- The response variable is logical. That is, it takes True or False values.

Call predict()

```
print(mdl_mass_vs_length.predict(explanatory_data))
```

Predicting inside a DataFrame

```
explanatory_data = pd.DataFrame(
  {"length_cm": np.arange(20, 41)})

prediction_data = explanatory_data.assign(
  mass_g=mdl_mass_vs_length.predict(explanatory_data)

print(prediction_data)
```

```
0
1
2
3
4
..
16
```

Showing predictions

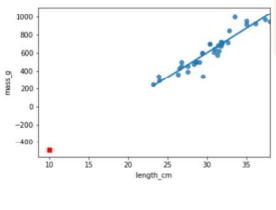
```
import matplotlib.pyplot as plt
import seaborn as sns
fig = plt.figure()
sns.regplot(x="length_cm",
            y="mass_g",
            ci=None,
            data=bream,)
sns.scatterplot(x="length_cm",
                y="mass_g",
                data=prediction_data,
                color="red",
                marker="s")
plt.show()
```

Extrapolating

- Extrapolating means making predictions outside the range of observed data.

```
little_bream = pd.DataFrame({"length_cm": [10]})

pred_little_bream = little_bream.assign(
    mass_g=mdl_mass_vs_length.predict(little_bream))

print(pred_little_bream)
```

```
   length_cm      mass_g
0         10 -489.847756
```



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

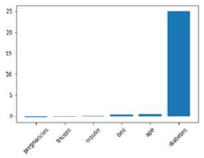
Lasso regression in scikit-learn

```
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

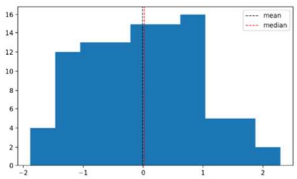
- Lasso can select important features of a dataset
- Shrinks the coefficients of less important features to zero
- Features not shrunk to zero are selected by lasso

```
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
```



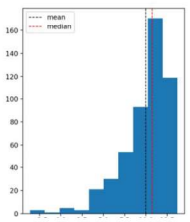

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

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?

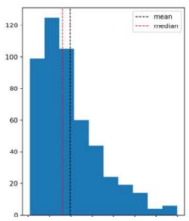


When data is skewed, the mean and median are different. The mean is pulled in the direction of the skew, so it's lower than the median on the left-skewed data, and higher than the median on the right-skewed data. Because the mean is pulled around by the extreme values, it's better to use the median since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

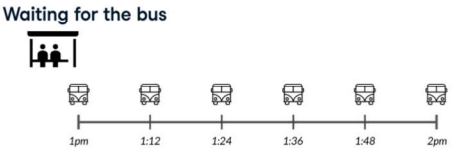
Outlier: data point that is substantially different from the others

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

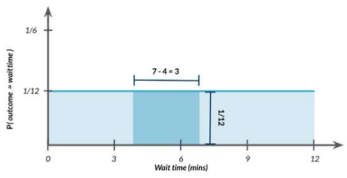
- We can use discrete distributions to model situations that involve discrete or countable variables, but how can we model continuous variables?
- Let's start with an example. The city bus arrives once every twelve minutes, so if you show up at a random time, you could wait anywhere from 0 minutes if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

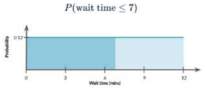
$$P(4 \leq \text{wait time} \leq 7) = 3 \times 1/12 = 3/12$$



Uniform distribution in Python

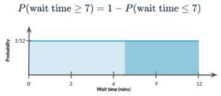
- Let's use the uniform distribution in Python to calculate the probability of waiting 7 minutes or less. We need to import uniform from scipy.stats. We can call uniform.cdf and pass it 7, followed by the lower and upper limits, which in our case is 0 and 12. The probability of waiting less than 7 minutes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