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.

```
min_time = 0
max_time = 30
```

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.

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

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

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

```
# 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)
```

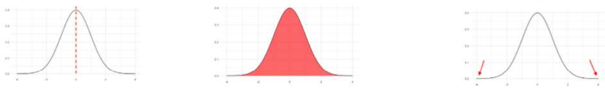
Simulating wait times

- 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.

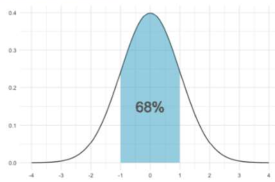
```
# Set random seed to 334
np.random.seed(334)

# 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()
```

Normal distribution

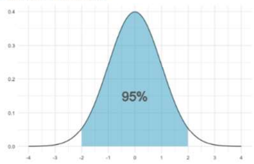
- It's one of the most important probability distributions you'll learn about since a countless number of statistical methods rely on it, and it applies to more real-world situations than the distributions we've covered so far.
- The normal distribution looks like a "bell curve".
  - First, it's symmetrical, 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.



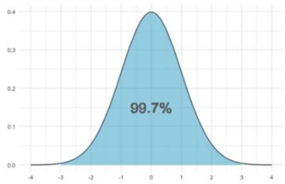
68% falls within 1 standard deviation



95% falls within 2 standard deviations

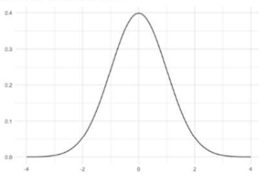


99.7% falls within 3 standard deviations

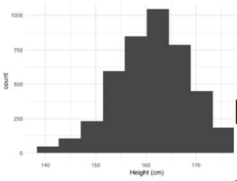


This is sometimes called the 68-95-99-point-7 rule.

Normal distribution



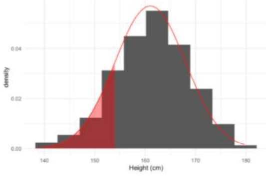
Women's heights from NHANES



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.

What percent of women are shorter than 154 cm?

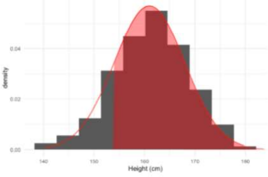



```
from scipy.stats import norm
norm.cdf(154, 161, 7)
```

```
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
```

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.

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)}

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

 nt(prediction_data)
```

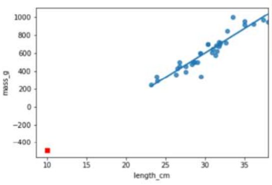
```
0
1
2
3
4
..
16
```

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
```



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()
```