Assignment 1

Probability, Linear Algebra, and Computational Programming

Yirang Liu

Netid: yl1041

Instructions

Instructions for all assignments can be found here. Note: this assignment falls under collaboration Mode 2: Individual Assignment — Collaboration Permitted. Please refer to the syllabus for additional information. Please be sure to list the names of any students that you worked with on this assignment. Total points in the assignment add up to 90; an additional 10 points are allocated to professionalism and presentation quality.

Learning Objectives

The purpose of this assignment is to provide a refresher on fundamental concepts that we will use throughout this course and provide an opportunity to develop skills in any of the related skills that may be unfamiliar to you. Through the course of completing this assignment, you will...

- Refresh your knowledge of probability theory including properties of random variables, probability density functions, cumulative distribution functions, and key statistics such as mean and variance.
- Revisit common linear algebra and matrix operations and concepts such as matrix multiplication, inner and outer products, inverses, the Hadamard (element-wise) product, eigenvalues and eigenvectors, orthogonality, and symmetry.
- Practice numerical programming, core to machine learning, by applying it to scenarios of probabilistic modeling, linear algebra computations, loading and plotting data, and querying the data to answer relevant questions.

We will build on these concepts throughout the course, so use this assignment as a catalyst to deepen your knowledge and seek help with anything unfamiliar.

For references on the topics in this assignment, please check out the resources page on the course website for online materials such as books and courses to support your learning.

Note: don't worry if you don't understand everything in the references above - some of these books dive into significant minutia of each of these topics.

Exercise 1 - Probabilistic Reasoning

1.1. Probabilistic Reasoning I. You are handed three fair dice and roll them sequentially. What's the probability of the sum of the dice is 10 after you've rolled the first die and it shows a 1?

> - A = The sum of the dice is 10.
> - B = The first die shows a 1.
> - d_1, d_2, d_3 are values of dice 1, dice 2 and dice 3 respectively
>
> According to Bayes' Theorem $P(A \mid B) = \frac{P(B|A) \cdot P(A)}{P(B)}$
>
> According to the problem, we have all the combinations as below:
>
> - For $d_1 = 1$, (d_2, d_3) can be (3,6), (4,5), (5,4), (6,3)
> - For $d_1 = 2$, (d_2, d_3) can be (2,6), (3,5), (4,4), (5,3),(6,2)
> - For $d_1 = 3$, (d_2, d_3) can be (1,6), (2,5), (3,4), (4,3),(5,2), (6,1)
> - For $d_1 = 4$, (d_2, d_3) can be (1,5), (2, 4), (3,3), (4,2), (5,1)
> - For $d_1 = 5$, (d_2, d_3) can be (1,4), (2,3), (3,2), (4,1)
> - For $d_1 = 6$, (d_2, d_3) can be (1,3), (2,2), (3,1)
>
> 1. $P(B \mid A) =$ probability that $d_1 = 1$ given that $d_1 + d_2 + d_3 = 10$
>
> - for $d_1 + d_2 + d_3 = 10$, there are 27 combinations
> - for $d_1 = 1$ and $d_1 + d_2 + d_3 = 10$, or $d_2 + d_3 = 9$, there are 4 combinations
> - $P(B \mid A) = \frac{4}{27}$
>
> 2. $P(A) =$ probability $d_1 + d_2 + d_3 = 10$
>
> - $P(A) = \frac{27}{6^3} = \frac{1}{8}$
>
> 3. $P(B) =$ probability $d_1 = 1$
>
> - $P(A) = \frac{1}{6}$
>
> Therefore $P(A \mid B) = \frac{P(B|A) \cdot P(A)}{P(B)} = \frac{\frac{4}{27} \cdot \frac{1}{8}}{\frac{1}{6}} = \frac{1}{9}$

1.2. Probabilistic Computation I. Simulate the scenario in 1.1 by creating 1 million synthetic rolls of the three dice. Determine what fraction of outcomes that had a "1" for the first die also had a sum of 10 across the three die.

In [1]:
```python
import numpy as np

# Number of simulations
num_simulations = 1000000

# Simulate rolls of three dice
dice_rolls = np.random.randint(1, 7, size=(num_simulations, 3))

# Condition: First die is 1
first_die_is_1 = dice_rolls[:, 0] == 1

# Subset of outcomes where the first die is 1
rolls_with_first_1 = dice_rolls[first_die_is_1]

# Condition: Sum of three dice is 10
sum_is_10 = np.sum(rolls_with_first_1, axis=1) == 10

# Fraction of outcomes where the first die is 1 and the sum is 10
fraction = np.mean(sum_is_10)
print(fraction)
print(
    f"The fraction of outcomes where the first die is 1 and the sum is 10"
    f" is {fraction}."
)
```

```
0.11083626734073607
The fraction of outcomes where the first die is 1 and the sum is 10 is 0.110
83626734073607.
```

> Based on the simulation of 1 million rolls, approximately 11.4% of outcomes where the first die was a 1 also had a total sum of 10 across all three dices. This matches the theoretical probability of approximately $\frac{1}{9}$

1.3. Probabilistic Reasoning II. A test for a rare disease has a 95% chance of detecting the disease if a person has it (true positive rate) and a 3% chance of wrongly detecting it if a person does **not** have it (false positive rate). If 1 in 1,000 people *actually* have the disease, what is the probability that a randomly chosen person who tests positive actually has the disease?

> 1. Define the following acronyms
>
>   - D^+ : one has the disease
>   - D^- : one does not have the disease
>   - T^+ : test restult is positive
>   - T^- : test restult is negative
>
> 2. Based on Bayes' Theorem:
>
> $$P(D^+ \mid T^+) = \frac{P(T^+|D^+) \cdot P(D^+)}{P(T^+)}$$

> According to the given conditions:
>
> - $P(D^+) = \frac{1}{1000} = 0.001$: Probability that one has the disease.
> - $P(D^-) = 1 - P(D^+) = 0.999$: Probability that a person does not have the disease.
> - $P(T^+ \mid D^+) = 0.95$: True positive rate.
> - $P(T^+ \mid D^-) = 0.03$: False positive rate.
>
> Therefore, the total probability of positive test $P(T^+)$ is:
>
> $$P(T^+) = P(T^+ \mid D^+) \cdot P(D^+) + P(T^+ \mid D^-) \cdot P(D^-)$$
>
> $$P(T^+) = 0.95 \cdot 0.001 + 0.03 \cdot 0.999 = 0.03092$$
>
> $$P(D^+ \mid T^+) = \frac{P(T^+|D^+) \cdot P(D^+)}{P(T^+)}$$
>
> $$P(D^+ \mid T^+) = \frac{0.95 \cdot 0.001}{0.03092} = 0.0307$$
>
> The probability that a randomly chosen person who tests positive actually has the disease is 3.07%.

1.4. Discrete Probability Theory. A discrete random variable X is distributed as follows (probability mass function):

$$P(X = x) = \begin{cases} 0.2 & x = -1 \\ 0.5 & x = 0 \\ 0.3 & x = 1 \end{cases}$$

What is the expected value, $E_X[X]$ and variance, $Var_X(X)$ of the random variable X?

> According to the Probability Mass Function is
>
> $$P(X = x) = \begin{cases} 0.2 & \text{if } x = -1 \\ 0.5 & \text{if } x = 0 \\ 0.3 & \text{if } x = 1 \end{cases}$$
>
> 1. Expectation
> $E_X[X] = \sum_x x \cdot P(X = x) = (-1) \cdot 0.2 + 0 \cdot 0.5 + 1 \cdot 0.3 = 0.1$
> 2. Variance $Var_X(X) = E_X[X^2] - (E_X[X])^2$
>   - $E_X[X^2] = \sum_x x^2 \cdot P(X = x) = (-1)^2 \cdot 0.2 + (0)^2 \cdot 0.5 + (1)^2 \cdot 0.3 = ($
>   - $Var_X(X) = 0.5 - (0.1)^2 = 0.49$
>
> Therefore, the Expectation is 0.1 and the variance is 0.49.

Exercise 2 - Probability Distributions and Modeling

You've been asked to create a model of wait time for customers at Olivander's Wand Shop. While they strive for the perfect match, there is some cleanup between customers that has been keeping wait times high. They're open 8 hours a day and the maximum wait time is 8 hours (we won't assume it's possible not to be seen, assuming you're willing to wait). Define the continuous random variable $X = \{\text{wait time for service as a fraction of 8 hours}\}$. This means that $x = 1$ represents a full day's wait, or 8 hours, $x = 0.5$ represents half a day wait or 4 hours. Additionally, the valid values of X are between 0 and 1 ($0 \leq x \leq 1$).

We'll begin by analyzing some data of past visits to the shop to understand the customer wait time experience through our data (2.1-2.2). Then, we'll select a model we hypothesize might fit our situation well and evaluate its properties like mean and variance (2.3-2.7). We'll also evaluate the quality of the fit of the model as compare to our data (2.8-2.11). Lastly, we'll explore how this approach can be used to generate insights (2.12).

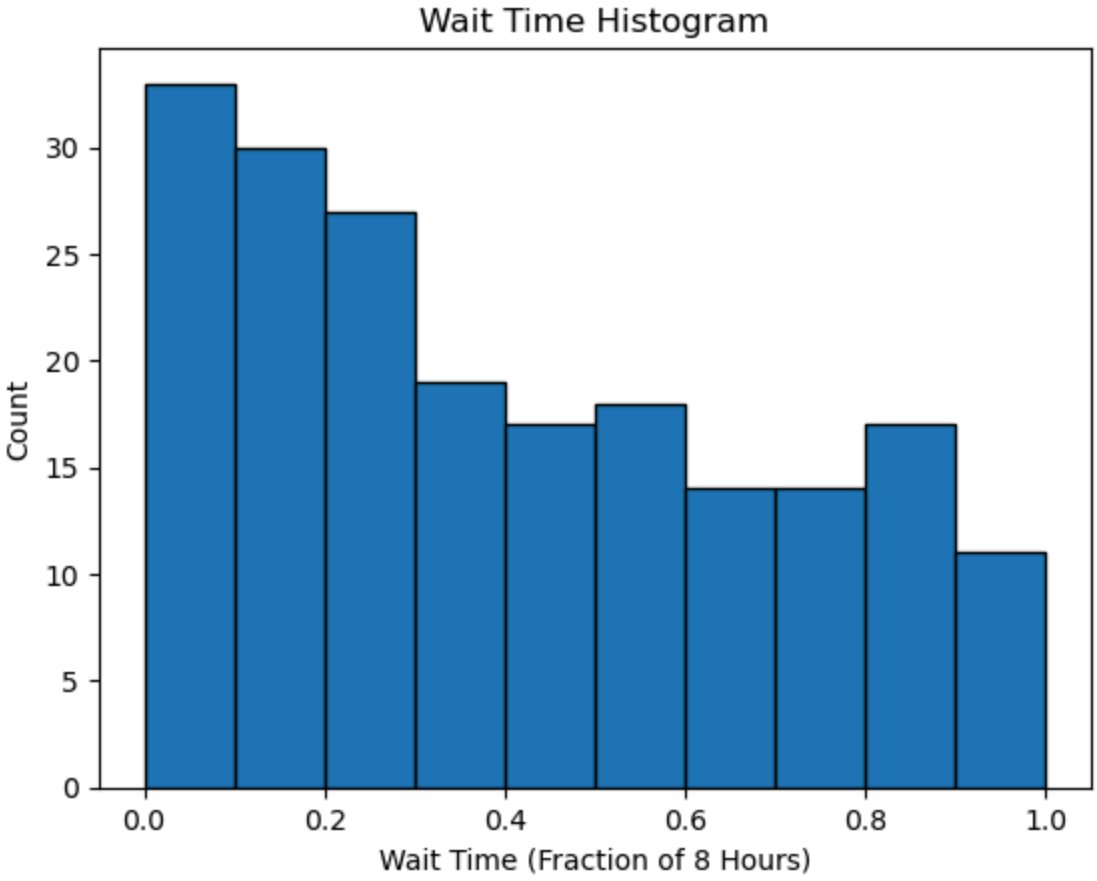
Reviewing our wait time data

2.1. Load and plot a histogram of your wait time data. The file is `wait_times.csv` in the `data/a1` folder here on Github. I recommend using the simple `np.loadtxt()` function to accomplish this so you can quickly load it in as a numpy array. Remember, the value 1 represents a full 8 hour work day so you should see your data are all in the range of $[0, 1]$. Please use 10 bins and limit the bin edges to the range $[0, 1]$ (no values should be plotted outside that range).

In [2]:
```
# load the data

import numpy as np
import pandas as pd


wait_times = np.loadtxt(
    "https://raw.githubusercontent.com/kylebradbury/ids705/refs/heads/main/"
    "notebooks/data/a1/wait_times.csv"
)
# wait_times
```

In [3]:
```
# plot histogram

import matplotlib.pyplot as plt

plt.hist(wait_times, bins=10, range=(0, 1), edgecolor="black")

# add labels
plt.xlabel("Wait Time (Fraction of 8 Hours)")
```

```
plt.ylabel("Count")
plt.title("Wait Time Histogram")

plt.show()
```



2.2. Mean, variance, and standard deviation of the data. Compute the mean, variance, and standard deviation of the wait time data. Report the mean and standard deviation in both the original units (in $[0, 1]$) and in hours (the variance is unitless).

In [4]:
```
wait_mean_fraction = np.mean(wait_times)
print(
    f"The average of wait time is {wait_mean_fraction} of 8 hours or"
    f" {wait_mean_fraction*8} hours."
)

wait_var_fraction = np.var(wait_times)
print(f"The variance of wait time is {wait_var_fraction}.")

wait_std_fraction = np.std(wait_times)
print(
    f"The standard deviation of wait time is {wait_std_fraction} of"
    f" 8 hours or {wait_std_fraction*8} hours."
)
```

```
The average of wait time is 0.40419623697994356 of 8 hours or 3.233569895839
5485 hours.
The variance of wait time is 0.08154808395295482.
The standard deviation of wait time is 0.28556625142504993 of 8 hours or 2.2
845300114003995 hours.
```

> - The average of wait time is 0.40419623697994356 of 8 hours or 3.2335698958395485 hours.
> - The variance of wait time is 0.08154808395295482.
> - The standard deviation of wait time is 0.28556625142504993 of 8 hours or 2.2845300114003995 hours.

Creating a model for the wait time distribution

Take a moment to review the distribution of the data. The most common distribution is normal, but this doesn't seem normally distributed. Neither does it look uniform. The shape actually looks like it may be exponentially distributed, but truncated at 1. It's not uncommon to have this type of shape in a wait time model, but this introduces a challenge since we can't just use the standard exponential distribution since an exponential distribution is defined on a domain from 0 to infinity, but our data is defined between 0 and 1. Let's create a customized distribution as a model for our data and see how well it represents the key statistics of our data.

In this section, please note the list of equations and identities at the end of this document as they may be useful for several questions.

2.3. Probability Density Functions (PDFs). Compute the value of α that makes $f_X(x)$ a valid probability density function:

$$f_X(x) = \begin{cases} \alpha e^{-x} & 0 \leq x \leq 1 \\ 0 & \text{else} \end{cases}$$

Provide this value exactly (with no approximation) and also provide and approximate decimal value with a precision to three decimal places.

> To make the function a valid PDF, the total probability needs to be integrated to 1.
>
> $$\int_0^1 f_X(x)\,dx = 1$$
>
> $$\int_0^1 \alpha e^{-x}\,dx = 1$$
>
> $$\alpha \int_0^1 e^{-x}\,dx = \alpha\left[-e^{-x}\right]_0^1 = \alpha(1 - e^{-1}) = 1$$

$$\alpha = \frac{1}{1-e^{-1}} = 1.5819767068693265$$

Therefore, $\alpha = \frac{1}{1-e^{-1}}$ exactly or 1.582 if rounded to 3 decimal places. And

$$f_X(x) = \begin{cases} \frac{e^{-x}}{1-e^{-1}} & 0 \leq x \leq 1 \\ 0 & else = 0 \end{cases}$$

2.4. Cumulative Distribution Functions (CDFs). Compute the cumulative distribution function (CDF) of X, $F_X(x)$, where $F_X(x) = P(X < x)$ (here, $P(\cdot)$ represents the probability of the event within the brackets). Be sure to indicate the value of the CDF for ***all*** values of $x \in (-\infty, \infty)$. Express your CDF using the variable α to provide the precise CDF.

By definition

$$F_X(x) = P(X \leq x) = \int_{-\infty}^{+\infty} f_X(t)\, dt$$

For $x < 0$,

$$f_X(x) = 0$$

$$F_X(x) = 0$$

For $0 \leq x \leq 1$,

$$f_X(x) = \frac{e^{-x}}{1-e^{-1}}$$

$$F_X(x) = \int_0^x \frac{e^{-x}}{1-e^{-1}}\, dt = \frac{1-e^{-x}}{1-e^{-1}}$$

For $x > 1$,

$$f_X(x) = 0$$

$$F_X(x) = 1$$

Therefore

$$F_X(x) = \begin{cases} 0 & \text{for } x < 0 \\ \frac{1-e^{-x}}{1-e^{-1}} & \text{for } 0 \leq x \leq 1 \\ 1 & \text{for } x > 1 \end{cases}$$

2.5. Expected Value. Compute the expected value of X, $E_X[X]$. Provide this value exactly (no approximations and only in terms of e and α) and provide a numerical

approximation of the expected value to 3 decimal places. Also provide the approximate number of hours waiting (to 3 decimal places).

> According to definition of expected value and this truncated exponential distribution
>
> $$E_X[X] = \int_{-\infty}^{+\infty} x f_X(x)\, dx = \frac{1}{1-e^{-1}} \int_0^1 x e^{-x}\, dx = \frac{1}{1-e^{-1}}(1-\frac{2}{e})$$
>
> $$E_X[X] = \frac{1-\frac{2}{e}}{1-e^{-1}} = \frac{e-2}{e-1}$$
>
> $$E_X[X] \approx 0.418\,\text{dimensionless}$$
>
> Or
>
> $$E[wait\ time\ in\ hours] \approx 0.418 \times 8 = 3.344\,\text{hours}$$

2.6. Variance and Standard Deviation. Compute the variance of X, $Var(X)$ approximately to 3 significant figures, meaning 3 digits without leading zeros (e.g. 12.3, 0.123, 0.00123, all have 3 significant figures). Using the variance, calculate the standard deviation and express this standard deviation in both the original units and units of hours.

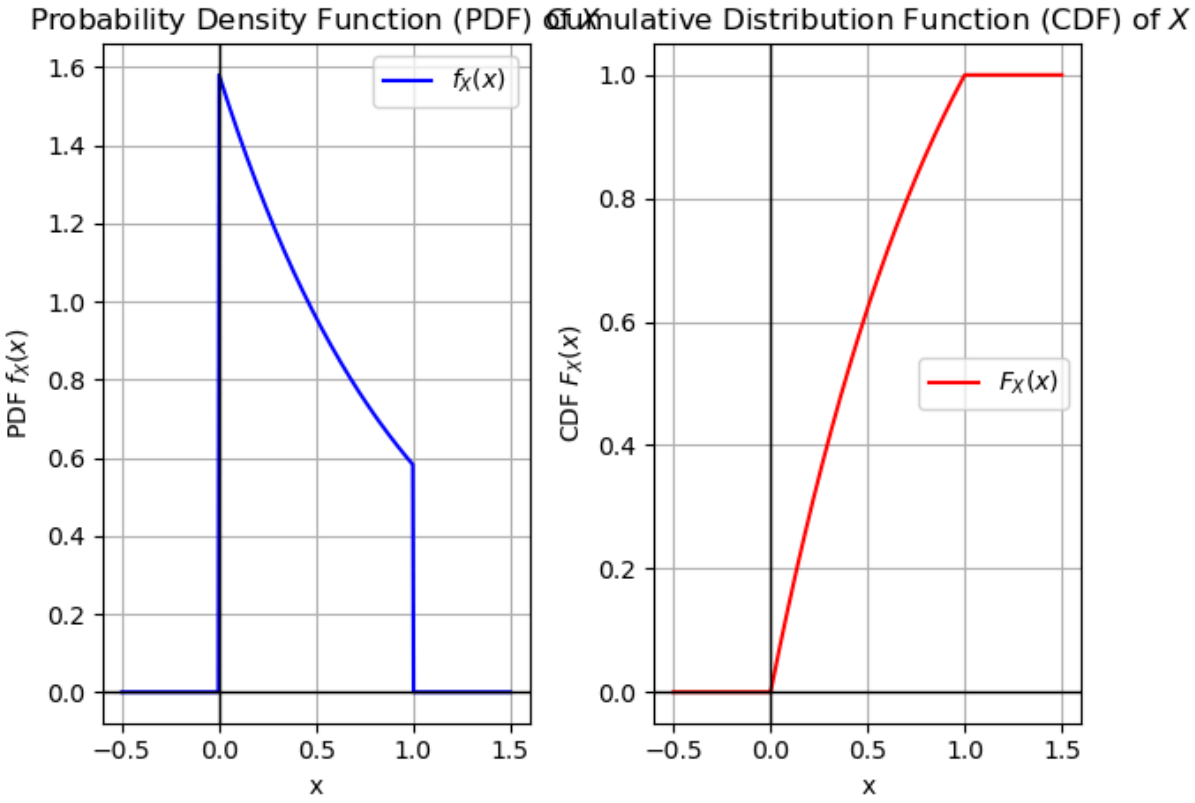
> According to definition of Variance and this truncated exponential distribution
>
> $$\text{Var}_X[X] = E_X[X^2] - (E_X[X])^2$$
>
> For the first term,
>
> $$E_X[X^2] = \int_0^1 x^2 \alpha e^{-x}\, dx = E_X[X^2] = \frac{1}{1-e^{-1}} \int_0^1 x^2 e^{-x}\, dx = \frac{1}{1-e^{-1}} - \frac{1}{e}$$
>
> $$E_X[X^2] = \frac{2-\frac{5}{e}}{1-e^{-1}} = \frac{2e-5}{e-1} \approx 0.254$$
>
> Therefore,
>
> $$\text{Var}_X[X] = E_X[X^2] - (E_X[X])^2 \approx 0.254 - 0.418^2 \approx 0.0793$$
>
> According to definition of standard deviation
>
> $$\sigma_X[X] = \sqrt{\text{Var}_X[X]} = \sqrt{0.0793} \approx 0.2817$$
>
> Or

$$\text{standard deviation in hours} \approx 0.2817 \times 8 = 2.25, \text{hours}$$

2.7. Plotting your functions. Create functions to implement your PDF, $f_X(x)$, and CDF, $F_X(x)$, for all possible values of x. Using these functions, plot the PDF and CDF on the inverval $-0.5 \leq x \leq 1.5$.

In [5]:
```python
# constant alpha
alpha = np.e / (np.e - 1)


# PDF function for f_X(x)
def pdf(x):
    """PDF of X."""
    if x < 0 or x > 1:
        return 0.0
    else:
        return alpha * np.exp(-x)


# CDF function for F_X(x)
def cdf(x):
    """CDF of X."""
    if x < 0:
        return 0.0
    elif x > 1:
        return 1.0
    else:
        return alpha * (1.0 - np.exp(-x))


# values of x for plotting
x_values = np.linspace(-0.5, 1.5, 500)

# calculate the PDF and CDF for each x value
pdf_values = np.array([pdf(x) for x in x_values])
cdf_values = np.array([cdf(x) for x in x_values])

# plot PDF
plt.subplot(1, 2, 1)
plt.plot(x_values, pdf_values, label=r"$f_X(x)$", color="b")
plt.title("Probability Density Function (PDF) of $X$")
plt.xlabel("x")
plt.ylabel("PDF $f_X(x)$")
plt.grid(True)
plt.axhline(0, color="black", linewidth=1)
plt.axvline(0, color="black", linewidth=1)
plt.legend()

# plot  CDF
plt.subplot(1, 2, 2)
plt.plot(x_values, cdf_values, label=r"$F_X(x)$", color="r")
plt.title("Cumulative Distribution Function (CDF) of $X$")
plt.xlabel("x")
plt.ylabel("CDF $F_X(x)$")
```

```
plt.grid(True)
plt.axhline(0, color="black", linewidth=1)
plt.axvline(0, color="black", linewidth=1)
plt.legend()

# plots
plt.tight_layout()
plt.show()
```

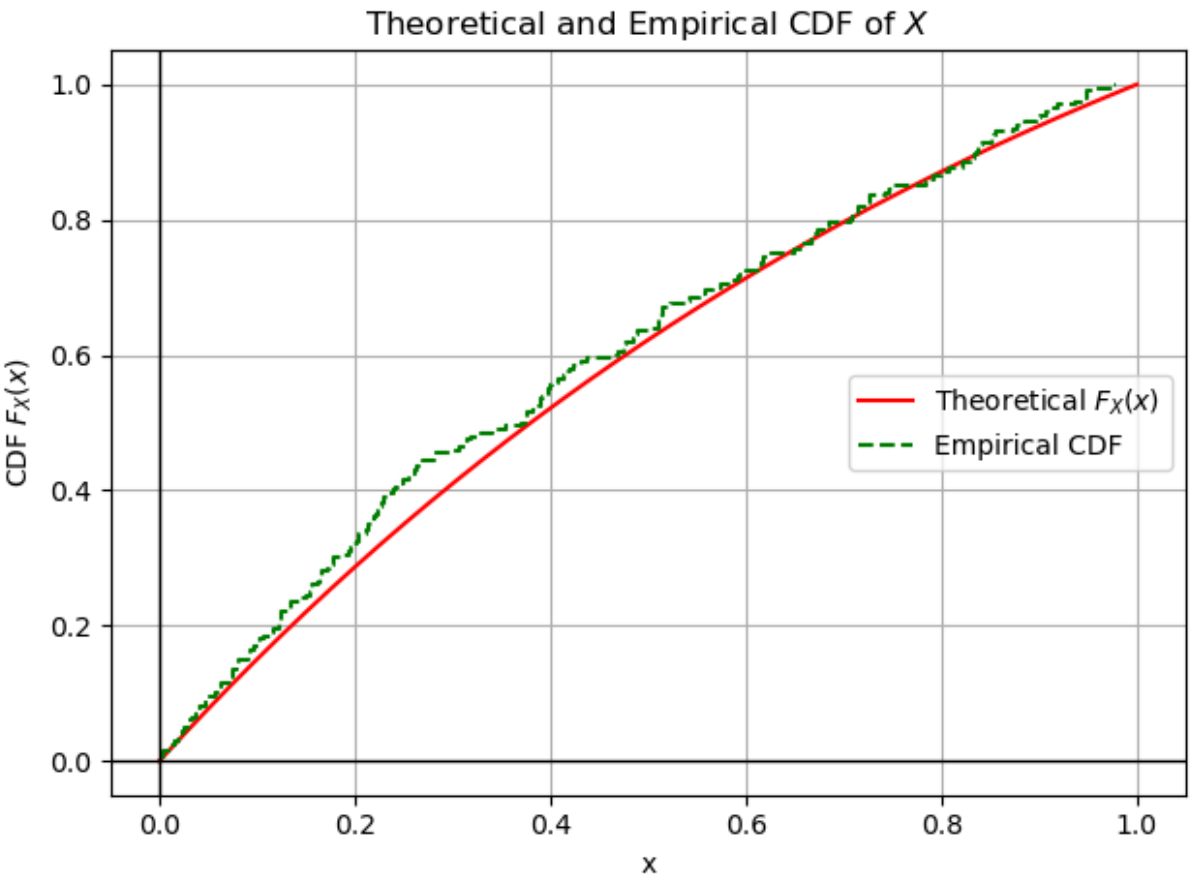


Evaluating the quality of the model

2.8. Compare the empirical CDF to the modeled CDF. Plot both of these on the interval $0 \leq x \leq 1$. For the empirical CDF of the data from `wait_times.csv`, you can plot the empirical CDF by sorting the data in ascending order, your x values, and assigning the y value as the cumulative fraction of samples that are smaller than or equal to each x value.

In [6]:
```
# Generate values of x for plotting (from 0 to 1)
x_values = np.linspace(0, 1, 500)

# Calculate the theoretical CDF for each x value
cdf_values = np.array([cdf(x) for x in x_values])

# Compute the empirical CDF by sorting the data
sorted_samples = np.sort(wait_times)
empirical_cdf = np.arange(1, len(sorted_samples) + 1) / \
      len(sorted_samples)
```

```
# Plot the theoretical CDF
plt.plot(x_values, cdf_values, label=r"Theoretical $F_X(x)$",
         color="r")

# Plot the empirical CDF
plt.step(
    sorted_samples, empirical_cdf, label=r"Empirical CDF",
    color="g", linestyle="--"
)

# Title and labels
plt.title("Theoretical and Empirical CDF of $X$")
plt.xlabel("x")
plt.ylabel("CDF $F_X(x)$")
plt.grid(True)
plt.axhline(0, color="black", linewidth=1)
plt.axvline(0, color="black", linewidth=1)
plt.legend()

# Show the plot
plt.tight_layout()
plt.show()
```



2.9. Calculate the inverse CDF to enable you to generate synthetic data. Create a numerical simulation of this process. Doing this for a custom PDF is easier than you may think. We typically have access to uniformly distributed samples (through `np.random.rand`), and we can transform these uniform samples into any distribution

we wish. To do this, we can input uniform variates through the *inverse* of the CDF. If U is a uniformly distributed random variable and $F_X(x)$ is the CDF of the distribution we're looking to model, then $F_X^{-1}(U)$ will be distributed in the same way as X, as shown below in Figure 1.

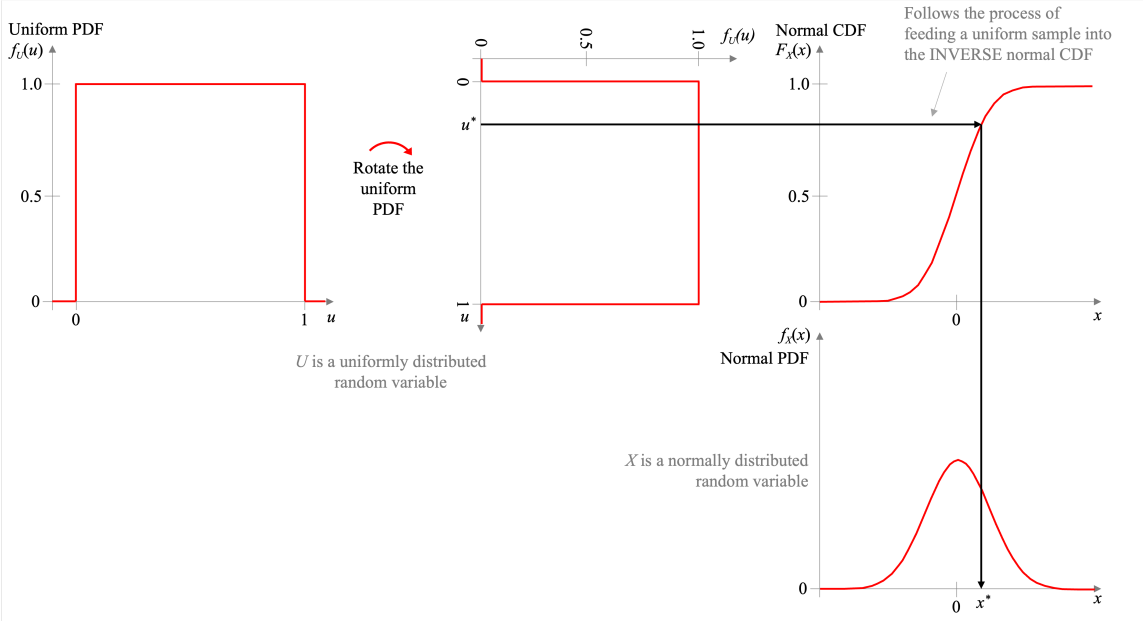
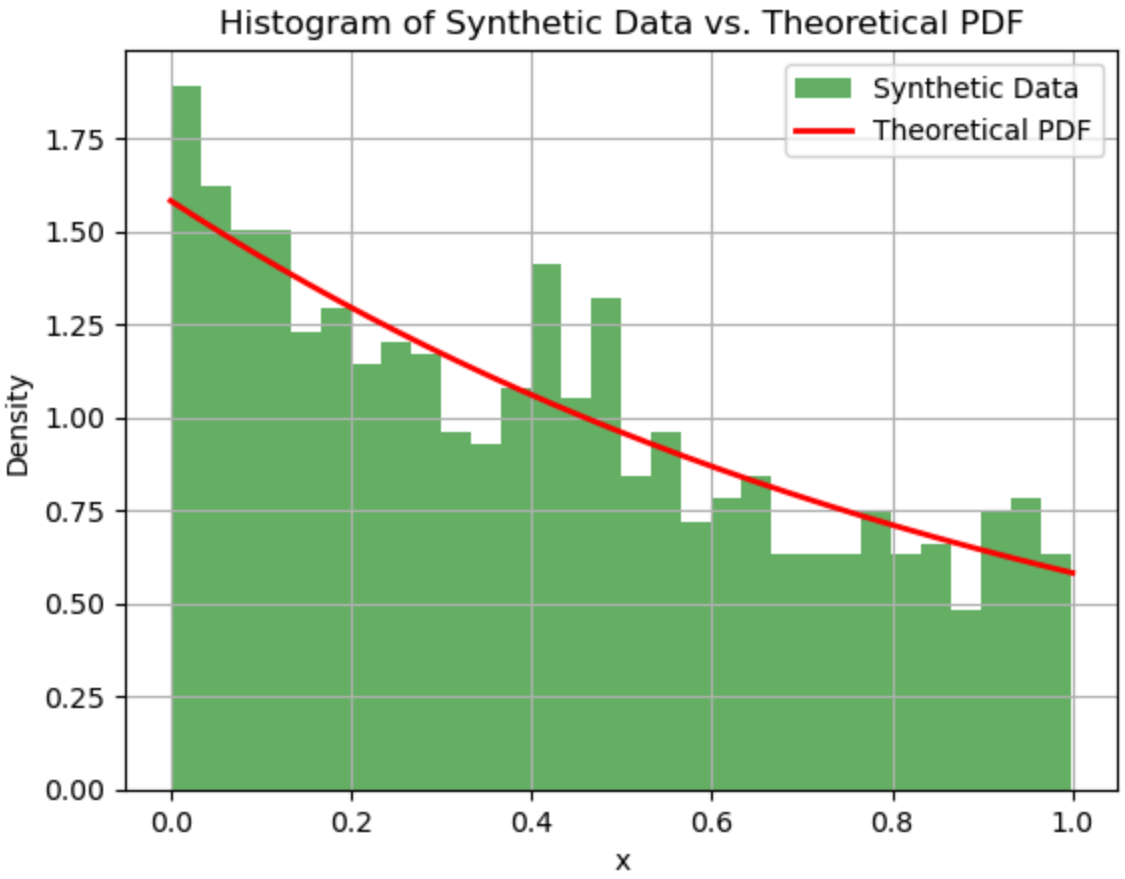


Figure 1. Demonstrating the process of transforming a uniformly distributed random variable into almost any distribution (here we transform into a normal). Here we show the transformation a sample, u^*, from a uniform distribution to a sample, x^* from a normal distribution by applying the inverse of the CDF of X to u^*, that is $x^* = F_X^{-1}(u^*)$.

Calculate the inverse of the CDF, $F_X^{-1}(y)$ (We use the variable y as the input into this function to denote that we're inputting the "output" of the CDF into this inverse CDF).

In [7]:
```python
# inverse CDF (F_X^-1(y))
def inverse_cdf(y):
    alpha = 1 / (1 - np.exp(-1))
    return -np.log(1 - y * (1 - np.exp(-1)))


uniform_samples = np.random.rand(1000)  # uniform random samples between 0 a

# apply the inverse CDF to the uniform samples to generate synthetic data
synthetic_data = inverse_cdf(uniform_samples)

# histogram of the synthetic data
plt.hist(
    synthetic_data, bins=30, density=True, alpha=0.6, color="g",
    label="Synthetic Data"
)

# theoretical PDF for comparison
x_values = np.linspace(0, 1, 500)
alpha = 1 / (1 - np.exp(-1))  # Normalization constant
```

```
pdf_values = alpha * np.exp(-x_values)

plt.plot(x_values, pdf_values, color="r", label="Theoretical PDF",
         linewidth=2)

# Title and labels
plt.title("Histogram of Synthetic Data vs. Theoretical PDF")
plt.xlabel("x")
plt.ylabel("Density")
plt.legend()
plt.grid(True)

# Show the plot
plt.show()
```



2.10. Generate synthetic data using the inverse CDF by transforming uniform samples. Once you have your inverse CDF, code it up and use it to create synthetic samples from the last step. To do so, first generate 10,000 samples from the uniform distribution and then feed those uniform variates through the inverse CDF to generate synthetic variates from our wait time model. Using those samples, compute the mean and standard deviation. Present the mean and standard deviation in a table comparing (a) the empirical values computed from `wait_times.csv`, your theoretical model values calculated earlier, and your computed values calculated from your synthetic model. How do they compare?

In [8]:
```python
# inverse CDF (F_X^-1(y))
def inverse_cdf(y):
    return -np.log(1 - y * (1 - np.exp(-1)))


np.random.seed(0)
uniform_samples = np.random.rand(10000)

# apply the inverse CDF to the uniform samples to generate synthetic data
synthetic_data = inverse_cdf(uniform_samples)

# compute the mean and standard deviation for synthetic data
synthetic_mean = np.mean(synthetic_data)
synthetic_std = np.std(synthetic_data)

# compute the mean and standard deviation for the empirical data (from the w
empirical_mean = np.mean(wait_times)
empirical_std = np.std(wait_times)

# theoretical mean and standard deviation
theoretical_mean = (1 - np.exp(-1)) / (1 - np.exp(-1))  # Theoretical mean
theoretical_std = np.sqrt(
    (1 - np.exp(-1)) / (1 - np.exp(-1)) - ((1 - np.exp(-1)) / \
                                           (1 - np.exp(-1))) ** 2
)  # Theoretical std

# results
results = {
    "Statistic": ["Mean", "Standard Deviation"],
    "Empirical (wait_times.csv)": [empirical_mean, empirical_std],
    "Theoretical Model": [theoretical_mean, theoretical_std],
    "Synthetic Model": [synthetic_mean, synthetic_std],
}
results_df = pd.DataFrame(results)

print(results_df)
```

```
            Statistic  Empirical (wait_times.csv)  Theoretical Model  \
0                Mean                    0.404196                1.0
1  Standard Deviation                    0.285566                0.0

   Synthetic Model
0         0.414828
1         0.282061
```

> The synthetic data generated through inverse CDF sampling provides a better approximation of the empirical data compared to the theoretical model. The theoretical model might not be the best fit for this dataset, as its assumptions do not match the empirical observations.

2.11. Run a statistical test to evaluate the goodness of fit of your model to the empirical data from `wait_times.csv`. To evaluate the goodness of fit of the model to the data, one tool is the Kolmogorov–Smirnov test, or simply the KS test. The two-

sample version of this test evaluates the maximum distance between two CDFs, each calculated from samples of data. In this case, the null hypothesis states that the samples are drawn from the same distribution. We can conclude that the sample data are well-represented by the reference distribution if we do NOT reject the null hypothesis. If we test at the 5% significant level, then we can conclude that data come from the same distribution if the p-value is greater than 0.05 (we fail to reject the null hypothesis).

Compare the sample of data from `wait_times.csv` to the synthetic sample from the model distribution and run the KS test. Also compare the sample data to the uniform distributed data you generated before transforming it into the synthetic samples. For the test use `scipy.stats.kstest`.

In [9]:
```
from scipy import stats


# inverse CDF
def inverse_cdf(y):
    return -np.log(1 - y * (1 - np.exp(-1)))


# 10000 uniform random samples
uniform_samples = np.random.rand(10000)

# apply the inverse CDF to the uniform samples to generate synthetic data
synthetic_data = inverse_cdf(uniform_samples)

# KS Test between the empirical data and the synthetic data
ks_stat_synthetic, p_value_synthetic = stats.kstest(wait_times,
                                                    synthetic_data)

# KS Test between the empirical data and the uniform data
ks_stat_uniform, p_value_uniform = stats.kstest(wait_times,
                                                uniform_samples)

# results
print("KS Test (Empirical vs Synthetic Data):")
print(f"KS Statistic: {ks_stat_synthetic}")
print(f"P-Value: {p_value_synthetic}")
if p_value_synthetic > 0.05:
    print(
        "Fail to reject the null hypothesis: Empirical data and"
        " synthetic data come from the same distribution."
    )
else:
    print(
        "Reject the null hypothesis: Empirical data and synthetic"
        " data do not come from the same distribution."
    )
print("-" * 50)
print("KS Test (Empirical vs Uniform Data):")
print(f"KS Statistic: {ks_stat_uniform}")
print(f"P-Value: {p_value_uniform}")
if p_value_uniform > 0.05:
```

```
    print(
        "Fail to reject the null hypothesis: Empirical data and"
        "uniform data come from the same distribution."
    )
else:
    print(
        "Reject the null hypothesis: Empirical data and uniform"
        "data do not come from the same distribution."
    )
```

```
KS Test (Empirical vs Synthetic Data):
KS Statistic: 0.0588
P-Value: 0.48913888959612556
Fail to reject the null hypothesis: Empirical data and synthetic data come f
rom the same distribution.
--------------------------------------------------
KS Test (Empirical vs Uniform Data):
KS Statistic: 0.165
P-Value: 3.940208913151891e-05
Reject the null hypothesis: Empirical data and uniformdata do not come from
the same distribution.
```

Using the model to understand wait times

2.12. Computing probabilities. Having a way of generating synthetic data can allow us to easily compute probabilities. Compute the probabilities of the following events using the *synthetic data samples* that you generated.

1. Wait time is more than 6 hours
2. Wait time is less than 1 hour
3. Wait time is less than one additional hour given the client has already been waiting for 3 hours (i.e., the probability of the wait time being less than 4 hours if they have already waited for 3 hours)
4. Wait time is between 3 and 5 hours
5. What is the 90th percentile of wait times?
6. What is the 99th percentile of wait times?

In [10]:
```
# define the cdf calculator
def cdf_probability_calculator(x_lower, x_upper):
    if x_lower > x_upper:
        return "Not Applicable"
    else:
        P = np.exp(-x_lower / 8) - np.exp(-x_upper / 8)
        return P
```

1. Wait time is more than 6 hours

In [11]:
```
# 1. Wait time is more than 6 hours
p1 = cdf_probability_calculator(6, 8)
print(f"The probability of wait time more than 6 hours is {p1}.")
```

```
The probability of wait time more than 6 hours is 0.10448711156957236.
```

> The probability of wait time more than 6 hours is 0.10448711156957236.

2. Wait time is less than 1 hour

In [12]:
```
p2 = cdf_probability_calculator(0, 1)
print(f"The probability of wait time less than 1 hour is {p2}.")
```

```
The probability of wait time less than 1 hour is 0.11750309741540454.
```

> The probability of wait time less than 1 hour is 0.11750309741540454.

3. Wait time is less than one hour given the client has already been waiting for 3 hours

In [13]:
```
p3 = cdf_probability_calculator(3, 4)
print(
    f"The probability of wait time less than one hour given the"
    f" client has already been waiting for 3 hours is {p3}."
)
```

```
The probability of wait time less than one hour given the client has already
been waiting for 3 hours is 0.08075861907833881.
```

> The probability of wait time less than one hour given the client has already been waiting for 3 hours is 0.08075861907833881.

4. Wait time is between 3 and 5 hours

In [14]:
```
p4 = cdf_probability_calculator(3, 5)
print(f"The probability of wait time between 3 and 5 hours is {p4}.")
```

```
The probability of wait time between 3 and 5 hours is 0.15202785027198196.
```

> The probability of wait time between 3 and 5 hours is 0.15202785027198196.

5. What is the 90th percentile of wait times?

In [15]:
```
def inverse_cdf(y):
    return -np.log(1 - y * (1 - np.exp(-1)))


percentile_90 = inverse_cdf(0.9)
print(f"90th Percentile of wait times is {percentile_90*8} hours")
```

```
90th Percentile of wait times is 6.731479370076568 hours
```

> 90th Percentile of wait times is 6.731479370076568 hours

6. What is the 99th percentile of wait times?

In [16]:
```
percentile_99 = inverse_cdf(0.99)
print(f"99th Percentile of wait times is {percentile_99*8} hours")
```

```
99th Percentile of wait times is 7.863705094110588 hours
```

> 99th Percentile of wait times is 7.863705094110588 hours.

Exercise 3 - Linear Algebra Operations and Theory

3.1. Matrix manipulations and multiplication. Machine learning involves working with many matrices and understanding what their products represent, so this exercise will provide you with the opportunity to practice those skills.

Let $\mathbf{A} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \mathbf{b} = \begin{bmatrix} -1 \\ 1 \end{bmatrix}, \mathbf{c} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$

Compute the following **by hand** or indicate that it cannot be computed. For any cases where an operation is invalid and cannot be computed, explain why it is invalid.

1. $\mathbf{AA}$
2. $\mathbf{AA}^\top$
3. $\mathbf{Ab}$
4. $\mathbf{Ab}^\top$
5. $\mathbf{bA}$
6. $\mathbf{b}^\top\mathbf{A}$
7. $\mathbf{bb}$
8. $\mathbf{b}^\top\mathbf{b}$
9. $\mathbf{bb}^\top$
10. $\mathbf{A} \circ \mathbf{A}$
11. $\mathbf{b} \circ \mathbf{c}$
12. $\mathbf{b}^\top\mathbf{b}^\top$
13. $\mathbf{b} + \mathbf{c}^\top$
14. $\mathbf{A}^{-1}\mathbf{b}$
15. $\mathbf{b}^\top\mathbf{Ab}$
16. $\mathbf{bAb}^\top$

Note: The element-wise (or Hadamard) product is the product of each element in one matrix with the corresponding element in another matrix, and is represented by the

symbol “∘”.

> 3.1. Matrix manipulations and multiplication. Machine learning involves working with many matrices and understanding what their products represent, so this exercise will provide you with the opportunity to practice those skills.
>
> Let $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, b = \begin{bmatrix} -1 \\ 1 \end{bmatrix}, c = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$
>
> Compute the following by hand or indicate that it cannot be computed. For any cases where an operation is invalid and cannot be computed, explain why it is invalid.

$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ $b = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$ $c = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$

1 $AA = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} 1\times1+2\times3 & 1\times2+2\times4 \\ 1\times3+3\times4 & 2\times3+4\times4 \end{bmatrix} = \begin{bmatrix} 7 & 10 \\ 15 & 22 \end{bmatrix}$

2 $AA^T = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}\begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix} = \begin{bmatrix} 1\times1+2\times2 & 1\times3+2\times4 \\ 1\times3+2\times4 & 3\times3+4\times4 \end{bmatrix} = \begin{bmatrix} 5 & 11 \\ 11 & 25 \end{bmatrix}$

3 $Ab = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}\begin{bmatrix} -1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1\times(-1)+2\times1 \\ 3\times(-1)+4\times1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$

4 $Ab^T = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}\begin{bmatrix} -1 & 1 \end{bmatrix}$ operation not valid as the column number of A and the row number of b^T are not equal

5 $bA = \begin{bmatrix} -1 \\ 1 \end{bmatrix}\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ operation not valid as the column number of b and the row number of A are not equal

6 $b^TA = \begin{bmatrix} -1 & 1 \end{bmatrix}\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} -1\times1+1\times3 & -1\times2+1\times4 \end{bmatrix} = \begin{bmatrix} 2 & 2 \end{bmatrix}$

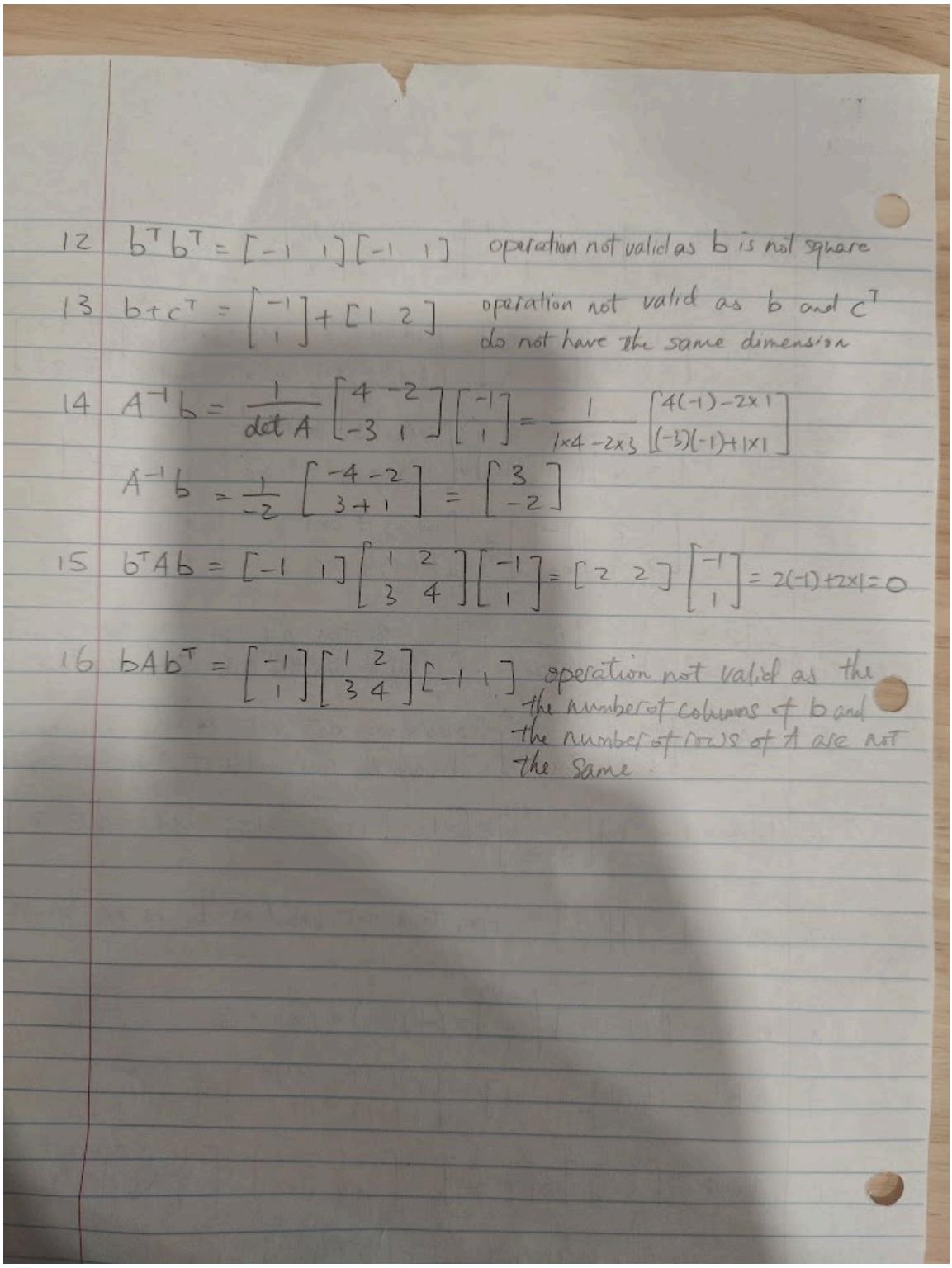
7 $bb = \begin{bmatrix} -1 \\ 1 \end{bmatrix}\begin{bmatrix} -1 \\ 1 \end{bmatrix}$ operation not valid as b is not square

8 $b^Tb = \begin{bmatrix} -1 & 1 \end{bmatrix}\begin{bmatrix} -1 \\ 1 \end{bmatrix} = (-1)(-1)+1\times1 = 2$

9 $bb^T = \begin{bmatrix} -1 \\ 1 \end{bmatrix}\begin{bmatrix} -1 & 1 \end{bmatrix} = \begin{bmatrix} (-1)(-1) & (-1)\times1 \\ 1(-1) & 1\times1 \end{bmatrix} = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$

10 $A \circ A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \circ \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} 1\times1 & 2\times2 \\ 3\times3 & 4\times4 \end{bmatrix} = \begin{bmatrix} 1 & 4 \\ 9 & 16 \end{bmatrix}$

11 $b \circ c = \begin{bmatrix} -1 \\ 1 \end{bmatrix} \circ \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} (-1)\times1 \\ 1\times2 \end{bmatrix} = \begin{bmatrix} -1 \\ 2 \end{bmatrix}$



3.2. Matrix manipulations and multiplication using Python. Repeat 3.1, but this time using Python. If you are using a vector, make sure the dimensions of the vector match what you'd expect, for example, matrix $\mathbf{b}$ is a $[2 \times 1]$ vector. In NumPy, unless you're specify, you'll like create a one-dimensional array of length 2 rather than a $[2 \times 1]$ vector if you don't specify - be careful of this potential pitfall. Refer to NumPy's tools for handling matrices. There may be circumstances when Python **will** produce an output, but based on the dimensions of the matrices involved, the linear algebra operation is not

possible. ***Note these cases and explain why they occur***. Please provide both the Python code AND the output of that code showing your result. If the output is an error, comment out the code and note that it cannot be computed.

Be sure to use the right operator for each operation: Matrix multiplication: `@`; Element-wise multication: `*`. For this exercise, **only** use one of those to operators for matrix or vector multiplication.

In [17]:
```
import numpy as np

# given matrices
A = np.array([[1, 2], [3, 4]])
b = np.array([[-1], [1]])
c = np.array([[1], [2]])

print(f"1. AA = {A@A}")
print(f"2. AA^T = {A@A.T}")
print(f"3. Ab = {A@b}")
# print(f'4. Ab^T = {np.dot(A,b.T)}')
print(
    f"4. Ab_T cannot be computed. Since the A has the dimension"
    f" of (2,2) and b^T has the dimension of (1,2) They are not"
    f" aligned."
)
# print(f'5. bA = {np.dot(b,A)}')
print(
    f"5. bA cannot be computed. Since the b has the dimension "
    f"of (2,1) and A has the dimension of (2,2) They are not aligned."
)
print(f"6. b^TA = {b.T@A}")
# print(f'7. bb = {np.dot(b,b)}')
print(
    f"7. bb cannot be computed. Since the b has the dimension of "
    f"(2,1), it is not square"
)
print(f"8. b^Tb = {b.T@b}")
print(f"9. bb^T = {b@b.T}")
print(f"10. AoA = {A*A}")
print(f"11. boc = {b*c}")
# print(f'12. b^Tb^T = {np.dot(b.T,b.T)}')
print(
    f"12. b^Tb^T cannot be computed. Since the b^T has the dimension "
    f"of (1,2), it is not square"
)
print(f"13. b+c^T = {b+c.T}")
print(
    f"13. b+c^T cannot be computed. The b has the dimension of (2,1)"
    f" and c^T has the dimension of (1,2). They do not have the same"
    f" dimension to do addition. However, NumPy would still provide"
    f" an output as above. It is due to broadcasting, which is a "
    f"feature of NumPy to automatically expand smaller arrays to"
    f" match'the dimensions of the larger arrays during arithmetic"
    f" operations. In this case, b is expanded from"
    f" {np.array([[-1],[1]])} to {np.array([[-1,-1],[1,1]])} "
```

```
    f" to meet the larger number of columns of c^T and c^T is"
    f" expanded from {np.array([[1, 2]])} to"
    f" {np.array([[1, 2],[1,2]])} to meet the larger number of"
    f" rows of b."
)
print(f"14. A^-1b = {np.linalg.inv(A)@b}")
print(f"15. b^TAb = {b.T@A@b}")
# print(f'16. bAb^T = {b@A@b.T}')
print(
    f"16. bAb^T cannot be computed. Since the b has the dimension"
    f" of (2,1), A has the dimension of (2,2), and b^T has the "
    f"dimension of (1,2). They are not aligned."
)
```

```
1. AA = [[ 7 10]
 [15 22]]
2. AA^T = [[ 5 11]
 [11 25]]
3. Ab = [[1]
 [1]]
4. Ab_T cannot be computed. Since the A has the dimension of (2,2) and b^T h
as the dimension of (1,2) They are not aligned.
5. bA cannot be computed. Since the b has the dimension of (2,1) and A has t
he dimension of (2,2) They are not aligned.
6. b^TA = [[2 2]]
7. bb cannot be computed. Since the b has the dimension of (2,1), it is not
square
8. b^Tb = [[2]]
9. bb^T = [[ 1 -1]
 [-1  1]]
10. AoA = [[ 1  4]
 [ 9 16]]
11. boc = [[-1]
 [ 2]]
12. b^Tb^T cannot be computed. Since the b^T has the dimension of (1,2), it
is not square
13. b+c^T = [[0 1]
 [2 3]]
13. b+c^T cannot be computed. The b has the dimension of (2,1) and c^T has t
he dimension of (1,2). They do not have the same dimension to do addition. H
owever, NumPy would still provide an output as above. It is due to broadcast
ing, which is a feature of NumPy to automatically expand smaller arrays to m
atch'the dimensions of the larger arrays during arithmetic operations. In th
is case, b is expanded from [[-1]
 [ 1]] to [[-1 -1]
 [ 1  1]]  to meet the larger number of columns of c^T and c^T is expanded f
rom [[1 2]] to [[1 2]
 [1 2]] to meet the larger number of rows of b.
14. A^-1b = [[ 3.]
 [-2.]]
15. b^TAb = [[0]]
16. bAb^T cannot be computed. Since the b has the dimension of (2,1), A has
the dimension of (2,2), and b^T has the dimension of (1,2). They are not ali
gned.
```

3.3. Vector Norms. Norms are the effective lengths of vectors. For example, the Euclidean norm, or L_2 norm (denoted as $||\cdot||_2$), is the most common of several types of norms. The L_2 norm can be calculated for a vector

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

as follows:

$$||\mathbf{x}||_2 = \sqrt{\sum_{k=1}^{n} x_k^2} = \sqrt{\mathbf{x}^\top \mathbf{x}}$$

What is the L_2 norm of vectors $\mathbf{d}_1 = \begin{bmatrix} 2^{-1/2} \\ -2^{-1/2} \\ 0 \end{bmatrix}$ and $\mathbf{d}_2 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$?

$$||(d_1)||_2 = \sqrt{(d_1)^T (d_1)} = \sqrt{\begin{bmatrix} 2^{-1/2} & -2^{-1/2} & 0 \end{bmatrix} \begin{bmatrix} 2^{-1/2} \\ -2^{-1/2} \\ 0 \end{bmatrix}} = \sqrt{(2^{-1/2})^2 + (}$$

$$||(d_2)||_2 = \sqrt{(d_2)^T (d_2)} = \sqrt{\begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}} = \sqrt{0^2 + 0^2 + 1^2} = 1$$

3.4. Orthogonality and unit vectors. Orthogonal vectors are frequently used in machine learning in topics such as Principal Components Analysis and feature engineering for creating decorrelated features. Knowing what an orthogonal or orthonomal basis is for a space is an important concept. Find all values of unit vectors, $\mathbf{d}_3$, that complete an orthonormal basis in a three-dimensional Euclidean space along with the two vectors: $\mathbf{d}_1 = \begin{bmatrix} 2^{-1/2} \\ -2^{-1/2} \\ 0 \end{bmatrix}$ and $\mathbf{d}_2 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$.

For review, vector $\mathbf{x}_1$ is orthonormal to vector $\mathbf{x}_2$ if (a) $\mathbf{x}_1$ is orthogonal to vector $\mathbf{x}_2$ AND $\mathbf{x}_1$ is a unit vector. Orthogonal vectors are perpendicular, which implies that their inner product is zero. A unit vector is of length 1 (meaning its L_2 norm is 1).

> To make d_3 a unit vector orthogonal to d_1 and d_2, d_3 needs to satisfy the following 3 conditions:
>
> 1. $d_3 \cdot d_1 = 0$
> 2. $d_3 \cdot d_2 = 0$
> 3. $||d_3|| = 1$
>
> Set $d_3 = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$
>
> Then we have
>
> 1. $d_3 \cdot d_1 = 2^{-1/2} \cdot x - 2^{-1/2} \cdot y + 0 \cdot z = 0$
> 2. $d_3 \cdot d_2 = 0 \cdot x + 0 \cdot y + 1 \cdot z = 0$
> 3. $||d_3|| = \sqrt{x^2 + y^2 + z^2} = 1$
>
> Therefore, $d_3 = \pm \begin{bmatrix} 2^{-1/2} \\ 2^{-1/2} \\ 0 \end{bmatrix}$

3.5. Eigenvectors and eigenvalues. Eigenvectors and eigenvalues are useful for numerous machine learning algorithms, but the concepts take time to solidly grasp. They are used extensively in machine learning including in Principal Components Analysis (PCA) and clustering algorithms. For an intuitive review of these concepts, explore this interactive website at Setosa.io. Also, the series of linear algebra videos by Grant Sanderson of 3Brown1Blue are excellent and can be viewed on youtube here. For these questions, numpy may once again be helpful.

1. In Python, calculate the eigenvalues and corresponding eigenvectors of matrix $\mathbf{B} = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 5 \\ 3 & 5 & 6 \end{bmatrix}$
2. Choose one of the eigenvector/eigenvalue pairs, $\mathbf{v}$ and λ, and show that $\mathbf{Bv} = \lambda\mathbf{v}$. This relationship extends to higher orders: $\mathbf{BBv} = \lambda^2\mathbf{v}$
3. Show that the eigenvectors are orthogonal to one another (e.g. their inner product is zero - just compute the inner product and show it is approximately 0). This is true for eigenvectors from real, symmetric matrices. In three dimensions or less, this means that the eigenvectors are perpendicular to each other. Typically we use the orthogonal basis of our standard x, y, and z, Cartesian coordinates, which allows us, if we combine them linearly, to represent any point in a 3D space. But any three orthogonal vectors can do the same. This property is used, for example, in PCA to identify the dimensions of greatest variation.

1. In Python, calculate the eigenvalues and corresponding eigenvectors of matrix $\mathbf{B} = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 5 \\ 3 & 5 & 6 \end{bmatrix}$

In [18]:
```
import numpy as np

# Define the matrix B
B = np.array([[1, 2, 3], [2, 4, 5], [3, 5, 6]])

# Compute eigenvalues and eigenvectors
eigenvalues, eigenvectors = np.linalg.eig(B)

print(
    f"The eigenvalues of matrix B are lambda_1 = {eigenvalues[0]},"
    f" lambda_2 = {eigenvalues[1]}, and lambda_3 = {eigenvalues[2]}."
)
print(
    f"The eigenvectors of matrix B are v_1 = {eigenvectors[0]},"
    f" v_2 = {eigenvectors[1]}, and v_3 = {eigenvectors[2]}."
)
```

```
The eigenvalues of matrix B are lambda_1 = 11.344814282762083, lambda_2 = -
0.5157294715892573, and lambda_3 = 0.17091518882717874.
The eigenvectors of matrix B are v_1 = [-0.32798528 -0.73697623  0.5910090
5], v_2 = [-0.59100905 -0.32798528 -0.73697623], and v_3 = [-0.73697623  0.5
9100905  0.32798528].
```

2. Choose one of the eigenvector/eigenvalue pairs, $\mathbf{v}$ and λ, and show that $\mathbf{Bv} = \lambda\mathbf{v}$. This relationship extends to higher orders: $\mathbf{BBv} = \lambda^2\mathbf{v}$.

For $\mathbf{v}_1 = \begin{bmatrix} -0.32798528 \\ -0.73697623 \\ 0.59100905 \end{bmatrix}$ and $\lambda_1 = 11.3448$

We have

$$\mathbf{Bv}_1 = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 5 \\ 3 & 5 & 6 \end{bmatrix} \begin{bmatrix} -0.32798528 \\ -0.73697623 \\ 0.59100905 \end{bmatrix} = \begin{bmatrix} 1 \cdot (-0.32798528) + 2 \cdot (-0.736976 \\ 2 \cdot (-0.32798528) + 4 \cdot (-0.736976 \\ 3 \cdot (-0.32798528) + 5 \cdot (-0.736976 \end{bmatrix}$$

$$\lambda_1\mathbf{v}_1 = 11.3448 \begin{bmatrix} -0.32798528 \\ -0.73697623 \\ 0.59100905 \end{bmatrix} = \begin{bmatrix} -3.72093206 \\ -6.70488789 \\ -8.36085845 \end{bmatrix}$$

Therefore, $\mathbf{Bv}_1 = \lambda_1\mathbf{v}_1$

Also

$$\mathbf{BBv}_1 = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 5 \\ 3 & 5 & 6 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 5 \\ 3 & 5 & 6 \end{bmatrix} \begin{bmatrix} -0.32798528 \\ -0.73697623 \\ 0.59100905 \end{bmatrix} = \begin{bmatrix} -42.2132832 \\ -76.06570795 \\ -94.85238636 \end{bmatrix}$$

$$\lambda_1^2 \mathbf{v}_1 = 11.3448_2 \begin{bmatrix} -0.32798528 \\ -0.73697623 \\ 0.59100905 \end{bmatrix} = \begin{bmatrix} -42.2132832 \\ -76.06570795 \\ -94.85238636 \end{bmatrix}$$

Therefore, $\mathbf{BBv}_1 = \lambda_1^2 \mathbf{v}_1$

In [19]:
```
print(f"Bv_1 = {B@eigenvectors[:, 0]}.")
print(f"λ_1v_1 = {eigenvalues[0] * eigenvectors[:, 0]}.")
print(f"Bv_1 = λ_1v_1")

print(f"BBv_1 = {B@B@eigenvectors[:, 0]}.")
print(f"λ_1^2v_1 = {eigenvalues[0] * eigenvalues[0] *
                    eigenvectors[:, 0]}.")
print(f"BBv_1 = λ_1^2v_1")
```

```
Bv_1 = [-3.72093206 -6.70488789 -8.36085845].
λ_1v_1 = [-3.72093206 -6.70488789 -8.36085845].
Bv_1 = λ_1v_1
BBv_1 = [-42.2132832  -76.06570795 -94.85238636].
λ_1^2v_1 = [-42.2132832  -76.06570795 -94.85238636].
BBv_1 = λ_1^2v_1
```

3. Show that the eigenvectors are orthogonal to one another (e.g. their inner product is zero - just compute the inner product and show it is approximately 0). This is true for eigenvectors from real, symmetric matrices. In three dimensions or less, this means that the eigenvectors are perpendicular to each other. Typically we use the orthogonal basis of our standard x, y, and z, Cartesian coordinates, which allows us, if we combine them linearly, to represent any point in a 3D space. But any three orthogonal vectors can do the same. This property is used, for example, in PCA to identify the dimensions of greatest variation.

$$\mathbf{v}_1 \cdot \mathbf{v}_2 = \begin{bmatrix} -0.32798528 & -0.73697623 & 0.59100905 \end{bmatrix} \begin{bmatrix} -0.59100905 \\ -0.32798528 \\ -0.73697623 \end{bmatrix} = 0$$

$$\mathbf{v}_1 \cdot \mathbf{v}_3 = \begin{bmatrix} -0.32798528 & -0.73697623 & 0.59100905 \end{bmatrix} \begin{bmatrix} -0.73697623 \\ 0.59100905 \\ 0.32798528 \end{bmatrix} = 0$$

$$\mathbf{v}_2 \cdot \mathbf{v}_3 = \begin{bmatrix} -0.59100905 \\ -0.32798528 \\ -0.73697623 \end{bmatrix} \begin{bmatrix} -0.73697623 \\ 0.59100905 \\ 0.32798528 \end{bmatrix} = 0$$

> Therefore, all the eigenvectors are orthogonal to an another.

In [20]:
```
print(f"v_1v_2 = {eigenvectors[:, 0] @ eigenvectors[:, 1]}.")
print(f"v_1v_3 = {eigenvectors[:, 0] @ eigenvectors[:, 2]}.")
print(f"v_2v_3 = {eigenvectors[:, 1] @ eigenvectors[:, 2]}.")
```

```
v_1v_2 = 3.5330797728269967e-16.
v_1v_3 = -4.778175388788784e-16.
v_2v_3 = -6.1377913704128e-16.
```

Exercise 4 - Numerical Programming with Data

Loading data and gathering insights from a real dataset. In data science, we often need to have a sense of the idiosyncrasies of the data, how they relate to the questions we are trying to answer, and to use that information to help us to determine what approach, such as machine learning, we may need to apply to achieve our goal. This exercise provides practice in exploring a dataset and answering question that might arise from applications related to the data.

Your objective. For this dataset, your goal is to answer the questions below about electricity generation in the United States.

Data. The data for this problem can be found in the `data\a1\` subfolder in the `notebooks` folder on github. The filename is `egrid2016.xlsx` . This dataset is the Environmental Protection Agency's (EPA) Emissions & Generation Resource Integrated Database (eGRID) containing information about all power plants in the United States, the amount of generation they produce, what fuel they use, the location of the plant, and many more quantities. We'll be using a subset of those data.

The fields we'll be using include:

field	description
SEQPLT16	eGRID2016 Plant file sequence number (the index)
PSTATABB	Plant state abbreviation
PNAME	Plant name
LAT	Plant latitude
LON	Plant longitude
PLPRMFL	Plant primary fuel
CAPFAC	Plant capacity factor
NAMEPCAP	Plant nameplate capacity (Megawatts MW)
PLNGENAN	Plant annual net generation (Megawatt-hours MWh)
PLCO2EQA	Plant annual CO2 equivalent emissions (tons)

For more details on the data, you can refer to the eGrid technical documents. For example, you may want to review page 51 and the section "Plant Primary Fuel (PLPRMFL)", which gives the full names of the fuel types including WND for wind, NG for natural gas, BIT for Bituminous coal, etc.

There also are a couple of "gotchas" to watch out for with this dataset:

- The headers are on the second row and you'll want to ignore the first row (they're more detailed descriptions of the headers).
- NaN values represent blanks in the data. These will appear regularly in real-world data, so getting experience working with these sorts of missing values will be important.

Questions to answer:

4.1. Which power plant generated the most energy in 2016 (measured in MWh)?

In [21]:
```
import numpy as np
import pandas as pd

path = "https://github.com/kylebradbury/ids705/raw/refs/heads/" \
"main/notebooks/data/a1/egrid2016.xlsx"
egrid = pd.read_excel(path, skiprows=[0])
print(egrid.columns)
egrid.head(5)
```

```
Index(['SEQPLT16', 'PSTATABB', 'PNAME', 'LAT', 'LON', 'PLPRMFL', 'CAPFAC',
       'NAMEPCAP', 'PLNGENAN', 'PLCO2EQA'],
      dtype='object')
```

Out[21]:

	SEQPLT16	PSTATABB	PNAME	LAT	LON	PLPRMFL	CAPFAC	NA
0	1	AK	7-Mile Ridge Wind Project	63.210689	-143.247156	WND	NaN	
1	2	AK	Agrium Kenai Nitrogen Operations	60.673200	-151.378400	NG	NaN	
2	3	AK	Alakanuk	62.683300	-164.654400	DFO	0.05326	
3	4	AK	Allison Creek Hydro	61.084444	-146.353333	WAT	0.01547	
4	5	AK	Ambler	67.087980	-157.856719	DFO	0.13657	

In [22]:
```
egrid["PLNGENAN"] = pd.to_numeric(egrid["PLNGENAN"], errors="coerce")

idx_max_gen = egrid["PLNGENAN"].idxmax()
```

```
plant_max_gen = egrid.loc[idx_max_gen, "PNAME"]
print(f"The power plant generated the most energy in 2016 is"
    f" {plant_max_gen}.")
```

```
The power plant generated the most energy in 2016 is Palo Verde.
```

> 4.1. Which power plant generated the most energy in 2016 (measured in MWh)?
>
> - The power plant generated the most energy in 2016 is Palo Verde.

4.2. Which power plant produced the most CO2 emissions (measured in tons)?

In [23]:
```
egrid["PLCO2EQA"] = pd.to_numeric(egrid["PLCO2EQA"], errors="coerce")
idx_max_co2 = egrid["PLCO2EQA"].idxmax()
plant_max_co2 = egrid.loc[idx_max_co2, "PNAME"]
print(f"The power plant produced the most CO2 emissions is"
    f" {plant_max_co2}.")
```

```
The power plant produced the most CO2 emissions is James H Miller Jr.
```

> For 4.2 Which power plant produced the most CO2 emissions (measured in tons)?
>
> - The power plant produced the most CO2 emissions is James H Miller Jr.

4.3. What is the primary fuel of the plant with the most CO2 emissions?

In [24]:
```
fuel_max_co2 = egrid.loc[idx_max_co2, "PLPRMFL"]

print(f"The primary fuel of the plant with the most CO2"
    f" emissions is {fuel_max_co2}.")
```

```
The primary fuel of the plant with the most CO2 emissions is SUB.
```

> For 4.3 What is the primary fuel of the plant with the most CO2 emissions?
>
> - The primary fuel of the plant with the most CO2 emissions is SUB.

4.4. What is the name of the northern-most power plant in the United States?

In [25]:
```
egrid["LAT"] = pd.to_numeric(egrid["LAT"], errors="coerce")
idx_north = egrid["LAT"].idxmax()
north_plant_name = egrid.loc[idx_north, "PNAME"]
print(f"The northern-most power plant in the United States"
    f" is {north_plant_name}.")
```

```
The northern-most power plant in the United States is Barrow.
```

> For 4.4 What is the name of the northern-most power plant in the United States?
>
> - The northern-most power plant in the United States is Barrow.

4.5. What is the state where the northern-most power plant in the United States is located?

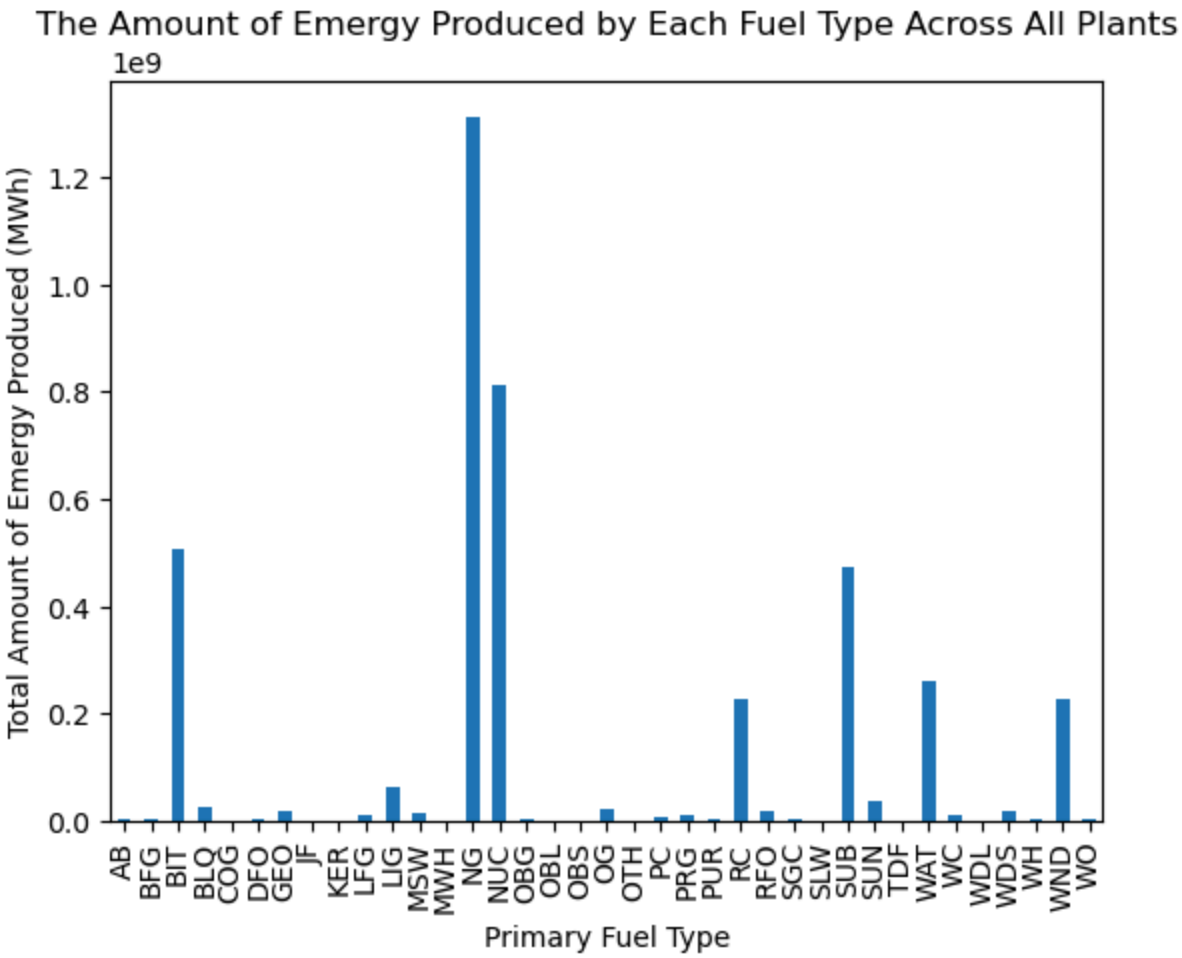
In [26]:
```
north_state = egrid.loc[idx_north, "PSTATABB"]

print(f"The northern-most power plant in the United States is"
    f" {north_state}.")
```

```
The northern-most power plant in the United States is AK.
```

> For 4.5 What is the state where the northern-most power plant in the United States is located?
>
> - The northern-most power plant in the United States is AK.

4.6. Plot a bar plot showing the amount of energy produced by each fuel type across all plants.

In [27]:
```
import matplotlib.pyplot as plt

fuel_sums = egrid.groupby("PLPRMFL")["PLNGENAN"].sum()
fuel_sums.plot.bar()
plt.xlabel("Primary Fuel Type")
plt.ylabel("Total Amount of Emergy Produced (MWh)")
plt.title("The Amount of Emergy Produced by Each Fuel Type"
    f" Across All Plants")
plt.show()
```



4.7. From the plot in (D), which fuel for generation produces the most energy (MWh) in the United States?

> For 4.7, from the plot in 4.6, which fuel for generation produces the most energy (MWh) in the United States?
>
> - The fuel for generation produces the most energy (MWh) in the United States is NG.

4.8. Which state has the largest number of hydroelectric plants? In this case, each power plant counts once so regardless of how large the power plant is, we want to determine which state has the most of them. Note the primary fuel for hydroelectric plants is listed as water in the documentation.

In [28]:
```
egrid_hydro = egrid[egrid["PLPRMFL"] == "WAT"]  # or any relevant label
counts = egrid_hydro.groupby("PSTATABB").size()
state_hydro_most = counts.idxmax()
print(
    f"The state with the largest number of hydroelectric"
    f" plants is {state_hydro_most}."
)
```

```
The state with the largest number of hydroelectric plants is CA.
```

> 4.8 Which state has the largest number of hydroelectric plants?
>
> - The state with the largest number of hydroelectric plants is CA.

4.9. Which state(s) has generated the most energy (MWh) using coal? If there are more than one, list the state abbreviations in alphabetical order, separated with commas (but no spaces). You may also want to explore the documentation for the `isin()` method for pandas. Note: in the eGrid documentation, there are multiple types of coal listed; be sure to factor in each type of coal.

In [29]:
```
coal_codes = ["BIT", "SUB", "LIG", "RC", "WC"]
coal_egrid = egrid[egrid["PLPRMFL"].isin(coal_codes)]
coal_sums_by_state = coal_egrid.groupby("PSTATABB")["PLNGENAN"].sum()
max_coal_gen_state = coal_sums_by_state.idxmax()
max_coal_gen_state
print(f"The state that generated the most energy using coal"
    f" is {max_coal_gen_state}.")
```

```
The state that generated the most energy using coal is TX.
```

> For 4.9 Which state(s) has generated the most energy (MWh) using coal?
>
> - The state that generated the most energy using coal is TX.

4.10. Which primary fuel produced the *most* CO2 emissions in the United States? We would like to compare natural gas, coal, oil, and renewables but the current categories are much more specific than that. As a first step, group the data as shown below, replacing the existing labels with the replacements suggested. For example, BIT and LIG should be replaced with COAL.

- COAL = BIT, LIG, RC, SUB, WC
- OIL = DFO, JF, KER, RFO, WO
- GAS = BFG, COG, LFG, NG, OG, PG, PRG
- RENEW = GEO, SUN, WAT, WDL, WDS, WND

You may want to create a function that does this replacement prior to running your code. You can check whether or not it was successful by verifying that each of the values that should be replaced has been replaced - check that before moving on with the question.

You will want to use 'PLCO2EQA' to answer this question as it's the quantity of emissions each plant generates.

In [30]:
```
mapping = {
    # coal
    "BIT": "COAL",
    "LIG": "COAL",
```

```
    "RC": "COAL",
    "SUB": "COAL",
    "WC": "COAL",
    # oil
    "DFO": "OIL",
    "JF": "OIL",
    "KER": "OIL",
    "RFO": "OIL",
    "WO": "OIL",
    # gas
    "BFG": "GAS",
    "COG": "GAS",
    "LFG": "GAS",
    "NG": "GAS",
    "OG": "GAS",
    "PG": "GAS",
    "PRG": "GAS",
    # renew
    "GEO": "RENEW",
    "SUN": "RENEW",
    "WAT": "RENEW",
    "WDL": "RENEW",
    "WDS": "RENEW",
    "WND": "RENEW",
}

egrid["Fuel4"] = egrid["PLPRMFL"].map(mapping).fillna("OTHER")

co2_sums = egrid.groupby("Fuel4")["PLCO2EQA"].sum()
main_fuel = co2_sums.idxmax()

print(
    f"The primary fuel type that produced the most CO2"
    f" emissions in the United States is {main_fuel}."
)
```

```
The primary fuel type that produced the most CO2 emissions in the United Sta
tes is COAL.
```

> For 4.9 Which primary fuel produced the *most* CO2 emissions in the United States?
>
> - The primary fuel type that produced the most CO2 emissions in the United States is COAL.

Appendix: Definitions and identities

The symbology in this assignment conforms to the following convention:

Symbol Example	Meaning	Possible variations
X	A random variable	Upper case, non-bolded letter
$\bar{X}$	The complement of a random variable	Upper case, non-bolded letter with bar
$\mathbf{x}$	Vector ($N \times 1$)	Lower case, bolded letters/symbols
$\mathbf{X}$	Matrix ($N \times M$)	Upper case, bolded letters/symbols
$P(\cdot)$	Probability of the event within the parenthesis	Parenthesis may include one event or more events
$A \cap B$	Intersection of A and B, that is the case of events A and B occurring simultaneously	Two random variables represented by upper case unbolded letters

Below is a list of potentially helpful identities and equations for reference.

Identities and equations	Description
$E_X[X] = \int_{-\infty}^{\infty} x f_X(x) dx$	Expected value of continuous random variable X
$Var_X(X) = E_X[X^2] - E_X[X]^2$	Variance of random variable X
$\sigma_X(X) = \sqrt{Var_X(X)}$	Standard deviation of X as a function of variance
$P(X\|Y) = \frac{P(X \cap Y)}{P(Y)}$	Conditional probability of event X given event Y has occurred
$P(Y\|X) = \frac{P(X\|Y)P(Y)}{P(X)}$	Bayes' Rule
$F_X(x) = \int_{-\infty}^{x} f_X(x) dx$	CDF as a function of PDF
$f_X(x) = \frac{dF_X(x)}{dx}$	PDF as a function of CDF

Identities and equations	Description
$P(X \leq x) = F_X(x)$	Probabilistic definition of the CDF
$P(a < X \leq b) = F_X(b) - F_X(a)$	Probability the X lies between a and b
$P(A) + P(\bar{A}) = 1$	The sum of the probability of an event and its complement is 1
$P(Y) = P(Y\|X)P(X) + P(Y\|\bar{X})P(\bar{X})$	Law of Total Probability
$\int e^{-x}dx = -e^{-x}$	Indefinite integral
$\int xe^{-x}dx = -e^{-x}(x+1)$	Indefinite integral
$\int x^2e^{-x}dx = -e^{-x}(x^2+2x+2)$	Indefinite integral
$\begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1} = \frac{1}{ad-cb}\begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$	2×2 matrix inversion formula