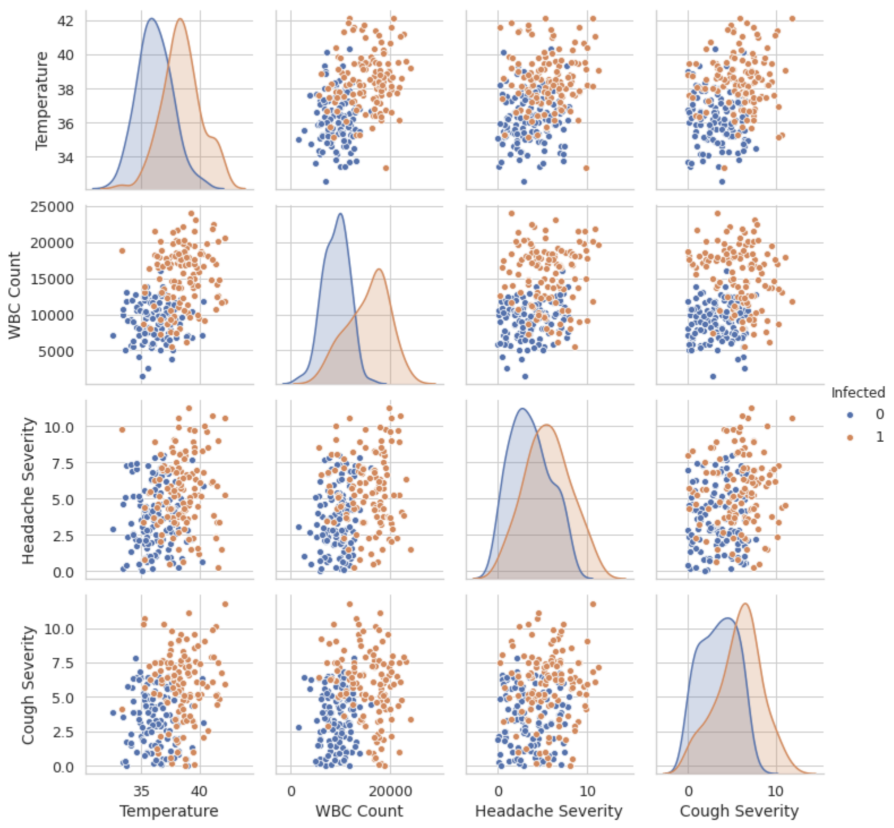
Scenario 1*:* To prevent the spread of an infectious disease, a vaccine needs to be distributed as quickly and efficiently as possible to the 15 cities that have had major outbreaks. How can you optimize the route between the cities? For this scenario, you should select cities that are relevant to the disease that you will choose for PART 2. It may be helpful to include a map of these cities to visualize the problem and bring it to life (either an existing map or create your own).

Scenario 2*:* Suppose that a new virus is starting to spread, and many clinics do not have sophisticated diagnostic tools and must be able to determine whether or not a patient has this dangerous virus based solely on easily measured symptoms. You have been collecting information on symptoms (temperature, WBC count, headache severity, and cough severity) and you need to determine which patients have this new disease and which have only a milder illness. Plots that provide an overview of the data are pasted below. The data can be accessed at this link (1 = Infected, 0 = Not infected) and read into Python using the following line:

```
pandas.read_csv('https://course-
resources.minerva.edu/uploaded_files/mu/00307707-8412/assignment-
disease-data.csv')
```



Question 1 of 17

Optional (especially for scenario 1): Include a map of the cities here.

Python 3 (384MB RAM) | Edit Run All Cells Kernel Ready | |

Drop or upload a file here

Question 2 of 17

Optimization Problem (~200 words): Describe the optimization problem for your scenario: what is the objective function? What are the decision variables? Are there any constraints? Clearly articulate each component so that it's clear how the objective value would be measured and how the decision variables would impact it.

Normal

Scenario 2 is a classification problem where we want to optimize (maximize) the diagnostic accuracy, measured by the percentage of patients classified correctly into 2 classes (Infected & Not infected) out of the total patient.
2 optimization problems:
Objective function 1: I control which feature to split on and the splitting boundary (decision variables) to maximize the purity or homogeneity within each split, which in turn maximizes the **training accuracy** (objective value) because the decision boundary determines whether the training data is divided into regions containing only either infected or uninfected.
Constraint:

- Hyperparameters (e.g. min_sample_leaf inhibits maximizing homogeneity of every leaf if its sample is too small) to avoid overfitting
- Achieve balance between underfitting (not capturing the underlying relationship complexity) and overfitting (memorizing training data and cannot be generalized to unseen data)
- Minimize false negative because it is more harmful than false positive (Type II error - uninfected diagnostic while being infected)

Objective function 2: I search for the best hyperparameter combination (decision variables) to avoid overfitting, thus maximizing the **validation accuracy** (objective value) to select the model that maximizes test accuracy.
Constraint:

- Computation time maybe expensive, but must be traded off to maximize accuracy because the problem is life-and-death.

Question 3 of 17

Optimization Technique (~200 words): What process can be used to find the optimal solution for your scenario? Identify and describe an existing **algorithm** that could be used to complete this process, including the inputs, outputs, required steps, and termination condition. Be sure that your process actually solves the problem that you identified above

Python 3 (384MB RAM) | Edit Run All Cells Kernel Ready | |

explanation, you should address whether your algorithm would lead to the global optimum and you may wish to compare your algorithm with other possible optimization techniques.

Normal

1. Maximizing training accuracy:
As the search space of split boundaries is unlimited, I use greedy instead of computational-exhaustive brute force.

- Algorithm (Random Forest):
    - Input: Training data, the initial estimate of the number of trees to combine
    - Steps:
        - (1) Randomly select a distinct subset of data and a distinct subset of features for each tree
            - (2) Decide each split on locally highest purity without considering future splits
            - (4) Repeat until meeting hyperparameter constraints or when training accuracy barely improves
            - (5) Each tree gives a label
    - Output: the majority of labels
- Limitation: not guarantee global optimum
- Advantage:
    - The randomness in step (1) holistically explores data, which avoids local optimum and underfitting
    - Combining different trees neutralized any biased or overfitted trees, which avoids overfitting and increases generalization

2. Maximizing validation accuracy
As the data sample is small-medium (250), plus each tree only trains on a proportion of data points, the search space for the best hyperparameter combination is small. Therefore, I use brute force with Grid Search cross-validation.

- Algorithm:
    - Input: Trained RandomForest model, a small-range hyperparameter grid.
    - Steps: For each combination of hyperparameter grid, modify the trained RF model's splitting
    - Output: the validated RandomForest model with the highest validation accuracy
- Advantage: guarantee global optimum

# Footnote: To explain why I do not choose perceptron classification: As 2 classes mostly overlap, separating by a linear function (perceptron) is less accurate than tree classification.

Python 3 (384MB RAM) | Edit Run All Cells Kernel Ready | |

Optional: Draw a flowchart to illustrate the process. The flowchart is a way to illustrate the main steps of the algorithm described above, ideally making them more detailed and precise. Check out this flowchart manual (option reading from session 4.2) from R. K. Walia for suggestions on common conventions to follow.

Drop or upload a file here

Question 5 of 17

Optional: Use this cell for commentary on the flow chart, justifying why this showcases a strong application of #algorithms (~100 words).

Normal

Question 6 of 17

Optional challenge: Create a program in Python to implement this optimization process. Please provide a thorough explanation of how the code works, both holistically and using in-line comments. You must also provide at least one "test case" that proves that your code is properly implemented. Testing your code is crucial to verify that your work is correct and is a great way to exercise algorithmic thinking. The test can be something like, "In [this case], it's clear that the only possible solution is X; let's check that with our code. Yes, the code outputs X as expected, so we can in principle induce that the implementation is correct." Please note that calling a function from a Python library (e.g., sklearn) without any attempt to explain how the function works or optimize its performance would not constitute a strong application.

Normal

Code Cell 1 of 14

```
In [6]
# Load libraries
import pandas as pd
import numpy as np
#from sklearn.tree import DecisionTreeClassifier # Import Decision
Tree Classifier
from sklearn.model_selection import train_test_split # Import
```

Python 3 (384MB RAM) | Edit   Run All Cells   Kernel Ready | |

```
from sklearn import metrics #Import scikit-learn metrics module for accuracy calculation
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.model_selection import GridSearchCV

#read data
data = pd.read_csv ('https://course-resources.minerva.edu/uploaded_files/mu/00307707-8412/assignment-disease-data.csv')
feature_cols = ["Infected", "Temperature",  "WBC Count",    "Headache Severity", "Cough Severity"]
X = data[feature_cols] # Features
y = data ["Infected"] # Target variable

# Split the train dataset into training set and validation set
X_trainval, X_test, y_trainval, y_test = train_test_split(X, y, test_size=0.15, random_state=42)   # 15% test set
X_train, X_val, y_train, y_val = train_test_split (X_trainval, y_trainval, test_size=0.15/0.85, random_state=42)  # 70% training set and 15% validation set

# standardize the dataset to transform the values of each variable so that they have a mean of zero and a standard deviation of one to account for  variables that have different scales or units of measurement
scaler = StandardScaler()
X_train_std = scaler.fit_transform(X_train)
X_val_std = scaler.transform(X_val)
X_test_std = scaler.transform(X_test)

```

Run Code

Code Cell 2 of 14

```
In [7]:
#Building Random Forest Classification model using Scikit-learn.
#using n_estimator as the initial number of classification to bundle in the random forest
rf = RandomForestClassifier(random_state=42, n_estimators=10)
# Train Random Forest Classification model
rf.fit (X_train_std, y_train)


#Validation
```

Python 3 (384MB RAM) | Edit Run All Cells Kernel Ready | |

```
param_grid = {
    'n_estimators': np.arange(10, 15),
    'max_depth': np.arange(2, 7),
    'min_samples_split': np.arange(7, 10),
    'min_samples_leaf': np.arange(7, 10),
    'max_features': np.arange (2,5)
}

# perform hyperparameter tuning using randomized search cross-validation
rf_random = GridSearchCV(estimator=rf, param_grid=param_grid, cv=3, verbose=2, n_jobs=-1)
rf_random.fit(X_train_std, y_train)

# print the best hyperparameters found during tuning
print("Best Hyperparameters:", rf_random.best_params_)

rf_best = RandomForestClassifier(n_estimators=rf_random.best_params_['n_estimators'],
 max_depth=rf_random.best_params_['max_depth'],
 min_samples_split=rf_random.best_params_['min_samples_split'],
 min_samples_leaf=rf_random.best_params_['min_samples_leaf'],
                                max_features = rf_random.best_params_['max_features'])
rf_best.fit(X_train_std, y_train)

#Training accuracy
y_train_pred = rf_best.predict(X_train_std)
train_accuracy = metrics.accuracy_score(y_train, y_train_pred)

# Calculate validation accuracy
y_val_pred = rf_best.predict(X_val_std)
val_accuracy = metrics.accuracy_score(y_val, y_val_pred)



print(f"Training accuracy: {train_accuracy:.4f}")
print(f"Validation accuracy: {val_accuracy:.4f}")
```

Run Code

Python 3 (384MB RAM) | Edit Run All Cells Kernel Ready | |

Out [7]

```
Fitting 3 folds for each of 675 candidates, totalling 2025 fits
[CV] max_depth=2, max_features=2, min_samples_leaf=7, min_samples_split=7,
n_estimators=10
[CV]  max_depth=2, max_features=2, min_samples_leaf=7, min_samples_split=7,
n_estimators=10, total=   0.0s
[CV] max_depth=2, max_features=2, min_samples_leaf=7, min_samples_split=7,
n_estimators=10
[CV]  max_depth=2, max_features=2, min_samples_leaf=7, min_samples_split=7,
n_estimators=10, total=   0.0s
[CV] max_depth=2, max_features=2, min_samples_leaf=7, min_samples_split=7,
n_estimators=10
[CV]  max_depth=2, max_features=2, min_samples_leaf=7, min_samples_split=7,
n_estimators=10, total=   0.0s
[CV] max_depth=2, max_features=2, min_samples_leaf=7, min_samples_split=7,
n_estimators=11
[CV]  max_depth=2, max_features=2, min_samples_leaf=7, min_samples_split=7,
n_estimators=11, total=   0.0s
[CV] max_depth=2, max_features=2, min_samples_leaf=7, min_samples_split=7,
n_estimators=11
[CV]  max_depth=2, max_features=2, min_samples_leaf=7, min_samples_split=7,
n_estimators=11, total=   0.0s
[CV] max_depth=2, max_features=2, min_samples_leaf=7, min_samples_split=7,
n_estimators=11
[CV]  max_depth=2, max_features=2, min_samples_leaf=7, min_samples_split=7,
n_estimators=11, total=   0.0s
[CV] max_depth=2, max_features=2, min_samples_leaf=7, min_samples_split=7,
n_estimators=12
[CV]  max_depth=2, max_features=2, min_samples_leaf=7, min_samples_split=7,
n_estimators=12, total=   0.0s
[CV] max_depth=2, max_features=2, min_samples_leaf=7, min_samples_split=7,
n_estimators=12
[CV]  max_depth=2, max_features=2, min_samples_leaf=7, min_samples_split=7,
n_estimators=12, total=   0.0s
```

```
[Parallel(n_jobs=-1)]: Using backend SequentialBackend with 1 concurrent
workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s
[CV]  max_depth=2, max_features=2, min_samples_leaf=7, min_samples_split=7,
n_estimators=14, total=   0.0s
[CV] max_depth=2, max_features=2, min_samples_leaf=7, min_samples_split=8,
n_estimators=10
[CV]  max_depth=2, max_features=2, min_samples_leaf=7, min_samples_split=8,
n_estimators=10, total=   0.0s
[CV] max_depth=2, max_features=2, min_samples_leaf=7, min_samples_split=8,
n_estimators=10
[CV]  max_depth=2, max_features=2, min_samples_leaf=7, min_samples_split=8,
n_estimators=10, total=   0.0s
[CV] max_depth=2, max_features=2, min_samples_leaf=7, min_samples_split=8,
n_estimators=10
[CV]  max_depth=2, max_features=2, min_samples_leaf=7, min_samples_split=8,
n_estimators=10, total=   0.0s
[CV] max_depth=2, max_features=2, min_samples_leaf=7, min_samples_split=8,
n_estimators=11
[CV]  max_depth=2, max_features=2, min_samples_leaf=7, min_samples_split=8,
n_estimators=11, total=   0.0s
[CV] max_depth=2, max_features=2, min_samples_leaf=7, min_samples_split=8,
```

Python 3 (384MB RAM) | Edit   Run All Cells   Kernel Ready | |

```
[CV]  max_depth=2, max_features=2, min_samples_leaf=7, min_samples_split=8,
n_estimators=11, total=   0.0s
[CV] max_depth=2, max_features=2, min_samples_leaf=7, min_samples_split=8,
n_estimators=11
[CV]  max_depth=2, max_features=2, min_samples_leaf=7, min_samples_split=8,
n_estimators=11, total=   0.0s
[CV] max_depth=2, max_features=2, min_samples_leaf=7, min_samples_split=8,
n_estimators=12
[CV]  max_depth=2, max_features=2, min_samples_leaf=7, min_samples_split=8,
n_estimators=12, total=   0.0s
[CV] max_depth=2, max_features=2, min_samples_leaf=7, min_samples_split=8,
n_estimators=12
[CV]  max_depth=2, max_features=2, min_samples_leaf=7, min_samples_split=8,
[Parallel(n_jobs=-1)]: Done 2025 out of 2025 | elapsed:   29.1s finished
```

Code Cell 3 of 14

In [8]

```python
# (optional) use this code cell for the optional challenge
# TEST SET PERFORMANCE
# Combine the training and validation data
# To use the tuned rf_best model on real-world data, you would need to fit it on the entire dataset, including both the training and validation data
# -> Make predictions on new, unseen data.
X_train_full = np.concatenate((X_train_std, X_val_std), axis=0)
y_train_full = np.concatenate((y_train, y_val), axis=0)

# Fit the rf_best model on the entire dataset
rf_best_full = RandomForestClassifier (random_state = 42, max_depth=rf_random.best_params_['max_depth'], min_samples_leaf=rf_random.best_params_['min_samples_leaf'], min_samples_split= rf_random.best_params_['min_samples_split'], n_estimators=rf_random.best_params_['n_estimators'], max_features = rf_random.best_params_['max_features'])

rf_best_full.fit(X_train_full, y_train_full)

# Make predictions on the test data using the tuned rf_best_full model
y_test_pred = rf_best_full.predict(X_test_std)

# Calculate the test accuracy
test_accuracy = metrics.accuracy_score(y_test, y_test_pred)
print("Test Accuracy:", test_accuracy)
```

Run Code

Python 3 (384MB RAM) | Edit Run All Cells Kernel Ready | |

```
Out [8]     Test Accuracy: 1.0
```

PART 2: SIMULATION

The SIR model of the spread of disease is commonly used to help understand how a disease might move through a population. You were introduced to this with the NetLogo agent-based model in NS50 and will review it again in Weeks 7 and 8 of CS51. Check out the class readings to learn about this model.

For this assignment, you will select one disease of your choice to model. Please choose a disease from THIS LIST to investigate. If you would like to select your own disease to model, you may email your professor with the disease and parameter descriptions for approval. You must select an infectious disease (one that is transmitted from person to person through a viral, bacterial, or parasitic agent), not a genetic or environmental disease.

2.1 Numerical Modeling and Simulation

For this part of the assignment, you'll consider the SIR model described by the set of differential equations below, and the numerical simulation in Python via Euler's method.

$$\frac{dS}{dt} = -\frac{b}{N}S(t)I(t)$$

$$\frac{dI}{dt} = \frac{b}{N}S(t)I(t) - kI(t)$$

$$\frac{dR}{dt} = kI(t)$$

Optional challenge: Modify the basic SIR model to add a layer of real-world complexity. A few ideas are listed below. Explain the key features of the extended model, including the modified differential equations and a full description and classification of any new variables and parameters following the steps above. The expectation here is to use this extended model for the rest of the analysis in this section, including the simulation and the interpretations.

- Vaccination
- Antibiotic use and/or development of antibiotic resistance
- Variability in population susceptibility (e.g. children and the elderly have different rates of infection compared to young adults).
- Birth and death rates in the population

(2.1.1) Variables and Parameters (~250 words total for the next three cells) [#variables]: This section serves to set up an initial analysis of the SIR model.

Question 7 of 17

State the disease you selected to model.

Normal

The COVID outbreak in District of Columbia in USA from March 1 2021 to May 31 2021

Question 8 of 17

Identify the relevant **variables** of the model, fully classify what type of variables they are, and explain what they mean in the context of your model. Next, identify appropriate numerical values and units for the **initial values** of the variables. You're encouraged to use empirical data if possible to justify these values. You may also complete a well-reasoned #estimation for any values that are difficult to justify with empirical data. Include APA citations for any external sources used. Note that you can work with population values S, I, R, or proportions, S/N, I/N, R/N, as long as you are consistent.

Normal

As SIR is based on differential equations, expressing how variables change as a function of their own rate of change (derivatives) based on the change in time. Therefore, all variables are continuous.

- Independent variable: time **t** (day)
- Dependent variables:

S = S(t): the number of *susceptible* individuals. The deduction in S = the addition in I when individuals become infected.

I = I(t): the number of *infected* individuals. The deduction in I = the addition in R when individuals recover/ dead.

R = R(t): the number of *recovered* individuals. They cannot be infected again and cannot spread the disease.

- Initial value:
    - t0 = 0 (unit: day)
    - I0 = 10646 individual
    - R0 = 30038 individuals
    - S0 = Population - I0 - R0= 700000 - 10646 - 30038 = 659316 individuals

Python 3 (384MB RAM) | Edit Run All Cells Kernel Ready | |

Question 9 of 17

Explain what the relevant **parameters** (b and k) are and what they mean. For your model, identify and justify appropriate numerical values and units for b and k. As above, you may include a well-reasoned estimate using empirical data to support your justification. Further, explain what it would mean for the parameters (b and k) to be smaller or larger. Consider what real-world factors, in the context of the disease you selected, would reduce or increase these parameters.

Normal B I U

b = Transmission rate (unit: per day) is the rate at which susceptible individuals become infected.

- Estimation: 0.085 -> Once every 11.76 days, an infected person will infect another susceptible/ 8.5% of susceptible population gets infected per day.
- If b increase, the days for an infected to infect a susceptible decreases -> increase disease transmission.
- Factors that decrease b:
    - Early social distancing measures reduce contact between susceptible and infected individuals
    - Large-scale vaccination can create herd immunity that prevents disease transmission to unvaccinated susceptible individuals

k = Recovery Rate (unit: per day) is the rate at which infected individuals recover.

- Estimation: 0.062 -> Infected individual recovers after 16.1 days/ 6.2% of infected population recover per day.
- If k increase -> the recover period decrease -> the quicker the infected population decreases -> decrease disease transmission.
- Effective healthcare treatments can increase the recovery rate by reducing the duration and severity of the illness.

Question 10 of 17

(2.1.2) Euler's Method Description (~150 words) [#algorithms]: Explain what it means to solve the SIR differential equations and how Euler's method works as an algorithm to achieve a numerical solution via simulation. In your explanation, identify the inputs, outputs, and steps that the algorithm takes, and consider the role of the step size ("h") in the algorithm.

Normal B I U

The solution of SIR is 3 differential functions S(t), I(t), and R(t) which indicates how those variables change in time.

Python 3 (384MB RAM) | Edit Run All Cells Kernel Ready | |

Because the formula is unknown within a continuous timeframe, Euler's method approximates the derivatives dS/dt, dI/dt, and dR/dt using S(t), I(t), and R(t) at discrete time steps.

- Input:
  - t0 & t_end: time range
  - Step size h: size of each interval.
  - Initial condition (S0, I0, Ro) at t0.
  - Population size N.
- Steps:
  - Update S0, I0, R0 at next time step using functions of step size and derivatives (Substitute derivatives by the relation between S, I, R and b, k)
  - Repeat updating S, I, R until reaching t_end
- Output: multiple discrete bivariate data points (S1,t1), (I1,t1), (R1,t1) etc

Tuning step size h is tuning the hyperparameter to select the most accurate approximation of the real function. A smaller "h" means a smaller time interval thus minimizing approximation error in each timestep.

Question 11 of 17

(2.1.3) Euler’s Method Implementation [#algorithms, #dataviz]: Define a function that implements a numerical simulation to derive the implications of your model using Euler’s method in Python. Your simulation must generate at least one relevant visualization of the disease dynamics (see required analysis below), including a descriptive figure legend and caption. You may need to adjust the run-time and step size in your simulation to ensure the visualization is maximally informative. Include thorough comments in your code to convey your understanding of the implementation of Euler’s method.

Normal

- t(n+1) = t(n) + h
- S(n+1) = S(n) - h * (b/N * S(n) * I(n))
- I (n+1) = I(n) + h * (b/N * S(n) * I(n) - k * I(n))
- R(n+1) = R(n) + h * (k * I(n))

Code Cell 4 of 14

In [1]

```python
# Euler’s Method Implementation
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams.update({'font.size': 15})
plt.rcParams["figure.figsize"] = [8,5]

```

Python 3 (384MB RAM) | Edit Run All Cells Kernel Ready | |

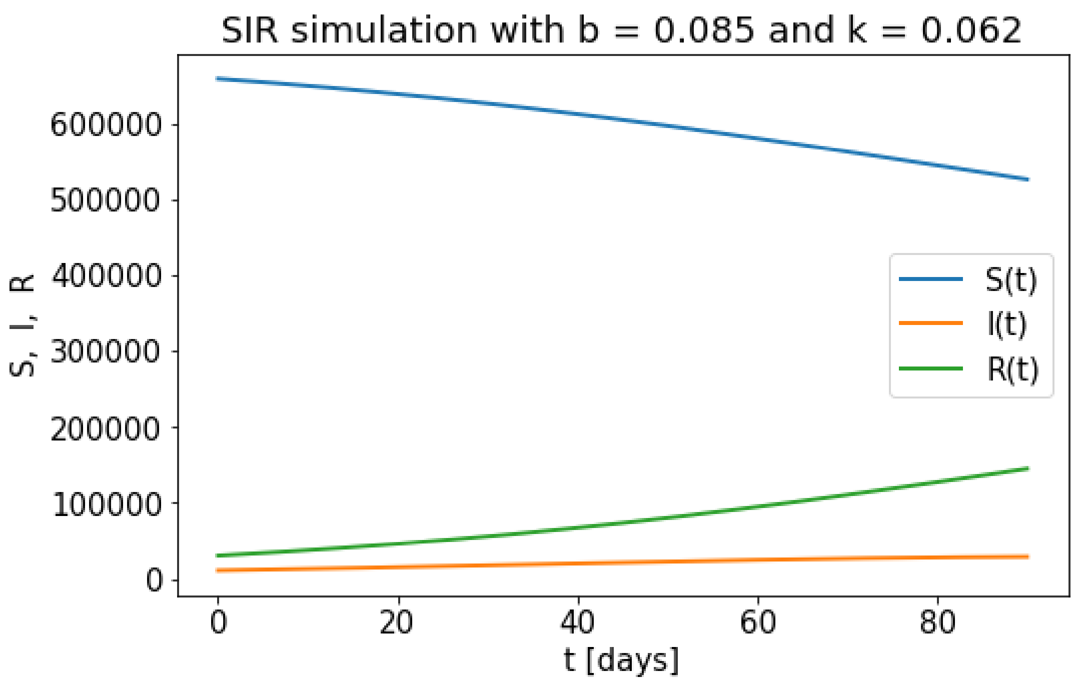
```python
def SIR_Euler(b,k,initial_conds):
    t0 = 0 #
    t_end = 90 #
    h = 0.05 #
    steps = int((t_end - t0)/h + 1) # number of steps

    # variables:
    t = np.linspace(t0, t_end, steps) # storing t values
    S = np.zeros(steps) # for storing S values
    I = np.zeros(steps) # for storing I values
    R = np.zeros(steps)

    # initial conditions:
    S[0] = initial_conds[0]
    I[0] = initial_conds[1]
    R[0] = initial_conds[2]
    N = 700000

    for n in range(steps-1): # range(start, stop, step)
        S[n+1] = S[n] - h * (b/N * S[n] * I[n])
        I[n+1] = I[n] + h * (b/N * S[n] * I[n] - k * I[n])
        R[n+1] = R[n] + h * (k * I[n])

    plt.plot(t,S,linewidth=2,label='S(t)')
    plt.plot(t,I,linewidth=2,label='I(t)')
    plt.plot(t,R,linewidth=2,label='R(t)')
    plt.xlabel('t [days]')
    plt.ylabel('S,  I,  R')
    plt.legend(loc='best')
    plt.title ("SIR simulation with b = 0.085 and k = 0.062")
    plt.show()

# parameters:
infection_rate = 0.085 #
recovery_rate = 0.062 #

# initial conditions:

I0 = 10646
R0 = 30038
S0 = 659316
initial_vals = [S0,I0, R0]
# call the function to run the simulation
```

Python 3 (384MB RAM) | Edit Run All Cells Kernel Ready | |

```python
SIR_Euler(b=infection_rate, k=recovery_rate,
initial_conds=initial_vals)
```

Run Code

Out [1]



Code Cell 5 of 14

In [2]

```python
# extra# Euler’s Method Implementation
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams.update({'font.size': 15})
plt.rcParams["figure.figsize"] = [8,5]

def SIR_Euler(b,k,initial_conds):
    t0 = 0 #
    t_end = 90 #

    h = 0.05 #
    steps = int((t_end - t0)/h + 1) # number of steps

    # variables:
    t = np.linspace(t0, t_end, steps) # storing t values
    S = np.zeros(steps) # for storing S values
    I = np.zeros(steps) # for storing I values
```

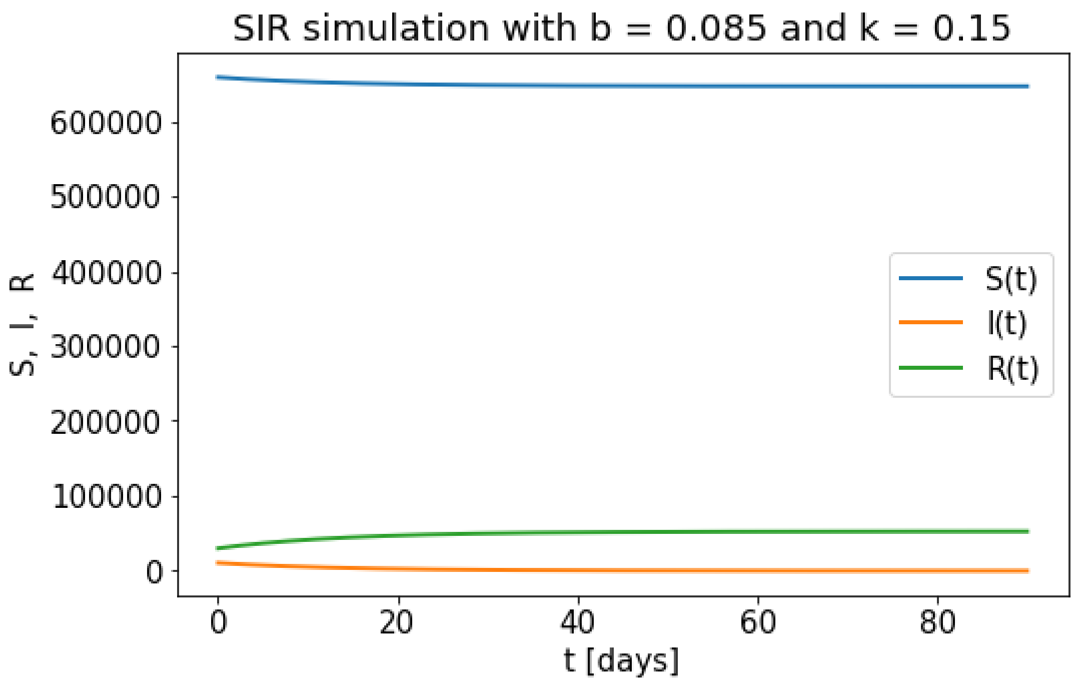
Python 3 (384MB RAM) | Edit Run All Cells Kernel Ready | |

```
    R = np.zeros(steps)

    # initial conditions:
    S[0] = initial_conds[0]
    I[0] = initial_conds[1]
    R[0] = initial_conds[2]
    N = 700000

    for n in range(steps-1): # range(start, stop, step)
        S[n+1] = S[n] - h * (b/N * S[n] * I[n])
        I[n+1] = I[n] + h * (b/N * S[n] * I[n] - k * I[n])
        R[n+1] = R[n] + h * (k * I[n])

    plt.plot(t,S,linewidth=2,label='S(t)')
    plt.plot(t,I,linewidth=2,label='I(t)')
    plt.plot(t,R,linewidth=2,label='R(t)')
    plt.xlabel('t [days]')
    plt.ylabel('S,  I,  R')
    plt.legend(loc='best')
    plt.title ("SIR simulation with b = 0.085 and k = 0.15")
    plt.show()

# parameters:
infection_rate = 0.085 #
recovery_rate = 0.15 #

# initial conditions:

I0 = 10646
R0 = 30038
S0 = 659316
initial_vals = [S0,I0, R0]
# call the function to run the simulation
SIR_Euler(b=infection_rate, k=recovery_rate,
initial_conds=initial_vals)
```

Run Code

Python 3 (384MB RAM) | Edit Run All Cells Kernel Ready | |

Out [2]



Code Cell 6 of 14

In [3]

```
# Euler’s Method Implementation
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams.update({'font.size': 15})
plt.rcParams["figure.figsize"] = [8,5]

def SIR_Euler(b,k,initial_conds):
    t0 = 0 #
    t_end = 90 #

    h = 0.05 #
    steps = int((t_end - t0)/h + 1) # number of steps

    # variables:
    t = np.linspace(t0, t_end, steps) # storing t values
    S = np.zeros(steps) # for storing S values
    I = np.zeros(steps) # for storing I values
    R = np.zeros(steps)

    # initial conditions:
    S[0] = initial_conds[0]
    I[0] = initial_conds[1]
    R[0] = initial_conds[2]
```

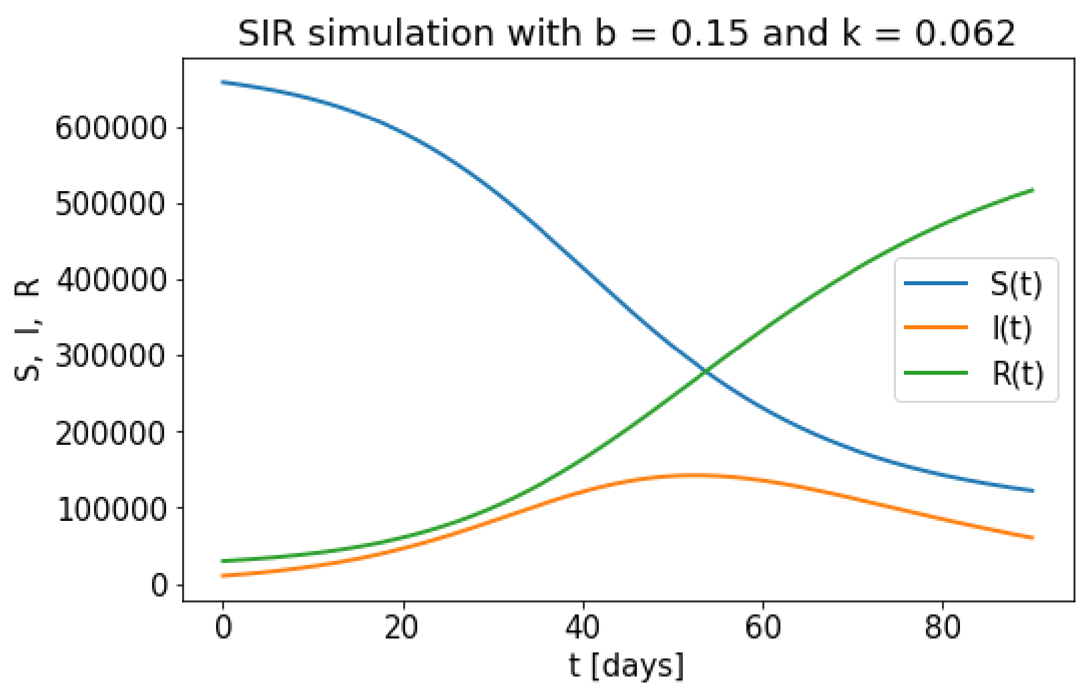
Python 3 (384MB RAM) | Edit Run All Cells Kernel Ready | |

```python

    for n in range(steps-1): # range(start, stop, step)
        S[n+1] = S[n] - h * (b/N * S[n] * I[n])
        I[n+1] = I[n] + h * (b/N * S[n] * I[n] - k * I[n])
        R[n+1] = R[n] + h * (k * I[n])

    plt.plot(t,S,linewidth=2,label='S(t)')
    plt.plot(t,I,linewidth=2,label='I(t)')
    plt.plot(t,R,linewidth=2,label='R(t)')
    plt.xlabel('t [days]')
    plt.ylabel('S,  I,  R')
    plt.legend(loc='best')
    plt.title ("SIR simulation with b = 0.15 and k = 0.062")
    plt.show()

# parameters:
infection_rate = 0.15 #
recovery_rate = 0.062 #

# initial conditions:

I0 = 10646
R0 = 30038
S0 = 659316
initial_vals = [S0,I0, R0]
# call the function to run the simulation
SIR_Euler(b=infection_rate, k=recovery_rate,
initial_conds=initial_vals)
```

Run Code

Python 3 (384MB RAM) | Edit Run All Cells Kernel Ready | |

Out [3]



Code Cell 7 of 14

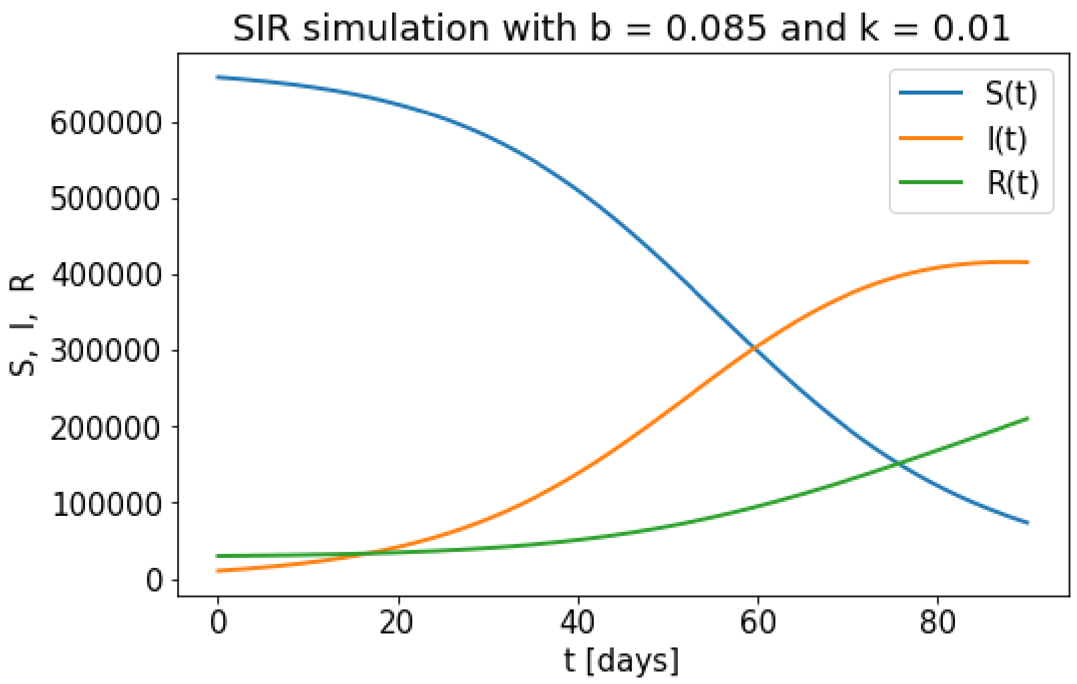
In [4]

```
# extra
# Euler’s Method Implementation
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams.update({'font.size': 15})
plt.rcParams["figure.figsize"] = [8,5]

def SIR_Euler(b,k,initial_conds):
    t0 = 0 #
    t_end = 90 #

    h = 0.05 #
    steps = int((t_end - t0)/h + 1) # number of steps

    # variables:
    t = np.linspace(t0, t_end, steps) # storing t values
    S = np.zeros(steps) # for storing S values
    I = np.zeros(steps) # for storing I values
    R = np.zeros(steps)

    # initial conditions:
    S[0] = initial_conds[0]
    I[0] = initial_conds[1]
    N = 700000

    for n in range(steps-1): # range(start, stop, step)
        S[n+1] = S[n] - h * (b/N * S[n] * I[n])
        I[n+1] = I[n] + h * (b/N * S[n] * I[n] - k * I[n])
        R[n+1] = R[n] + h * (k * I[n])

    plt.plot(t,S,linewidth=2,label='S(t)')
    plt.plot(t,I,linewidth=2,label='I(t)')
    plt.plot(t,R,linewidth=2,label='R(t)')
    plt.xlabel('t [days]')
    plt.ylabel('S,  I,  R')
    plt.legend(loc='best')
    plt.title ("SIR simulation with b = 0.085 and k = 0.01")
    plt.show()

# parameters:
infection_rate = 0.085 #
recovery_rate = 0.01 #

# initial conditions:

I0 = 10646
R0 = 30038
S0 = 659316
initial_vals = [S0,I0, R0]
# call the function to run the simulation
SIR_Euler(b=infection_rate, k=recovery_rate,
initial_conds=initial_vals)
```

Run Code

Python 3 (384MB RAM) | Edit Run All Cells Kernel Ready | |

Out [4]



Code Cell 8 of 14

```
In [ ]
```

Run Code

Question 12 of 17

(2.1.4) Results and Interpretation [#modeling, #dataviz]:

Interpret the results of the numerical simulation by making reference the output in the visualization(s). To fully interpret the results, you should re-run the simulation above multiple times with varying parameter inputs (b and k) and observe the behavior of your model. Include at least two additional visualizations here to support your answer. Does the behavior align with what you would expect these adjustments to have in reality (given what you wrote in the description of the parameters above)? (<150 words)

Normal

Graph 1: the infected population stabilizes, as new infections and recoveries balance each other out because infection and recovery rate is approximately equal (0.085 vs 0.062). The epidemic doesn't totally reach equilibrium because infection rate is still

Python 3 (384MB RAM) | Edit Run All Cells Kernel Ready | |

slightly larger than recovery rate. The equilibrium case is achieved when increasing k in Graph 2.

When infection rate increases (Graph 3), both susceptible population decreases and infected population increases more quickly. When the susceptible and recovery lines cross, the infected population peaks and begins to decrease as individuals recover.

When recovery rate decreases (Graph 4), the epidemic is more severe. Additionally, since the change (0.062 -> 0.01) is larger than the increase in infection rate in Graph 3 (0.085 -> 0.15), a larger outbreak is predicted. Indeed, when the infected and susceptible line crosses, the infected population continues to peak and does not decline yet. This is in contrast with Graph 2.

Question 13 of 17

Explain how useful this model is by considering the following guiding questions: What insights can be gained? What could it be used for? How closely do the results match what you'd expect in reality? What are the most notable assumptions of this model and what impact do they have on its usefulness? (<150 words)

Normal

Euler SIR model is a simplified and deterministic system of differential equations for the whole population to simulate disease transmission. It can provide estimates of the size and duration, the peak and decline period of the pandemic thus helping allocate resources such as medical supplies, hospital beds, etc.

The approximation fits the reality: According to COVID-19 Surveillance - DC Coronavirus, COVID cases declined from 100 to 50 approximately.

In contrast to the agent-based approach that simulates individual-level behaviors, social circle interactions, and demographic characteristics, Euler SIR model assumes individuals are identical and homogeneous meaning that people have the same susceptibility and rate of recovery, thus the disease spreads uniformly and consistently. It also does not consider variations in the infection and recovery rate over time, through the impact of vaccination or social distancing. It assumes that recovered individuals are immune to further infection and cannot transmit the disease. Those assumptions limit the accuracy and generalizability of the model to specific cases.

Question 14 of 17

Optional: include at least one multidimensional phase space plot and provide a full interpretation of what it shows. (<100 words)

Python 3 (384MB RAM) | Edit Run All Cells Kernel Ready | |