Project Final Submission Template

Step 1a: Planning

Identify the information in the file your program will read

Describe (all) the information that is available. Be sure to note any surprising or unusual features. (For example, some information sources have missing data, which may be blank or flagged using values like -99, NaN, or something else.)

The file contains all recorded plane crash data from 1918 to 2022. For each crash, it contains:

- The date of the crash. The date is written in '*year-month-day*' format.

- The time of the crash. The time format is written in '*xH xM xS*' (Hour, Minute, Second), where 'H' is in 24-hour time convention. Note that some values have been inputted oddly in that they contain no 'Hour' value; one example being '7M 0S'—this may be worth excluding in analysis. Data contains missing values denoted by *'NA'*.

- The type of aircraft. There are 1119 unique aircrafts listed in the dataset.

- The operator name. There are 8959 unique operator names. Operators are those that legally authorizes, uses, or causes to be used an aircraft. This includes names of people, and different airlines. Note that if the operator is a person, their first initials and last name is used (eg. A. D. Keith).

- The plane registration code. This is a string of varying length, composed of a mix of alpha-numeric characters, and '-' symbols. This uniquely identifies a single plane. Note that some registration codes are just a single number (eg. 5, 37), this is normal. Data contains missing values denoted by *'NA'*.

- The flight phase in which the crash took place. This is one of (5): takeoff (climb), flight (during the flight), landing (descent or approach), taxiing, or parking. Data contains missing values denoted by *'NA'*.

- The flight type. This describes what purpose the plane was flying for. This includes flights for 'testing', 'training', 'military', etc. There are 31 unique flight types. Data contains missing values denoted by *'NA'*.

- Whether or not there were survivors. This column contains 'yes' and 'no'. Note that the *NA* values could either be missing data or a valid status of 'unknown'. This is because a value of *NA* could mean that the plane was never discovered upon crashing, thus whether or not there were survivors are unknown—this may be worth excluding in analysis to avoid the innacuracy.

- The crash site type. This is one of (6): airport (less than 10km from the airport of departure/landing), plain/valley, lake/sea/ocean/river, mountains, dessert, or city. Data contains missing values denoted by *'NA'*.

- The flight schedule. This represents the *entire* route of the plane. The locations are separated by '-'. For example, a plane going from A to B to C location will be displayed as 'A-B-C'. Data contains missing values denoted by *'NA'*.

- The manufacturer's serial number (msn). This identifies the manufacturer of that plane. It is an alpha-numeric string, sometimes containing just integers and alphabetical characters. It also includes some symbols such as '/'. Data contains missing values denoted by *'NA'*.

- The year of manufacture (yom). This is the year the plane was manufactured. Data contains missing values denoted by *'NA'*.

- The flight number. This is a strange column as it has no other value other than 'NA' for any of the rows. Ideally, it is supposed to represent the plane's flight code/number. Data contains missing values denoted by *'NA'*.

- The crash location. This shows the different locations around the world where the crash occured in. Note that if the crash occured in the ocean, the value would be the ocean name appended with 'All World'. For example, a crash in the Pacific Ocean is written as 'Pacific Ocean All World'. In addition, if it occured in a non-specific enough land location, 'All *country name*' is appended. For example, a crash in 'Norfolk Island' will be written as 'Norfolk Island All Australia'. Data contains missing values denoted by *'NA'*.

- The country in which the crash occured. There are 218 unique countries in this column. Note that when a crash occures in an ocean, the country will be written as 'World'.

- The region where the crash occured. This is one of (9): North America, Europe, Africa, Central America, Asia, World, South America, Oceania, or Antarctica. These are the continents, but the continent of 'South America' is broken down into 2 different parts: 'Central America' and 'South America'. Note that, 'World' represents that the crash occured in an ocean. Data contains missing values denoted by *'NA'*.

- The number of crew members on board. Note that it is valid if a plane has 0 crew members as some are unmanned aircrafts. Data contains missing values denoted by *'NA'*.

- The number of crew fatalities. Data contains missing values denoted by *'NA'*.

- The number of passengers on board. Data contains missing values denoted by *'NA'*.

- The number of passenger fatalities. Data contains missing values denoted by *'NA'*.

- The number of 'other' fatalities; that are not crew or passenger. For instance, a plane crashing into a factory and killing 2 workers would mean a 'other fatalities' number of 2. Data contains missing values denoted by *'NA'*.

- The total number of fatalities. This is the sum of the number of crew, passenger, and other fatalities Data contains missing values denoted by *'NA'*.

- The circumstances of the crash. This shows a long description or report on what specifically happened for the crash to occur. Data contains missing values denoted by *'NA'*.

- The cause of the crash. This shows a category of what caused the crash and is one of (6): technical failure, unknown, weather, human factor, other causes, terrorism act/hijacking/sabotage.

Step 1b: Planning

Brainstorm ideas for what your program will produce

Select the idea you will build on for subsequent steps

You must brainstorm at least three ideas for graphs or charts that your program could produce and choose the one that you'd like to work on. You can choose between a line chart, histogram, bar chart, scatterplot, or pie chart.

If you would like to change your project idea from what was described in the proposal, you will need to get permission from your project TA. This is intended to help ensure that your new project idea will meet the requirements of the project. Please see the project proposal for things to be aware of when communicating with your project TA.

1. To calculate the Average Mortality Rate (total on board fatalities/total onboard) for each of the flight phases (takeoff, flight, landing, parking, taxiing) during a crash. This would be displayed on a bar graph. The x-axis can contain the flight phase, and the y-axis would be the average mortality rate.

2. To calculate the Average Mortality Rate (total on board fatalities/total onboard) per crash for each crash region (Africa, Europe, North America, etc.). This would be displayed on a bar graph. The x-axis can contain the crash region, and the y-axis can contain the average mortality rate.

3. Investigate how Average Mortality Rates (total on board fatalities/total onboard) changes overtime. This would be displayed on a line graph. The x-axis will contain the years (1918-2022) and the y-axis will contain the average mortality rate for that year.

4. To calculate the Average Mortality Rate (total on board fatalities/total onboard) for each plane model during a crash. This would be displayed on a bar graph. The x-axis

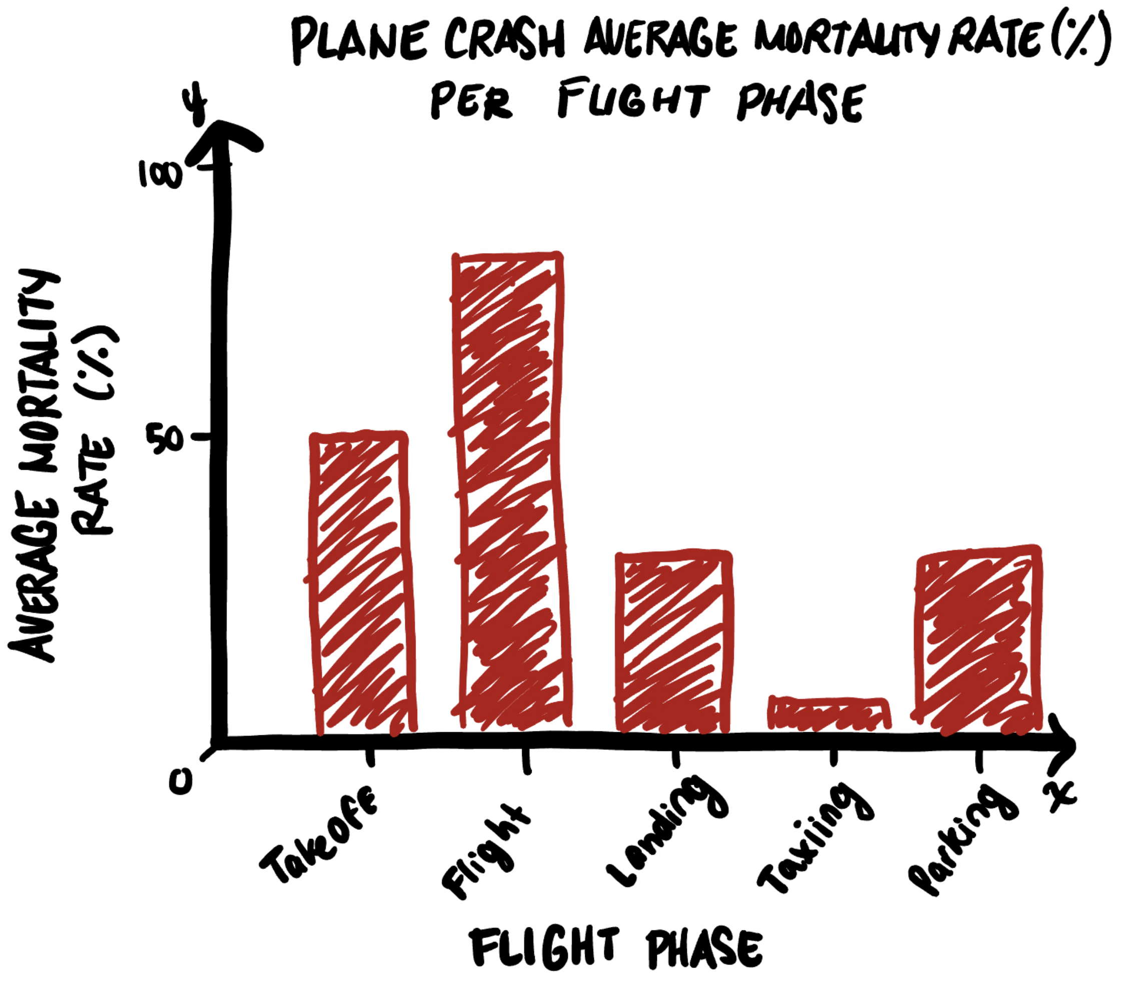
can contain the model of the plane, and the y-axis would be the average mortality rate. Due to the number of plane models (1000+), it is possible to only display the Top 10.

Note: Our original proposal was to compare the average mortality rate per plane model. After getting confirmation and approval from our project TA, we have decided to go with our new Idea 1. We now hope to find the average mortality rate (total on board fatalities/total onboard) for each flight phase during a plane crash. This would be displayed as a bar graph, where the x-axis will contain the flight phase, and the y-axis would be the average mortality rate.

Step 1c: Planning

Write or draw examples of what your program will produce

You must include an image that shows what your chart or plot will look like. You can insert an image using the Insert Image command near the bottom of the Edit menu.



Step 2a: Building

Document which information you will represent in your data definitions

Before you design data definitions in the code cell below, you must explicitly document here which information in the file you chose to represent and why that information is crucial to the chart or graph that you'll produce when you complete step 2c.

We are interested in using the flight phase (the 6th column in our dataset and index 5), the total number of passengers on board (the 19th column and index 18), the total number of crew on board (the 17th column and index 16), the number of passenger fatalities (the 20th column, and index 19), and the number of crew fatalities (the 18th column, and index 17).

Since we are interested in the average mortality rate per flight phase of a given crash, we need to store these 5 pieces of information. This is important since we are creating a calculation dervied from a combination of those columns.

To calculate the mortality rate of a single crash, we are going to be using the formula (total on board fatalities/total onboard) where 'total on board fatalities' are how many of both crew and passengers perished, and 'total onboard' are how many passengers and crew were on the flight. To obtain an average, we are categorizing based on flight phase, hence needing the 'flight phase' column.

Note: The average mortality rate calculated in this project will strictly include ON BOARD passengers and crew only. This does NOT include external deaths/'other fatalities'—which are fatalities outside of being in the plane.

Design data definitions

In [1]:
```
from cs103 import *
import csv
from typing import NamedTuple, List
from enum import Enum
import matplotlib.pyplot as plt
```

In [2]:
```
##################
# Data Definitions

FlightPhase = Enum('FlightPhase', ['takeoff', 'flight', 'landing', 'taxiing', '

# interp. a flight phase which is one of Takeoff (climb) ('takeoff'), flight ('
#     Landing (descent or approach) ('landing'), Taxiing ('taxiing'), or Parki

# examples are redundant for enumerations

@typecheck
def fn_for_flight_phase(fp: FlightPhase) -> ...:
    # template from enumeration (5 cases) and atomic distinct (5 times)
    if fp == FlightPhase.takeoff:
        return ...
    elif fp == FlightPhase.flight:
        return ...
    elif fp == FlightPhase.landing:
        return ...
    elif fp == FlightPhase.taxiing:
        return ...
    elif fp == FlightPhase.parking:
        return ...




Crash = NamedTuple('Crash', [('phase', FlightPhase),           # FlightPhase
                             ('num_passengers', int),          # in range [0, .
                             ('num_crew', int),                # in range [0, .
                             ('fatalities_passenger', int),    # in range [0, .
                             ('fatalities_crew', int)])        # in range [0, .

# interp. a crash with a flight phase ('phase'), the number of passengers on bo
#   the number of crew members on board ('num_crew'), the number of passenger f
#   and the number of crew fatalities ('fatalities_crew').

C1 = Crash(FlightPhase.takeoff, 10, 5, 10, 5)
C2 = Crash(FlightPhase.landing, 1, 1, 0, 0)
C3 = Crash(FlightPhase.taxiing, 200, 30, 150, 10)

@typecheck
def fn_for_crash(c: Crash) -> ...:
  # template based on compound (5 fields) and reference rule
  return ...(fn_for_flight_phase(c.phase), # FlightPhase
             c.num_passengers,             # in range [0, ...)
             c.num_crew,                   # in range [0, ...)
             c.fatalities_passenger,       # in range [0, ...)
             c.fatalities_crew)            # in range [0, ...)




# List[Crash]
# interp. a list of crashes

LOC0 = []
LOC1 = [C1, C2, C3]
LOC2 = [C3]

@typecheck
def fn_for_loc(loc: List[Crash]) -> ...:
    # template based on arbitrary-sized data and reference rule
    # description of the accumulator
    acc = ...      # type: ...

    for c in loc:
        acc = ...(fn_for_crash(c), acc)
    return ...(acc)
```

In [3]:
```
# we may also need a List[str] and List[float], so we will create a data defin

# List[str]
# interp. a list of strings
```

```
LOS1 = []
LOS2 = ['hello', 'world']

@typecheck
def fn_for_los(los: List[str]) -> ...:
  # template based on arbitrary-sized data

    # description of accumulator
    acc = ... # type: ...

    for s in los:
        acc = ...(s, acc)

    return ...(acc)



# List[float]
# interp. a list of floats

LOF1 = []
LOF2 = [0.0, 0.001, 5]
LOF2 = [100.5, 200.5, 1000.00]
LOF3 = [-34.4, -0.4, 10.4]

@typecheck
def fn_for_lof(lof: List[float]) -> ...:
    # template based on arbitrary-sized data
    # description of accumulator

    acc = ... # type: ...

    for s in los:
        acc = ...(s, acc)

    return ...(acc)
```

Step 2b and 2c: Building

Design a function to read the information and store it as data in your program

Design functions to analyze the data

Complete these steps in the code cell below. You will likely want to rename the analyze function so that the function name describes what your analysis function does.

Unless approved by your project TA, you cannot use libraries such as `numpy` or `pandas`. The project is meant as a way for you to demonstrate your knowledge of the learning goals in this course. While it is convinent to use external libraries, it will do all the work and will not help us gauge your mastery of the concepts.

You also cannot use built in list functions (e.g., `sum` or `average`) when writing code to do your substantial computation. Normally we encourage you to make use of what

is already available but in this case, the final project involves demonstrating skills from class (e.g., how to work with a list). Using pre-built functions for this does not enable you to demonstrate what you know.

If you wish to change your project idea, you must first obtain permission from your TA. When contacting your TA, please provide a valid reason for why you want to change your project. Each time you change your topic idea, your TA will have to evaluate it to see if it will meet all of the project requirements. This is non-trivial task during one of the busiest times of the semester. As such, the deadline for project idea changes will be 3 business days before the deadline. Note that the deliverable deadline will not be extended and there is no compensation for the time you spent on the previous idea.

In [4]:
```
#-------------------------------------------------------------------------------
#                                               FUNCTIONS
#-------------------------------------------------------------------------------


@typecheck
def main(filename: str) -> None:
    """
    Reads the crash information from the given filename, analyzes the data, and
        bar chart that shows the Average Mortality Rate (%) of plane crashes fo
        the Flight Phases (returning None).

    Returns an empty plot with the title "Average Mortality Rate by Flight Phas
        x-axis labeled with "Flight Phase" and a y-axis labeled with "Average M
        if there is no data (returning None).
    """
    # return None # stub

    # Template from HtDAP, based on function composition
    return analyze_crash(read(filename))


@typecheck
def analyze_crash(loc: List[Crash]) -> None:
    """
    Takes a list of crash data and returns a bar chart that shows the Average M
        of plane crashes for each of the Flight Phases (returning None).

    """
    # return None # stub
    # template based on Composition
    # Steps:
        # Step 1: From a full list of crashes, get a filtered list by each flig
    list_flight = filter_phase(loc, FlightPhase.flight)
    list_landing = filter_phase(loc, FlightPhase.landing)
    list_parking = filter_phase(loc, FlightPhase.parking)
    list_takeoff = filter_phase(loc, FlightPhase.takeoff)
    list_taxiing = filter_phase(loc, FlightPhase.taxiing)

        # Step 2: Take the filtered output from Step 1 and calculate a list of
    mortality_flight = get_mortality_rates(list_flight)
    mortality_landing = get_mortality_rates(list_landing)
    mortality_parking = get_mortality_rates(list_parking)
    mortality_takeoff = get_mortality_rates(list_takeoff)
    mortality_taxiing = get_mortality_rates(list_taxiing)

        # Step 3: Take the output from Step 2 and calculate the average of eac
    mortality_avg_flight = get_average(mortality_flight)
    mortality_avg_landing = get_average(mortality_landing)
    mortality_avg_parking = get_average(mortality_parking)
    mortality_avg_takeoff = get_average(mortality_takeoff)
    mortality_avg_taxiing = get_average(mortality_taxiing)

        # Step 4: Return the result from Step 3 and set it to a variable to re
    y_vals = [mortality_avg_flight,
              mortality_avg_landing,
              mortality_avg_parking,
              mortality_avg_takeoff,
              mortality_avg_taxiing]

        # Step 5: Create variables to represent the x_values (in correct order
    x_vals = ['Flight', 'Landing', 'Parking', 'Takeoff', 'Taxiing']

        # Step 6: Graph the x and y values into a bar chart.
    create_bar_chart(x_vals, y_vals)

    return None



@typecheck
def create_bar_chart(los: List[str], lof: List[float]) -> None:
    """
    Plots and shows a bar chart with the given x_value ('los') and y_value ('y

    The parameter 'los' is a list of strings as to represent the string labels
    """
    # return None # stub
    # template based on visualization

    # set figure size
    plt.rcParams["figure.figsize"] = (8, 6)

    # set variable names for the axes
    x_vals = los
    y_vals = lof

    # initialize the bar graph
    plt.bar(x_vals, y_vals,alpha=0.8, color='b')

    # set the labels for the x-axis, y-axis, and plot title, and fontsizes
    plt.xlabel('Flight Phase', fontsize=14)
    plt.ylabel('Average Mortality Rate (%)', fontsize=14)
    plt.title("Average Mortality Rate by Flight Phase", fontsize=20)


    # setting y-axis limit to display 0-100 (as a %)
    plt.ylim([0, 100.00])
    # show the plot
    plt.show()

    return None


#------------------------------------------------------------------------------
# Functions to Filter FlightPhases


@typecheck
def filter_phase(loc: List[Crash], phase: FlightPhase) -> List[Crash]:
    """
    From a list of crashes, returns a list of crashes that only occured during

    """
    # return [] # stub
    # template based on List[Crash], with one additional parameter (FlightPhase

    # a list of crashes in the flight phase seen so far
    crashes_in_phase = []      # type: List[Crash]

    for c in loc:
        if in_flight_phase(c, phase):
            crashes_in_phase.append(c)

    return crashes_in_phase



@typecheck
def in_flight_phase(c: Crash, phase: FlightPhase) -> bool:
    """
    Return True if the given crash's ('c') flight phase is the same as the give
    """
    # return False # stub
    # template from Crash and one additional parameter (FlightPhase)

    return same_phase(c.phase, phase)



@typecheck
def same_phase(phase1: FlightPhase, phase2: FlightPhase) -> bool:
    """
    Compares 2 given FlightPhases. Return True if 'phase1' matches 'phase2'; Fa
    """
    # return False # stub
    # template from FlightPhase (2 times)

    return phase1 == phase2


#------------------------------------------------------------------------------
# Functions to obtain Mortality Rate

@typecheck
```

```
def get_mortality_rate(c: Crash) -> float:
    """
    From a given crash ('c'), calculate and return the Mortality Rate as a perc

    Mortality rate is calculated by the function: ((Total Fatalities / Total On

    Returns 0.00 if Total On Board is 0 (dividing by 0 is forbidden).
    """
    # return 0.0 # stub
    # template based on Crash

    if get_total_on_board(c) > 0:
        return round(((get_total_fatalities(c)) / (get_total_on_board(c))) * (1
    else:
        return 0.00


@typecheck
def get_total_on_board(c: Crash) -> int:
    """
    Takes a crash ('c') and adds the total number of passenger and crew on boar
    """
    # return 0 # stub
    # template based on Crash

    return c.num_passengers + c.num_crew


@typecheck
def get_total_fatalities(c: Crash) -> int:
    """
    Takes a crash ('c') and adds the total number of passenger and crew fatalit
    """
    # return 0 # stub
    # template based on Crash

    return c.fatalities_passenger + c.fatalities_crew


@typecheck
def get_mortality_rates(loc: List[Crash]) -> List[float]:
    """
    From a list of crashes, return a list of its Mortality Rates.
    """
    # return [] # stub
    # template based on List[Crash]

    # a list of mortality rates seen so far
    list_mortality_rates = []      # type: List[float]

    for c in loc:
        list_mortality_rates.append(get_mortality_rate(c))

    return list_mortality_rates


#-----------------------------------------------------------------------------
```

```
# Functions to obtain Average Mortality Rate


@typecheck
def get_average(lof: List[float]) -> float:
    """
    From a given list of floats ('lof'), return the Average.
    """

    # return 0.0 # stub
    # template based on List[float]

    if len(lof) > 0:
        return get_sum(lof) / len(lof)
    else:
        return 0.00


@typecheck
def get_sum(lof: List[float]) -> float:
    """
    Return the sum of all elements in a list of floats ('lof').
    """
    # return 0.0 # stub
    # template based on List[float]

    # the sum of elements seen so far
    total = 0 # type: float

    for f in lof:
        total = total + f

    return total


#--------------------------------------------------------------------------------

@typecheck
def read(filename: str) -> List[Crash]:
    """
    reads information from the specified file and returns a list of crashes.
    """
    # return []  #stub
    # Template from HtDAP
    # loc contains the result so far
    loc = [] # type: List[Crash]

    with open(filename) as csvfile:

        reader = csv.reader(csvfile)
        next(reader) # skip header line

        for row in reader:
            # you may not need to store all the rows, and you may need
            # to convert some of the strings to other types

            # row[5]  -> flight phase
            # row[18] -> number of passengers on board
            # row[16] -> number of crew members on board
            # row[19] -> number of fatalities (passenger)
            # row[17] -> number of fatalities (crew)

            if is_reliable(row):
                c = Crash(parse_flight_phase(row[5]),
                          parse_int(row[18]),
                          parse_int(row[16]),
                          parse_int(row[19]),
                          parse_int(row[17]))

                loc.append(c)

    return loc



@typecheck
def is_reliable(row: List[str]) -> bool:
    """
    From a list of strings, return True if the flight phase (row[5]), number of
        number of crew members on board (row[16]), number of passenger fataliti
        the number of crew fatalities (row[17]) are not 'NA'; False otherwise.

    Flight phases, number of passengers, number of crew, number of passenger fa
        number of crew fatalities with an 'NA' value are NOT reliable since the
    """
    # return False # stub
    # template from List[str]

    return row[5] != 'NA'and row[18] != 'NA' and row[16] != 'NA' and row[19] !=


@typecheck
def parse_flight_phase(flight_phase: str) -> FlightPhase:
    """
    Given a flight_phase which must be one of 'Takeoff (climb)','Flight', 'Lanc
        'Taxiing', or 'Parking', return the corresponding flight phase.

    """
#     return FlightPhase.takeoff # stub

    # template from atomic non-distinct
    if (flight_phase == 'Takeoff (climb)'):
        return FlightPhase.takeoff
    elif (flight_phase == 'Flight'):
        return FlightPhase.flight
    elif (flight_phase == 'Landing (descent or approach)'):
        return FlightPhase.landing
    elif (flight_phase == 'Taxiing'):
        return FlightPhase.taxiing
    elif (flight_phase == 'Parking'):
        return FlightPhase.parking
```

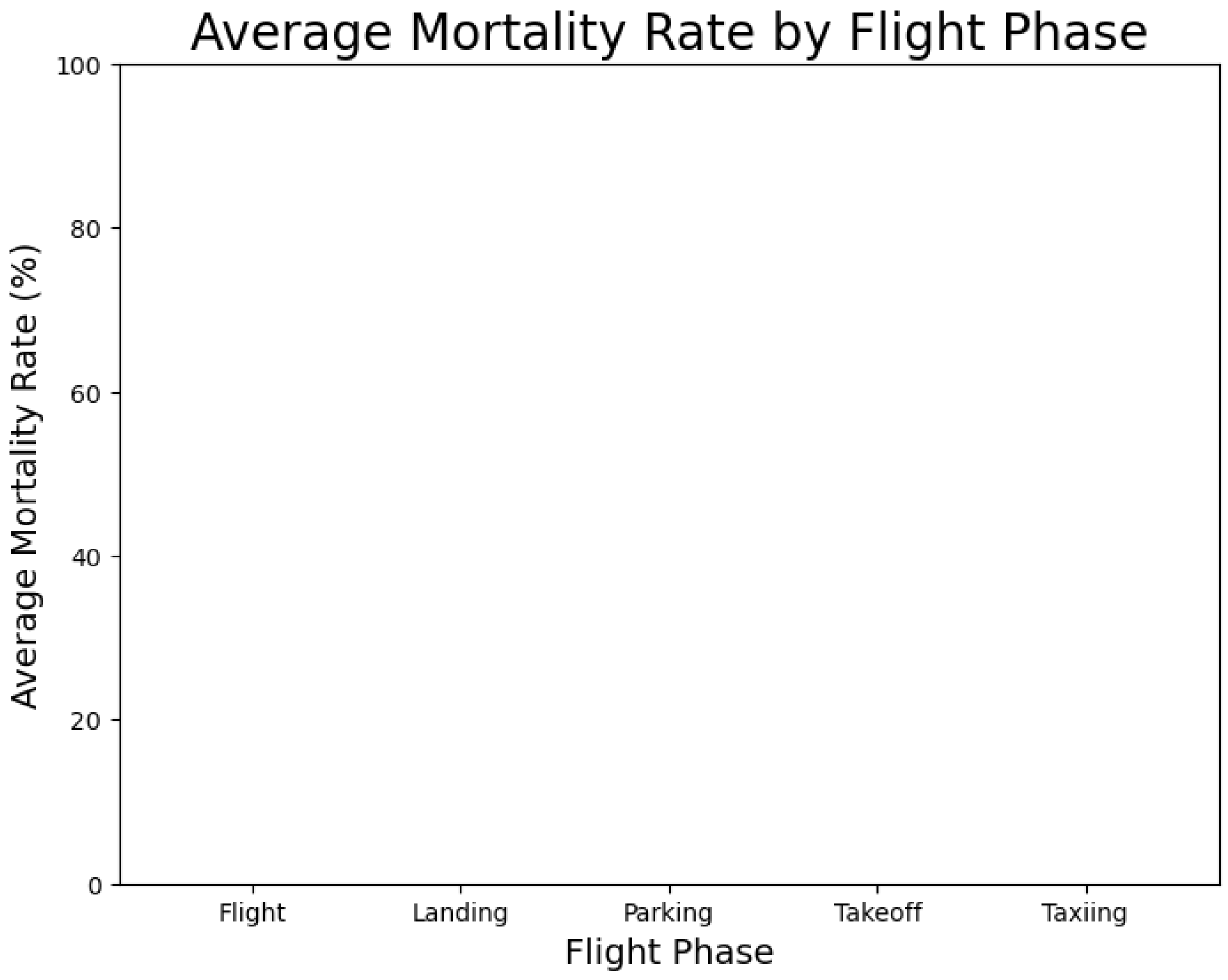
```
#-----------------------------------------------------------------------------
#                                                    TESTING
#-----------------------------------------------------------------------------

# Testing for main
print('main')
start_testing()
        # Empty
# Should produce a plot with title: " Average Mortality Rate by Flight Phase",
# x-axis should have a label showing the different flight phases ('Flight', 'La
# and the y-axis would show a label showing the Average Mortality Rate percenta
# This particular graph should be empty.
expect(main('test_empty.csv'), None)

        # Normal Case
# Should produce a plot with title: " Average Mortality Rate by Flight Phase",
# x-axis should have a label showing the different flight phases ('Flight', 'La
# and the y-axis would show a label showing the Average Mortality Rate percenta
# This graph would have 5 visible bars. The values in the bar chart are:
# 'Flight' (y at 100.00%), 'Landing' (y at 60.00%), 'Parking' (y at 13.60%), 'T
# The bar graph should look roughly like this:

#          |
#   100.00 |     #
#          |     #
#    80.00 |     #
#          |     #
#    60.00 |     #          #
#          |     #          #                               #
#    40.00 |     #          #                    #          #
#          |     #          #                    #          #
#    20.00 |     #          #                    #          #
#          |     #          #         #          #          #
#     0.00 +--------------------------------------------------------
#             Flight     Landing    Parking   Takeoff    Taxiing


expect(main('test_plane_crash_normal_case.csv'), None)

        # One Line
# Should produce a plot with title: " Average Mortality Rate by Flight Phase",
# x-axis should have a label showing the different flight phases ('Flight', 'La
# and the y-axis would show a label showing the Average Mortality Rate percenta
# This graph would have 1 visible bar at 'Landing' (y at 100.00%) and should lo

#          |
#   100.00 |                #
#          |                #
#    80.00 |                #
#          |                #
#    60.00 |                #
#          |                #
#    40.00 |                #
#          |                #
#    20.00 |                #
#          |                #
#     0.00 +--------------------------------------------------------
#             Flight     Landing    Parking   Takeoff    Taxiing

expect(main('test_plane_crash_one_line.csv'), None)


        # Two Lines
# Should produce a plot with title: " Average Mortality Rate by Flight Phase",
# x-axis should have a label showing the different flight phases ('Flight', 'La
# and the y-axis would show a label showing the Average Mortality Rate percenta
# This graph would have 1 visible bar at 'Takeoff' (y at 90.56%) and should loo

#          |
#   100.00 |
#          |                                    #
#    80.00 |                                    #
#          |                                    #
#    60.00 |                                    #
#          |                                    #
#    40.00 |                                    #
#          |                                    #
#    20.00 |                                    #
#          |                                    #
#     0.00 +-------------------------------------------------------
#             Flight     Landing    Parking   Takeoff    Taxiing

expect(main('test_plane_crash_two_lines.csv'), None)


        # All Phase Types
# Should produce a plot with title: " Average Mortality Rate by Flight Phase",
# x-axis should have a label showing the different flight phases ('Flight', 'La
# and the y-axis would show a label showing the Average Mortality Rate percenta
# This graph would have 2 visible bars at 'Landing' and 'Takeoff'. The values i
# 'Flight' (y at 0.00%), 'Landing' (y at 50.00%), 'Parking' (y at 0.00%), 'Take
# The bar graph should look roughly like this:


#          |
#   100.00 |                                   #
#          |                                   #
#    80.00 |                                   #
#          |                                   #
#    60.00 |                                   #
#          |                #                  #
#    40.00 |                #                  #
#          |                #                  #
#    20.00 |                #                  #
#          |                #                  #
#     0.00 +-------------------------------------------------------
#             Flight     Landing    Parking   Takeoff    Taxiing

expect(main('test_plane_crash_all_phase_types.csv'), None)

        # Multi-line with NA
# Should produce a plot with title: " Average Mortality Rate by Flight Phase",
# x-axis should have a label showing the different flight phases ('Flight', 'La
# and the y-axis would show a label showing the Average Mortality Rate percenta
# This graph would have 1 visible bar in 'Landing'. The values in the bar chart
# Flight' (y at 0.00%), 'Landing' (y at 20.00%), 'Parking' (y at 0.00%), 'Takeo
# The bar graph should look roughly like this:


#         |
#  100.00 |
#         |
#   80.00 |
#         |
#   60.00 |
#         |
#   40.00 |
#         |
#   20.00 |                #
#         |                #
#    0.00 +---------------------------------------------------------
#            Flight     Landing    Parking   Takeoff    Taxiing
expect(main('test_plane_crash_multi_line_with_na.csv'), None)


        # All NA
# Should produce a plot with title: " Average Mortality Rate by Flight Phase",
# x-axis should have a label showing the different flight phases ('Flight', 'La
# and the y-axis would show a label showing the Average Mortality Rate percenta
# This particular graph should be empty.
expect(main('test_plane_crash_all_na.csv'), None)
summary()



# Testing for ananlyze_crash
print('analyze_crash')
# start_testing()

# Should produce a plot with title: " Average Mortality Rate by Flight Phase",
# x-axis should have a label showing the different flight phases ('Flight', 'La
# and the y-axis would show a label showing the Average Mortality Rate percenta
# This particular graph should be empty.
expect(analyze_crash([]), None)

# Should produce a plot with title: " Average Mortality Rate by Flight Phase",
# x-axis should have a label showing the different flight phases ('Flight', 'La
# and the y-axis would show a label showing the Average Mortality Rate percenta
# This graph should have a bar at 'Landing' (y at roughly 35%),'Takeoff' (y at
#  Taxiing (y at roughly 69%) and would look vaguely like this:

#         |
#  100.00 |                                    #
#         |                                    #
#   80.00 |                                    #
#         |                                    #           #
#   60.00 |                                    #           #
#         |                                    #           #
#   40.00 |                                    #           #
#         |                #                   #           #
#   20.00 |                #                   #           #
#         |                #                   #           #
#    0.00 +-------------------------------------------------------
#            Flight     Landing    Parking   Takeoff    Taxiing

expect(analyze_crash([C1, Crash(FlightPhase.landing, 4, 10, 2, 3),C3]), None)


# Should produce a plot with title: " Average Mortality Rate by Flight Phase",
# x-axis should have a label showing the different flight phases ('Flight', 'La
# and the y-axis would show a label showing the Average Mortality Rate percenta
# This graph should have a bar at 'Flight' (y at 10%),'Landing' (y at roughly 3
#  Taxiing (y at roughly 69%) and would look vaguely like this:

#         |
#  100.00 |                                    #
#         |                                    #
#   80.00 |                                    #
#         |                                    #
#   60.00 |                                    #           #
#         |                #                   #           #
#   40.00 |                #                   #           #
#         |                #                   #           #
#   20.00 |                #         #         #           #
#         |     #          #         #         #           #
#    0.00 +-------------------------------------------------------
#            Flight     Landing    Parking   Takeoff    Taxiing

expect(analyze_crash([C1, Crash(FlightPhase.landing, 5, 5, 0, 5),
                           Crash(FlightPhase.flight, 5, 5, 0, 1),
                           Crash(FlightPhase.parking, 5, 5, 0, 2), C3]), None)
summary()



# Testing for create_bar_chart
print('create_bar_chart')
start_testing()

# Should produce a plot with title: " Average Mortality Rate by Flight Phase",
# x-axis should have a label showing the different flight phases ('Flight', 'La
# and the y-axis would show a label showing the Average Mortality Rate percenta
# This particular graph should be empty.
expect(create_bar_chart([],[]), None)

# Should produce a plot with title: " Average Mortality Rate by Flight Phase",
# x-axis should have a label showing the different flight phases ('Flight', 'La
# the y-axis would show a label showing the Average Mortality Rate percentage (
# Additionally, there should be 2 bars on the chart: Flight at 50.00, and Landi


#         |
#  100.00 |                #
#         |                #
#         |                #
#         |                #
#         |                #
#   50.00 |    #           #
#         |    #           #
#         |    #           #
#         |    #           #
#         |    #           #
#         +-----------------------------------
#            Flight     Landing

expect(create_bar_chart(['Flight', 'Landing'],[50.00, 100.00]), None)

# Should produce a plot with title: " Average Mortality Rate by Flight Phase",
# x-axis should have a label showing the different flight phases ('Flight', 'La
# the y-axis would show a label showing the Average Mortality Rate percentage (
# Additionally, there should be 3 bars on the chart: 'A' (y-value of 32.00), 'E
# The bar graph would look roughly like this:

#         |
#  100.00 |
#         |
#   80.00 |
#         |
#   60.00 |            #
#         |            #
#   40.00 |            #
#         |    #       #
#   20.00 |    #       #
#         |    #       #         #
#    0.00 +-----------------------------------
#              A           B         C

expect(create_bar_chart(['A', 'B', 'C'],[32.00, 65.00, 10.00]), None)
summary()




#-----------------------------------------------------------------------------
# TESTING FOR FILTERING THE PHASES

# Testing for filter_phase
print('filter_phase')
start_testing()

    # Empty List; return empty list
expect(filter_phase(LOC0, FlightPhase.flight), [])
expect(filter_phase(LOC0, FlightPhase.landing), [])
expect(filter_phase(LOC0, FlightPhase.parking), [])
expect(filter_phase(LOC0, FlightPhase.takeoff), [])
expect(filter_phase(LOC0, FlightPhase.taxiing), [])

    # With Matching flight phase
        # flight
expect(filter_phase([Crash(FlightPhase.flight, 300, 10, 260, 30),
                    Crash(FlightPhase.takeoff, 20, 1, 10, 0),
                    Crash(FlightPhase.flight, 10, 5, 10, 5),
                    Crash(FlightPhase.takeoff, 300, 25, 300, 4)], FlightPhase.f

        # landing
```

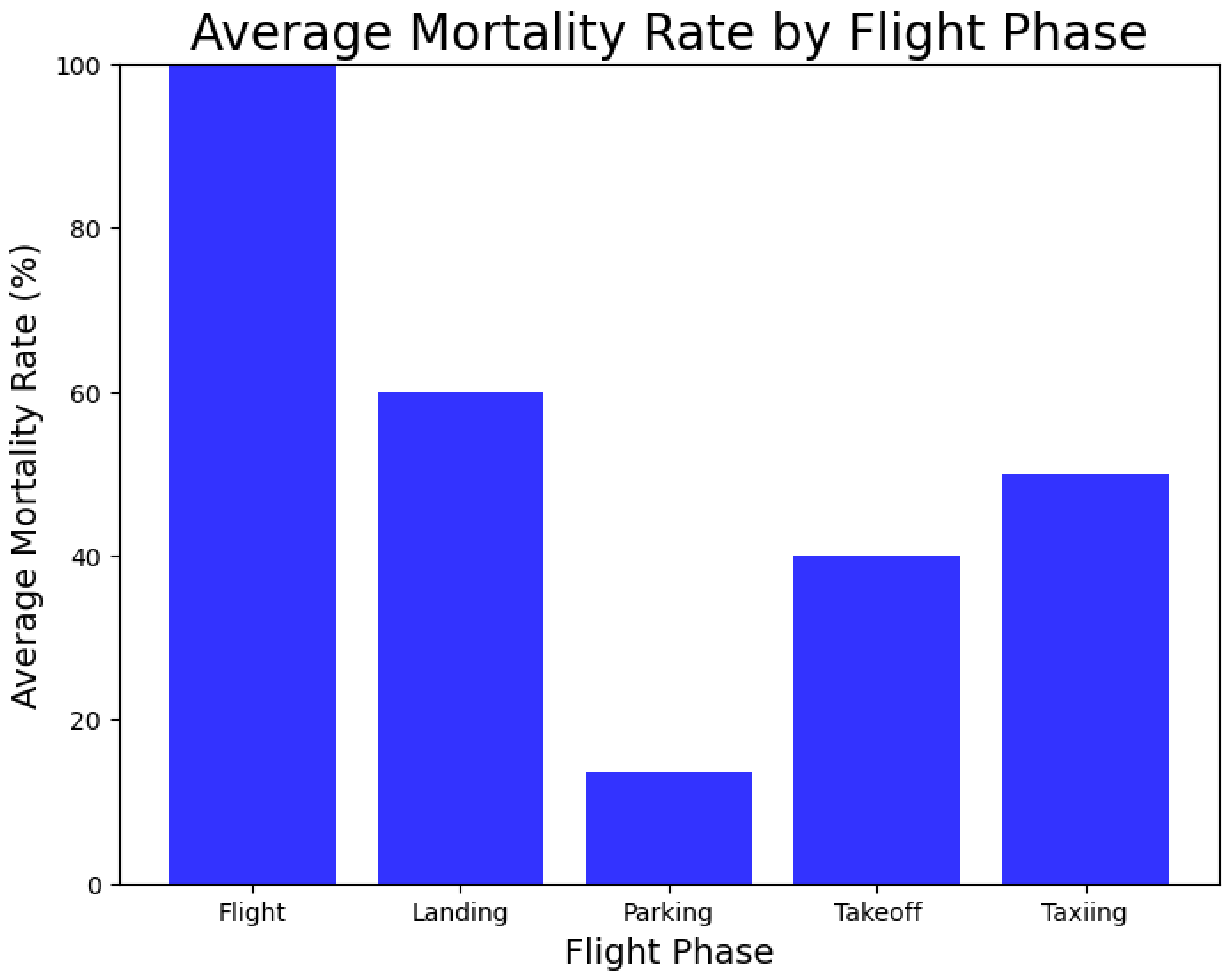
```
expect(filter_phase(LOC1, FlightPhase.landing), [C2])
        # parking
expect(filter_phase([Crash(FlightPhase.parking, 300, 10, 260, 30),
                    Crash(FlightPhase.takeoff, 20, 1, 10, 0),
                    Crash(FlightPhase.flight, 10, 5, 10, 5),
                    Crash(FlightPhase.takeoff, 300, 25, 300, 4)], FlightPhase.p
        # takeoff
expect(filter_phase(LOC1, FlightPhase.takeoff), [C1])
        # taxiing
expect(filter_phase(LOC1, FlightPhase.taxiing), [C3])
summary()




# Testing for in_flight_phase
print('in_flight_phase')
start_testing()

    # Matching flight phase
expect(in_flight_phase(C1, FlightPhase.takeoff), True)
expect(in_flight_phase(C2, FlightPhase.landing), True)
expect(in_flight_phase(C3, FlightPhase.taxiing), True)

    # Not Matching flight phase
expect(in_flight_phase(C2, FlightPhase.flight), False)
expect(in_flight_phase(C1, FlightPhase.landing), False)
expect(in_flight_phase(C2, FlightPhase.parking), False)
expect(in_flight_phase(C2, FlightPhase.takeoff), False)
expect(in_flight_phase(C2, FlightPhase.taxiing), False)
summary()



# Testing for same_phase
print('same_phase')
start_testing()

    # Are the same phase
expect(same_phase(FlightPhase.flight, FlightPhase.flight), True)
expect(same_phase(FlightPhase.landing, FlightPhase.landing), True)
expect(same_phase(FlightPhase.parking, FlightPhase.parking), True)
expect(same_phase(FlightPhase.takeoff, FlightPhase.takeoff), True)
expect(same_phase(FlightPhase.taxiing, FlightPhase.taxiing), True)

    # Are not the same phase
expect(same_phase(FlightPhase.landing, FlightPhase.flight), False)
expect(same_phase(FlightPhase.parking, FlightPhase.landing), False)
expect(same_phase(FlightPhase.takeoff, FlightPhase.parking), False)
expect(same_phase(FlightPhase.taxiing, FlightPhase.takeoff), False)
expect(same_phase(FlightPhase.flight, FlightPhase.taxiing), False)
summary()



#-----------------------------------------------------------------------------
# PART 2: TESTING FOR FINDING MORTALITY RATES
# Testing for get_mortality_rate
print('get_mortality_rate')
start_testing()

    # Full fatality crash
expect(get_mortality_rate(C1), 100.00)
expect(get_mortality_rate(Crash(FlightPhase.takeoff, 300, 30, 300, 30)), 100.0

    # Full fatality; no passengers on board
expect(get_mortality_rate(Crash(FlightPhase.taxiing, 0, 1, 0, 1)), 100.00)
expect(get_mortality_rate(Crash(FlightPhase.flight, 0, 150, 0, 150)), 100.00)

    # Full fatality; no crew on board
expect(get_mortality_rate(Crash(FlightPhase.parking, 2, 0, 2, 0)), 100.00)
expect(get_mortality_rate(Crash(FlightPhase.landing, 240, 0, 240, 0)), 100.00)

    # No fatalities
expect(get_mortality_rate(C2), 0.00)
expect(get_mortality_rate(Crash(FlightPhase.takeoff, 300, 30, 0, 0)), 0.00)

    # No fatalities, no passengers on board
expect(get_mortality_rate(Crash(FlightPhase.flight, 0, 10, 0, 0)), 0.00)
expect(get_mortality_rate(Crash(FlightPhase.taxiing, 0, 35, 0, 0)), 0.00)

    # No fatalities, no one on board (unmanned aerial vehicle (UAV)/ drone)
expect(get_mortality_rate(Crash(FlightPhase.flight, 0, 0, 0, 0)), 0.00)
expect(get_mortality_rate(Crash(FlightPhase.taxiing, 0, 0, 0, 0)), 0.00)

    # Mixed Fatalities
expect(get_mortality_rate(Crash(FlightPhase.flight, 420, 26, 69, 3)), round(((
expect(get_mortality_rate(Crash(FlightPhase.landing, 30, 2, 7, 0)), round(((7 +
expect(get_mortality_rate(Crash(FlightPhase.parking, 10, 3, 1, 2)), round(((1 +
expect(get_mortality_rate(Crash(FlightPhase.takeoff, 500, 42, 321, 20)), round(
expect(get_mortality_rate(Crash(FlightPhase.taxiing, 249, 34, 53, 20)), round((
summary()


# Testing for get_total_on_board
print('get_total_on_board')
start_testing()

    # On board more than 0
expect(get_total_on_board(C1), 15)
expect(get_total_on_board(C2), 2)

        # No Crew
expect(get_total_on_board(Crash(FlightPhase.parking, 10, 0, 0, 0)), 10)

        # No Passengers
expect(get_total_on_board(Crash(FlightPhase.parking, 0, 20, 0, 0)), 20)

    # 0 passengers and crew on board
expect(get_total_on_board(Crash(FlightPhase.parking, 0, 0, 0, 0)), 0)
summary()
```

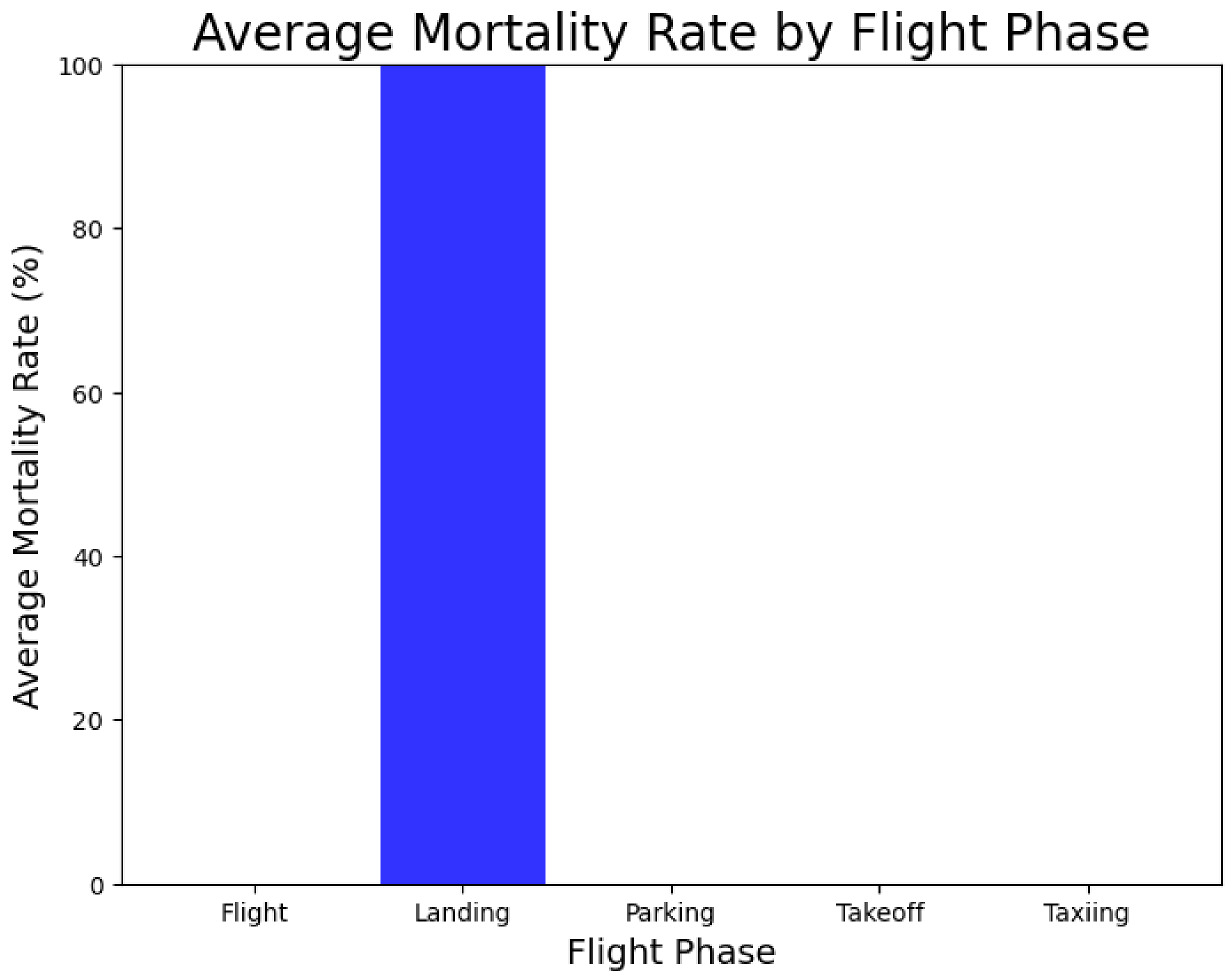
```
# Testing for get_total_fatalities
print('get_total_fatalities')
start_testing()

    # With Fatalities
expect(get_total_fatalities(C1), 15)
expect(get_total_fatalities(C3), 160)
expect(get_total_fatalities(Crash(FlightPhase.takeoff, 600, 40, 532, 24)), 556)

    # One Fatality
expect(get_total_fatalities(Crash(FlightPhase.takeoff, 240, 20, 0, 1)), 1)
expect(get_total_fatalities(Crash(FlightPhase.takeoff, 4, 1, 1, 0)), 1)

    # No Fatalities
expect(get_total_fatalities(C2), 0)
expect(get_total_fatalities(Crash(FlightPhase.takeoff, 302, 40, 0, 0)), 0)
summary()


# Testing for get_mortality_rates
print('get_mortality_rates')
start_testing()
    # Empty List
expect(get_mortality_rates(LOC0),[])
    # Non-Empty List
expect(get_mortality_rates(LOC1), [100.00, 0.00, 69.57])
expect(get_mortality_rates([C1, C2]), [100.00, 0.00])
    # All 0.00
expect(get_mortality_rates([Crash(FlightPhase.takeoff, 40, 5, 0, 0),
                            Crash(FlightPhase.takeoff, 302, 40, 0, 0),
                            Crash(FlightPhase.takeoff, 530, 36, 0, 0)]), [0.00,
    # All above 0.00
expect(get_mortality_rates([Crash(FlightPhase.takeoff, 40, 5, 3, 2),
                            Crash(FlightPhase.takeoff, 302, 40, 300, 0),
                            Crash(FlightPhase.takeoff, 530, 36, 31, 2)]), [11.1
summary()


#--------------------------------------------------------------------------------
# TESTING FOR FINDING AVERAGE MORTALITY RATES

# Testing for get_average
print('get_average')
start_testing()

expect(get_average([1500.00, 220.00]), (1500.00 + 220.00) / 2)
expect(get_average([5.5, 10.5, 6.5, 90.0]), 28.125)
expect(get_average([-5.5, -200.2, -234.4]), -146.7)

    # Zero average
expect(get_average([0, 0, 0, 0, 0, 0]), 0)
summary()



# Testing for get_sum
```

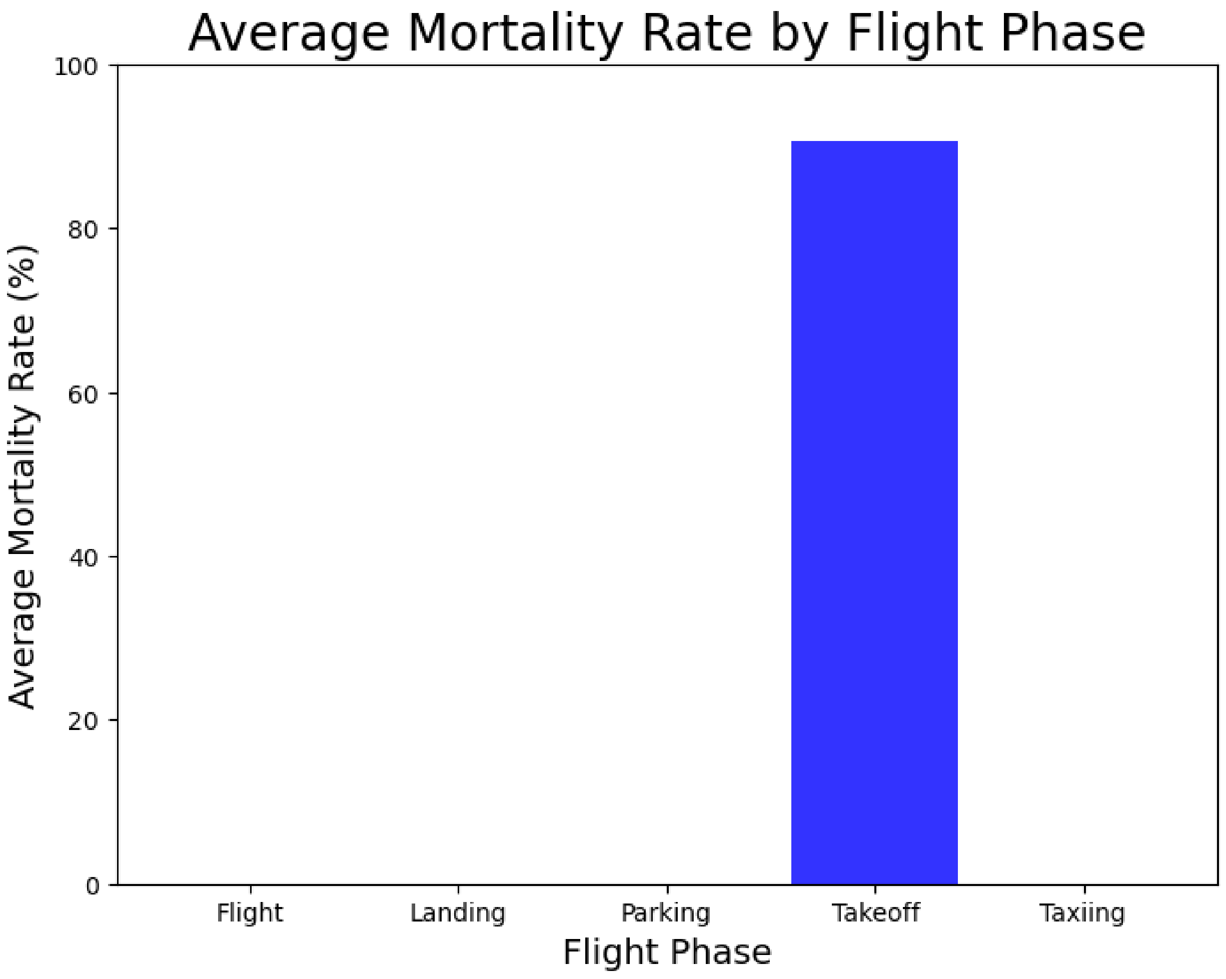
```
print('get_sum')
start_testing()

    # Sum
expect(get_sum([1.5, 2]), 3.50)
expect(get_sum([1500.00, 220.00]), (1500.00 + 220.00))
expect(get_sum([150.0, 0.00]), (150.0 + 0.00))
expect(get_sum([0.00, 547.53, 900.00]), (0 + 547.53 + 900.00))
        # With Negative numbers
expect(get_sum([(-2.50), 547.5]), ((-2.50) + (547.5)))

    # Zero Sum
expect(get_sum([0, 0, 0, 0, 0, 0]), 0)
expect(get_sum([10, -10, 10, -10]), 0)

summary()

#------------------------------------------------------------------------------
# TESTING FOR READ FUNCTIONS

# Testing for read
print('read')

start_testing()

expect(read('test_empty.csv'), [])
expect(read('test_plane_crash_one_line.csv'), [Crash(FlightPhase.landing, 3, 1,
expect(read('test_plane_crash_two_lines.csv'), [Crash(FlightPhase.takeoff, 0, 6
                                                Crash(FlightPhase.takeoff, 37,

expect(read('test_plane_crash_all_phase_types.csv'), [Crash(FlightPhase.takeof
                                                      Crash(FlightPhase.flight,
                                                      Crash(FlightPhase.landing
                                                      Crash(FlightPhase.taxiing
                                                      Crash(FlightPhase.parking

expect(read('test_plane_crash_multi_line_with_na.csv'), [Crash(FlightPhase.flig
                                                        Crash(FlightPhase.fligh
                                                        Crash(FlightPhase.landi
                                                        Crash(FlightPhase.landi
expect(read('test_plane_crash_all_na.csv'), [])
summary()




print('is_reliable')
# Testing: is_reliable()

start_testing()
    # No 'NA' in listed rows
expect(is_reliable(['1918-06-08','NA','Handley Page V/1500','Handley Page Aircr
                        'Cricklewood - Cricklewood','NA','1918','NA','Cricklewo
                        'Assembled at Cricklewood Airfield in May 1918, the air
                    'Technical failure']), True)
    # There is an 'NA' in one of the listed rows
expect(is_reliable(['1918-08-23','NA','Tellier T.3','Portuguese Air Force - Aer
                     'NA','NA','NA','NA','Cascais Estremadura - Lisbon District'
                     'Unknown']), False)
    # All listed rows are 'NA'
expect(is_reliable(['1918-08-23','NA','Tellier T.3','Portuguese Air Force - Aer
                     'NA','NA','NA','NA','Cascais Estremadura - Lisbon District'
                     'Unknown']), False)
summary()



print('parse_flight_phase')
# Testing: parse_flight_phase()
start_testing()

expect(parse_flight_phase('Takeoff (climb)'), FlightPhase.takeoff)
expect(parse_flight_phase('Flight'), FlightPhase.flight)
expect(parse_flight_phase('Landing (descent or approach)'), FlightPhase.landing
expect(parse_flight_phase('Taxiing'), FlightPhase.taxiing)
expect(parse_flight_phase('Parking'), FlightPhase.parking)

summary()
```

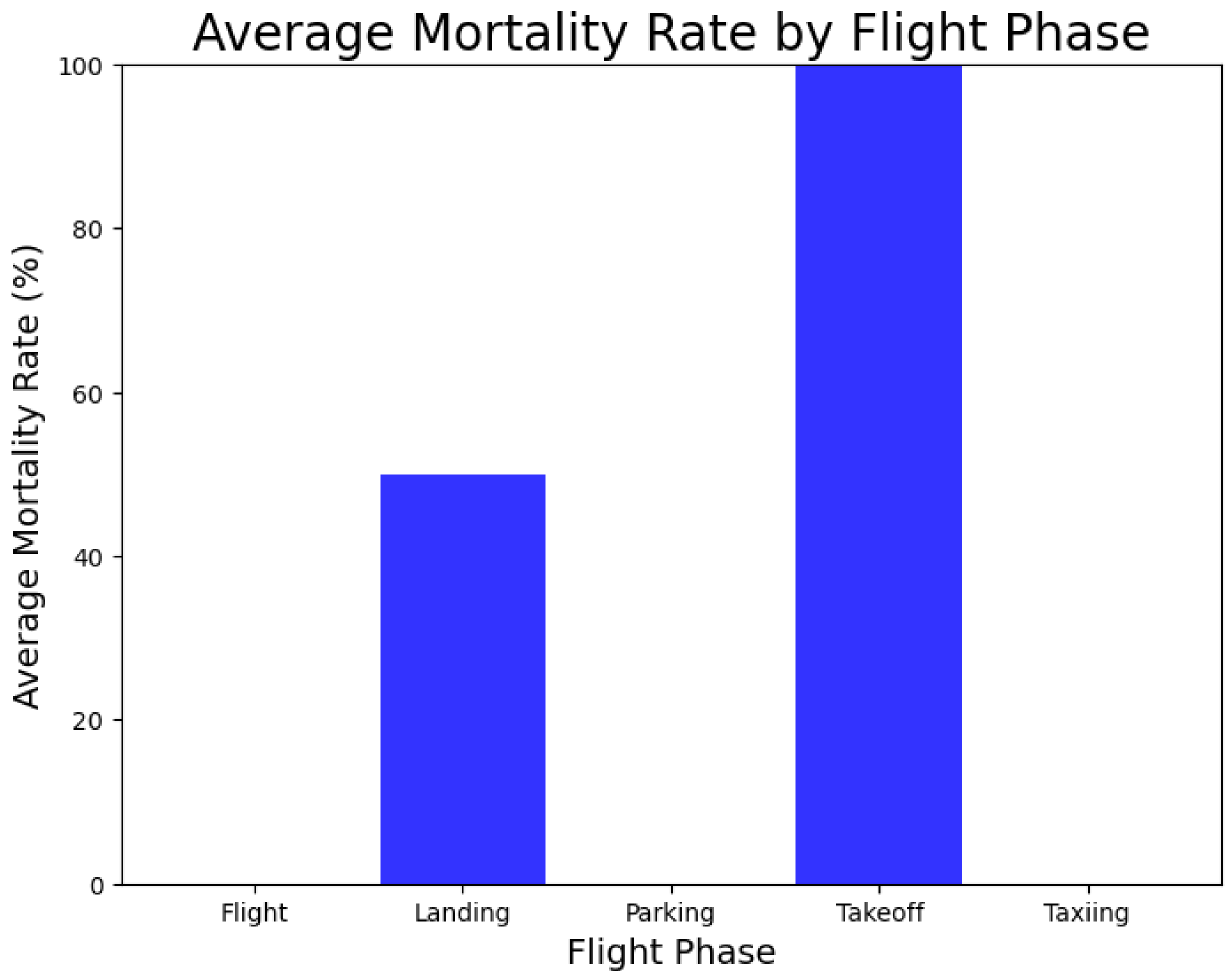
```
main
```

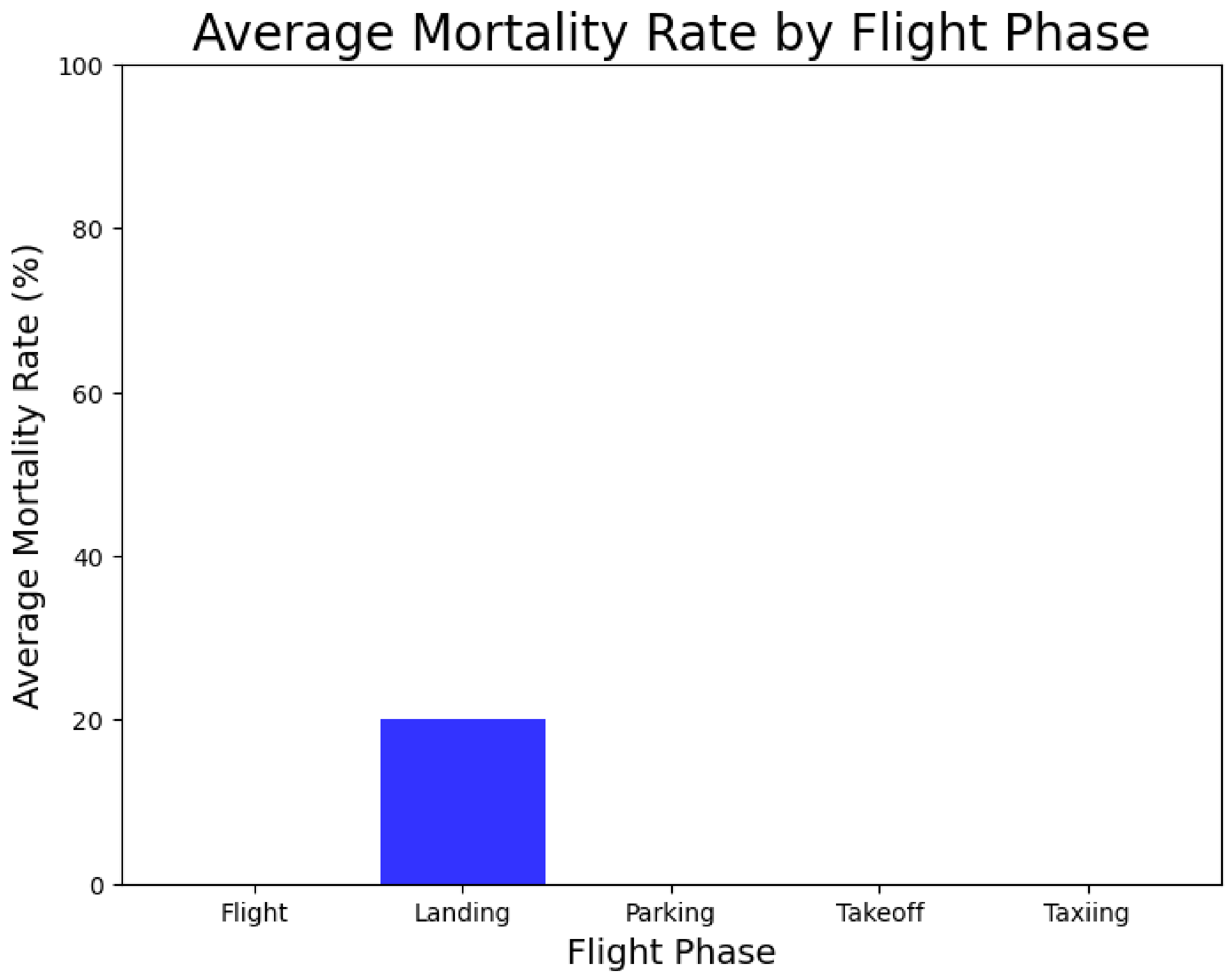


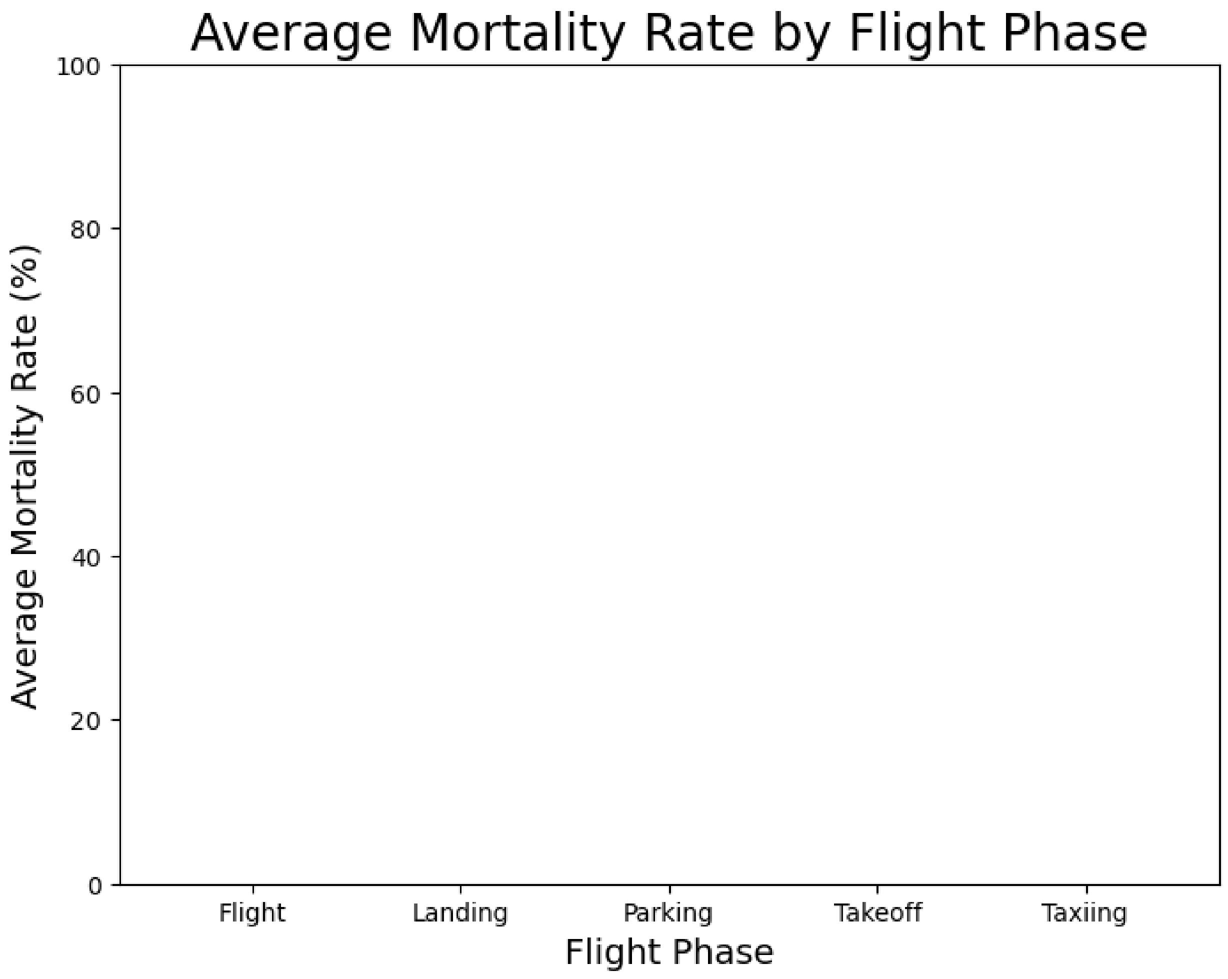

Average Mortality Rate by Flight Phase
Average Mortality Rate (%)
0
20
40
60
80
100
Flight
Landing
Parking
Takeoff
Taxiing
Flight Phase


Average Mortality Rate by Flight Phase
Average Mortality Rate (%)
100
80
60
40
20
0
Flight
Landing
Parking
Takeoff
Taxiing
Flight Phase

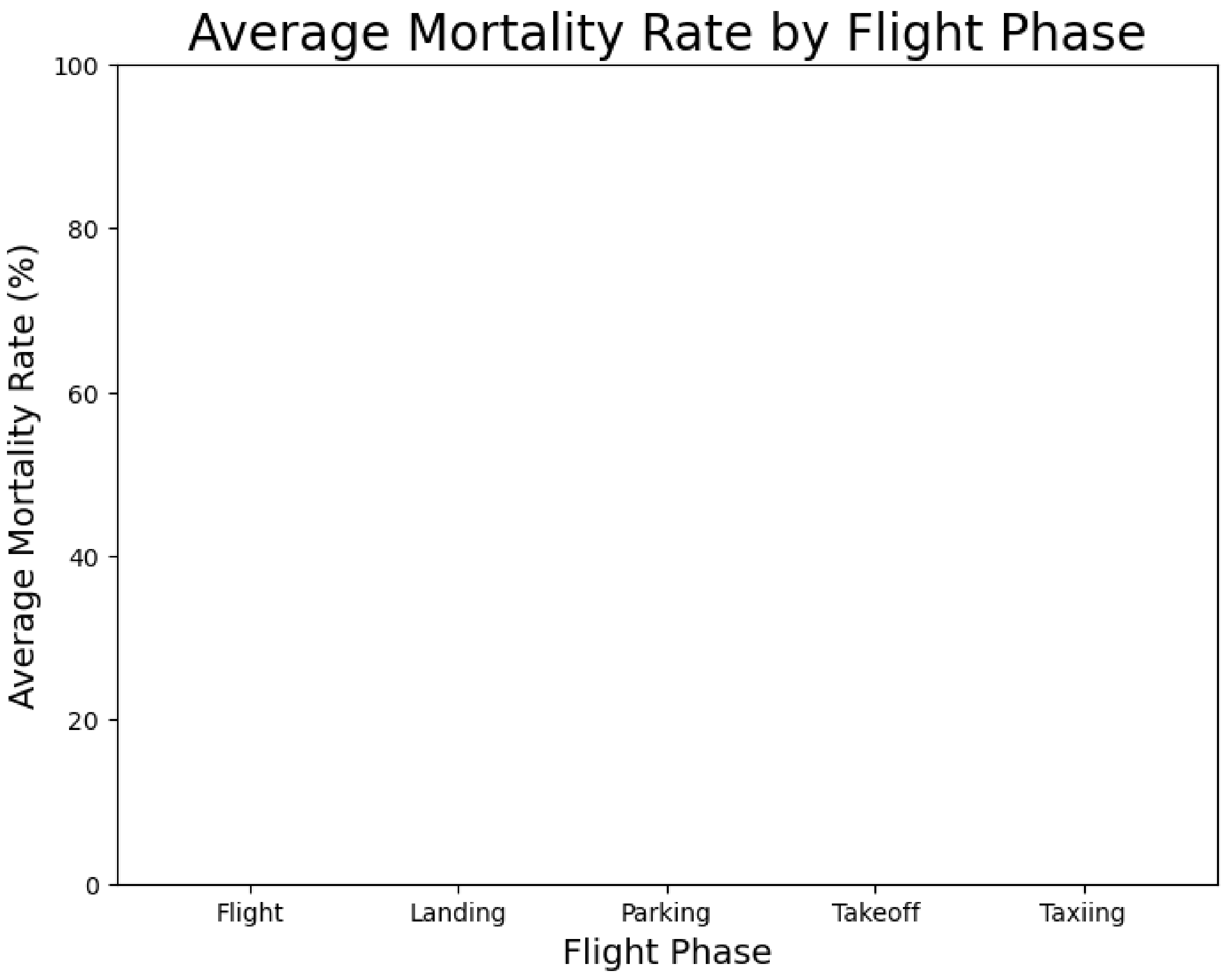

Average Mortality Rate by Flight Phase
Average Mortality Rate (%)
100
80
60
40
20
0
Flight
Landing
Parking
Takeoff
Taxiing
Flight Phase

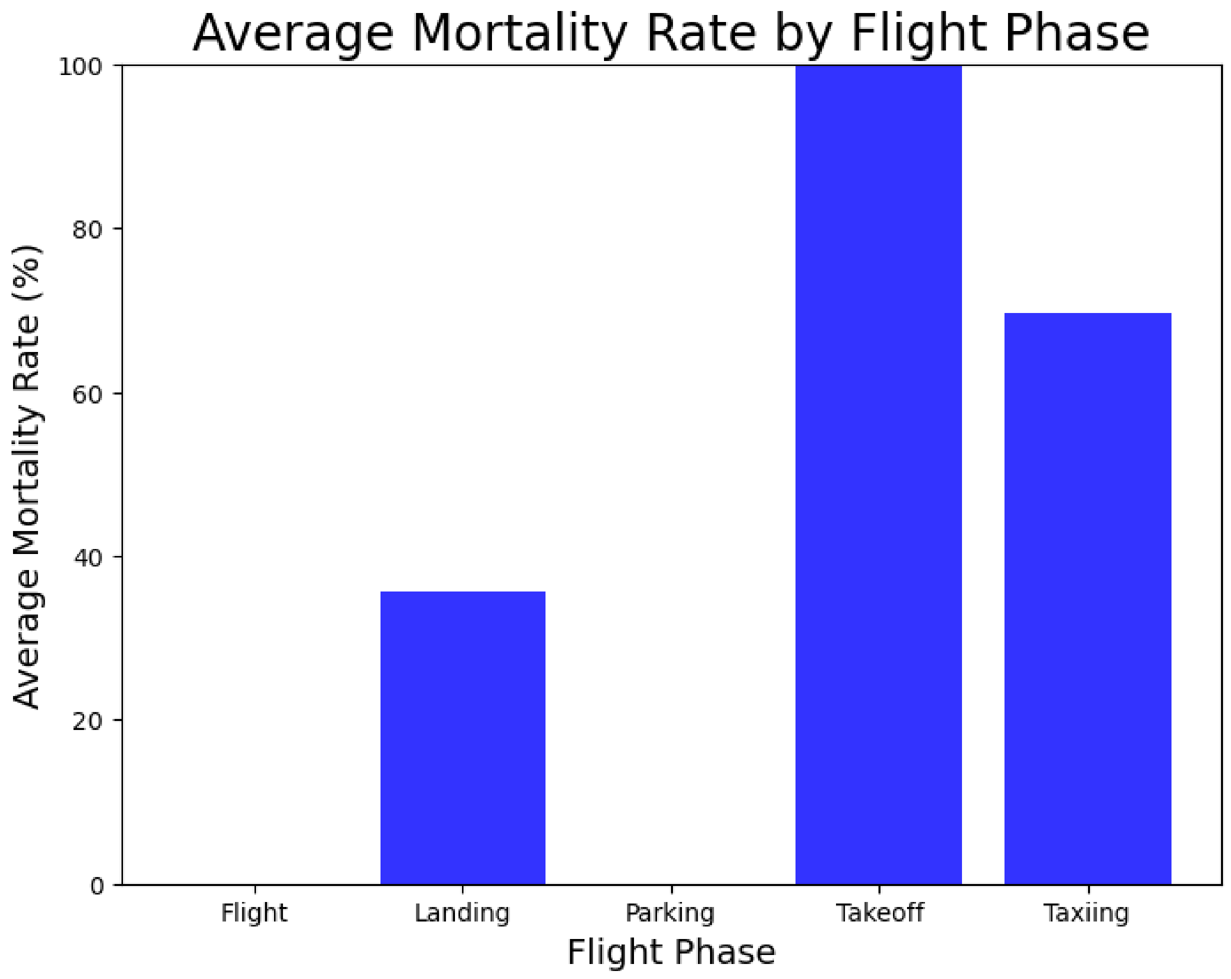

Average Mortality Rate by Flight Phase
Average Mortality Rate (%)
100
80
60
40
20
0
Flight
Landing
Parking
Takeoff
Taxiing
Flight Phase

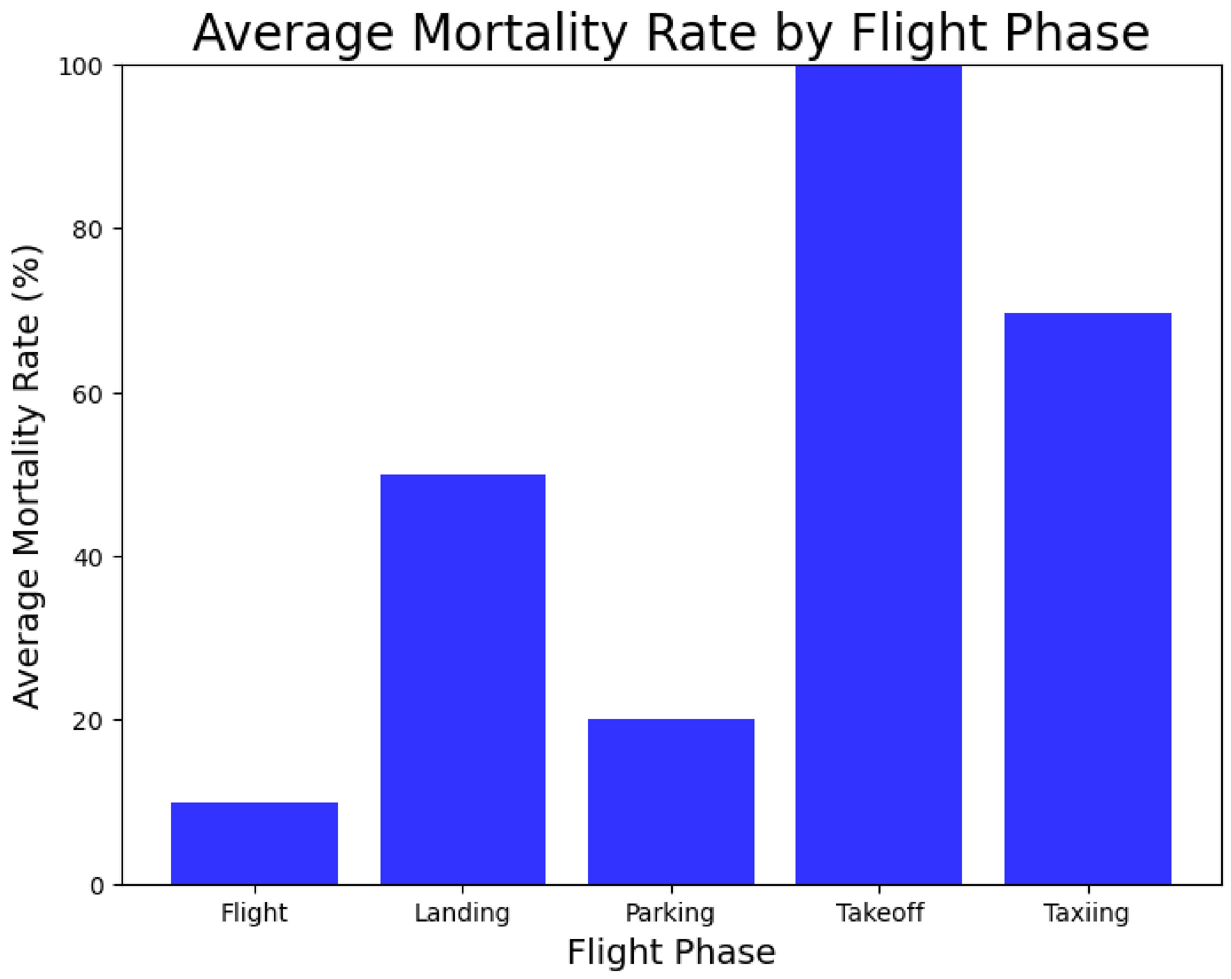

Average Mortality Rate by Flight Phase
Average Mortality Rate (%)
100
80
60
40
20
0
Flight
Landing
Parking
Takeoff
Taxiing
Flight Phase



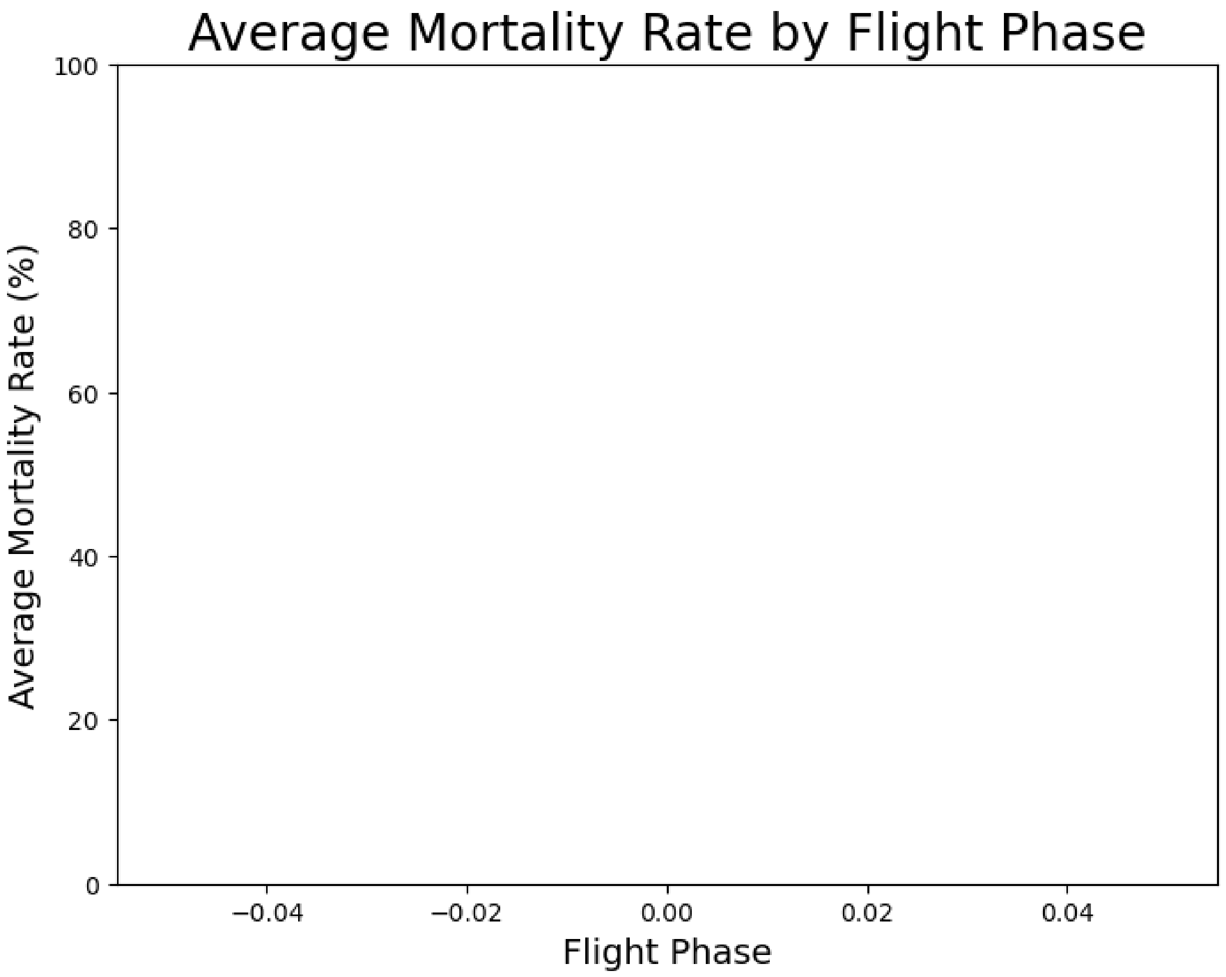
```
7 of 7 tests passed
analyze_crash
```

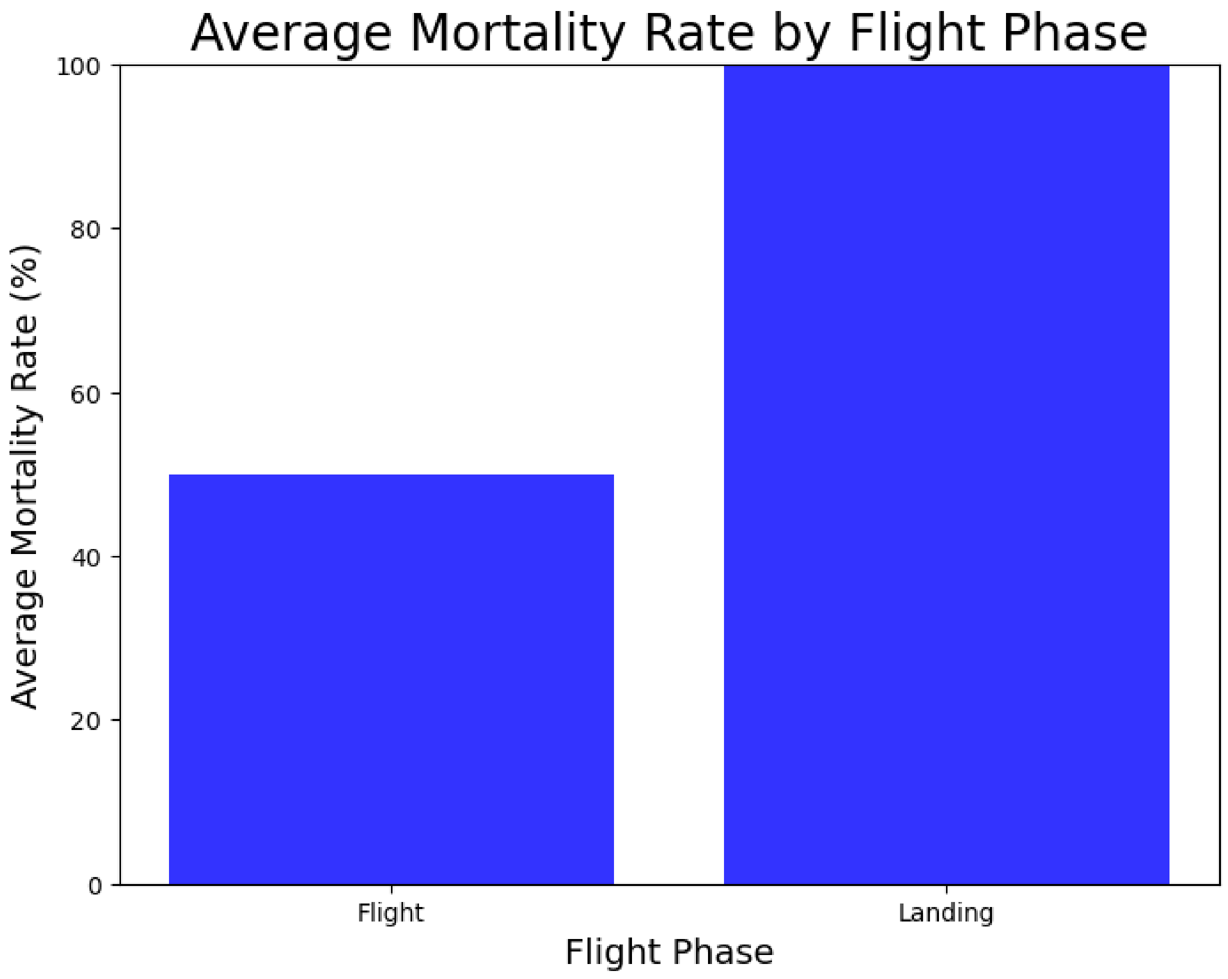

Average Mortality Rate by Flight Phase
Average Mortality Rate (%)
100
80
60
40
20
0
Flight
Landing
Parking
Takeoff
Taxiing
Flight Phase

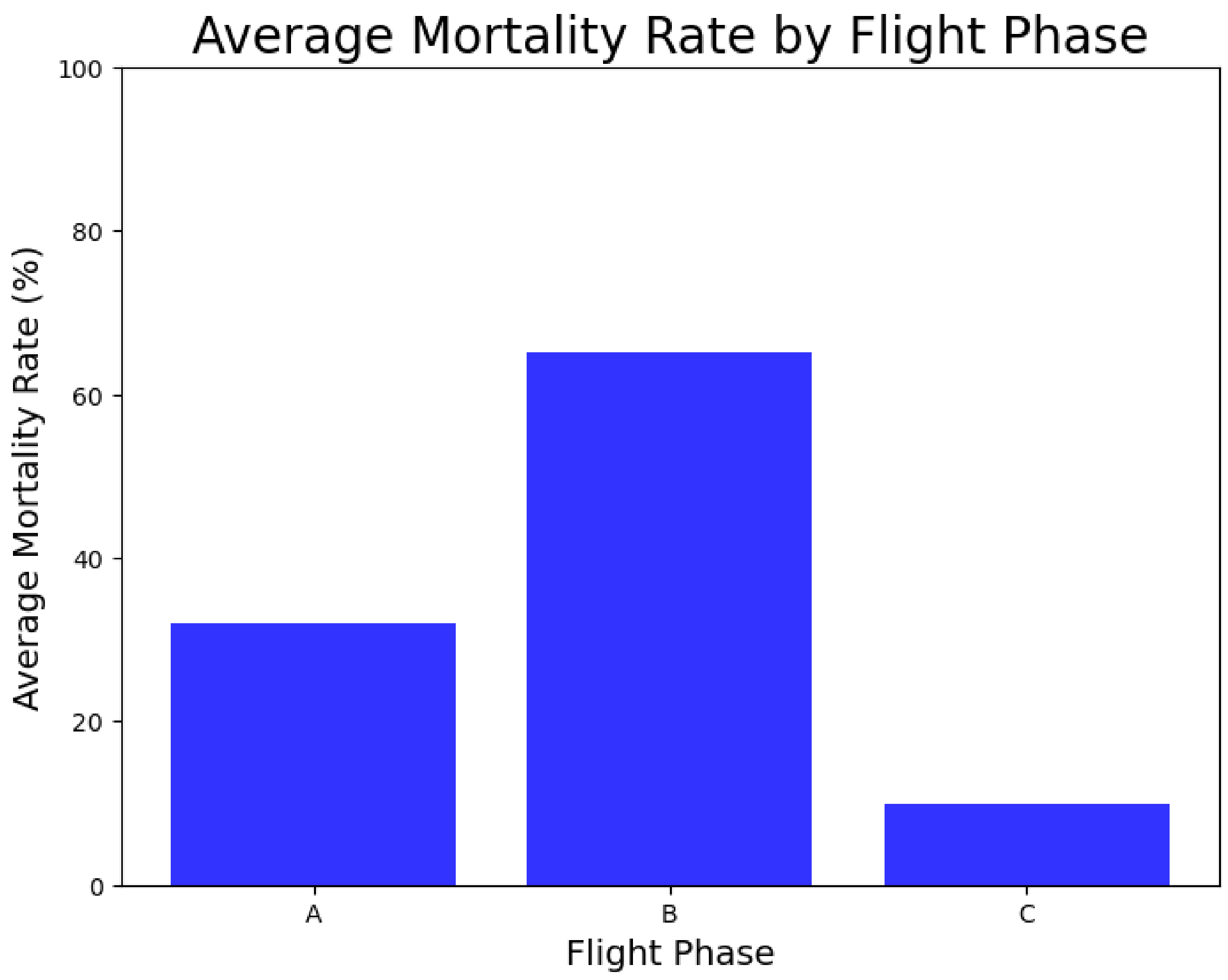

Average Mortality Rate by Flight Phase
Average Mortality Rate (%)
100
80
60
40
20
0
Flight
Landing
Parking
Takeoff
Taxiing
Flight Phase



```
3 of 3 tests passed
create_bar_chart
```


Average Mortality Rate by Flight Phase
Average Mortality Rate (%)
100
80
60
40
20
0
−0.04
−0.02
0.00
0.02
0.04
Flight Phase


Average Mortality Rate by Flight Phase
Average Mortality Rate (%)
0
20
40
60
80
100
Flight
Landing
Flight Phase

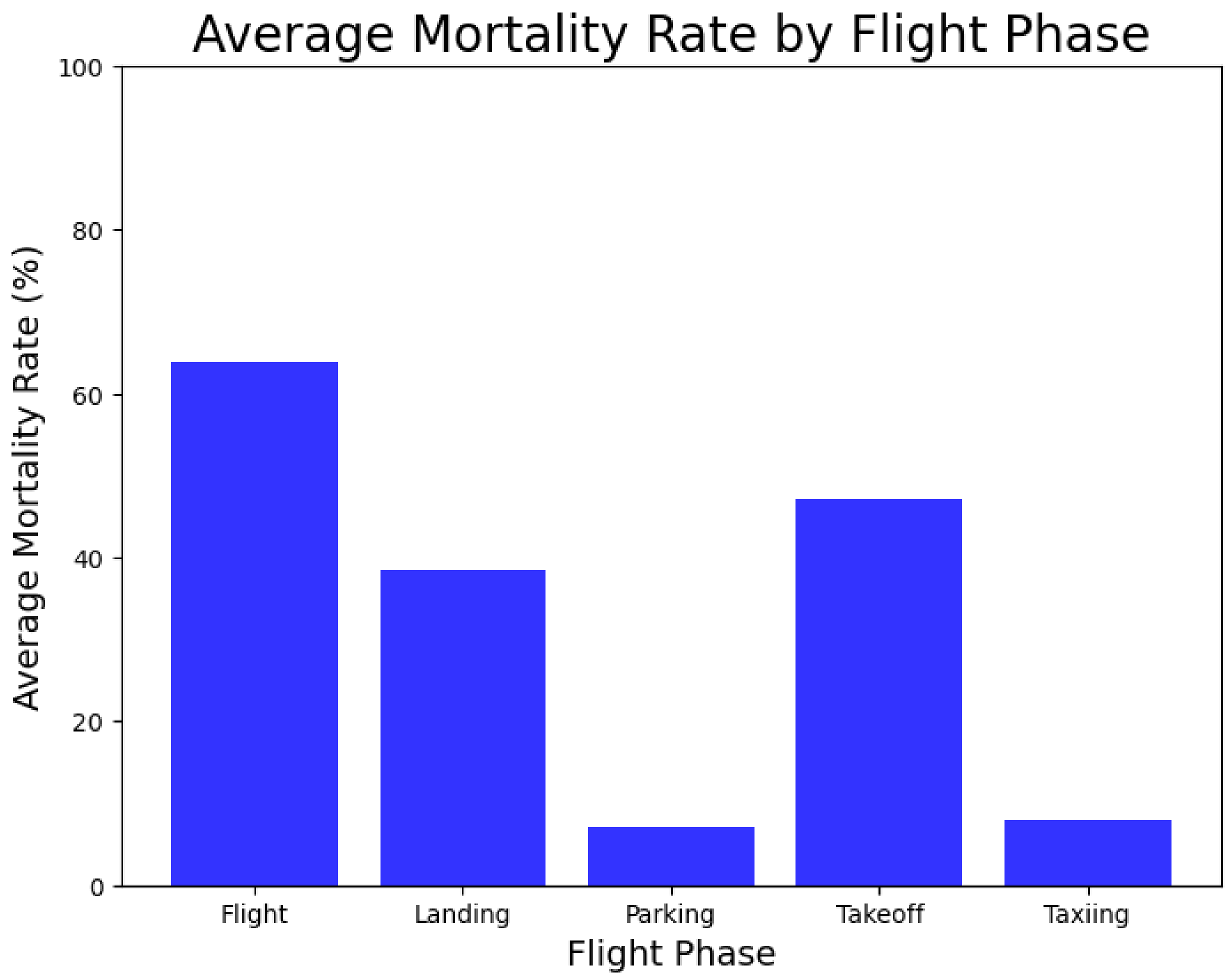


```
3 of 3 tests passed
filter_phase
10 of 10 tests passed
in_flight_phase
8 of 8 tests passed
same_phase
10 of 10 tests passed
get_mortality_rate
17 of 17 tests passed
get_total_on_board
5 of 5 tests passed
get_total_fatalities
7 of 7 tests passed
get_mortality_rates
5 of 5 tests passed
get_average
4 of 4 tests passed
get_sum
7 of 7 tests passed
read
6 of 6 tests passed
is_reliable
3 of 3 tests passed
parse_flight_phase
5 of 5 tests passed
```

Final Graph/Chart

Now that everything is working, you must call `main` on the intended information source in order to display the final graph/chart:

```
In [5]: main('Plane Crashes.csv')
```



```
In [6]: # Be sure to select ALL THE FILES YOU NEED (including csv's)
        # when you submit. As usual, you cannot edit this cell.
        # Instead, run this cell to start the submission process.
        from cs103 import submit

        COURSE = 101855
        ASSIGNMENT = 1291659 # Final submission

        submit(COURSE, ASSIGNMENT)

        # If your submission fails, SUBMIT by downloading your files and uploading them
        # to Canvas. You can learn how on the page "How to submit your Jupyter noteboo
        # on our Canvas site.
```

```
Valid(value=True, description='Token')
SelectMultiple(description='Files', index=(0,), layout=Layout(height='100%', w
idth='50%'), options=('project_f…
Button(description='submit', icon='check', style=ButtonStyle(), tooltip='submi
t')
```

Please double check your submission on Canvas to ensure that the right files (Jupyter file + CSVs) have been submitted and that the files do not contain unexpected errors.

You should always check your submission on Canvas. It is your responsibility to ensure that the correct file has been submitted for grading. Regrade or accomodation requests using reasoning such as "I didn't realize I submitted the wrong file"/"I didn't realize the submission didn't work"/"I didn't realize I didn't save before submitting so some of my work is missing" will not be considered.