Stat 151 Exam 1

Your Name

Instructions

- The exam is designed to assess both your fluency and your competency with the skills in Stat 151.
  - You may not finish within 2 hours – that is ok.
  - There is redundancy built in
- There are two sections on this exam.
  - Each section is broken down into a number of skills that should be attempted in order.
  - Attempt Section 1 before you start Section 2
  - Complete a section in one language before attempting it in the other.
  - Both sections are designed to be equally easy in R or Python.
- If you get stuck:
  - Two pre-written hints for each section/skill combination can be obtained for 30% of the value of that question each.
  - If the hints do not help, you can get a solution that will allow you to attempt the next skill, but you will not receive credit for the skill.

Allowed resources:

- notes
- homework
- Statistical Computing in R and Python (the course textbook)

Use of any other resources will result in a 0 on the exam and an academic misconduct report.

Your code is expected to run on my machine when I grade this exam. You will receive reduced credit for code which does not work correctly, whether or not it produces explicit errors.

1 Prime Factorization

In this problem, you will work towards building code that will decompose a number into its prime factors.

We'll start by installing a package that helps work with prime numbers… `primes` in R, `SymPy` in Python.

1.1 Skill: Install Packages

R

Write code to install the `primes` package in R. Run the code.

Python

Write code to install the `SymPy` package in Python. If you choose to install the package via the terminal, place your code in the bash chunk; otherwise, place your code in the python chunk.

Thinking Criticially

You must complete this question for either R or Python above.

1. Add `, eval = F` to the code chunk header to stop that code from running every time the document is executed.
2. What is another way to keep the code in the quarto document but prevent it from running?

1.2 Skill: Loading packages

Load the packages you just installed in R and Python using the chunks below.

R

Python

Hint: SymPy should be typed as `sympy` when loading the package.

1.3 Skill: Using prewritten functions

R

Use the `generate_primes(min, max)` function to generate all the primes between 1 and 47, inclusive. Store these numbers in a variable called `myprimes`.

Python

Use the `primerange(a, b)` function in `SymPy` to get all primes between 1 and 47, inclusive. Store these numbers in a variable called `myprimes`.

You may have to convert the output of `primerange` to a `list` by wrapping the function call in `list()`. It may be convenient to then convert your list into a numpy array using `np.array(list(...))`. If you choose to do this, you will also need to import the numpy library.

Thinking Critically

Do the function parameters you used in R and python differ? Why?

> **Reminder: Modular Division**
>
> Remember that modular division gives you the remainder when dividing a number by another number. `x %% 3` (R) or `x % 3` (Python) will give you the remainder when `x` is divided by 3.
> If `x` is an integer and the divisor is an integer, the result of modular division will also be an integer.

1.4 Skill: Indexing and Subsets

Using your `primes` variables (in both R and python), determine which primes are a factor of a second variable, `x`. You may want to test your code on values like `x=18`, `x=25`, `x = 34`, `x=43`, Your code should output only the prime factors of `x`.

R

Python

> **Reminder: Logs with different bases**
>
> The operation $\log_a(x) = b$ is defined the number b such that $a^b = x$ for fixed a and x. You can use this information to determine the maximum number of times a factor could be repeated when calculating the prime factorization of a number.
> You will need to load the `math` library in Python to access the `math.log(a, Base)` function. In R, the `log(x, base)` function is built in.

1.5 Skill: Writing Functions

In the previous section, you determined which primes were factors of a given number. Of course, it is possible to have a number which has multiple occurrences of the same prime factor.

In R and python, write a function, `prime_n(x, prime)`, which will determine how many times a single, pre-specified prime number `prime` can be evenly divided into a number `x`. You may want to calculate an upper bound for this possibility to help you in your search. Your function should take parameters `x` and `prime` and return the integer number of times `prime` occurs in the prime factorization of `x`.

R

Python

1.6 Skill: Data Frames, Loops

Putting the pieces together, use your function and your list of prime factors to determine the prime factorization of a given number. Store your factorization as a data frame with two columns: factor and power, where factor contains the factor and power contains the number of times that factor appears in the prime factorization.

For instance, your result for `x=18` should be a data frame that looks like this:

factor	power
2	1
3	2

Hint:

- R: `rbind(df, row)` will add `row` to the bottom of `df` if `df` is a data frame
- Python: `pandas.concat([df, row])` will add `row` to the bottom of `df` if `df` is a data frame.

Planning

Using the provided scratch paper (please put your name at the top), sketch a basic program flow map that shows how the code you've already written fits together to solve this problem. Identify any bits of logic you need to write to solve the problem.

My solution is sketched out on sheet _____

R

Python

1.7 Skill: String Operations

Take the data frame you created in the previous problem and write a `format_factorization(df)` function that will output the results of that data frame as a string, so that the data frame containing the prime factorization of 18 that is shown above would return "2^1 x 3^2".

Hint:

- Python: in a DataFrame, you can convert the whole column to a string using `df.colname.astype("str")` (replace df, colname with appropriate data frame name and column name)

R

Python

1.8 Skill: Control Statements

Take the code you wrote in the previous part and use it to create a function `prime_factorize` that will return the prime factorization of a number. If the number provided is a prime, your function should return " is prime" instead of returning that the factorization is ^1 (which is not as clear).

Planning

What modifications will you need to make to handle any number? e.g. what if the number is greater than 47?

What modifications will you need to make to handle prime numbers?

How can you use previously written code and functions to accomplish this task?

What additional code do you need to write?

R

Python

1.9 Skill: User-proofing your function

It is never safe to assume that your user knows what they are doing. Can you make your function from the previous part more robust by testing the user input to ensure that it conforms to your expectations?

Planning

What assumptions does your previous answer make about parameters?

What do you need to test to ensure those assumptions are met?

R

Python

2 Numerical Integration

This section will walk you through implementing numerical integration of a function.

2.1 Skill: Installing Packages

1. Write code to install the `SciPy` package in Python. It contains functions you will need for the remainder of this exam. If you choose to install the package via the terminal, place your code in the bash chunk; otherwise, place your code in the python chunk.

Python

```
```

2.2 Skill: Loading packages

Load the packages you just installed in Python using the chunk below. If you did not manage to install the package, write the code you think you should use to load the packages.

Hint: Scipy should be typed as `scipy` when loading the package. You will only need the `norm` function in the scipy package, so you may want to be selective as to how you load the package.

Python

```
```

Thinking Critically

There are multiple ways to load python packages. If I want to use the 'bar' function from the package 'foo', explain how you would load the package in order for the following code to be valid. If there is no valid way to load the package and use the function as listed, then state that.

1. `foo.bar()`
2. `bar()`
3. `f.bar()`
4. `bar.foo()`

2.3 Skill: Using Prewritten Functions

The `dnorm(x, mean = 0, sd = 1)` function provides the value of the normal probability distribution density function in R. The `rv = scipy.stats.norm(x, location = 0, scale = 1)` function defines a normal random variable, and then `rv.pdf()` can be used to get the probability density function in Python.

In R and Python, define a sequence of `x = -3, ..., 3` that has length 100. Calculate `y = f(x)`, where `f` is the normal PDF.

R

Python

Plot your data

Create a line or scatter plot showing the relationship between x and $y = f(x)$ in R or Python.

2.4 Skill: Mathematical Operations

Integrating the normal PDF to calculate $P(x \leq a)$ is extremely difficult to do by hand. For a long time before computers were common, statisticians used tables that contained the value of $P(x \leq a)$ for many different values of a. Luckily, with computers, we can avoid that!

The simplest method for numerical integration involves approximating $f(x)$ using a step function - essentially, evaluating $f(x)$ at evenly spaced x values, and then assuming that $f(x) \approx f(x_i)$ for $x \in [x_i, x_{i+1})$. Then, it is possible to turn $\int_a^b f(x)dx$ into $\sum_{x=x_0}^{x_n} f(x_i) \cdot (x_{i+1} - x_i)$

Write code to calculate the approximate integral of the normal distribution PDF from -3 to 3, **using the x and y you created previously.**

R

Python

Thinking Critically

Is your answer reasonable? Approximately what value would you expect when integrating the standard normal PDF from -3 to 3?

2.5 Skill: Writing Functions, Data Frames

Write a function, `norm_step` that takes arguments `a`, `b`, and `delta` and returns a data frame containing columns `x` and `y` that define the transition points (in `delta` increments) along the step function approximation to the normal pdf.

Hint:

- R: `seq(from, to, length.out = ...)` or `seq(from, to, by=...)`
- Python: `linspace(start, stop, num)` from the `numpy` package, or `arange(start, stop, step)` from the `numpy` package

Planning

Using the provided scratch paper (please put your name at the top), sketch a basic program flow map that shows how the code you've already written can be used to solve this problem. Identify any bits of logic you need to write to solve the problem.

My solution is sketched out on sheet _____

R

Python

2.6 Skill: Functions, Mathematical Operations

Using your `norm_step` function, define a function `norm_int` that calculates the integral from `a` to `b` using step size `step`. Your function should call your `norm_step` function and should return a single numerical value.

R

Python

2.7 Skill: Data Frames, Loops

Using your function, generate a table (data frame) showing $P(x < b)$ for each value of b between -3 and 3, at 0.1 increments. (e.g. $b = \{-3, -2.9, -2.8, ..., 0, 0.1, ..., 2.9, 3.0\}$)

You may assume that $a = -\infty$ or a value sufficiently negative that $P(x < a) \approx 0$ (-10 is probably good enough).

R

Python