ExamJune_2025_problem

June 11, 2025

1 Exam 16th of June 2025, 13.00-18.00 for the course 1MS041 (Introduction to Data Science / Introduktion till dataanalys)

1.1 Instructions:

1. Complete the problems by following instructions.
2. When done, submit this file with your solutions saved, following the instruction sheet.

This exam has 3 problems for a total of 40 points, to pass you need 20 points. The bonus will be added to the score of the exam and rounded afterwards.

1.2 Some general hints and information:

- Try to answer all questions even if you are uncertain.
- Comment your code, so that if you get the wrong answer I can understand how you thought this can give you some points even though the code does not run.
- Follow the instruction sheet rigorously.
- This exam is partially autograded, but your code and your free text answers are manually graded anonymously.
- If there are any questions, please ask the exam guards, they will escalate it to me if necessary.

1.3 Tips for free text answers

- Be VERY clear with your reasoning, there should be zero ambiguity in what you are referring to.
- If you want to include math, you can write LaTeX in the Markdown cells, for instance `$f(x)=x^2$` will be rendered as $f(x) = x^2$ and `$$f(x) = x^2$$` will become an equation line, as follows

$$f(x) = x^2$$

Another example is `$$f_{Y \mid X}(y,x) = P(Y = y \mid X = x) = \exp(\alpha \cdot x + \beta)$$` which renders as

$$f_{Y|X}(y, x) = P(Y = y \mid X = x) = \exp(\alpha \cdot x + \beta)$$

1.4 Finally some rules:

- You may not communicate with others during the exam, for example:
  - You cannot ask for help in Stack-Overflow or other such help forums during the Exam.
  - You may not communicate with AI's, for instance ChatGPT.
  - Your on-line and off-line activity is being monitored according to the examination rules.

1.5 Good luck!

```
[ ]: # Insert your anonymous exam ID as a string in the variable below
examID="XXX"
```

1.6 Exam vB, PROBLEM 1

Maximum Points = 14

In this problem you will do rejection sampling from complicated distributions, you will also be using your samples to compute certain integrals, a method known as Monte Carlo integration: (Keep in mind that choosing a good sampling distribution is often key to avoid too much rejection)

1. [4p] Fill in the remaining part of the function `problem1_rejection` in order to produce samples from the below distribution using rejection sampling: (Hint: F is the distribution function)

$$F[x] = \begin{cases} 0, & x \leq 0 \\ \frac{e^{x^2}-x^2-1}{e-2}, & 0 < x < 1 \\ 1, & x \geq 1 \end{cases}$$

2. [2p] Produce 100000 samples (**use fewer if it takes too long (more than 10 sec) and you cannot find a solution**) and put the answer in `problem1_samples` from the above distribution and plot the histogram together with the true density.
3. [2p] Use the above 100000 samples (`problem1_samples`) to approximately compute the integral

$$\int_0^1 \sin(x) \frac{2(e^{x^2}-1)x}{e-2} dx$$

and store the result in `problem1_integral`.

4. [2p] Use Hoeffdings inequality to produce a 95% confidence interval of the integral above and store the result as a tuple in the variable `problem1_interval`

5. [4p] Fill in the remaining part of the function `problem1_rejection_2` in order to produce samples from the below distribution using rejection sampling:

$$F[x] = \begin{cases} 0, & x \leq 0 \\ 20xe^{20-1/x}, & 0 < x < \frac{1}{20} \\ 1, & x \geq \frac{1}{20} \end{cases}$$

   Hint: this is tricky because if you choose the wrong sampling distribution you reject at least 9 times out of 10. You will get points based on how long your code takes to create a certain number of samples, if you choose the correct sampling distribution you can easily create 100000 samples within 2 seconds.

```
[ ]: # Part 1

def problem1_rejection(n_samples=1):
    # Distribution from part 1
```

```
    # write the code in this function to produce samples from the distribution␣
↪in the assignment
    # Make sure you choose a good sampling distribution to avoid unnecessary␣
↪rejections

    # Return a numpy array of length n_samples
    return XXX
```

[ ]:
```
# Part 2

problem1_samples = XXX
```

[ ]:
```
# Part 3

problem1_integral = XXX
```

[ ]:
```
# Part 4

problem1_interval = [XXX,XXX]
```

[ ]:
```
# Part 5

def problem1_rejection_2(n_samples=1):
    # Distribution from part 2
    # write the code in this function to produce samples from the distribution␣
↪in the assignment
    # Make sure you choose a good sampling distribution to avoid unnecessary␣
↪rejections

    # Return a numpy array of length n_samples
    return XXX
```

Local Test for Exam vB, PROBLEM 1 Evaluate cell below to make sure your answer is valid. You **should not** modify anything in the cell below when evaluating it to do a local test of your solution. You may need to include and evaluate code snippets from lecture notebooks in cells above to make the local test work correctly sometimes (see error messages for clues). This is meant to help you become efficient at recalling materials covered in lectures that relate to this problem. Such local tests will generally not be available in the exam.

[ ]:
```
# This cell is just to check that you got the correct formats of your answer
import numpy as np
try:
    assert(isinstance(problem1_rejection(10), np.ndarray))
except:
    print("Try again. You should return a numpy array from problem1_rejection")
```

```
else:
    print("Good, your problem1_rejection returns a numpy array")

try:
    assert(isinstance(problem1_samples, np.ndarray))
except:
    print("Try again. your problem1_samples is not a numpy array")
else:
    print("Good, your problem1_samples is a numpy array")

try:
    assert(isinstance(problem1_integral, float))
except:
    print("Try again. your problem1_integral is not a float")
else:
    print("Good, your problem1_integral is a float")

try:
    assert(isinstance(problem1_interval, list) or isinstance(problem1_interval,␣
 ↪tuple)) , "problem1_interval not a tuple or list"
    assert(len(problem1_interval) == 2) , "problem1_interval does not have␣
 ↪length 2, it should have a lower bound and an upper bound"
except Exception as e:
    print(e)
else:
    print("Good, your problem1_interval is a tuple or list of length 2")

try:
    assert(isinstance(problem1_rejection_2(10), np.ndarray))
except:
    print("Try again. You should return a numpy array from problem1_rejection_2")
else:
    print("Good, your problem1_rejection_2 returns a numpy array")
```

1.7 Exam vB, PROBLEM 2

Maximum Points = 14

In this problem we have data consisting of user behavior on a website. The pages of the website are just numbers in the dataset $0, 1, 2, \ldots$ and each row consists of a user, a source and a destination page. This signifies that the user was on the source page and clicked a link leading them to the destination page. The goal is to improve the user experience by decreasing load time of the next page visited, as such we need a good estimate for the next site likely to be visited. We will model this using a homogeneous Markov chain, each row in the data-file then corresponds to a single realization of a transition.

1. [3p] Load the data in the file `data/websites.csv` and construct a matrix of size `n_pages`

`x n_pages` which is the maximum likelihood estimate of the true transition matrix for the Markov chain. Here the ordering of the states are exactly the ones in the data-file, that is page 0 has index 0 in the matrix.

2. [4p] A page loads in Exp(3) (Exponentially distributed with mean 1/3) seconds if not preloaded and loads with Exp(20) (Exponentially distributed with mean 1/20) seconds if preloaded and we only preload the most likely next site. Given that we start in page 8 simulate 10000 load times from page 1 (that is, only a single step), store the result in the variable indicated in the cell. Repeat the experiment but this time preload the two most likely pages and store the result in the indicated variable.
3. [3p] Compare the average (empirical) load time from part 2 with the theoretical one of no pre-loading. Does the load time improve, how did you come to this conclusion? (Explain in the free text field).
4. [4p] Calculate the stationary distribution of the Markov chain and calculate the expected load time with respect to the stationary distribution.

[ ]:
```
# Part 1: 3 points

# Load the data from the file data/websites.csv and estimate the transition␣
 ↪matrix of the Markov chain
# Store the estimated transition matrix in the variable␣
 ↪problem2_transition_matrix below
problem2_transition_matrix = XXX # A numpy array of shape (problem2_n_states,␣
 ↪problem2_n_states)
# Store the number of states in the variable problem2_n_states below
problem2_n_states = XXX # An integer
```

[ ]:
```
# Part 2: 4 points

# Simulate the website load times for the next page of 10000 users that are␣
 ↪currently on page 8 (recall indexing starts at 0) when we only preload the␣
 ↪most likely page.
# Store the simulated page load times in the variable␣
 ↪problem2_page_load_times_top below
problem2_page_load_times_top = XXX # A numpy array of shape (10000,)

# Repeat the simulation of load times for the next page of 10000 users that are␣
 ↪currently on page 8 when we preload the two most likely pages.
# Store the simulated page load times in the variable␣
 ↪problem2_page_load_times_two below
problem2_page_load_times_two = XXX # A numpy array of shape (10000,)
```

[ ]:
```
# Part 3: 3 points

# Calculate the true expected load time for loading a page without pre-loading␣
 ↪the next page and store it in the variable below
problem2_avg = XXX # A float
```

```
# Is the average load time for loading a page without pre-loading the next page␣
 ↪larger than the average load time for loading a page after pre-loading the␣
 ↪next most likely page?
problem2_comparison = XXX # True / False
```

1.8 Free text answer

Put the explanation for **part 3** of how you made the decision about `problem2_comparison` below this line in this **cell**. In order to enter edit mode you can doubleclick this cell or select it and press enter.

[ ]:
```
# Part 4: 4 points

# Begin by calculating the stationary distribution of the Markov chain and store␣
 ↪it in the variable below
# WARNING: Since the transition matrix is not symmetric, numpy might make the␣
 ↪output of the eigenvectors complex, you can use np.real() to get the real part␣
 ↪of the eigenvectors
# Store the stationary distribution in the variable below called␣
 ↪problem2_stationary_distribution
problem2_stationary_distribution = XXX # A numpy array of shape␣
 ↪(problem2_n_states,)

# Now use the above stationary distribution to calculate the average load time␣
 ↪for loading a page after pre-loading the next most likely page according to␣
 ↪the stationary distribution
# Store the average load time in the variable below
problem2_avg_stationary = XXX # A float
```

Local Test for Exam vB, PROBLEM 2 Evaluate cell below to make sure your answer is valid. You **should not** modify anything in the cell below when evaluating it to do a local test of your solution. You may need to include and evaluate code snippets from lecture notebooks in cells above to make the local test work correctly sometimes (see error messages for clues). This is meant to help you become efficient at recalling materials covered in lectures that relate to this problem. Such local tests will generally not be available in the exam.

[ ]:
```
# This cell is just to check that you got the correct formats of your answer
import numpy as np
try:
    assert isinstance(problem2_transition_matrix, np.ndarray)
except:
    print("Try again. your problem2_transition_matrix is not a numpy array")
else:
    print("Good, your problem2_transition_matrix is a numpy array")
```

```
try:
    assert isinstance(problem2_n_states, int)
except:
    print("Try again. your problem2_n_states is not an integer")
else:
    print("Good, your problem2_n_states is an integer")

try:
    assert problem2_transition_matrix.shape == (problem2_n_states,␣
 ↪problem2_n_states)
except:
    print("Try again. your problem2_transition_matrix does not have the correct␣
 ↪shape")
else:
    print("Good, your problem2_transition_matrix has the correct shape")

try:
    assert isinstance(problem2_page_load_times_top, np.ndarray),␣
 ↪"problem2_page_load_times_top is not a numpy array"
    assert problem2_page_load_times_top.shape == (10000,),␣
 ↪"problem2_page_load_times_top does not have the correct shape"
except Exception as e:
    print(e)
else:
    print("Good, your problem2_page_load_times_top is a numpy array of shape␣
 ↪(10000,)")

try:
    assert isinstance(problem2_page_load_times_two, np.ndarray),␣
 ↪"problem2_page_load_times_two is not a numpy array"
    assert problem2_page_load_times_two.shape == (10000,),␣
 ↪"problem2_page_load_times_two does not have the correct shape"
except Exception as e:
    print(e)
else:
    print("Good, your problem2_page_load_times_two is a numpy array of shape␣
 ↪(10000,)")

try:
    assert isinstance(problem2_avg, float), "problem2_avg is not a float"
except Exception as e:
    print(e)
else:
    print("Good, your problem2_avg is a float")
try:
```

```
    assert isinstance(problem2_comparison, bool), "problem2_comparison is not a␣
 ↪boolean"
except Exception as e:
    print(e)
else:
    print("Good, your problem2_comparison is a boolean")
try:
    assert isinstance(problem2_stationary_distribution, np.ndarray),␣
 ↪"problem2_stationary_distribution is not a numpy array"
    assert problem2_stationary_distribution.shape == (problem2_n_states,),␣
 ↪"problem2_stationary_distribution does not have the correct shape"
except Exception as e:
    print(e)
else:
    print("Good, your problem2_stationary_distribution is a numpy array of shape␣
 ↪(problem2_n_states,)")
try:
    assert isinstance(problem2_avg_stationary, float), "problem2_avg_stationary␣
 ↪is not a float"
except Exception as e:
    print(e)
else:
    print("Good, your problem2_avg_stationary is a float")
```

1.9 Exam vB, PROBLEM 3

Maximum Points = 12

In this problem we are interested in fraud detection in an e-commerce system. In this problem we are given the outputs of a classifier that predicts the probabilities of fraud, your goal is to explore the threshold choice as in individual assignment 4. The costs associated with the predictions are:

- **True Positive (TP)**: Detecting fraud and blocking the transaction costs the company 100 (manual review etc.)
- **True Negative (TN)**: Allowing a legitimate transaction has no cost.
- **False Positive (FP)**: Incorrectly classifying a legitimate transaction as fraudulent costs 120 (customer dissatisfaction plus operational expenses for reversing the decision).
- **False Negative (FN)**: Missing a fraudulent transaction costs the company 600 (e.g., fraud loss plus potential reputational damage or penalties).

The code cells contain more detailed instructions, THE FIRST CODE CELL INITIALIZES YOUR VARIABLES

1. [3p] Complete filling the function `cost` to compute the average cost of a prediction model under a certain prediction threshold. Plot the cost as a function of the threshold (using the validation data provided in the first code cell of this problem), between 0 and 1 with 0.01 increments.

2. [2.5p] Find the threshold that minimizes the cost and calculate the cost at that threshold on the validation data. Also calculate the precision and recall at the optimal threshold on the validation data on class 1 and 0.
3. [2.5p] Repeat step 2, but this time find the best threshold to minimize the $0-1$ loss. Calculate the difference in cost between the threshold found in part 2 with the one just found in part 3.
4. [4p] Provide a confidence interval around the optimal cost (with 95% confidence) applied to the test data and explain all the assumption you made.

[ ]:
```
# RUN THIS CELL TO GET THE DATA

# We start by loading the data

import pandas as pd

PROBLEM3_DF = pd.read_csv('data/fraud.csv')
Y = PROBLEM3_DF['Class'].values
X = PROBLEM3_DF[['V%d' % i for i in range(1,5)]+['Amount']].values

# We will split the data into training, testing and validation sets
from Utils import train_test_validation
PROBLEM3_X_train, PROBLEM3_X_test, PROBLEM3_X_val, PROBLEM3_y_train,␣
↪PROBLEM3_y_test, PROBLEM3_y_val =␣
↪train_test_validation(X,Y,shuffle=True,random_state=1)

# From this we will train a logistic regression model
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
lr.fit(PROBLEM3_X_train,PROBLEM3_y_train)

# THE FOLLOWING CODE WILL PRODUCE THE ARRAYS YOU NEED FOR THE PROBLEM

PROBLEM3_y_pred_proba_val = lr.predict_proba(PROBLEM3_X_val)[:,1]
PROBLEM3_y_true_val = PROBLEM3_y_val

PROBLEM3_y_pred_proba_test = lr.predict_proba(PROBLEM3_X_test)[:,1]
PROBLEM3_y_true_test = PROBLEM3_y_test
```

[ ]:
```
# Part 1: 3 points
# Implement the following function that calculates the cost of a binary␣
↪classifier according to
# the specification in the problem statement
# See the comments inside the function for details of the parameters
def cost(y_true,y_predict_proba,threshold):
    # y_true is a numpy array of shape (n_samples,) with binary labels
    # y_predict_proba is a numpy array of shape (n_samples,) with predicted␣
↪probabilities
    # threshold is a float between 0 and 1
```

```
    # When returning the cost, you should return the average cost per sample
    # thus it should be a value

    return XXX # A float


# Provide the code below to plot the cost as a function of the threshold
# using the validation data, specifically the arrays PROBLEM3_y_true_val and␣
 ↪PROBLEM3_y_pred_proba_val.
# The plot should be between 0 and 1 with 0.01 increments
# The y-axis should be the cost and the x-axis should be the threshold
```

[ ]:
```
# Part 2: 2.5 points

# Use the cost function you just implemented above to find the threshold that␣
 ↪minimizes the cost
# using the validation data, specifically the arrays PROBLEM3_y_true_val and␣
 ↪PROBLEM3_y_pred_proba_val.
# Store the threshold in the variable below
problem3_threshold = XXX # A float between 0 and 1

# Now calculate the cost of the classifier using the validation data and the␣
 ↪threshold you just found
# using the validation data, specifically the arrays PROBLEM3_y_true_val and␣
 ↪PROBLEM3_y_pred_proba_val.
# Store the cost in the variable below
problem3_cost_val = XXX # A float

# Using the threshold you just found, calculate the predicted labels of the␣
 ↪classifier on the validation data
# put the predicted labels in the variable below
problem3_y_pred_val = XXX # A numpy array of shape (n_samples,) with values 0 or␣
 ↪1

# Calculate the precision and recall of the classifier of class 1 using the␣
 ↪threshold you just found
# using the validation data, specifically the arrays PROBLEM3_y_true_val and␣
 ↪PROBLEM3_y_pred_proba_val.

problem3_precision_1 = XXX # A float between 0 and 1
problem3_recall_1 = XXX # A float between 0 and 1

# Calculate the precision and recall of the classifier of class 0 using the␣
 ↪threshold you just found
```

```
# using the validation data, specifically the arrays PROBLEM3_y_true_val and␣
 ↪PROBLEM3_y_pred_proba_val.

problem3_precision_0 = XXX # A float between 0 and 1
problem3_recall_0 = XXX # A float between 0 and 1
```

[ ]:
```
# Part 3: 2.5 points

# Find the threshold that minimizes the $0-1$ loss using the validation data
# specifically the arrays PROBLEM3_y_true_val and PROBLEM3_y_pred_proba_val.
# Store the threshold in the variable below

problem3_threshold_01 = XXX # A float between 0 and 1

# Now calculate the difference in cost (using the cost function you implemented␣
 ↪in step 1) between the optimal one chosen in part 2 and the one chosen in part␣
 ↪3 by taking the cost with the threshold found in part 3 and subtracting the␣
 ↪cost with the threshold found in part 2 to get a positive value
problem3_cost_difference = XXX # A float
```

[ ]:
```
# Part 4: 4 points

# Using the threshold problem3_threshold use Hoeffdings inequality to provide a␣
 ↪confidence interval
# for the cost of the classifier with 95 % confidence using the test data.
# Specifically the arrays PROBLEM3_y_true_test and PROBLEM3_y_pred_proba_test.
# Store the lower and upper bounds of the confidence interval in the variables␣
 ↪below

problem3_lower_bound = XXX # A float
problem3_upper_bound = XXX # A float
```

1.10 Free text answer

Put your explanation for part 4 below this line in this **cell**. Doubleclick to enter edit mode as before.

Local Test for Exam vB, PROBLEM 3 Evaluate cell below to make sure your answer is valid. You **should not** modify anything in the cell below when evaluating it to do a local test of your solution. You may need to include and evaluate code snippets from lecture notebooks in cells above to make the local test work correctly sometimes (see error messages for clues). This is meant to help you become efficient at recalling materials covered in lectures that relate to this problem. Such local tests will generally not be available in the exam.

[ ]:
```
# This cell is just to check that you got the correct formats of your answer
import numpy as np
try:
```

```
    assert callable(cost), "cost is not a function"
except:
    print("Try again. your cost is not a function")
else:
    print("Good, your cost is a function")
try:
    assert isinstance(PROBLEM3_y_pred_proba_val, np.ndarray),␣
 ↪"PROBLEM3_y_pred_proba_val is not a numpy array"
    assert PROBLEM3_y_pred_proba_val.shape == (len(PROBLEM3_y_true_val),),␣
 ↪"PROBLEM3_y_pred_proba_val does not have the correct shape"
except Exception as e:
    print(e)
else:
    print("Good, your PROBLEM3_y_pred_proba_val is a numpy array of shape␣
 ↪(len(PROBLEM3_y_true_val),)")
try:
    assert isinstance(PROBLEM3_y_true_val, np.ndarray), "PROBLEM3_y_true_val is␣
 ↪not a numpy array"
    assert PROBLEM3_y_true_val.shape == (len(PROBLEM3_y_pred_proba_val),),␣
 ↪"PROBLEM3_y_true_val does not have the correct shape"
except Exception as e:
    print(e)
else:
    print("Good, your PROBLEM3_y_true_val is a numpy array of shape␣
 ↪(len(PROBLEM3_y_pred_proba_val),)")
try:
    assert isinstance(PROBLEM3_y_pred_proba_test, np.ndarray),␣
 ↪"PROBLEM3_y_pred_proba_test is not a numpy array"
    assert PROBLEM3_y_pred_proba_test.shape == (len(PROBLEM3_y_true_test),),␣
 ↪"PROBLEM3_y_pred_proba_test does not have the correct shape"
except Exception as e:
    print(e)
else:
    print("Good, your PROBLEM3_y_pred_proba_test is a numpy array of shape␣
 ↪(len(PROBLEM3_y_true_test),)")
try:
    assert isinstance(PROBLEM3_y_true_test, np.ndarray), "PROBLEM3_y_true_test␣
 ↪is not a numpy array"
    assert PROBLEM3_y_true_test.shape == (len(PROBLEM3_y_pred_proba_test),),␣
 ↪"PROBLEM3_y_true_test does not have the correct shape"
except Exception as e:
    print(e)
else:
    print("Good, your PROBLEM3_y_true_test is a numpy array of shape␣
 ↪(len(PROBLEM3_y_pred_proba_test),)")
```

```
try:
    assert isinstance(cost(np.array([1,1,0,0]),np.array([0.9,0.8,0.1,0.2]),0.5),␣
 ↪float), "cost does not return a float"
except Exception as e:
    print(e)
else:
    print("Good, your cost function returns a float")
try:
    assert cost(np.array([1,1,0,0]),np.array([0.9,0.8,0.1,0.2]),0.5) == 50.0,␣
 ↪"cost does not return the correct value for the test case"
except Exception as e:
    print(e)
else:
    print("Good, your cost function returns the correct value for the test case")

try:
    assert isinstance(problem3_threshold, float), "problem3_threshold is not a␣
 ↪float"
    assert 0 <= problem3_threshold <= 1, "problem3_threshold is not between 0␣
 ↪and 1"
except Exception as e:
    print(e)
else:
    print("Good, your problem3_threshold is a float between 0 and 1")
try:
    assert isinstance(problem3_cost_val, float), "problem3_cost_val is not a␣
 ↪float"
    assert problem3_cost_val >= 0, "problem3_cost_val is not non-negative"
except Exception as e:
    print(e)
else:
    print("Good, your problem3_cost_val is a float that is non-negative")
try:
    assert isinstance(problem3_y_pred_val, np.ndarray), "problem3_y_pred_val is␣
 ↪not a numpy array"
    assert problem3_y_pred_val.shape == (len(PROBLEM3_y_true_val),),␣
 ↪"problem3_y_pred_val does not have the correct shape"
except Exception as e:
    print(e)
else:
    print("Good, your problem3_y_pred_val is a numpy array of shape␣
 ↪(len(PROBLEM3_y_true_val),)")
try:
    assert np.all(np.isin(problem3_y_pred_val, [0, 1])), "problem3_y_pred_val␣
 ↪does not contain only 0s and 1s"
except Exception as e:
```

```
    print(e)
else:
    print("Good, your problem3_y_pred_val contains only 0s and 1s")
try:
    assert isinstance(problem3_precision_1, float), "problem3_precision_1 is not␣
 ↪a float"
    assert 0 <= problem3_precision_1 <= 1, "problem3_precision_1 is not between␣
 ↪0 and 1"
except Exception as e:
    print(e)
else:
    print("Good, your problem3_precision_1 is a float between 0 and 1")
try:
    assert isinstance(problem3_recall_1, float), "problem3_recall_1 is not a␣
 ↪float"
    assert 0 <= problem3_recall_1 <= 1, "problem3_recall_1 is not between 0 and␣
 ↪1"
except Exception as e:
    print(e)
else:
    print("Good, your problem3_recall_1 is a float between 0 and 1")
try:
    assert isinstance(problem3_precision_0, float), "problem3_precision_0 is not␣
 ↪a float"
    assert 0 <= problem3_precision_0 <= 1, "problem3_precision_0 is not between␣
 ↪0 and 1"
except Exception as e:
    print(e)
else:
    print("Good, your problem3_precision_0 is a float between 0 and 1")
try:
    assert isinstance(problem3_recall_0, float), "problem3_recall_0 is not a␣
 ↪float"
    assert 0 <= problem3_recall_0 <= 1, "problem3_recall_0 is not between 0 and␣
 ↪1"
except Exception as e:
    print(e)
else:
    print("Good, your problem3_recall_0 is a float between 0 and 1")
try:
    assert isinstance(problem3_threshold_01, float), "problem3_threshold_01 is␣
 ↪not a float"
    assert 0 <= problem3_threshold_01 <= 1, "problem3_threshold_01 is not␣
 ↪between 0 and 1"
except Exception as e:
    print(e)
```

```
else:
    print("Good, your problem3_threshold_01 is a float between 0 and 1")
try:
    assert isinstance(problem3_cost_difference, float),␣
 ↪"problem3_cost_difference is not a float"
    assert problem3_cost_difference >= 0, "problem3_cost_difference is not␣
 ↪non-negative"
except Exception as e:
    print(e)
else:
    print("Good, your problem3_cost_difference is a float that is non-negative")
try:
    assert isinstance(problem3_lower_bound, float), "problem3_lower_bound is not␣
 ↪a float"
    assert problem3_lower_bound >= 0, "problem3_lower_bound is not non-negative"
except Exception as e:
    print(e)
else:
    print("Good, your problem3_lower_bound is a float that is non-negative")
try:
    assert isinstance(problem3_upper_bound, float), "problem3_upper_bound is not␣
 ↪a float"
    assert problem3_upper_bound >= 0, "problem3_upper_bound is not non-negative"
except Exception as e:
    print(e)
else:
    print("Good, your problem3_upper_bound is a float that is non-negative")
```