EMSE 6574 Final Exam (Fall 2024)

Instructions

Please write your name and university-issued email address below in the space provided.

Name: _____

Email Address: _____

You will have 100 minutes to answer the questions contained herein. You may submit the exam at any time within that period. Once you begin the exam, you may not leave the room until you submit it.

You are expected to not consult with any other source of information during the exam period. No phones! Please take a moment before the exam starts to clear your desk of all materials except for your cheat sheet. There should be no talking for any reason during the exam period. If you have a question about the exam material, raise your hand and wait for an opportunity to ask an instructor for clarification.

When you are ready, you may begin. Good luck!

Evaluation

The weight of each section is detailed below. Partial credit may be awarded, and there is no penalty for guessing, so you are encouraged to not leave any questions blank.

The exam will be graded "on a curve". Keep focused, stay light, and don't spend too long on any single question. If you get stuck on one question, you are encouraged to move on and pick up as many points as possible.

Module 1	**Weight (102%)**
Part I: Programming and Analytics	18%
Part II: Datatypes	18%
Part III: Code Tracing	18%
Part IV: Data Processing (A, B, and C)	48%
Modules 2 and 3	**Weight (58%)**
Part V: Data Analytics	30%
Part VI: Machine Learning	28%
Total Exam Weight:	**160%**

I. Python for Programming and Data Analytics [18%]

1. What is the difference between **software development** and **machine learning**?

2. When we perform data analysis, we have a choice between a number of different programming languages and tools. So, **why is Python a good choice**, as compared to other languages and tools? List at least three benefits of Python, and briefly explain each.

    a.

    b.

    c.

3. For each of the following Python **packages**, briefly describe its main purpose. In other words, what does that package help us do? Be specific.

    a. Requests:

    b. Pandas:

    c. Plotly / Plotly Express:

    d. SciPy:

    e. Sklearn / Scikit-learn:

II. Python Datatypes [18%]

4. For each of the following example Python objects, what is its **datatype**? FYI: for nested objects, specify only the datatype of the parent (outermost) object. Provide your answers in the column on the right.

   a. `True` *Datatype*: _____

   b. `None` *Datatype*: _____

   c. `99` *Datatype*: _____

   d. `9.99` ` *Datatype*: _____

   e. `"Buy our products!!"` *Datatype*: _____

   f. `["AAPL", "MSFT", "GOOGL", "NFLX", "SPOT"]` *Datatype*: _____

   g. `{"symbol": "AAPL", "product": "iPhone"}` *Datatype*: _____

   h. `[{"symbol": "AAPL"}, {"symbol": "GOOGL"}]` *Datatype*: _____

   Use the provided `report` variable below to answer the questions below. For each of the objects, what is its **datatype**?

   ```
   report = {"symbol": "AAPL", "sales": [85, 90, 95, 100, 105]}
   ```

   i. `report` *Datatype*: _____

   j. `report["symbol"]` *Datatype*: _____

   k. `report["sales"]` *Datatype*: _____

   Use the provided `result` variable below to answer the questions below. For each of the objects, what is its **datatype**?

   ```
   result = ({"symbol": "AAPL"} == {"symbol": "GOOGL"})
   ```

   l. `result` *Datatype*: _____

III. Code Tracing [18%]

5. Reference the code expressions provided below. There are a number of **print statements**. For each print statement, **what result will we see**? Provide your answers in the "output" space below each corresponding "input". In the case of multiple print statements, make sure to display them all. Case matters, so ensure the results you write observe the proper capitalization.

a. Input:

```
if 5 + 5 == 10:
    print("YEP")
print("BOTTOM")
```

Output:

b. Input:

```
if 5 + 5 != 10:
    print("YEP")
print("BOTTOM")
```

Output:

c. Input:

```
letters = list("abcdef")
print(letters)
```

Output:

d. Input:

```
message = "good luck, have fun"

message_parts = message.split(", ")

print(len(message_parts))
print(message_parts[0].upper())
print(message_parts[1].upper())
```

Output:

e. Given:

```
def calculate_bonus(sales):
    if sales >= 1000:
        rate = 0.10
    else:
        rate = 0.05

    return sales * rate

# unit tests (all passing):
assert calculate_bonus(600) == 30.0
assert calculate_bonus(800) == 40.0
assert calculate_bonus(1000) == 100.0
assert calculate_bonus(1200) == 120.0
```

Input:

```
total_bonus = 0

employees = {'AJ': 1100, 'Dan': 700, 'Priya': 1500}

for employee, sales in employees.items():
    bonus = calculate_bonus(sales)

    print(employee.upper(), "BONUS:", bonus)

    total_bonus += bonus

print("TOTAL BONUS:", total_bonus)
```

Output:

IV. Data Processing (Crunch the Zoom Transcript Data)

See the Python variable called `captions` provided at the end of the exam booklet. This data represents a transcript from a Zoom video call meeting between teammates at a software development company. Use this data to answer all the questions in this part.

NOTE: in this `captions` data, assume the speaker name appears at the beginning of the caption text, and is followed by a colon (":") to separate the speaker name from what they said (i.e "speaker name: what they said"). For simplicity, assume there are no other colons present in the caption text.

A. Captions [14%]

6. Given the provided `captions` variable, **write Python code** to answer the following questions.

    a. Display / print the number of captions (i.e. `11`).

    b. Display / print the full text of the first caption (i.e. `'Hiroshi: Good morning, everyone. Let's get started. Today we're discussing the next steps for the product design and development.'`).

    c. Display / print the duration of this meeting, which is the same as the end time of the last caption (i.e. five minutes, or `'00:05:00'`).

    d. Loop through the captions, and print the start time and full text of each.

B. Speaker Names [24%]

7. Given the provided `captions` variable, **write Python code** to answer the following questions.

    a. Display / print the name of the first speaker (i.e. `'Hiroshi'`).

    b. Write a custom function called `parse_speaker_name` to output the speaker name for a given caption text. The function should accept a parameter input called `caption_text` (the caption text, assumed to be in the format of "speaker name: what they said"). The function should return the corresponding speaker name.

    The working version of this function should make the following unit tests pass:

    ```
    example_text = "Hiroshi: Good morning, everyone. Let's get started..."
    assert parse_speaker_name(example_text) == "Hiroshi"
    ```

    NOTE: regardless of whether you get this question correct, in future questions, you can reference / use the `parse_speaker_name` function, as if it is functioning properly, and passing the tests.

    c. Use the `parse_speaker_name` function to display / print the name of the last speaker.

8. Given the provided `captions` variable, **write Python code** to answer the following questions.

    a. Who spoke? Collect a list of unique speaker names (i.e. `['Hiroshi', 'Aisha', 'Carlos']`) and store them in a variable called `speaker_names`. Then print this list of speakers.

    NOTE: regardless of whether you get this question correct, in future questions, you can reference / use the `speaker_names` variable, as if it has the correct values.

    b. How many times did each speaker speak? For each unique speaker, print their name, and the **number of times they spoke** in total. You can either print the results for each speaker separately (each on its own line), or as a single dictionary (i.e. `{'Hiroshi': 5, 'Aisha': 4, 'Carlos': 2}`).

C. Speaking Time [10%]

Assume you have access to a working function called `parse_timestamp`, which converts a given timestamp string into a `datetime.timedelta` object, which can be used to perform operations such as comparisons, addition, and subtraction between different durations of time. If you have never worked with this kind of datatype before, consult the tests below to learn more.

The working version of this function is assumed to make the following unit tests pass:

```
start_time = parse_timestamp("00:00:30")
end_time = parse_timestamp("00:01:15")
duration = parse_timestamp("00:00:45")

# adding and subtracting times:
assert end_time - start_time == duration
assert start_time + duration == end_time

# comparing times:
assert start_time < end_time

# accessing number of seconds elapsed:
assert start_time.seconds == 30
assert end_time.seconds == 75
assert duration.seconds == 45
```

9. Given the provided `captions` variable and the information provided above, **write Python code** to answer the following questions.

    a. How long did each speaker speak? For each unique speaker, print their name, and the total **duration of time (in seconds)** they spoke over the entire meeting. You can either print the results for each speaker separately (each on its own line), or as a single dictionary (i.e. `{'Hiroshi': 135, 'Aisha': 105, 'Carlos': 60}`).

V. Data Analytics [30%]

10. Reference the code cells provided below. Assume these cells are run from top to bottom in a notebook. For each set of input cells, **what result will we see**? Provide your answers in the "output" space below each corresponding "input".

```
from pandas import DataFrame

data = {
    'Movie': [
        "Harry Potter and the Sorcerer's Stone",
        "Harry Potter and the Chamber of Secrets",
        "Harry Potter and the Prisoner of Azkaban",
        "Harry Potter and the Goblet of Fire",
        "Harry Potter and the Order of the Phoenix",
        "Harry Potter and the Half-Blood Prince",
        "Harry Potter and the Deathly Hallows: Part 1",
        "Harry Potter and the Deathly Hallows: Part 2"
    ],
    'Rating': [7.6, 7.4, 7.9, 7.7, 7.5, 7.6, 7.7, 8.1],
    'Year': [2001, 2002, 2004, 2005, 2007, 2009, 2010, 2011]
}
df = DataFrame(data)
```

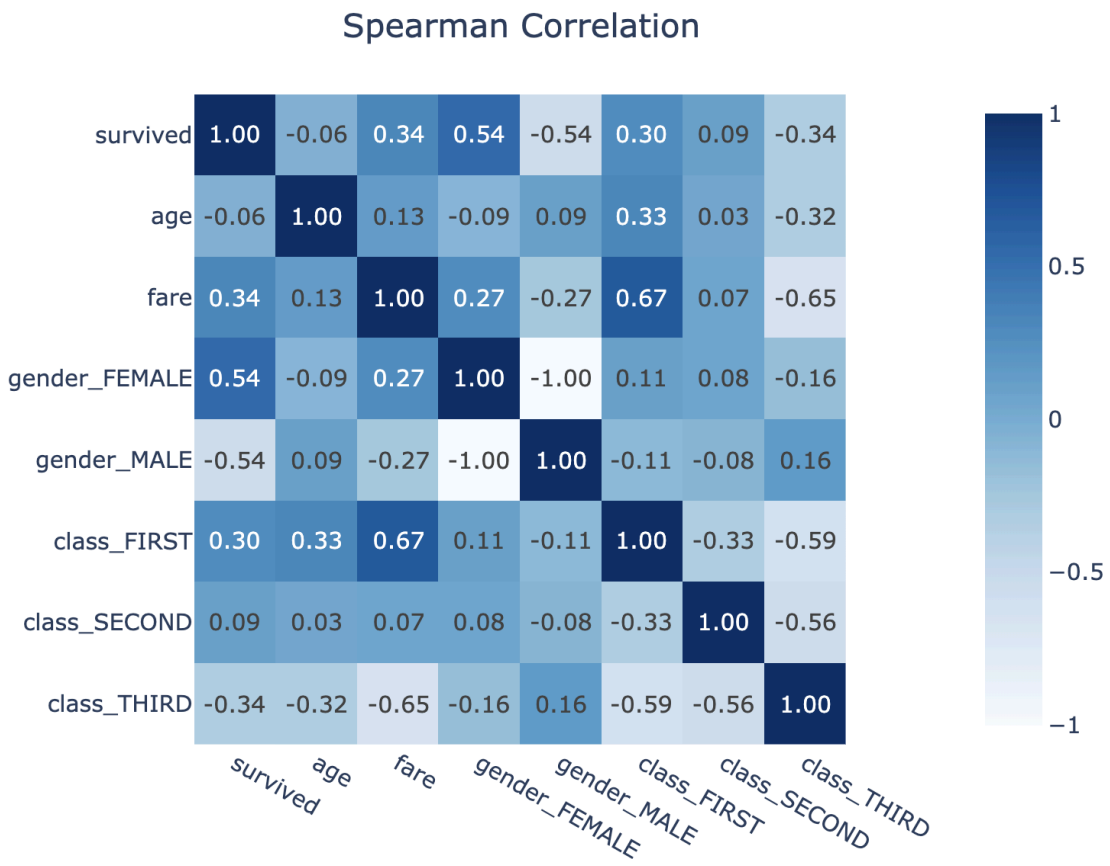
a. Input:
```
df.shape
```
Output:

b. Input:
```
results = df.sort_values(by=["Rating"], ascending=False)
results["Movie"].values[0]
```
Output:

c. Input:
```
df.rename(columns={"Movie":"Title"})
df.columns.tolist()
```
Output:

d. Input:
```
df.drop(columns=["Title"], errors="ignore")
df.columns.tolist()
```
Output:

11. Use the **correlation matrix** below to answer the following questions.



a. On the diagonal, the values are all equal to one. Why? What does this mean?

b. Which feature is most positively correlated with the target "survived"?

c. Which feature is most negatively correlated with the target "survived"?

d. Which feature is most positively correlated with the feature "fare"?

e. Which feature is most negatively correlated with the feature "fare"?

f. Which pair of two features are most negatively correlated with each other?

VI. Machine Learning [28%]

12. See the correlation matrix from the previous page. Suppose our goal is to use this data to train a **classification model** to predict whether a given passenger survived, using the other variables as features. Your colleague shows you the following code and asks you for a code review.

```
1  from sklearn.linear_model import LogisticRegression
2
3  target = "survived"
4  y = df[target]
5  x = df.drop(columns=[target])
6
7  model = LogisticRegression()
8  model.fit(x, y)
```

a. Besides the target, there is one additional feature which should probably be dropped (on line 5). Which feature is that? Why?

b. There is an important step missing from this process (between lines 5 and 7). What is the missing step, and why is it important / necessary?

c. During the model initialization (on line 7), there is an important parameter missing. What is the missing parameter, and why is it important / necessary?

d. After fixing your colleague's code and properly training the `LogisticRegression` model, you proceed to evaluate the model's performance. What **metrics** could you use? List the names of four appropriate metrics. Describe / explain each of the metrics, including equations as appropriate. Identify some pros and cons of each.

Metric 1:

Metric 2:

Metric 3:

Metric 4:

This page has been left intentionally blank. Feel free to make notes on it. Its contents will not be evaluated.

*The `captions` variable provided on the following two pages is to be used in conjunction with in the **Data Processing** part of the exam. Feel free to detach these pages and use them as a visual reference during the exam. If you do detach them, write your name on them and remember to return them at the end of the exam along with the rest of your exam booklet!*

```
captions = [
  {
    'id': 1,
    'start': '00:00:00',
    'end': '00:00:20',
    'text': "Hiroshi: Good morning, everyone. Let's get started. Today
we're discussing the next steps for the product design and development."
  },
  {
    'id': 2,
    'start': '00:00:20',
    'end': '00:00:45',
    'text': 'Aisha: Right, so we have the initial prototype ready. The user
feedback we collected has been positive, but there are a few concerns about
the usability.'
  },
  {
    'id': 3,
    'start': '00:00:45',
    'end': '00:01:10',
    'text': 'Hiroshi: Agreed. We need to address the navigation flow and
some interface issues. Our goal is to simplify the user journey without
compromising on features.'
  },
  {
    'id': 4,
    'start': '00:01:10',
    'end': '00:01:40',
    'text': 'Carlos: I think we should also consider performance
optimization early on. The product needs to load faster, especially on
mobile devices.'
  },
  {
    'id': 5,
    'start': '00:01:40',
    'end': '00:02:05',
    'text': 'Aisha: Absolutely, mobile optimization is a priority. We
should start looking at integrating some responsive design techniques in
this phase.'
  },

  # ... (continued on next page)
```

```
  # ... (continued from previous page)

  {
    'id': 6,
    'start': '00:02:05',
    'end': '00:02:35',
    'text': "Hiroshi: Let's also think about the color scheme and
typography. We need something modern but still professional to appeal to
our target audience."
  },
  {
    'id': 7,
    'start': '00:02:35',
    'end': '00:03:00',
    'text': "Aisha: I'll have the design team draft a couple of options by
the end of the week. Does everyone agree to review them next Monday?"
  },
  {
    'id': 8,
    'start': '00:03:00',
    'end': '00:03:30',
    'text': "Carlos: That works for me. In the meantime, I'll start working
on improving load times and checking compatibility with older browsers."
  },
  {
    'id': 9,
    'start': '00:03:30',
    'end': '00:04:00',
    'text': "Hiroshi: Sounds good. Let's set another meeting for next
Friday to finalize everything before our presentation to stakeholders."
  },
  {
    'id': 10,
    'start': '00:04:00',
    'end': '00:04:30',
    'text': "Aisha: Great! I think we're on track, but let's keep
communication open if anyone runs into any issues."
  },
  {
    'id': 11,
    'start': '00:04:30',
    'end': '00:05:00',
    'text': "Hiroshi: Thanks, everyone! Let's make sure we're hitting our
milestones. Meeting adjourned."
  }
]
```

*FYI this is the function referenced in **Question 8**.*

```
from datetime import timedelta

def parse_timestamp(time_str):
    """Converts a timestamp string to a timedelta object.

        Params :
            time_str (string) : in format 'hh:mm:ss' or 'hh:mm:ss.ms'

        Returns a datetime.timedelta object

        Examples:

            parse_timestamp('01:14:30')

            parse_timestamp('01:14:30.390')

    """
    h, m, s = time_str.split(':')

    if "." in s:
        # handles when there are milliseconds present, like when s = "30.390"
        s, ms = s.split(".")
    else:
        ms = 0

    return timedelta(hours=int(h), minutes=int(m),
                     seconds=int(s), milliseconds=int(ms)
                    )
```