CPSC 330 - Applied Machine Learning

Homework 1: Programming with Python

Due date: Tuesday Sep 14, 2021 at 11:59pm.

About this assignment:

The main purpose of this assignment is to check whether your programming knowledge is adequate to take CPSC 330. This assignment covers two python packages, `numpy` and `pandas`, which we'll be using throughout the course. For some of you, Python/numpy/pandas will be familiar; for others, it will be new. Either way, if you find this assignment very difficult then that could be a sign that you will struggle later on in the course. While CPSC 330 is a machine learning course rather than a programming course, programming will be an essential part of it.

Also, as part of this assignment you will likely need to consult the documentation for various Python packages we're using. This is, of course, totally OK and in fact strongly encouraged. Reading and interpreting documentation is an important skill, and in fact is one of the skills this assignment is meant to assess. For Python refresher, check out Python notes and Python resources.

Set-up

In order to do this assignment, you will need to set up the CPSC 330 software stack, which is Python and Jupyter. For software install help, see here. One you have the software stack installed, you should be able to run the next cell, which imports some packages needed for the assignment.

Imports

In [ ]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
```

Points

Each question or sub-question will have a number of points allocated to it, which is indicated right below the question name.

Instructions

rubric={points:6}

Follow the CPSC 330 homework instructions, which include information on how to do your assignment and how to submit your assignment.

Exercise 1: Loading files with Pandas

rubric={points:12}

When working with tabular data, you will typically be creating Pandas dataframes by reading data from .csv files using `pd.read_csv()`. The documentation for this function is available here.

In the "data" folder in this homework repository there are 6 different .csv files named `wine_#.csv/.txt`. Look at each of these files and use `pd.read_csv()` to load these data so that they resemble the following:

Bottle	Grape	Origin	Alcohol	pH	Colour	Aroma
1	Chardonnay	Australia	14.23	3.51	White	Floral
2	Pinot Grigio	Italy	13.20	3.30	White	Fruity
3	Pinot Blanc	France	13.16	3.16	White	Citrus
4	Shiraz	Chile	14.91	3.39	Red	Berry
5	Malbec	Argentina	13.83	3.28	Red	Fruity

You are provided with tests that use `df.equals()` to check that all the dataframes are identical. If you're in a situation where the two dataframes look identical but `df.equals()` is returning `False`, it may be an issue of types - try checking `df.index`, `df.columns`, or `df.info()`.

In [ ]:
```
df1 = None
df2 = None
df3 = None
df4 = None
df5 = None
df6 = None
```

In [ ]:
```
### BEGIN SOLUTION
# 1, no special considerations here
df1 = pd.read_csv("data/wine_1.csv", index_col=0)
# 2, need to skip first row
df2 = pd.read_csv("data/wine_2.csv", index_col=0, header=1)
# 3, need to read only first 5 rows
df3 = pd.read_csv("data/wine_3.csv", index_col=0, nrows=5)
# 4, need to specify tab-delimited
df4 = pd.read_csv("data/wine_4.txt", index_col=0, delimiter="\t")
# 5, need to use only the first 6 columns
df5 = pd.read_csv("data/wine_5.csv", index_col=0, usecols=[0, 1, 2, 3, 4, 5, 6])
# 6, need to use pd.read_csv() with tab delimiter
df6 = pd.read_csv(
    "data/wine_6.txt",
    index_col=0,
    delimiter="\t",
    header=1,
    nrows=5,
    usecols=[1, 2, 3, 4, 5, 6, 7],
)
### END SOLUTION
```

In [ ]:
```
for i, df in enumerate([df2, df3, df4, df5, df6]):
    assert df1.equals(df), f"df1 not equal to df{i + 2}"
```

```
print("All tests passed.")
```

In [ ]:
```
df
```

Exercise 2: The Titanic dataset

The file *titanic.csv* contains data of 1309 passengers who were on the Titanic's unfortunate voyage. For each passenger, the following data are recorded:

- survival - Survival (0 = No; 1 = Yes)
- class - Passenger Class (1 = 1st; 2 = 2nd; 3 = 3rd)
- name - Name
- sex - Sex
- age - Age
- sibsp - Number of Siblings/Spouses Aboard
- parch - Number of Parents/Children Aboard
- ticket - Ticket Number
- fare - Passenger Fare
- cabin - Cabin
- embarked - Port of Embarkation (C = Cherbourg; Q = Queenstown; S = Southampton)
- boat - Lifeboat (if survived)
- body - Body number (if did not survive and body was recovered)

In this exercise you will perform a number of wrangling operations to manipulate and extract subsets of the data.

Note: many popular datasets have sex as a feature where the possible values are male and female. This representation reflects how the data were collected and is not meant to imply that, for example, gender is binary.

2(a)

rubric={points:1}

Load the `titanic.csv` dataset into a pandas dataframe named `titanic_df`.

In [ ]:
```
titanic_df = None

### BEGIN SOLUTION
titanic_df = pd.read_csv("data/titanic.csv")
titanic_df
### END SOLUTION
```

In [ ]:
```
assert set(titanic_df.columns) == set(
    [
        "pclass",
        "survived",
        "name",
        "sex",
        "age",
```

```
        "sibsp",
        "parch",
        "ticket",
        "fare",
        "cabin",
        "embarked",
        "boat",
        "body",
        "home.dest",
    ]
), "All required columns are not present"
assert len(titanic_df.index) == 1309, "Wrong number of rows in dataframe"
print("Success")
```

2(b)

rubric={points:2}

The column names `sibsp` and `parch` are not very descriptive. Use `df.rename()` to rename these columns to `siblings_spouses` and `parents_children` respectively.

In [ ]:
```
titanic_df = titanic_df.rename(
    columns={"sibsp": "siblings_spouses", "parch": "parents_children"}
)
```

In [ ]:
```
### BEGIN SOLUTION
rename_map = {"sibsp": "siblings_spouses", "parch": "parents_children"}
titanic_df = titanic_df.rename(columns=rename_map)
### END SOLUTION
```

In [ ]:
```
assert set(["siblings_spouses", "parents_children"]).issubset(
    titanic_df.columns
), "Column names were not changed properly"
print("Success")
```

2(c)

rubric={points:2}

We will practice indexing different subsets of the dataframe in the following questions.

Select the column `age` using single bracket notation `[]`. What type of object is returned?

In [ ]:
```
### BEGIN SOLUTION
# A series is returned when using single-bracket notation.
titanic_df["age"]
type(titanic_df["age"])
### END SOLUTION
```

2(d)

rubric={points:2}

Now select the `age` using double bracket notation `[[]]`. What type of object is returned?

In [ ]:
```
### BEGIN SOLUTION
# A dataframe is returned when using double-bracket notation.
titanic_df[["age"]]
```

```
type(titanic_df[["age"]])
### END SOLUTION
```

2(e)

rubric={points:1}

Select the columns `pclass`, `survived`, and `age` using a single line of code.

In [ ]:
```
### BEGIN SOLUTION
titanic_df[["pclass", "survived", "age"]].head()
### END SOLUTION
```

2(f)

rubric={points:2}

Use the `iloc` method to obtain the first 5 rows of the columns `name`, `sex` and `age` using a single line of code.

In [ ]:
```
titanic_df.iloc[:5][["name", "sex", "age"]]
```

In [ ]:
```
### BEGIN SOLUTION
titanic_df.iloc[:5, [2, 3, 4]]
### END SOLUTION
```

2(g)

rubric={points:2}

Now use the `loc` method to obtain the first 5 rows of the columns `name`, `sex` and `age` using a single line of code.

In [ ]:
```
titanic_df.loc[0:4, "name":"age"]
```

In [ ]:
```
### BEGIN SOLUTION
titanic_df.loc[0:4, "name":"age"]
### END SOLUTION
```

2(h)

rubric={points:2}

How many passengers survived (`survived = 1`) the disaster? Hint: try using `df.query()` or `[]` notation to subset the dataframe and then `df.shape` to check its size.

In [ ]:
```
titanic_df[titanic_df["survived"] == 1].shape[0]
```

In [ ]:
```
### BEGIN SOLUTION
titanic_df[titanic_df["survived"] == 1].shape[0]
# shows that 500 passengers survived
### END SOLUTION
```

2(i)

rubric={points:1}

How many passengers that survived the disaster (`survived = 1`) were over 60 years of age?

In [ ]:
```
### BEGIN SOLUTION
titanic_df[(titanic_df["survived"] == 1) & (titanic_df["age"] > 60)].shape[0]
# shows that 8 passengers survived
### END SOLUTION
```

2(j)

rubric={points:2}

What was the lowest and highest fare paid to board the titanic? Store your answers as floats in the variables `lowest` and `highest` .

In [ ]:
```
lowest = None
highest = None
### BEGIN SOLUTION
lowest = titanic_df["fare"].min()  # shows that lowest fare is 0.
highest = titanic_df["fare"].max()  # shows that highest fare is 512.3292.
### END SOLUTION
```

2(k)

rubric={points:1}

Sort the dataframe by fare paid (most to least).

In [ ]:
```
### BEGIN SOLUTION
titanic_df_sorted = titanic_df.sort_values(by="fare", ascending=False)
titanic_df_sorted
### END SOLUTION
```

2(l)

rubric={points:1}

Save the sorted dataframe to a .csv file called 'titanic_fares.csv' using `to_csv()` .

In [ ]:
```
### BEGIN SOLUTION
titanic_df_sorted.to_csv("titanic_fares.csv")
### END SOLUTION
```

2(m)

rubric={points:3}

Create a scatter plot of fare (y-axis) vs. age (x-axis). Make sure to follow the guidelines on figures. You are welcome to use pandas built-in plotting or `matplotlib` .

In [ ]:
```
### BEGIN SOLUTION
titanic_df.plot.scatter(x="age", y="fare", title="fare vs. age for the titanic");
### END SOLUTION
```

2(n)

rubric={points:3}

Create a histogram of `embarked` values.

> Make sure to name the axes and give a title to your plot.

In [ ]:
```
### BEGIN SOLUTION
titanic_df["embarked"].value_counts().plot.bar().set(
    xlabel="embarked", ylabel="count", title="titanic embarked histogram"
);
### END SOLUTION
```

Exercise 3: Treasure Hunt

In this exercise, we will generate various collections of objects either as a list, a tuple, or a dictionary. Your task is to inspect the objects and look for treasure, which in our case is a particular object: **the character "T"**.

Your tasks:

For each of the following cases, index into the Python object to obtain the "T" (for Treasure).

> Please do not modify the original line of code that generates `x` (though you are welcome to copy it). You are welcome to answer this question "manually" or by writing code - whatever works for you. However, your submission should always end with a line of code that prints out `'T'` at the end (because you've found it).

In [ ]:
```
import string

letters = string.ascii_uppercase
```

The first one is done for you as an example.

Example question

In [ ]:
```
x = ("nothing", {-i: l for i, l in enumerate(letters)})
x
```

Example answer:

In [ ]:
```
x[1][-19]
```

> Note: In these questions, the goal is not to understand the code itself, which may be confusing. Instead, try to probe the types of the various objects. For example `type(x)` reveals that `x` is a tuple, and `len(x)` reveals that it has two elements. Element 0 just contains "nothing", but element 1 contains more stuff, hence `x[1]`. Then we can again probe `type(x[1])` and see that it's a dictionary. If you `print(x[1])` you'll see that the letter "T" corresponds to the key -19, hence `x[1][-19]`.

3(a)

rubric={points:2}

In [ ]:
```
# Do not modify this cell
x = [
    [letters[i] for i in range(26) if i % 2 == 0],
    [letters[i] for i in range(26) if i % 2 == 1],
]
```

In [ ]:
```
### BEGIN SOLUTION
x[1][9]
### END SOLUTION
```

3(b)

rubric={points:2}

In [ ]:
```
# Do not modify this cell
np.random.seed(1)
x = np.random.choice(list(set(letters) - set("T")), size=(100, 26), replace=True)
x[np.random.randint(100), np.random.randint(26)] = "T"
```

In [ ]:
```
### BEGIN SOLUTION
for i in range(100):
    if "T" in x[i]:
        print(x[i])
x[95, 2]
### END SOLUTION
```

3(c)

rubric={points:3}

In [ ]:
```
# Do not modify this cell
n = 26
x = dict()
for i in range(n):
    x[string.ascii_lowercase[i]] = {
        string.ascii_lowercase[(j + 1) % n]: [[letters[j]] if j - 2 == i else None]
        for j in range(n)
    }
```

In [ ]:
```
### BEGIN SOLUTION
for key in x.keys():
    for inner_key, inner_value in x[key].items():
        if inner_value != [None]:
            print(key, inner_key, inner_value[0][0])

print(x["r"]["u"][0][0])
for val in x.values():
    if "T" in val:
        print(val)
### END SOLUTION
```

In [ ]:
```
### BEGIN SOLUTION
x["r"]["u"][0][0]
### END SOLUTION
```

Submission instructions

PLEASE READ: When you are ready to submit your assignment do the following:

1. Run all cells in your notebook to make sure there are no errors by doing `Kernel -> Restart Kernel and Clear All Outputs` and then `Run -> Run All Cells`.
2. Notebooks with cell execution numbers out of order or not starting from "1" will have marks deducted. Notebooks without the output displayed may not be graded at all (because we need to see the output in order to grade your work).
3. Upload the assignment using Gradescope's drag and drop tool. Check out this Gradescope Student Guide if you need help with Gradescope submission.

In [ ]: