hw1

September 12, 2023

[1]:
```
# Initialize Otter
import otter
grader = otter.Notebook("hw1.ipynb")
```

1 CPSC 330 - Applied Machine Learning

1.1 Homework 1: Programming with Python

Due date: See the Calendar.

1.1.1 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. That said, do not use Large Language Model tools such as ChatGPT to complete your assignment; it would be self-deceptive and by doing so you will only be hurting your own learning.

For Python refresher, check out Python notes and Python resources.

1.1.2 Set-up

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

Setting up the software stack can be frustrating and challenging. But remember that it is an integral part of becoming a data scientist or machine learning engineer. This is going to be a valuable skill for your future self. Make the most of the tutorials available today and tomorrow, as the TAs are ready to assist you with the setup.

1.2 Imports

[2]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
```

1.3 Points

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

1.4 Instructions

rubric={points}

PLEASE READ: 1. Before submitting the assignment, 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. Follow the CPSC 330 homework instructions, which include information on how to do your assignment and how to submit your assignment. 4. Upload the assignment using Gradescope's drag and drop tool. Check out this Gradescope Student Guide if you need help with Gradescope submission. 5. Make sure that the plots and output are rendered properly in your submitted file. If the .ipynb file is too big and doesn't render on Gradescope, also upload a pdf or html in addition to the .ipynb so that the TAs can view your submission on Gradescope.

Points: 6

1.5 Instructions

rubric={points}

PLEASE READ: 1. Before submitting the assignment, 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. Follow the CPSC 330 homework instructions, which include information on how to do your assignment and how to submit your assignment. 4. Upload the assignment using Gradescope's drag and drop tool. Check out this Gradescope Student Guide if you need help with Gradescope submission. 5. Make sure that the plots and output are rendered properly in your submitted file. If the .ipynb file is too big and doesn't render on Gradescope, also upload a pdf or html in addition to the .ipynb so that the TAs can view your submission on Gradescope.

Points: 6

1.6 Exercise 1: Loading files with Pandas

rubric={points}

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()`.

Your solution_1

Points: 12

[3]:
```
df1 = pd.read_csv('data/wine_1.csv',)
df2 = pd.read_csv('data/wine_2.csv', header= 1)
df3 = pd.read_csv('data/wine_3.csv', engine='python', skipfooter=2)
df4 = pd.read_csv('data/wine_4.txt', delimiter='\t')
df5 = pd.read_csv('data/wine_5.csv', usecols={'Bottle', 'Grape', 'Origin',␣
↪'Alcohol', 'pH', 'Colour', 'Aroma'})
df6 = pd.read_csv('data/wine_6.txt', delimiter='\t',␣
↪engine='python',skipfooter=2,header=1, usecols={'Bottle', 'Grape', 'Origin',␣
↪'Alcohol', 'pH', 'Colour', 'Aroma'})
```

[4]:
```
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.")
```

```
All tests passed.
```

1.7 Exercise 2: The Titanic dataset

The file *data/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.

2.1 rubric={points}

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

Your solution_2.1

Points: 1

[5]:
```
titanic_df = pd.read_csv('./data/titanic.csv')
...
```

[6]:
```
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")
```

```
Success
```

2.2 rubric={points}

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

Your solution_2.2

Points: 2

[7]:
```
...
titanic_df.rename(columns={"sibsp": "siblings_spouses", "parch":␣
↪"parents_children"}, inplace=True)
```

[8]:
```
assert set(["siblings_spouses", "parents_children"]).issubset(
    titanic_df.columns
), "Column names were not changed properly"
print("Success")
```

```
Success
```

2.3 rubric={points}

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?

Your solution_2.3

Points: 2

[9]:
```
...
nameVar = titanic_df['age']
print(nameVar)
print(type(nameVar))
# it returns a Series (1d array)
```

```
0       29.0000
1        0.9167
2        2.0000
3       30.0000
4       25.0000
         ...
1304    14.5000
1305        NaN
1306    26.5000
1307    27.0000
1308    29.0000
Name: age, Length: 1309, dtype: float64
<class 'pandas.core.series.Series'>
```

2.4 rubric={points}

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

Your solution_2.4

Points: 2

[10]:
```
...
ageObj = titanic_df[["age"]]
print(type(ageObj))
# it returns a panada DataFrame object (2d table)
```

```
<class 'pandas.core.frame.DataFrame'>
```

2.5 rubric={points}

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

Your solution_2.5

Points: 1

[11]:
```
...
titanic_df[['pclass','survived','age']]
```

[11]:

	pclass	survived	age
0	1	1	29.0000
1	1	1	0.9167
2	1	0	2.0000
3	1	0	30.0000
4	1	0	25.0000
...	...	...	...
1304	3	0	14.5000
1305	3	0	NaN
1306	3	0	26.5000
1307	3	0	27.0000
1308	3	0	29.0000

[1309 rows x 3 columns]

2.6 rubric={points}

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

Your solution_2.6

Points: 2

[12]:
```
...
titanic_df.iloc[:5]
```

[12]:

	pclass	survived	name	sex	\
0	1	1	Allen, Miss. Elisabeth Walton	female	
1	1	1	Allison, Master. Hudson Trevor	male	
2	1	0	Allison, Miss. Helen Loraine	female	

```
3       1         0             Allison, Mr. Hudson Joshua Creighton    male
4       1         0  Allison, Mrs. Hudson J C (Bessie Waldo Daniels)  female

       age  siblings_spouses  parents_children  ticket      fare    cabin  \
0  29.0000                 0                 0   24160  211.3375       B5
1   0.9167                 1                 2  113781  151.5500  C22 C26
2   2.0000                 1                 2  113781  151.5500  C22 C26
3  30.0000                 1                 2  113781  151.5500  C22 C26
4  25.0000                 1                 2  113781  151.5500  C22 C26

  embarked boat   body                        home.dest
0        S    2    NaN                     St Louis, MO
1        S   11    NaN  Montreal, PQ / Chesterville, ON
2        S  NaN    NaN  Montreal, PQ / Chesterville, ON
3        S  NaN  135.0  Montreal, PQ / Chesterville, ON
4        S  NaN    NaN  Montreal, PQ / Chesterville, ON
```

2.7 rubric={points}

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

Your solution_2.7

Points: 2

[13]:
```
...
titanic_df.loc[0:4, ['name', 'sex','age']]
```

[13]:
```
                                             name     sex      age
0                    Allen, Miss. Elisabeth Walton  female  29.0000
1                   Allison, Master. Hudson Trevor    male   0.9167
2                     Allison, Miss. Helen Loraine  female   2.0000
3             Allison, Mr. Hudson Joshua Creighton    male  30.0000
4  Allison, Mrs. Hudson J C (Bessie Waldo Daniels)  female  25.0000
```

2.8 rubric={points}

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.

Your solution_2.8

Points: 2

[14]:
```
...
titanic_df.query('survived ==1 ').shape[0]
titanic_df[titanic_df['survived'] == 1].shape[0]
```

[14]:
```
500
```

2.9 rubric={points}

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

Your solution_2.9

Points: 1

[15]:
```
...
titanic_df.query('survived == 1').query('age > 60').shape[0]
```

[15]:
```
8
```

2.10 rubric={points}

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

Your solution_2.10

Points: 2

[16]:
```
lowest = titanic_df['fare'].min()
highest = titanic_df['fare'].max()
print(lowest)
print(highest)
...
```

```
0.0
512.3292
```

[16]:
```
Ellipsis
```

2.11 rubric={points}

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

Your solution_2.11

Points: 1

[17]:
```
...
sorted_titanic_df = titanic_df.sort_values(by=['fare'], ascending=False)
sorted_titanic_df.head()
```

[17]:
```
     pclass  survived                                             name  \
183       1         1                            Lesurer, Mr. Gustave J
302       1         1                                 Ward, Miss. Anna
49        1         1                    Cardeza, Mr. Thomas Drake Martinez
50        1         1  Cardeza, Mrs. James Warburton Martinez (Charlo...
113       1         1                           Fortune, Miss. Mabel Helen

        sex   age  siblings_spouses  parents_children    ticket      fare  \
```

```
183    male  35.0                0                0  PC 17755  512.3292
302  female  35.0                0                0  PC 17755  512.3292
49     male  36.0                0                1  PC 17755  512.3292
50   female  58.0                0                1  PC 17755  512.3292
113  female  23.0                3                2     19950  263.0000

           cabin embarked boat  body  \
183         B101        C    3   NaN
302          NaN        C    3   NaN
49   B51 B53 B55        C    3   NaN
50   B51 B53 B55        C    3   NaN
113  C23 C25 C27        S   10   NaN

                                        home.dest
183                                           NaN
302                                           NaN
49   Austria-Hungary / Germantown, Philadelphia, PA
50                     Germantown, Philadelphia, PA
113                                  Winnipeg, MB
```

2.12 rubric={points}

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

Your solution_2.12

Points: 1

[18]:
```
...
sorted_titanic_df.to_csv('titanic_fares.csv', encoding='utf-8')
```

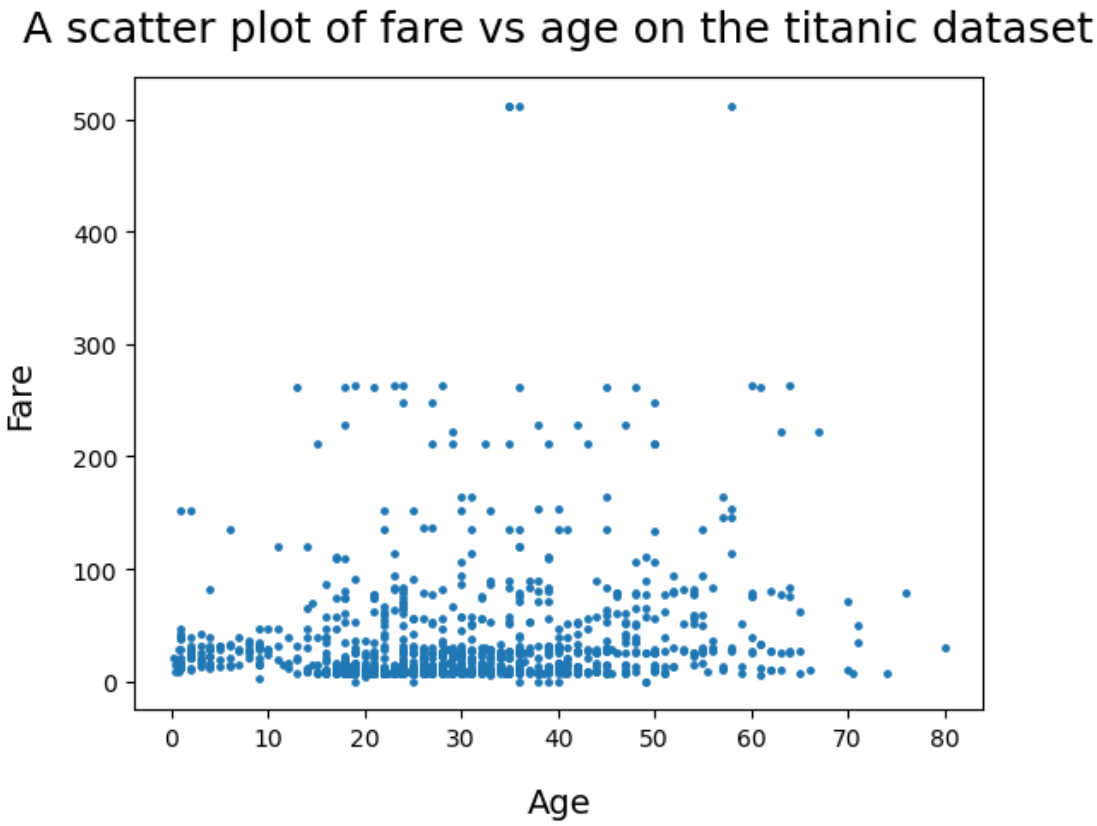
2.13 rubric={points:3}

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`.

Your solution_2.13

Points: 3

[19]:
```
...
titanic_df.plot.scatter(x='age', y='fare', s=6)
plt.title('A scatter plot of fare vs age on the titanic dataset', fontsize = 18,␣
 ↪pad = 15)
plt.xlabel('Age', fontsize=14, labelpad=15)
plt.ylabel('Fare', fontsize=14, labelpad=15)
plt.show()
```



2.14 rubric={points}

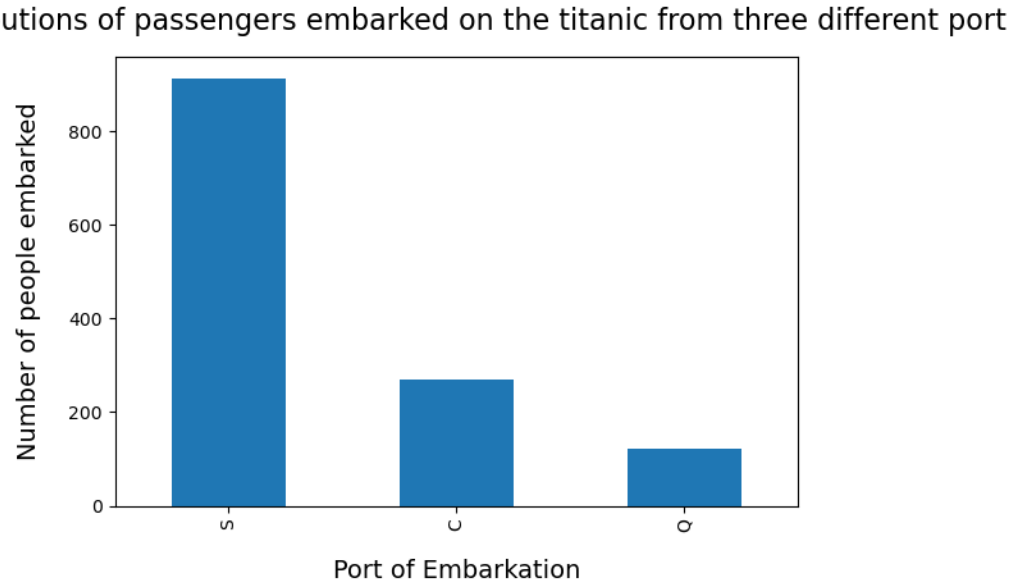
Create a bar chart of `embarked` values.

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

Your solution_2.14

Points: 3

[20]:
```
...
titanic_df['embarked'].value_counts().plot.bar()
plt.title('Distributions of passengers embarked on the titanic from three␣
 ↪different port', fontsize=16, pad = 15)
plt.xlabel('Port of Embarkation', fontsize = 14, labelpad=15)
plt.ylabel('Number of people embarked', fontsize = 14, labelpad=15)
plt.tight_layout()
```



1.8 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).

[21]:
```
import string
letters = string.ascii_uppercase
```

The first one is done for you as an example.

Example question

[22]:
```
x = ("nothing", {-i: l for i, l in enumerate(letters)})
```

Example answer:

[23]:
```
x[1][-19]
```

[23]:
```
'T'
```

> 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]`.

3.1 rubric={points}

[24]:
```
# 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],
]
```

Your solution_3.1

Points: 2

[25]:
```
...
# print(x[0])
# print(x[1])
# for i in range(len(x[1])):
#     if x[1][i] == 'T':
#         print(i)
x[1][9]
```

[25]:
```
'T'
```

3.2 rubric={points}

[26]:
```
# 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"
```

Your solution_3.2

Points: 2

[27]:
```
...
for i in range(len(x)):
    for j in range(len(x[i])):
        if (x[i][j] == "T"):
            print(i, j)
x[95][2]
```

```
95 2
```

[27]:
```
'T'
```

3.3 rubric={points}

[28]:
```
# 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)
    }
```

Your solution_3.3

Points: 3

[29]:
```
...

for i in x.keys():
    for j in x.get(i).keys():
        l = x.get(i).get(j);
        if(l[0] and l[0][0]== "T"):
            print(i, j)
x.get('r').get('u')[0][0]
```

```
r u
```

[29]:
```
'T'
```

Before submitting your assignment, please make sure you have followed all the instructions in the Submission Instructions section at the top.

Well done!!


WELL
DONE!