DataIntegrationAndAnalysis Exercise Readme

Eunha Park (Registration number :12303629)

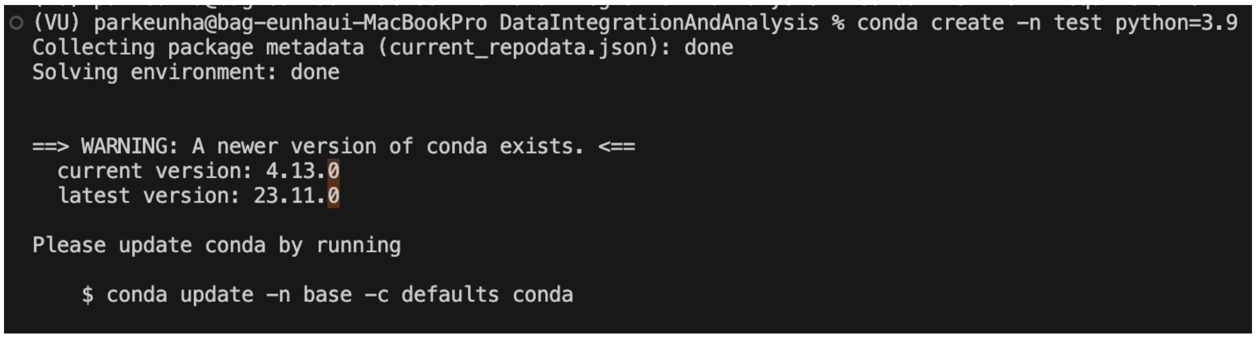
Donghyeon Kang (Registration number : 12303601)

Yujin Jang (Registration number:12303598)

0. Prepare the environment

1. Download the folder and work in that folder (file name is written in relative path)
2. Install anaconda
2. Create conda env with python version 3.9

```
conda create -n test python=3.9
```



```
(VU) parkeunha@bag-eunhaui-MacBookPro DataIntegrationAndAnalysis % conda create -n test python=3.9
Collecting package metadata (current_repodata.json): done
Solving environment: done


==> WARNING: A newer version of conda exists. <==
  current version: 4.13.0
  latest version: 23.11.0

Please update conda by running

    $ conda update -n base -c defaults conda
```

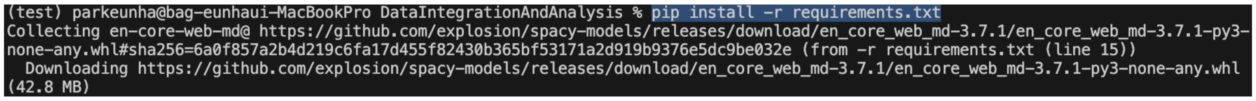
4. Activate conda env made in step 2

```
conda activate test
```

```
(VU) parkeunha@bag-eunhaui-MacBookPro DataIntegrationAndAnalysis % conda activate test
```

5. Install requirements

```
pip install -r requirements.txt
```



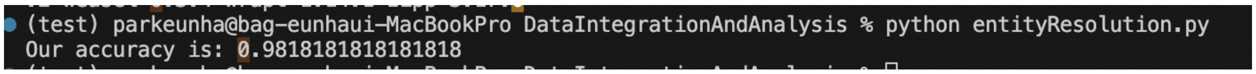
```
(test) parkeunha@bag-eunhaui-MacBookPro DataIntegrationAndAnalysis % pip install -r requirements.txt
Collecting en-core-web-md@ https://github.com/explosion/spacy-models/releases/download/en_core_web_md-3.7.1/en_core_web_md-3.7.1-py3-
none-any.whl#sha256=6a0f857a2b4d219c6fa17d455f82430b365bf53171a2d919b9376e5dc9be032e (from -r requirements.txt (line 15))
  Downloading https://github.com/explosion/spacy-models/releases/download/en_core_web_md-3.7.1/en_core_web_md-3.7.1-py3-none-any.whl
(42.8 MB)
```

1. Task 1

Reproducing Results>

Run entityResolution.py then you can get accuracy!!

```
python entityResolution.py
```



```
(test) parkeunha@bag-eunhaui-MacBookPro DataIntegrationAndAnalysis % python entityResolution.py
Our accuracy is: 0.9818181818181818
```

Task 1 Implementation >

Step1 : Prepare data

```
def prepare_data(yelp_data, zomato_data):
    # Drop ratings and number of Reviews
    yelp_data = yelp_data.drop(columns=["RATING","NO_OF_REVIEWS'
    zomato_data = zomato_data.drop(columns=["RATING","NO_OF_REVI

    # Drop duplicates (row of same content)
    yelp_data = yelp_data.drop_duplicates()
    zomato_data = zomato_data.drop_duplicates()
    # name string lower
    yelp_data['NAME'] = yelp_data['NAME'].map(str.lower)
    zomato_data['NAME'] = zomato_data['NAME'].map(str.lower)


    # address string lower
    yelp_data['ADDRESS'] = yelp_data['ADDRESS'].map(str.lower)
    zomato_data['ADDRESS'] = zomato_data['ADDRESS'].map(str.lowe

    # remove , and . from address
    yelp_data['ADDRESS'] = yelp_data['ADDRESS'].str.replace(",",
    zomato_data['ADDRESS'] = zomato_data['ADDRESS'].str.replace(
    yelp_data['ADDRESS'] = yelp_data['ADDRESS'].str.replace(".",
    zomato_data['ADDRESS'] = zomato_data['ADDRESS'].str.replace(


    #regular expresion of phone number
    standard = re.compile('\(\d\d\d\)\s\d\d\d\-\d\d\d\d')

    for datasets in [yelp_data, zomato_data]:
        for index, row in datasets.iterrows():
            #check typo in phonenumber
            if standard.match(row['PHONENUMBER']) == None:
                datasets.drop(index,inplace=True)



    return yelp_data, zomato_data
```

1) We considered "RATING","NO_OF_REVIEWS" columns as less important and dropped them because they were not relevant to entity resolution.

2) We dropped duplicates that are the rows of same content

3) We changed values of “NAME”,”ADDRESS” columns into lowercase

4) We removed , and . from address to make easier matching

5) If certain rows of phone number didn’t match with regular expression of phone number we dropped them. (Because in Labeled Dataset there were no restaurants who have unusual phone number)

Step2 : Blocking

```
#Step 2: Blocking scheme
def blocking(df):
    blocks = collections.defaultdict(list)
    for index, row in df.iterrows():
        title = row['NAME'][:3]
        blocks[title].append(row)

    return blocks
```

In view of Labeled dataset, we thought **ltable** and **rtable** names are very similar when the column value of gold is 1.

So we decided to use blocking key as “**first three characters of the name**”.

Step 3 : Identify and delete the duplicates with in each Block

We made yelp and zomato blocks and delete duplicate within each blocks.

```
block_yelp = del_duplicate(block_yelp)
block_zomato = del_duplicate(block_zomato)
```

```
def del_duplicate(blocks):
    for key,block in blocks.items():
        pair = list(combinations(block,2))
        del_list = set()
        for y, z in pair:
            score = SequenceMatcher(None, y['NAME'], z['NAME'])
            score += SequenceMatcher(None, y['PHONENUMBER'], z[
            score += SequenceMatcher(None, y['ADDRESS'], z['ADDR
            if score >= 2.5:
                del_list.add(z['ID'])
        new_block = []
        for row in block:
            if row['ID'] in del_list:
                continue
            new_block.append(row)
        blocks[key] = new_block

    return blocks
```

Within each block, we made combinations of pairs.

And we get ‘NAME’ and ‘PHONENUMBER’ and ‘ADDRESS’ similarity score of pair using SequenceMatcher.

If score exceed certain threshold(2.5) , we consider both pair contains same entity.

So we put one entry of a pair to the deletion list.

Except for entity in the deletion list, we make new block with existing entities.

It return blocks which deletes duplicates in each block.

Step 4: Find perfect matches and report accuracy

```
for key, block in block_yelp.items():
    if key in block_zomato:
        find_perfect_match(block_zomato[key],block)
```

Our goal is to do entity resolution. So if the blocking key is in both block_yelp and block_zomato we call the *find_perfect_match* function.

```
#block 1-> zomato block2-> yelp
def find_perfect_match(block1,block2):
    pair = list(product(block1,block2))
    for y, z in pair:
        score = SequenceMatcher(None, y['NAME'], z['NAME']).rati
        score += SequenceMatcher(None, y['PHONENUMBER'], z['PHON
        score += SequenceMatcher(None, y['ADDRESS'], z['ADDRESS
        if score >= 2.1:
           # print(y['ID'])
            match_list.add((int(y['ID'])-1450000000000,int(z['ID
        if score <2.1:
            no_match_list.add((int(y['ID'])-1450000000000,int(z
```

In this function, as Step3: del_duplicate function , it compares 'NAME' and 'PHONENUMBER' and 'ADDRESS' of each pair(product of block yelp and block zomato with same key)

And they get similarity score using SequenceMatcher.

But we now decide our threshold as 2.1(according to experiments) and consider the pair is matched if exceeds threshold.

Otherwise, if the score was less than threshold we consider them as no_match.

Then we compute the accuracy.

```
def compute_accuracy():
    correct = 0
    wrong = 0
    labeled_data = pd.read_csv("restaurants1/csv_files/labeled_d
    for e in match_list:
        ltable_id ,rtable_id = e
        condition = ((labeled_data['ltable._id']==ltable_id ) &
        if labeled_data[condition].empty==1:
            continue
        else:
            selected_rows = labeled_data[condition].iloc[0]
            #print(selected_rows)

            if selected_rows['gold']==1:
                #print("correct")
                correct +=1
            elif selected_rows['gold']==0:
                wrong +=1

    for e in no_match_list:
        ltable_id ,rtable_id = e
        condition = ((labeled_data['ltable._id']==ltable_id )& 
        if labeled_data[condition].empty==1:
            continue
        else:
            selected_rows = labeled_data[condition].iloc[0]
            #print(selected_rows)

            if selected_rows['gold']==1:
                wrong +=1
            elif selected_rows['gold']==0:
                correct +=1

    accuracy = float(correct/(correct+wrong))
    print("Our accuracy is: " + str(accuracy))
```

For each entry in match_list , we firstly check that pair exists in Labeled dataset.

If it exists and gold value is one, we made correct matching. But if the gold value is zero it was wrong matching.

For each entry in no_match_list, we also firstly check that pair exists in Labeled dataset.

If it exists and gold value is zero we made correct matching. But if the gold value is one it was wrong matching.

And we get accuracy(correct/(correct+wrong)) of **0.9818181818**! (It takes about 1minute)

p.s.>

We thought that the main goal of task 01 is **entity resolution**.
So we think it is very important to find matching that has gold 1 value in Labeled dataset .

We found that number of our match_list entry that has gold 1 in labeled dataset is 108!

And there is only 126 entries that has gold 1 values in Labeled dataset.

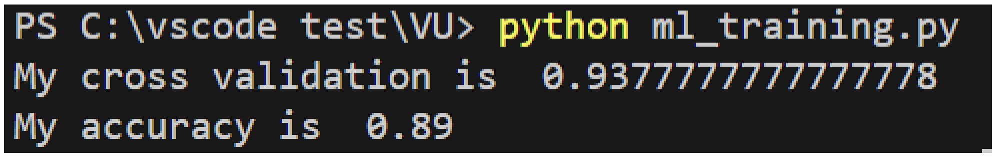
So we got almost 85% of matching entries. (If you think this is accuracy, we got almost 85%)

2. Task 2

Reproducing Results>

Run ml_training.py then you can get cross validation accuracy and prediction accuracy!!

```
python ml_training.py
```



```
PS C:\vscode test\VU> python ml_training.py
My cross validation is  0.9377777777777778
My accuracy is  0.89
```

Task 2 Implementation>

We used Support Vector machine(SVM) for machine learning classifier and we used labeled_data.csv from the website for training the model.

Step 1. Prepare the data and train the model

```
def prepare_input(data):
    train_X = []
    for x in data:
        name_score = SequenceMatcher(None, x[0], x[2]).ratio()
        address_score = SequenceMatcher(None, x[1], x[3]).ratio
        train_X.append([name_score, address_score])
    return train_X

train_data = pd.read_csv("restaurants1/csv_files/labeled_data.cs
raw_train_X = train_data.drop(columns=["_id","ltable._id","ltabl
x = prepare_input(raw_train_X)
y = list(train_data['gold'].values)

svm_model = svm.SVC(C=1, kernel = 'linear', gamma="scale")
svm_model.fit(x,y)
```

Before training the model, we cleaned data for the model. First, we dropped useless column for training such as _id, ltable._id, ltable._PHONENUMBER, rtable._id, rtable.PHONEMUMBER and gold column. After that, we calculated the similarity between name and address of each table(right and left table). We used the similarity of name and address between each table and trained the model to find the threshold of gold.

Step 2. Try different hyper-parameter and report cross validation accuracy.

```
svm_model = svm.SVC(C=1, kernel = 'poly', gamma="scale")
svm_model.fit(x,y) # training using labeled_data.csv
scores = cross_val_score(svm_model, x, y, cv=3)
print('My cross validation is ', scores.mean())
```

We changed kernel type and regularization prameter. When the hyper-parameter is poly type and regularization parmeter = 1, it was the hisgest validation accuracy(0.96). If we choose the linear type for kernel type, the validation accuracy was 0.937 which is lower than poly kernel type.

Step 3. Predict the instances in PredictX.csv

```
prediction = svm_model.predict(predicting_x)
print("My accuracy is ", mt.accuracy_score(list(prediction),lis
```

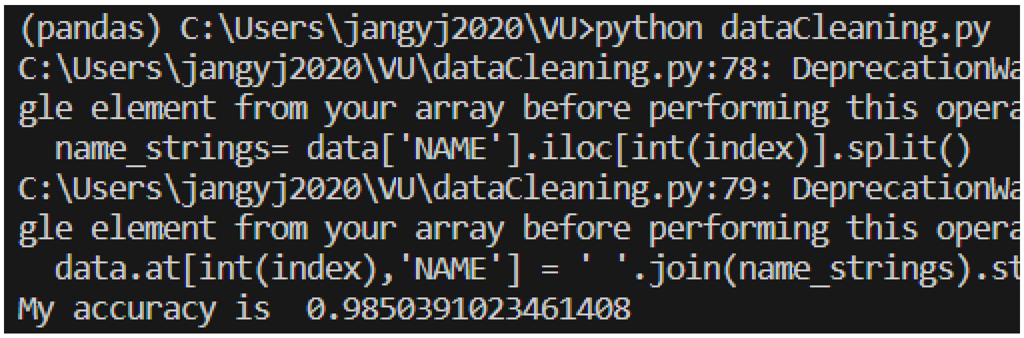
After training, we predict the the instances and get the accuracy of **0.865** by comparing the prediction and ground true data.

3. Task 3

Reproducing Results>

Run dataCleaning.py then you can get accuracy!!

```
python dataCleaning.py
```



```
(pandas) C:\Users\jangyj2020\VU>python dataCleaning.py
C:\Users\jangyj2020\VU\dataCleaning.py:78: DeprecationWa
gle element from your array before performing this opera
  name_strings= data['NAME'].iloc[int(index)].split()
C:\Users\jangyj2020\VU\dataCleaning.py:79: DeprecationWa
gle element from your array before performing this opera
  data.at[int(index),'NAME'] = ' '.join(name_strings).st
My accuracy is  0.9850391023461408
```

Task 3 Implementation >

Cleaning Policy

We found some pattern between yelp.csv and yelp_error.csv.

1. Change “!” to “i” in yelp_error.csv
2. Change “$” to “s” in yelp_error.csv
3. Change “@” to “a” in yelp_error.csv

```
def data_cleaning(data):

    for n in ['NAME', 'ADDRESS']:
        data[n] = data[n].str.replace("!", "i")
        data[n] = data[n].str.replace("$", "s")
        data[n] = data[n].str.replace("@", "a")
...
```

4. In NAME and ADDRESS cloumn, if the word is not a number like building number etc, change “3” to “e” and “9” to “g”.
5. Add one blank behind all restaurant’s name and compare with yelp.csv and delete one blank behind of restaurant’s name which is different with yelp.csv

```
def data_cleaning(data):
    ...
    cnt = 0
    for i, row in data.iterrows():
        address_strings = row['ADDRESS'].split()
        for arr_index, x in enumerate(address_strings):
            if x=='&' or n_pattern.match(x):
                continue
            else :
                address_strings[arr_index]=address_strings[ar
            if (x.isdigit()):
                continue
            else :
                next = 0
                for pattern in patterns:
                    if pattern.match(address_strings[arr_inde
                        next = 1
                if (next==1):
                    continue
                else:
                    address_strings[arr_index]=address_string
                    address_strings[arr_index]=address_string

                name_strings = row['NAME'].split()
        for arr_index, x in enumerate(name_strings):
            if x=='&' or n_pattern.match(x):
                continue
            else :
                name_strings[arr_index]=name_strings[arr_inde
            if(x.isdigit()):
                continue
            else :
                next = 0
                for pattern in patterns:
                    if pattern.match(name_strings[arr_index]
                        next = 1
                if (next==1):
                    continue
                else:
                    name_strings[arr_index]=name_strings[arr_
                    name_strings[arr_index]=name_strings[arr_
        data.at[i,'ADDRESS'] =' '.join(address_strings).stri
        data.at[i,'NAME'] =' '.join(name_strings).strip()+'

    return data

def data_cleaning_2(data,rows):
    for row_num in rows:
        index= data[data['ID']==row_num].index.values
        name_strings= data['NAME'].iloc[int(index)].split()
        data.at[int(index),'NAME'] = ' '.join(name_strings).s
    return data
```

Calculate accuracy

We calculate the ratio of difference row number before and after data cleaning.

number of total error(A): the number of different row between yelp.csv and yelp_error.csv before data cleaning

number of dirty(B): the number of different row between yelp.csv and yelp_error.csv after data cleaning

```
accuracy = (A-B)/A
```