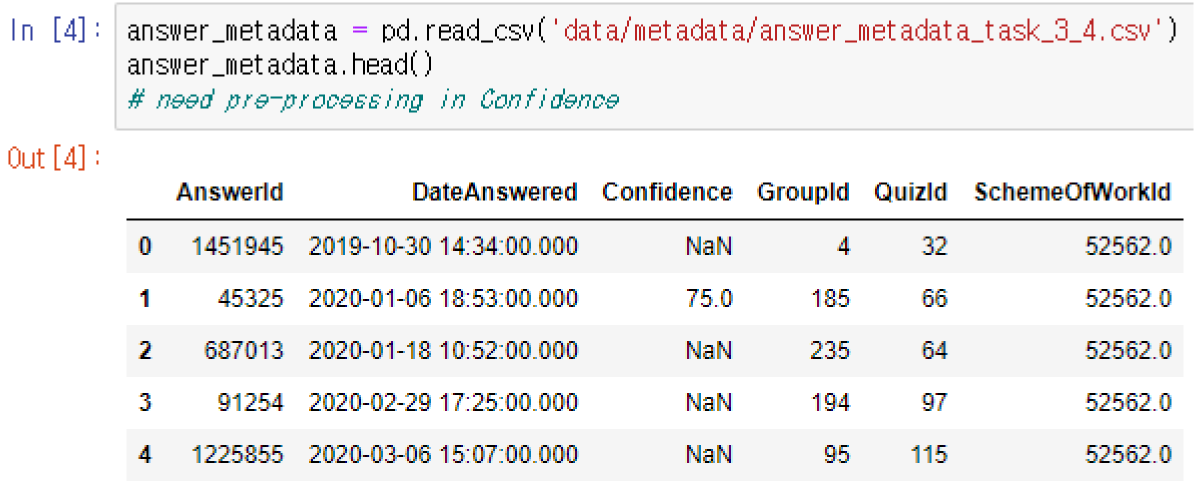
1. Data exploration and preprocessing.

When looking at the metadata, some values are missing in the ‘confidence; column, Therefore, preprocessing for this item is necessary.

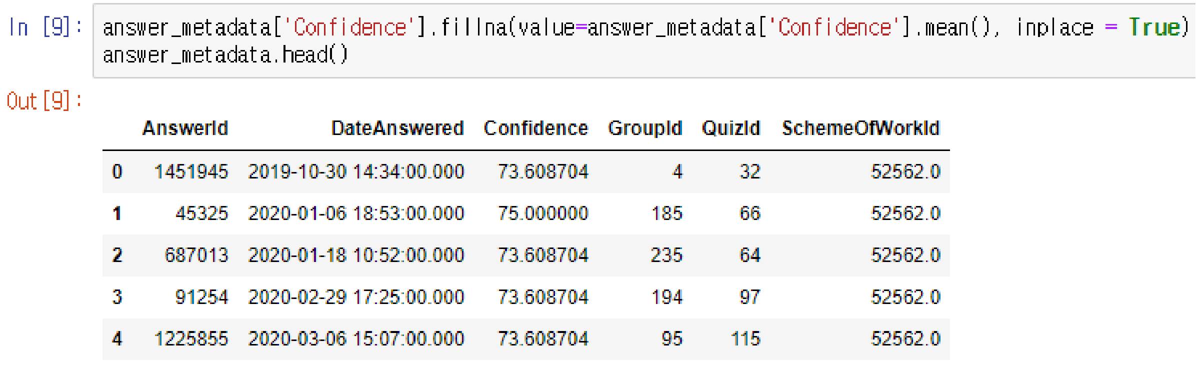


```
In [4]: answer_metadata = pd.read_csv('data/metadata/answer_metadata_task_3_4.csv')
        answer_metadata.head()
        # need pre-processing in Confidence
```

Out [4]:

	AnswerId	DateAnswered	Confidence	GroupId	QuizId	SchemeOfWorkId
0	1451945	2019-10-30 14:34:00.000	NaN	4	32	52562.0
1	45325	2020-01-06 18:53:00.000	75.0	185	66	52562.0
2	687013	2020-01-18 10:52:00.000	NaN	235	64	52562.0
3	91254	2020-02-29 17:25:00.000	NaN	194	97	52562.0
4	1225855	2020-03-06 15:07:00.000	NaN	95	115	52562.0

answer_metadata['Confidence'].fillna(value=answer_metadata['Confidence'].mean(),inplace=True): This line fills the missing values in the "Confidence" column with the mean value of that column. The **fillna** function is used to replace the missing values. The **value** parameter is set to **answer_metadata['Confidence'].mean()** which calculates the mean of the "Confidence" column. The **inplace=True** parameter ensures that the changes are made directly to the DataFrame without creating a new copy.



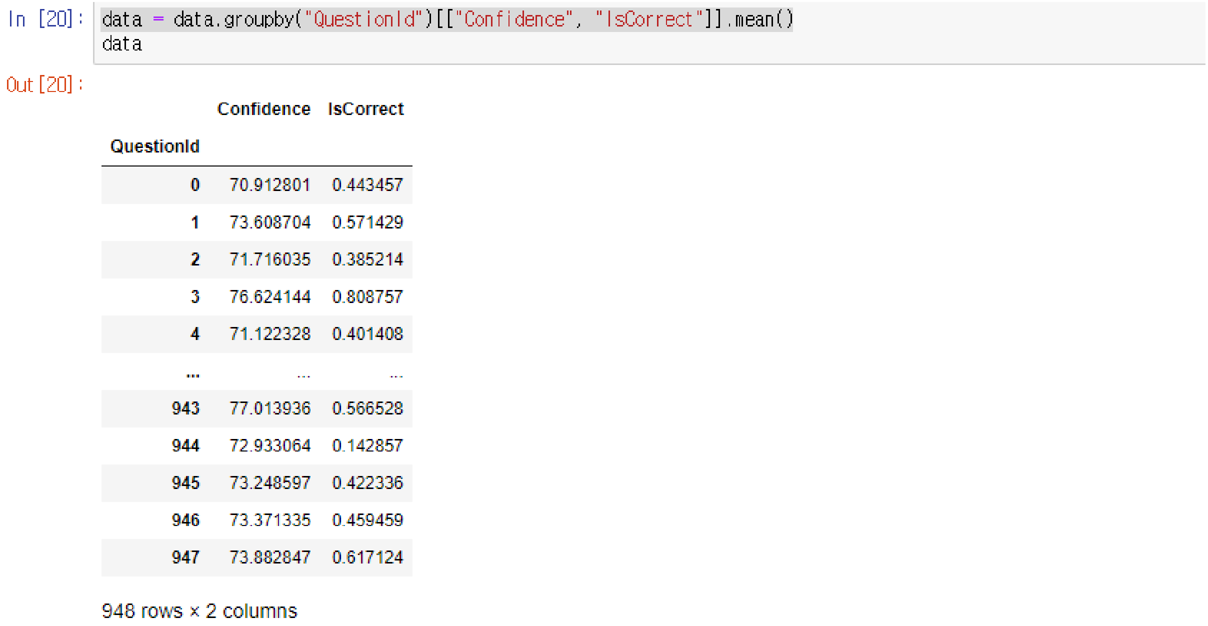
```
In [9]: answer_metadata['Confidence'].fillna(value=answer_metadata['Confidence'].mean(), inplace = True)
        answer_metadata.head()
```

Out [9]:

	AnswerId	DateAnswered	Confidence	GroupId	QuizId	SchemeOfWorkId
0	1451945	2019-10-30 14:34:00.000	73.608704	4	32	52562.0
1	45325	2020-01-06 18:53:00.000	75.000000	185	66	52562.0
2	687013	2020-01-18 10:52:00.000	73.608704	235	64	52562.0
3	91254	2020-02-29 17:25:00.000	73.608704	194	97	52562.0
4	1225855	2020-03-06 15:07:00.000	73.608704	95	115	52562.0

2. Calculate quality by merging and grouping the data

data = data.merge(answer_metadata, on="AnswerId", how="left"): This line merges the "data" DataFrame with the "answer_metadata" DataFrame based on a common column named "AnswerId". The **merge()** function is used to perform the join operation. The **on** parameter specifies the column used for matching the data, which in this case is "AnswerId". The **how** parameter is set to "left", indicating that all the rows from the left DataFrame ("data") will be included in the merged DataFrame, while matching rows from the right DataFrame ("answer_metadata") will be added if available. If there are no matching rows in "answer_metadata" for a particular "AnswerId" in "data", the columns from "answer_metadata" will have missing values in the merged DataFrame.



In [20]:

```
data = data.groupby("QuestionId")[["Confidence", "IsCorrect"]].mean()
data
```

Out [20]:

	Confidence	IsCorrect
QuestionId		
0	70.912801	0.443457
1	73.608704	0.571429
2	71.716035	0.385214
3	76.624144	0.808757
4	71.122328	0.401408
...	...	...
943	77.013936	0.566528
944	72.933064	0.142857
945	73.248597	0.422336
946	73.371335	0.459459
947	73.882847	0.617124

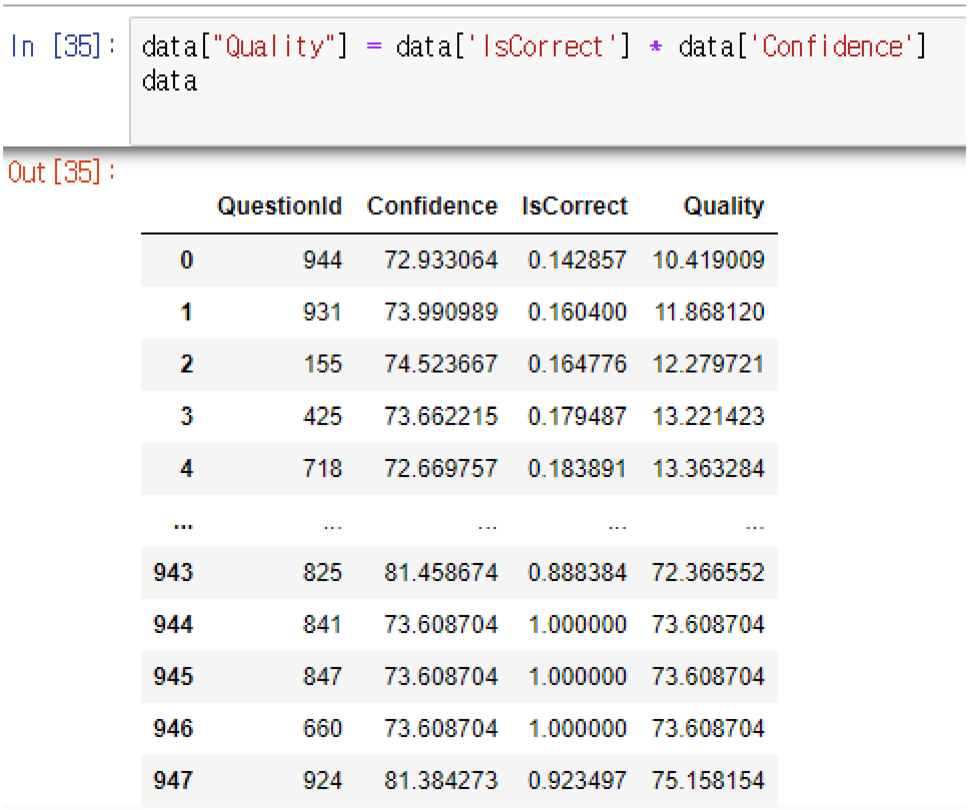
948 rows × 2 columns

data['IsCorrect']: This retrieves the values from the "IsCorrect" column of the "data" DataFrame. Assuming the "IsCorrect" column contains binary values (0 or 1) indicating whether an answer is correct or not.

data['Confidence']: This retrieves the values from the "Confidence" column of the "data" DataFrame. Assuming the "Confidence" column contains numerical values representing the confidence level of the answer.

data['IsCorrect'] * data['Confidence']: This performs element-wise multiplication between the values in the "IsCorrect" and "Confidence" columns. Since "IsCorrect" contains binary values (0 or 1), multiplying it by the "Confidence" values effectively gives higher weight to correct answers (where "IsCorrect" is 1) and lower weight to incorrect answers (where "IsCorrect" is 0). The result of the multiplication will be stored in the new "Quality" column.

data["Quality"] = ...: This assigns the resulting values from the multiplication to a new column called "Quality" in the "data" DataFrame.



```
In [35]: data["Quality"] = data['IsCorrect'] * data['Confidence']
         data
```

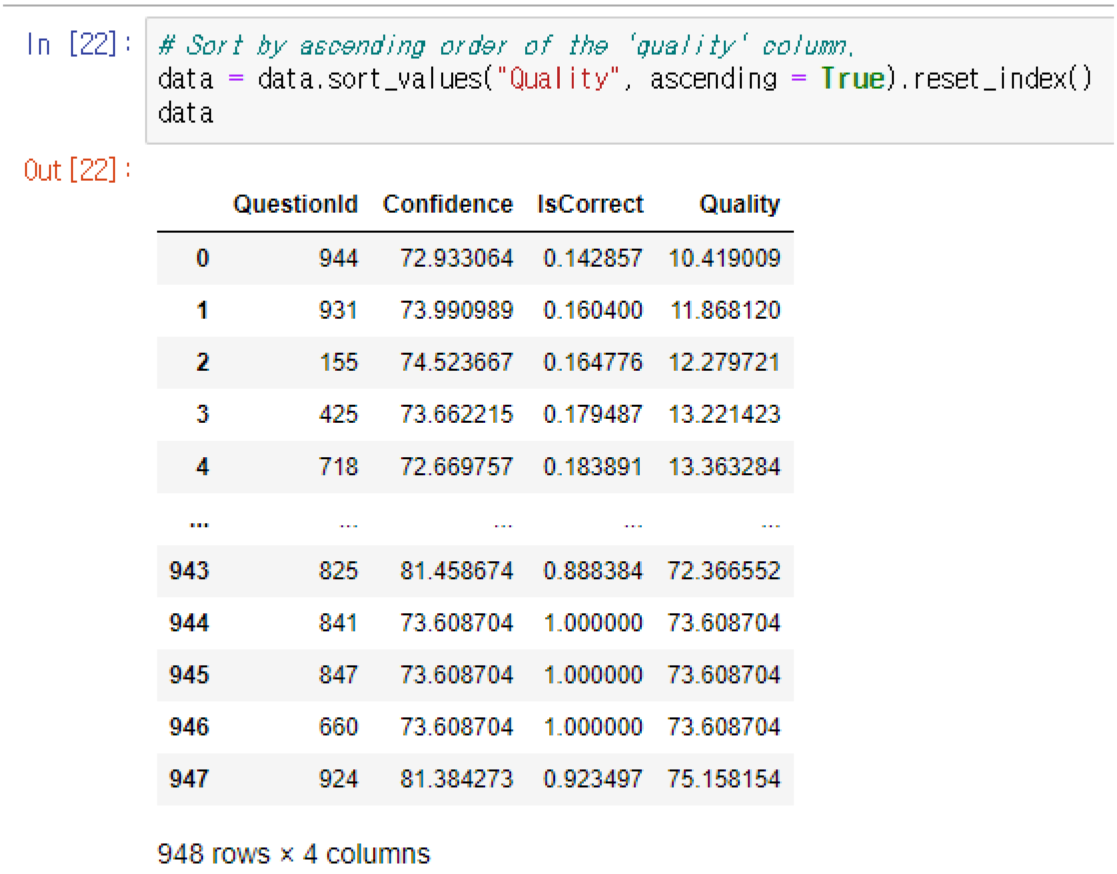
Out[35]:

	QuestionId	Confidence	IsCorrect	Quality
0	944	72.933064	0.142857	10.419009
1	931	73.990989	0.160400	11.868120
2	155	74.523667	0.164776	12.279721
3	425	73.662215	0.179487	13.221423
4	718	72.669757	0.183891	13.363284
...	...	...	...	...
943	825	81.458674	0.888384	72.366552
944	841	73.608704	1.000000	73.608704
945	847	73.608704	1.000000	73.608704
946	660	73.608704	1.000000	73.608704
947	924	81.384273	0.923497	75.158154

3. Sorting Quality

data = data.sort_values("Quality", ascending=True): This line sorts the DataFrame "data" based on the values in the "Quality" column. The **sort_values()** function is used to perform the sorting operation. The column to sort by is specified with the **by** parameter, which in this case is "Quality". The **ascending** parameter is set to **True**, indicating that the values should be sorted in ascending order.

.reset_index(): This line resets the index of the DataFrame "data" after sorting. By default, the index values are reassigned to a new range starting from 0. The **reset_index()** function is used to reset the index.



In [22]:

```
# Sort by ascending order of the 'quality' column.
data = data.sort_values("Quality", ascending = True).reset_index()
data
```

Out [22]:

	QuestionId	Confidence	IsCorrect	Quality
0	944	72.933064	0.142857	10.419009
1	931	73.990989	0.160400	11.868120
2	155	74.523667	0.164776	12.279721
3	425	73.662215	0.179487	13.221423
4	718	72.669757	0.183891	13.363284
...	...	...	...	...
943	825	81.458674	0.888384	72.366552
944	841	73.608704	1.000000	73.608704
945	847	73.608704	1.000000	73.608704
946	660	73.608704	1.000000	73.608704
947	924	81.384273	0.923497	75.158154

948 rows × 4 columns

Reads a template CSV file into a DataFrame called "submission", then performs a loop over the rows of another DataFrame called "data". For each row in "data", it finds the index of the first occurrence where the "QuestionId" column matches the current loop index "i", and assigns this index value to the "ranking" column in the "submission" DataFrame. Finally, it converts the "ranking" column to integers, and saves the "submission" DataFrame to a new CSV file. Here is a breakdown of what each line does:

submission = pd.read_csv('data/submission/template.csv'): This line reads a template CSV file named 'template.csv' from the 'data/submission/' directory into a DataFrame called "submission". The **pd.read_csv()** function from the pandas library is used to read the CSV file.

for i in range(len(data)):: This line initiates a loop over the range of values from 0 to the length of the DataFrame "data".

index = data[data['QuestionId'] == i].index: This line filters the rows of the DataFrame "data" based on the condition where the value in the "QuestionId" column equals the current loop index "i". It then retrieves the index of the filtered rows and assigns it to the variable "index".

submission.loc[i, 'ranking'] = index[0]: This line assigns the value of the first element in the "index" variable (the index of the first occurrence where the condition is satisfied) to the "ranking" column in the "submission" DataFrame at the current loop index "i".

submission['ranking'] = submission['ranking'].astype('int'): This line converts the "ranking" column in the "submission" DataFrame to integer data type using the **astype()** function. This step ensures that the column contains integer values.

submission.to_csv('data/submission/20172656.csv'): This line saves the "submission" DataFrame to a new CSV file named '20172656.csv' in the 'data/submission/' directory. The **to_csv()** function from the pandas library is used to write the DataFrame to a CSV file.

I will attach the csv file separately.

Github: https://github.com/blueeye09/Machine_Learning