data_transformation&analysis_python

March 13, 2024

1 Preprocessing, wrangling, file creation, and analysis in Python (everything except IRT)

1.0.1 Load required libraries

[1]:
```
# -*- coding: utf-8 -*-
"""
Created on Thu March 7 19:09:06 2024
@author: brian_local
"""
import pandas as pd
import numpy as np
import re
from scipy.stats import mannwhitneyu
import statsmodels.api as sm
```

1.0.2 Import data that was cleaned in Excel, one frame for main data, one for question-to-tag correspondence

[2]:
```
df_main = pd.read_excel('original_data/data.xlsx', sheet_name = 0)
df_tags = pd.read_excel('original_data/data.xlsx', sheet_name = 1)
```

1.0.3 Create calculated fields

- Note assumption in creating passing score Boolean variable that pass percentage rounds to the nearest whole percent, i.e. that 91.6% (120/131) passes

[3]:
```
df_main['score'] = df_main.loc[:,'q01-score':'q14-score'].sum(axis=1)
df_main['score_pct'] = (df_main['score']/df_main['max_score']).round(2)
df_main['total_q_time'] = df_main.loc[:,'q01-time':'q14-time'].sum(axis=1)
df_main['score_pass_bool'] = np.where(df_main['score_pct']>=.92, 1, 0)
df_main['copy_paste_bool'] = np.where(df_main['copy_paste_ct']>0, 1, 0)
```

1.0.4 Identify duration columns (those that include 'time') and convert to hours for easier reading

- Also created a starttime variable in the hopes that the time part of 'Attempt starttime' is legit

[4]:
```
time_columns = df_main.filter(like='time').columns
df_main[time_columns] = df_main[time_columns] / 3600

df_main['attempt_time_of_day'] = df_main['attempt_start_dt'].dt.time
```

1.0.5 Create three data frames and corresponding csv files: one with all clean data, one with test-level only, one with question-level only

[5]:
```
pattern = re.compile(r'^q\d+')
test_columns = [col for col in df_main.columns if not pattern.match(col)]
quest_columns  = [col for col in df_main.columns if pattern.match(col)]

df_main.to_csv('clean_main_data.csv')
df_test = df_main[test_columns]
df_test.to_csv('clean_test_only_data.csv')
```

1.0.6 Removing Question 1 from further analysis

- Worth 1/10th of all other questions
- Stands apart as only questions involving DDL/DML instead of DQL SQL
- Appears to involve just running a script or two
- Can't impact pass fail (as long as percents are rounded)
- Admittedly, especially given missing values for Q1, simplifies analysis significantly

[6]:
```
df_quest = df_main[quest_columns].iloc[:,1:].copy()
df_quest['id'] = df_main['cand_id']
df_quest.to_csv('clean_quest_only_data.csv')
```

1.0.7 Question-level data transformation

- Separating questions level data by category (score, duration, lines of code, compile counts)
- Melting each to long form
- Reassembling all categories
- Joining ("merging" in pandas) to M:N question "tags" for tag-level analysis

[7]:
```
score_columns = [col for col in df_quest.columns if 'score' in col]
time_columns = [col for col in df_quest.columns if 'time' in col]
loc_columns = [col for col in df_quest.columns if 'loc' in col]
compile_columns = [col for col in df_quest.columns if 'compile' in col]

df_scores = df_quest[score_columns].copy()
df_times = df_quest[time_columns].copy()
df_locs = df_quest[loc_columns].copy()
df_compiles = df_quest[compile_columns].copy()

df_scores['id'] = df_quest['id']
df_times['id'] = df_quest['id']
df_locs['id'] = df_quest['id']
```

```
df_compiles['id'] = df_quest['id']

df_scores_melted = pd.melt(df_scores, id_vars=['id'], var_name='question',␣
 ↪value_name='score')
df_times_melted = pd.melt(df_times, id_vars=['id'], var_name='question',␣
 ↪value_name='time')
df_locs_melted = pd.melt(df_locs, id_vars=['id'], var_name='question',␣
 ↪value_name='loc')
df_compiles_melted = pd.melt(df_compiles, id_vars=['id'], var_name='question',␣
 ↪value_name='compile&test_ct')

df_scores_melted['question'] = df_scores_melted['question'].str.
 ↪replace('-score', '')
df_times_melted['question'] = df_times_melted['question'].str.replace('-time',␣
 ↪'')
df_locs_melted['question'] = df_locs_melted['question'].str.replace('-loc', '')
df_compiles_melted['question'] = df_compiles_melted['question'].str.
 ↪replace('-compile&test_ct', '')


df_long1 = pd.merge(df_scores_melted, df_times_melted, on=['id', 'question'])
df_long2 = pd.merge(df_locs_melted, df_compiles_melted, on=['id', 'question'])
df_long = pd.merge(df_long1, df_long2, on=['id', 'question'])
df_long_w_tags = pd.merge(df_long, df_tags, left_on = "question", right_on =␣
 ↪"Qnum", how = "inner")
df_long.to_csv('clean_quest_long_data.csv')
df_long_w_tags.to_csv('clean_quest_w_tags_long_data.csv')
```

–end data transformation work–

1.1 Analysis of copy/paste and out-of-window time behavior's impact on scores

1.1.1 Treating Copy/Paste as Yes/No

[8]:
```
group1 = df_main[df_main['copy_paste_bool'] == 1]['score']
group2 = df_main[df_main['copy_paste_bool'] == 0]['score']
mean_group1 = group1.mean().round(2)
mean_group2 = group2.mean().round(2)
print(f'Mean of copy/pasters {mean_group1}\nmean of non-copy/pasters␣
 ↪{mean_group2}')
```

```
Mean of copy/pasters 122.76
mean of non-copy/pasters 104.1
```

1.2 Is this difference statistically significant?

- Normally would do a simple t-test

- Can't here, because the assumption of normality is completely violated (see Tableau viz of scores histogram)
- Log transformation – converting score to ln(score) – won't fix in this case because modes are at extremes
- Conduct nonparametric Mann-Whitney U test instead
- Hypothesis is that copy/pasters > non-copy/pasters at alpha = 0.05

[9]:
```
stat, p_value = mannwhitneyu(group1, group2, alternative='greater')
print(f'Mann-Whitney U Test Statistic: {stat}, P-value: {p_value}')
```

```
Mann-Whitney U Test Statistic: 68893.5, P-value: 2.244793123162359e-15
```

H0 that µgroup1 = µgroup2 is rejected at alpha = 0.05 There is statistical evidence that the copy-pasters score higher

1.3 Models that treat copy/paste and out of window counts as continuous

- Slightly dubious given that they are counts, esp. given copy/paste ct range
- Some stats traditions would be fine with it, others not

[10]:
```
x = sm.add_constant(df_main['copy_paste_ct'])
y = df_main['score']

model = sm.OLS(y, x).fit()

print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  score   R-squared:                       0.025
Model:                            OLS   Adj. R-squared:                  0.023
Method:                 Least Squares   F-statistic:                     16.25
Date:                Wed, 13 Mar 2024   Prob (F-statistic):           6.20e-05
Time:                        08:33:44   Log-Likelihood:                -3242.1
No. Observations:                 648   AIC:                             6488.
Df Residuals:                     646   BIC:                             6497.
Df Model:                           1
Covariance Type:            nonrobust
================================================================================
=
                    coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------------
-
const           108.9188      1.606     67.837      0.000     105.766
112.072
copy_paste_ct     0.3514      0.087      4.032      0.000       0.180
0.523
==============================================================================
Omnibus:                      270.353   Durbin-Watson:                   1.755
```

```
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              803.847
Skew:                          -2.135   Prob(JB):                    2.80e-175
Kurtosis:                       6.396   Cond. No.                         20.9
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

- The model and each of its coefficients are significant ($p < .001$)
- Each instance of copy/paste increases the expected score by over one third of one point.
- There is significant evidence of non-normally distributed error terms, however, which threatens an assumption of the test, and warrants further investigation

1.4 Adding a second independent variable for window exit count

[11]:
```
x2 = df_main[['copy_paste_ct', 'window_exit_ct']]

# Again add a constant term for the intercept
x2 = sm.add_constant(x2)

model = sm.OLS(y, x2).fit()
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  score   R-squared:                       0.045
Model:                            OLS   Adj. R-squared:                  0.042
Method:                 Least Squares   F-statistic:                     15.29
Date:                Wed, 13 Mar 2024   Prob (F-statistic):           3.24e-07
Time:                        08:33:44   Log-Likelihood:                -3235.1
No. Observations:                 648   AIC:                             6476.
Df Residuals:                     645   BIC:                             6490.
Df Model:                           2
Covariance Type:            nonrobust
================================================================================
==
                     coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------------
--
const            105.3279      1.857     56.726      0.000     101.682
108.974
copy_paste_ct      0.3032      0.087      3.475      0.001       0.132
0.475
window_exit_ct     0.0555      0.015      3.743      0.000       0.026
0.085
==============================================================================
```

```
Omnibus:                      253.134   Durbin-Watson:                   1.762
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              696.310
Skew:                          -2.030   Prob(JB):                    6.28e-152
Kurtosis:                       6.051   Cond. No.                         159.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

- Very similar to univariate copy/paste OLS above, with effect divided between copy/paste and window exit counts

1.5 Bottom Lines

- There is significant evidence that copy/pasting is associated with improved scores
- Plagiarism is clearly a concern given all the (invariant, negative) plagiarism columns in raw data
- **IF** copy/paste plagiarism is a prospective issue, further investigation is warranted
- I routinely see my homework problem set questions asked and (less frequently) answered on stack overflow
- Not to mention, e.g., Chegg, course hero, braindump sites, etc.

2 Preliminary, Basic Item-Level Stats

Question Difficulty

[12]:
```
df_scores.drop('id',axis=1, inplace = True)
agg = df_scores.agg('sum')/(df_scores.agg('count')*10)
agg['total-score'] = agg.values.mean()
df_agg = agg.to_frame(name="mean").round(3)
df_agg['rank'] = agg.rank()



print(df_agg)
```

```
              mean  rank
q02-score    0.924  14.0
q03-score    0.920  12.0
q04-score    0.921  13.0
q05-score    0.909  11.0
q06-score    0.861   8.0
q07-score    0.847   3.0
q08-score    0.852   4.5
q09-score    0.887  10.0
q10-score    0.886   9.0
q11-score    0.852   4.5
q12-score    0.860   7.0
q13-score    0.631   1.0
```

```
q14-score    0.773   2.0
total-score  0.856   6.0
```

2.1 Super-Simple Question Discrimination

- How well the question helps identify high performers from low performers
- LOTS of ways to do this, some sophisticated (to be reviewed later time permitting)
- One of the simplest: how does question correlate with overall test score?

[13]:
```
item_to_total_correlations = df_scores.corrwith(df_main['score'], method =
 ↪'spearman')
print(item_to_total_correlations)
```

```
q02-score    0.472960
q03-score    0.488587
q04-score    0.481544
q05-score    0.504897
q06-score    0.610922
q07-score    0.634638
q08-score    0.618950
q09-score    0.564130
q10-score    0.570847
q11-score    0.620656
q12-score    0.614843
q13-score    0.838059
q14-score    0.728214
dtype: float64
```

- Note use of Spearman's rho because score data are ordinal scale
- Note further that use of full total score for correlation to individual question scores double counts each question's contribution
- Thus, compute and adjusted total by subtracting each question from the overall total when computing that question's correlation, as follows:

2.1.1 Adjusted question discrimination correlation

[14]:
```
df_adj_correl_totals = abs(df_scores.sub(df_main['score'], axis=0))
adj_item_to_total_correlations = df_scores.corrwith(df_adj_correl_totals,
 ↪method = 'spearman')
print(adj_item_to_total_correlations)
```

```
q02-score    0.468658
q03-score    0.486626
q04-score    0.476920
q05-score    0.491664
q06-score    0.597306
q07-score    0.619549
q08-score    0.586859
q09-score    0.558504
```

```
q10-score    0.567403
q11-score    0.592348
q12-score    0.600645
q13-score    0.558598
q14-score    0.663751
dtype: float64
```

2.2 Bottom Line for Basic Item-Level Stats

- Don't reveal all that much
- Questions on the whole seem a bit on the easy order relative to taker capacity, which could be good or bad depending on nature of test
- One potential counterexample to the 80/20 rule: More sophisticated item-level analysis through Item Response Theory (IRT) may reveal more. See IRT notebook for details