Research Article

BIOMEDICAL Journal of Scientific & Technical Research

ISSN: 2574 -1241 DOI: 10.26717/BJSTR.2023.53.008448

Appendix 2-6

Appendix 2

Model 2.1—— Key Python Code for Z-Score Evaluation Model

```
#Model 2.1
# Z-Score Evaluation Model
# Import the required libraries
# read the attachment data, extract the original scores of each expert and each work
data = pd.read_excel('data.xlsx')
data = data.iloc[:351, [1,3,5,7,9,11,13,15]] # only keep the original score column
data.columns = ['expert1', 'expert2', 'expert3', 'expert4', 'expert5','expert6', 'expert7', 'expert8'] # rename the column name
data.index = ['work' + str(i) for i in range(1, 352)] # rename the column name
print(data.head()) # view the first five rows of data
# Calculate the standard score and Z-score of each expert and each work
# define a function, according to the formula in Attachment 1
def standard_score(x):
    mean = x.mean() # calculate the average,
    std = x.std() # calculate the standard deviation
    return (x - mean) / std * 10 + 50 # return the standard score
# define a function, calculate the Z-score of each expert
def z_score(x):
    mean = data.values.mean() # calculate the mean of the scores of all works
    std = data.values.std() # calculate the standard deviation of the scores of all works
    return (x - mean) / std # return the Z-score
#Convert the scores of each expert to standard scores and Z-scores
print(data.head())
 # Sort the works according to two schemes , get the ranking of the works
# Standard score scheme
data['std_total'] = data[['std_score1', 'std_score2', 'std_score3', 'std_score4', 'std_score5','std_score6', 'std_score7', 'std_score8']].sum(axis=1)
# Calculate the total standard score of each work
data['std_rank'] = data[' std_total '].rank(ascending=False, method='min') # Calculate the standard score ranking of each work, in descending order of total score, take the smallest ranking for the same total score
```

```
data = data.sort_values(by=' std_rank ') # Sort by ranking

print(data.head())

# Z-score scheme

data['Z-total'] = data[['Z-score1', 'Z-score2', 'Z-score3', 'Z-score4', 'Z-score5','Z-score6', 'Z-score7', 'Z-score8']].sum(axis=1) # Calculate the total Z-score of each work

data['Z-rank'] = data[' Z-total '].rank(ascending=False, method='min') # Calculate the Z-score ranking of each work, in descending order of total score, take the smallest ranking for the same total score

data = data.sort_values(by='Z-rank') # Sort by ranking

print(data.head())

#  Compare the ranking results of the two schemes, analyze the changes and differences in ranking

# Calculate the correlation coefficient of the ranking of the two schemes

corr = data[' std_rank '].corr(data[' Z-rank ']) # Use the Pearson correlation coefficient

print('  The correlation coefficient of the rankings of the two schemes is: ', corr)

#  Draw a scatter plot of the rankings of the two schemes

plt.scatter(data['std_rank'], data['Z-rank'], s=1) # se the scatter plot to show the relationship between the rankings of the two schemes

plt.xlabel('Standard score ranking ')

plt.ylabel('Z-score ranking ')

plt.title('scatter plot of the rankings of the two schemes ')

plt.rcParams['font.sans-serif'] = ['SimHei']

plt.show()

df = pd.DataFrame(data)

df.to_excel('output.xlsx', index=True)
```

Appendix 3

Model 2.2——Key Python Code for PCA Evaluation Model

```
# Read the attachment data table 1

file_path = 'd1.xlsx'

data = pd.read_excel(file_path)

data.head()

#Calculate the average of the original score and the standard score of the first stage

data[' The average of the original score of the first stage '] = data[['org1', 'org2', 'org3', 'org4', 'org5']].mean(axis=1)

data[' The standard score of the original score of the first stage '] = data[['std-score1', 'std-score2', 'std-score3', 'std-score4', 'std-score5']].mean(axis=1)

# Create a PCA model and fit the first stage review data

pca_first_stage = PCA(n_components=1)

pca_first_stage.fit(data[[' The average of the original score of the first stage ', ' The standard score of the original score of the first stage ']])
```

```
# Use the PCA model to transform the data and calculate the comprehensive standard score

data[' Comprehensive standard score of the first stage '] = pca_first_stage.transform(data[[' The average of the original score of the first stage ', ' The standard score of the original score of the first stage ']])

# Calculate the average of the original score and the standard score of the second stage

data['  average of the original score of the second stage '] = data[['std-score 6', ' std-score 7', ' std-score 8']].mean(axis=1)

data[' average of the standard score of the second stage, '] = data[['std-score 6', 'std-score 7', 'std-score 8']].mean(axis=1)

# Create a PCA model and fit the first stage review data

pca_first_stage = PCA(n_components=1)

pca_first_stage.fit(data[[' average of the original score of the second stage ', ' average of the standard score of the second stage ']])

# Use the PCA model to transform the data and calculate the comprehensive standard score data[' Comprehensive standard score of the second stage '] = pca_first_stage.transform(data[[' The average of the original score of the first stage ', ' The standard score of the original score of the first stage ']])

# Calculate the comprehensive standard score, which can be adjusted according to the needs alpha = 0.2  # First stage weight

beta = 0.8  # Second stage weight

data[' Comprehensive standard score '] = alpha * data[' Comprehensive standard score of the first stage '] + beta * data[' Comprehensive standard score of the second stage ']

# Sorting by Comprehensive Standard Score

# data["Comprehensive standard score ranking"]= data.sort_values(by='Comprehensive standard score', ascending=False)

data["PCA standard score ranking"]= data[' Comprehensive standard score '].rank(ascending=False, method='min')

# data1.head()

# print(sorted_data)

# # Save the processed results to a new Excel table

# data1.to_excel('pca_result.xlsx', index=False)

# Standard Score Scheme

data['std-total'] = data[['std-score1', 'std-score2', 'std-score3', 'std-score4', 'std-score5','std-score6', 'std-score7', 'std-score8']].sum(axis=1) # Calculate the total score for each work

data['std-rank'] = data['std-total'].rank(ascending=False, method='min') # Calculate the standard score ranking for each work, sort by total score in descending order, and choose the smallest rank for equal total scores

# data2 = data.sort_values(by='std-rank')  # Sort by ranking

# data2.to_excel('Standard_Result.xlsx', index=False)

# data2.head()

# file3_path = 'Standard_Result.xlsx'

# data4 = pd.read_excel(file3_path)

# data4.head()

# Compare the ranking results of the two schemes, analyze changes and differences in ranking

# Calculate the correlation coefficient between the rankings of the two schemes

corr = data['std-rank'].corr(data["PCA standard score ranking "]) # Using Pearson correlation coefficient
```

```
print('the correlation coefficient between the ranking of two schemes is:', corr)
# Plot a scatter plot of the rankings of the two schemes
plt.scatter(data['std-rank'], data["PCA standard score ranking "], s=1)
plt.xlabel(' Standard Score Ranking')
plt.ylabel(' PCA Standard Score Ranking')
plt.title(' Scatter Plot of Traditional Standard Score and PCA Ranking')
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.show()
```

Appendix 4

Model 2.3——Key Python Code for Fuzzy Evaluation Model

```
# Fuzzy evaluation model
#Read the data table
# Define the maximum and minimum scores for calculating membership function
m_score = 100
mi_score = 0
# Define the set of evaluation results for outputting the final evaluation
eval_set = ["First Prize", "Second Prize", "Third Prize", "Not Awarded"]
# Define the scoring weight of experts for calculating comprehensive fuzzy set
eval_weight = np.array([0.4, 0.3, 0.2, 0.1])
#Define the membership function for calculating the membership degree of each score
#Triangular fuzzy number membership function
def membership_function(score):
    if score < mi_score:
        return 0
    elif score <= (m_score + mi_score) / 2:
        return (score - mi_score) / (m_score - mi_score)
    elif score <= m_score:
        return (m_score - score) / (m_score - mi_score)
    else:
        return 0
#Calculate the fuzzy set for each work, representing the evaluation result of each work
Each fuzzy set is a one-dimensional array with a length equal to the length of the evaluation result set
Each element represents the membership degree of the evaluation result of this work
fuzzy_set = []
for i in range(len(eval_mat)): # Iterate through each work
```

```
    fuzzy_set.append([])# Create an empty fuzzy set
    for j in range(len(eval_set)): # Iterate through each evaluation result
      fuzzy_set[i].append(0) # Initialize the membership degree of this evaluation result to 0
      for k in range(len(eval_mat[i])): # Iterate through each expert's score
  # Calculate the membership degree of the expert's score
        membership = membership_function(eval_mat[i][k])
        # If the membership degree is greater than the current membership degree, update the membership degree of this evaluation result
  # This is equivalent to taking the maximum membership degree for each expert's score
        if membership > fuzzy_set[i][j]:
          fuzzy_set[i][j] = membership
  # Convert the fuzzy set to a numpy array for easier subsequent calculations
  fuzzy_set = np.array(fuzzy_set)
  #Calculate the comprehensive fuzzy set for each work, representing the comprehensive level of each work
  # Each comprehensive fuzzy set is a one-dimensional array with a length equal to the length of the evaluation result set
  # Each element represents the membership degree of the comprehensive evaluation result of this work
  # Weighted average method of comprehensive calculation
  fuzzy_set_com = np.dot(fuzzy_set, eval_weight)
  # Calculate the ranking of each work, used for outputting the final ranking
  # Each ranking is an integer, representing the relative position of this work among all works
  # Principle of maximum membership degree sorting method
  fuzzy_set_rank = np.argsort(-fuzzy_set_com) + 1
  #Output the evaluation result, comprehensive evaluation result, and ranking of each work
  for i in range(len(eval_mat)): # Iterate through each work
    print("The evaluation result of work%d is:" % (i + 1)) # Output the work number
    for j in range(len(eval_set)): # Iterate through each evaluation result
      print("%s: %.2f" % (eval_set[j], fuzzy_set[i][j])) # Output the evaluation result and membership degree
     print("The comprehensive evaluation result of work%d is:" % (i + 1)) # Output the work numberr
    for j in range(len(eval_set)): # Iterate through each evaluation result
      print("%s: %.2f" % (eval_set[j], fuzzy_set_com[i][j])) # Output the evaluation result and comprehensive membership degree
  print("The ranking of work%d is:%d" % (i + 1, fuzzy_set_rank[i])) # Output the work number and ranking
    print()# Output an empty line for easy reading
  # Analyze and solve the results, such as using charts, descriptive statistics, correlation analysis, etc., to display and explain the distribution characteristics, ranking situation, difference in evaluation scores of experts, etc
  #Import the matplotlib library to draw charts
```

Appendix 5

Model 3—— Key Python Code for "Extreme Difference" Model

```
# Read table data from the image
data = pd.read_excel("Data 2.1.xlsx")
# Calculate the average and range of scores in two stages of evaluation
data.columns = ['Expert 1', 'Expert 2', 'Expert 3', 'Expert 4', 'Expert 5', 'Expert 6', 'Expert 7', 'Expert 8']
data.index = [i for i in range(1, 241)]
data["Average Score in Stage 1"] = data.iloc[:, 1:6].mean(axis=1)
data["Average Score in Stage 2"] = data.iloc[:, 6:9].mean(axis=1)
data["Range in Stage 1"] = data.iloc[:, 1:6].max(axis=1) - data.iloc[:, 1:6].min(axis=1)
data["Range in Stage 2"] = data.iloc[:, 6:9].max(axis=1) - data.iloc[:, 6:9].min(axis=1)
print(data)
# Rename row indices
# Plot the average scores and ranges of scores in two stages as a line chart
plt.Figure(figsize=(10, 6))
plt.plot(data.index, data["Average Score in Stage 1"], label="Average Score in Stage 1", color="blue", marker="o")
plt.plot(data.index, data["Average Score in Stage 2"], label="Average Score in Stage 2", color="red", marker="s")
plt.plot(data.index, data["Range in Stage 1"], label="Range in Stage 1", color="green", marker="*")
plt.plot(data.index, data["Range in Stage 2"], label="Range in Stage 2", color="orange", marker="+")
plt.xlabel("Work Number")
plt.ylabel("Evaluation Score")
plt.title("Line chart of average scores and ranges in two stages of evaluation")
plt.legend()
plt.show()
```

Appendix 6

Model 4——Key Python Code for Difference Perception Review Model

```
################Data Preprocessing ##################
# Read data and randomly generate an ID
data = pd.read_excel('data_orginal.xlsx')
def random_code():
    return ''.join(random.choices(string.ascii_uppercase + string.digits, k=4))
data.insert(0, 'work-ID', data.apply(lambda x: random_code(), axis=1))
index_asd = data.columns.get_loc(''First Review Standard Score Range'')
n=5
```

```
# Insert a new column at the specified position with the name "Average Score of First Review Criteria"
data.insert(index_asd + 1, ' Mean value of the first review standard scores', (data['std-score1'] + data['std-score2']+ data['std-score3']+ data['std-score4']+ data['std-score5']) / n)
################## Statistical Analysis ##################
# Sort the entire data table in descending order based on the values of the "First Review Criteria Score Average" column
data.sort_values(by=' Mean value of the first review standard scores ', ascending=False, inplace=True)
# Calculate the total number of participating teams
all_group = data.shape[0]
print("Total number of teams:",all_group)
# Use the notnull() and sum() functions to find the number of teams that entered the second review comprehensive judgment scores
double_review= data[' second review standard score range '].notnull().sum()
print("number of teams that were subsequently evaluated and re-evaluated for their grades: ",double_review)
# Sort the first 'm' data in descending order
data.iloc[:double_review] = data.iloc[:double_review].sort_values(by=' Final scores ', ascending=False)
# Concatenate the sorted data table and the original data table starting from row 353
data= pd.concat([data.iloc[:double_review], data.iloc[double_review:]])
# Calculate the total number of teams that did not win
unnominated_teams = len(data[data['award'] == ' Not awarded'])
print("The number of unnominated teams:",unnominated_teams)
#Teams that are directly awarded third prize have no qualifications for further second-level award evaluation
first_third=all_group-double_review-unnominated_teams
##############Build the model################
class MyModel:
    def __init__(self, data):
        self.data = data
    def first_calculate_rank(self):
    # Calculate the average score of each team, you can design the calculation formula as needed
        self.data[' Mean value of the first review standard scores '] = self.data[['std-score1', 'std-score2', 'std-score3', 'std-score4', 'std-score5']].mean(axis=1)
        # Sort the teams in descending order based on the comprehensive scores
        self.data = self.data.sort_values(by=' Mean value of the first review standard scores ', ascending=False)
    def select_top_n_teams(self, n):
      # Select the first n groups of teams for secondary evaluation, top_n_teams is the data table for secondary evaluation
    self.data['award'] = 0  # Initialize the award column, all teams are initially 0
        self.data[' final score'] =0
```

```
        top_n_teams = self.data.head(n)
        for index, row in top_n_teams.iterrows():
            if row[' second review standard score range '] < 20:
                top_n_teams.at[index, ' final score'] = row["Mean value of the first review standard scores "] + row["std-score 6"] + row["std-score 7"] +row["std-score 8"]
            else:
                top_n_teams.at[index, 'final score'] =row["Mean value of the first review standard scores "] + row["std-score6"] +row["std-score7"] +row["std-score8"]
        one_half=top_n_teams.sort_values(by=" final score ", ascending=False)
        # Fill the first 28 values of the "award" column with 1, and fill the 267 cells from the 29th to the 28th with 2, and fill the last 57 cells with 3
        one_half['award'].iloc[:28] = 1
        one_half['award'].iloc[28:28+267] = 2
        one_half['award'].replace(0, 3, inplace=True)
        return one_half
    def assign_awards(self, m):
    # Assign awards and evaluation levels
        self.data['award'] = 0
        self.data['final score'] =0
    # Assign awards to the first n groups of teams
        top_n_teams = self.select_top_n_teams(n)
        # Assign third prize to the (n+1) to m groups of teams
        self.data.loc[top_n_teams.index[-1]+1:top_n_teams.index[-1]+m, 'award'] =3
    # Assign "not awarded" to the (m+1) to the last group of teams
        self.data.loc[top_n_teams.index[-1]+m+1:, 'award'] =0
    # Assign final scores directly to third prize and not awarded
        self.data.loc[top_n_teams.index[-1]+1:, ' final score'] =self.data["std-score 1"]+self.data["std-score 2"]+self.data["std-score 3"]+self.data["std-score 4"]+self.data["std-score 5"]
        Second_half=self.data[n:]
        return Second_half
############Create evaluation model object############
model = MyModel(df)
one_half= model.select_top_n_teams(double_review)
print(len(one_half))
one_half.to_excel("1_half.xlsx")
last_half=model.assign_awards(first_third)
```

```
last_half.to_excel("2_half.xlsx")
Result = pd.concat([one_half, last_half])
Result['rank1'] = range(1, len(Result) + 1)
Result.to_excel('combined_Result.xlsx', index=False)
sort_data = pd.read_excel('data_sort.xlsx')
merged_data = pd.merge(Result, sort_data, on='work ID')
# Calculate correlation coefficient
correlation_coefficient = merged_data['rank'].corr(merged_data['rank 1'])
# Subsequent heatmap visualization
#################Model Validation#################
#First, obtain the data, this part is omitted
#Model Validation
n_experiments = 100
results = []
for _ in range(n_experiments):
# Introduce random errors or change parameters on simulated data
   noisy_data = data.copy()
noisy_data['First Review Standard Score'] += np.random.normal(0, 2, n_samples)  # Example: Introduce random errors
noisy_data['Final Score'] = noisy_data['First Review Standard Score'] * 0.6 + noisy_data['Second Review Standard Score'] * 0.4  # Example, adjust according to the actual model
   noisy_data['Rank'] = noisy_data['Final Score'].rank(ascending=False)
   results.append(noisy_data['Rank'].values)
# Convert the results into a DataFrame
results_df = pd.DataFrame(results)
#Visualization display (box plot)
```

ISSN: 2574-1241

DOI: 10.26717/BJSTR.2023.53.008448

Bin Zhao. Biomed J Sci & Tech Res

Submission Link: https://biomedres.us/submit-manuscript.php