AMRITA SCHOOL OF ENGINEERING

19CSE304 - Foundations of Data Science

CASE STUDY REVIEW TOPIC:

Problem Recommendation

STUDENT DETAILS:

S.NO	NAME	ROLL NO
1	Guhan M	CB.EN.U4CSE19425
2	M Sri Hari	CB.EN.U4CSE19435
3	Praveen Kumar R	CB.EN.U4CSE19451
4	S Pranav Adith	CB.EN.U4CSE19458

Overview

The use of coding platforms to support students acquiring programming skills is common nowadays because this type of software contains a large collection of programming exercises to be solved by students.

A common problem that students face when using coding platforms is information overload, as choosing the right problem to solve can be quite frustrating due to the large number of problems offered. Hence, the aim of this paper is to support students with the information overload problem by using a collaborative filtering recommendation approach that filters out programming problems suitable for students' programming skills.

It uses an enriched user-problem matrix that implies a better student role representation, facilitating the computation of closer neighborhoods and hence a more accurate recommendation.

A case study is carried out on a coding platform real dataset showing that the proposal outperforms other previous approaches.

Goals

In this challenge we are required to build a model to predict the number of attempts taken by participants for a successful submission to online programming challenges. Data of programmers and questions they solved previously were given along with the time they took to solve the questions.

About dataset

In order to analyzing users we have used

Attributes details:

- `user_id`
- `problem_id`
- `attempts_range`
- `level_type  points`
- `tags`
- `submission_count`
- `problem_solved`
- `contribution`
- `country`
- `follower_count`
- `last_online_time_seconds`
- `max_rating`
- `rating`
- `rank`
- `Registration_time_seconds`

	user_id	problem_id	attempts_range	level_type	points	tags	submission_count	problem_solved
0	user_232	prob_6507	1	B	1000.0	strings	53	47
1	user_232	prob_5071	4	A	500.0	implementation	53	47
2	user_232	prob_703	2	A	500.0	brute force,implementation	53	47
3	user_232	prob_3935	1	C	1000.0	greedy,sortings	53	47
4	user_232	prob_164	2	A	500.0	brute force,constructive algorithms,math	53	47

contribution	country	follower_count	last_online_time_seconds	max_rating	rating	rank	registration_time_seconds
0	Bangladesh	1	1503633778	307.913	206.709	beginner	1432110935
0	Bangladesh	1	1503633778	307.913	206.709	beginner	1432110935
0	Bangladesh	1	1503633778	307.913	206.709	beginner	1432110935
0	Bangladesh	1	1503633778	307.913	206.709	beginner	1432110935
0	Bangladesh	1	1503633778	307.913	206.709	beginner	1432110935

Data Preprocessing

The first step in the process of analyzing air quality is to pre-process the dataset obtained. It is a way of converting this raw data into a much-desired form so that useful information can be derived from it, which is fed into the training model. Our dataset has few null values and outliers which must be handled using necessary methods.

```
df.columns
```

```
Index(['user_id', 'problem_id', 'attempts_range', 'level_type', 'points',
       'tags', 'submission_count', 'problem_solved', 'contribution', 'country',
       'follower_count', 'last_online_time_seconds', 'max_rating', 'rating',
       'rank', 'registration_time_seconds'],
      dtype='object')
```

```
df.shape
```

```
(155295, 16)
```

CHECKING FOR NAN VALUES

```
df.isnull().sum()
```

```
user_id                          0
problem_id                       0
attempts_range                   0
level_type                     620
points                       29075
tags                         15427
submission_count                 0
problem_solved                   0
contribution                     0
country                      37853
follower_count                   0
last_online_time_seconds         0
max_rating                       0
rating                           0
rank                             0
registration_time_seconds        0
dtype: int64
```

```
df['country'].mode()[0], df['level_type'].mode()[0], df['tags'].mode()[0], df['points'].mode()[0]
```

```
('India', 'A', 'implementation', 500.0)
```

```python
df['country'].fillna('India', inplace=True)
```

```python
df['level_type'].fillna('A', inplace=True)
```

```python
df['points'].fillna(500.0, inplace=True)
```

```python
df['tags'].fillna(df['tags'].mode()[0], inplace=True)
```

HERE NAN VALUES WERE REMOVED

```python
df.isnull().sum()
```

```
user_id                     0
problem_id                  0
attempts_range              0
level_type                  0
points                      0
tags                        0
submission_count            0
problem_solved              0
contribution                0
country                     0
follower_count              0
last_online_time_seconds    0
max_rating                  0
rating                      0
rank                        0
registration_time_seconds   0
dtype: int64
```

Outlier Detection

Outliers increase the variability in your data, which decreases statistical power

Using IQR method

```
[ ] q1 = np.percentile(df['points'], 25)
    q1
```

```
500.0
```

```
[ ] q3 = np.percentile(df['points'], 75)
    q3
```

```
1000.0
```

```
[ ] iqr = q3 - q1
    iqr
```

```
500.0
```

```
[ ] cut_off = iqr*1.5
    lower, upper = q1 - cut_off, q3 + cut_off
    print(lower)
    print(upper)
```

```
-250.0
1750.0
```

```
[ ] outliers = [x for x in df['points'] if (x<lower) or (x>upper)]
    len(outliers)
```

```
13124
```

Using 3 standard deviation method

```
[ ] mean = df['points'].mean()
    mean
```

```
900.9499787499839
```

```
[ ] std = df['points'].std()
    std
```

```
547.1329251780865
```

```
[ ] cut_off = std*3
    lower, upper = mean - cut_off, mean + cut_off
    print(lower)
    print(upper)
```

```
-740.4487967842755
2542.348754284243
```

```
[ ] outliers = [x for x in df['points'] if (x<lower) or (x>upper)]
    len(outliers)
```

```
1045
```

```
[ ] df_no_outlier = df.loc[(df['points'] > lower) & (df['points'] < upper)]
    df_no_outlier.shape
```

```
(154249, 16)
```

```
[ ] df = df_no_outlier.copy()
    df.shape
```

```
(154249, 16)
```

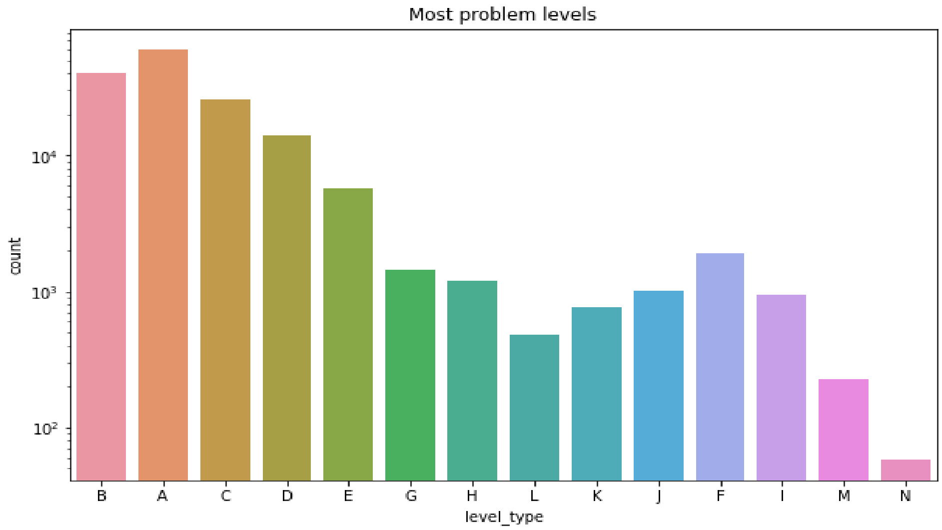
Visualization

For easy understanding we use visualization for graphical representation

logarithmic graph to check the count of level type problems

```
plt.figure(figsize=(10, 6))
sns.countplot(df['level_type'])
plt.yscale("log")
plt.title('Most problem levels')
```

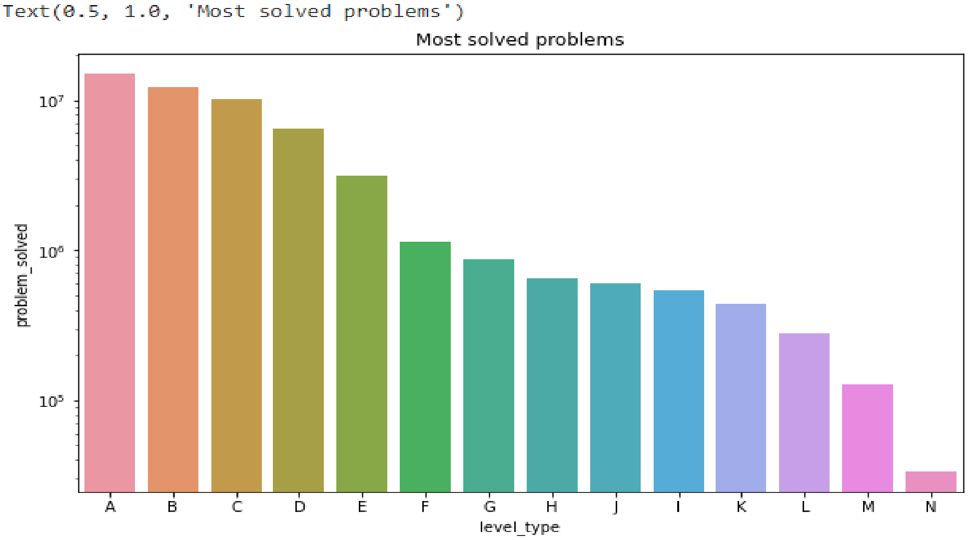
```
Text(0.5, 1.0, 'Most problem levels')
```



logarithmic graph to check the most solved problem vs the level type problems

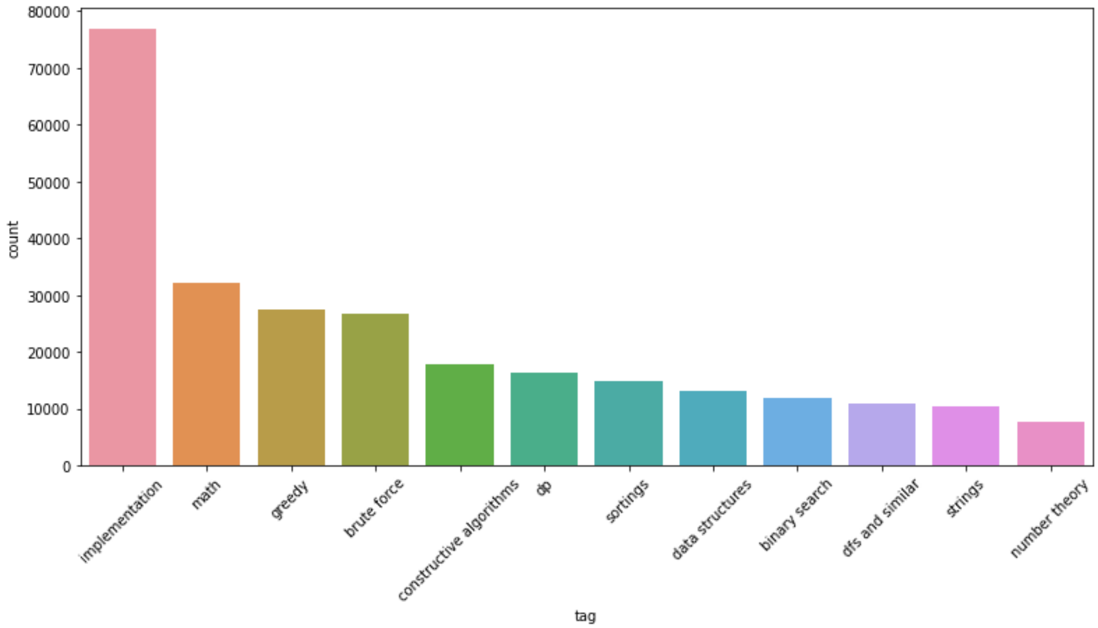
```
plt.figure(figsize=(10, 6))
sns.barplot(x=level_type_df['level_type'], y=level_type_df['problem_solved'])
plt.yscale("log")
plt.title('Most solved problems')
```

```
Text(0.5, 1.0, 'Most solved problems')
```



This graph to check the count of tag

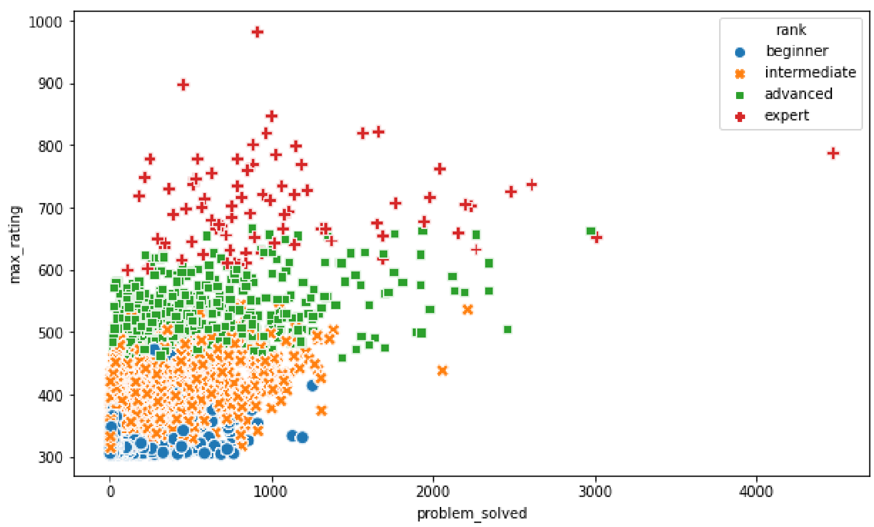
```
plt.figure(figsize=(13, 6))
sns.barplot(x=tag_df['tag'].head(12), y=tag_df['count'].head(12))
plt.xticks(rotation=45)
plt.show()
```



inference: implementation has higher counts than any other tags

This scatterplot shows us solved problems vs maximum rating with hue: rank

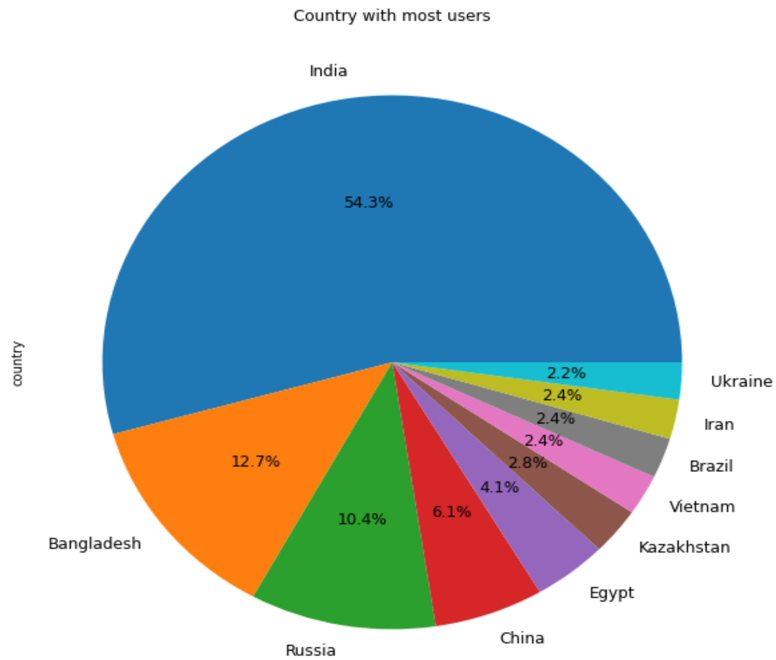
```
plt.figure(figsize=(10, 6))
sns.scatterplot(x=df['problem_solved'], y=df['max_rating'], hue=df['rank'], style=df['rank'], s=80)
plt.show()
```



inference: experts data are spread unevenly and their max ratings are higher

Here we can see the users from different countries

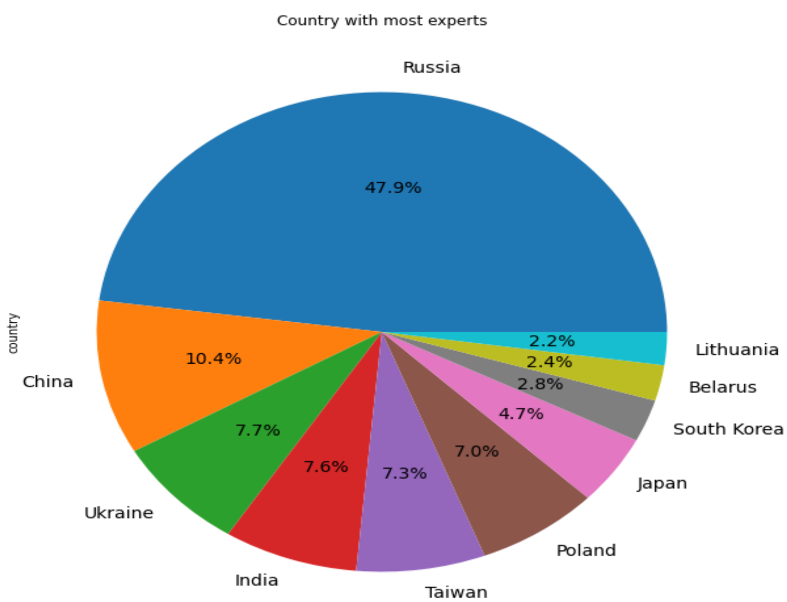
```
plt.figure(figsize=(10, 10))
plt.title('Country with most users')
users_df.plot.pie(y=df.values, autopct='%1.1f%%', textprops={'fontsize': 12})
plt.show()
```



inference: India has the most no. of users

Here we can see the experts users from different countries

```
plt.figure(figsize=(10,10))
plt.title('Country with most experts')
exp_values.plot.pie(y=exp_values.values, autopct='%1.1f%%', textprops={'fontsize': 14})
plt.show()
#
```



inference: Russia has the most no. of experts

Handling Imbalance Data

Imbalanced datasets mean that the number of observations differs for the classes in a classification dataset. This imbalance can lead to inaccurate and biased results while building the model. The distribution of an imbalanced dataset is characterized by very high differences between the classes involved. To handle this, techniques such as oversampling and under sampling have been used.

CHECKING IMBALANCE OR NOT:

SMOTE[Synthetic Minority Oversampling Technique.]

This is a statistical technique for increasing the number of cases in your dataset in a balanced way. The module works by generating new instances from existing minority cases that you supply as input.

```
#SMOTE
from imblearn.over_sampling import SMOTE
from collections import Counter

smote = SMOTE()

# fit predictor and target variable
x_smote, y_smote = smote.fit_resample(X_train, y_train)

print('Original dataset shape', Counter(y_train))
print('Resample dataset shape', Counter(y_smote))

# print('Original dataset shape', Counter(X_train))
# print('Resample dataset shape', Counter(x_smote))
```

```
Original dataset shape Counter({1: 57634, 2: 32912, 3: 9867, 4: 3785, 6: 2076, 5: 1700})
Resample dataset shape Counter({1: 57634, 2: 57634, 3: 57634, 4: 57634, 5: 57634, 6: 57634})
```

Under and Over sampling technique:

After splitting the data into TRAIN and TEST data we have to apply the technique

Under Sampling

```
[ ] from imblearn.under_sampling import RandomUnderSampler

    rus = RandomUnderSampler(random_state=2, replacement=True)# fit predictor and target variable

    X_rus, y_rus = rus.fit_resample(X_train, y_train)

    print('original dataset shape:', Counter(y_train))
    print('Resample dataset shape', Counter(y_rus))

    original dataset shape: Counter({1: 57634, 2: 32912, 3: 9867, 4: 3785, 6: 2076, 5: 1700})
    Resample dataset shape Counter({1: 1700, 2: 1700, 3: 1700, 4: 1700, 5: 1700, 6: 1700})
```

Over Sampling

```
[ ] from imblearn.over_sampling import RandomOverSampler

    ros = RandomOverSampler(random_state=2)

    X_ros, y_ros = ros.fit_resample(X_train, y_train)

    print('Original dataset shape', Counter(y_train))
    print('Resample dataset shape', Counter(y_ros))

    Original dataset shape Counter({1: 57634, 2: 32912, 3: 9867, 4: 3785, 6: 2076, 5: 1700})
    Resample dataset shape Counter({1: 57634, 2: 57634, 3: 57634, 4: 57634, 5: 57634, 6: 57634})
```

Descriptive Statistics

There are 3 main types of descriptive statistics:

- The distribution concerns the frequency of each value.
- The central tendency concerns the averages of the values.
- The variability or dispersion concerns how spread out the values are

```
df.mean()
```

```
attempts_range                      1.750501
level_type                          1.402550
points                            887.097330
submission_count                  370.607168
problem_solved                    334.494959
contribution                        5.403400
country                            33.275846
follower_count                     59.866119
max_rating                        406.687068
rating                            367.743888
rank                                1.697645
tag_*special                        0.005673
tag_2-sat                           0.002574
tag_binary search                   0.077699
tag_bitmasks                        0.015572
tag_brute force                     0.173836
tag_chinese remainder theorem       0.000421
tag_combinatorics                   0.018282
tag_constructive algorithms         0.116124
tag_data structures                 0.084435
tag_dfs and similar                 0.070464
```

```
df.median()
```

```
attempts_range                      1.000
level_type                          1.000
points                            500.000
submission_count                  236.000
problem_solved                    208.000
contribution                        0.000
country                            31.000
follower_count                     20.000
max_rating                        382.454
rating                            355.218
rank                                1.000
tag_*special                        0.000
tag_2-sat                           0.000
tag_binary search                   0.000
tag_bitmasks                        0.000
tag_brute force                     0.000
tag_chinese remainder theorem       0.000
tag_combinatorics                   0.000
tag constructive algorithms         0.000
```

```
df.var()
```

```
attempts_range                          1.153422
level_type                              3.466920
points                             272636.903907
submission_count                   157441.416451
problem_solved                     141333.249808
contribution                          356.595981
country                               359.734480
follower_count                      60469.410094
max_rating                           9782.603908
rating                              12418.312242
rank                                    1.456672
tag_*special                            0.005641
tag_2-sat                               0.002567
tag_binary search                       0.071662
tag_bitmasks                            0.015330
tag_brute force                         0.143618
tag_chinese remainder theorem           0.000421
tag_combinatorics                       0.017948
tag_constructive algorithms             0.102640
```

```
df.std()
```

```
attempts_range                     1.073975
level_type                         1.861967
points                           522.146439
submission_count                 396.788881
problem_solved                   375.943147
contribution                      18.883749
country                           18.966668
follower_count                   245.905287
max_rating                        98.907047
rating                           111.437481
rank                               1.206927
tag_*special                       0.075103
tag_2-sat                          0.050667
tag_binary search                  0.267698
tag_bitmasks                       0.123814
tag_brute force                    0.378969
tag_chinese remainder theorem      0.020524
tag_combinatorics                  0.133970
```

```
df.kurt()
```

```
attempts_range                      4.004866
level_type                          7.810169
points                              0.730704
submission_count                   13.473733
problem_solved                     15.392129
contribution                       23.802643
country                            -0.112283
follower_count                   1265.690854
max_rating                          2.202784
rating                              0.823675
rank                               -1.621636
tag_*special                      171.295868
tag_2-sat                         383.551576
tag_binary search                   7.954713
tag_bitmasks                       59.234680
tag_brute force                     0.963038
tag_chinese remainder theorem    2368.138762
tag_combinatorics                  49.718500
```

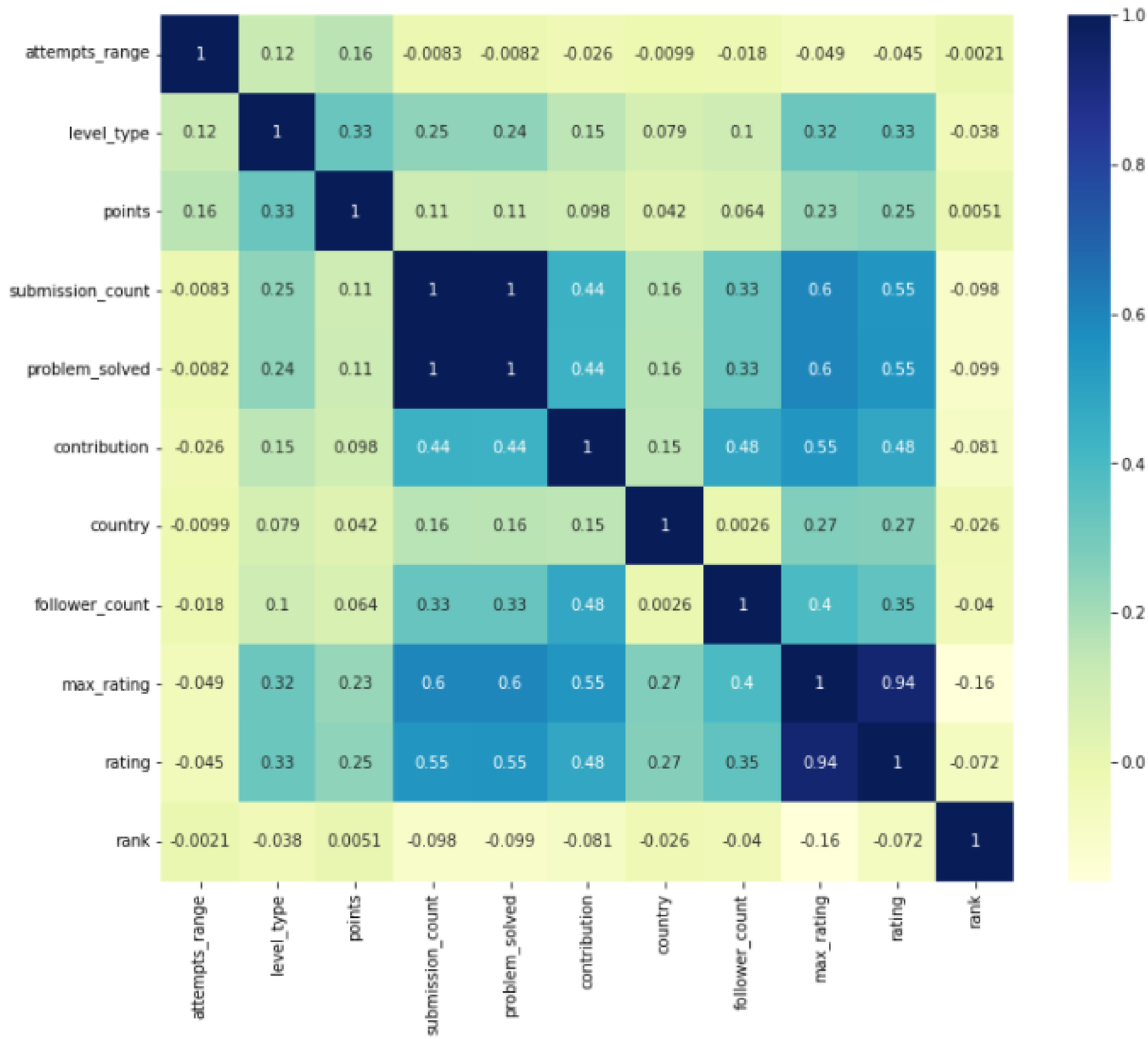
```
df.skew()
```

```
attempts_range                    1.915188
level_type                        2.427905
points                            1.248408
submission_count                  2.837701
problem_solved                    3.017744
contribution                      4.492542
country                           0.551704
follower_count                   30.905768
max_rating                        1.356209
rating                            0.617473
rank                             -0.065370
tag_*special                     13.164104
tag_2-sat                        19.635341
tag_binary search                 3.155093
tag_bitmasks                      7.825210
tag_brute force                   1.721344
tag_chinese remainder theorem    48.683756
tag_combinatorics                 7.191513
```

Removing duplicated and highly correlated columns

```
fig, ax = plt.subplots(figsize=(12,10))
heatmap_df=df.select_dtypes(exclude=['object'])
heatmap_df=heatmap_df.iloc[:,0:11]
# heatmap_df.head()
sns.heatmap(heatmap_df.corr(), annot=True, cmap='YlGnBu', ax=ax)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc7b6dd8c10>
```



Feature Selection

Feature Selection

```
[83] from sklearn.feature_selection import VarianceThreshold

     var_threshold = VarianceThreshold(threshold=0)
     var_threshold.fit(df.select_dtypes(exclude=['object', 'datetime']))
```

```
VarianceThreshold(threshold=0)
```

```
[84] var_threshold.get_support()
```

```
array([ True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True])
```

```
[85] df.select_dtypes(exclude=['object', 'datetime']).columns[var_threshold.get_support()]
```

```
Index(['attempts_range', 'level_type', 'points', 'submission_count',
       'problem_solved', 'contribution', 'country', 'follower_count',
       'max_rating', 'rating', 'rank', 'tag_*special', 'tag_2-sat',
       'tag_binary search', 'tag_bitmasks', 'tag_brute force',
       'tag_chinese remainder theorem', 'tag_combinatorics',
       'tag_constructive algorithms', 'tag_data structures',
       'tag_dfs and similar', 'tag_divide and conquer', 'tag_dp', 'tag_dsu',
       'tag_expression parsing', 'tag_fft', 'tag_flows', 'tag_games',
       'tag_geometry', 'tag_graph matchings', 'tag_graphs', 'tag_greedy',
       'tag_hashing', 'tag_implementation', 'tag_math', 'tag_matrices',
       'tag_meet-in-the-middle', 'tag_number theory', 'tag_probabilities',
       'tag_schedules', 'tag_shortest paths', 'tag_sortings',
       'tag_string suffix structures', 'tag_strings', 'tag_ternary search',
       'tag_trees', 'tag_two pointers'],
      dtype='object')
```

Model Building

Logistic Regression

It establishes a relationship between dependent variable (Y) and one or more independent variables (X) using a best fit straight line

Logistic Regression

```
[96] from sklearn.linear_model import LogisticRegression
     from sklearn import metrics
     from sklearn.metrics import classification_report
     from sklearn.model_selection import cross_val_score

     LogReg = LogisticRegression(random_state=2)
     LogReg.fit(X_train,y_train)
     predicted_values = LogReg.predict(X_test)
     x = metrics.accuracy_score(y_test, predicted_values)
     model.append('Logistic Regression')
     accuracy.append(x*100)
     print(classification_report(y_test, predicted_values))
     print("Logistic Regression's Accuracy is: ", x*100)
```

```
              precision    recall  f1-score   support

           1       0.54      0.98      0.70     24846
           2       0.35      0.04      0.07     14053
           3       0.00      0.00      0.00      4114
           4       0.00      0.00      0.00      1631
           5       0.00      0.00      0.00       746
           6       0.00      0.00      0.00       885

    accuracy                           0.54     46275
   macro avg       0.15      0.17      0.13     46275
weighted avg       0.40      0.54      0.40     46275

Logistic Regression's Accuracy is:  53.70934629929768
```

The accuracy for X_train and y_train is 53.7%. The precisions for 3, 4, 5, 6 are very low because of imbalanced data

Random Forest Classifier

Random forest classifiers can be used to solve regression or classification problems. The random forest algorithm is made up of a collection of decision trees, and each tree in the ensemble consists of a data sample drawn from a training set with replacement, called the bootstrap sample.

Random Forest

```
[102] from sklearn.ensemble import RandomForestClassifier

      RF = RandomForestClassifier(n_estimators=50, random_state=0)
      RF.fit(X_train,y_train)
      predicted_values = RF.predict(X_test)
      x = metrics.accuracy_score(y_test, predicted_values)
      accuracy.append(x*100)
      model.append('Random Forest')
      print(classification_report(y_test,predicted_values))
      print("RF's Accuracy is: ", x*100)
```

```
              precision    recall  f1-score   support

           1       0.59      0.76      0.66     24846
           2       0.35      0.29      0.32     14053
           3       0.15      0.05      0.07      4114
           4       0.07      0.02      0.03      1631
           5       0.04      0.01      0.02       746
           6       0.14      0.04      0.06       885

    accuracy                           0.50     46275
   macro avg       0.22      0.19      0.19     46275
weighted avg       0.44      0.50      0.46     46275

RF's Accuracy is:  50.37277147487844
```

K-Nearest Neighbors

K-nearest neighbors (KNN) is a type of supervised learning algorithm used for both regression and classification. KNN tries to predict the correct class for the test data by calculating the distance between the test data and all the training points. Then select the K number of points which is closest to the test data. The KNN algorithm calculates the probability of the test data belonging to the classes of 'K' training data and which class holds the highest probability will be selected. In the case of regression, the value is the mean of the 'K' selected training points.

K- Nearest Neighbors

[106]
```
from sklearn.neighbors import KNeighborsClassifier

KNN = KNeighborsClassifier(n_neighbors=7)
KNN.fit(X_train, y_train)
predicted_values = KNN.predict(X_test)
x = metrics.accuracy_score(y_test, predicted_values)
accuracy.append(x*100)
model.append('K-Nearest Neighbors')
print(classification_report(y_test,predicted_values))
print("KNN's Accuracy is: ", x*100)
```

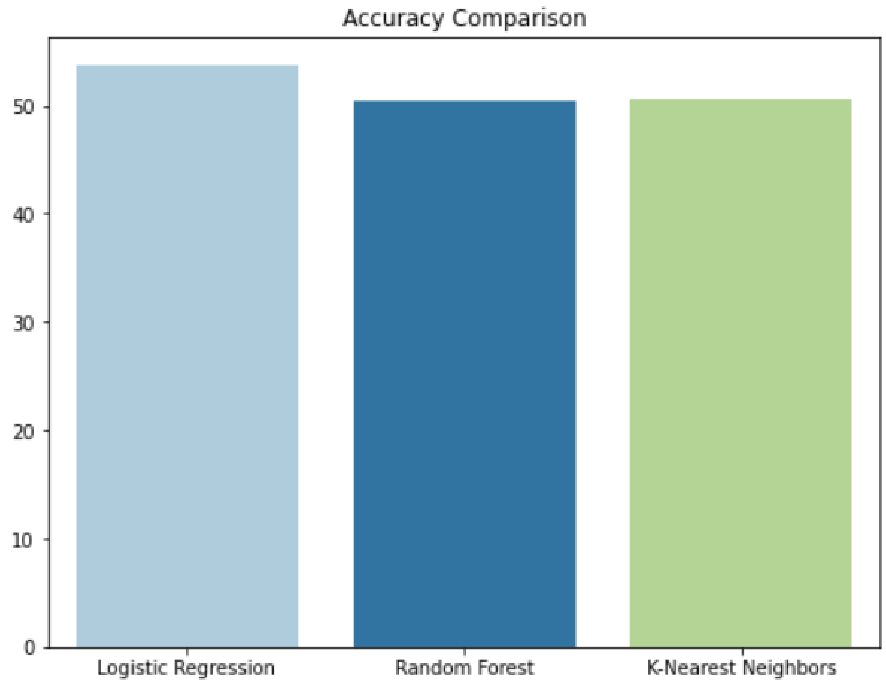
```
              precision    recall  f1-score   support

           1       0.56      0.81      0.66     24846
           2       0.33      0.22      0.27     14053
           3       0.13      0.02      0.04      4114
           4       0.08      0.00      0.01      1631
           5       0.00      0.00      0.00       746
           6       0.13      0.00      0.01       885

    accuracy                           0.51     46275
   macro avg       0.21      0.18      0.16     46275
weighted avg       0.42      0.51      0.44     46275

KNN's Accuracy is:  50.61480280929227
```

Model Comparison

Model Comparision

```
[108] plt.figure(figsize=(8,6))
      plt.title('Accuracy Comparison')
      sns.barplot(x=model, y=accuracy, palette='Paired')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa6904f1210>
```



Hypothesis Testing

Z-test

A coding expert believes that the mean of the problems solved is greater than the average problems solved, which is 334.4. Assume the population standard deviation is 375.9. A random sample of 500 problems is selected, and the mean of the sample is 336.7. At α = 0.05, is there enough evidence to reject the claim?

```
[ ] #H0 : μ = 334.4, Ha :  μ > 334.4
    n = 500
    xbar = 336.7
    mu = 334.4
    sigma = 375.9
    alpha = 0.05
```

```
[ ] Z_critical = abs(st.norm.ppf(alpha))
    Z_critical
```

```
1.6448536269514729
```

```
[ ] z=(xbar-mu)/(sigma/np.sqrt(n))
    z
```

```
0.13681714148043475
```

```
[ ] if (z < Z_critical): #Right-tailed test
        print("Accept Null hypothesis")
    else:
        print("Reject Null hypothesis")
```

```
Accept Null hypothesis
```

Z-Test using P-value

A coding expert believes that the mean of the problems solved is greater than the average problems solved, which is 334.4. Assume the population standard deviation is 375.9. A random sample of 500 problems is selected, and the mean of the sample is 336.7. At α = 0.05, is there enough evidence to reject the claim? Use P-value method

```
#H0 : μ = 334.4, Ha :  μ > 334.4
n = 500
xbar = 336.7
mu = 334.4
sigma = 375.9
alpha = 0.05
```

```
z=(xbar-mu)/(sigma/np.sqrt(n))
z
```

```
0.13681714148043475
```

```
p_val=(1-st.norm.cdf(z))*2
p_val
```

```
0.891175334116848
```

```
if (p_val>alpha):
    print("Accept Null hypothesis")
else:
    print("Reject null hypothesis")
```

```
Accept Null hypothesis
```

t-Test

```
population_mean = df1['submission_count'].mean()
print("Population Mean :", population_mean)
population_std = df1['submission_count'].std()
print("Population Standard Deviation :", population_std)
population_var = df1['submission_count'].var()
print("Population Variance :", population_var)
```

```
Population Mean : 370.60716763155676
Population Standard Deviation : 396.7888814607207
Population Variance : 157441.41645084985
```

```
sample_mean = sample_df['submission_count'].mean()
print("Sample Mean :", sample_mean)
sample_std = sample_df['submission_count'].std()
print("Sample Standard Deviation :", sample_std)
sample_var = sample_df['submission_count'].var()
print("Sample Variance :", sample_var)
```

```
Sample Mean : 377.876
Sample Standard Deviation : 429.0192392414105
Sample Variance : 184057.50763927863
```

An online coding platform claims that the average submissions is 370.6. A random sample of 500 problems had a mean submission of 373.1. The sample standard deviation is 384.0. Is there enough evidence to reject the coding platform's claim at α = 0.05? Assume the variable is normally distributed.

```
[ ] #H0 : μ = 370.6 Ha : μ != 370.6
    n = 500
    degrees_of_freedom = n-1
    xbar = 373.1
    mu = 370.6
    s = 384
    alpha = 0.05
```

```
[ ] t = (xbar-mu)/(s/np.sqrt(n))
    t

    0.14557734228514257
```

```
[ ] t_critical = abs(st.t.ppf(alpha/2, degrees_of_freedom))
    t_critical

    1.9647293909876653
```

```
[ ] if (t>t_critical):
        print("Accept Null hypothesis")
    else:
        print("Reject null hypothesis")

    Reject null hypothesis
```

t-Test using P-value

An online coding platform claims that average submissions is greater than the average of submissions for all problems. A random sample of 500 problems has a mean of 373.1 and a standard deviation of 384. If the average submissions of all problems is 370.6, is there enough evidence to support the coding platform's claim at α = 0.05? Use P-value method

```
[ ] #H0 : μ = 370.6 Ha : mu > 370.6
    n = 500
    degrees_of_freedom = n-1
    xbar = 373.1
    mu = 370.6
    s = 384
    alpha = 0.05
```

```
[ ] t = (xbar-mu)/(s/np.sqrt(n))
    t
```

```
0.14557734228514257
```

```
[ ] p_val = (1-st.t.cdf(abs(t), degrees_of_freedom))
    p_val
```

```
0.44215692044895305
```

```
[ ] if (p_val>alpha):
        print("Accept Null hypothesis")
    else:
        print("Reject null hypothesis")
```

```
Accept Null hypothesis
```

Chi-Square Test

```
population_mean = df1['attempts_range'].mean()
print("Population Mean :", population_mean)
population_std = df1['attempts_range'].std()
print("Population Standard Deviation :", population_std)
population_var = df1['attempts_range'].var()
print("Population Variance :", population_var)
```

```
Population Mean : 1.7505008136195372
Population Standard Deviation : 1.0739748212746787
Population Variance : 1.1534219167319781
```

```
[ ] sample_mean = sample_df['attempts_range'].mean()
    print("Sample Mean :", sample_mean)
    sample_std = sample_df['attempts_range'].std()
    print("Sample Standard Deviation :", sample_std)
    sample_var = sample_df['attempts_range'].var()
    print("Sample Variance :", sample_var)
```

```
Sample Mean : 1.646
Sample Standard Deviation : 0.991286688399073
Sample Variance : 0.9826492985972008
```

Competitive Programmer wishes to see if the variance of attempts taken of 500 problems is less than the variance of the population, which is 1.15. The variance of the attempts taken of 500 problems was 1.2. Test the claim at α = 0.05

```
[ ] #H0 : σ2 =1.15 Ha : sigma_sq < 1.15
    n = 500
    degrees_of_freedom = n-1
    s_square = 1.2
    sigma_square = 1.15
    alpha = 0.05
```

```
[ ] chi_square = ((n-1)*s_square)/sigma_square
    chi_square

    520.695652173913
```

```
[ ] chi_square_critical = st.chi2.ppf(alpha, degrees_of_freedom)
    chi_square_critical

    448.19882158627
```

```
[ ] if (chi_square > chi_square_critical):
        print("Accept Null hypothesis")
    else:
        print("Reject null hypothesis")

    Accept Null hypothesis
```

Chi-Square Test using P-Value

A Competitive Programmer knows that the standard deviation of the attempts taken to solve a problem is 1.07. A random sample of 500 problems is selected and the standard deviation 1.09. At α = 0.05, can it be concluded that the standard deviation has changed? Use the P-value method.

```
#H0 : σ = 1.07  Ha : sigma != 1.07
n = 500
degrees_of_freedom = n-1
s = 1.05
sigma = 1.07
alpha = 0.05
```

```
chi_square = ((n-1)*(s**2))/sigma**2
chi_square
```

```
480.5201327626867
```

```
p_val = st.chi2.cdf(chi_square, degrees_of_freedom)*2
p_val
```

```
0.5678773134529609
```

```
if (p_val>alpha):
    print("Accept Null hypothesis")
else:
    print("Reject null hypothesis")
```

```
Accept Null hypothesis
```

Conclusion

The selected dataset has been preprocessed by removing irrelevant columns, replacement of null values with empty spaces and reformation of strings without special/Numeric characters. Appropriate columns have been selected and visualized for getting insightful inference regarding the dataset using seaborn and matplotlib packages. The main objective of building a better model to analyze the recommended problems for the competitive coding program....!!!


Thank You