In [1]:
```
import pandas as pd
from sklearn import preprocessing
import sklearn.model_selection as ms
from sklearn import linear_model
import sklearn.metrics as sklm
import numpy as np
import numpy.random as nr
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as ss
import math
import sweetviz as sv
from tabulate import tabulate

%matplotlib inline
```

In [ ]:
```
```

In [2]:
```
pd.set_option('display.max_rows', None)
```

Loading Data

In [3]:
```
# df = pd.read_csv('train_data/train_task_1_2.csv', usecols = [0,1,2,3,4,5], dtype={'Q
df = pd.read_csv('train_data/train_task_1_2.csv') #Reading the train datasets
```

In [4]:
```
df.info() #Checking the info
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15867850 entries, 0 to 15867849
Data columns (total 6 columns):
 #   Column         Dtype
---  ------         -----
 0   QuestionId     int64
 1   UserId         int64
 2   AnswerId       int64
 3   IsCorrect      int64
 4   CorrectAnswer  int64
 5   AnswerValue    int64
dtypes: int64(6)
memory usage: 726.4 MB
```

In [5]:
```
df.head()
```

Out[5]:

	QuestionId	UserId	AnswerId	IsCorrect	CorrectAnswer	AnswerValue
0	16997	65967	12453206	0	4	2
1	16531	62121	15686710	1	1	1
2	15911	50013	13598796	0	3	1
3	1701	104909	10511925	0	4	3
4	22896	21748	941747	0	1	4

REMOVE DUPLICATES IF PRESENT

NB: We cannot remove duplicate in this type of dataset because students are subjected to answers different questions, also same questions can be answered by different students. Removing the duplicates value might make the model bias

In [4]:
```
# df.drop_duplicates(subset='UserId', keep = 'first', inplace = True)
```

In [5]:
```
# df.loc[df.duplicated(), :]
```

In [6]:
```
df.shape
```

Out[6]:
```
(15867850, 6)
```

In [6]:
```
# df_test_1 = pd.read_csv('test_data/test_public_answers_task_1.csv')
```

In [7]:
```
# df_test_1.head()
```

In [8]:
```
# df_test_2 = pd.read_csv('test_data/test_public_answers_task_2.csv')
```

In [9]:
```
# df_test_2.head()
```

In [4]:
```
df_std = pd.read_csv('metadata/student_metadata_task_1_2.csv') #Reading the student da
#This data wont be enough to get the best model beacuase the features are not really e
```

In [5]:
```
df_std.head()
```

Out[5]:

	UserId	Gender	DateOfBirth	PremiumPupil
0	58022	2	2006-08-01 00:00:00.000	1.0
1	104674	0	NaN	NaN
2	32020	2	2008-11-01 00:00:00.000	1.0
3	81780	2	2004-07-01 00:00:00.000	NaN
4	99967	0	NaN	NaN

In [6]:
```
df_std.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 118971 entries, 0 to 118970
Data columns (total 4 columns):
 #   Column        Non-Null Count   Dtype  
---  ------        --------------   -----  
 0   UserId        118971 non-null  int64  
 1   Gender        118971 non-null  int64  
 2   DateOfBirth   80581 non-null   object 
 3   PremiumPupil  34386 non-null   float64
dtypes: float64(1), int64(2), object(1)
memory usage: 3.6+ MB
```

In [12]:
```
df_std.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 118971 entries, 0 to 118970
Data columns (total 4 columns):
 #   Column        Non-Null Count   Dtype
---  ------        --------------   -----
 0   UserId        118971 non-null  int64
 1   Gender        118971 non-null  int64
 2   DateOfBirth   80581 non-null   object
 3   PremiumPupil  34386 non-null   float64
dtypes: float64(1), int64(2), object(1)
memory usage: 3.6+ MB
```

In [5]:
```
# The train datasets and students data sets are related in terms of UserId, therefore
df3= pd.merge(df, df_std, how ='left', on='UserId')
```

In [8]:
```
df3.head()
```

Out[8]:

	QuestionId	UserId	AnswerId	IsCorrect	CorrectAnswer	AnswerValue	Gender	DateOfBirth	Prer
0	16997	65967	12453206	0	4	2	1	2002-11-01 00:00:00.000	
1	16531	62121	15686710	1	1	1	1	2007-11-01 00:00:00.000	
2	15911	50013	13598796	0	3	1	1	2004-09-01 00:00:00.000	
3	1701	104909	10511925	0	4	3	1	2006-01-01 00:00:00.000	
4	22896	21748	941747	0	1	4	1	NaN	

In [9]:
```
df3.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 15867850 entries, 0 to 15867849
Data columns (total 9 columns):
 #   Column         Dtype
---  ------         -----
 0   QuestionId     int64
 1   UserId         int64
 2   AnswerId       int64
 3   IsCorrect      int64
 4   CorrectAnswer  int64
 5   AnswerValue    int64
 6   Gender         int64
 7   DateOfBirth    object
 8   PremiumPupil   float64
dtypes: float64(1), int64(7), object(1)
memory usage: 1.2+ GB
```

In [6]:
```
df_ans = pd.read_csv('metadata/answer_metadata_task_1_2.csv') # reading the answer dat
```

In [11]:
```
df_ans.head()
```

Out[11]:

	AnswerId	DateAnswered	Confidence	GroupId	QuizId	SchemeOfWorkId
0	11808339.0	2020-03-17 07:55:00.000	NaN	4186	14854	NaN
1	98649.0	2018-12-18 14:23:00.000	NaN	9427	16895	28237.0
2	259238.0	2019-02-21 12:41:00.000	NaN	7651	1127	8386.0
3	17027648.0	2020-03-01 18:13:00.000	NaN	8719	5126	40960.0
4	6579101.0	2019-03-22 14:57:00.000	NaN	9020	13445	NaN

In [7]:
```
df4= pd.merge(df3, df_ans, how ='left', on='AnswerId') # There is a relationship betwe
```

In [14]:
```
df4.head()
```

Out[14]:

	QuestionId	UserId	AnswerId	IsCorrect	CorrectAnswer	AnswerValue	Gender	DateOfBirth	Prer
0	16997	65967	12453206	0	4	2	1	2002-11-01 00:00:00.000	
1	16531	62121	15686710	1	1	1	1	2007-11-01 00:00:00.000	
2	15911	50013	13598796	0	3	1	1	2004-09-01 00:00:00.000	
3	1701	104909	10511925	0	4	3	1	2006-01-01 00:00:00.000	
4	22896	21748	941747	0	1	4	1	NaN	

In [8]:
```
df4.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 15867850 entries, 0 to 15867849
Data columns (total 14 columns):
 #   Column          Dtype  
---  ------          -----  
 0   QuestionId      int64  
 1   UserId          int64  
 2   AnswerId        int64  
 3   IsCorrect       int64  
 4   CorrectAnswer   int64  
 5   AnswerValue     int64  
 6   Gender          int64  
 7   DateOfBirth     object 
 8   PremiumPupil    float64
 9   DateAnswered    object 
 10  Confidence      float64
 11  GroupId         int64  
 12  QuizId          int64  
 13  SchemeOfWorkId  float64
dtypes: float64(3), int64(9), object(2)
memory usage: 1.8+ GB
```

In [18]:
```
df_std_ques = pd.read_csv('metadata/question_metadata_task_1_2.csv') #Just checking ot
```

In [12]:
```
df_std_ques.head()
```

Out[12]:

	QuestionId	SubjectId
0	13090	[3, 32, 71, 77, 141, 185, 186, 214]
1	1855	[3, 71, 75, 86, 178]
2	10423	[3, 32, 38, 239]
3	2290	[3, 32, 33, 144]
4	12785	[3, 32, 33, 144]

In [13]:
```
df_sub = pd.read_csv('metadata/subject_metadata.csv')df_std_ques = pd.read_csv('metada
```

In [14]:
```
df_sub.head()
```

Out[14]:

	SubjectId	Name	ParentId	Level
0	3	Maths	NaN	0
1	32	Number	3.0	1
2	33	BIDMAS	144.0	3
3	34	Upper and Lower Bounds	141.0	3
4	35	Calculator Use	32.0	2

In [12]:
```
df4.dtypes
```

Out[12]:
```
QuestionId          int64
UserId              int64
AnswerId            int64
IsCorrect           int64
CorrectAnswer       int64
AnswerValue         int64
Gender              int64
DateOfBirth        object
PremiumPupil      float64
DateAnswered       object
Confidence        float64
GroupId             int64
QuizId              int64
SchemeOfWorkId    float64
dtype: object
```

In [9]:
```
df4.shape
```

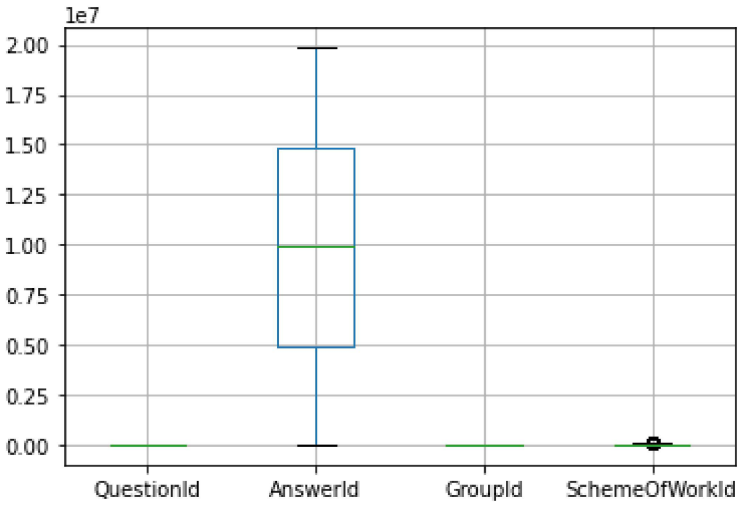
Out[9]:
```
(15867850, 14)
```

DETECTING OUTLIERS in the numerical columns

In [8]:
```
#Spliting the columns into Numerical columns and categorical columns so as to be able
numeric_col= ['QuestionId','AnswerId','GroupId','SchemeOfWorkId']
categorical_col = ['IsCorrect','CorrectAnswer','AnswerValue','Gender']
```

In [9]:
```
df4.boxplot(numeric_col)
```

Out[9]:
```
<AxesSubplot:>
```



SchemeOFWorkId has Outliers which lie at the lower bound.

In [10]:
```
#Treating the outliers on SChemeOfWorkId
for x in ['SchemeOfWorkId']:
    q75,q25 = np.percentile(df4.loc[:,x],[75,25])
    intr_qr = q75-q25

    max = q75+(1.5*intr_qr)
    min = q25-(1.5*intr_qr)

    df4.loc[df4[x] < min,x] = np.nan
    df4.loc[df4[x] > max,x] = np.nan
```

Thus, i have used numpy.percentile() method to calculate the values of Q1 and Q3. Further, i have replaced the outliers with numpy.nan as the NULL values.

Having replaced the outliers with nan, let us now check the sum of null values or missing values using the below code:

In [11]:
```
df4.isnull().sum()
```

Out[11]:
```
QuestionId              0
UserId                  0
AnswerId                0
IsCorrect               0
CorrectAnswer           0
AnswerValue             0
Gender                  0
DateOfBirth       5091938
PremiumPupil      9904268
DateAnswered            0
Confidence       14361798
GroupId                 0
QuizId                  0
SchemeOfWorkId    8085318
dtype: int64
```

Now, we can use any of the below techniques to treat the NULL values:

by either imputing the missing values with Mean, median etc. or Drop the null values (if the proportion is comparatively less) Here, i later drop the null values using pandas.dataframe.dropna() function later in this notebook. The reason is cos if i drop all na value now without dropping the Confidence column that contains about 99% of nan values then we will be left with just 1% datas to work with which doesnt make any sense. So we need to dropConfidence column first before dropping all rows with nan values

In [8]:
```
# df4 = df4.dropna(axis = 0)
# df4= df4.drop('SchemeOfWorkId', axis = 0, inplace =True)
```

In [ ]:
```
# df4.isnull().sum()
```

In [13]:
```
df4.shape
```

Out[13]:
```
(538835, 14)
```

In [ ]:
```

```

In [15]:
```
print(df4.columns.tolist()) #tRying to visualize  the columns of the total merged data
```

```
['QuestionId', 'UserId', 'AnswerId', 'IsCorrect', 'CorrectAnswer', 'AnswerValue', 'Ge
nder', 'DateOfBirth', 'PremiumPupil', 'DateAnswered', 'Confidence', 'GroupId', 'QuizI
d', 'SchemeOfWorkId']
```

In [17]:
```
print(df4['QuizId'].unique())
```

```
[15391  4681 17226 ...  7108  2018  3545]
```

In [ ]:
```
# print(df4['QuizId'].value_counts())
```

USING SWEETVIZ FOR VISUALIZATION BEFORE DATA PREPROCESSING

In [8]:
```
import sweetviz as sv
```

In [ ]:
```

```

In [9]:
```
df4_report = sv.analyze([df4, 'Train'], target_feat='IsCorrect') #FOR Iscorrect Visual
```

```
                                             |          | [  0%]   00:00 -> (? left)
```

In [10]:
```
df4_report.show_html('Report_IsCorrect.html')
```

```
Report Report.html was generated! NOTEBOOK/COLAB USERS: the web browser MAY not pop u
p, regardless, the report IS saved in your notebook/colab files.
```

FOR ANSWERVALUE

In [22]:
```
feat_config = sv.FeatureConfig(skip = 'UserId',force_num=['AnswerValue']) #FOR ANSWERV
```

In [23]:
```
df4_report_2 = sv.analyze([df4, 'Train'], None, feat_config)
```

```
                                                |          | [  0%]   00:00 -> (? left)
```

In [24]:
```
df4_report_2.show_html('Report_AnswerValue.html')
```

```
Report Report_AnswerValue.html was generated! NOTEBOOK/COLAB USERS: the web browser M
AY not pop up, regardless, the report IS saved in your notebook/colab files.
```

In [18]:
```
print(df4['QuizId'].count()) #Checking for the total number of QuizId
```

```
15867850
```

In [32]:
```
def quality_report(df):
    """
    Description: Displays quality of data in terms of missing values,
    unique numbers, datatypes etc.

    Arguments: Dataframe
    """
    dtypes = df.dtypes
    nuniq = df.T.apply(lambda x: x.nunique(), axis=1)
    total = df.isnull().sum().sort_values(ascending = False)
    percent = (df.isnull().sum()/df.isnull().count()*100).sort_values(ascending = Fals
    quality_df  = pd.concat([total, percent, nuniq, dtypes], axis=1, keys=['Total NaN'
    display(quality_df)
```

In [33]:
```
quality_report(df4) #Noe we can see why we cant just drop all nan values cos Confidenc
```

	Total NaN	Percent of NaN	Nunique	Dtype
Confidence	14361798	90.508783	5	float64
PremiumPupil	9904268	62.417202	2	float64
SchemeOfWorkId	8085318	50.954086	58	float64
DateOfBirth	5091938	32.089653	758	object
QuestionId	0	0.000000	27613	int64
UserId	0	0.000000	118971	int64
AnswerId	0	0.000000	15867850	int64
IsCorrect	0	0.000000	2	int64
CorrectAnswer	0	0.000000	4	int64
AnswerValue	0	0.000000	4	int64
Gender	0	0.000000	4	int64
DateAnswered	0	0.000000	724041	object
GroupId	0	0.000000	11838	int64
QuizId	0	0.000000	17227	int64

In [8]:
```
df4.describe()
```

Out[8]:

	QuestionId	UserId	AnswerId	IsCorrect	CorrectAnswer	AnswerValue	
count	1.586785e+07	1.586785e+07	1.586785e+07	1.586785e+07	1.586785e+07	1.586785e+07	1.586
mean	1.380277e+04	5.952587e+04	9.916806e+06	6.429503e-01	2.502599e+00	2.488978e+00	1.142
std	7.967465e+03	3.428394e+04	5.725901e+06	4.791297e-01	1.104868e+00	1.094272e+00	7.627
min	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	1.000000e+00	1.000000e+00	0.000
25%	6.890000e+03	2.980900e+04	4.957568e+06	0.000000e+00	2.000000e+00	2.000000e+00	1.000
50%	1.383000e+04	5.958500e+04	9.916784e+06	1.000000e+00	3.000000e+00	2.000000e+00	1.000
75%	2.069400e+04	8.912500e+04	1.487596e+07	1.000000e+00	3.000000e+00	3.000000e+00	2.000
max	2.761200e+04	1.189700e+05	1.983481e+07	1.000000e+00	4.000000e+00	4.000000e+00	3.000

The ranges of values in the numerical columns seem reasonable too, so we may not have to do much data cleaning or correction.

In [35]:
```
df4['IsCorrect'].unique()
```

Out[35]:
```
array([0, 1], dtype=int64)
```

In [36]:
```
df4['UserId'].unique()
```

Out[36]:
```
array([65967, 62121, 50013, ..., 11580, 67004,  2090], dtype=int64)
```

In [37]:
```
df4['AnswerValue'].unique()
```

Out[37]:
```
array([2, 1, 3, 4], dtype=int64)
```

In [38]:
```
df4['GroupId'].unique()
```

Out[38]:
```
array([5026, 9607, 4975, ..., 4346, 6615, 6013], dtype=int64)
```

In [41]:
```
print(df4['GroupId'].count())
```

```
15867850
```

In [42]:
```
df4['Gender'].unique()
```

Out[42]:
```
array([1, 0, 2, 3], dtype=int64)
```

In [ ]:
```

```

In [47]:
```
df4['IsCorrect'].unique()
```

Out[47]:
```
array([0, 1], dtype=int64)
```

In [46]:
```
df4['PremiumPupil'].unique()
```

Out[46]:
```
array([ 0.,  1., nan])
```

In [9]:
```
df4.drop('Confidence', axis = 1, inplace =True) #Now i dropped the Confidence Column
```

In [9]:
```
df4.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 15867850 entries, 0 to 15867849
Data columns (total 13 columns):
 #   Column          Dtype  
---  ------          -----  
 0   QuestionId      int64  
 1   UserId          int64  
 2   AnswerId        int64  
 3   IsCorrect       int64  
 4   CorrectAnswer   int64  
 5   AnswerValue     int64  
 6   Gender          int64  
 7   DateOfBirth     object 
 8   PremiumPupil    float64
 9   DateAnswered    object 
 10  GroupId         int64  
 11  QuizId          int64  
 12  SchemeOfWorkId  float64
dtypes: float64(2), int64(9), object(2)
memory usage: 1.7+ GB
```

In [9]:
```
df4.shape
```

Out[9]:
```
(15867850, 13)
```

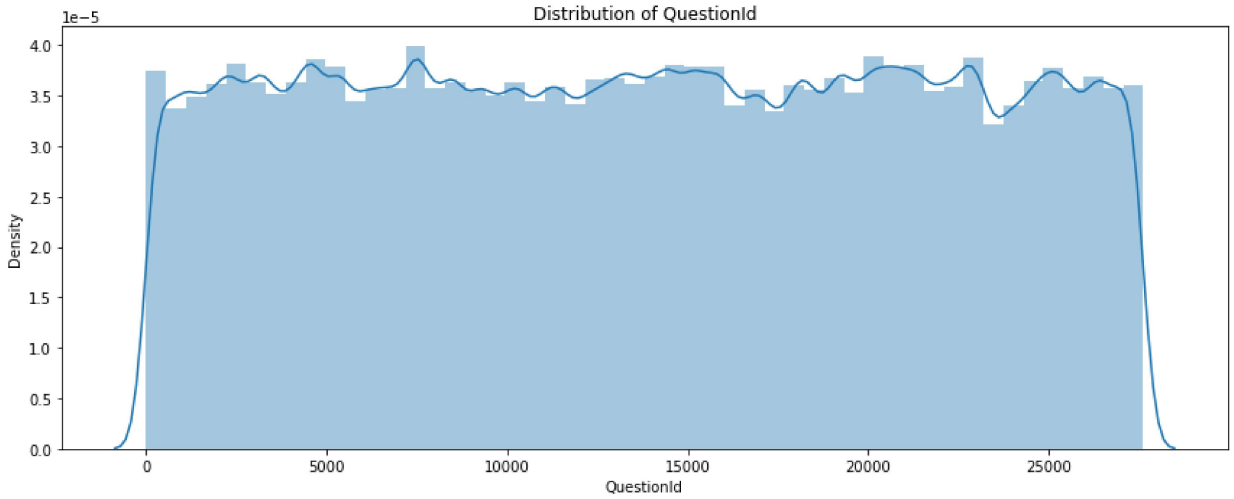
In [50]:
```
def numeric_distribution_plot(df):
    """
    Description : Gives distribution plot for all the numeric features
    in the dataframe passed

    Argument : Dataframe
    """
    for col in df.columns:
        if df[col].dtype != 'object':
            print(df[col].describe())
            plt.figure(figsize=(12,5))
            plt.title("Distribution of "+col)
            ax = sns.distplot(df[col].dropna())
            plt.tight_layout()
            plt.show()
```

In [51]:
```
numeric_distribution_plot(df4)
```

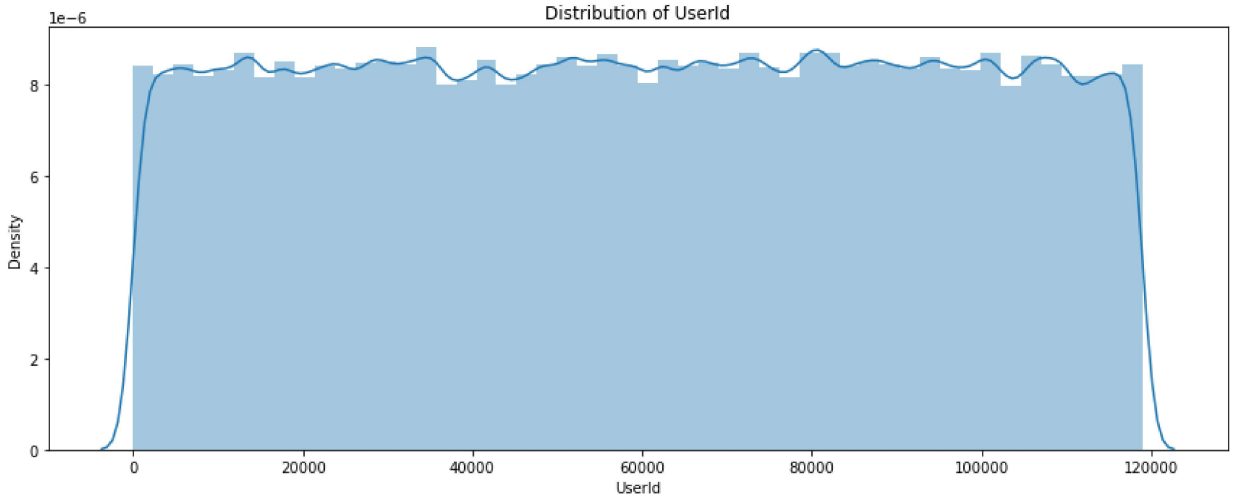
```
count    1.586785e+07
mean     1.380277e+04
std      7.967465e+03
min      0.000000e+00
25%      6.890000e+03
50%      1.383000e+04
75%      2.069400e+04
max      2.761200e+04
Name: QuestionId, dtype: float64
```

```
C:\Users\OLUBAYODE\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



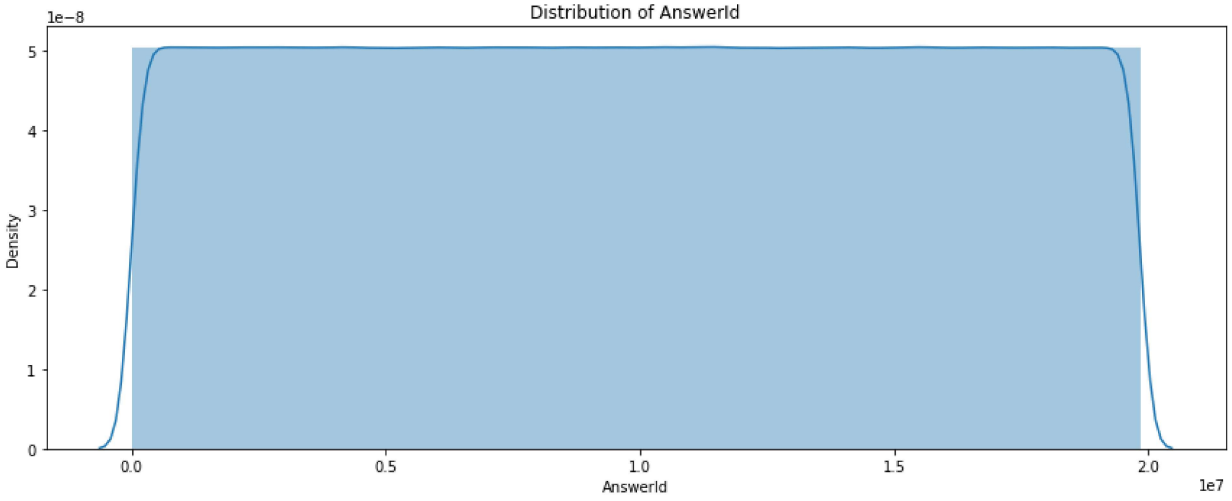

```
count    1.586785e+07
mean     5.952587e+04
std      3.428394e+04
min      0.000000e+00
25%      2.980900e+04
50%      5.958500e+04
75%      8.912500e+04
max      1.189700e+05
Name: UserId, dtype: float64
```

```
C:\Users\OLUBAYODE\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



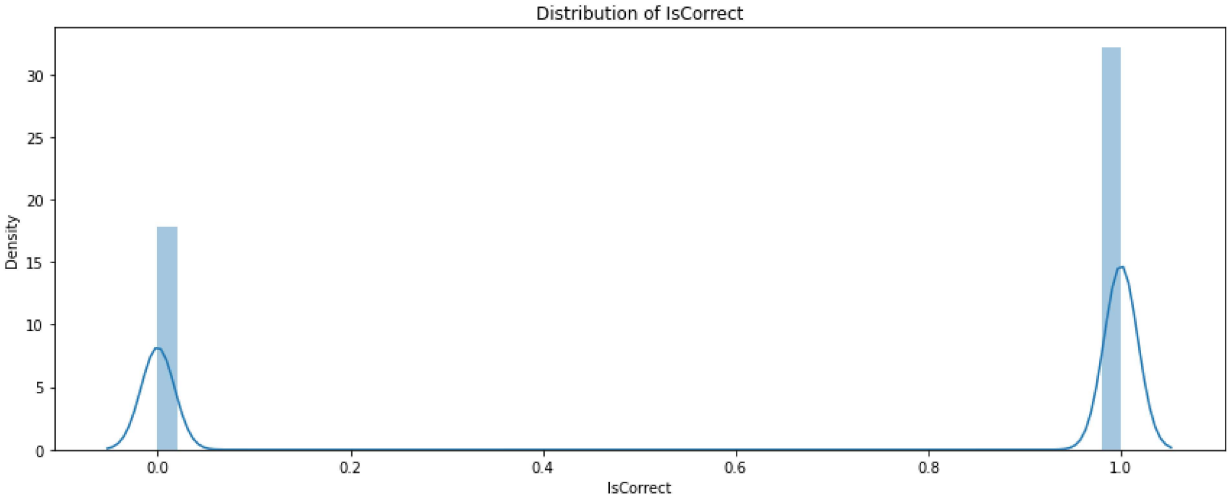
```
count    1.586785e+07
mean     9.916806e+06
std      5.725901e+06
min      0.000000e+00
25%      4.957568e+06
50%      9.916784e+06
75%      1.487596e+07
max      1.983481e+07
Name: AnswerId, dtype: float64
```

```
C:\Users\OLUBAYODE\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



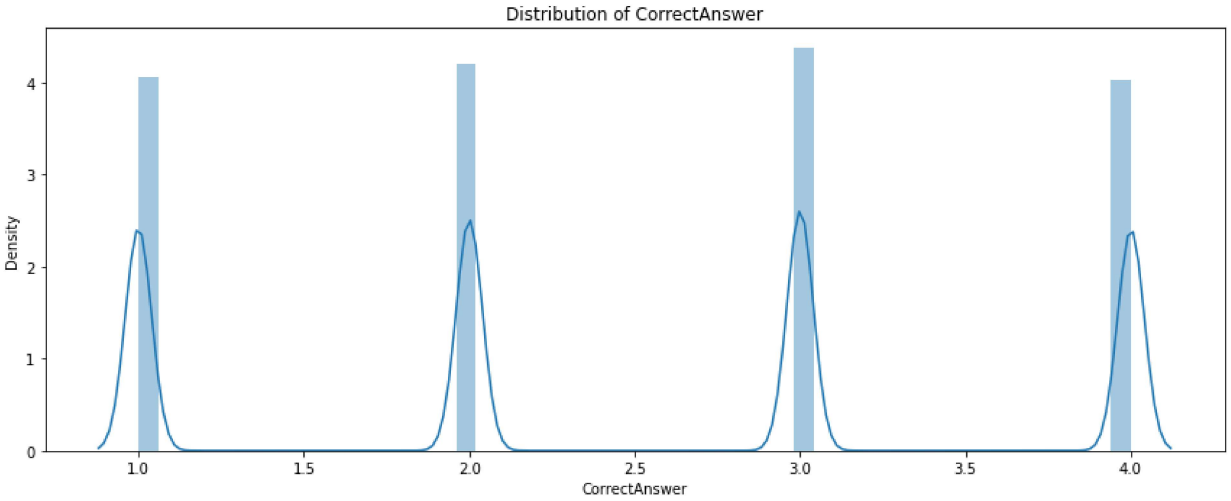

```
count    1.586785e+07
mean     6.429503e-01
std      4.791297e-01
min      0.000000e+00
25%      0.000000e+00
50%      1.000000e+00
75%      1.000000e+00
max      1.000000e+00
Name: IsCorrect, dtype: float64
```

```
C:\Users\OLUBAYODE\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



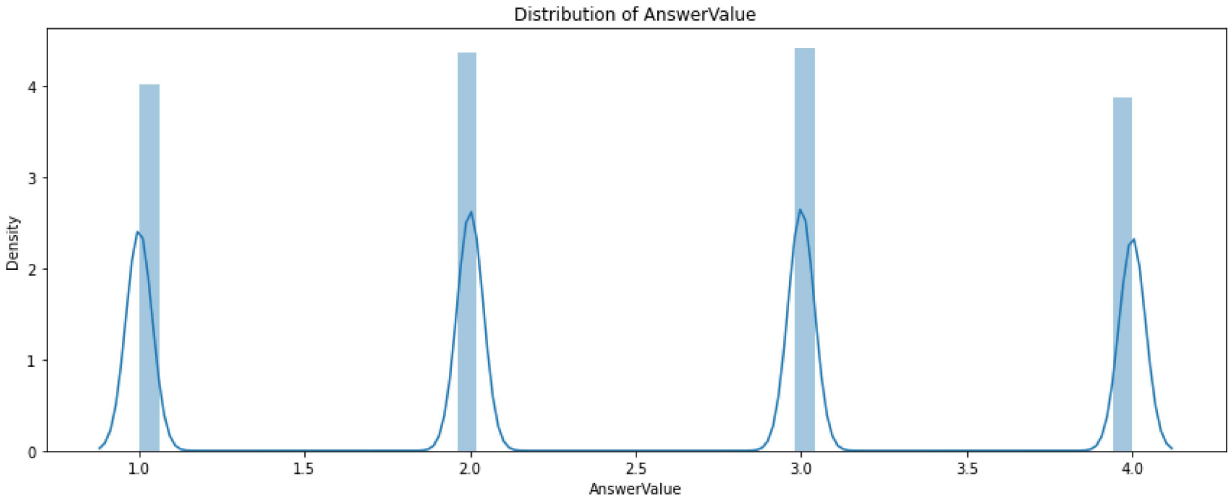

```
count    1.586785e+07
mean     2.502599e+00
std      1.104868e+00
min      1.000000e+00
25%      2.000000e+00
50%      3.000000e+00
75%      3.000000e+00
max      4.000000e+00
Name: CorrectAnswer, dtype: float64
```

```
C:\Users\OLUBAYODE\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



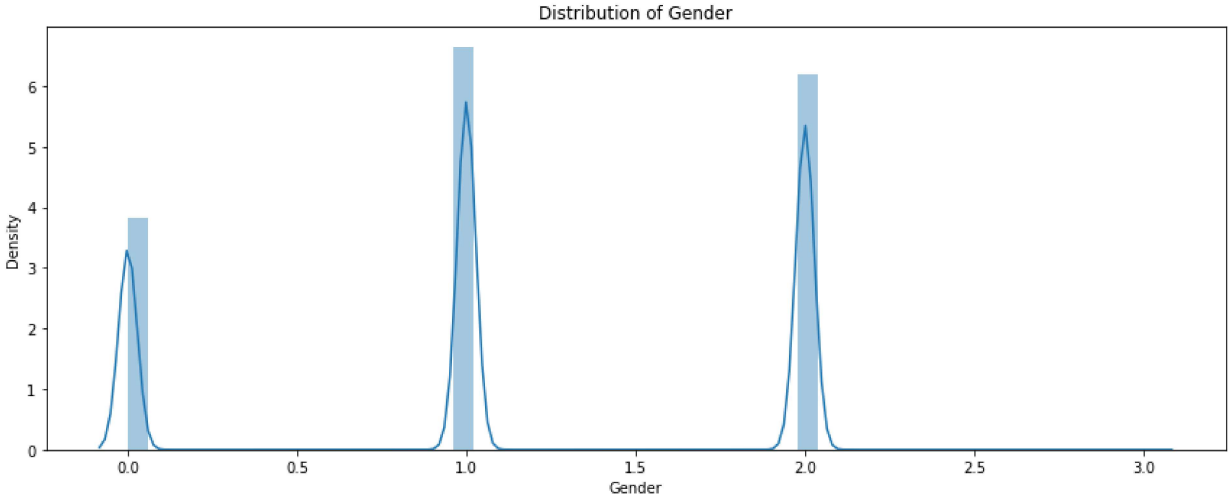

```
count    1.586785e+07
mean     2.488978e+00
std      1.094272e+00
min      1.000000e+00
25%      2.000000e+00
50%      2.000000e+00
75%      3.000000e+00
max      4.000000e+00
Name: AnswerValue, dtype: float64
```

```
C:\Users\OLUBAYODE\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



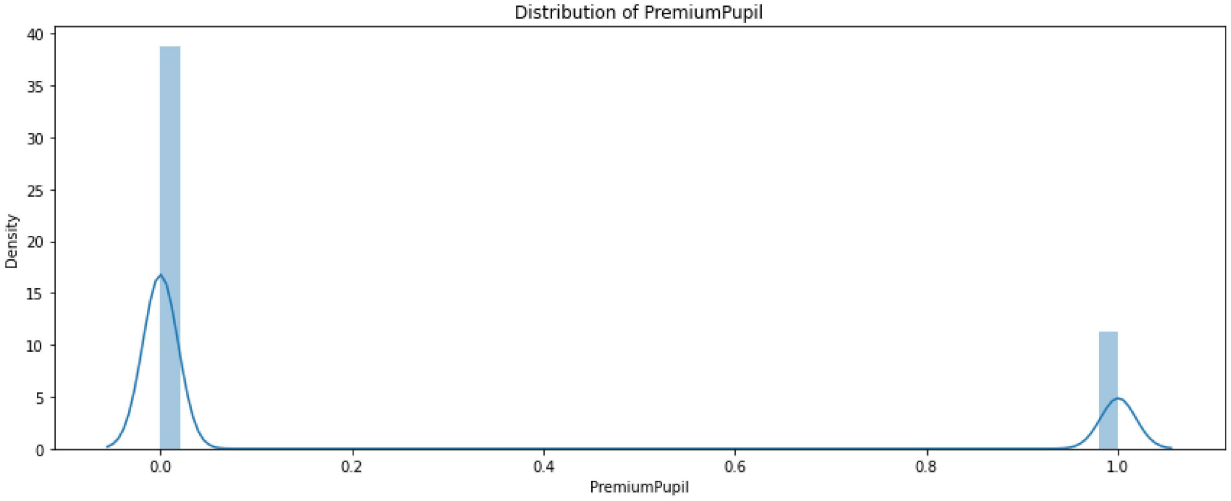

```
count    1.586785e+07
mean     1.142230e+00
std      7.627059e-01
min      0.000000e+00
25%      1.000000e+00
50%      1.000000e+00
75%      2.000000e+00
max      3.000000e+00
Name: Gender, dtype: float64
```

```
C:\Users\OLUBAYODE\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



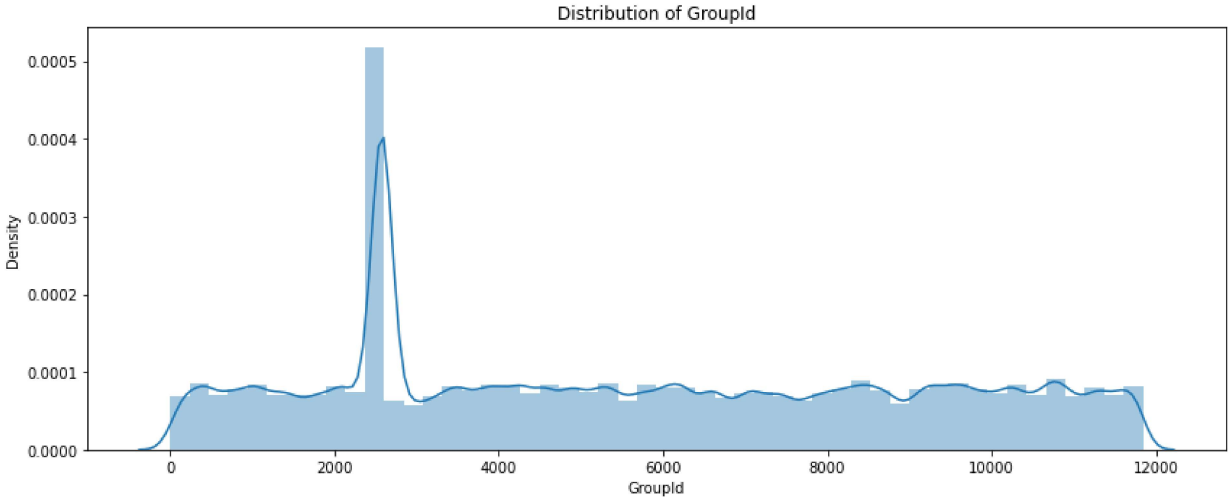
```
count    5.963582e+06
mean     2.249465e-01
std      4.175471e-01
min      0.000000e+00
25%      0.000000e+00
50%      0.000000e+00
75%      0.000000e+00
max      1.000000e+00
Name: PremiumPupil, dtype: float64
```

```
C:\Users\OLUBAYODE\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureW
arning: `distplot` is a deprecated function and will be removed in a future version.
Please adapt your code to use either `displot` (a figure-level function with similar
flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



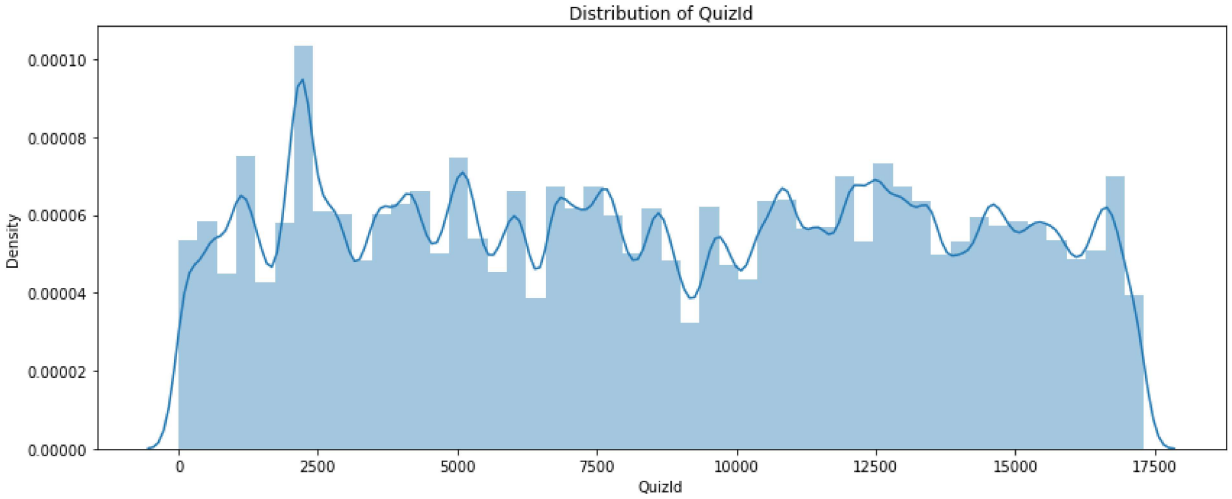

```
count    1.586785e+07
mean     5.611313e+03
std      3.406082e+03
min      0.000000e+00
25%      2.580000e+03
50%      5.259000e+03
75%      8.612000e+03
max      1.184300e+04
Name: GroupId, dtype: float64
```

```
C:\Users\OLUBAYODE\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureW
arning: `distplot` is a deprecated function and will be removed in a future version.
Please adapt your code to use either `displot` (a figure-level function with similar
flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



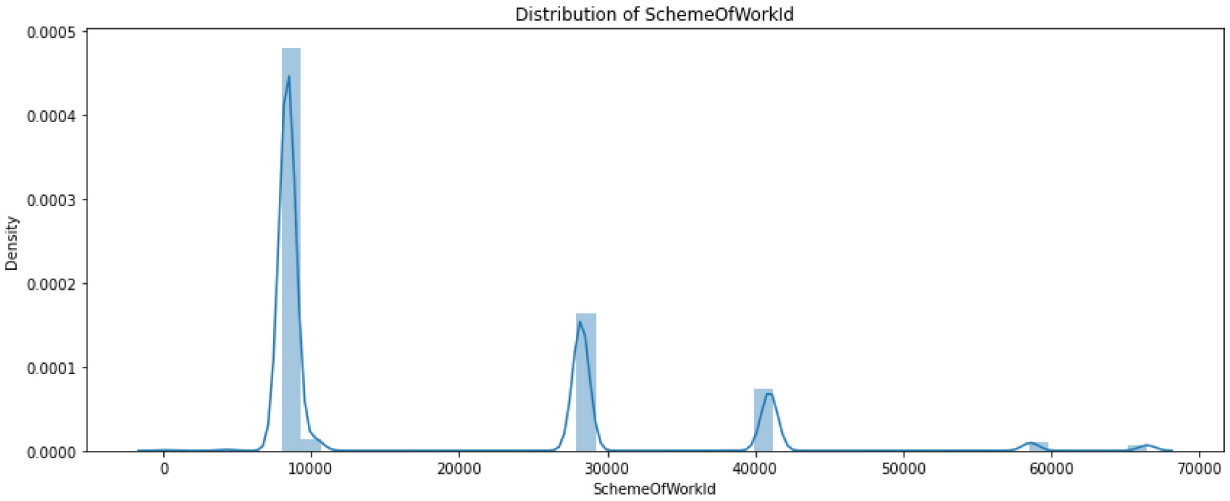

```
count    1.586785e+07
mean     8.525428e+03
std      4.991572e+03
min      0.000000e+00
25%      4.143000e+03
50%      8.465000e+03
75%      1.285200e+04
max      1.730400e+04
Name: QuizId, dtype: float64
```

```
C:\Users\OLUBAYODE\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureW
arning: `distplot` is a deprecated function and will be removed in a future version.
Please adapt your code to use either `displot` (a figure-level function with similar
flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```




```
count    7.782532e+06
mean     1.718552e+04
std      1.335999e+04
min      7.000000e+00
25%      8.386000e+03
50%      8.418000e+03
75%      2.822800e+04
max      6.648800e+04
Name: SchemeOfWorkId, dtype: float64
```

```
C:\Users\OLUBAYODE\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

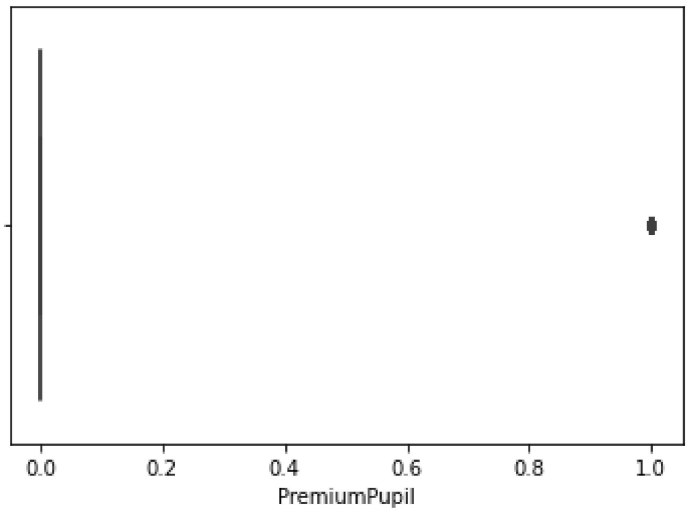


The graphs above shows the behaviour of the features of datasets, some are uniform distribution while some are not

In [52]:
```
sns.boxplot(df4['PremiumPupil'])
```

```
C:\Users\OLUBAYODE\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[52]:
```
<AxesSubplot:xlabel='PremiumPupil'>
```

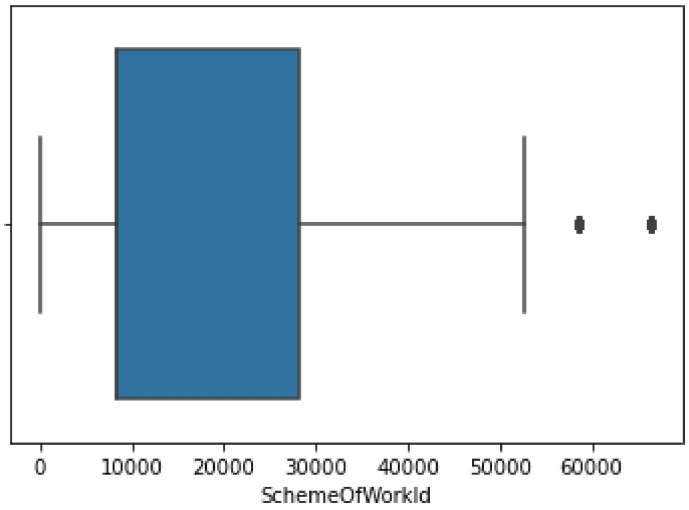


It is very obvious from the BoxPlot graph above that PremiumPupil data isnt skewed so we can fill missing value using mean

In [53]:
```
sns.boxplot(df4['SchemeOfWorkId'])
```

```
C:\Users\OLUBAYODE\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[53]:
```
<AxesSubplot:xlabel='SchemeOfWorkId'>
```



It is very obvious from the graph above that SchemeOfWorkId data is skewed so we can fill missing value using median

DATA PREPROCESSING

In [14]:
```
df4['PremiumPupil'].unique()
```

Out[14]:
```
array([ 0.,  1., nan])
```

In [10]:
```
# df4['PremiumPupil'] = df4['PremiumPupil'].fillna(df4['PremiumPupil'].mean())
```

In [11]:
```
# df4['PremiumPupil'] = df4['PremiumPupil'].fillna(2)
```

In [13]:
```
df4['PremiumPupil'].value_counts()
```

Out[13]:
```
0.0    4622095
1.0    1341487
Name: PremiumPupil, dtype: int64
```

In [10]:
```
df4.shape
```

Out[10]:
```
(15867850, 13)
```

Dropping all Nans Values

In [10]:
```
df4.dropna(inplace = True)  #Drop all nans values
```

In [11]:
```
df4.isnull().sum() #checking the data out if there is still missing value
```

Out[11]:
```
QuestionId        0
UserId            0
AnswerId          0
IsCorrect         0
CorrectAnswer     0
AnswerValue       0
Gender            0
DateOfBirth       0
PremiumPupil      0
DateAnswered      0
GroupId           0
QuizId            0
SchemeOfWorkId    0
dtype: int64
```

In [12]:
```
df4.shape
```

Out[12]:
```
(4181925, 13)
```

In [12]:
```
df4.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4181925 entries, 0 to 15867842
Data columns (total 13 columns):
 #   Column          Dtype  
---  ------          -----  
 0   QuestionId      int64  
 1   UserId          int64  
 2   AnswerId        int64  
 3   IsCorrect       int64  
 4   CorrectAnswer   int64  
 5   AnswerValue     int64  
 6   Gender          int64  
 7   DateOfBirth     object 
 8   PremiumPupil    float64
 9   DateAnswered    object 
 10  GroupId         int64  
 11  QuizId          int64  
 12  SchemeOfWorkId  float64
dtypes: float64(2), int64(9), object(2)
memory usage: 446.7+ MB
```

In [12]:
```
df4.head()
```

Out[12]:

	QuestionId	UserId	AnswerId	IsCorrect	CorrectAnswer	AnswerValue	Gender	DateOfBirth	Pre
0	16997	65967	12453206	0	4	2	1	2002-11-01 00:00:00.000	
2	15911	50013	13598796	0	3	1	1	2004-09-01 00:00:00.000	
10	21568	87935	17488010	1	4	4	1	2003-11-01 00:00:00.000	
11	15962	77530	11435172	0	2	3	1	2003-12-01 00:00:00.000	
14	20640	99529	12214186	0	3	4	2	2003-09-01 00:00:00.000	

In [14]:
```
df4.tail()
```

Out[14]:

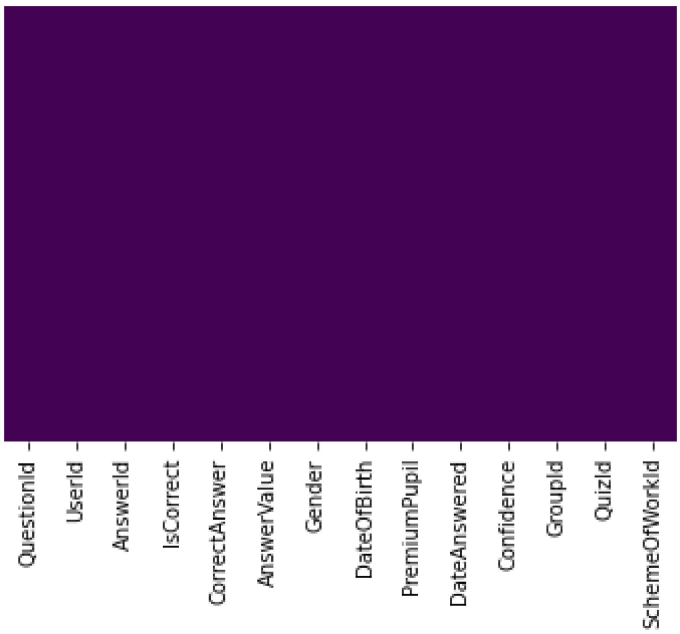
	QuestionId	UserId	AnswerId	IsCorrect	CorrectAnswer	AnswerValue	Gender	DateOfBir
15867825	24020	4986	1623250	1	4	4	1	2004-03-00:00:00.0
15867834	17701	74706	13409775	0	1	4	2	2003-10-00:00:00.0
15867840	5257	64370	8654825	0	2	3	1	2003-10-00:00:00.0
15867841	20974	103299	15005781	0	3	4	1	2008-08-00:00:00.0
15867842	13734	7108	3854266	0	4	2	2	2007-10-00:00:00.0

In [101…
```
df4['GroupId'].unique()
```

Out[101]:
```
array([5188, 1249, 6729, ..., 1154, 3843, 7094], dtype=int64)
```

In [13]:
```
import seaborn as sns
sns.heatmap(df4.isnull(),yticklabels=False,cbar=False,cmap='viridis') #Checking to see
```

Out[13]:
```
<AxesSubplot:>
```



Feature Engineering

Convert the DateAnswered column from its original data type to DateTime objects

In [13]:
```
#Converting to datetime object for easy data preprocessing i.e pd.to_datetime(df['Date
df4['DateAnswered'] = pd.to_datetime(df4['DateAnswered'], format='%Y-%m-%d %H:%M:%S')
```

```
df4['DateAnswered'].dtypes
time = df4['DateAnswered'].iloc[0]
time
```

Out[13]:
```
Timestamp('2019-04-24 13:42:00')
```

Convert the DateOfBirth column from its original data type to DateTime objects

In [14]:
```
#Converting to datetime object for easy data preprocessing i.e pd.to_datetime(df['Date
df4['DateOfBirth'] = pd.to_datetime(df4['DateOfBirth'], format='%Y-%m-%d %H:%M:%S') #,
df4['DateOfBirth'].dtypes
time_ = df4['DateOfBirth'].iloc[0]
time_
```

Out[14]:
```
Timestamp('2002-11-01 00:00:00')
```

Grabing specific attributes from a Datetime object by calling them Now that the timestamp column are actually DateTime objects, I use .apply() to create 3 new columns called Hour, Month, and Day of Week for the DateOfAnswered. I created these columns based off of the timeStamp column.

In [15]:
```
# Grabing specific attributes from a Datetime object by calling them Now that the time
df4['HourAnswered'] = df4['DateAnswered'].apply(lambda x: x.hour)
df4['MonthAnswered'] =df4['DateAnswered'].apply(lambda x: x.month)
df4['YearAnswered'] = df4['DateAnswered'].apply(lambda x: x.year)
df4['Day of Week Answered'] = df4['DateAnswered'].apply(lambda x: x.dayofweek)
df4.head()
```

Out[15]:

	QuestionId	UserId	AnswerId	IsCorrect	CorrectAnswer	AnswerValue	Gender	DateOfBirth	Pre
0	16997	65967	12453206	0	4	2	1	2002-11-01	
2	15911	50013	13598796	0	3	1	1	2004-09-01	
10	21568	87935	17488010	1	4	4	1	2003-11-01	
11	15962	77530	11435172	0	2	3	1	2003-12-01	
14	20640	99529	12214186	0	3	4	2	2003-09-01	

Visualizing my new column Day of Week to be an integer 0-6. By Changing the integer values to the actual string names of day of the week: The dictionary below is a guide.

{0:'Mon',1:'Tue',2:'Wed',3:'Thu',4:'Fri',5:'Sat',6:'Sun'}

In [16]:
```
Day_of__the_Week_Ans = {0:'Mon',1:'Tue',2:'Wed',3:'Thu',4:'Fri',5:'Sat',6:'Sun'}
```

In [17]:
```
df4['Day of Week Ans'] = df4['Day of Week Answered'].apply(lambda x: Day_of__the_Week_
df4.head()
```

Out[17]:

	QuestionId	UserId	AnswerId	IsCorrect	CorrectAnswer	AnswerValue	Gender	DateOfBirth	Pre
0	16997	65967	12453206	0	4	2	1	2002-11-01	
2	15911	50013	13598796	0	3	1	1	2004-09-01	
10	21568	87935	17488010	1	4	4	1	2003-11-01	
11	15962	77530	11435172	0	2	3	1	2003-12-01	
14	20640	99529	12214186	0	3	4	2	2003-09-01	

Exracting Year from the DateOfBirth so that i can calculate the age of the students

In [18]:
```
df4['YearBirth'] = df4['DateOfBirth'].apply(lambda x: x.year) #Exracting Year from the
```

In [16]:
```
df4.head()
```

Out[16]:

	QuestionId	UserId	AnswerId	IsCorrect	CorrectAnswer	AnswerValue	Gender	DateOfBirth	Pre
0	16997	65967	12453206	0	4	2	1	2002-11-01	
2	15911	50013	13598796	0	3	1	1	2004-09-01	
10	21568	87935	17488010	1	4	4	1	2003-11-01	
11	15962	77530	11435172	0	2	3	1	2003-12-01	
14	20640	99529	12214186	0	3	4	2	2003-09-01	

Age of each students when a students was when answered the questions

In [19]:
```
df4['Age'] = df4['YearAnswered'] - df4['YearBirth'] #Creating Age column  of each stud
# df4['Age']= df4['Age']/np.timedelta64(1, 'y')
```

```
df4['Age'] = df4['Age'].astype(int)
```

In [22]:
```
df4.head()
```

Out[22]:

	QuestionId	UserId	AnswerId	IsCorrect	CorrectAnswer	AnswerValue	Gender	DateOfBirth	Pre
0	16997	65967	12453206	0	4	2	1	2002-11-01	
2	15911	50013	13598796	0	3	1	1	2004-09-01	
10	21568	87935	17488010	1	4	4	1	2003-11-01	
11	15962	77530	11435172	0	2	3	1	2003-12-01	
14	20640	99529	12214186	0	3	4	2	2003-09-01	

In [25]:
```
df4.describe()
```

Out[25]:

	QuestionId	UserId	AnswerId	IsCorrect	CorrectAnswer	AnswerValue	
count	4.181925e+06	4.181925e+06	4.181925e+06	4.181925e+06	4.181925e+06	4.181925e+06	4.181
mean	1.383825e+04	5.914567e+04	9.916765e+06	5.582764e-01	2.490195e+00	2.485801e+00	1.478
std	7.999773e+03	3.434876e+04	5.725922e+06	4.965923e-01	1.109890e+00	1.094755e+00	4.995
min	2.000000e+00	1.000000e+00	3.000000e+00	0.000000e+00	1.000000e+00	1.000000e+00	0.000
25%	6.879000e+03	2.914600e+04	4.959380e+06	0.000000e+00	1.000000e+00	2.000000e+00	1.000
50%	1.389500e+04	5.940200e+04	9.914638e+06	1.000000e+00	2.000000e+00	2.000000e+00	1.000
75%	2.083100e+04	8.875300e+04	1.487752e+07	1.000000e+00	3.000000e+00	3.000000e+00	2.000
max	2.761200e+04	1.189700e+05	1.983480e+07	1.000000e+00	4.000000e+00	4.000000e+00	2.000

In [ ]:
```
# Dropping the Day of Week Ans since it is not a numerical value
#df4.drop('Day of Week Ans', inplace = True)
```

Exploratory Analysis and Visualization

Let's explore the data by visualizing the distribution of values in some columns of the dataset, and the relationships between "IsCorrect","AnswerValue" and other columns.

We'll use libraries Matplotlib, Seaborn and Plotly for visualization.

In [20]:
```
import plotly.express as px
```

```
import matplotlib
```

The following settings will improve the default style and font sizes for our charts

In [21]:
```
sns.set_style('darkgrid')
matplotlib.rcParams['font.size'] = 14
matplotlib.rcParams['figure.figsize'] = (10, 6)
matplotlib.rcParams['figure.facecolor'] = '#00000000'
```

Age

Age is a numeric column. The minimum age in the dataset is 6 and the maximum age is 19. Check the graph plotted above. Thus, we can visualize the distribution of age using a histogram with 47 bins (one for each year) and a box plot. We'll use plotly to make the chart interactive, but you can create similar charts using Seaborn.

In [28]:
```
df4.Age.describe()
```

Out[28]:
```
count    4.181925e+06
mean     1.399566e+01
std      1.957412e+00
min      5.000000e+00
25%      1.300000e+01
50%      1.500000e+01
75%      1.500000e+01
max      3.800000e+01
Name: Age, dtype: float64
```

In [29]:
```
fig = px.histogram(df4,
                   x='Age',
                   marginal='box',
                   color = 'IsCorrect',
                   color_discrete_sequence=['red', 'grey'],
                   nbins=47,
                   title='Distribution of Age')
fig.update_layout(bargap=0.1)
fig.show()
```

The distribution of ages in the dataset is not uniform distribution neither does it follow a Gaussian distribution, with 100k+ students at age 12 and 15,We can make the following observations from the above graph: The distribution follows a power law

Insight: why there are many students between ages 11 and 16 givien more questions than those below 11 years? Though most Students with age lower than 11 years still pass the exams

More Questions were given to students that are older than 11 years between age 11 and age 16. Meaning that students who are younger are have lesser chances of answering the plenty question.

CATEGORICAL COLUMNS VISUALIZATION

IsCorrect

Let's visualize the distribution of "IsCorrect" i.e. the corrects answers. This is the column we're trying to predict for task 1. Let's also use the categorical column "PremiumPupil" to distinguish the IsCorrect for PremiumPupil.

In [ ]:
```
fig = px.histogram(df4,
```

```
                    x='IsCorrect',
                    marginal='box',
                    color='PremiumPupil',
                    color_discrete_sequence=['red', 'purple'],
                    title='Correct Answered Questions')
fig.update_layout(bargap=0.1)
fig.show()
```

There are many 0 PremiumPupil close to 80% students that got the questions correctly, Unlike the PremiumPupil 1 students so also in the wrong questions when IscCorrect is 0 there are . This shows that finaciancIly disdavantage students without School free meals are likely to select the corrects answers, there are PremiumPupil 0 (financially disadvantage students that selcted the wrong answers. Why this is so should be investigated.

From the graph above i suggested that the school has a high level of low disadavantage students compare to financial advantage students

Visualization of the distribution of IsCorrect in connection with other factors like "gender" and "Day of Week Ans".

In [51]:
```
fig = px.histogram(df4,
                    x = 'IsCorrect',
                    marginal='box',
                    color = 'Gender',
                    color_discrete_sequence = ["blue","red","green"],
                    title = 'Different IsCorrect over genders'
                   )
fig.update_layout(bargap=0.1)
fig.show()
```

There are more feamle that got the questions correctly than male while there equal ratio of male to female that got the answer wrongly. There is a low significant level of unknow gender 0 correctly, you can see that in green color. This can as well be investigated

In [50]:
```
fig = px.histogram(df4,
                   x = 'Day of Week Ans',
                   marginal='box',
                   color = 'IsCorrect',
                   color_discrete_sequence = ["blue","yellow"],
                   title = 'Different Correct Answered over genders'
                  )
fig.update_layout(bargap=0.1)
fig.show()
```

There is a unifrom distribution between day Tue, Sun, Mon among the students that select the correct answer, while day Thurs has the highest level of day with students that picked correct answered, sp studenst tends to pass very wel on thursday. I suggested thursday is the day many of the students do come to school to answer the questions probably because thursday might be a day with a less stress day of the week for the students. But this can be furthered be investigated. Saturday has the hihest ratio of 26.74k students selecting the correct answers to number of 12.409k students picking wrong answers.

I will recommend saturday for students to answer the questions.

HoursAnswered

In [65]:
```
fig = px.histogram(df4,
                   x='HourAnswered',
                   marginal='box',
                   color = 'IsCorrect',
                   color_discrete_sequence=['red', 'grey'],
                   nbins=47,
                   title='Distribution of HoursAnswered')
fig.update_layout(bargap=0.1)
fig.show()
```

Here in the distribution of Hours take to answer a question over IsCorrect we see that question answered bewtween 15 to 20 hours have highest number of correct Answers.

MonthAnswered

In [74]:
```
fig = px.histogram(df4,
                   x='MonthAnswered',
                   marginal='box',
                   color = 'IsCorrect',
                   color_discrete_sequence=['red', 'grey'],
                   nbins=47,
                   title='Distribution of MonthAnswered')
fig.update_layout(bargap=0.1)
fig.show()
```

lesser questions are answered during month May,June, July, Auguest, Probably it is alway most likely to be a summer holiday.

AnswerValue

In [79]:
```
fig = px.histogram(df4,
                   x='AnswerValue',
                   marginal='box',
                   color='IsCorrect',
                   color_discrete_sequence=['red', 'purple'],
                   title='Correct Valued of answered Questions')
fig.update_layout(bargap=0.1)
fig.show()
```

There is a unifrom distribution between anservalue of 1,2,3 for correct answers except in correct value 4

NUMERICAL VISUALIZATION

QUIZID DISTRIBUTION

In [100…

```
fig = px.histogram(df4,
                   x='QuizId',
                   marginal='box',
                   color = 'IsCorrect',
                   color_discrete_sequence=['red', 'grey'],
                   nbins=47,
                   title='Distribution of QuizId')
fig.update_layout(bargap=0.1)
fig.show()
```

There are fewer total number of people that passed the quiz compare to those that didnt pass it

GROUPID DISTRIBUTION

In [106…

```
fig = px.histogram(df4,
                   x='GroupId',
                   marginal='box',
                   color = 'IsCorrect',
                   color_discrete_sequence=['blue', 'grey'],
                   nbins=47,
                   title='Distribution of GroupId')
fig.update_layout(bargap=0.1)
fig.show()
```

This follows the same pattern as the graph above

Visualizing how the "QuizId" column is related to other columns ("IsCorrect", "AnswerValue").

In [84]:
```
px.violin(df4,x = 'IsCorrect',y = 'QuizId')
```

There doesn't seem to have a strong trend among this variables.

In [86]:
```
px.violin(df4,x = 'AnswerValue',y = 'QuizId')
```

There doesn't seem to have a strong trend among this variables

In [88]:
```
px.violin(df4,x = 'AnswerValue',y = 'Age')
```

In [89]:
```
px.violin(df4,x = 'IsCorrect',y = 'Age')
```

In [ ]:

In [90]:
```
px.violin(df4,x = 'IsCorrect',y = 'HourAnswered')
```

The above violin plot doesn't seem to have a strong trend among this variables

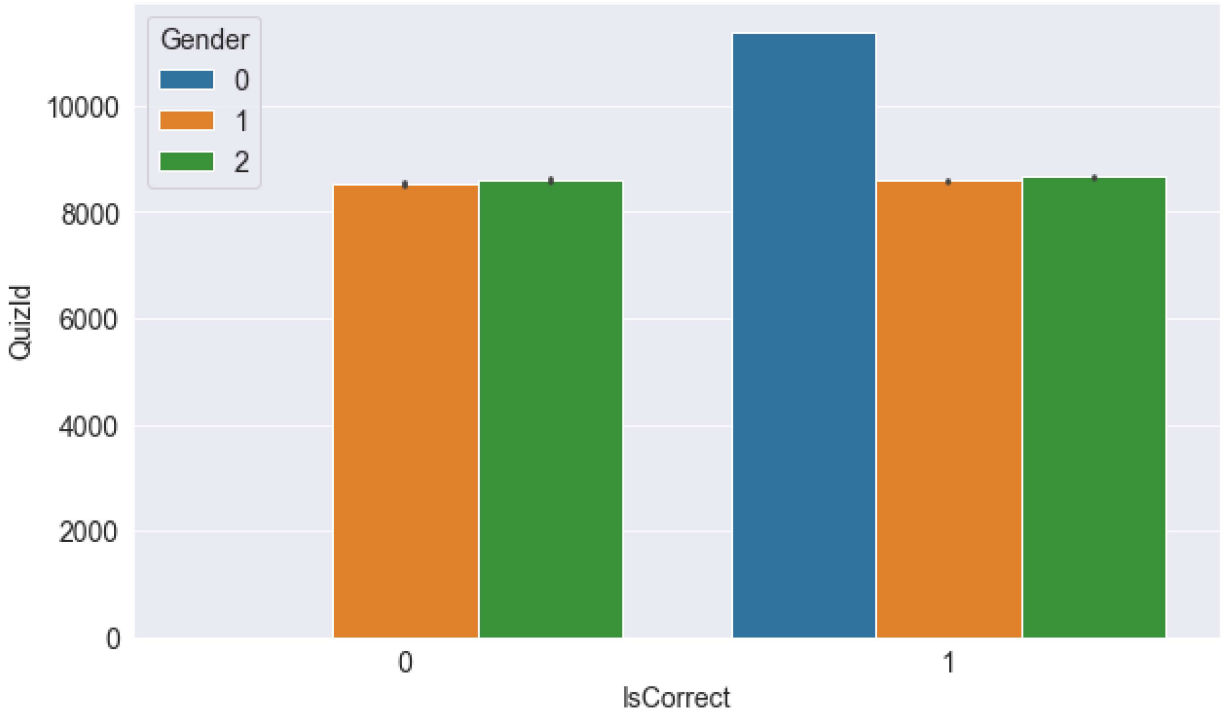
In [94]:
```
px.violin(df4,x = 'HourAnswered',y = 'Age', color = "IsCorrect")
```

There seems to be a trend here, the longer the Hours, the higher the age. There are many answered questions withn the hour of 5 and 20 hours

CATEGORICAL AND NUMERICAL RELATIONSHIP VISUALIZATION

In [95]:
```
sns.barplot(data = df4,x = 'IsCorrect',y = 'QuizId',hue = "Gender")
```

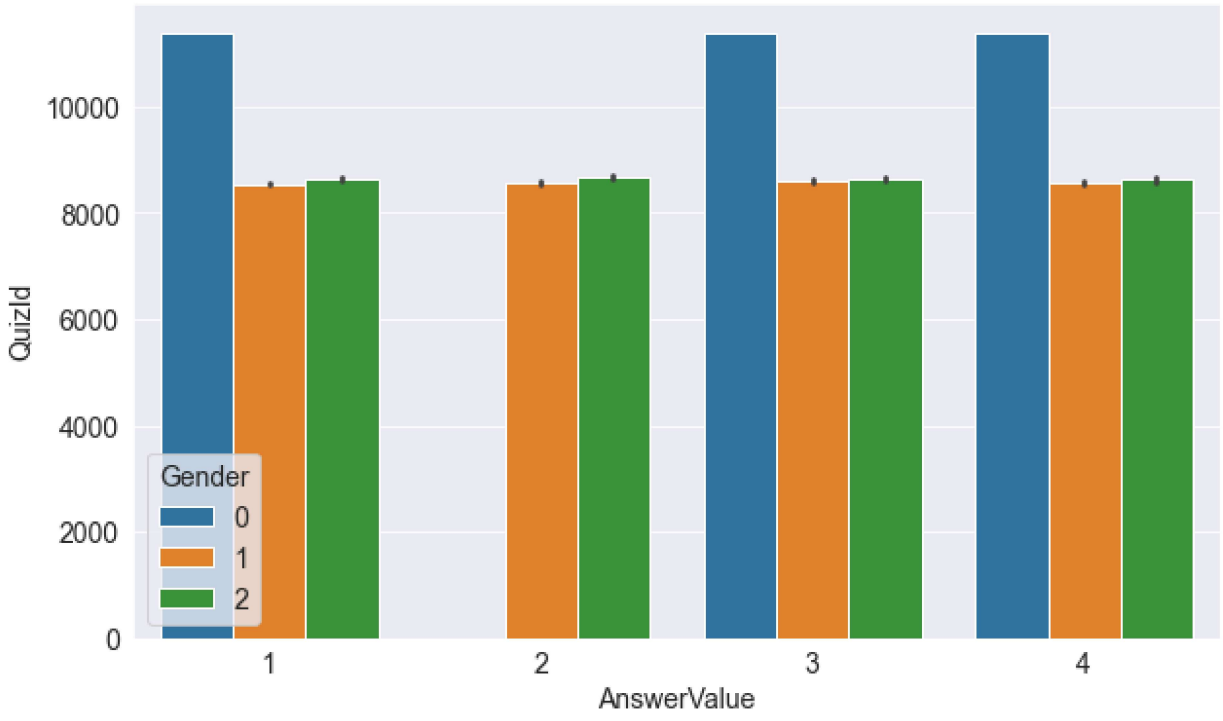
Out[95]:
```
<AxesSubplot:xlabel='IsCorrect', ylabel='QuizId'>
```



These Plot shows that no female selected wrong answer for the Quiz. Seems the guirls are the smartest in the class

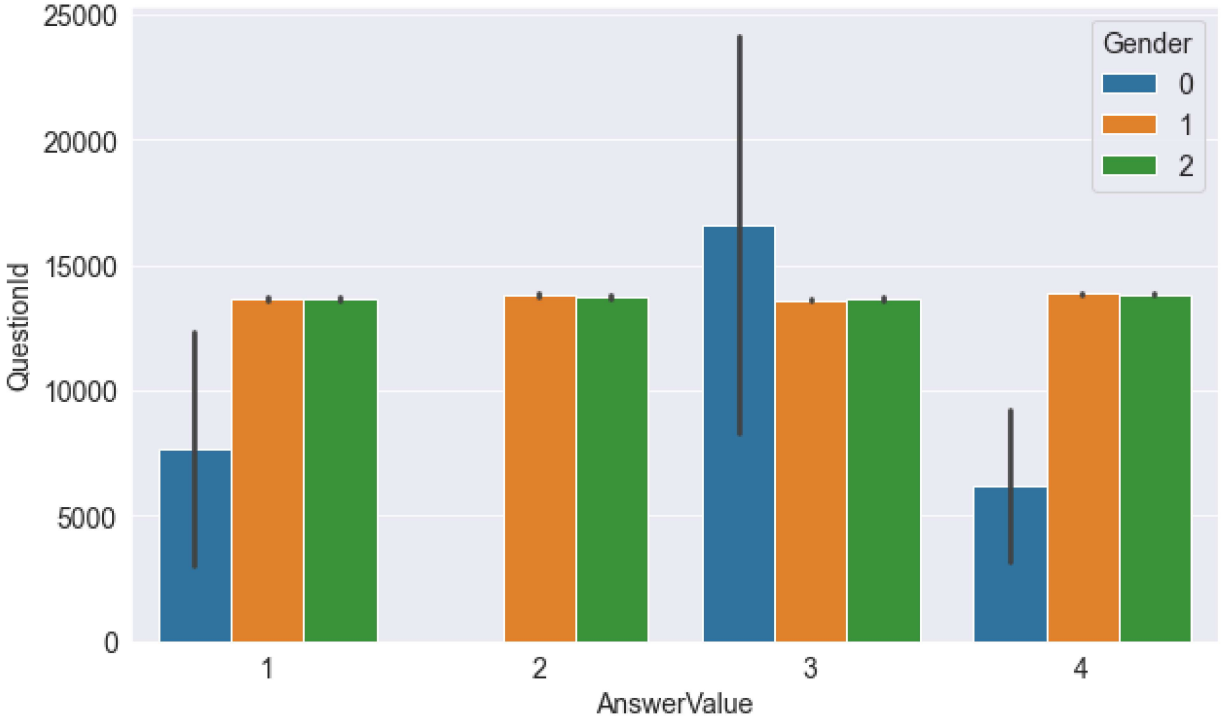
In [96]:
```
sns.barplot(data = df4,x = 'AnswerValue',y = 'QuizId',hue = "Gender")
```

Out[96]:
```
<AxesSubplot:xlabel='AnswerValue', ylabel='QuizId'>
```



In [99]:
```
sns.barplot(data = df4,x = 'AnswerValue',y = 'QuestionId',hue = "Gender")
```

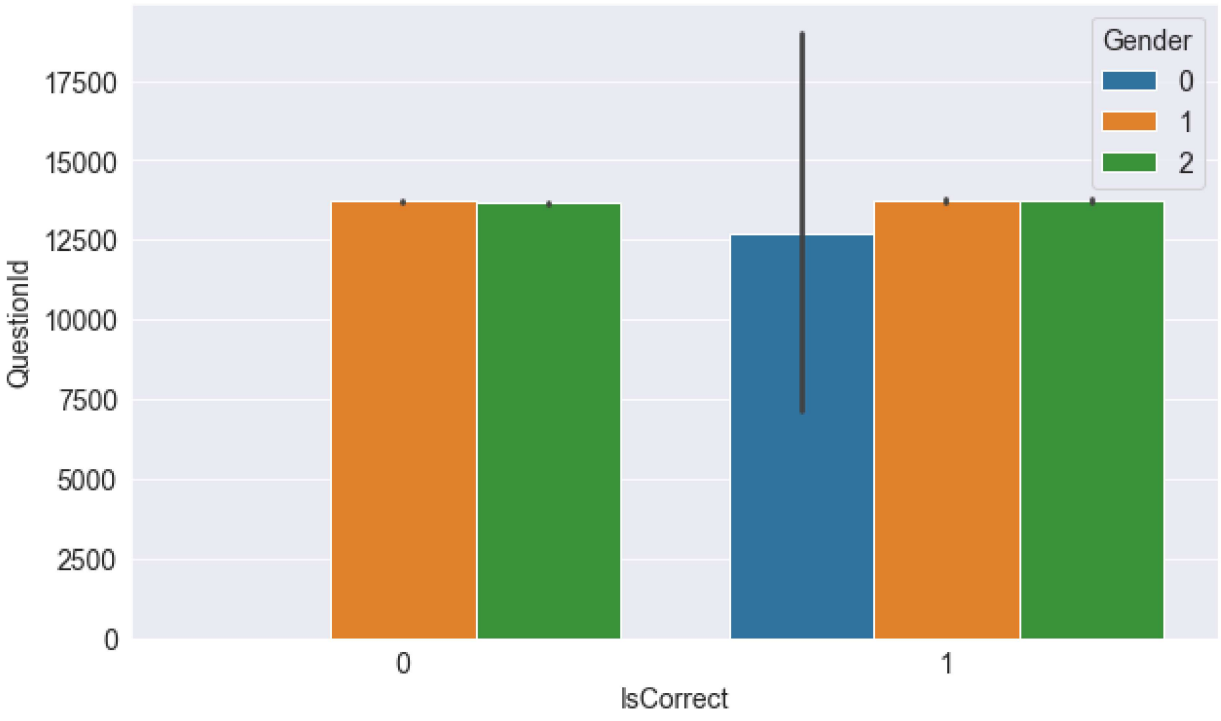
Out[99]:
```
<AxesSubplot:xlabel='AnswerValue', ylabel='QuestionId'>
```



These last two plots still shows that female has the number in selecting the right answer value excpet in answervalue 2. Reasons for this must be investigated

In [97]:
```
sns.barplot(data = df4,x = 'IsCorrect',y = 'QuestionId',hue = "Gender")
```

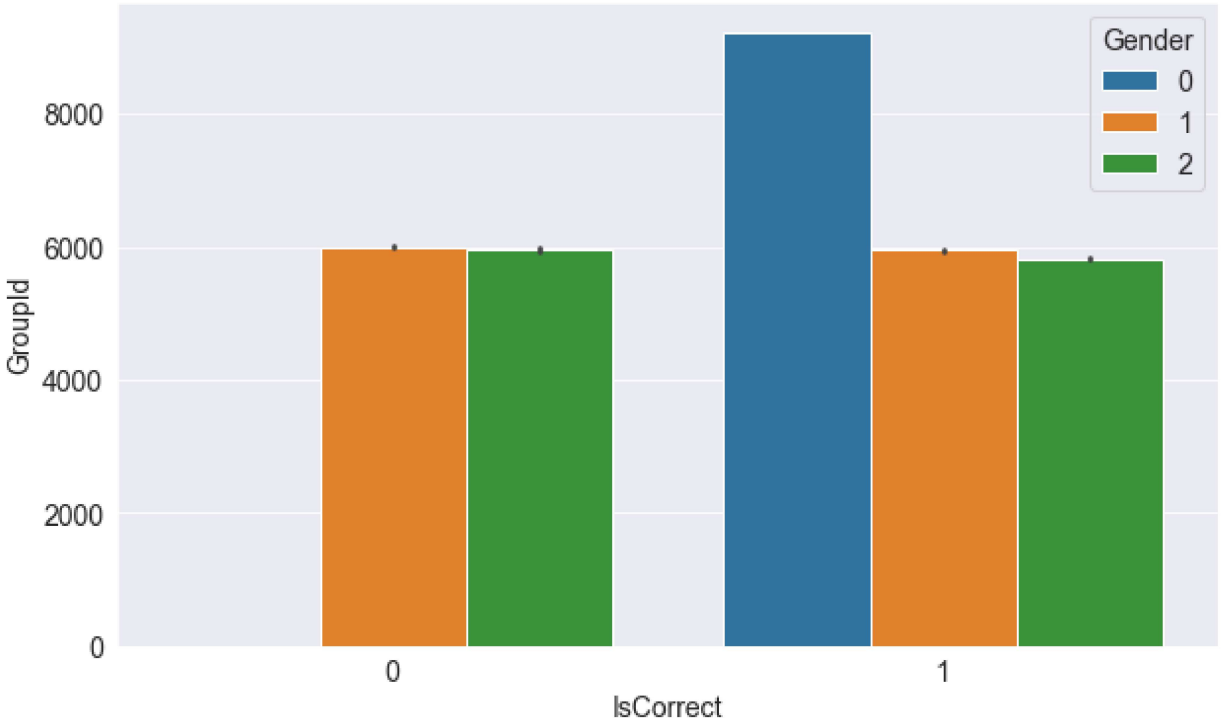
Out[97]:
```
<AxesSubplot:xlabel='IsCorrect', ylabel='QuestionId'>
```



This Plot suports our inference from previous plots of QuizId and IsCorrect plot. This plot shows no females ddint picked InCorrect Answer.The females are smarter than the males

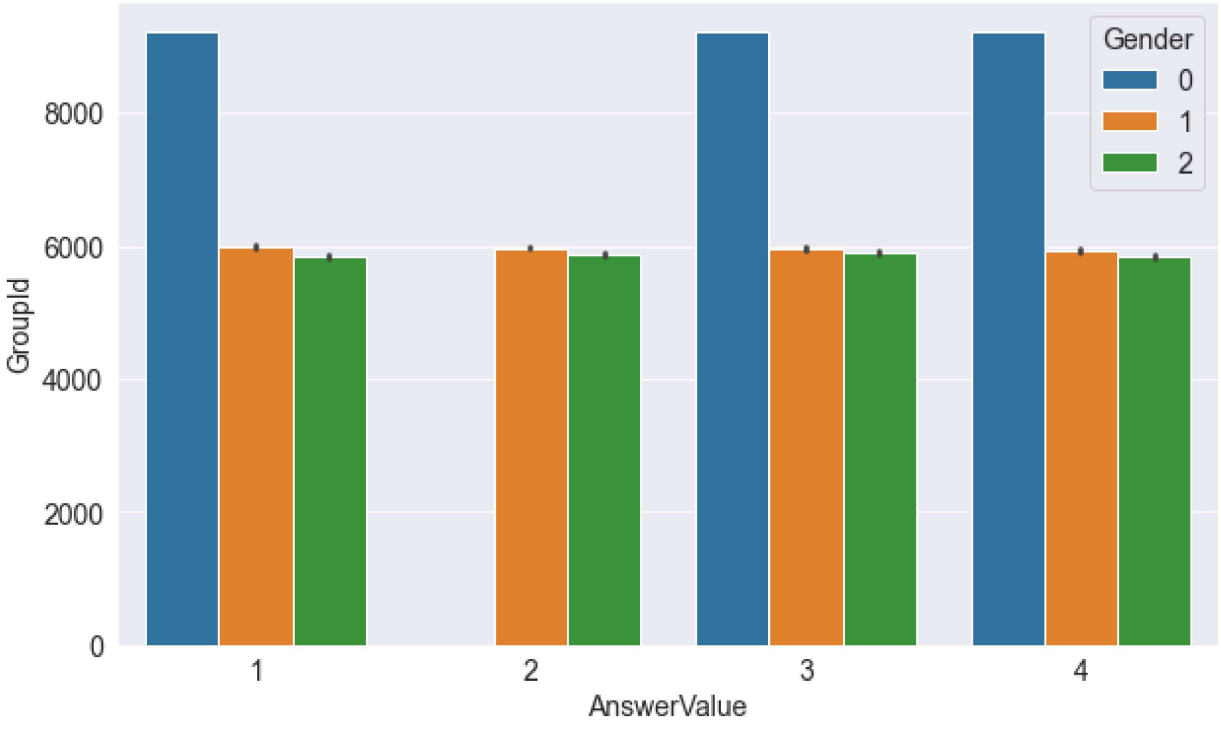
In [102…
```
sns.barplot(data = df4,x = 'IsCorrect',y = 'GroupId',hue = "Gender")
```

Out[102]: <AxesSubplot:xlabel='IsCorrect', ylabel='GroupId'>



In [103…
```
sns.barplot(data = df4,x = 'AnswerValue',y = 'GroupId',hue = "Gender")
```

Out[103]: <AxesSubplot:xlabel='AnswerValue', ylabel='GroupId'>



Correlation

As you can tell from the analysis, the values in some columns are more closely related to the values in "IsCorrect" and "AnswerValue" compared to other columns. E.g. "HourAnswered" and

"ISCorrect" seem to grow together, whereas some doesn't.

This relationship is often expressed numerically using a measure called the correlation coefficient, which can be computed using the .corr method of a Pandas series

In [24]:
```
df4.IsCorrect.corr(df4.Age)
```

Out[24]:
```
-0.03663529906221947
```

In [25]:
```
df4.AnswerValue.corr(df4.Age)
```

Out[25]:
```
-0.009126366724915033
```

In [26]:
```
df4.IsCorrect.corr(df4.HourAnswered)
```

Out[26]:
```
0.025150678519481846
```

In [27]:
```
df4.AnswerValue.corr(df4.HourAnswered)
```

Out[27]:
```
-0.0028718606682533896
```

In [28]:
```
df4.Gender.corr(df4.IsCorrect)
```

Out[28]:
```
-0.028378188352729476
```

In [29]:
```
df4.Gender.corr(df4.AnswerValue)
```

Out[29]:
```
0.004684896807754907
```

Here's how correlation coefficients can be interpreted

Strength: The greater the absolute value of the correlation coefficient, the stronger the relationship.

The extreme values of -1 and 1 indicate a perfectly linear relationship where a change in one variable is accompanied by a perfectly consistent change in the other. For these relationships, all of the data points fall on a line. In practice, you won't see either type of perfect relationship.

A coefficient of zero represents no linear relationship. As one variable increases, there is no tendency in the other variable to either increase or decrease.

When the value is in-between 0 and +1/-1, there is a relationship, but the points don't all fall on a line. As r approaches -1 or 1, the strength of the relationship increases and the data points tend to fall closer to a line.

Direction: The sign of the correlation coefficient represents the direction of the relationship.

Positive coefficients indicate that when the value of one variable increases, the value of the other variable also tends to increase. Positive relationships produce an upward slope on a scatterplot.

Negative coefficients represent cases when the value of one variable increases, the value of the other variable tends to decrease. Negative relationships produce a downward slope.

Pandas dataframes also provide a .corr method to compute the correlation coefficients between all pairs of numeric columns.

In [30]:
```
df4.corr()
```

Out[30]:

	QuestionId	UserId	AnswerId	IsCorrect	CorrectAnswer	AnswerValue	Gende
QuestionId	1.000000	-0.000482	-0.000474	0.003461	0.020430	0.002032	-0.000418
UserId	-0.000482	1.000000	0.000663	0.005306	-0.001082	-0.000947	0.016718
AnswerId	-0.000474	0.000663	1.000000	-0.000317	0.000469	0.000152	-0.000143
IsCorrect	0.003461	0.005306	-0.000317	1.000000	-0.008872	-0.004482	-0.028378
CorrectAnswer	0.020430	-0.001082	0.000469	-0.008872	1.000000	0.419281	0.002939
AnswerValue	0.002032	-0.000947	0.000152	-0.004482	0.419281	1.000000	0.004685
Gender	-0.000418	0.016718	-0.000143	-0.028378	0.002939	0.004685	1.000000
PremiumPupil	0.001929	0.004511	0.000099	-0.062236	-0.000364	0.004149	-0.005232
GroupId	0.002391	-0.012552	-0.000080	-0.003611	-0.000641	-0.001067	-0.003121
QuizId	0.008789	-0.000151	-0.001404	0.008861	0.010617	-0.000353	0.000603
SchemeOfWorkId	-0.002624	0.013740	-0.000669	0.056542	-0.017107	-0.009121	-0.034802
HourAnswered	-0.001600	0.005241	-0.000638	0.025151	-0.000545	-0.002872	-0.063160
MonthAnswered	0.002013	0.000408	0.000434	0.023742	0.008555	-0.000202	-0.003200
YearAnswered	-0.006134	-0.004789	-0.000842	-0.019094	-0.004690	-0.000586	0.025574
Day of Week Answered	-0.000849	0.006918	-0.000252	0.018168	-0.001983	-0.002092	-0.034558
YearBirth	-0.013685	-0.003556	-0.000123	0.029333	0.018505	0.008666	0.028388
Age	0.012042	0.002056	-0.000156	-0.036635	-0.020642	-0.009126	-0.020664

In [117…
```
# sns.heatmap(df4.corr(), cmap='Blues', annot=True)
# plt.title('Correlation Matrix');
```

In [33]:
```
#Checking correlations for ANnserValue which id the target for task 2
correlations = df4.corr().unstack().sort_values(ascending=False) # Build correlation m
correlations = pd.DataFrame(correlations).reset_index() # Convert to dataframe
correlations.columns = ['col1', 'col2', 'correlation'] # Label it
correlations.query("col1 == 'AnswerValue' & col2 != 'AnswerValue'") # Filter by variab
# output of this code will give correlation of column  with all the other columns
```

Out[33]:

	col1	col2	correlation
18	AnswerValue	CorrectAnswer	0.419281
83	AnswerValue	YearBirth	0.008666
103	AnswerValue	Gender	0.004685
107	AnswerValue	PremiumPupil	0.004149
120	AnswerValue	QuestionId	0.002032
136	AnswerValue	AnswerId	0.000152
152	AnswerValue	MonthAnswered	-0.000202
157	AnswerValue	QuizId	-0.000353
169	AnswerValue	YearAnswered	-0.000586
184	AnswerValue	UserId	-0.000947
185	AnswerValue	GroupId	-0.001067
198	AnswerValue	Day of Week Answered	-0.002092
202	AnswerValue	HourAnswered	-0.002872
213	AnswerValue	IsCorrect	-0.004482
228	AnswerValue	SchemeOfWorkId	-0.009121
230	AnswerValue	Age	-0.009126

In [34]:

```
#Checking correlations for IsCorrecte which id the target for task 2
correlations = df4.corr().unstack().sort_values(ascending=False) # Build correlation m
correlations = pd.DataFrame(correlations).reset_index() # Convert to dataframe
correlations.columns = ['col1', 'col2', 'correlation'] # Label it
correlations.query("col1 == 'IsCorrect' & col2 != 'IsCorrect'") # Filter by variable
# output of this code will give correlation of column v2 with all the other columns
```

Out[34]:

	col1	col2	correlation
29	IsCorrect	SchemeOfWorkId	0.056542
38	IsCorrect	YearBirth	0.029333
50	IsCorrect	HourAnswered	0.025151
54	IsCorrect	MonthAnswered	0.023742
62	IsCorrect	Day of Week Answered	0.018168
77	IsCorrect	QuizId	0.008861
96	IsCorrect	UserId	0.005306
109	IsCorrect	QuestionId	0.003461
155	IsCorrect	AnswerId	-0.000317
212	IsCorrect	GroupId	-0.003611
214	IsCorrect	AnswerValue	-0.004482
226	IsCorrect	CorrectAnswer	-0.008872
247	IsCorrect	YearAnswered	-0.019094
259	IsCorrect	Gender	-0.028378
267	IsCorrect	Age	-0.036635
274	IsCorrect	PremiumPupil	-0.062236

FEATURES INTERACTION (ADDITION /MULTIPLICATION/ SUBTRACTION E.T.C) FOR PREPARAING DATA FOR TASK 1

In [21]:
```
df4.head(5)
```

Out[21]:

	QuestionId	UserId	AnswerId	IsCorrect	CorrectAnswer	AnswerValue	Gender	DateOfBirth	Pre
0	16997	65967	12453206	0	4	2	1	2002-11-01	
2	15911	50013	13598796	0	3	1	1	2004-09-01	
10	21568	87935	17488010	1	4	4	1	2003-11-01	
11	15962	77530	11435172	0	2	3	1	2003-12-01	
14	20640	99529	12214186	0	3	4	2	2003-09-01	

In [22]:
```
df4.drop(['DateAnswered','Day of Week Ans','DateOfBirth'], axis = 1, inplace= True)  #
```

In [21]:
```
df4.shape
```

Out[21]:
```
(4181925, 17)
```

In [23]:
```
poly_feature_1 = ['CorrectAnswer', 'AnswerValue'] # 2 raise to power of 2 features (i
poly_feature_2 = ['HourAnswered', 'MonthAnswered','Day of Week Answered'] # 2 raise to
```

In [ ]:
```

```

In [24]:
```
from sklearn.preprocessing import PolynomialFeatures
```

In [25]:
```
poly = PolynomialFeatures(degree=2, interaction_only=True, include_bias=False)

poly1 = poly.fit_transform(df4[poly_feature_1])
poly2 = poly.fit_transform(df4[poly_feature_2])
```

In [26]:
```
df4.shape
```

Out[26]:
```
(4181925, 17)
```

In [27]:
```
#Renaming the features to poly1_.....n, poly2_....n
df_poly1 = pd.DataFrame(poly1, columns=[f"poly1_{i}" for i in range(poly1.shape[1])])
df_poly2 = pd.DataFrame(poly2, columns=[f"poly2_{i}" for i in range(poly2.shape[1])])
```

In [28]:
```
df_poly1.columns
```

Out[28]:
```
Index(['poly1_0', 'poly1_1', 'poly1_2'], dtype='object')
```

In [29]:
```
df_poly2.columns
```

Out[29]:
```
Index(['poly2_0', 'poly2_1', 'poly2_2', 'poly2_3', 'poly2_4', 'poly2_5'], dtype='obje
ct')
```

In [30]:
```
df4.columns
```

Out[30]:
```
Index(['QuestionId', 'UserId', 'AnswerId', 'IsCorrect', 'CorrectAnswer',
       'AnswerValue', 'Gender', 'PremiumPupil', 'GroupId', 'QuizId',
       'SchemeOfWorkId', 'HourAnswered', 'MonthAnswered', 'YearAnswered',
       'Day of Week Answered', 'YearBirth', 'Age'],
      dtype='object')
```

In [31]:
```
df4.shape
```

Out[31]:
```
(4181925, 17)
```

In [ ]:
```

```

In [32]:
```
# # Merging the datas created from polynomial to the whole datsets
new_data = pd.concat([df4.reset_index(drop=True), df_poly1.reset_index(drop=True)], a
new_data = pd.concat([new_data.reset_index(drop=True), df_poly2.reset_index(drop=True)
```

In [33]:
```
new_data.shape
```

Out[33]:
```
(4181925, 26)
```

In [34]:
```
new_data['Perfect_Correct'] = new_data[poly_feature_1].sum(axis = 1) # sum of all feat
new_data['Perfect_Date'] = new_data[poly_feature_2].sum(axis = 1) # sum of all feature
```

In [35]:
```
new_data['Perfect_Correct'].head()
```

Out[35]:
```
0    6
1    4
2    8
3    5
4    7
Name: Perfect_Correct, dtype: int64
```

In [36]:
```
new_data['Perfect_Date'].head()
```

Out[36]:
```
0    19
1    32
2    27
3    32
4    39
Name: Perfect_Date, dtype: int64
```

In [43]:
```
new_data.shape
```

Out[43]:
```
(4181925, 28)
```

In [44]:
```
new_data.columns
```

Out[44]:
```
Index(['QuestionId', 'UserId', 'AnswerId', 'IsCorrect', 'CorrectAnswer',
       'AnswerValue', 'Gender', 'PremiumPupil', 'GroupId', 'QuizId',
       'SchemeOfWorkId', 'HourAnswered', 'MonthAnswered', 'YearAnswered',
       'Day of Week Answered', 'YearBirth', 'Age', 'poly1_0', 'poly1_1',
       'poly1_2', 'poly2_0', 'poly2_1', 'poly2_2', 'poly2_3', 'poly2_4',
       'poly2_5', 'Perfect_Correct', 'Perfect_Date'],
      dtype='object')
```

In [37]:
```
# new_data['year_month'] =
new_data['Year_Month'] = new_data['YearAnswered'].apply(str) + "_"  + new_data['MonthA
```

In [38]:
```
new_data['Year_Month'].head()
```

Out[38]:
```
0     2019_4
1    2018_11
2     2019_5
3    2018_11
4    2018_12
Name: Year_Month, dtype: object
```

In [47]:
```
new_data.head()
```

Out[47]:

	QuestionId	UserId	AnswerId	IsCorrect	CorrectAnswer	AnswerValue	Gender	PremiumPupil	Gr
0	16997	65967	12453206	0	4	2	1	0.0	
1	15911	50013	13598796	0	3	1	1	1.0	
2	21568	87935	17488010	1	4	4	1	1.0	
3	15962	77530	11435172	0	2	3	1	1.0	
4	20640	99529	12214186	0	3	4	2	0.0	

5 rows × 29 columns

In [48]:
```
new_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4181925 entries, 0 to 4181924
Data columns (total 29 columns):
 #   Column                Dtype  
---  ------                -----  
 0   QuestionId            int64  
 1   UserId                int64  
 2   AnswerId              int64  
 3   IsCorrect             int64  
 4   CorrectAnswer         int64  
 5   AnswerValue           int64  
 6   Gender                int64  
 7   PremiumPupil          float64
 8   GroupId               int64  
 9   QuizId                int64  
 10  SchemeOfWorkId        float64
 11  HourAnswered          int64  
 12  MonthAnswered         int64  
 13  YearAnswered          int64  
 14  Day of Week Answered  int64  
 15  YearBirth             int64  
 16  Age                   int32  
 17  poly1_0               float64
 18  poly1_1               float64
 19  poly1_2               float64
 20  poly2_0               float64
 21  poly2_1               float64
 22  poly2_2               float64
 23  poly2_3               float64
 24  poly2_4               float64
 25  poly2_5               float64
 26  Perfect_Correct       int64  
 27  Perfect_Date          int64  
 28  Year_Month            object 
dtypes: float64(11), int32(1), int64(16), object(1)
memory usage: 909.3+ MB
```

In [39]:
```
# bin_label = [1,2,3,4,5]
new_data['Age_bin'] = pd.qcut(new_data.Age, q = [0, .2, .4, .6, .8, 1], labels=False)
```

In [40]:
```
new_data['Age_bin'].head()
```

Out[40]:
```
0    4
1    1
2    3
3    2
4    2
Name: Age_bin, dtype: int64
```

In [52]:
```
new_data['Age_bin'].unique()
```

Out[52]:
```
array([4, 1, 3, 2, 0], dtype=int64)
```

In [53]:
```
new_data['Age_bin'].value_counts()
```

Out[53]:
```
2    1279942
1    1197915
0     860137
3     724825
4     119106
Name: Age_bin, dtype: int64
```

In [41]:
```
new_data['Age_bin'].value_counts()# Calculating mean, std, max, min for each of the Ag

new_data['Age' + '_mean_on_yr_mth'] = new_data['Age'].groupby(new_data['Year_Month']).
new_data['Age' + '_std_on_yr_mth'] = new_data['Age'].groupby(new_data['Year_Month']).t
new_data['Age' + '_max_on_yr_mth'] = new_data['Age'].groupby(new_data['Year_Month']).t
new_data['Age' + '_min_on_yr_mth'] = new_data['Age'].groupby(new_data['Year_Month']).t
```

In [42]:
```
new_data['Age' + '_mean_on_yr_mth'].head()
```

Out[42]:
```
0    14.470042
1    13.680786
2    14.327149
3    13.680786
4    13.608579
Name: Age_mean_on_yr_mth, dtype: float64
```

In [43]:
```
new_data['Age' + '_std_on_yr_mth'].head()
```

Out[43]:
```
0    1.991865
1    1.605366
2    2.099209
3    1.605366
4    1.793765
Name: Age_std_on_yr_mth, dtype: float64
```

In [44]:
```
new_data['Year_Month'].unique() #I need to do label encoding for this
```

Out[44]:
```
array(['2019_4', '2018_11', '2019_5', '2018_12', '2019_6', '2019_10',
       '2019_1', '2020_2', '2019_3', '2019_2', '2020_1', '2019_12',
       '2019_11', '2020_3', '2020_4', '2018_9', '2018_10', '2019_8',
       '2019_9', '2019_7', '2020_5'], dtype=object)
```

In [58]:
```
type(new_data['Year_Month'])
```

Out[58]:
```
pandas.core.series.Series
```

In [45]:
```
new_data['Year_Month_int'], _ = pd.factorize(new_data['Year_Month']) # label encoding
```

In [46]:
```
new_data['Year_Month_int'].head()
```

Out[46]:
```
0    0
1    1
2    2
3    1
4    3
Name: Year_Month_int, dtype: int64
```

In [47]:
```
new_data['Year_Month_int'].unique() # new_data['year_month'] has been encoded numerica
```

Out[47]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20], dtype=int64)
```

In [48]:
```
type(new_data['Year_Month_int'])
```

Out[48]:
```
pandas.core.series.Series
```

In [49]:
```
new_data.columns
```

Out[49]:
```
Index(['QuestionId', 'UserId', 'AnswerId', 'IsCorrect', 'CorrectAnswer',
       'AnswerValue', 'Gender', 'PremiumPupil', 'GroupId', 'QuizId',
       'SchemeOfWorkId', 'HourAnswered', 'MonthAnswered', 'YearAnswered',
       'Day of Week Answered', 'YearBirth', 'Age', 'poly1_0', 'poly1_1',
       'poly1_2', 'poly2_0', 'poly2_1', 'poly2_2', 'poly2_3', 'poly2_4',
       'poly2_5', 'Perfect_Correct', 'Perfect_Date', 'Year_Month', 'Age_bin',
       'Age_mean_on_yr_mth', 'Age_std_on_yr_mth', 'Age_max_on_yr_mth',
       'Age_min_on_yr_mth', 'Year_Month_int'],
      dtype='object')
```

In [50]:
```
new_data.shape
```

Out[50]:
```
(4181925, 35)
```

In [51]:
```
new_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4181925 entries, 0 to 4181924
Data columns (total 35 columns):
 #   Column                Dtype
---  ------                -----
 0   QuestionId            int64
 1   UserId                int64
 2   AnswerId              int64
 3   IsCorrect             int64
 4   CorrectAnswer         int64
 5   AnswerValue           int64
 6   Gender                int64
 7   PremiumPupil          float64
 8   GroupId               int64
 9   QuizId                int64
 10  SchemeOfWorkId        float64
 11  HourAnswered          int64
 12  MonthAnswered         int64
 13  YearAnswered          int64
 14  Day of Week Answered  int64
 15  YearBirth             int64
 16  Age                   int32
 17  poly1_0               float64
 18  poly1_1               float64
 19  poly1_2               float64
 20  poly2_0               float64
 21  poly2_1               float64
 22  poly2_2               float64
 23  poly2_3               float64
 24  poly2_4               float64
 25  poly2_5               float64
 26  Perfect_Correct       int64
 27  Perfect_Date          int64
 28  Year_Month            object
 29  Age_bin               int64
 30  Age_mean_on_yr_mth    float64
 31  Age_std_on_yr_mth     float64
 32  Age_max_on_yr_mth     int32
 33  Age_min_on_yr_mth     int32
 34  Year_Month_int        int64
dtypes: float64(13), int32(3), int64(18), object(1)
memory usage: 1.0+ GB
```

In [52]:
```
new_data.head()
```

Out[52]:

	QuestionId	UserId	AnswerId	IsCorrect	CorrectAnswer	AnswerValue	Gender	PremiumPupil	Gr
0	16997	65967	12453206	0	4	2	1	0.0	
1	15911	50013	13598796	0	3	1	1	1.0	
2	21568	87935	17488010	1	4	4	1	1.0	
3	15962	77530	11435172	0	2	3	1	1.0	
4	20640	99529	12214186	0	3	4	2	0.0	

5 rows × 35 columns

In [53]:
```
new_data.drop('Year_Month', axis = 1, inplace = True) # we need to drop this cos we al
```

Scaling the ID groups using Log

In [54]:
```
new_data['l_QuestionId']= np.log(new_data['QuestionId'])
new_data['l_AnswerId']= np.log(new_data['AnswerId'])
new_data['l_GroupId']= np.log(new_data['GroupId'])
new_data['l_QiuzId']= np.log(new_data['QuizId'])
new_data['l_SchemeOfWorkId']= np.log(new_data['SchemeOfWorkId'])
```

In [55]:
```
new_data.head()
```

Out[55]:

	QuestionId	UserId	AnswerId	IsCorrect	CorrectAnswer	AnswerValue	Gender	PremiumPupil	Gr
0	16997	65967	12453206	0	4	2	1	0.0	
1	15911	50013	13598796	0	3	1	1	1.0	
2	21568	87935	17488010	1	4	4	1	1.0	
3	15962	77530	11435172	0	2	3	1	1.0	
4	20640	99529	12214186	0	3	4	2	0.0	

5 rows × 39 columns

In [56]:
```
new_data.drop(['QuestionId','AnswerId','GroupId','QuizId','SchemeOfWorkId'], axis = 1,
```

In [71]:
```
new_data.shape
```

Out[71]:
```
(4181925, 34)
```

In [72]:
```
new_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4181925 entries, 0 to 4181924
Data columns (total 34 columns):
 #   Column                Dtype  
---  ------                -----  
 0   UserId                int64  
 1   IsCorrect             int64  
 2   CorrectAnswer         int64  
 3   AnswerValue           int64  
 4   Gender                int64  
 5   PremiumPupil          float64
 6   HourAnswered          int64  
 7   MonthAnswered         int64  
 8   YearAnswered          int64  
 9   Day of Week Answered  int64  
 10  YearBirth             int64  
 11  Age                   int32  
 12  poly1_0               float64
 13  poly1_1               float64
 14  poly1_2               float64
 15  poly2_0               float64
 16  poly2_1               float64
 17  poly2_2               float64
 18  poly2_3               float64
 19  poly2_4               float64
 20  poly2_5               float64
 21  Perfect_Correct       int64  
 22  Perfect_Date          int64  
 23  Age_bin               int64  
 24  Age_mean_on_yr_mth    float64
 25  Age_std_on_yr_mth     float64
 26  Age_max_on_yr_mth     int32  
 27  Age_min_on_yr_mth     int32  
 28  Year_Month_int        int64  
 29  l_QuestionId          float64
 30  l_AnswerId            float64
 31  l_GroupId             float64
 32  l_QiuzId              float64
 33  l_SchemeOfWorkId      float64
dtypes: float64(17), int32(3), int64(14)
memory usage: 1.0 GB
```

REMOVING OUTLIERS

Since we ve created new features, then it becomes necessary to check and remove potetial outliers likely to have generated. The outliers of the numeric column should be treated

In [57]:
```
def remove_outliers(data):
    for col in data:
        high = data[col].mean() + 3*data[col].std()
        low = data[col].mean() - 3*data[col].std()
        data[col] = np.where(
            data[col] > high,
            high,
            np.where(
                data[col] < low,
                low,
                data[col]))
```

```
remove_outliers(new_data)
```

In [59]:
```
new_data.shape
```

Out[59]:
```
(4181925, 34)
```

Saving the Preprocessed dataframe into a csv file

In [61]:
```
new_data.to_csv('processed_data.csv') #saving the processed data for task 1 modelling
```

FEATURES INTERACTION (ADDITION /MULTIPLICATION/ SUBTRACTION E.T.C) FOR PREPARAING DATA FOR TASK 2

Then we need few adjustments to preprocess task 2 data, we will use the preproceed_data.csv of task 1 to continue the preparation for task 2 data preprocessing

In [2]:
```
#Load the preprocessed data for task 2 data preprocessing
df_2 =pd.read_csv('processed_data.csv')
```

In [3]:
```
df_2.shape
```

Out[3]:
```
(4181925, 35)
```

In [4]:
```
df_2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4181925 entries, 0 to 4181924
Data columns (total 35 columns):
 #   Column                Dtype
---  ------                -----
 0   Unnamed: 0            int64
 1   UserId                float64
 2   IsCorrect             float64
 3   CorrectAnswer         float64
 4   AnswerValue           float64
 5   Gender                float64
 6   PremiumPupil          float64
 7   HourAnswered          float64
 8   MonthAnswered         float64
 9   YearAnswered          float64
 10  Day of Week Answered  float64
 11  YearBirth             float64
 12  Age                   float64
 13  poly1_0               float64
 14  poly1_1               float64
 15  poly1_2               float64
 16  poly2_0               float64
 17  poly2_1               float64
 18  poly2_2               float64
 19  poly2_3               float64
 20  poly2_4               float64
 21  poly2_5               float64
 22  Perfect_Correct       float64
 23  Perfect_Date          float64
 24  Age_bin               float64
 25  Age_mean_on_yr_mth    float64
 26  Age_std_on_yr_mth     float64
 27  Age_max_on_yr_mth     float64
 28  Age_min_on_yr_mth     float64
 29  Year_Month_int        float64
 30  l_QuestionId          float64
 31  l_AnswerId            float64
 32  l_GroupId             float64
 33  l_QiuzId              float64
 34  l_SchemeOfWorkId      float64
dtypes: float64(34), int64(1)
memory usage: 1.1 GB
```

In [5]:
```
# Remove the poly feature 1 since that what we will be predicting in task 2 and Perfec
df_2.drop(['Unnamed: 0','poly1_0','poly1_1','poly1_2','Perfect_Correct'], axis = 1, ir
```

In [6]:
```
df_2.head()
```

Out[6]:

	UserId	IsCorrect	CorrectAnswer	AnswerValue	Gender	PremiumPupil	HourAnswered	MonthAr
0	65967.0	0.0	4.0	2.0	1.0	0.0	13.0	
1	50013.0	0.0	3.0	1.0	1.0	1.0	18.0	
2	87935.0	1.0	4.0	4.0	1.0	1.0	22.0	
3	77530.0	0.0	2.0	3.0	1.0	1.0	21.0	
4	99529.0	0.0	3.0	4.0	2.0	0.0	21.0	

5 rows × 30 columns

In [7]:
```
df_2.shape
```

Out[7]:
```
(4181925, 30)
```

In [8]:
```
poly_feature_1 = ['IsCorrect','CorrectAnswer'] # 2 raise to power of 2 features
```

In [9]:
```
from sklearn.preprocessing import PolynomialFeatures
poly = PolynomialFeatures(degree=2, interaction_only=True, include_bias=False)

poly1 = poly.fit_transform(df_2[poly_feature_1])
```

In [10]:
```
#Renaming the features to poly1_.....n
df_poly1 = pd.DataFrame(poly1, columns=[f"poly1_{i}" for i in range(poly1.shape[1])])
```

In [11]:
```
df_poly1.columns
```

Out[11]:
```
Index(['poly1_0', 'poly1_1', 'poly1_2'], dtype='object')
```

In [12]:
```
# # Merging the datas created from polynomial to the whole datsets
new_data_2 = pd.concat([df_2.reset_index(drop=True), df_poly1.reset_index(drop=True)],
```

In [13]:
```
new_data_2.shape
```

Out[13]:
```
(4181925, 33)
```

In [14]:
```
new_data_2['Perfect_Correct'] = new_data_2[poly_feature_1].sum(axis = 1) # sum of all
```

In [15]:
```
new_data_2.shape
```

Out[15]:
```
(4181925, 34)
```

Saving the Preprocessed dataframe into a csv file

In [16]:
```
new_data_2.to_csv('processed_data_2.csv') #saved withouth encoding
```

In [ ]:
```

```

Using Sweetviz visualization to check my data behaviour for both IsCorrect target and AnswerValue

In [ ]:
```
new_report = sv.analyze([new_data, 'Train'], target_feat='IsCorrect')
```

In [ ]:
```
new_report_1.show_html('New_Report_1.html')
```

In [ ]:
```
feat_config = sv.FeatureConfig(skip = 'UserId',force_num=['AnswerValue'])
new_report_2 = sv.analyze([new_data, 'Train'], None, feat_config)
```

In [ ]:
```
new_report_2.show_html('New_Report_2.html')
```

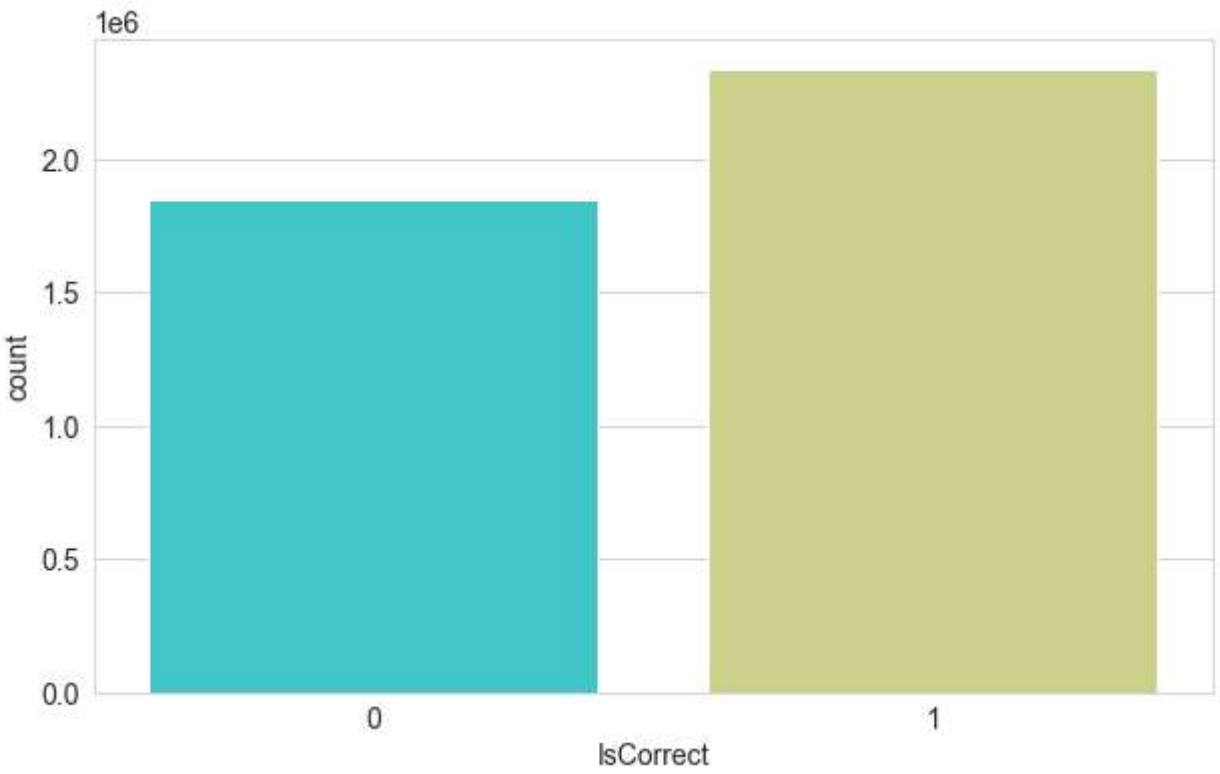
FIRST TASKS PREDICTING IsCorrect

In [79]:
```
new_data['IsCorrect'].value_counts()
```

Out[79]:
```
1    2334670
0    1847255
Name: IsCorrect, dtype: int64
```

In [80]:
```
sns.set_style('whitegrid')
sns.countplot(x='IsCorrect',data=new_data,palette='rainbow')
```

Out[80]:
```
<AxesSubplot:xlabel='IsCorrect', ylabel='count'>
```



I dont need to do class sampling e.g using SMOTE to undersample or oversample

In [73]:
```
correlations = new_data.corr().unstack().sort_values(ascending=False) # Build correlat
correlations = pd.DataFrame(correlations).reset_index() # Convert to dataframe
correlations.columns = ['col1', 'col2', 'correlation'] # Label it
```

```
correlations.query("col1 == 'IsCorrect' & col2 != 'IsCorrect'") # Filter by variable
# output of this code will give correlation of column v2 with all the other columns
```

Out[73]:

	col1	col2	correlation
164	IsCorrect	poly1_2	0.157131
183	IsCorrect	l_SchemeOfWorkId	0.060895
216	IsCorrect	Perfect_Date	0.037700
225	IsCorrect	poly2_3	0.030804
229	IsCorrect	poly2_5	0.029362
231	IsCorrect	YearBirth	0.029333
254	IsCorrect	poly2_0	0.025151
256	IsCorrect	HourAnswered	0.025151
260	IsCorrect	poly2_1	0.023742
261	IsCorrect	MonthAnswered	0.023742
275	IsCorrect	poly2_4	0.019568
287	IsCorrect	poly2_2	0.018168
289	IsCorrect	Day of Week Answered	0.018168
298	IsCorrect	Age_max_on_yr_mth	0.015451
323	IsCorrect	Year_Month_int	0.011142
412	IsCorrect	l_QiuzId	0.005772
420	IsCorrect	UserId	0.005306
441	IsCorrect	l_QuestionId	0.004645
596	IsCorrect	l_AnswerId	-0.000223
819	IsCorrect	poly1_1	-0.004482
821	IsCorrect	AnswerValue	-0.004482
842	IsCorrect	l_GroupId	-0.007007
864	IsCorrect	Perfect_Correct	-0.007944
879	IsCorrect	poly1_0	-0.008872
881	IsCorrect	CorrectAnswer	-0.008872
906	IsCorrect	Age_min_on_yr_mth	-0.012071
960	IsCorrect	YearAnswered	-0.019094
985	IsCorrect	Age_std_on_yr_mth	-0.021922
991	IsCorrect	Age_bin	-0.023552
993	IsCorrect	Age_mean_on_yr_mth	-0.023709
1004	IsCorrect	Gender	-0.028378
1028	IsCorrect	Age	-0.036635
1048	IsCorrect	PremiumPupil	-0.062236

Splitting Datasets And Performing Standard Scaler

In [74]:
```
X = new_data.drop('UserId',axis=1)
y = new_data['IsCorrect']
```

In [76]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
```

In [77]:
```
x_scaled= scaler.fit_transform(X)
```

In [ ]:
```

```

In [81]:
```
from sklearn.model_selection import train_test_split,cross_val_score
X_train, X_test, y_train, y_test = train_test_split(x_scaled, y, test_size=0.45)
```

In [ ]:
```

```

MODELLING

In [90]:
```
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import log_loss, f1_score
from sklearn.metrics import classification_report,confusion_matrix, accuracy_score
from sklearn.svm import SVC
from sklearn import svm
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import AdaBoostClassifier
# from sklearn.grid_search import GridSearchCV

# NB_mod = GaussianNB() #for continuous or numerical features.
# NB_mod = BernoulliNB() #for features with binary values
# dt= DecisionTreeClassifier()
# logreg = LogisticRegression()
```

1. LINEAR SVM

In [88]:
```
svm_lin_mod = svm.LinearSVC(class_weight = {0:0.70, 1:0.30}, C = 10)
svm_lin_mod.fit(X_train, y_train)
```

```
C:\Users\OLUBAYODE\Anaconda3_\lib\site-packages\sklearn\svm\_base.py:1206: Convergenc
eWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
```

Out[88]:
```
LinearSVC(C=10, class_weight={0: 0.7, 1: 0.3})
```

In [89]:
```
y_pred_lin = svm_lin_mod.predict(X_test)
```

In [91]:
```
print(accuracy_score(y_test, y_pred_lin))
print('Total accuracy: %.5f' % accuracy_score(y_test, y_pred_lin))
```

```
print(classification_report(y_test, y_pred_lin))
print(confusion_matrix(y_test, y_pred_lin))
```

```
1.0
Total accuracy: 1.00000
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    831488
           1       1.00      1.00      1.00   1050379

    accuracy                           1.00   1881867
   macro avg       1.00      1.00      1.00   1881867
weighted avg       1.00      1.00      1.00   1881867

[[ 831488       0]
 [      0 1050379]]
```

In [92]:
```
#We would define a function to print out the confusion matrix and other metric with th

def print_metrics(labels, predictions):
    metrics = sklm.precision_recall_fscore_support(labels, predictions)
    conf = sklm.confusion_matrix(labels, predictions)
    print('                 Confusion matrix')
    print('                 Actual positive    Actual negative')
    print('Score positive    %6d' % conf[0,0] + '             %5d' % conf[0,1])
    print('Score negative    %6d' % conf[1,0] + '             %5d' % conf[1,1])
    print('')
    print('Accuracy  %0.2f' % sklm.accuracy_score(labels, predictions))
    print(' ')
    print('           Positive      Negative')
    print('Num case   %6d' % metrics[3][0] + '        %6d' % metrics[3][1])
    print('Precision  %6.2f' % metrics[0][0] + '        %6.2f' % metrics[0][1])
    print('Recall     %6.2f' % metrics[1][0] + '        %6.2f' % metrics[1][1])
    print('F1         %6.2f' % metrics[2][0] + '        %6.2f' % metrics[2][1])



print_metrics(y_test, y_pred_lin)
```

```
                 Confusion matrix
                 Actual positive    Actual negative
Score positive    831488                0
Score negative         0             1050379

Accuracy  1.00

           Positive      Negative
Num case   831488        1050379
Precision    1.00          1.00
Recall       1.00          1.00
F1           1.00          1.00
```

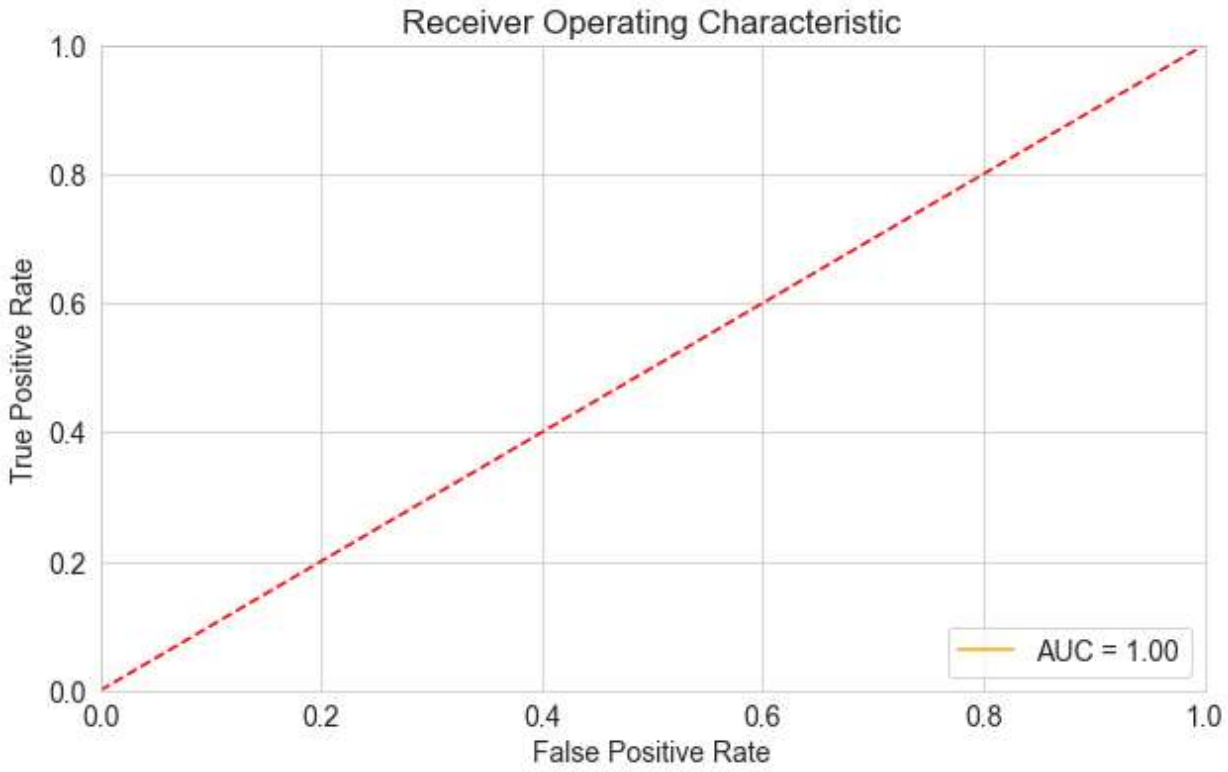
In [95]:
```
prob_lin = svm_lin_mod._predict_proba_lr(X_test)
print(prob_lin[:5,:])
```

```
[[0.73105857 0.26894143]
 [0.73105858 0.26894142]
 [0.26894143 0.73105857]
 [0.26894144 0.73105856]
 [0.26894144 0.73105856]]
```

In [96]:
```
def plot_auc(labels, probs):
    ## Compute the false positive rate, true positive rate
    ## and threshold along with the AUC
    fpr, tpr, threshold = sklm.roc_curve(labels, probs[:,1])
    auc = sklm.auc(fpr, tpr)

    ## Plot the result
    plt.title('Receiver Operating Characteristic')
    plt.plot(fpr, tpr, color = 'orange', label = 'AUC = %0.2f' % auc)
    plt.legend(loc = 'lower right')
    plt.plot([0, 1], [0, 1],'r--')
    plt.xlim([0, 1])
    plt.ylim([0, 1])
    plt.ylabel('True Positive Rate')
    plt.xlabel('False Positive Rate')
    plt.show()

plot_auc(y_test, prob_lin)
```



Our result seems to be Overfitted

2. RBF KERNEL SVM

In [ ]:
```
nr.seed(3456)
## Define the dictionary for the grid search and the model object to search on
param_grid = {"C": [0.1, 1], "gamma":[1.0/50.0, 1.0/200.0]}
## Define the SVM model
svc_clf = svm.SVC(class_weight = {0:0.70, 1:0.30})

## Perform the grid search over the parameters
clf = ms.GridSearchCV(estimator = svc_clf, param_grid = param_grid,
                      scoring = 'roc_auc',
                      return_train_score = True)
```

```
clf.fit(x_scaled, y)
print(clf.best_estimator_.C)
print(clf.best_estimator_.gamma)
```

In [ ]:
```
nr.seed(498)
cv_estimate = ms.cross_val_score(clf, x_scaled, y)

print('Mean performance metric = %4.3f' % np.mean(cv_estimate))
print('SDT of the metric       = %4.3f' % np.std(cv_estimate))
print('Outcomes by cv fold')
for i, x in enumerate(cv_estimate):
    print('Fold %2d    %4.3f' % (i+1, x))
```

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(x_scaled, y, test_size=0.45)
```

In [ ]:
```
nr.seed(1115)
svm_mod = svm.SVC(C = clf.best_estimator_.C,
                  gamma = clf.best_estimator_.gamma,
                  class_weight = {0:0.70, 1:0.30},
                  probability=True)
svm_mod.fit(X_train, y_train)
```