Project Report IDS

Team members:

Priyal Bhagwanani (16UCC071)
Jatin Keswani (16UCS081)
Preksha Pandey (16UCS139)
Sakshi Srivastava (16UCS164)

Problem statement:

Given some data about trainees who took an exam, predict whether the trainee will pass or fail.

Data source:

Kaggle

Goal:

The objective and the goal of our project is to explore the training data and get inferences from it and hence apply ML classification algorithms which finds an accurate answer for the testing dataset.

Preprocessing:

The dataset consists of training of data. Now we read the file and check the shape of the data. We also saw the column features.

For Train data:

```
Training Dataset Shape:
(73147, 16)
```

```
Training Dataset Columns/Features:
id                            object
program_id                    object
program_type                  object
program_duration               int64
test_id                        int64
test_type                     object
difficulty_level              object
trainee_id                     int64
gender                        object
education                     object
city_tier                      int64
age                          float64
total_programs_enrolled        int64
is_handicapped                object
trainee_engagement_rating    float64
is_pass                        int64
dtype: object
```

So we have 73147 rows in the train set. We also have 16 columns in total including the is_pass column.

Now we split the given dataset into 2 parts in ratio 7:3 where the 70% data becomes the training data and the rest 30% becomes the test data.

Now we need to predict the ‘is_pass’ column for the test data. So we divide it as follows

In [4]:
```
Y_Data = DataSet['is_pass']
X_Data = DataSet

TrainData, TestData, Y_Train, Y_Test_actual = train_test_split(X_Data , Y_Data, test_size=0.3, random_state = 0)
```

Now we check missing data percentage in train and test data.

In train data:

In [5]:
```
# checking missing data percentage in train data
total = TrainData.isnull().sum().sort_values(ascending = False)
percent = (TrainData.isnull().sum()/TrainData.isnull().count()*100).sort_values(ascending = False)
missing_TrainData  = pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])
missing_TrainData.head(30)
```

Out[5]:

	Total	Percent
age	19450	37.986797
trainee_engagement_rating	54	0.105465
is_pass	0	0.000000
is_handicapped	0	0.000000
total_programs_enrolled	0	0.000000
city_tier	0	0.000000
education	0	0.000000
gender	0	0.000000
trainee_id	0	0.000000
difficulty_level	0	0.000000
test_type	0	0.000000
test_id	0	0.000000
program_duration	0	0.000000
program_type	0	0.000000
program_id	0	0.000000
id	0	0.000000

In test data:

In [6]:
```
# checking missing data percentage in test data
total = TestData.isnull().sum().sort_values(ascending = False)
percent = (TestData.isnull().sum()/TestData.isnull().count()*100).sort_values(ascending = False)
missing_TestData  = pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])
missing_TestData.head(30)
```

Out[6]:

	Total	Percent
age	8279	37.726134
trainee_engagement_rating	23	0.104807
is_pass	0	0.000000
is_handicapped	0	0.000000
total_programs_enrolled	0	0.000000
city_tier	0	0.000000
education	0	0.000000
gender	0	0.000000
trainee_id	0	0.000000
difficulty_level	0	0.000000
test_type	0	0.000000
test_id	0	0.000000
program_duration	0	0.000000
program_type	0	0.000000
program_id	0	0.000000
id	0	0.000000

Observations:

1. The missing data in the train data is ‘age’ and ‘trainee_engagement_rating’ and the percent missing are 37.98% and 0.105% respectively.
2. The missing data in the test data is ‘age’ and ‘trainee_engagement_rating’ and the percent missing are 37.72% and 0.104% respectively.

3. We observe that the percent of missing data is almost same in both the test and train dataset.

Visualization:

Now we create a heat map for the visualisation of the correlation between the various columns of the training and testing dataset.

For train data:

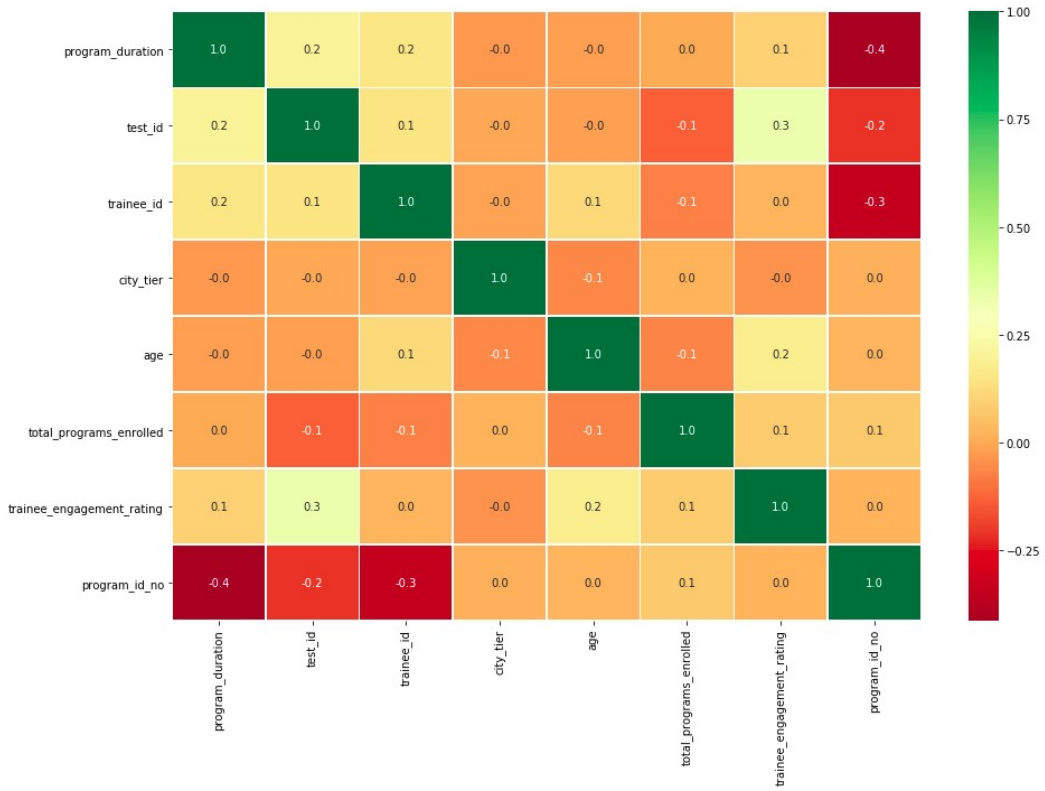
```
In [11]:
1 # TRAIN DATA HeatMap
2 f,ax = plt.subplots(figsize=(15, 10))
3 sns.heatmap(TrainData.corr(), annot=True, linewidths=.5, fmt= '.1f',ax=ax)
```

```
Out[11]: <matplotlib.axes._subplots.AxesSubplot at 0x7fcf6441c160>
```

For test data:

```
In [12]:
# TEST DATA HeatMap
f,ax = plt.subplots(figsize=(15, 10))
sns.heatmap(TestData.corr(), annot=True, linewidths=.5, fmt= '.1f',ax=ax, cmap="RdYlGn")
```

```
Out[12]: <matplotlib.axes._subplots.AxesSubplot at 0x7fcf64002da0>
```



We know that the pair of features that have their correlation factor as 1, (or approximately 1) and have some useful correlation between them, they are said to be highly correlated. Thus in such cases, we either drop the feature or we merge both the features and create a new feature which will retain the usefulness of both the features. This is done because of the following two reasons:

1. It will help in reducing the preprocessing time
2. It will help to produce more accurate results

We have used 3 helper functions for plotting univariate graphs, bivariate graphs and for finding unique values in both the dataset.

1. plot_bar_counts_categorical - for plotting univariate graphs

In [14]:
```
# This function returns the count plot of a column with percentage of each class
def plot_bar_counts_categorical(data_se, title, figsize, sort_by_counts=False):
    info = data_se.value_counts()
    info_norm = data_se.value_counts(normalize=True)
    categories = info.index.values
    counts = info.values
    counts_norm = info_norm.values
    fig, ax = plt.subplots(figsize=figsize)
    if data_se.dtype in ['object']:
        if sort_by_counts == False:
            inds = categories.argsort()
            counts = counts[inds]
            counts_norm = counts_norm[inds]
            categories = categories[inds]
        ax = sns.barplot(counts, categories, orient = "h", ax=ax)
        ax.set(xlabel="count", ylabel=data_se.name)
        ax.set_title("Distribution of " + title)
        for n, da in enumerate(counts):
            ax.text(da, n, str(da)+ ",  " + str(round(counts_norm[n]*100,2)) + " %", fontsize=10, va='center')
    else:
        inds = categories.argsort()
        counts_sorted = counts[inds]
        counts_norm_sorted = counts_norm[inds]
        ax = sns.barplot(categories, counts, orient = "v", ax=ax)
        ax.set(xlabel=data_se.name, ylabel='count')
        ax.set_title("Distribution of " + title)
        for n, da in enumerate(counts_sorted):
            ax.text(n, da, str(da)+ ",  " + str(round(counts_norm_sorted[n]*100,2)) + " %", fontsize=10, ha='cent
```

2. count_plot_by_hue - for plotting bivariate graphs

In [13]:
```
def count_plot_by_hue(data_se, hue_se, title, figsize, sort_by_counts=False):
    if sort_by_counts == False:
        order = data_se.unique()
        order.sort()
    else:
        order = data_se.value_counts().index.values
    off_hue = hue_se.nunique()
    off = len(order)
    fig, ax = plt.subplots(figsize=figsize)
    ax = sns.countplot(y=data_se, hue=hue_se, order=order, ax=ax)
    ax.set_title(title)
    patches = ax.patches
    for i, p in enumerate(ax.patches):
        x=p.get_bbox().get_points()[1,0]
        y=p.get_bbox().get_points()[:,1]
        total = x
        p = i
        q = i
        while(q < (off_hue*off)):
            p = p - off
            if p >= 0:
                total = total + (patches[p].get_bbox().get_points()[1,0] if not np.isnan(patches[p].get_bbox().ge
            else:
                q = q + off
                if q < (off*off_hue):
                    total = total + (patches[q].get_bbox().get_points()[1,0] if not np.isnan(patches[q].get_bbox(

        perc = str(round(100*(x/total), 2)) + " %"

        if not np.isnan(x):
            ax.text(x, y.mean(), str(int(x)) + ",  " + perc, va='center')
    plt.show()
```

3. train_test_data_check - for finding unique values

In [15]:

```
# This function return the multivariable analysis of two dataframe in term of unique column values
def train_test_data_check(train_df, test_df, cols=None, use_all_cols=True):
    if cols == None:
        if use_all_cols:
            train_cols = set(train_df.columns)
            test_cols = set(test_df.columns)
            cols = train_cols.intersection(test_cols)
        else:
            train_cols = set(train_df.select_dtypes(['object', 'category']).columns)
            test_cols = set(test_df.select_dtypes(['object', 'category']).columns)
            cols = train_cols.intersection(test_cols)

    for i, col in enumerate(cols):
        display(HTML('<h3><font id="'+ col + '-ttdc' + '" color="blue">' + str(i+1) + ') ' + col + '</font></h3>'
        print("Datatype : " + str(train_df[col].dtype) )
        print(str(train_df[col].dropna().nunique()) + " unique " + col  + " in Train dataset")
        print(str(test_df[col].dropna().nunique()) + " unique " + col  + " in Test dataset")
        extra = len(set(test_df[col].dropna().unique()) - set(train_df[col].dropna().unique()))
        print(str(extra) + " extra " + col + " in Test dataset")
        if extra == 0:
            display(HTML('<h5><font color="green"> All values present in Test dataset also present in Train datas
        else:
            display(HTML('<h5><font color="green">' + str(extra) + ' ' +  col + ' are not present in Train datase
```

Now we will see whether all the values present in the test data are there in train data using the train_test_data_check()

In [13]:

```
train_test_data_check(TrainData, TestData)
```

1) test_type

```
Datatype : object
2 unique test_type in Train dataset
2 unique test_type in Test dataset
0 extra test_type in Test dataset
```

All values present in Test dataset also present in Train dataset for column test_type

2) trainee_id

```
Datatype : int64
17206 unique trainee_id in Train dataset
12499 unique trainee_id in Test dataset
1294 extra trainee_id in Test dataset
```

1294 trainee_id are not present in Train dataset which are in Test dataset

3) id

```
Datatype : object
51202 unique id in Train dataset
21945 unique id in Test dataset
21945 extra id in Test dataset
```

21945 id are not present in Train dataset which are in Test dataset

4) gender

```
Datatype : object
2 unique gender in Train dataset
2 unique gender in Test dataset
0 extra gender in Test dataset
```

All values present in Test dataset also present in Train dataset for column gender

5) city_tier

```
Datatype : int64
4 unique city_tier in Train dataset
4 unique city_tier in Test dataset
0 extra city_tier in Test dataset
```

All values present in Test dataset also present in Train dataset for column city_tier

6) total_programs_enrolled

```
Datatype : int64
13 unique total_programs_enrolled in Train dataset
13 unique total_programs_enrolled in Test dataset
0 extra total_programs_enrolled in Test dataset
```

All values present in Test dataset also present in Train dataset for column total_programs_enrolled

7) is_pass

```
Datatype : int64
2 unique is_pass in Train dataset
2 unique is_pass in Test dataset
0 extra is_pass in Test dataset
```

All values present in Test dataset also present in Train dataset for column is_pass

8) program_duration

```
Datatype : int64
10 unique program_duration in Train dataset
10 unique program_duration in Test dataset
0 extra program_duration in Test dataset
```

All values present in Test dataset also present in Train dataset for column program_duration

9) test_id

```
Datatype : int64
188 unique test_id in Train dataset
188 unique test_id in Test dataset
0 extra test_id in Test dataset
```

All values present in Test dataset also present in Train dataset for column test_id

10) difficulty_level

```
Datatype : object
4 unique difficulty_level in Train dataset
4 unique difficulty_level in Test dataset
0 extra difficulty_level in Test dataset
```

All values present in Test dataset also present in Train dataset for column difficulty_level

11) id

```
Datatype : object
51202 unique id in Train dataset
21945 unique id in Test dataset
21945 extra id in Test dataset
```

21945 id are not present in Train dataset which are in Test dataset

12) education

```
Datatype : object
5 unique education in Train dataset
5 unique education in Test dataset
0 extra education in Test dataset
```

All values present in Test dataset also present in Train dataset for column education

13) is_pass

```
Datatype : int64
2 unique is_pass in Train dataset
2 unique is_pass in Test dataset
0 extra is_pass in Test dataset
```

All values present in Test dataset also present in Train dataset for column is_pass

14) gender

```
Datatype : object
2 unique gender in Train dataset
2 unique gender in Test dataset
0 extra gender in Test dataset
```

All values present in Test dataset also present in Train dataset for column gender

15) city_tier

```
Datatype : int64
4 unique city_tier in Train dataset
4 unique city_tier in Test dataset
0 extra city_tier in Test dataset
```

All values present in Test dataset also present in Train dataset for column city_tier

16) education

```
Datatype : object
5 unique education in Train dataset
5 unique education in Test dataset
0 extra education in Test dataset
```

All values present in Test dataset also present in Train dataset for column education

Observation:

1. For ‘id’
    a. Datatype: object
    b. 51202 unique id in Train dataset
    c. 21945 unique id in the Test dataset
    d. 21945 extra id in the Test dataset

Hence we observe that 21945 id are not present in Train dataset which are in Test dataset

2. For ‘trainee_id’
    a. Datatype: int64
    b. 17206 unique trainee_id in Train dataset
    c. 12499 unique trainee_id in Test dataset
    d. 1294 extra trainee_id in Test dataset

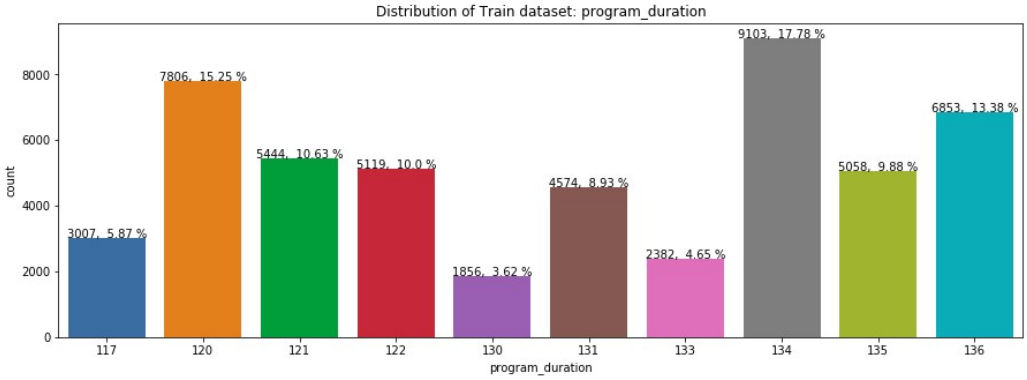
Hence we observe that 1294 trainee_id are not present in Train dataset which are in Test dataset

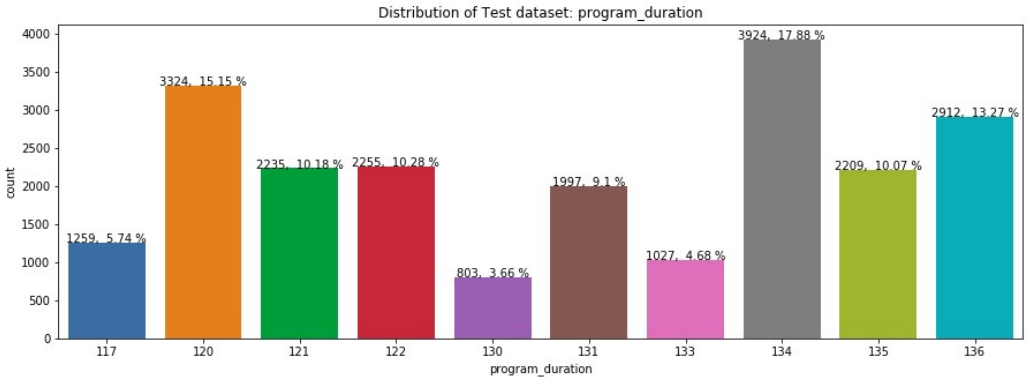
3. For the rest of the columns, all values present in Test dataset are also present in Train dataset

Univariate Analysis:

1.

```
In [14]:
1 plot_bar_counts_categorical(TrainData['program_duration'], 'Train dataset: program_duration', (15,5))
2 plot_bar_counts_categorical(TestData['program_duration'], 'Test dataset: program_duration', (15,5))
```

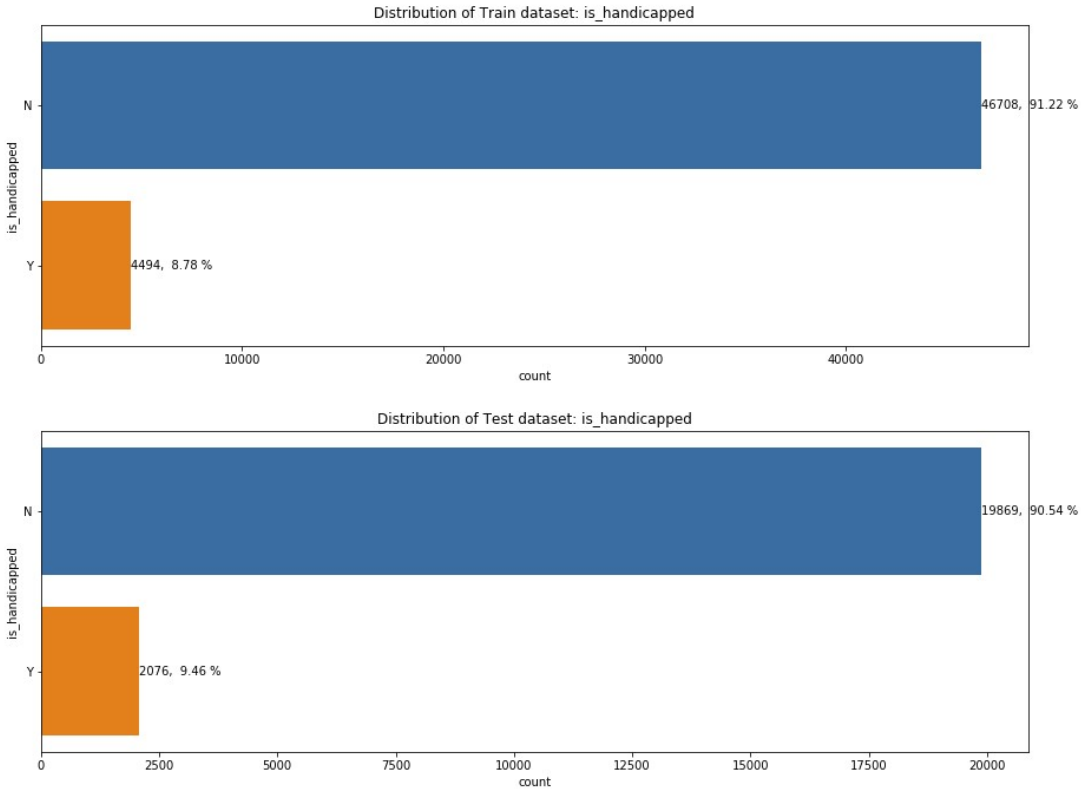




Comparing the distribution of the train and test dataset on the basis of ‘program_duration’
Result: Almost the same distribution

2.

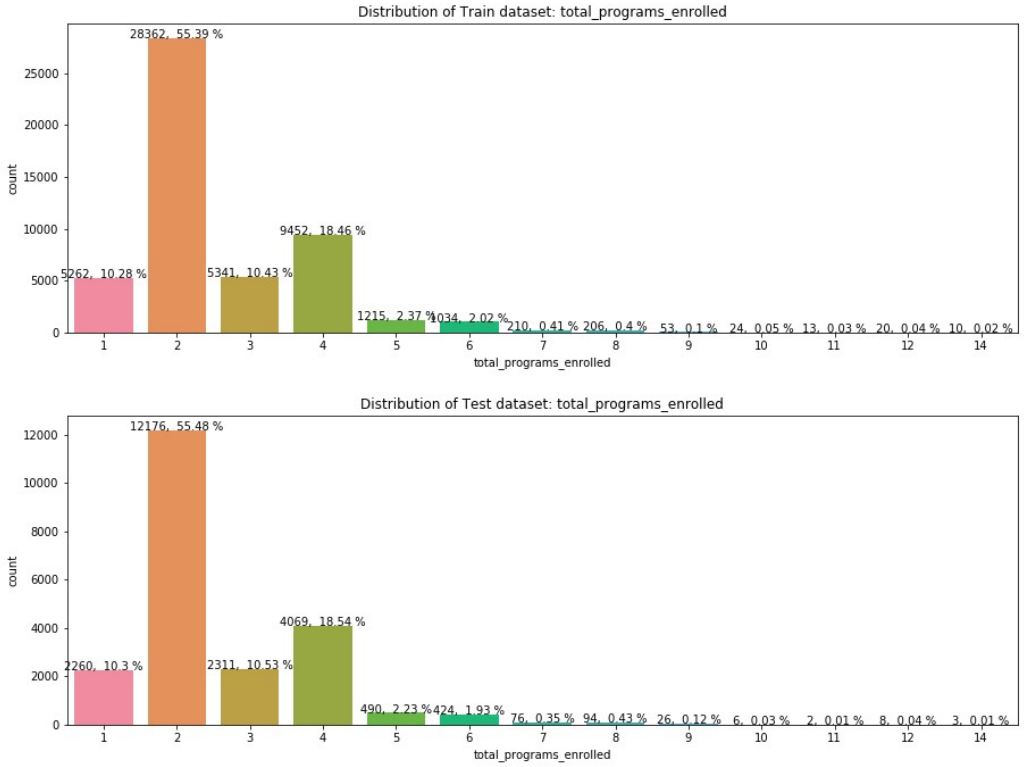
```
In [17]:  1 plot_bar_counts_categorical(TrainData['is_handicapped'], 'Train dataset: is_handicapped', (15,5))
          2 plot_bar_counts_categorical(TestData['is_handicapped'], 'Test dataset: is_handicapped', (15,5))
```



Comparing the distribution of the train and test dataset on the basis of ‘is_handicapped’
Result: Almost the same distribution

3.

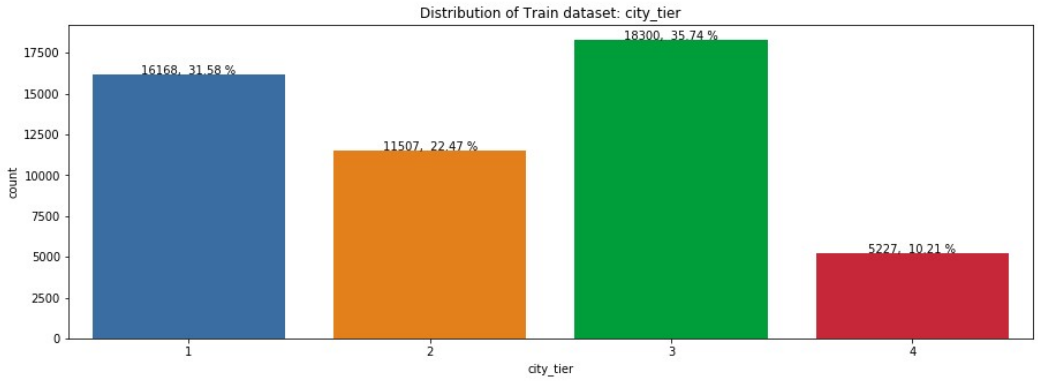
```
In [18]:
1 plot_bar_counts_categorical(TrainData['total_programs_enrolled'], 'Train dataset: total_programs_enrolled', (15,5
2 plot_bar_counts_categorical(TestData['total_programs_enrolled'], 'Test dataset: total_programs_enrolled', (15,5))
```

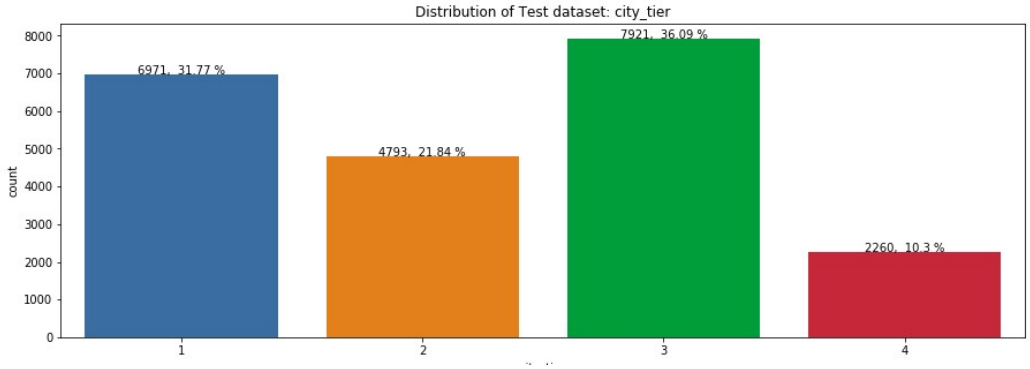


Comparing the distribution of the train and test dataset on the basis of ‘total_programs_enrolled’
Result: Almost the same distribution

4.

```
In [19]:
1 plot_bar_counts_categorical(TrainData['city_tier'], 'Train dataset: city_tier', (15,5))
2 plot_bar_counts_categorical(TestData['city_tier'], 'Test dataset: city_tier', (15,5))
```

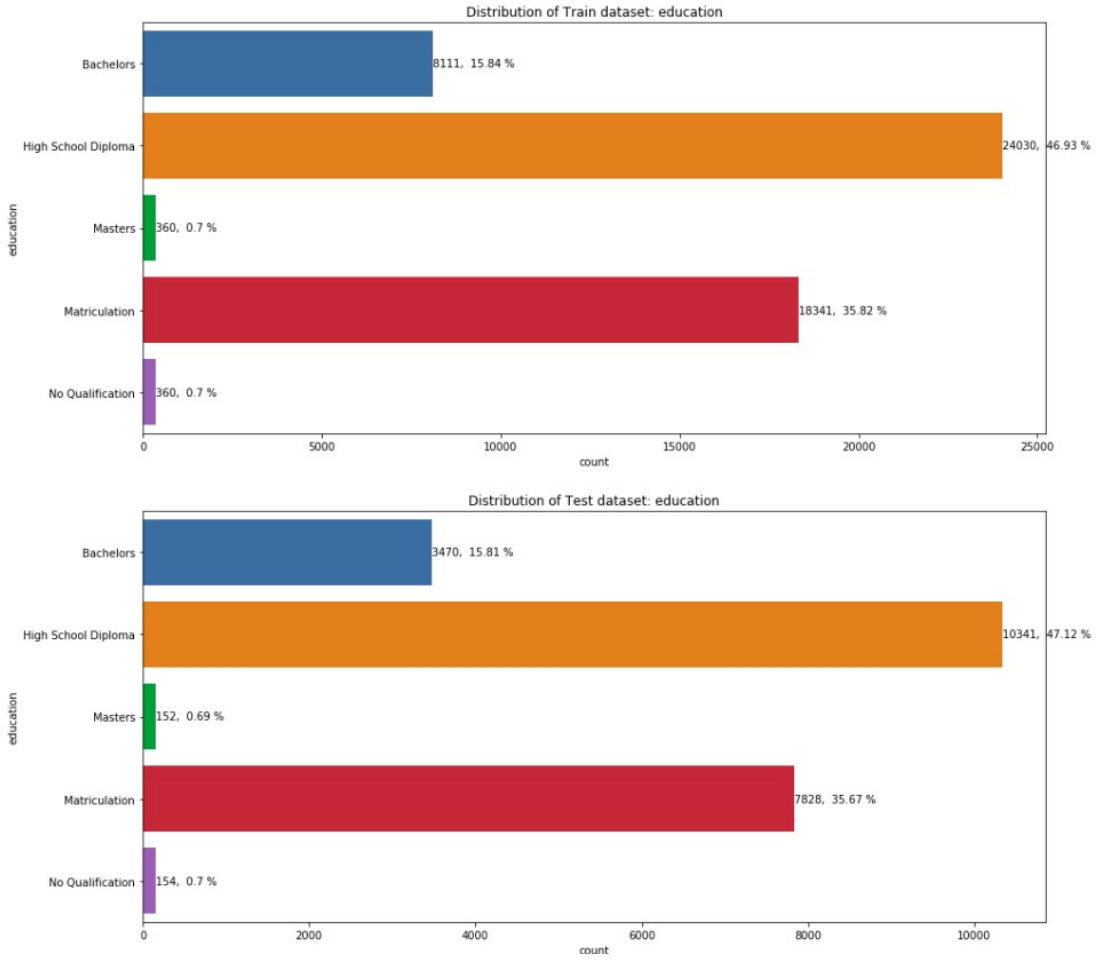




Comparing the distribution of the train and test dataset on the basis of ‘city_tier’
Result: Almost the same distribution

5.

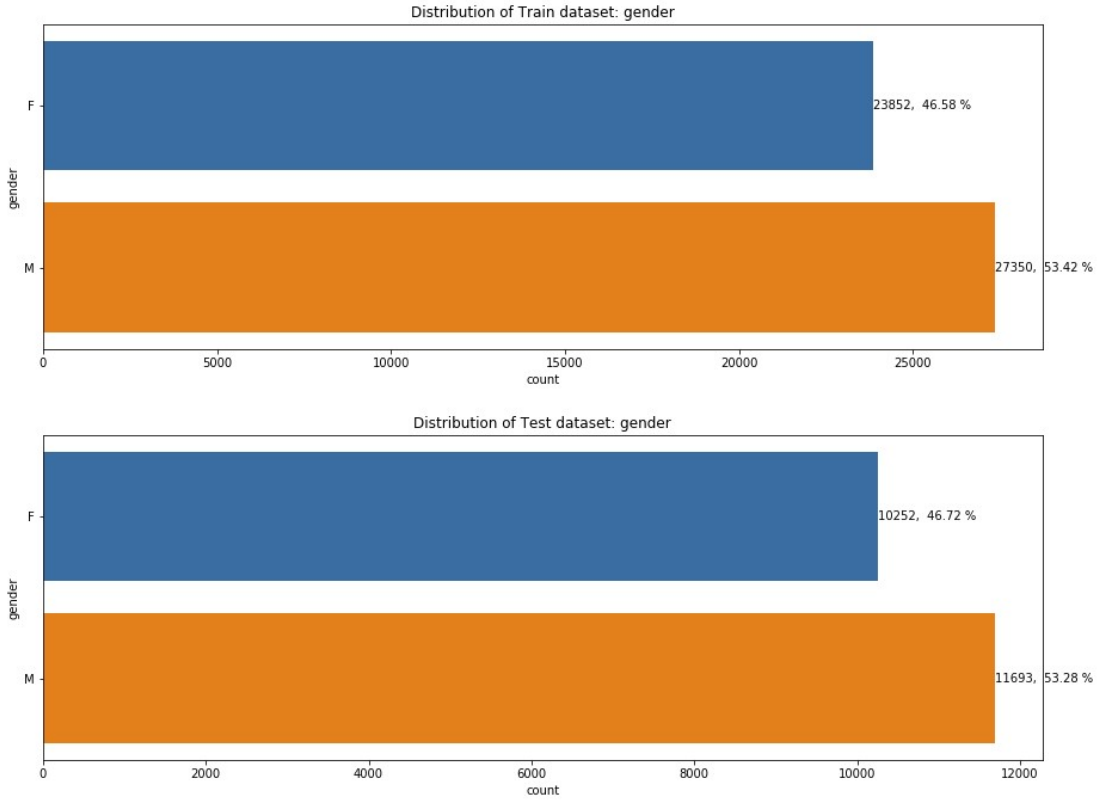
```
plot_bar_counts_categorical(TestData['education'], 'Test dataset: education', (15,7))
```



Comparing the distribution of the train and test dataset on the basis of ‘education’
Result: Almost the same distribution

6.

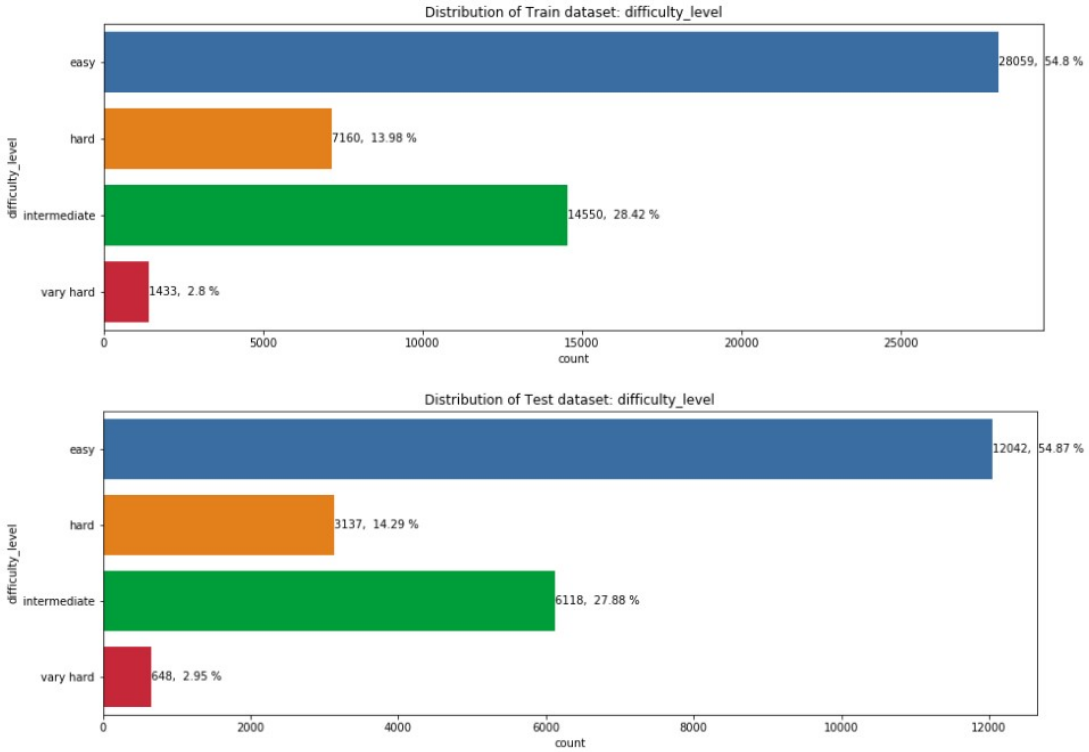
```
In [21]:
1 plot_bar_counts_categorical(TrainData['gender'], 'Train dataset: gender', (15,5))
2 plot_bar_counts_categorical(TestData['gender'], 'Test dataset: gender', (15,5))
```



Comparing the distribution of the train and test dataset on the basis of ‘gender’
Result: Almost the same distribution

7.

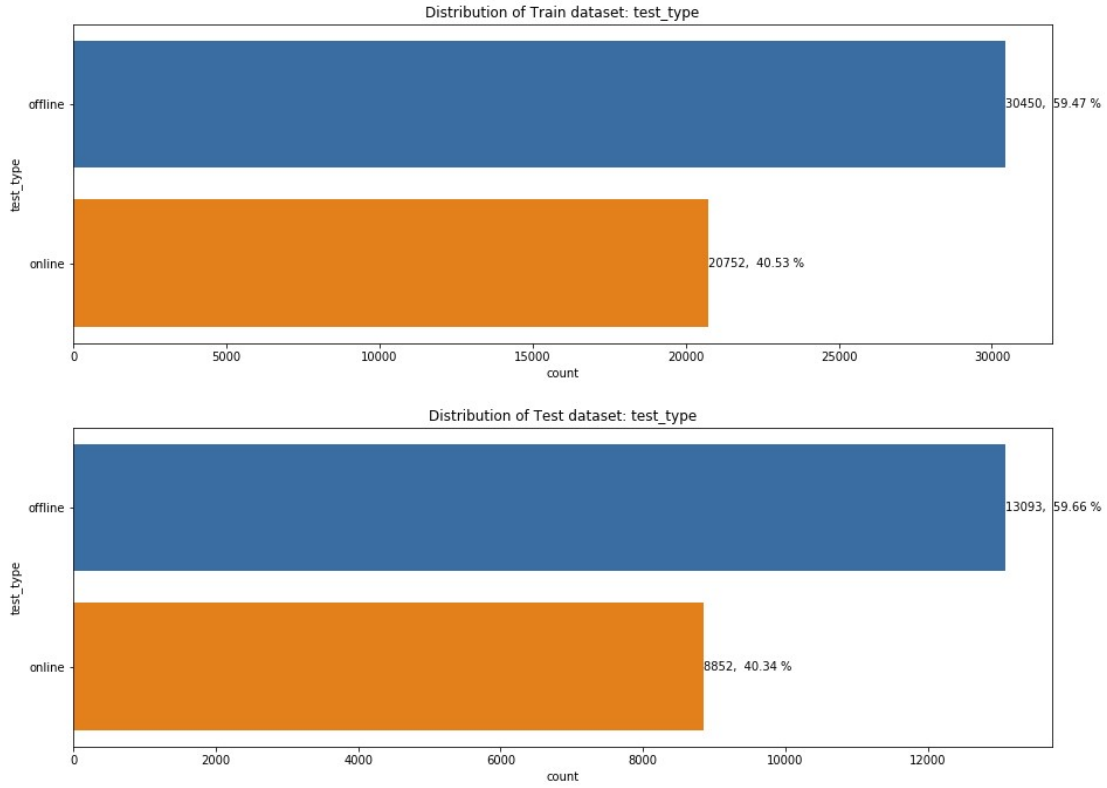
```
In [22]:
1 plot_bar_counts_categorical(TrainData['difficulty_level'], 'Train dataset: difficulty_level', (15,5))
2 plot_bar_counts_categorical(TestData['difficulty_level'], 'Test dataset: difficulty_level', (15,5))
```



Comparing the distribution of the train and test dataset on the basis of ‘difficulty_level’
Result: Almost the same distribution

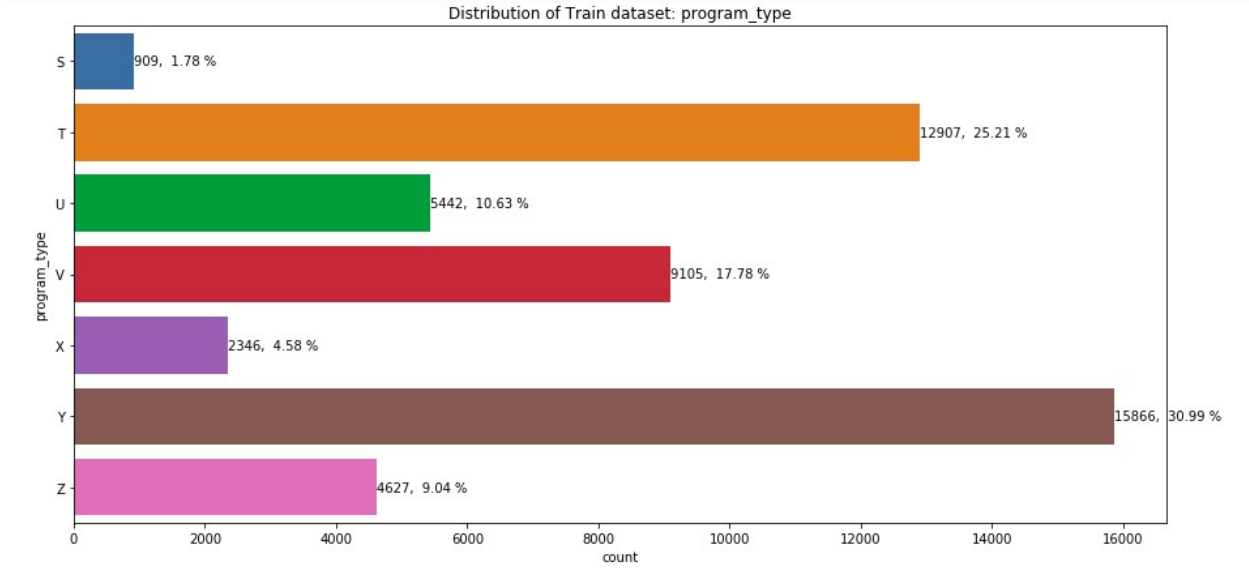
8.

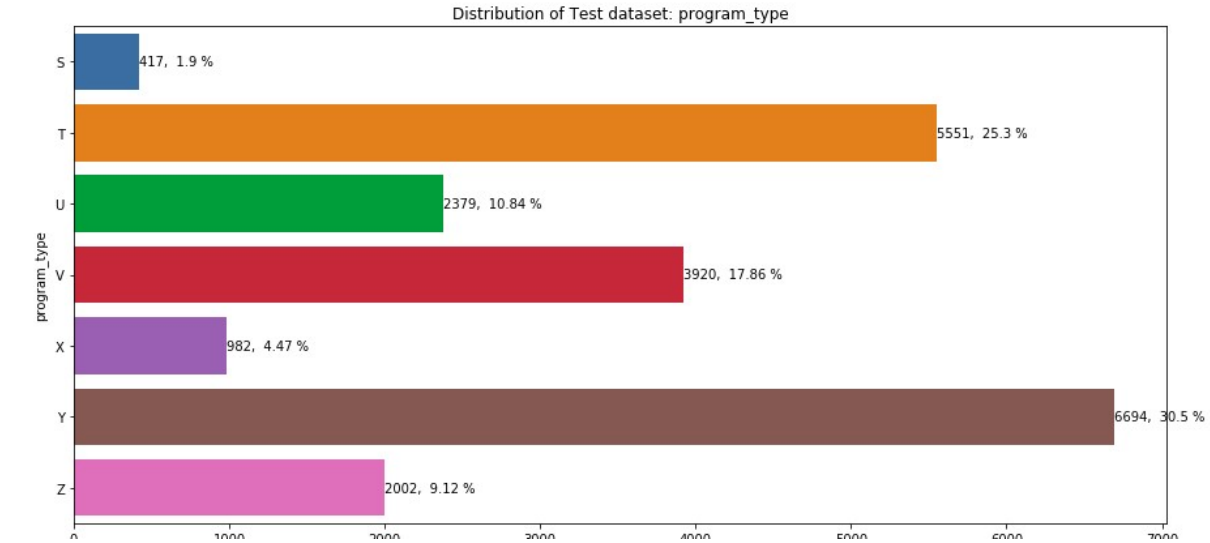
```
In [23]:
1 plot_bar_counts_categorical(TrainData['test_type'], 'Train dataset: test_type', (15,5))
2 plot_bar_counts_categorical(TestData['test_type'], 'Test dataset: test_type', (15,5))
```



Comparing the distribution of the train and test dataset on the basis of ‘test_type’
Result: Almost the same distribution

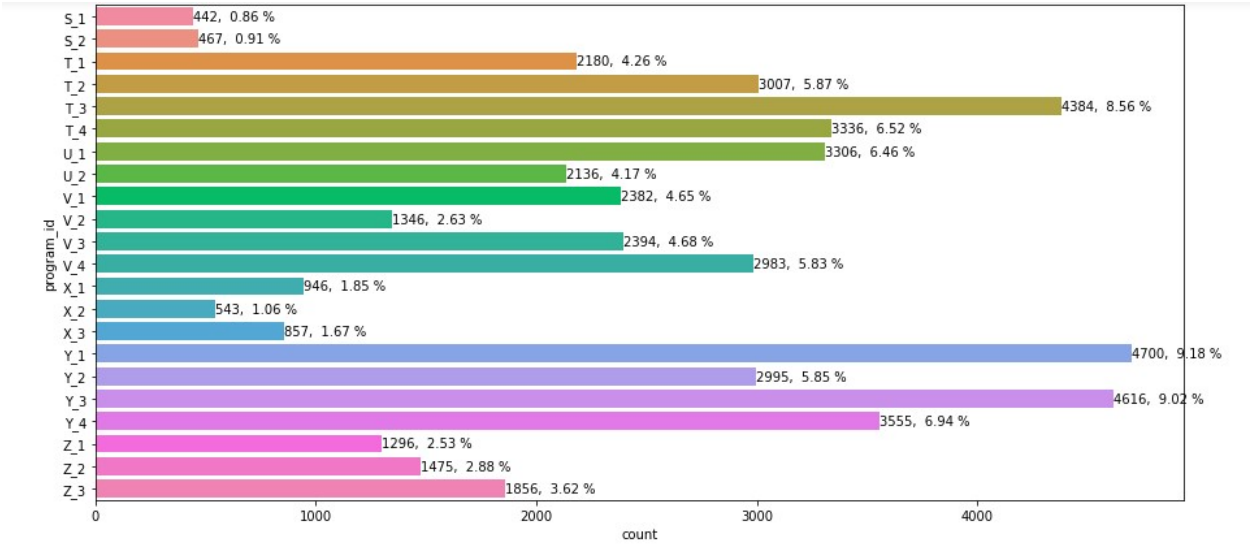
9.

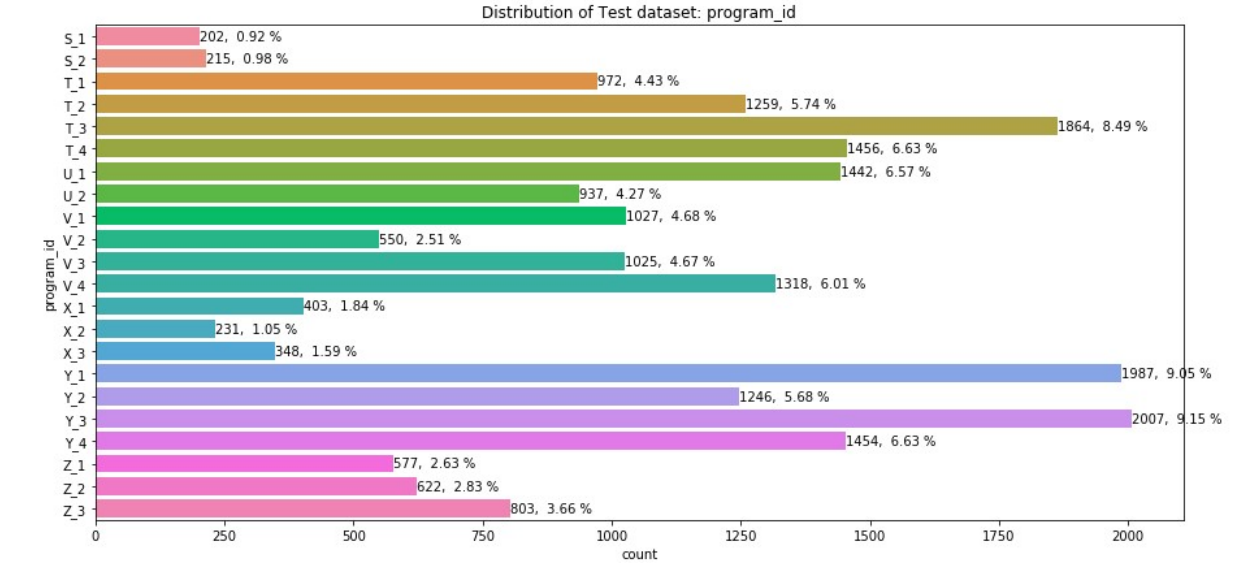




Comparing the distribution of the train and test dataset on the basis of ‘program_type’
Result: Almost the same distribution

10.



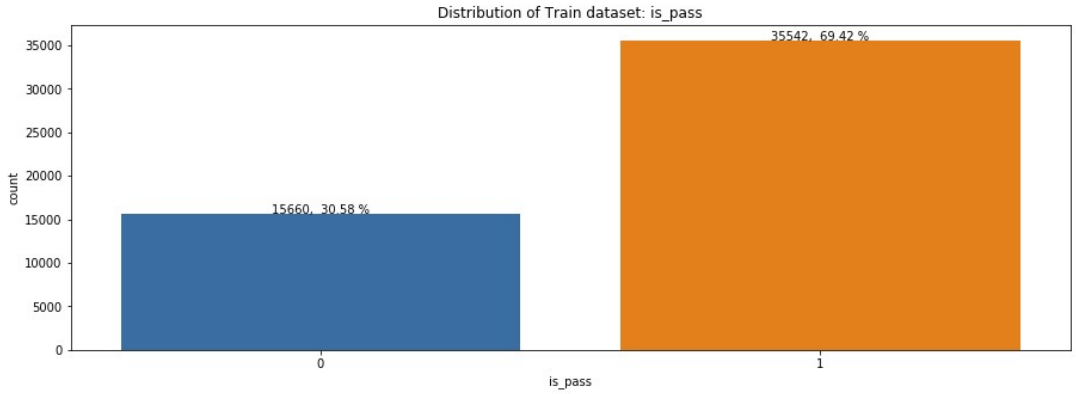


Comparing the distribution of the train and test dataset on the basis of ‘program_id’
Result: Almost the same distribution

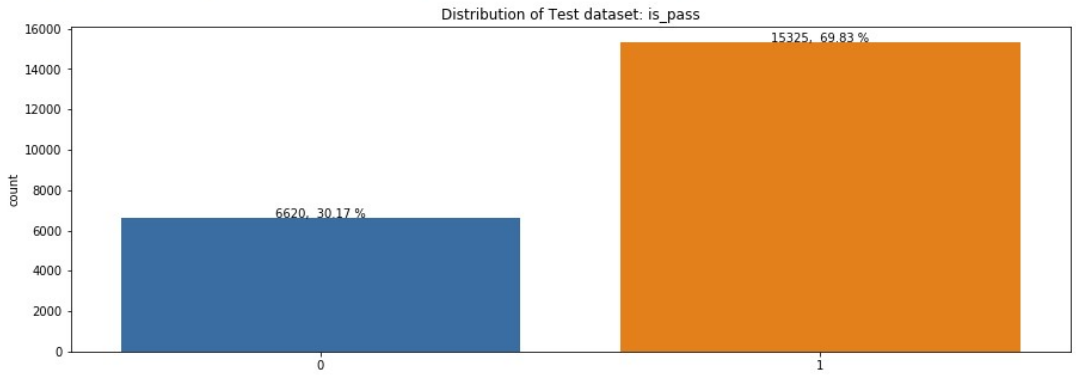
11.

Output Variable ---> 'is_pass'

In [26]:
```
plot_bar_counts_categorical(TrainData['is_pass'], 'Train dataset: is_pass', (15,5))
```



In [27]:
```
plot_bar_counts_categorical(TestData['is_pass'], 'Test dataset: is_pass', (15,5))
```

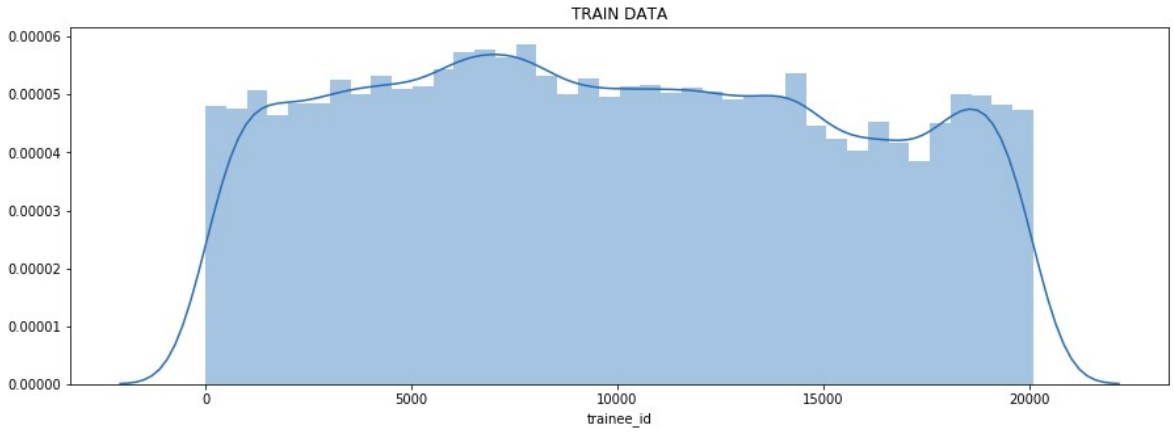


Plotting the distribution of the train and test dataset on the basis of 'program_id'
Result: The number the candidates who passed the test in the training set is approximately equal to the number of candidates who passed the test in the test set.

We will predict the number of students who passed the test using ML algorithms and then compare with these results to check the accuracy of our algorithm

12.

In [28]:
```
plt.figure(figsize=(15, 5))
sns.distplot(TrainData["trainee_id"])
plt.title('TRAIN DATA')
plt.show()

plt.figure(figsize=(15, 5))
sns.distplot(TestData["trainee_id"])
plt.title('TEST DATA')
plt.show()
```



- For these graphs, we infer that the range of students who sat for the exams ranges from 0-20,000.
- Also, we can say that one student gave more than one test (otherwise the histogram bars would have been of equal size).
- The test id in which the maximum number of trainees appeared is 187 for both the train and test dataset

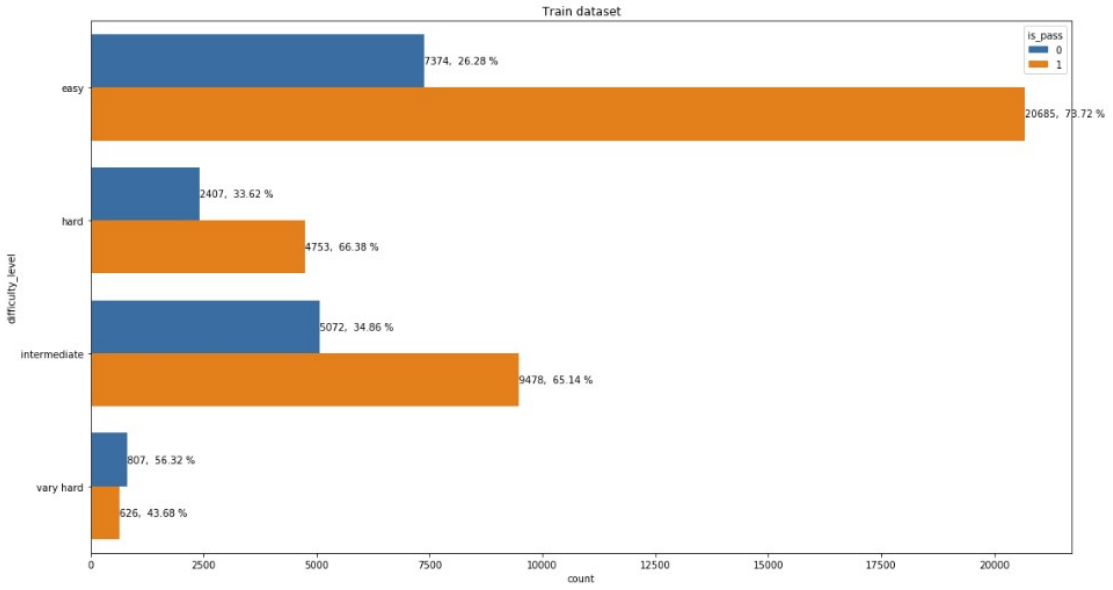
Hence we can conclude that the 'trainee_id' feature cannot be removed.

Bivariate Analysis:

Now we did the bivariate analysis i.e. we found out that how is the 'is_pass' related with the other columns of the dataset.

1.

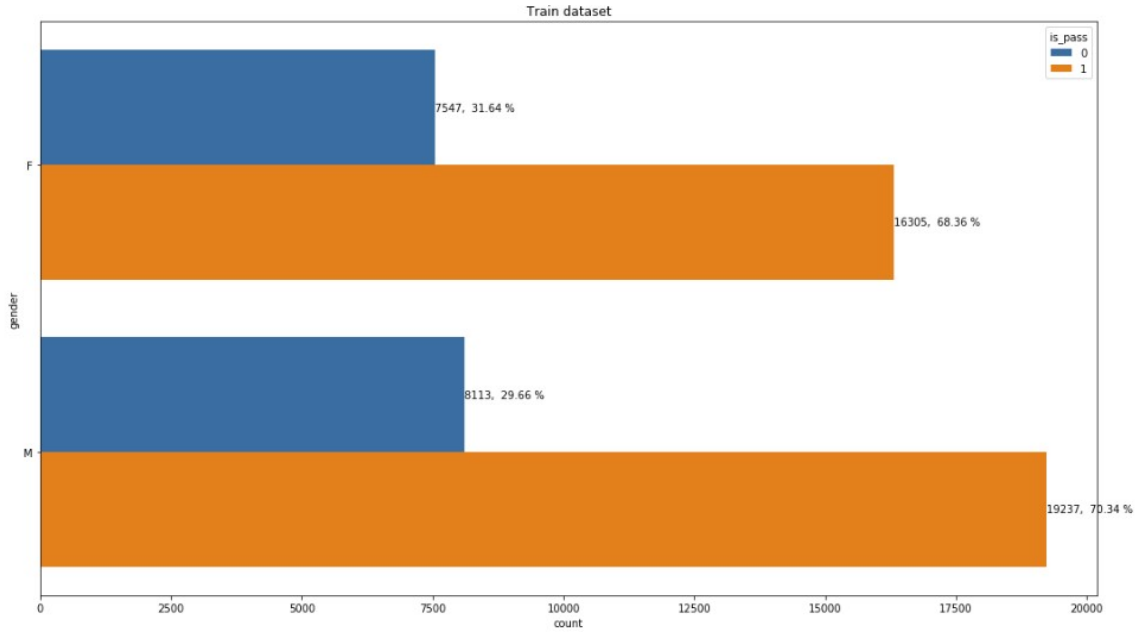
```
In [29]: 1 count_plot_by_hue(TrainData['difficulty_level'], TrainData['is_pass'].astype(str), 'Train dataset', (18,10))
```



Conclusion: The pass percentage decreases and the number of students taking the test decreases as the difficulty level of the test increases.

2.

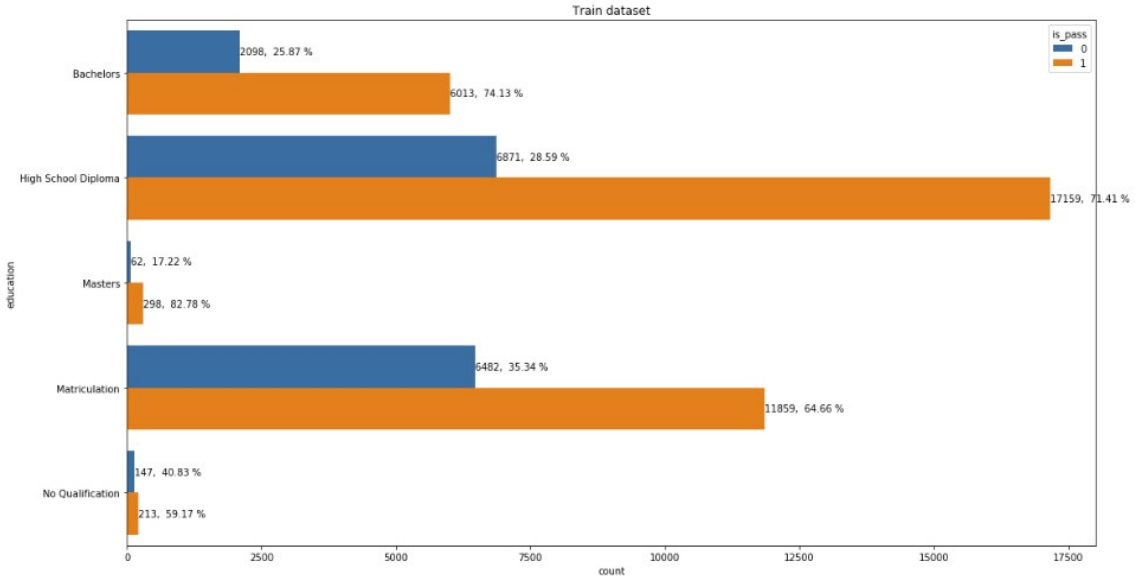
```
n [30]: 1 count_plot_by_hue(TrainData['gender'], TrainData['is_pass'].astype(str), 'Train dataset', (18,10))
```



Conclusion: The pass percentage does not depend on the gender of the person.

3.

```
[31]: 1 count_plot_by_hue(TrainData['education'], TrainData['is_pass'].astype(str), 'Train dataset', (18,10))
```



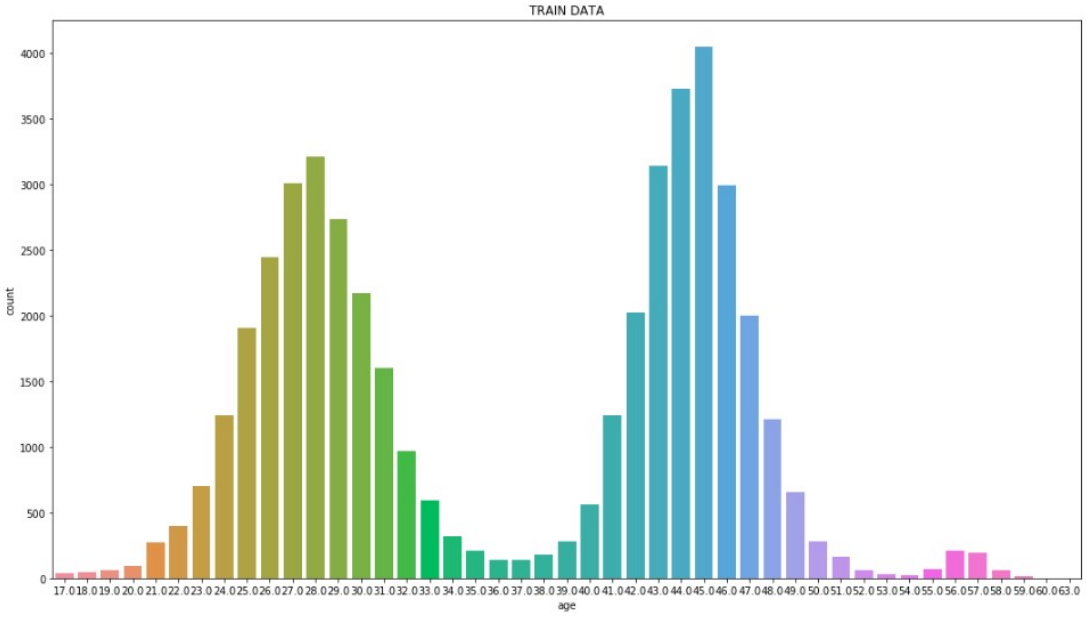
Conclusion: People who did maters give the test in less number but have a higher chance of passing the test as compared to others.

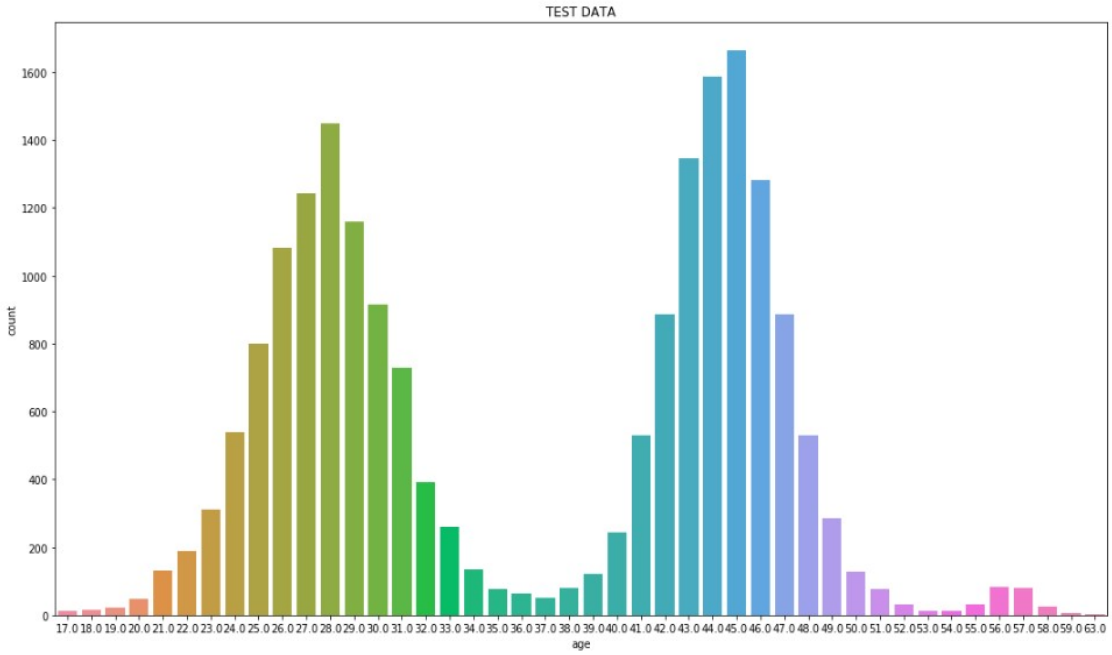
Filling the missing values:

1. These are the plots for the ‘age’ column which consists of missing values.

In [36]:

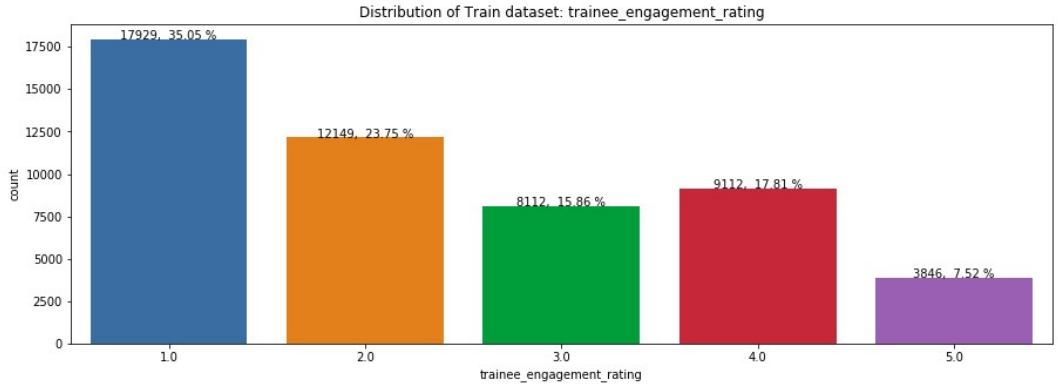
```
plt.figure(figsize=(18, 10))
sns.countplot(TrainData["age"])
plt.title('TRAIN DATA')
plt.show()

plt.figure(figsize=(18, 10))
sns.countplot(TestData["age"])
plt.title('TEST DATA')
plt.show()
```



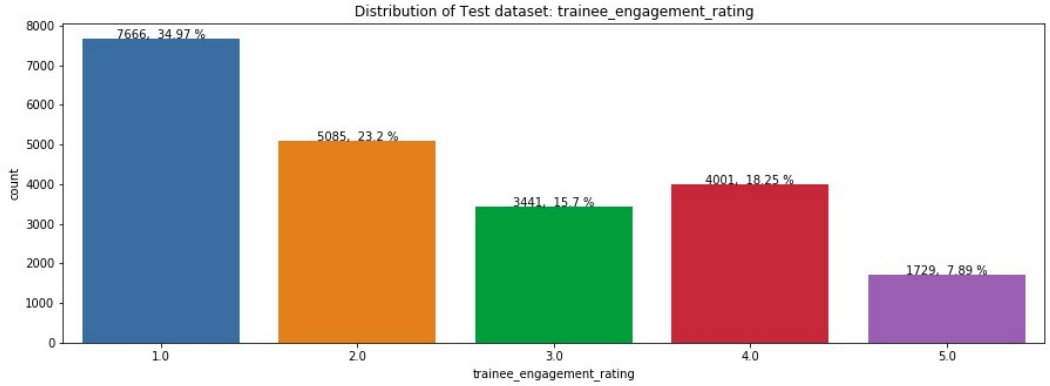


2. These are the plots for the ‘trainee_engagement_rating’ column which consists of missing values.

In [34]:
```
plot_bar_counts_categorical(TrainData['trainee_engagement_rating'], 'Train dataset: trainee_engagement_rating',
```



In [35]:
```
plot_bar_counts_categorical(TestData['trainee_engagement_rating'], 'Test dataset: trainee_engagement_rating', (
```



Now we fill the missing values for ‘age’ and ‘trainee_engagement_rating’.

Filling Missing Values

In [38]:
```
TrainData['trainee_engagement_rating'].fillna(TrainData['trainee_engagement_rating'].mode()[0],inplace=True)
TestData['trainee_engagement_rating'].fillna(TrainData['trainee_engagement_rating'].mode()[0],inplace=True)

TrainData['age'].fillna(TrainData['age'].median(),inplace=True)
TestData['age'].fillna(TestData['age'].median(),inplace=True)
```

Hence now we have

	Total	Percent
program_id_no	0	0.0
trainee_id	0	0.0
program_id	0	0.0
program_type	0	0.0
program_duration	0	0.0
test_id	0	0.0
test_type	0	0.0
difficulty_level	0	0.0
gender	0	0.0
is_pass	0	0.0
education	0	0.0
city_tier	0	0.0
age	0	0.0
total_programs_enrolled	0	0.0
is_handicapped	0	0.0
trainee_engagement_rating	0	0.0
id	0	0.0

TRAIN DATA

	Total	Percent
program_id_no	0	0.0
trainee_engagement_rating	0	0.0
is_handicapped	0	0.0
total_programs_enrolled	0	0.0
age	0	0.0
city_tier	0	0.0
education	0	0.0
gender	0	0.0
trainee_id	0	0.0
difficulty_level	0	0.0
test_type	0	0.0
test_id	0	0.0
program_duration	0	0.0
program_type	0	0.0
program_id	0	0.0
id	0	0.0

TEST DATA

Model:

In the code below LabelEncoder() function is used because the computer doesn't understand the algorithm which uses the usual English language, so we need to map these values to some numeric values. For example, if the column name "Difficulty Level" has values easy, intermediate, hard and very hard, then LabelEncoder() function will map these values to 0,1,2 and 3 respectively. This will enable the computer to understand the algorithm easily.

In [41]:
```
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score,confusion_matrix
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score,confusion_matrix
```

```
/home/preksha/anaconda3/lib/python3.7/site-packages/sklearn/ensemble/weight_boosting.py:29: DeprecationWarning: n
umpy.core.umath_tests is an internal NumPy module and should not be imported. It will be removed in a future NumP
y release.
  from numpy.core.umath_tests import inner1d
```

In [42]:
```
X_Train=TrainData
X_Test=TestData
```

In [43]:
```
le_pid = LabelEncoder()
le_ptype = LabelEncoder()
le_ttest = LabelEncoder()
le_difflev = LabelEncoder()
le_gend = LabelEncoder()
le_edu = LabelEncoder()
le_handi = LabelEncoder()

X_Train['program_id_enc'] = le_pid.fit_transform(X_Train['program_id'])
X_Train['program_type_enc'] = le_ptype.fit_transform(X_Train['program_type'])
X_Train['test_type_enc'] = le_ttest.fit_transform(X_Train['test_type'])
X_Train['difficulty_level_enc'] = le_difflev.fit_transform(X_Train['difficulty_level'])
X_Train['gender_enc'] = le_gend.fit_transform(X_Train['gender'])
X_Train['education_enc'] = le_edu.fit_transform(X_Train['education'])
X_Train['is_handicapped_enc'] = le_handi.fit_transform(X_Train['is_handicapped'])
```

```
X_Test['program_id_enc'] = le_pid.transform(X_Test['program_id'])
X_Test['program_type_enc'] = le_ptype.transform(X_Test['program_type'])
X_Test['test_type_enc'] = le_ttest.transform(X_Test['test_type'])
X_Test['difficulty_level_enc'] = le_difflev.transform(X_Test['difficulty_level'])
X_Test['gender_enc'] = le_gend.transform(X_Test['gender'])
X_Test['education_enc'] = le_edu.transform(X_Test['education'])
X_Test['is_handicapped_enc'] = le_handi.transform(X_Test['is_handicapped'])
```

In [45]:
```
X_Train.head()
```

Out[45]:

ed	trainee_engagement_rating	is_pass	program_id_enc	program_type_enc	test_type_enc	difficulty_level_enc	gender_enc	education_enc	is_handicapped_enc
N	4.0	1	6	2	0	2	1	1	0
N	1.0	0	10	3	0	3	0	1	0
N	1.0	1	4	1	0	0	0	4	0
N	4.0	1	18	5	1	0	1	3	0
N	1.0	1	5	1	1	0	0	1	0

In [46]:
```
X_Test.head()
```

Out[46]:

ed	trainee_engagement_rating	is_pass	program_id_enc	program_type_enc	test_type_enc	difficulty_level_enc	gender_enc	education_enc	is_handicapped_enc
N	2.0	1	8	3	0	3	0	1	0
N	2.0	0	5	1	1	0	0	1	0
N	1.0	1	20	6	1	0	1	3	0
N	1.0	0	9	3	0	2	0	3	0
N	3.0	0	8	3	0	3	0	1	0

Now we will use 3 different classification algorithms to predict the model:

1. Decision Tree Based Algorithm

1 MODEL:

In [52]:
```
dtree=DecisionTreeClassifier(criterion='gini', random_state=0)
dtree.fit(X_Train,Y_Train)
```

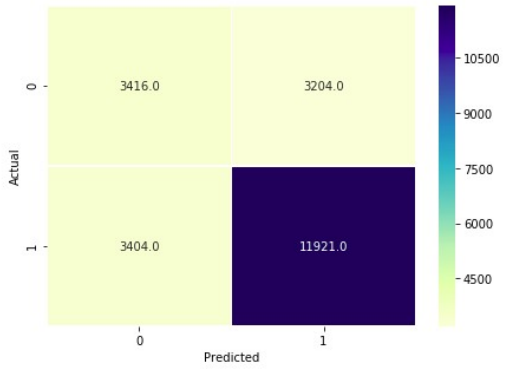
Out[52]:
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=0,
            splitter='best')
```

In [53]:
```
Y_Test_pred = dtree.predict(X_Test)
```

In [54]:
```
confusion_matrix(Y_Test_actual,Y_Test_pred)
```

Out[54]:
```
array([[ 3416,  3204],
       [ 3404, 11921]])
```

In [55]:
```
f,ax = plt.subplots(figsize=(7, 5))
sns.heatmap(confusion_matrix(Y_Test_actual,Y_Test_pred), annot=True, linewidths=.5, fmt= '.1f',ax=ax, cmap="YlGn
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```



In [56]:
```
accuracy_score(Y_Test_actual,Y_Test_pred)
```

Out[56]:
```
0.6988835725677831
```

Conclusions:

- From the HeatMap we can conclude that when the actual result was true, then our algorithm predicted 11921 values correctly. Similarly, we can say that when the actual result was false then 3416 times our algorithm predicted that the result was false.
- Also, we can say that our algorithm predicted incorrectly 3204+3404 times i.e. when the actual value is true it predicts false and when the actual value is false predicts true
- We see that this algorithm gives 69.88% accurate result

2. Random Forest Classifier

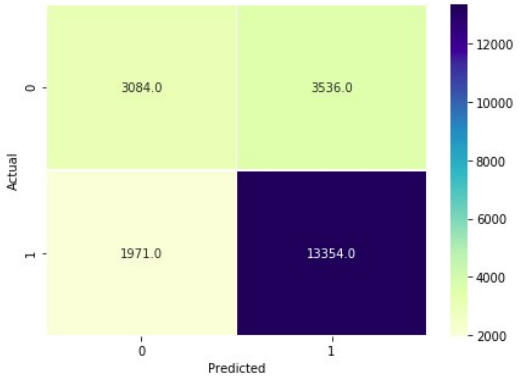
2 MODEL:

In [57]:
```
rf_clf = RandomForestClassifier(n_estimators=200, n_jobs=-1)
rf_clf.fit(X_Train,Y_Train)
Y_Test_pred = rf_clf.predict(X_Test)
```

In [58]:
```
confusion_matrix(Y_Test_actual,Y_Test_pred)
```

Out[58]:
```
array([[ 3084,  3536],
       [ 1971, 13354]])
```

In [59]:
```
f,ax = plt.subplots(figsize=(7, 5))
sns.heatmap(confusion_matrix(Y_Test_actual,Y_Test_pred), annot=True, linewidths=.5, fmt= '.1f',ax=ax, cmap=
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```



In [60]:
```
accuracy_score(Y_Test_actual,Y_Test_pred)
```

Out[60]:
```
0.7490544543176122
```

Conclusions:

- From the HeatMap we can conclude that when the actual result was true, then our algorithm predicted 13354 values correctly. Similarly, we can say that when the actual result was false then 3084 times our algorithm predicted that the result was false.
- Also, we can say that our algorithm predicted incorrectly 3536+1971 times i.e. when the actual value is true it predicts false and when the actual value is false predicts true
- We see that this algorithm gives 74.90% accurate result

3. Logistic Regression

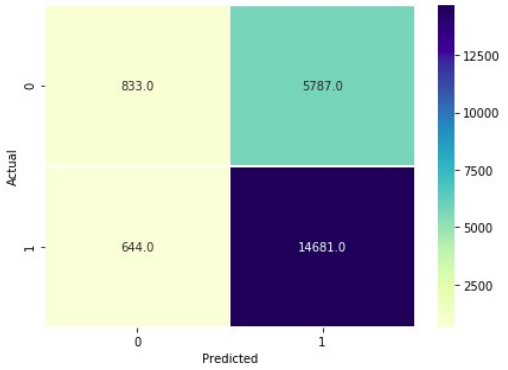
3 MODEL:

```
In [61]:
log_clf = LogisticRegression()
log_clf.fit(X_Train,Y_Train)
Y_Test_pred = log_clf.predict(X_Test)
```

```
In [62]:
confusion_matrix(Y_Test_actual,Y_Test_pred)
```

```
Out[62]:
array([[  833,  5787],
       [  644, 14681]])
```

```
In [63]:
f,ax = plt.subplots(figsize=(7, 5))
sns.heatmap(confusion_matrix(Y_Test_actual,Y_Test_pred), annot=True, linewidths=.5, fmt= '.1f',ax=ax, cmap="YlG
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```




```
In [64]:
accuracy_score(Y_Test_actual,Y_Test_pred)
```

```
Out[64]:
0.7069491911597174
```

Conclusions:

- From the HeatMap we can conclude that when the actual result was true, then our algorithm predicted 14681 values correctly. Similarly, we can say that when the actual result was false then 833 times our algorithm predicted that the result was false.
- Also, we can say that our algorithm predicted incorrectly 5787+644 times i.e. when the actual value is true it predicts false and when the actual value is false predicts true
- We see that this algorithm gives 70.69% accurate result

Among the 3 algorithms, we see that the Random Forest algorithm is the most accurate with 74.90% accuracy

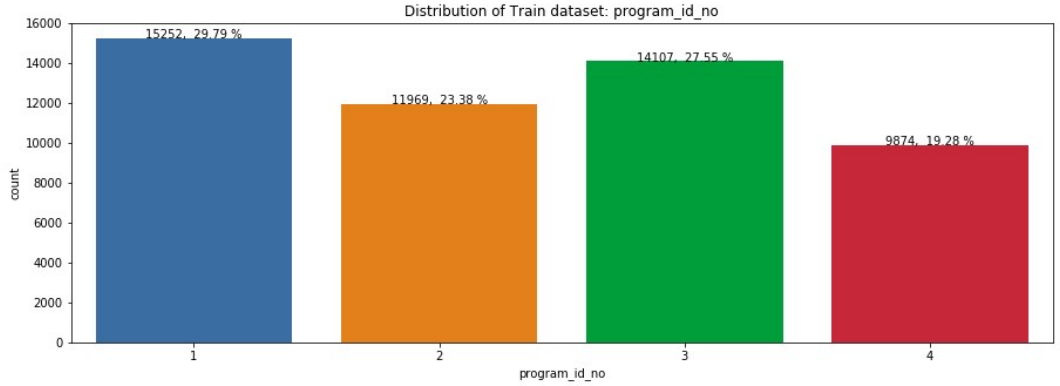
Feature Engineering Step:

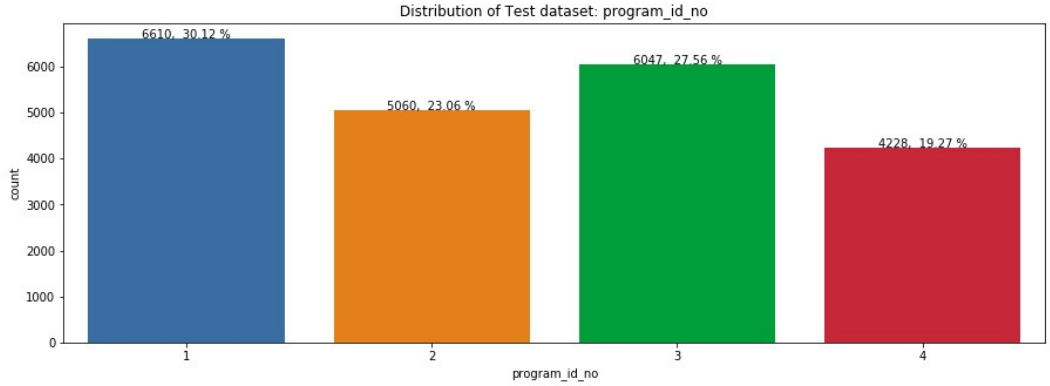
Now we try to improve the accuracy of our result so we did the feature engineering step in which we took the ‘program_id’ feature of the training data and reduced it to ‘program_id_no’

In [56]:
```
TrainData["program_id_no"] = TrainData["program_id"].str[2:].astype(int)
TestData["program_id_no"] = TestData["program_id"].str[2:].astype(int)
```

program_id	program_id_no
Y_1	1
T_1	1
Z_2	2
T_2	2
V_3	3

In [68]:
```
plot_bar_counts_categorical(TrainData['program_id_no'], 'Train dataset: program_id_no', (15,5))
plot_bar_counts_categorical(TestData['program_id_no'], 'Test dataset: program_id_no', (15,5))
```





Now we observe the result after applying feature engineering

1. Decision Tree Based Algorithm

```
In [72]:
dtree=DecisionTreeClassifier(criterion='gini', random_state=0)
dtree.fit(X_Train,Y_Train)
```

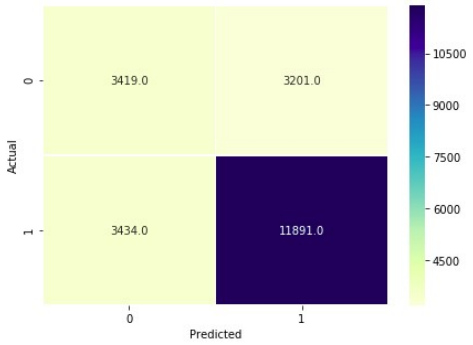
```
Out[72]:
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=0,
            splitter='best')
```

```
In [73]:
Y_Test_pred = dtree.predict(X_Test)
```

```
In [74]:
confusion_matrix(Y_Test_actual,Y_Test_pred)
```

```
Out[74]:
array([[ 3419,  3201],
       [ 3434, 11891]])
```

```
In [75]:
f,ax = plt.subplots(figsize=(7, 5))
sns.heatmap(confusion_matrix(Y_Test_actual,Y_Test_pred), annot=True, linewidths=.5, fmt= '.1f',ax=ax, cmap="YlGnB
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```




```
In [76]:
accuracy_score(Y_Test_actual,Y_Test_pred)
```

```
Out[76]:
0.6976532239690134
```

Conclusions:

- From the HeatMap we can conclude that when the actual result was true, then our algorithm predicted 11821 values correctly. Similarly, we can say that when the actual result was false then 3419 times our algorithm predicted that the result was false.
- Also, we can say that our algorithm predicted incorrectly 3201+3434 times i.e. when the actual value is true it predicts false and when the actual value is false predicts true
- We see that this algorithm gives 69.76% accurate result

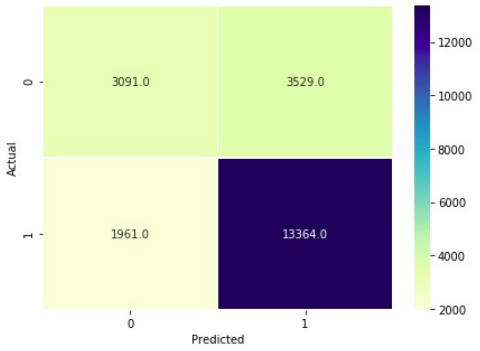
2. Random Forest Classifier

```
In [78]:
rf_clf = RandomForestClassifier(n_estimators=200, n_jobs=-1)
rf_clf.fit(X_Train,Y_Train)
Y_Test_pred = rf_clf.predict(X_Test)
```

```
In [79]:
confusion_matrix(Y_Test_actual,Y_Test_pred)
```

```
Out[79]:
array([[ 3091,  3529],
       [ 1961, 13364]])
```

```
In [80]:
f,ax = plt.subplots(figsize=(7, 5))
sns.heatmap(confusion_matrix(Y_Test_actual,Y_Test_pred), annot=True, linewidths=.5, fmt= '.1f',ax=ax, cmap="YlGnB
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```




```
In [81]:
accuracy_score(Y_Test_actual,Y_Test_pred)
```

```
Out[81]:
0.7498291182501708
```

Conclusions:

- From the HeatMap we can conclude that when the actual result was true, then our algorithm predicted 13364 values correctly. Similarly, we can say that when the actual result was false then 3091 times our algorithm predicted that the result was false.
- Also, we can say that our algorithm predicted incorrectly 3529+1961 times i.e. when the actual value is true it predicts false and when the actual value is false predicts true
- We see that this algorithm gives 74.98% accurate result

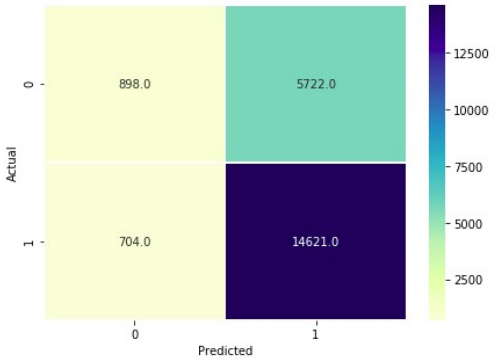
3. Logistic Regression

```
In [83]:
log_clf = LogisticRegression()
log_clf.fit(X_Train,Y_Train)
Y_Test_pred = log_clf.predict(X_Test)
```

```
In [84]:
confusion_matrix(Y_Test_actual,Y_Test_pred)
```

```
Out[84]:
array([[  898,  5722],
       [  704, 14621]])
```

```
In [85]:
f,ax = plt.subplots(figsize=(7, 5))
sns.heatmap(confusion_matrix(Y_Test_actual,Y_Test_pred), annot=True, linewidths=.5, fmt= '.1f',ax=ax, cmap="YlGnB
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```




```
In [86]:
accuracy_score(Y_Test_actual,Y_Test_pred)
```

```
Out[86]:
0.707177033492823
```

Conclusions:

- From the HeatMap we can conclude that when the actual result was true, then our algorithm predicted 14621 values correctly. Similarly, we can say that when the actual result was false then 898 times our algorithm predicted that the result was false.
- Also, we can say that our algorithm predicted incorrectly 5722+704 times i.e. when the actual value is true it predicts false and when the actual value is false predicts true
- We see that this algorithm gives 70.71% accurate result

Conclusion:

- After applying the feature engineering step we came to know that the random forest and the logistic regression gave an increased accuracy and for the decision tree algorithm the accuracy is decreased by some decimal points, but still we can say that the most accurate algorithm is the RandomForest with an accuracy of 74.98%
- We also came to know that the Random Forest algorithm is best suited for our dataset with and without the feature engineering step