Secondary classifier

- All the requirements are clearly defined at the start of project only.
- Not easily adaptable to any changes after the start of the project
- User is involved only during starting phase of the project.
- Extra Skilled professionals are not reuqired in team
- No need of tester from the start of proect. Tester is required only during testing phase
- Works only on small size projects
- Time taken for development of product is more
- cost reuqired is more.
- Risk analysis and management is done at moderate level.
- Risk is highly focused factor
- Documentation is very important or created at each phase pf project.
- Preferable for improvement of an old systems
- reusable components are Developed
- Flexibility in the process
- Good security provided
- Deployment time is less

In [36]:
```
import pandas as pd
import numpy as np
from tqdm import tqdm

import seaborn as sns
sns.set(rc={'figure.figsize':(6,4)})
import matplotlib.pyplot as plt
%matplotlib inline
```

In [37]:
```
col = [
    'All_the_requirements_are_clearly_defined_at_the_start_of_project_only',
    'Not_easily_adaptable_to_any_changes_after_the_start_of_the_project',
    'User_is_involved_only_during_starting_phase_of_the_project',
    'Extra_Skilled_professionals_are_not_reuqired_in_team',
    'No_need_of_tester_from_the_start_of_proect_Tester_is_required_only_during_testing_phase',
    'Works_only_on_small_size_projects',
    'Time_taken_for_development_of_product_is_more',
    'cost_reuqired_is_more',
    'Risk_analysis_and_management_is_done_at_moderate_level',
    'Risk_is_highly_focused_factor',
    'Documentation_is_very_important_or_created_at_each_phase_pf_project',
    'Preferable_for_improvement_of_an_old_systems',
    'reusable_components_are_Developed',
    'Flexibility_in_the_process',
    'Good_security_provided',
    'Deployment_time_is_less'
]
col = [item.lower() for item in col]
col
```

Out[37]:
```
['all_the_requirements_are_clearly_defined_at_the_start_of_project_only',
 'not_easily_adaptable_to_any_changes_after_the_start_of_the_project',
 'user_is_involved_only_during_starting_phase_of_the_project',
 'extra_skilled_professionals_are_not_reuqired_in_team',
 'no_need_of_tester_from_the_start_of_proect_tester_is_required_only_during_testing_phase',
 'works_only_on_small_size_projects',
 'time_taken_for_development_of_product_is_more',
 'cost_reuqired_is_more',
 'risk_analysis_and_management_is_done_at_moderate_level',
 'risk_is_highly_focused_factor',
 'documentation_is_very_important_or_created_at_each_phase_pf_project',
 'preferable_for_improvement_of_an_old_systems',
 'reusable_components_are_developed',
 'flexibility_in_the_process',
 'good_security_provided',
 'deployment_time_is_less']
```

In [38]:
```python
data = pd.DataFrame(columns=col)
data
```

Out[38]:

all_the_requirements_are_clearly_defined_at_the_start_of_project_only	not_easily_adaptable_to_any_changes_after_the_start_of_the_pr

In [39]:
```python
col_name = [f'Q_{i+1}' for i in range(len(data.columns))]
col_len = len(col_name)
data = pd.DataFrame(columns=col_name)

for i in tqdm(range(2**col_len)):
    data.loc[len(data.index)] = [*(bin(i).split('b')[1].zfill(col_len))]

data = data.apply(pd.to_numeric)
data = data.sample(frac=1).reset_index(drop=True)
data.head()
```

```
100%|████████████████████████████████████████████████████████████| 6553
6/65536 [06:38<00:00, 164.39it/s]
```

Out[39]:

	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7	Q_8	Q_9	Q_10	Q_11	Q_12	Q_13	Q_14	Q_15	Q_16
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
4	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0

In [46]:
```python
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 65536 entries, 0 to 65535
Data columns (total 22 columns):
 #   Column                         Non-Null Count  Dtype  
---  ------                         --------------  -----  
 0   Q_1                            65536 non-null  int64  
 1   Q_2                            65536 non-null  int64  
 2   Q_3                            65536 non-null  int64  
 3   Q_4                            65536 non-null  int64  
 4   Q_5                            65536 non-null  int64  
 5   Q_6                            65536 non-null  int64  
 6   Q_7                            65536 non-null  int64  
 7   Q_8                            65536 non-null  int64  
 8   Q_9                            65536 non-null  int64  
 9   Q_10                           65536 non-null  int64  
 10  Q_11                           65536 non-null  int64  
 11  Q_12                           65536 non-null  int64  
 12  Q_13                           65536 non-null  int64  
 13  Q_14                           65536 non-null  int64  
 14  Q_15                           65536 non-null  int64  
 15  Q_16                           65536 non-null  int64  
 16  rapid_application_development  65536 non-null  object 
 17  spiral                         0 non-null      float64
 18  prototyping                    0 non-null      float64
 19  scrum                          0 non-null      float64
 20  extreme_programming            0 non-null      float64
 21  feature_driven_development     0 non-null      float64
dtypes: float64(5), int64(16), object(1)
memory usage: 11.5+ MB
```

In [40]:
```
# c = ['Rapid_Application_Development',
#      'Spiral',
#      'Prototyping',
#      'Scrum',
#      'Extreme_Programming',
#      'Feature_driven_development'
# ]
# c = [item.lower() for item in c]
# c
```

In [79]:
```
data['rapid_application_development'] = np.nan
data['spiral'] = np.nan
data['prototyping'] = np.nan
data['scrum'] = np.nan
data['extreme_programming'] = np.nan
data['feature_driven_development'] = np.nan
data.head()
```

Out[79]:

	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7	Q_8	Q_9	Q_10	...	Q_13	Q_14	Q_15	Q_16	rapid_application_development	spiral	pr
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	NaN	NaN	
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	1	NaN	NaN	
2	0	0	0	0	0	0	0	0	0	0	...	0	0	1	0	NaN	NaN	
3	0	0	0	0	0	0	0	0	0	0	...	0	0	1	1	NaN	NaN	
4	0	0	0	0	0	0	0	0	0	0	...	0	1	0	0	NaN	NaN	

5 rows × 22 columns

In [80]:
```
# for i in col_name:
#     print(f"(data['{i}'] == 1) &")
```

In [82]:
```
data['rapid_application_development'] = np.where(
    (data['Q_1'] == 0) |
    (data['Q_2'] == 1) &
    (data['Q_3'] == 0) |
    (data['Q_4'] == 1) &
    (data['Q_5'] == 0) |
    (data['Q_6'] == 0) |
    (data['Q_7'] == 0) |
    (data['Q_8'] == 1) &
    (data['Q_9'] == 1) &
    (data['Q_10'] == 0) |
    (data['Q_11'] == 1) &
    (data['Q_12'] == 0) |
    (data['Q_13'] == 1) &
    (data['Q_14'] == 0) |
    (data['Q_15'] == 1) &
    (data['Q_16'] == 0), '1', '0')
data.head()
```

Out[82]:

	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7	Q_8	Q_9	Q_10	...	Q_13	Q_14	Q_15	Q_16	rapid_application_development	spiral	pr
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	1	NaN	
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	1	1	NaN	
2	0	0	0	0	0	0	0	0	0	0	...	0	0	1	0	1	NaN	
3	0	0	0	0	0	0	0	0	0	0	...	0	0	1	1	1	NaN	
4	0	0	0	0	0	0	0	0	0	0	...	0	1	0	0	1	NaN	

5 rows × 22 columns

In [83]:
```
data['rapid_application_development'].value_counts()
```

Out[83]:
```
rapid_application_development
1    63835
0     1701
Name: count, dtype: int64
```

In [84]:
```python
data['spiral'] = np.where(
    (data['Q_1'] == 0) |
    (data['Q_2'] == 0) |
    (data['Q_3'] == 0) |
    (data['Q_4'] == 1) &
    (data['Q_5'] == 0) |
    (data['Q_6'] == 1) &
    (data['Q_7'] == 1) &
    (data['Q_8'] == 1) &
    (data['Q_9'] == 0) |
    (data['Q_10'] == 1) &
    (data['Q_11'] == 1) &
    (data['Q_12'] == 0) |
    (data['Q_13'] == 1) &
    (data['Q_14'] == 1) &
    (data['Q_15'] == 0) |
    (data['Q_16'] == 0), '1', '0')
data.head()
```

Out[84]:

	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7	Q_8	Q_9	Q_10	...	Q_13	Q_14	Q_15	Q_16	rapid_application_development	spiral	pr
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	1	1	
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	1	1	1	
2	0	0	0	0	0	0	0	0	0	0	...	0	0	1	0	1	1	
3	0	0	0	0	0	0	0	0	0	0	...	0	0	1	1	1	1	
4	0	0	0	0	0	0	0	0	0	0	...	0	1	0	0	1	1	

5 rows × 22 columns

In [85]:
```python
data['spiral'].value_counts()
```

Out[85]:
```
spiral
1    63331
0     2205
Name: count, dtype: int64
```

In [86]:
```python
data['prototyping'] = np.where(
    (data['Q_1'] == 0) |
    (data['Q_2'] == 0) |
    (data['Q_3'] == 0) |
    (data['Q_4'] == 1) &
    (data['Q_5'] == 0) |
    (data['Q_6'] == 1) &
    (data['Q_7'] == 1) &
    (data['Q_8'] == 0) |
    (data['Q_9'] == 1) &
    (data['Q_10'] == 0) |
    (data['Q_11'] == 1) &
    (data['Q_12'] == 0) |
    (data['Q_13'] == 0) |
    (data['Q_14'] == 1) &
    (data['Q_15'] == 0) |
    (data['Q_16'] == 1), '1', '0')
data.head()
```

Out[86]:

	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7	Q_8	Q_9	Q_10	...	Q_13	Q_14	Q_15	Q_16	rapid_application_development	spiral	pr
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	1	1	
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	1	1	1	
2	0	0	0	0	0	0	0	0	0	0	...	0	0	1	0	1	1	
3	0	0	0	0	0	0	0	0	0	0	...	0	0	1	1	1	1	
4	0	0	0	0	0	0	0	0	0	0	...	0	1	0	0	1	1	

5 rows × 22 columns

In [87]:
```python
data['prototyping'].value_counts()
```

Out[87]:
```
prototyping
1    64969
0      567
Name: count, dtype: int64
```

In [88]:
```python
data['scrum'] = np.where(
    (data['Q_1'] == 0) |
    (data['Q_2'] == 0) |
    (data['Q_3'] == 0) |
    (data['Q_4'] == 0) |
    (data['Q_5'] == 0) |
    (data['Q_6'] == 0) |
    (data['Q_7'] == 0) |
    (data['Q_8'] == 0) |
    (data['Q_9'] == 0) |
    (data['Q_10'] == 0) |
    (data['Q_11'] == 0) |
    (data['Q_12'] == 1) &
    (data['Q_13'] == 1) &
    (data['Q_14'] == 1) &
    (data['Q_15'] == 1) &
    (data['Q_16'] == 1), '1', '0')
data.head()
```

Out[88]:

_7	Q_8	Q_9	Q_10	...	Q_13	Q_14	Q_15	Q_16	rapid_application_development	spiral	prototyping	scrum	extreme_programming
0	0	0	0	...	0	0	0	0	1	1	1	1	NaN
0	0	0	0	...	0	0	0	1	1	1	1	1	NaN
0	0	0	0	...	0	0	1	0	1	1	1	1	NaN
0	0	0	0	...	0	0	1	1	1	1	1	1	NaN
0	0	0	0	...	0	1	0	0	1	1	1	1	NaN

In [89]:
```python
data['scrum'].value_counts()
```

Out[89]:
```
scrum
1    65505
0       31
Name: count, dtype: int64
```

In [90]:
```
data['extreme_programming'] = np.where(
    (data['Q_1'] == 0) |
    (data['Q_2'] == 0) |
    (data['Q_3'] == 0) |
    (data['Q_4'] == 1) &
    (data['Q_5'] == 0) |
    (data['Q_6'] == 0) |
    (data['Q_7'] == 1) &
    (data['Q_8'] == 1) &
    (data['Q_9'] == 0) |
    (data['Q_10'] == 1) &
    (data['Q_11'] == 0) |
    (data['Q_12'] == 1) &
    (data['Q_13'] == 1) &
    (data['Q_14'] == 0) |
    (data['Q_15'] == 1) &
    (data['Q_16'] == 1), '1', '0')
data.head()
```

Out[90]:

_7	Q_8	Q_9	Q_10	...	Q_13	Q_14	Q_15	Q_16	rapid_application_development	spiral	prototyping	scrum	extreme_programming
0	0	0	0	...	0	0	0	0	1	1	1	1	1
0	0	0	0	...	0	0	0	1	1	1	1	1	1
0	0	0	0	...	0	0	1	0	1	1	1	1	1
0	0	0	0	...	0	0	1	1	1	1	1	1	1
0	0	0	0	...	0	1	0	0	1	1	1	1	1

In [91]:
```
data['extreme_programming'].value_counts()
```

Out[91]:
```
extreme_programming
1    64213
0     1323
Name: count, dtype: int64
```

In [92]:
```
data['feature_driven_development'] = np.where(
    (data['Q_1'] == 0) |
    (data['Q_2'] == 0) |
    (data['Q_3'] == 0) |
    (data['Q_4'] == 0) |
    (data['Q_5'] == 0) |
    (data['Q_6'] == 0) |
    (data['Q_7'] == 1) &
    (data['Q_8'] == 0) |
    (data['Q_9'] == 1) &
    (data['Q_10'] == 0) |
    (data['Q_11'] == 0) |
    (data['Q_12'] == 0) |
    (data['Q_13'] == 1) &
    (data['Q_14'] == 1) &
    (data['Q_15'] == 0) |
    (data['Q_16'] == 1), '1', '0')
data.head()
```

Out[92]:

	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7	Q_8	Q_9	Q_10	...	Q_13	Q_14	Q_15	Q_16	rapid_application_development	spiral	pı
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	1	1	
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	1	1	1	
2	0	0	0	0	0	0	0	0	0	0	...	0	0	1	0	1	1	
3	0	0	0	0	0	0	0	0	0	0	...	0	0	1	1	1	1	
4	0	0	0	0	0	0	0	0	0	0	...	0	1	0	0	1	1	

5 rows × 22 columns

In [93]:
```python
data['feature_driven_development'].value_counts()
```

Out[93]:
```
feature_driven_development
1    65473
0       63
Name: count, dtype: int64
```

In [94]:
```python
data
```

Out[94]:

Q_7	Q_8	Q_9	Q_10	...	Q_13	Q_14	Q_15	Q_16	rapid_application_development	spiral	prototyping	scrum	extreme_programmin
0	0	0	0	...	0	0	0	0	1	1	1	1	
0	0	0	0	...	0	0	0	1	1	1	1	1	
0	0	0	0	...	0	0	1	0	1	1	1	1	
0	0	0	0	...	0	0	1	1	1	1	1	1	
0	0	0	0	...	0	1	0	0	1	1	1	1	
...	...	...	...	...	...	...	...	...	...	...	...	...	
1	1	1	1	...	1	0	1	1	1	0	1	0	
1	1	1	1	...	1	1	0	0	0	1	1	0	
1	1	1	1	...	1	1	0	1	0	1	1	0	
1	1	1	1	...	1	1	1	0	1	1	0	0	
1	1	1	1	...	1	1	1	1	0	0	1	1	

In [95]:
```python
data.to_csv('dataset\dataset_under_W_and_A_raw.csv', index=False)
```

In [97]:
```python
data['under_waterfall_lable'] = data.apply(lambda x: str(x['rapid_application_development']) +
                                                      str(x['spiral']) +
                                                      str(x['prototyping']),
                                            axis=1)

data['under_agile_lable'] = data.apply(lambda x: str(x['scrum']) +
                                                  str(x['extreme_programming']) +
                                                  str(x['feature_driven_development']),
                                        axis=1)
```

In [98]:
```python
data
```

Out[98]:

_15	Q_16	rapid_application_development	spiral	prototyping	scrum	extreme_programming	feature_driven_development	under_wate
0	0	1	1	1	1	1	1	
0	1	1	1	1	1	1	1	
1	0	1	1	1	1	1	1	
1	1	1	1	1	1	1	1	
0	0	1	1	1	1	1	1	
...	...	...	...	...	...	...	...	
1	1	1	0	1	0	1	1	
0	0	0	1	1	0	0	1	
0	1	0	1	1	0	0	1	
1	0	1	1	0	0	0	0	
1	1	0	0	1	1	1	1	

In [99]:
```
data['under_waterfall_lable'].value_counts()
```

Out[99]:
```
under_waterfall_lable
111    61288
101     1980
011     1476
110      567
001      225
Name: count, dtype: int64
```

In [100]:
```
data['under_agile_lable'].value_counts()
```

Out[100]:
```
under_agile_lable
111    64177
101     1272
100       30
110       26
001       16
011        8
000        5
010        2
Name: count, dtype: int64
```

In [101]:
```
data.columns
```

Out[101]:
```
Index(['Q_1', 'Q_2', 'Q_3', 'Q_4', 'Q_5', 'Q_6', 'Q_7', 'Q_8', 'Q_9', 'Q_10',
       'Q_11', 'Q_12', 'Q_13', 'Q_14', 'Q_15', 'Q_16',
       'rapid_application_development', 'spiral', 'prototyping', 'scrum',
       'extreme_programming', 'feature_driven_development',
       'under_waterfall_lable', 'under_agile_lable'],
      dtype='object')
```

In [102]:
```
under_waterfall_data = data[['Q_1', 'Q_2', 'Q_3', 'Q_4', 'Q_5', 'Q_6', 'Q_7', 'Q_8', 'Q_9', 'Q_10',
       'Q_11', 'Q_12', 'Q_13', 'Q_14', 'Q_15', 'Q_16', 'under_waterfall_lable']]
under_waterfall_data
```

Out[102]:

	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7	Q_8	Q_9	Q_10	Q_11	Q_12	Q_13	Q_14	Q_15	Q_16	under_waterfall_lable
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	111
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	111
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	111
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	111
4	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	111
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
65531	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	101
65532	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	011
65533	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	011
65534	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	110
65535	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	001

65536 rows × 17 columns

In [103]:
```
under_waterfall_data['under_waterfall_lable'].value_counts()
```

Out[103]:
```
under_waterfall_lable
111    61288
101     1980
011     1476
110      567
001      225
Name: count, dtype: int64
```

In [105]:
```
under_waterfall_data['class'] = np.select(
    [under_waterfall_data['under_waterfall_lable'] == '001',
     under_waterfall_data['under_waterfall_lable'] == '011',
     under_waterfall_data['under_waterfall_lable'] == '101',
     under_waterfall_data['under_waterfall_lable'] == '110',
     under_waterfall_data['under_waterfall_lable'] == '111'],
    [0, 1, 2, 3, 4],
    default='unknown'
)
under_waterfall_data.head()
```

```
C:\Users\DELL\AppData\Local\Temp\ipykernel_9192\1067872787.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  under_waterfall_data['class'] = np.select(
```

Out[105]:

	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7	Q_8	Q_9	Q_10	Q_11	Q_12	Q_13	Q_14	Q_15	Q_16	under_waterfall_lable	class
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	111	4
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	111	4
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	111	4
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	111	4
4	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	111	4

In [106]:
```
under_waterfall_data['class'].value_counts()
```

Out[106]:
```
class
4    61288
2     1980
1     1476
3      567
0      225
Name: count, dtype: int64
```

In [107]:
```
under_waterfall_data.to_csv('dataset\dataset_under_waterfall_data_class.csv', index=False)
```

In [108]:
```
data.columns
```

Out[108]:
```
Index(['Q_1', 'Q_2', 'Q_3', 'Q_4', 'Q_5', 'Q_6', 'Q_7', 'Q_8', 'Q_9', 'Q_10',
       'Q_11', 'Q_12', 'Q_13', 'Q_14', 'Q_15', 'Q_16',
       'rapid_application_development', 'spiral', 'prototyping', 'scrum',
       'extreme_programming', 'feature_driven_development',
       'under_waterfall_lable', 'under_agile_lable'],
      dtype='object')
```

In [109]:
```python
under_agile_data = data[['Q_1', 'Q_2', 'Q_3', 'Q_4', 'Q_5', 'Q_6', 'Q_7', 'Q_8', 'Q_9', 'Q_10',
        'Q_11', 'Q_12', 'Q_13', 'Q_14', 'Q_15', 'Q_16', 'under_agile_lable']]
under_agile_data
```

Out[109]:

	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7	Q_8	Q_9	Q_10	Q_11	Q_12	Q_13	Q_14	Q_15	Q_16	under_agile_lable
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	111
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	111
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	111
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	111
4	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	111
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
65531	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	011
65532	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	001
65533	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	001
65534	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	000
65535	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	111

65536 rows × 17 columns

In [110]:
```python
under_agile_data['under_agile_lable'].value_counts()
```

Out[110]:
```
under_agile_lable
111    64177
101     1272
100       30
110       26
001       16
011        8
000        5
010        2
Name: count, dtype: int64
```

In [111]:
```python
under_agile_data['class'] = np.select(
    [under_agile_data['under_agile_lable'] == '000',
     under_agile_data['under_agile_lable'] == '001',
     under_agile_data['under_agile_lable'] == '010',
     under_agile_data['under_agile_lable'] == '011',
     under_agile_data['under_agile_lable'] == '100',
     under_agile_data['under_agile_lable'] == '101',
     under_agile_data['under_agile_lable'] == '110',
     under_agile_data['under_agile_lable'] == '111'],
    [0, 1, 2, 3, 4, 5, 6, 7],
    default='unknown'
)
under_agile_data.head()
```

```
C:\Users\DELL\AppData\Local\Temp\ipykernel_9192\4048808119.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  under_agile_data['class'] = np.select(
```

Out[111]:

	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7	Q_8	Q_9	Q_10	Q_11	Q_12	Q_13	Q_14	Q_15	Q_16	under_agile_lable	class
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	111	7
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	111	7
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	111	7
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	111	7
4	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	111	7

In [112]:
```
under_agile_data['class'].value_counts()
```
Out[112]:
```
class
7    64177
5     1272
4       30
6       26
1       16
3        8
0        5
2        2
Name: count, dtype: int64
```
In [113]:
```
under_agile_data.to_csv('dataset\dataset_under_agile_data_class.csv', index=False)
```
Model Training - Under Waterfall data

In [114]:
```
from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier

from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import log_loss
from sklearn.metrics import cohen_kappa_score
from sklearn.metrics import confusion_matrix
from sklearn import metrics

import pickle
import time

# for ignore warnings
import warnings
warnings.filterwarnings("ignore")
```
In [122]:
```
data = pd.read_csv('dataset\dataset_under_waterfall_data_class.csv')
data.head()
```
Out[122]:

	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7	Q_8	Q_9	Q_10	Q_11	Q_12	Q_13	Q_14	Q_15	Q_16	under_waterfall_lable	class
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	111	4
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	111	4
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	111	4
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	111	4
4	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	111	4

In [123]:
```
data.shape
```
Out[123]:
```
(65536, 18)
```

In [124]:
```
data.info()
```

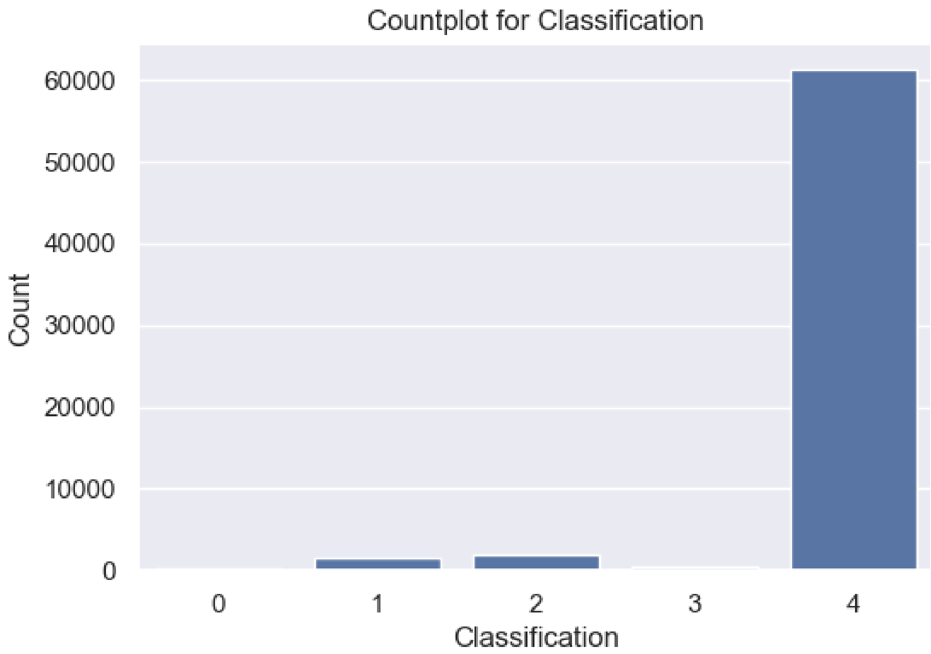
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 65536 entries, 0 to 65535
Data columns (total 18 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Q_1                   65536 non-null  int64
 1   Q_2                   65536 non-null  int64
 2   Q_3                   65536 non-null  int64
 3   Q_4                   65536 non-null  int64
 4   Q_5                   65536 non-null  int64
 5   Q_6                   65536 non-null  int64
 6   Q_7                   65536 non-null  int64
 7   Q_8                   65536 non-null  int64
 8   Q_9                   65536 non-null  int64
 9   Q_10                  65536 non-null  int64
 10  Q_11                  65536 non-null  int64
 11  Q_12                  65536 non-null  int64
 12  Q_13                  65536 non-null  int64
 13  Q_14                  65536 non-null  int64
 14  Q_15                  65536 non-null  int64
 15  Q_16                  65536 non-null  int64
 16  under_waterfall_lable 65536 non-null  int64
 17  class                 65536 non-null  int64
dtypes: int64(18)
memory usage: 9.0 MB
```

In [125]:
```
round(data.describe(),2)
```

Out[125]:

	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7	Q_8	Q_9	Q_10	Q_11	Q_12	Q_13	Q_14
count	65536.0	65536.0	65536.0	65536.0	65536.0	65536.0	65536.0	65536.0	65536.0	65536.0	65536.0	65536.0	65536.0	65536.0
mean	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
std	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
min	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
25%	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
50%	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
75%	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
max	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0

In [126]:
```
# Check If A Dataset Is Imbalanced
sns.countplot(x=data["class"])
plt.title('Countplot for Classification')
plt.xlabel('Classification')
plt.ylabel('Count')
plt.show()
```



In [127]:
```
# Drop columns
columns_to_drop = ['under_waterfall_lable']
data.drop(columns=columns_to_drop, inplace=True)
data = data.sample(frac=1).reset_index(drop=True)
data.head()
```

Out[127]:

	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7	Q_8	Q_9	Q_10	Q_11	Q_12	Q_13	Q_14	Q_15	Q_16	class
0	1	1	0	0	0	1	0	0	1	1	1	1	0	0	1	0	4
1	0	0	0	1	0	0	1	0	1	1	0	1	0	1	1	0	4
2	1	0	0	1	1	0	0	0	1	0	1	0	1	0	1	1	4
3	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	0	3
4	1	0	1	1	1	0	0	0	0	1	0	1	1	1	1	0	4

In [128]:
```
X, Y = data.drop(['class'], axis = 1),  data['class']
# train_test_split 80/20
X_train, X_test, y_train, y_test = train_test_split(X,Y, test_size = 0.2, random_state = 42, stratify = Y
```

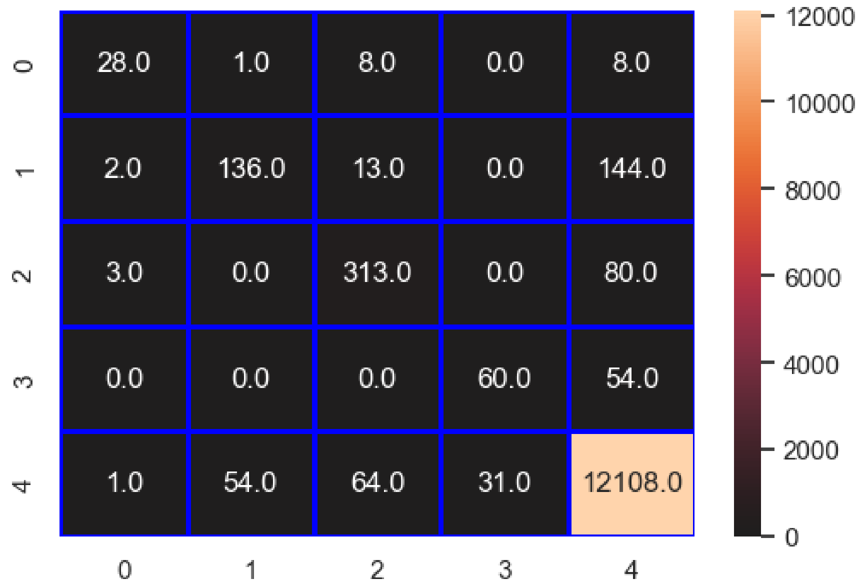
In [129]:
```
# Model initialization
lr_Classifier = LogisticRegression()
knn_Classifier = KNeighborsClassifier()
gnb_Classifier = GaussianNB()
dt_Classifier = DecisionTreeClassifier()
rf_Classifier = RandomForestClassifier()
model_list = [lr_Classifier, knn_Classifier, gnb_Classifier, dt_Classifier, rf_Classifier]
```

In [130]:

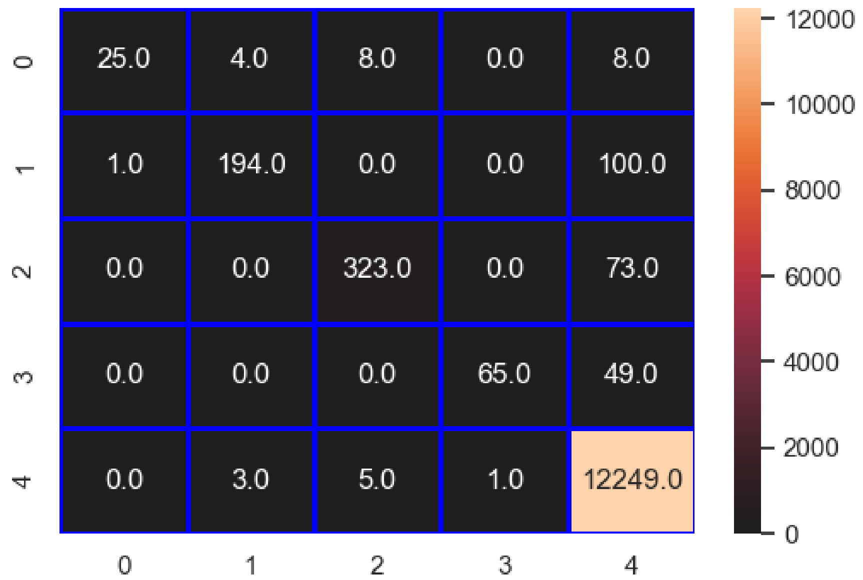
```
plot_data_list = []
def run_pipeline(X_train, X_test, y_train, y_test, classifier):
    # Model Information
    print(f"Modele name : {type(classifier).__name__}")

    # process 2 : train model
    classifier.fit(X_train, y_train)

    # process 4 : test model
    y_pred = classifier.predict(X_test)

    # process 5 : Perform k-fold cross-validation using cross_val_score
    scores = cross_val_score(classifier, X_train, y_train, cv=10, scoring='accuracy')
    print(f"10 K-Fold Accuracy_score : {np.round_(scores,4)}")
    print(f"10 K-Fold Average Accuracy_score : {round(np.average(scores)*100,2)} %")

    # process 6 : model evalution
    print("Accuracy_score:", round((accuracy_score(y_test, y_pred))*100,2),'%')
    print("Loss:", round((1-accuracy_score(y_test, y_pred))*100,2),'%')
    print("Cohen_kappa_score:", round((cohen_kappa_score(y_test, y_pred))*100,2),'%')
    print("Classification_report:\n",metrics.classification_report(y_test, y_pred))
    print("confusion_matrix:\n", confusion_matrix(y_test, y_pred))
    # plot confusion_matrix
    fig, ax = plt.subplots()
    fig.set_size_inches(6,4) # WH
    sns.heatmap(confusion_matrix(y_test, y_pred),
                annot=True,
                fmt=".1f",
                linewidths = 2,
                linecolor = "blue",
                center=0)
    plt.show()

    # process 7 : save model in pkl file
    filename = f'Models\\{str(type(classifier).__name__)}_acc{round((accuracy_score(y_test, y_pred))*100,
    pickle.dump(classifier, open(filename, 'wb'))

    # collect data for bar plot
    global plot_data_list
    plot_data_list.append([str(type(classifier).__name__),
                           round((accuracy_score(y_test, y_pred))*100,2)])

    # end
    print("==="*30)
    print("\n\n")
    time.sleep(0.5)
```

In [131]:
```python
for model in model_list:
    # for scaler in scaler_list:
    run_pipeline(X_train, X_test, y_train, y_test, model)

# plot data
# plot_df = pd.DataFrame(plot_data_list, columns=['classifier', 'scaler', 'accuracy_score'])
plot_df = pd.DataFrame(plot_data_list, columns=['classifier', 'accuracy_score'])
plot_df.to_csv(f"dataset\\accuracy_score_plot_data_Secondary_classifier_Under_Waterfall_Prediction.csv",

sns.set(rc={'figure.figsize':(10,6)})
# ax = sns.barplot(data=plot_df, x="classifier", y="accuracy_score", hue="scaler")
ax = sns.barplot(data=plot_df, x="classifier", y="accuracy_score") # , hue="classifier"
plt.title('Accuracy Score Plot')
plt.xlabel('Classifier')
plt.ylabel('Accuracy Score')
ax.tick_params(axis='x', rotation=5)
for i in ax.containers:
    ax.bar_label(i,)
plt.show()
```

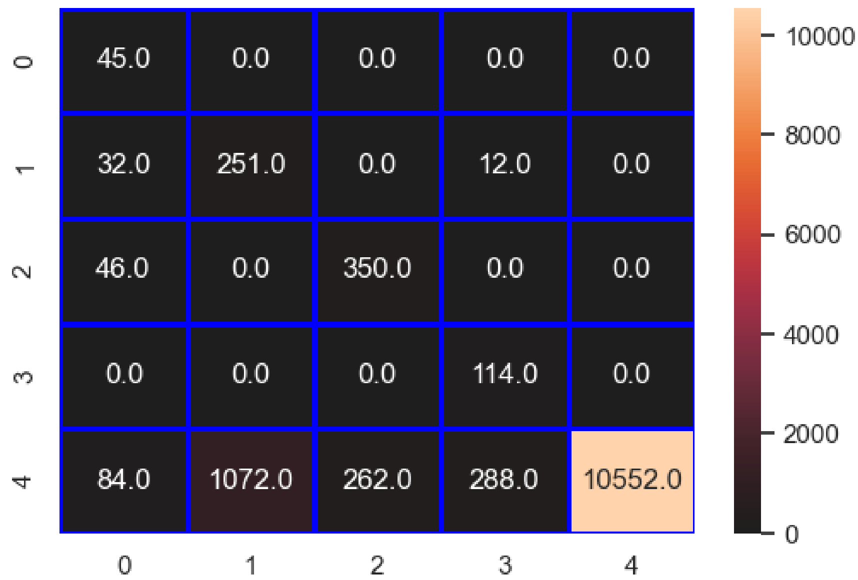
```
Modele name : LogisticRegression
10 K-Fold Accuracy_score : [0.9632 0.9689 0.9666 0.9653 0.9645 0.9645 0.9632 0.9636 0.9626 0.9687]
10 K-Fold Average Accuracy_score : 96.51 %
Accuracy_score: 96.47 %
Loss: 3.53 %
Cohen_kappa_score: 69.14 %
Classification_report:
               precision    recall  f1-score   support

           0       0.82      0.62      0.71        45
           1       0.71      0.46      0.56       295
           2       0.79      0.79      0.79       396
           3       0.66      0.53      0.59       114
           4       0.98      0.99      0.98     12258

    accuracy                           0.96     13108
   macro avg       0.79      0.68      0.72     13108
weighted avg       0.96      0.96      0.96     13108

confusion_matrix:
 [[   28     1     8     0     8]
 [    2   136    13     0   144]
 [    3     0   313     0    80]
 [    0     0     0    60    54]
 [    1    54    64    31 12108]]
```



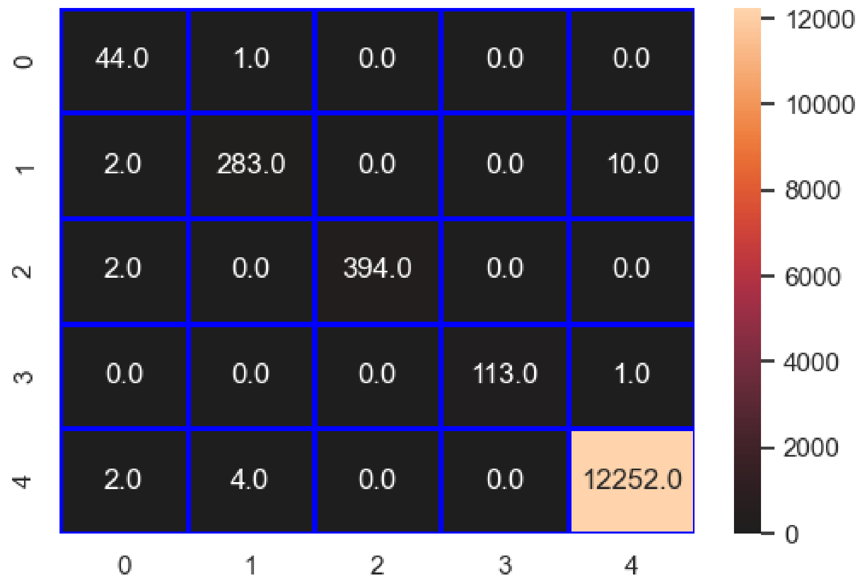
```
==========================================================================================


Modele name : KNeighborsClassifier
10 K-Fold Accuracy_score : [0.9811 0.9788 0.9817 0.9838 0.9784 0.9811 0.9847 0.9813 0.9798 0.9811]
10 K-Fold Average Accuracy_score : 98.12 %
Accuracy_score: 98.08 %
Loss: 1.92 %
Cohen_kappa_score: 82.29 %
Classification_report:
              precision    recall  f1-score   support

           0       0.96      0.56      0.70        45
           1       0.97      0.66      0.78       295
           2       0.96      0.82      0.88       396
           3       0.98      0.57      0.72       114
           4       0.98      1.00      0.99     12258

    accuracy                           0.98     13108
   macro avg       0.97      0.72      0.82     13108
weighted avg       0.98      0.98      0.98     13108

confusion_matrix:
 [[   25     4     8     0     8]
 [    1   194     0     0   100]
 [    0     0   323     0    73]
 [    0     0     0    65    49]
 [    0     3     5     1 12249]]
```



```
==========================================================================================


Modele name : GaussianNB
10 K-Fold Accuracy_score : [0.8583 0.86   0.8638 0.8627 0.8644 0.865  0.8674 0.8665 0.8581 0.8628]
10 K-Fold Average Accuracy_score : 86.29 %
Accuracy_score: 86.3 %
Loss: 13.7 %
Cohen_kappa_score: 43.66 %
Classification_report:
               precision    recall  f1-score   support

           0       0.22      1.00      0.36        45
           1       0.19      0.85      0.31       295
           2       0.57      0.88      0.69       396
           3       0.28      1.00      0.43       114
           4       1.00      0.86      0.93     12258

    accuracy                           0.86     13108
   macro avg       0.45      0.92      0.54     13108
weighted avg       0.96      0.86      0.90     13108

confusion_matrix:
 [[   45     0     0     0     0]
 [   32   251     0    12     0]
 [   46     0   350     0     0]
 [    0     0     0   114     0]
 [   84  1072   262   288 10552]]
```

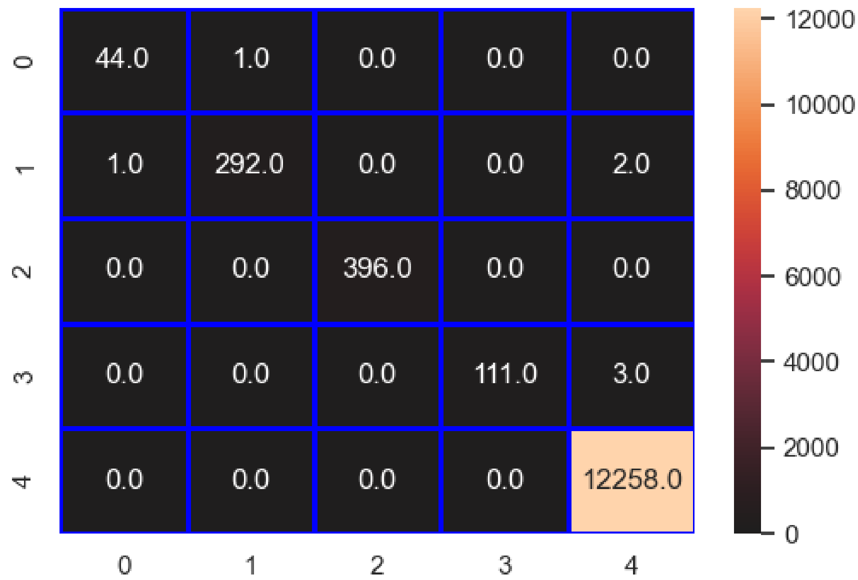


```
==========================================================================================

Modele name : DecisionTreeClassifier
10 K-Fold Accuracy_score : [0.9983 0.9968 0.9981 0.999  0.9973 0.9985 0.9992 0.9975 0.9987 0.9983]
10 K-Fold Average Accuracy_score : 99.82 %
Accuracy_score: 99.83 %
Loss: 0.17 %
Cohen_kappa_score: 98.64 %
Classification_report:
               precision    recall  f1-score   support

           0       0.88      0.98      0.93        45
           1       0.98      0.96      0.97       295
           2       1.00      0.99      1.00       396
           3       1.00      0.99      1.00       114
           4       1.00      1.00      1.00     12258

    accuracy                           1.00     13108
   macro avg       0.97      0.98      0.98     13108
weighted avg       1.00      1.00      1.00     13108

confusion_matrix:
 [[   44     1     0     0     0]
 [    2   283     0     0    10]
 [    2     0   394     0     0]
 [    0     0     0   113     1]
 [    2     4     0     0 12252]]
```

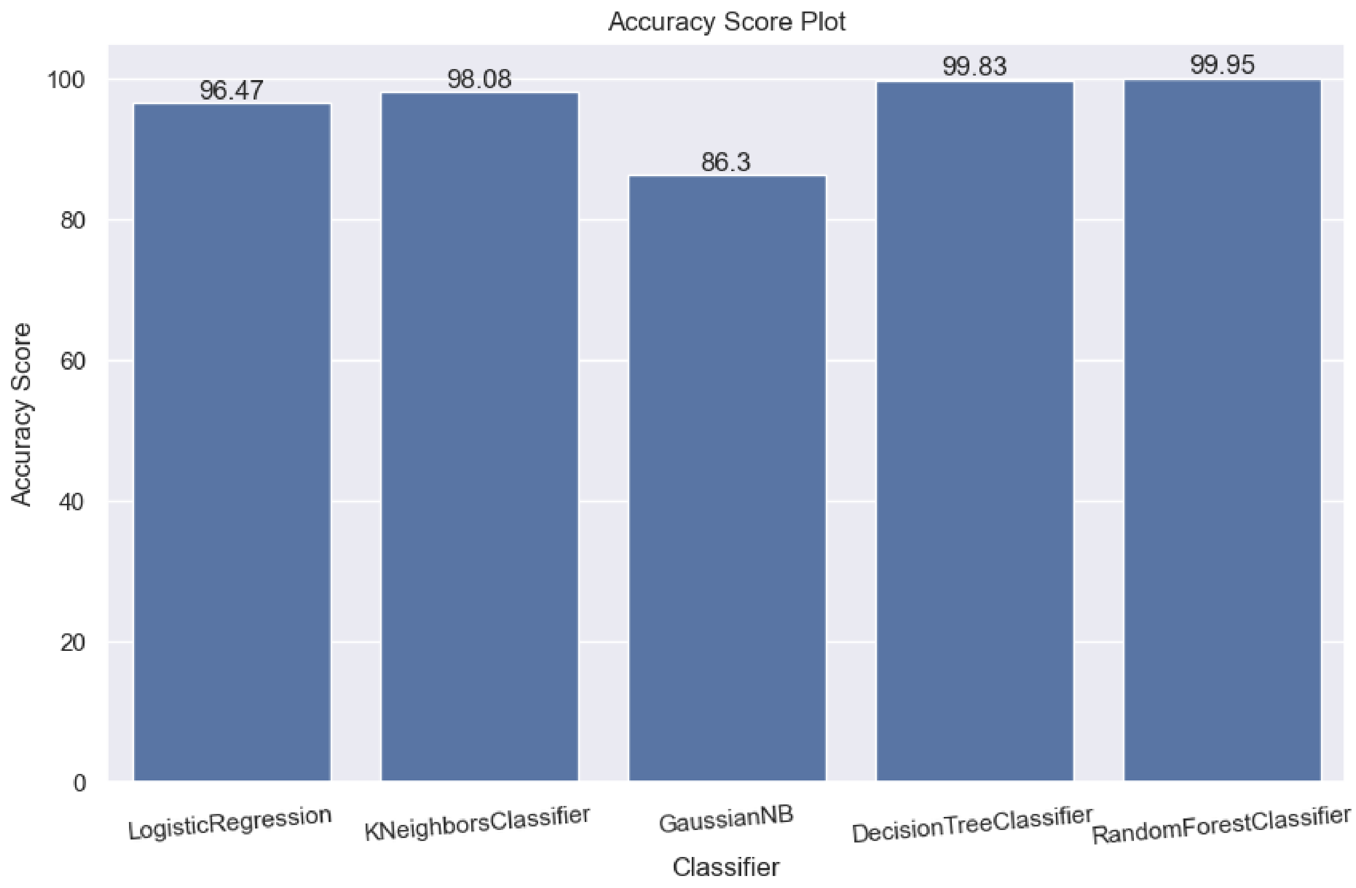


==

```
Modele name : RandomForestClassifier
10 K-Fold Accuracy_score : [0.9989 0.9989 0.9992 0.9996 0.9989 0.9985 0.9983 0.9992 0.9994 0.9987]
10 K-Fold Average Accuracy_score : 99.9 %
Accuracy_score: 99.95 %
Loss: 0.05 %
Cohen_kappa_score: 99.57 %
Classification_report:
               precision    recall  f1-score   support

           0       0.98      0.98      0.98        45
           1       1.00      0.99      0.99       295
           2       1.00      1.00      1.00       396
           3       1.00      0.97      0.99       114
           4       1.00      1.00      1.00     12258

    accuracy                           1.00     13108
   macro avg       0.99      0.99      0.99     13108
weighted avg       1.00      1.00      1.00     13108

confusion_matrix:
 [[   44     1     0     0     0]
 [    1   292     0     0     2]
 [    0     0   396     0     0]
 [    0     0     0   111     3]
 [    0     0     0     0 12258]]
```



==



Model Training - Under Agile data

In [132]:
```
data = pd.read_csv('dataset\dataset_under_agile_data_class.csv')
data.head()
```

Out[132]:

	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7	Q_8	Q_9	Q_10	Q_11	Q_12	Q_13	Q_14	Q_15	Q_16	under_agile_lable	class
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	111	7
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	111	7
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	111	7
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	111	7
4	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	111	7

In [133]:
```
data.shape
```

Out[133]: (65536, 18)

In [134]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 65536 entries, 0 to 65535
Data columns (total 18 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Q_1              65536 non-null  int64
 1   Q_2              65536 non-null  int64
 2   Q_3              65536 non-null  int64
 3   Q_4              65536 non-null  int64
 4   Q_5              65536 non-null  int64
 5   Q_6              65536 non-null  int64
 6   Q_7              65536 non-null  int64
 7   Q_8              65536 non-null  int64
 8   Q_9              65536 non-null  int64
 9   Q_10             65536 non-null  int64
 10  Q_11             65536 non-null  int64
 11  Q_12             65536 non-null  int64
 12  Q_13             65536 non-null  int64
 13  Q_14             65536 non-null  int64
 14  Q_15             65536 non-null  int64
 15  Q_16             65536 non-null  int64
 16  under_agile_lable  65536 non-null  int64
 17  class            65536 non-null  int64
dtypes: int64(18)
memory usage: 9.0 MB
```

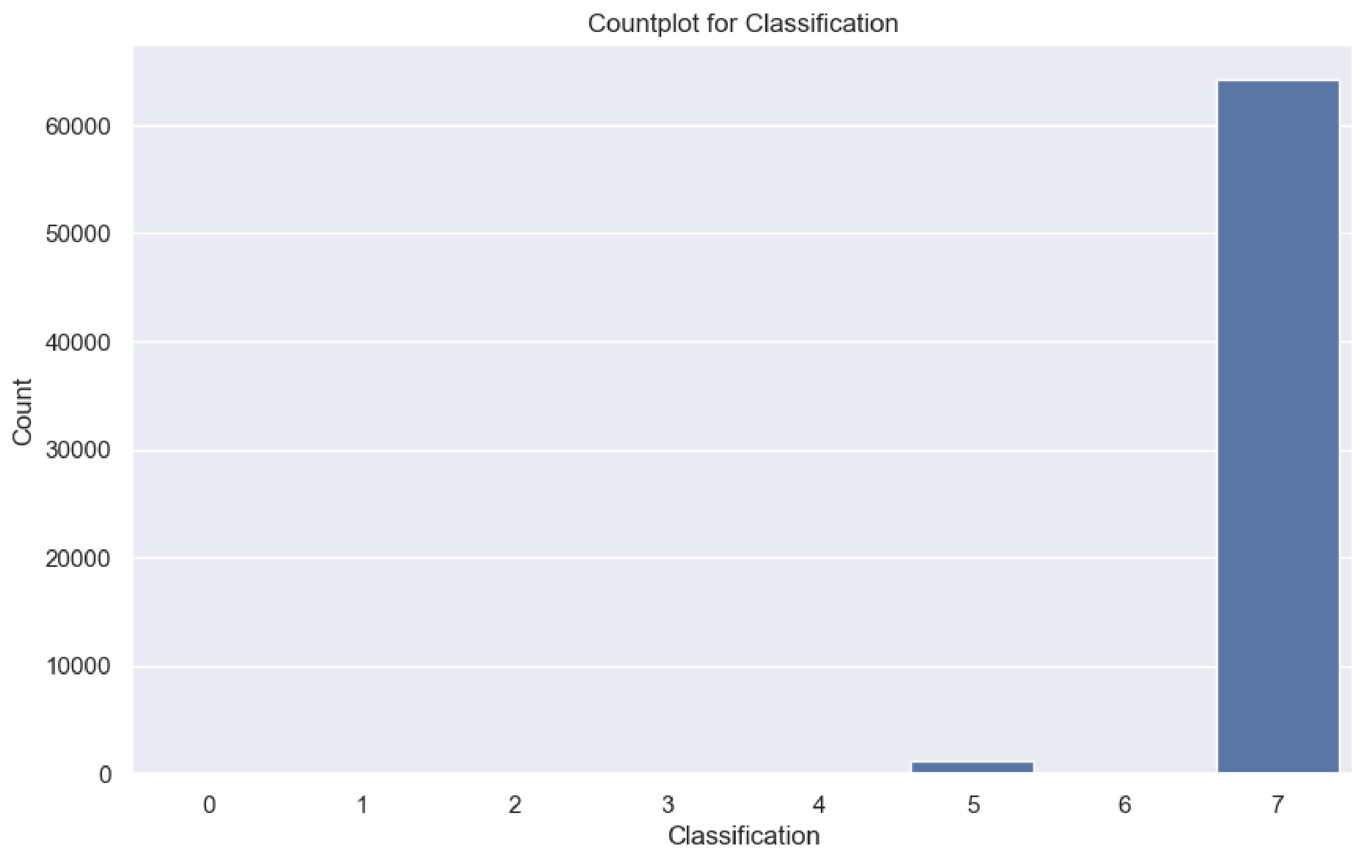
In [135]:

```
round(data.describe(),2)
```

Out[135]:

Q_2	Q_3	Q_4	Q_5	Q_6	Q_7	Q_8	Q_9	Q_10	Q_11	Q_12	Q_13	Q_14	Q_15	Q_16
5536.0	65536.0	65536.0	65536.0	65536.0	65536.0	65536.0	65536.0	65536.0	65536.0	65536.0	65536.0	65536.0	65536.0	65536.0
0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0

In [136]:
```
# Check If A Dataset Is Imbalanced
sns.countplot(x=data["class"])
plt.title('Countplot for Classification')
plt.xlabel('Classification')
plt.ylabel('Count')
plt.show()
```



In [137]:
```
# Drop columns
columns_to_drop = ['under_agile_lable']
data.drop(columns=columns_to_drop, inplace=True)
data = data.sample(frac=1).reset_index(drop=True)
data.head()
```

Out[137]:

	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7	Q_8	Q_9	Q_10	Q_11	Q_12	Q_13	Q_14	Q_15	Q_16	class
0	0	0	0	1	1	0	1	1	1	1	0	1	1	0	0	0	7
1	0	0	1	1	1	0	0	1	0	0	1	0	0	1	1	1	7
2	1	1	0	0	0	0	0	1	1	1	1	0	1	1	1	0	7
3	1	0	1	1	1	0	1	1	0	1	0	1	0	0	1	0	7
4	1	0	0	1	1	0	0	1	1	0	1	0	1	0	0	0	7

In [138]:
```
X, Y = data.drop(['class'], axis = 1),  data['class']
# train_test_split 80/20
X_train, X_test, y_train, y_test = train_test_split(X,Y, test_size = 0.2, random_state = 42, stratify = Y
```

In [139]:
```
# Model initialization
lr_Classifier = LogisticRegression()
knn_Classifier = KNeighborsClassifier()
gnb_Classifier = GaussianNB()
dt_Classifier = DecisionTreeClassifier()
rf_Classifier = RandomForestClassifier()
model_list = [lr_Classifier, knn_Classifier, gnb_Classifier, dt_Classifier, rf_Classifier]
```

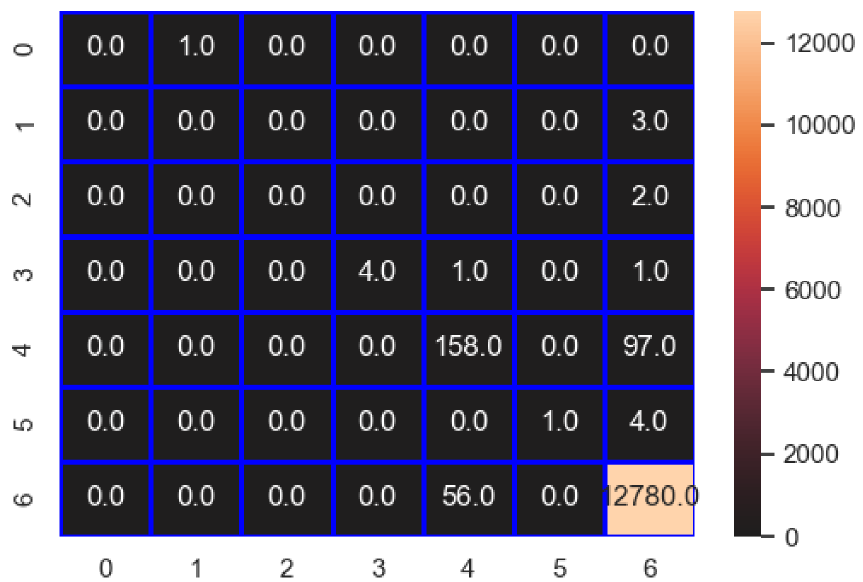
In [140]:

```
plot_data_list = []
def run_pipeline(X_train, X_test, y_train, y_test, classifier):
    # Model Information
    print(f"Modele name : {type(classifier).__name__}")

    # process 2 : train model
    classifier.fit(X_train, y_train)

    # process 4 : test model
    y_pred = classifier.predict(X_test)

    # process 5 : Perform k-fold cross-validation using cross_val_score
    scores = cross_val_score(classifier, X_train, y_train, cv=10, scoring='accuracy')
    print(f"10 K-Fold Accuracy_score : {np.round_(scores,4)}")
    print(f"10 K-Fold Average Accuracy_score : {round(np.average(scores)*100,2)} %")

    # process 6 : model evalution
    print("Accuracy_score:", round((accuracy_score(y_test, y_pred))*100,2),'%')
    print("Loss:", round((1-accuracy_score(y_test, y_pred))*100,2),'%')
    print("Cohen_kappa_score:", round((cohen_kappa_score(y_test, y_pred))*100,2),'%')
    print("Classification_report:\n",metrics.classification_report(y_test, y_pred))
    print("confusion_matrix:\n", confusion_matrix(y_test, y_pred))
    # plot confusion_matrix
    fig, ax = plt.subplots()
    fig.set_size_inches(6,4) # WH
    sns.heatmap(confusion_matrix(y_test, y_pred),
                annot=True,
                fmt=".1f",
                linewidths = 2,
                linecolor = "blue",
                center=0)
    plt.show()

    # process 7 : save model in pkl file
    filename = f'Models\\{str(type(classifier).__name__)}_acc{round((accuracy_score(y_test, y_pred))*100,
    pickle.dump(classifier, open(filename, 'wb'))

    # collect data for bar plot
    global plot_data_list
    plot_data_list.append([str(type(classifier).__name__),
                           round((accuracy_score(y_test, y_pred))*100,2)])

    # end
    print("==="*30)
    print("\n\n")
    time.sleep(0.5)
```

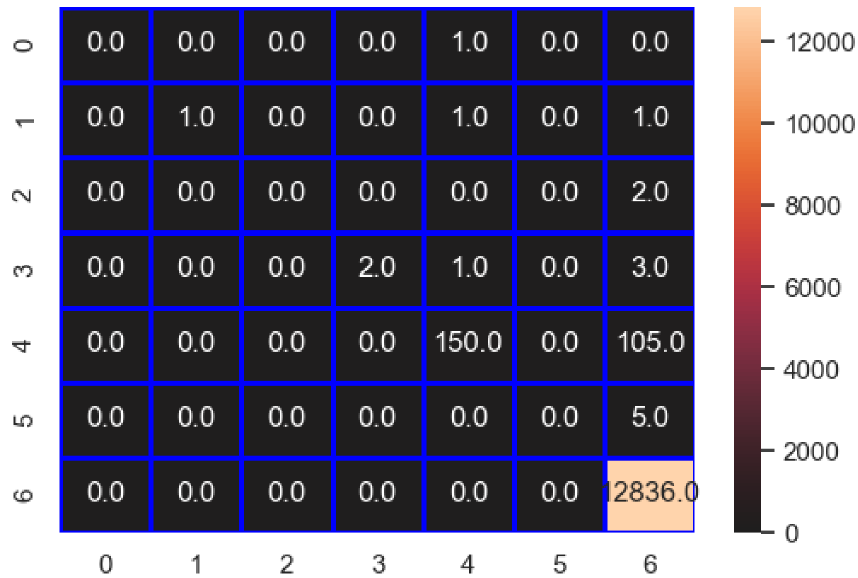
In [141]:

```
for model in model_list:
    # for scaler in scaler_list:
    run_pipeline(X_train, X_test, y_train, y_test, model)

# plot data
# plot_df = pd.DataFrame(plot_data_list, columns=['classifier', 'scaler', 'accuracy_score'])
plot_df = pd.DataFrame(plot_data_list, columns=['classifier', 'accuracy_score'])
plot_df.to_csv(f"dataset\\accuracy_score_plot_data_Secondary_classifier_Under_Agile_Prediction.csv", inde

sns.set(rc={'figure.figsize':(10,6)})
# ax = sns.barplot(data=plot_df, x="classifier", y="accuracy_score", hue="scaler")
ax = sns.barplot(data=plot_df, x="classifier", y="accuracy_score") # , hue="classifier"
plt.title('Accuracy Score Plot')
plt.xlabel('Classifier')
plt.ylabel('Accuracy Score')
ax.tick_params(axis='x', rotation=5)
for i in ax.containers:
    ax.bar_label(i,)
plt.show()
```

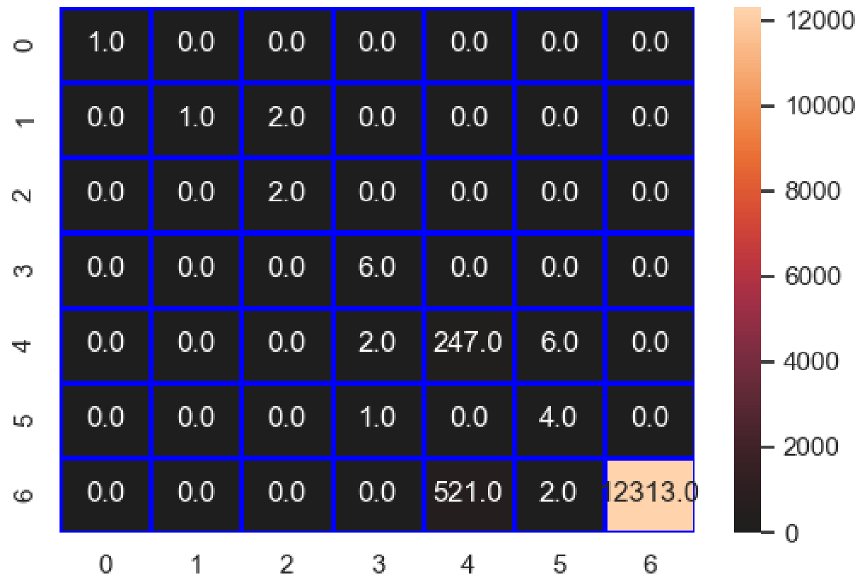
```
Modele name : LogisticRegression
10 K-Fold Accuracy_score : [0.9851 0.9861 0.988  0.9866 0.9886 0.9895 0.9878 0.9882 0.9882 0.9889]
10 K-Fold Average Accuracy_score : 98.77 %
Accuracy_score: 98.74 %
Loss: 1.26 %
Cohen_kappa_score: 65.93 %
Classification_report:
               precision    recall  f1-score   support

           0       0.00      0.00      0.00         1
           1       0.00      0.00      0.00         3
           3       0.00      0.00      0.00         2
           4       1.00      0.67      0.80         6
           5       0.73      0.62      0.67       255
           6       1.00      0.20      0.33         5
           7       0.99      1.00      0.99     12836

    accuracy                           0.99     13108
   macro avg       0.53      0.35      0.40     13108
weighted avg       0.99      0.99      0.99     13108

confusion_matrix:
 [[    0     1     0     0     0     0     0]
 [    0     0     0     0     0     0     3]
 [    0     0     0     0     0     0     2]
 [    0     0     0     4     1     0     1]
 [    0     0     0     0   158     0    97]
 [    0     0     0     0     0     1     4]
 [    0     0     0     0    56     0 12780]]
```



```
=========================================================================================

Modele name : KNeighborsClassifier
10 K-Fold Accuracy_score : [0.9922 0.9935 0.9907 0.9901 0.9937 0.9918 0.9924 0.991  0.9914 0.9928]
10 K-Fold Average Accuracy_score : 99.2 %
Accuracy_score: 99.09 %
Loss: 0.91 %
Cohen_kappa_score: 71.79 %
Classification_report:
               precision    recall  f1-score   support

           0       0.00      0.00      0.00         1
           1       1.00      0.33      0.50         3
           3       0.00      0.00      0.00         2
           4       1.00      0.33      0.50         6
           5       0.98      0.59      0.74       255
           6       0.00      0.00      0.00         5
           7       0.99      1.00      1.00     12836

    accuracy                           0.99     13108
   macro avg       0.57      0.32      0.39     13108
weighted avg       0.99      0.99      0.99     13108

confusion_matrix:
 [[    0     0     0     0     1     0     0]
 [    0     1     0     0     1     0     1]
 [    0     0     0     0     0     0     2]
 [    0     0     0     2     1     0     3]
 [    0     0     0     0   150     0   105]
 [    0     0     0     0     0     0     5]
 [    0     0     0     0     0     0 12836]]
```

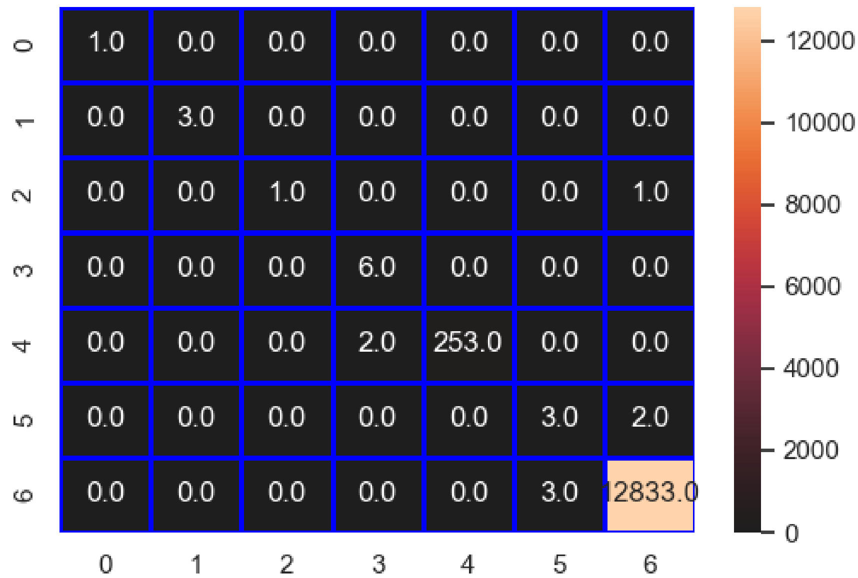


```
===========================================================================================


Modele name : GaussianNB
10 K-Fold Accuracy_score : [0.9573 0.9542 0.9542 0.9565 0.9575 0.9565 0.9554 0.9605 0.9567 0.9571]
10 K-Fold Average Accuracy_score : 95.66 %
Accuracy_score: 95.93 %
Loss: 4.07 %
Cohen_kappa_score: 48.43 %
Classification_report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         1
           1       1.00      0.33      0.50         3
           3       0.50      1.00      0.67         2
           4       0.67      1.00      0.80         6
           5       0.32      0.97      0.48       255
           6       0.33      0.80      0.47         5
           7       1.00      0.96      0.98     12836

    accuracy                           0.96     13108
   macro avg       0.69      0.87      0.70     13108
weighted avg       0.99      0.96      0.97     13108

confusion_matrix:
 [[    1     0     0     0     0     0     0]
 [    0     1     2     0     0     0     0]
 [    0     0     2     0     0     0     0]
 [    0     0     0     6     0     0     0]
 [    0     0     0     2   247     6     0]
 [    0     0     0     1     0     4     0]
 [    0     0     0     0   521     2 12313]]
```

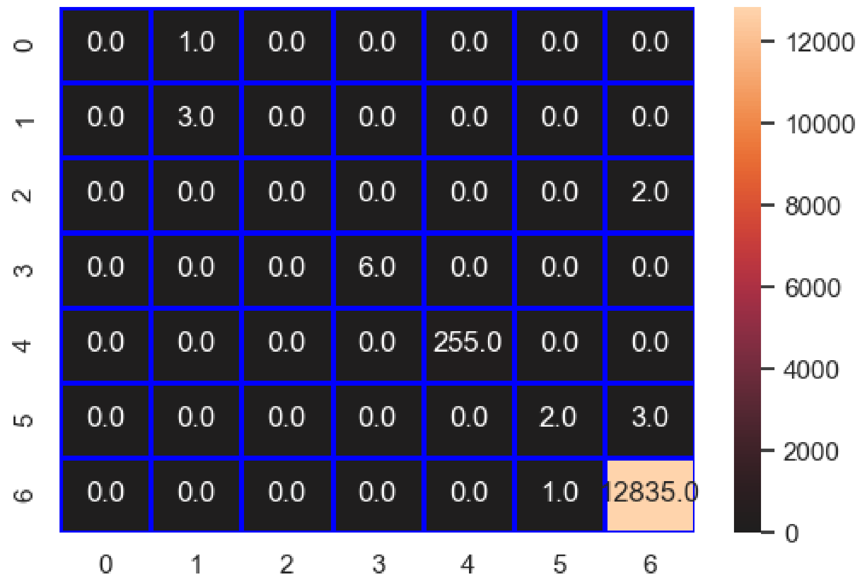


==

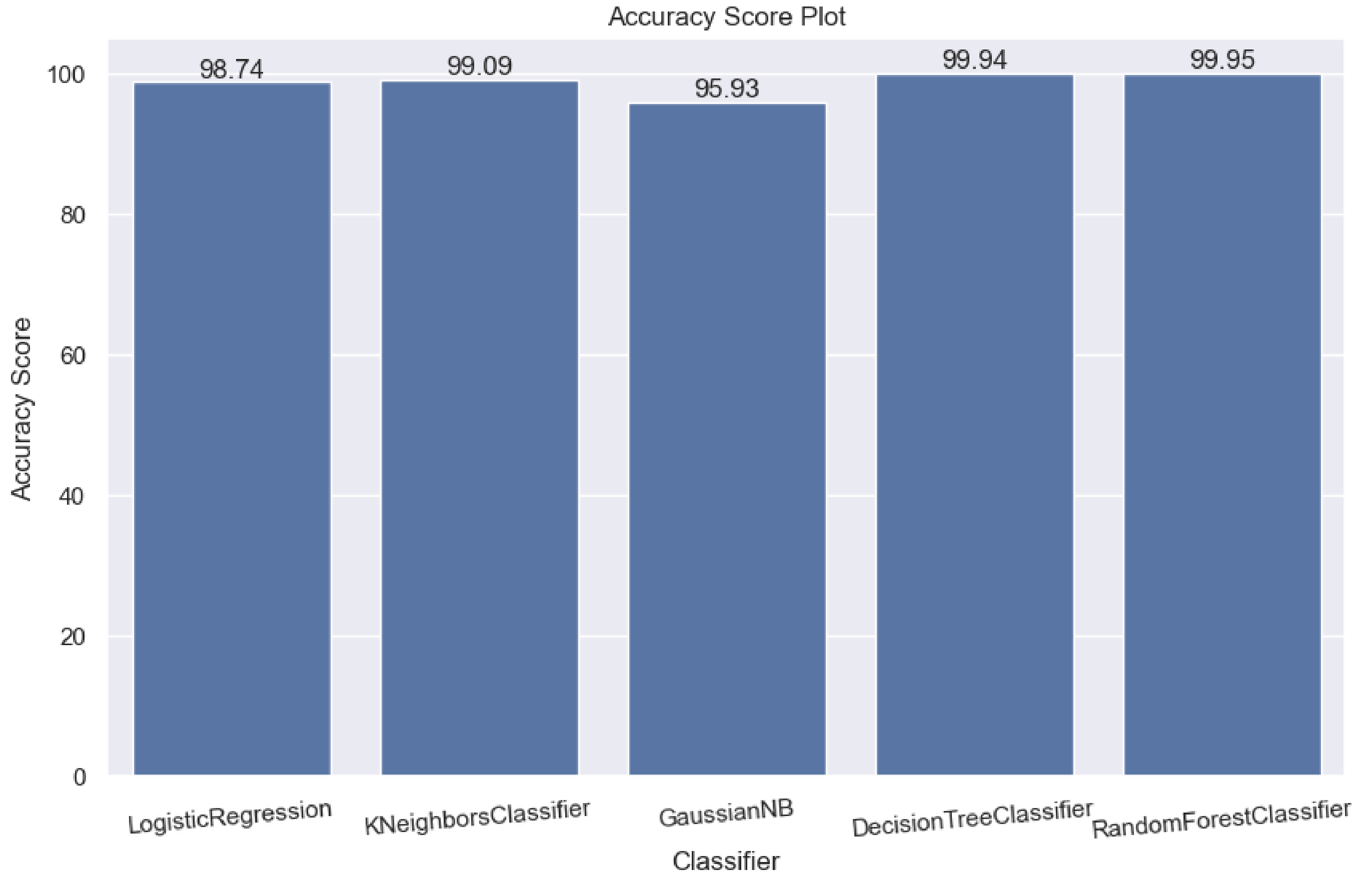
```
Modele name : DecisionTreeClassifier
10 K-Fold Accuracy_score : [0.9992 0.999  0.9994 0.9996 0.9994 0.999  0.9992 0.9994 0.9998 0.9994]
10 K-Fold Average Accuracy_score : 99.94 %
Accuracy_score: 99.94 %
Loss: 0.06 %
Cohen_kappa_score: 98.5 %
Classification_report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         1
           1       1.00      1.00      1.00         3
           3       1.00      0.50      0.67         2
           4       0.75      1.00      0.86         6
           5       1.00      0.99      1.00       255
           6       0.50      0.60      0.55         5
           7       1.00      1.00      1.00     12836

    accuracy                           1.00     13108
   macro avg       0.89      0.87      0.87     13108
weighted avg       1.00      1.00      1.00     13108

confusion_matrix:
 [[    1     0     0     0     0     0     0]
 [    0     3     0     0     0     0     0]
 [    0     0     1     0     0     0     1]
 [    0     0     0     6     0     0     0]
 [    0     0     0     2   253     0     0]
 [    0     0     0     0     0     3     2]
 [    0     0     0     0     0     3 12833]]
```



```
==========================================================================================


Modele name : RandomForestClassifier
10 K-Fold Accuracy_score : [0.9994 0.9996 0.9996 0.9989 0.9992 0.9996 0.9994 0.999  0.999  0.9992]
10 K-Fold Average Accuracy_score : 99.93 %
Accuracy_score: 99.95 %
Loss: 0.05 %
Cohen_kappa_score: 98.68 %
Classification_report:
               precision    recall  f1-score   support

           0       0.00      0.00      0.00         1
           1       0.75      1.00      0.86         3
           3       0.00      0.00      0.00         2
           4       1.00      1.00      1.00         6
           5       1.00      1.00      1.00       255
           6       0.67      0.40      0.50         5
           7       1.00      1.00      1.00     12836

    accuracy                           1.00     13108
   macro avg       0.63      0.63      0.62     13108
weighted avg       1.00      1.00      1.00     13108

confusion_matrix:
 [[    0     1     0     0     0     0     0]
 [    0     3     0     0     0     0     0]
 [    0     0     0     0     0     0     2]
 [    0     0     0     6     0     0     0]
 [    0     0     0     0   255     0     0]
 [    0     0     0     0     0     2     3]
 [    0     0     0     0     0     1 12835]]
```




```
==========================================================================================
```


Accuracy Score Plot
98.74
99.09
95.93
99.94
99.95
100
80
60
40
20
0
Accuracy Score
LogisticRegression
KNeighborsClassifier
GaussianNB
DecisionTreeClassifier
RandomForestClassifier
Classifier