CONTENTS

SOURCE

ATTRIBUTE INFORMATION

IMPORTING LIBRARIES

IMPORTING DATA

STATISTICAL SUMMARY

EXPLORATORY DATA ANALYSIS

DATA VISUALIZATION

Pandas Profiling Report

Overview

Dataset statistics

Number of variables	4
Number of observations	147269
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	4.5 MiB
Average record size in memory	32.0 B

Variable types

Categorical	2
Numeric	2

Variables

Select Columns

Name

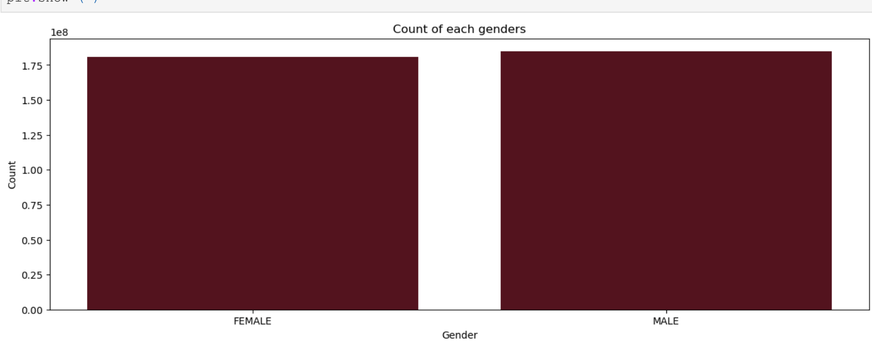
Categorical

INFERENCE

```
* Count is highly overall correlated with Probability

* Count is highly skewed (γ1 = 62.07429414)

* Probability is highly skewed (γ1 = 62.07429429)
```

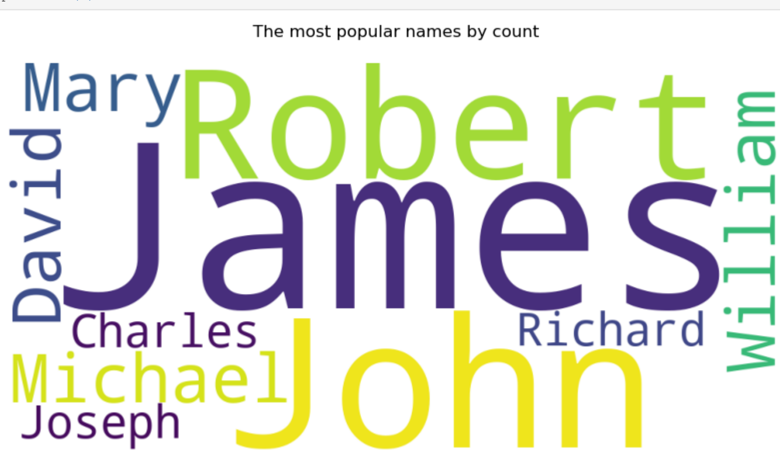
NAME AND GENDER

In [6]:
```
%matplotlib inline
# %matplotlib inline used to enable the inline plotting mode for Matplotlib, which means that the plots will be
# directly within the notebook interface, right below the code cell that generates them.
Count_DF = DF.groupby ( by = [ 'Gender' ] ).sum().reset_index ( )
plt.figure ( figsize = ( 15 , 5 ) )
plt.bar ( x = Count_DF.Gender , height = Count_DF.Count , color = '#53131E' )
plt.xlabel('Gender')
plt.ylabel('Count')
plt.title('Count of each genders')
plt.xticks ( ['F' , 'M'] , [ 'FEMALE' , 'MALE' ] )
plt.show ( )
```



MOST POPULAR NAMES

In [7]:
```
Popular_Names = WordCloud ( width = 1000 , height = 500 , background_color = 'white' ).generate_from_frequencie
plt.figure( figsize = ( 10 , 6 ) )
plt.axis ( 'off')
plt.title ( " The most popular names by count\n " )
plt.imshow ( Popular_Names ) ;
plt.show ( )
```



DATA PREPROCESSING

CHECKING FOR NULL VALUES

In [8]:
```
print ( 'The number of null values in each column of the data set is : ' )
DF.isnull ( ).sum ( )
```

Out[8]:
```
The number of null values in each column of the data set is :
Name           0
Gender         0
Count          0
Probability    0
dtype: int64
```

DATA TRANSFORMATION

In [9]:
```
LE = LabelEncoder ( )
DF.Name = LE.fit_transform ( DF.Name )
DF.Gender = LE.fit_transform ( DF.Gender )
DF
```

Out[9]:

	Name	Gender	Count	Probability
0	52308	1	5304407	1.451679e-02
1	57879	1	5260831	1.439753e-02
2	102035	1	4970386	1.360266e-02
3	84621	1	4579950	1.253414e-02
4	128418	1	4226608	1.156713e-02
...	...	...	...	...
147264	133797	1	1	2.736740e-09
147265	133823	1	1	2.736740e-09
147266	133843	1	1	2.736740e-09
147267	133892	1	1	2.736740e-09
147268	133902	1	1	2.736740e-09

147269 rows × 4 columns

In [10]:
```
print ( "The number of unique names in the data set is : " , DF.Name.nunique ( ) )
```
```
The number of unique names in the data set is :  133910
```

RANDOM SAMPLING

In [11]:
```
Train , Test = train_test_split ( DF , random_state = 42 )
Train_X = Train.iloc [ : , [ 0 , 2 , 3 ] ]
Train_Y = Train [ 'Gender' ]
Test_X = Test.iloc [ : , [ 0 , 2 , 3 ] ]
Test_Y = Test [ 'Gender' ]
print ( "The shape of the sampled train and test data is as follows : \n " )
print ( "\nTrain : " , Train.shape , "\tTest : " , Test.shape )
print ( "\nTrain_X : " , Train_X.shape , "\tTrain_Y : " , Train_Y.shape )
print ( "\nTest_X : " , Test_X.shape , "\tTest_Y : " , Test_Y.shape )
```
```
The shape of the sampled train and test data is as follows :


Train :  (110451, 4)    Test :  (36818, 4)

Train_X :  (110451, 3)  Train_Y :  (110451,)

Test_X :  (36818, 3)    Test_Y :  (36818,)
```

PREDICTIVE ANALYSIS

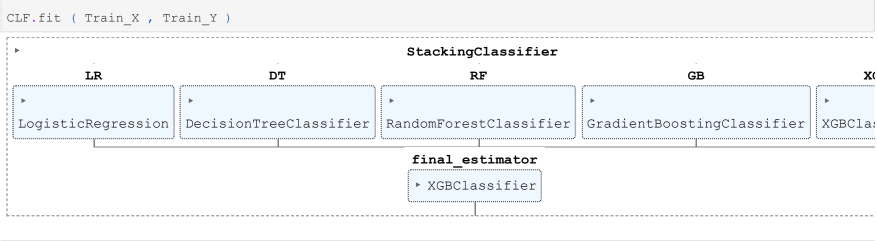
TUNING ALL THE ALGORITHMS

In [12]:
```
LR = LogisticRegression (  )
RF = RandomForestClassifier (  )
DT = DecisionTreeClassifier (  )
GB = GradientBoostingClassifier (  )
XGB = XGBClassifier (  )
Parameters_LR = { 'max_iter' : [ 100 , 500 , 700 ]  }
Parameters_RF = { 'n_estimators' : [ 100 , 150 ] ,
                  'criterion' : [ 'gini' , 'entropy' ] ,
                  'max_depth' : [ None , 5 , 10 ]
                }
Parameters_DT = {
                    'criterion' : [ 'gini' , 'entropy' ] ,
                    'max_depth' : [ None , 5 , 10 ] ,
                    'min_samples_split' : [ 2 , 5 , 7 ]
                }

Parameters_GB = {
                    'max_depth' : [ 3 , 5 , 10 ] ,
                    'learning_rate' : [ 0.1 , 0.01 ],
                    'n_estimators' : [ 150 , 200 ]
                }
Parameters_XGB =  {
                  'n_estimators': [ 200 , 300 ],
                  'learning_rate': [ 0.1, 0.01 ],
                  'max_depth': [ 3 , 5 , 7 ]
                }
Parameters = [ ( Parameters_LR , LR ) , ( Parameters_RF , RF ) , (Parameters_DT , DT ) , (Parameters_GB , GB )
Best_Parameters = [ ]
for i in Parameters :
    GS = GridSearchCV ( estimator = i [ 1 ] , param_grid = i [ 0 ] , cv = 5 )
    GS.fit ( Train_X , Train_Y )
    print ( "\nThe algorithm is : " , i [ 1 ] )
    print ( "\nThe best Train accuracy is : " , GS.best_score_ * 100 )
    print ( "\nThe best parameters are : " , GS.best_params_ )
    print ( "-----------------------------------------------------------------" )
    if i [ 1 ] == LR :
        LR = LogisticRegression ( **GS.best_params_ )
        Best_Parameters.append ( LR )
    elif i [ 1 ] == RF :
        RF = RandomForestClassifier ( **GS.best_params_ )
        Best_Parameters.append ( RF )
    elif i [ 1 ] == DT :
        DT = DecisionTreeClassifier ( **GS.best_params_ )
        Best_Parameters.append ( DT )
    elif i [ 1 ] == GB :
        GB = GradientBoostingClassifier ( **GS.best_params_ )
        Best_Parameters.append ( GB )
    elif i [ 1 ] == XGB :
        XGB = XGBClassifier ( **GS.best_params_ )
        Best_Parameters.append ( XGB )
```
```
The algorithm is :  LogisticRegression()

The best Train accuracy is :  60.87767436472442

The best parameters are :  {'max_iter': 100}
-----------------------------------------------------------------

The algorithm is :  RandomForestClassifier()

The best Train accuracy is :  61.36929383764866

The best parameters are :  {'criterion': 'gini', 'max_depth': 10, 'n_estimators': 100}
-----------------------------------------------------------------

The algorithm is :  DecisionTreeClassifier()

The best Train accuracy is :  62.12347289194481

The best parameters are :  {'criterion': 'entropy', 'max_depth': 10, 'min_samples_split': 2}
-----------------------------------------------------------------

The algorithm is :  GradientBoostingClassifier()

The best Train accuracy is :  64.00304129064563

The best parameters are :  {'learning_rate': 0.1, 'max_depth': 10, 'n_estimators': 150}
-----------------------------------------------------------------

The algorithm is :  XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              n_estimators=100, n_jobs=None, num_parallel_tree=None,
              predictor=None, random_state=None, ...)

The best Train accuracy is :  64.26016936938834

The best parameters are :  {'learning_rate': 0.1, 'max_depth': 7, 'n_estimators': 300}
-----------------------------------------------------------------
```

STACKING CLASSIFIER

In [18]:
```
Best_Parameters = [ ( 'LR' , LR ) , ( 'DT' , DT ) , ( 'RF' , RF ) , ( 'GB' , GB ) , ( 'XGB' , XGB ) ]
CLF = StackingClassifier ( estimators = Best_Parameters , final_estimator = XGB )
CLF.fit ( Train_X , Train_Y )
```

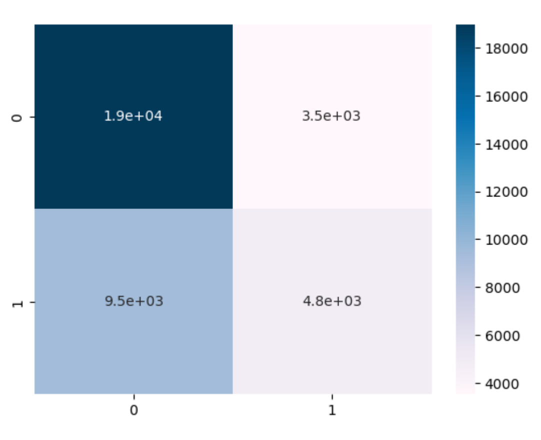
Out[18]:



In [19]:
```
Y_CLF = CLF.predict ( Test_X )
Accuracy_CLF = accuracy_score( Test_Y , Y_CLF ) * 100
Precision_CLF = precision_score ( Test_Y , Y_CLF ) * 100
Recall_CLF = recall_score ( Test_Y , Y_CLF ) * 100
F1_Score_CLF = f1_score ( Test_Y , Y_CLF ) * 100
print ( "The performance metrics of the Test Data is : \n" )
print ( "Accuracy = {}%\n\nPrecision = {}%\n\nRecall = {}%\n\nF1 Score = {}%".format ( Accuracy_CLF , Precision
```
```
The performance metrics of the Test Data is :

Accuracy = 64.64501059264491%

Precision = 57.74378585086042%

Recall = 33.75951931810243%

F1 Score = 42.608350601825315%
```

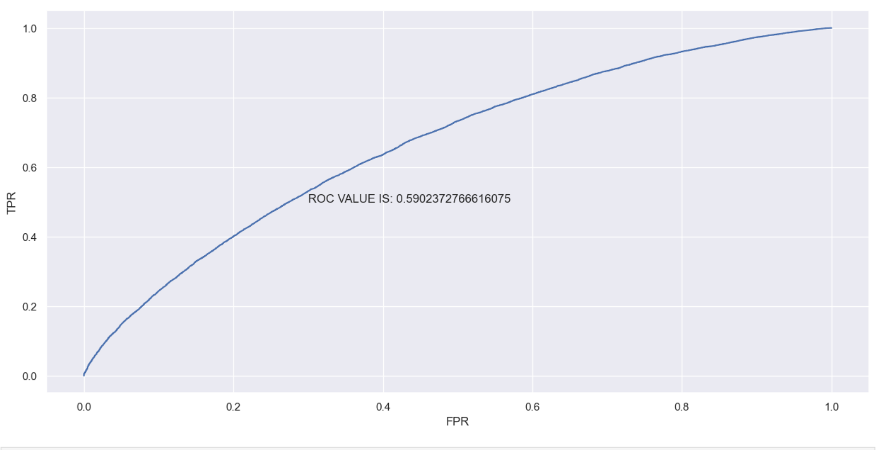
In [20]:
```
sns.heatmap ( confusion_matrix ( Test_Y , Y_CLF ) , cmap = 'PuBu' , annot = True )
```

Out[20]: `<AxesSubplot:>`



ROC CURVE

In [25]:
```
Predicted_Prob = CLF.predict_proba ( Test_X )
FPR, TPR, Threshold = roc_curve( Test_Y, Predicted_Prob [ : , 1 ] )
AUROC_Score = roc_auc_score( Test_Y , Y_CLF )
plt.style.use( "seaborn-dark" )
plt.figure ( figsize = ( 15 , 7 ) )
sns.set( style = 'darkgrid' )
plt.plot( FPR , TPR )
plt.xlabel( "FPR" )
plt.ylabel( "TPR" )
plt.text( x = 0.3, y = 0.5, s = "ROC VALUE IS: {}".format ( AUROC_Score ) )
plt.show ( )
```



In [ ]: