Gender prediction for a name

Approach:

- As we have to predict the gender i.e. male or female for a name this will fall into a classification task.
- We will prepare a balanced dataset of indian names [both male and female] and we will train our machine learning classification algorithm on that. [I have downloaded this dataset from github and made some modifications]
- After the algorithm is trained we will evaluate the model with test data which we will keep aside from the main dataset for evaluation purpose.
- Once evaluation is ready we will write a function which will take a name and output with prediction i.e. male or female

In [93]:
```
# Import pandas and numpy
import pandas as pd
import numpy as np
```

In [94]:
```
# Load the data
data=pd.read_csv('gender_pred_dataset.csv')
```

In [95]:
```
# Check numerical statictics of data using pandas 'info' method
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8228 entries, 0 to 8227
Data columns (total 2 columns):
Names     8228 non-null object
Gender    8228 non-null object
dtypes: object(2)
memory usage: 128.6+ KB
```

In [96]:
```
# Check 1st two rows using pandas 'head' method
data.head(2)
```

Out[96]:

	Names	Gender
0	Babita	F
1	Bageshri	F

In [97]:
```
# Check the distribution on target class
data['Gender'].value_counts()
# Female: 56.51%, male= 43.48%
```

Out[97]:
```
M    4650
F    3578
Name: Gender, dtype: int64
```

data cleaning

- missing values: None # check data.info() data
- categorical variable: sex column has categorical variable which we have convert to numerical

In [98]:
```
# convert categorical values to numerical
data['Gender_label']=data['Gender'].map({'F':0, 'M':1})
#data['Gender_label']=data['Gender'].replace({'F':0, 'M':1})
```

In [99]:
```
# Check the target lables in newly created column
data['Gender_label'].value_counts()
```

Out[99]:
```
1    4650
0    3578
Name: Gender_label, dtype: int64
```

In [100]:
```
# Prepare feature and target variable
X=data['Names']
y=data['Gender_label']
```

In [101]:
```
# split X and y into training and testing sets using scikit learn train_test_spli
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test=train_test_split(X, y, test_size=.33, random_sta
```

In [102]:
```
# check shape of training and test varible
X_train.shape, X_test.shape
```

Out[102]:
```
((5512,), (2716,))
```

In [92]:
```
# CountVectorizer: Convert a collection of text documents to a matrix of token co
# This implementation produces a sparse representation of the counts using scipy.

from sklearn.feature_extraction.text import CountVectorizer # # Import CountVecto
cv=CountVectorizer() # Create an instant of CountVectorizer
```

In [104]:
```
# learn training data vocabulary by fit method, then use it to create a document-
cv.fit(X_train)
X_train_cv=cv.transform(X_train).toarray() # use learned vocabulary to create a c
# To store all zero and non zero values we have to convert sparse matrix into a D
```

In [105]:
```
# transform testing data (using fitted vocabulary) into a document-term matrix
X_test_cv=cv.transform(X_test).toarray()
```

In [107]:
```
# import classificationalgorithms from scikit learn
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import MultinomialNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import confusion_matrix, accuracy_score
```

In [108]:
```
# Create a list of all classification algorithms
models=[]
models.append(('LoR', LogisticRegression()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('SVM', SVC()))
models.append(('NB', MultinomialNB()))
models.append(('DT', DecisionTreeClassifier()))
models.append(('RF', RandomForestClassifier()))
```

In [109]:
```
# check accuracy of all algorithms with test set
results=[]
names=[]
for name, model in models:
    model.fit(X_train_cv, y_train)
    y_pred=model.predict(X_test_cv)
    accuracy=accuracy_score(y_test,y_pred)
    results.append(accuracy)
    names.append(name)
    print(name,accuracy)
```

```
LoR 0.6226067746686303
KNN 0.47017673048600883
SVM 0.5651693667157585
NB 0.6226067746686303
DT 0.6226067746686303
RF 0.6145066273932254
```

- Above accuracy can be increased further by enhancing training set and tuning hyper parameter
- Let's use multinomial Naive Bayes classifier as it is suitable for classification with discrete features (e.g., word counts for text classification). The multinomial distribution normally requires integer feature counts. However, in practice, fractional counts such as tf-idf may also work.

In [116]:
```
nb=MultinomialNB()
nb.fit(X_train_cv, y_train)
```

Out[116]:
```
MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
```

In [117]:
```
# sample prediction
sample=['jay']
sample_cv=cv.transform(sample).toarray()
nb.predict(sample_cv)
```

Out[117]:
```
array([1], dtype=int64)
```

In [120]:
```
# lets define a function to do it
def genderpred(a):
    name=[a]
    name_cv=cv.transform(name).toarray()
    if nb.predict(name_cv)==0:
        print('Female')
    else:
        print('Male')
```

In [121]:
```
genderpred('Raju')
```

```
Male
```

In [122]:
```
genderpred('Priya')
```

```
Female
```

In [123]:
```
genderpred('Jay')
```

```
Male
```

In [124]:
```
genderpred('jaya')
```

```
Female
```

In [91]:
```
# for a list of names
name_list=["raju", "priya", "jay", "jaya"]
for name in name_list:
    print(genderpred(name))
```

```
Male
None
Female
None
Male
None
Female
None
```