In [1]:

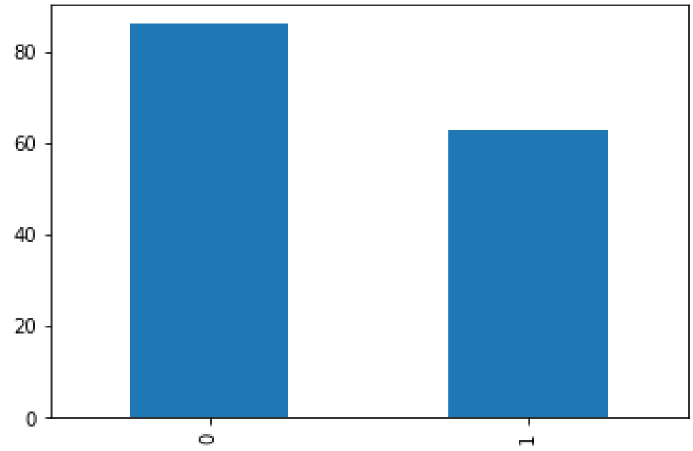
```
import pandas as pd
import numpy as np
import os
import time
os.chdir(r"C:\Users\Angelina\Downloads\Patient-name-deduplication-master")
from sklearn.metrics import f1_score, accuracy_score
import matplotlib.pyplot as plt
```

In [2]:

```
data = pd.read_csv('input.csv')
data['is_duplicate'].value_counts().plot(kind='bar')
```

Out[2]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x183256cebe0>
```



In [3]:

```
data.head()
```

Out[3]:

	ln	dob	gn	fn	is_duplicate
0	SMITH JR	01-03-1968	F	WILLIAM	0
1	ROTHMEYER JR	01-03-1968	F	WILLIAM	0
2	BLAND III	21-02-1962	F	WILLIAM	1
3	BLAND JR	21-02-1962	F	BILL	0
4	BLAND	21-02-1962	F	WILLIAM	1

In [4]:

```
#### The dob is converted to standard datetime format.

data.dob = pd.to_datetime(data.dob)
```

In [5]:

```
data.head()
```

Out[5]:

	ln	dob	gn	fn	is_duplicate
0	SMITH JR	1968-01-03	F	WILLIAM	0
1	ROTHMEYER JR	1968-01-03	F	WILLIAM	0
2	BLAND III	1962-02-21	F	WILLIAM	1
3	BLAND JR	1962-02-21	F	BILL	0
4	BLAND	1962-02-21	F	WILLIAM	1

In [6]:

```
data.dob.head(10)
```

Out[6]:

```
0   1968-01-03
1   1968-01-03
2   1962-02-21
3   1962-02-21
4   1962-02-21
5   1962-02-21
6   1954-08-06
7   1954-08-06
8   1953-10-25
9   1953-10-25
Name: dob, dtype: datetime64[ns]
```

In [7]:

```
data['name'] = data.fn + ' ' + data.ln
```

In [8]:

```
import hashlib
import base64
data = data.assign(concat  = data.dob.astype(str) + data.gn + data.fn + data.ln)
data['hash']=data['concat'].astype(str).str.encode('UTF-8').apply(lambda x: base
64.b64encode(hashlib.md5(x).digest()))
data
#data2=data
data.head()
```

Out[8]:

	ln	dob	gn	fn	is_duplicate	name	concat	
0	SMITH JR	1968-01-03	F	WILLIAM	0	WILLIAM SMITH JR	1968-01-03FWILLIAMSMITH JR	b'wKkl
1	ROTHMEYER JR	1968-01-03	F	WILLIAM	0	WILLIAM ROTHMEYER JR	1968-01-03FWILLIAMROTHMEYER JR	b'N2h
2	BLAND III	1962-02-21	F	WILLIAM	1	WILLIAM BLAND III	1962-02-21FWILLIAMBLAND III	b'LTt/
3	BLAND JR	1962-02-21	F	BILL	0	BILL BLAND JR	1962-02-21FBILLBLAND JR	b'pQf0T
4	BLAND	1962-02-21	F	WILLIAM	1	WILLIAM BLAND	1962-02-21FWILLIAMBLAND	b'XhFtQ

In [9]:

```
#### A list of unique dates of birth and unique genders is obtained.

unique_dob = data.dob.unique()
unique_sex = data.gn.unique()
unique_hash = data.hash.unique()
```

In [10]:

```
import distance
```

In [11]:

```
import time
start_h = time.time()
def deduplication_model(data, scoring_range = 10, step = 2):
    data['indices'] = list(range(len(data)))
    accuracy = []
    index = []
    final_step = 0
    for value in range(scoring_range):
        for i in unique_hash:
                sample = data[(data.hash == i)].reset_index(drop = True)
                for a in range(len(sample)):
                    comparison = sample[(sample.indices != sample.indices[a])].r
eset_index(drop = True)
                    scores = [distance.levenshtein(sample.name[a], comparison.na
me[x]) for x in range(len(comparison))]
                    compare = [comparison.indices[x] for x in range(len(comparis
on))]
                    try:
                        if sample.indices[a]>compare[scores.index(min(scores))]:
                            score = np.min(scores)
                            if score<=value:
                                index.append(sample.indices[a])
                    except ValueError:
                        pass
        prediction = []
        for k in range(len(data)):
            if data.indices[k] in index:
                prediction.append(1)
            else:
                prediction.append(0)

        data['prediction'] = prediction
        print('F1-score after ',value, 'iterations : ', f1_score(data.is_duplica
te, data.prediction, average = 'macro'))
        accuracy.append(f1_score(data.is_duplicate, data.prediction, average = 
'macro'))
        if len(accuracy)>1 and accuracy[-1] <= accuracy[-2]:
            final_step+=1
        if final_step>=step:
            value = value-final_step
            break

    index = []
    for i in unique_hash:
            sample = data[(data.hash == i)].reset_index(drop = True)
            for a in range(len(sample)):
                comparison = sample[(sample.indices != sample.indices[a])].reset
_index(drop = True)
                scores = [distance.levenshtein(sample.name[a], comparison.name[x
]) for x in range(len(comparison))]
                compare = [comparison.indices[x] for x in range(len(comparison
))]
                try:
                    if sample.indices[a]>compare[scores.index(min(scores))]:
                        score = np.min(scores)
                        if score<=value:
                            index.append(sample.indices[a])
                except ValueError:
                    pass
    prediction = []
    for k in range(len(data)):
        if data.indices[k] in index:
            prediction.append(1)
        else:
            prediction.append(0)
    return prediction, value
```

In [12]:

```
from sklearn.model_selection import train_test_split
train, test = train_test_split(data, test_size = 0.05, stratify = data.is_duplicate, random_state = 0)
train = train.reset_index(drop = True)
test = test.reset_index(drop = True)
performance, levenshtein_value_optimum = deduplication_model(train, scoring_range = 10, step = 3)
```

```
F1-score after  0 iterations :  0.6611481975967958
F1-score after  1 iterations :  0.6611481975967958
F1-score after  2 iterations :  0.6611481975967958
F1-score after  3 iterations :  0.6611481975967958
```

In [13]:

```
def deduplication_prediction(data, optimum_value):
    data['indices'] = list(range(len(data)))
    index = []
    for i in unique_hash:
            sample = data[(data.hash == i)].reset_index(drop = True)
            for a in range(len(sample)):
                comparison = sample[(sample.indices != sample.indices[a])].reset_index(drop = True)
                scores = [distance.levenshtein(sample.name[a], comparison.name[x]) for x in range(len(comparison))]
                compare = [comparison.indices[x] for x in range(len(comparison))]
                try:
                    if sample.indices[a]>compare[scores.index(min(scores))]:
                        score = np.min(scores)
                        if score<=optimum_value:
                            index.append(sample.indices[a])
                except ValueError:
                    pass
    prediction = []
    for k in range(len(data)):
        if data.indices[k] in index:
            prediction.append(1)
        else:
            prediction.append(0)
    return prediction
```

In [14]:

```
predictions = deduplication_prediction(test, levenshtein_value_optimum)
```

In [15]:

```
print('F1-score on test set:',accuracy_score(test.is_duplicate, predictions))
```

```
F1-score on test set: 0.625
```

In [16]:

```
train['prediction'] = performance
test['prediction'] = predictions
dataset = pd.concat([train, test], axis = 0)
dataset = dataset[(dataset.prediction != 1)].reset_index(drop = True).drop(labels = ['name', 'is_duplicate', 'prediction', 'indices'], axis = 1)
```

In [17]:

```
dataset.to_csv('11Deduplicated.csv', index = False)
end_h = time.time()
tt2 = end_h - start_h
print('Time taken: ')
tt2
```

```
Time taken: 
```

Out[17]:

```
1.6901249885559082
```

In [ ]:

```
```