AIM - To create a ML model to identify FM stations

Python version used -> python 3

Step - 1

Run RTL_power command

Install GQRX software then connect RTL_SDR dongle and open terminal.

Note : Following command only works for Linux and Mac OS.

COMMAND -> rtl_power -f min:max:bin -g gain -i interval -e runtime filename.ext where min is initial frequency max is terminal frequency bin is frequency interval interval in seconds

COMMAND I USED -

rtl_power -f 87M:108M:1k -g 20 -i 10 -e 5m logfile.csv

All the data is stored in a csv file logfile.csv.

Step - 2

Data cleaning

We will now convert obtained csv into a desireable pandas dataframe

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
dfs = pd.read_csv("logfile.csv", header=None)
```

Raw data collected from GQRX stored in df dataframe

In [2]:

```
dfs.head()
```

Out[2]:

	0	1	2	3	4	5	6	7	8	9	...	4093
0	2018-03-30	22:31:10	87000000	89625000	640.87	40	-45.44	-50.61	-52.59	-52.59	...	-53.91
1	2018-03-30	22:31:10	89625000	92250000	640.87	40	-55.90	-57.27	-57.36	-56.05	...	-56.50
2	2018-03-30	22:31:10	92250000	94875000	640.87	40	-40.56	-41.09	-40.24	-41.16	...	-41.91
3	2018-03-30	22:31:10	94875000	97500000	640.87	40	-41.38	-40.05	-39.69	-40.90	...	-44.02
4	2018-03-30	22:31:10	97500000	100125000	640.87	40	-52.95	-55.07	-54.32	-56.54	...	-52.79

5 rows × 4103 columns

In [3]:

```
dfs.describe()
```

Out[3]:

	2	3	4	5	6	7	
count	2.400000e+02	2.400000e+02	2.400000e+02	240.000000	240.000000	240.000000	240.0000
mean	9.618750e+07	9.881250e+07	6.408700e+02	40.400000	-49.489583	-50.506667	-50.1732
std	6.027200e+06	6.027200e+06	2.164564e-12	1.085469	7.330919	7.642566	7.4737
min	8.700000e+07	8.962500e+07	6.408700e+02	38.000000	-58.010000	-58.230000	-58.1700
25%	9.159375e+07	9.421875e+07	6.408700e+02	40.000000	-55.312500	-56.057500	-55.6450
50%	9.618750e+07	9.881250e+07	6.408700e+02	40.000000	-52.780000	-54.590000	-52.8600
75%	1.007812e+08	1.034062e+08	6.408700e+02	42.000000	-44.037500	-45.455000	-47.3600
max	1.053750e+08	1.080000e+08	6.408700e+02	42.000000	-34.140000	-34.750000	-33.8700

8 rows × 4101 columns

Here we'll convert raw data into a useful data which can be used for training/test

The raw data is being converted to a dataframe having 3 coloumns - Frequency captured, corresponding power and Is Fm station or not.

In [4]:

```
def conversion_function(df):
    x=[] # Stores all the frequencies
    y=[] # Stores corresponding power value
    z=[] # Stores is_FM, if 1 then yes, if 0 then no

    # following array is obtained from "https://en.wikipedia.org/wiki/Category:Lists_of_rad
    arr = [76.1, 76.5, 77.1, 78.8, 80.0, 80.2, 81.3, 82.5, 84.7, 87.5, 87.6, 87.7, 87.8, 87

    for j in range(0,len(df)):
        for i in range(6,4103):
            y.append(float(df[i][j]))
            r = (df[3][j]-df[2][j])/4096
            temp = df[3][j]+(r*(i-6))
            x.append(float(temp))
            check = round((temp/1000000),1)
            n=0
            for k in arr:
                if(check == k):
                    n=1
            z.append(int(n))
    df = pd.DataFrame({"Frequency":x,"Power":y,"Is_FM":z})
    return df
```

Below dataframe contains 3 coloumns - Frequency captured, power and Is_FM(1 means yes/0 means no)

In [5]:

```
dfs = conversion_function(dfs)
dfs.head()
```

Out[5]:

	Frequency	Is_FM	Power
0	8.962500e+07	1	-45.44
1	8.962564e+07	1	-50.61
2	8.962628e+07	1	-52.59
3	8.962692e+07	1	-52.59
4	8.962756e+07	1	-53.53

In [6]:

```
dfs.isnull().sum()
```

Out[6]:

```
Frequency    0
Is_FM        0
Power        0
dtype: int64
```

In [7]:

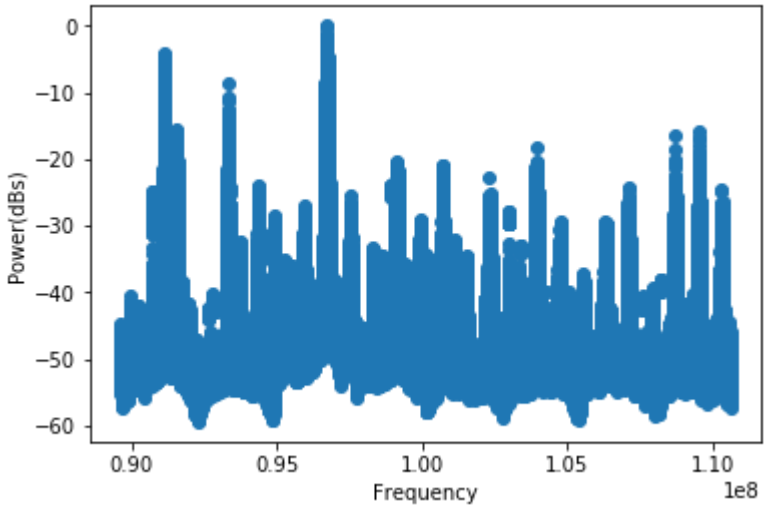
```
dfs.describe()
```

Out[7]:

	Frequency	Is_FM	Power
count	9.832800e+05	983280.000000	983280.000000
mean	1.001250e+08	0.577435	-47.791829
std	6.062204e+06	0.493968	7.172532
min	8.962500e+07	0.000000	-59.420000
25%	9.487500e+07	0.000000	-51.920000
50%	1.001250e+08	1.000000	-50.200000
75%	1.053750e+08	1.000000	-46.330000
max	1.106250e+08	1.000000	0.010000

In [8]:

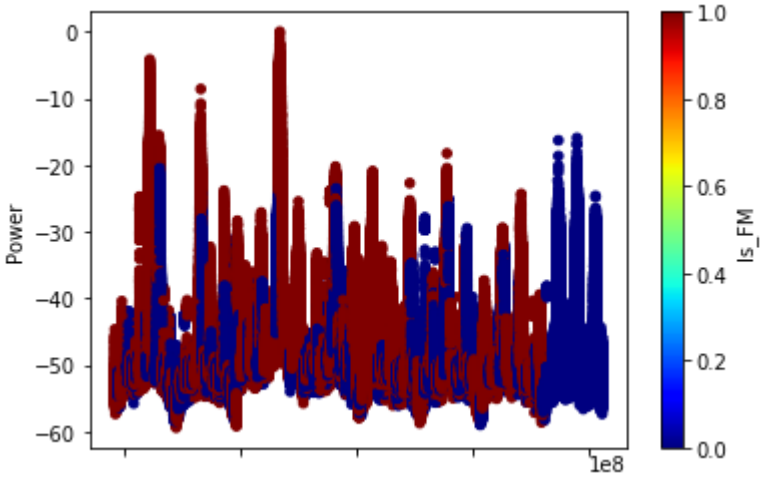
```
plt.scatter(dfs.Frequency,dfs.Power)
plt.ylabel('Power(dBs)')
plt.xlabel('Frequency')
plt.show()
```



Red values show it's a FM station. Blue values show it's not a FM station.

In [9]:

```
dfs.plot.scatter('Frequency', 'Power', c='Is_FM', colormap='jet')
plt.show()
```



Checking ratio of values in dataframe

In [10]:

```
dfs.groupby('Is_FM').count()
```

Out[10]:

Is_FM	Frequency	Power
0	415500	415500
1	567780	567780

Step 3 -

Train all the classification models to find the best one

Note - For training, frequency range is from 87 MHz to 108 MHz

We'll be using 12% of data as validation data hence it will not be used to train.

In [11]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import Perceptron
import xgboost as xgb

from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
import warnings
```

```
D:\python\lib\site-packages\sklearn\cross_validation.py:41: DeprecationWarni
ng: This module was deprecated in version 0.18 in favor of the model_selecti
on module into which all the refactored classes and functions are moved. Als
o note that the interface of the new CV iterators are different from that of
this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

In [12]:

```
X = dfs.drop('Is_FM', axis=1)
y = dfs['Is_FM']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.12, random_state=0)
```

In [13]:

```
regression_model = LogisticRegression()
regression_model.fit(X_train, y_train)

y_predict = regression_model.predict(X_train)
print("Using Logistic regression -\nTraining accuracy :",accuracy_score(y_train,y_predict)*


y_predict = regression_model.predict(X_test)
print("Validation accuracy :",accuracy_score(y_test,y_predict)*100,"%")
```

```
Using Logistic regression -
Training accuracy : 57.7290052075 %
Validation accuracy : 57.8495516721 %
```

In [14]:

```
knn = KNeighborsClassifier(n_neighbors = 3)
knn.fit(X_train, y_train)
y_predict = knn.predict(X_train)
print("Using KNN -\nTraining accuracy :",accuracy_score(y_train,y_predict)*100,"%")
y_predict = knn.predict(X_test)
print("Validation accuracy :",accuracy_score(y_test,y_predict)*100,"%")
```

```
Using KNN -
Training accuracy : 100.0 %
Validation accuracy : 100.0 %
```

In [15]:

```
perceptron = Perceptron()
warnings.filterwarnings("ignore")
perceptron.fit(X_train, y_train)
y_predict = perceptron.predict(X_train)
print("Using Perceptron -\nTraining accuracy :",accuracy_score(y_train,y_predict)*100,"%")
y_predict = perceptron.predict(X_test)
print("Validation accuracy :",accuracy_score(y_test,y_predict)*100,"%")
```

```
Using Perceptron -
Training accuracy : 42.2709947925 %
Validation accuracy : 42.1504483279 %
```

In [16]:

```
gradboost = xgb.XGBClassifier(n_estimators=1000)
gradboost.fit(X_train, y_train)
y_predict = gradboost.predict(X_train)
print("Using XGBoost -\nTraining accuracy :",accuracy_score(y_train,y_predict)*100,"%")
y_predict = gradboost.predict(X_test)
print("Validation accuracy :",accuracy_score(y_test,y_predict)*100,"%")
```

```
Using XGBoost -
Training accuracy : 100.0 %
Validation accuracy : 100.0 %
```

Fitting model with 100% of data to use it on randomly generated test data

In [17]:

```
regression_model = LogisticRegression()
regression_model.fit(X, y)

y_predict = regression_model.predict(X)
print("Using Logistic regression -\nTraining accuracy :",accuracy_score(y,y_predict)*100,"%

knn = KNeighborsClassifier(n_neighbors = 3)
knn.fit(X, y)

y_predict = knn.predict(X)
print("Using KNN -\nTraining accuracy :",accuracy_score(y,y_predict)*100,"%")

perceptron = Perceptron()
warnings.filterwarnings("ignore")
perceptron.fit(X, y)

y_predict = perceptron.predict(X)
print("Using Perceptron -\nTraining accuracy :",accuracy_score(y,y_predict)*100,"%")

gradboost = xgb.XGBClassifier(n_estimators=1000)
gradboost.fit(X, y)

y_predict = gradboost.predict(X)
print("Using XGBoost -\nTraining accuracy :",accuracy_score(y,y_predict)*100,"%")
```

```
Using Logistic regression -
Training accuracy : 57.7434708323 %
Using KNN -
Training accuracy : 100.0 %
Using Perceptron -
Training accuracy : 42.2565291677 %
Using XGBoost -
Training accuracy : 99.9986778944 %
```

Step - 4

Testing with test data in frequency range 87MHz to 108MHz

Command used to generate file -

rtl_power -f 87M:108M:1k -g 20 -i 10 -e 5m lgfile.csv

We'll follow similar steps as training to convert raw data to an useful form which we can use as test data.

In [18]:

```
dfs = pd.read_csv("lgfile.csv", header=None)
```

In [19]:

```
dfs.head()
```

Out[19]:

	0	1	2	3	4	5	6	7	8	9	...	4093
0	2018-04-15	16:06:13	87000000	89625000	640.87	38	-45.84	-55.77	-56.08	-57.61	...	-56.32
1	2018-04-15	16:06:13	89625000	92250000	640.87	38	-55.28	-57.92	-56.26	-55.92	...	-56.83
2	2018-04-15	16:06:13	92250000	94875000	640.87	38	-37.90	-37.65	-38.25	-39.42	...	-37.16
3	2018-04-15	16:06:13	94875000	97500000	640.87	38	-35.69	-39.10	-37.89	-38.95	...	-34.19
4	2018-04-15	16:06:13	97500000	100125000	640.87	38	-53.26	-52.24	-52.38	-53.54	...	-48.42

5 rows × 4103 columns

In [20]:

```
dfs.describe()
```

Out[20]:

	2	3	4	5	6	7	
count	2.400000e+02	2.400000e+02	2.400000e+02	240.000000	240.00000	240.000000	240.00000
mean	9.618750e+07	9.881250e+07	6.408700e+02	38.800000	-47.73125	-48.241125	-48.50691
std	6.027200e+06	6.027200e+06	2.164564e-12	0.981844	7.12496	7.177868	7.09413
min	8.700000e+07	8.962500e+07	6.408700e+02	38.000000	-57.59000	-58.660000	-59.31000
25%	9.159375e+07	9.421875e+07	6.408700e+02	38.000000	-53.90750	-54.582500	-54.74000
50%	9.618750e+07	9.881250e+07	6.408700e+02	38.000000	-51.58500	-51.545000	-52.29500
75%	1.007812e+08	1.034062e+08	6.408700e+02	40.000000	-40.36000	-40.975000	-41.15250
max	1.053750e+08	1.080000e+08	6.408700e+02	40.000000	-34.48000	-35.650000	-35.60000

8 rows × 4101 columns

In [21]:

```
dfs = conversion_function(dfs)
```

In [22]:

```
dfs.head()
```

Out[22]:

	Frequency	Is_FM	Power
0	8.962500e+07	1	-45.84
1	8.962564e+07	1	-55.77
2	8.962628e+07	1	-56.08
3	8.962692e+07	1	-57.61
4	8.962756e+07	1	-54.90

In [23]:

```
X = dfs.drop('Is_FM', axis=1)
y = dfs['Is_FM']
```

In [24]:

```
y_predict = regression_model.predict(X)
print("Using Logistic regression -\nAccuracy Obtained :",accuracy_score(y,y_predict)*100,"%
```

```
Using Logistic regression -
Accuracy Obtained : 57.7434708323 %
```

In [25]:

```
y_predict = knn.predict(X)
print("Using KNN -\nAccuracy Obtained :",accuracy_score(y,y_predict)*100,"%")
```

```
Using KNN -
Accuracy Obtained : 100.0 %
```

In [26]:

```
y_predict = perceptron.predict(X)
print("Using Perceptron -\nAccuracy Obtained :",accuracy_score(y,y_predict)*100,"%")
```

```
Using Perceptron -
Accuracy Obtained : 42.2565291677 %
```

In [27]:

```
y_predict = gradboost.predict(X)
print("Using XGBoost -\nAccuracy Obtained :",accuracy_score(y,y_predict)*100,"%")
```

```
Using XGBoost -
Accuracy Obtained : 99.9963387845 %
```

Step - 5

Testing with test data in frequency range 60MHz to 2400MHz

Command used to generate file -

rtl_power -f 60M:2400M:1k -g 20 -i 10 -e 60m logfile1.csv

As the file size of logfile1 is 1.7 Gb, we'll use below code to divide it into many csv files having 500 rows each. Total 111 files were made.

Logfile1 is too big tobe added to github hence generation of file is required.

In [28]:

```
#df = pd.read_csv("logfile1.csv", header=None,chunksize=500)
#warnings.filterwarnings("ignore")
#count = 1
path  = "C:/Users/Bomber11/Desktop/"  #Enter path where you want to store 100+ csv files
#for chunk in df:
#    name = path+"checkfile_"+str(count)+".csv"
#    chunk.to_csv(name,header=None,index=None)
#    print(count)
#    count+=1
```

Finding accuracy score for all csv files when rows having infinite values are dropped -

In [29]:

```
rm = 0
kn = 0
per = 0
xg = 0
for letscheck in range (1,112):
    dfs = pd.read_csv(path+"checkfile_"+str(letscheck)+".csv", header=None)
    warnings.filterwarnings("ignore")

    startf = dfs[2][0]
    endf = dfs[3][len(dfs)-1]

    dfs = conversion_function(dfs)
    dfs = dfs.replace([np.inf, -np.inf], np.nan)
    dfs = dfs.dropna(axis=0, how='any')

    X = dfs.drop('Is_FM', axis=1)
    y = dfs['Is_FM']

    print("From frequency range ", startf ," to ", endf)

    y_predict = regression_model.predict(X)
    rm = rm + accuracy_score(y,y_predict)
    print("Using Logistic regression -\nAccuracy Obtained :",accuracy_score(y,y_predict)*10

    y_predict = knn.predict(X)
    kn = kn + accuracy_score(y,y_predict)
    print("Using KNN -\nAccuracy Obtained :",accuracy_score(y,y_predict)*100,"%")

    y_predict = perceptron.predict(X)
    per = per + accuracy_score(y,y_predict)
    print("Using Perceptron -\nAccuracy Obtained :",accuracy_score(y,y_predict)*100,"%")

    y_predict = gradboost.predict(X)
    xg = xg + accuracy_score(y,y_predict)
    print("Using XGBoost -\nAccuracy Obtained :",accuracy_score(y,y_predict)*100,"%")
```

```
Accuracy Obtained : 98.2716729656 %
From frequency range  177525870  to  1576643369
Using Logistic regression -
Accuracy Obtained : 0.0 %
Using KNN -
Accuracy Obtained : 100.0 %
Using Perceptron -
Accuracy Obtained : 100.0 %
Using XGBoost -
Accuracy Obtained : 100.0 %
From frequency range  1576643370  to  2147483309
Using Logistic regression -
Accuracy Obtained : 0.0 %
Using KNN -
Accuracy Obtained : 100.0 %
Using Perceptron -
Accuracy Obtained : 100.0 %
Using XGBoost -
Accuracy Obtained : 100.0 %
```

In [33]:

```
; KNN = ",(kn/111)*100,"%\nTotal accuracy using Perceptron = ",(per/111)*100,"%\nTotal accur
```

```
Total accuracy using Logistic regression =  0.698660335002 %
Total accuracy using KNN =  97.9655933516 %
Total accuracy using Perceptron =  98.4084825221 %
Total accuracy using XGBoost =  97.9647016024 %
```

Assumption - 1 -> Let's replace all nan values mean of the column and compare the results

In [41]:

```
rm = 0
kn = 0
per = 0
xg = 0
for letscheck in range (1,112):
    dfs = pd.read_csv(path+"checkfile_"+str(letscheck)+".csv", header=None)
    warnings.filterwarnings("ignore")

    startf = dfs[2][0]
    endf = dfs[3][len(dfs)-1]

    dfs = conversion_function(dfs)
    dfs = dfs.replace([np.inf, -np.inf], np.nan)
    dfs = dfs.fillna(dfs.mean())

    X = dfs.drop('Is_FM', axis=1)
    y = dfs['Is_FM']

    print("From frequency range ", startf ," to ", endf)

    y_predict = regression_model.predict(X)
    rm = rm + accuracy_score(y,y_predict)
    print("Using Logistic regression -\nAccuracy Obtained :",accuracy_score(y,y_predict)*10

    y_predict = knn.predict(X)
    kn = kn + accuracy_score(y,y_predict)
    print("Using KNN -\nAccuracy Obtained :",accuracy_score(y,y_predict)*100,"%")

    y_predict = perceptron.predict(X)
    per = per + accuracy_score(y,y_predict)
    print("Using Perceptron -\nAccuracy Obtained :",accuracy_score(y,y_predict)*100,"%")

    y_predict = gradboost.predict(X)
    xg = xg + accuracy_score(y,y_predict)
    print("Using XGBoost -\nAccuracy Obtained :",accuracy_score(y,y_predict)*100,"%")
```

```
Accuracy Obtained : 98.2711252136 %
From frequency range  177525870  to  1576643369
Using Logistic regression -
Accuracy Obtained : 0.0 %
Using KNN -
Accuracy Obtained : 100.0 %
Using Perceptron -
Accuracy Obtained : 100.0 %
Using XGBoost -
Accuracy Obtained : 100.0 %
From frequency range  1576643370  to  2147483309
Using Logistic regression -
Accuracy Obtained : 0.0 %
Using KNN -
Accuracy Obtained : 100.0 %
Using Perceptron -
Accuracy Obtained : 100.0 %
Using XGBoost -
Accuracy Obtained : 100.0 %
```

In [43]:

```
; KNN = ",(kn/111)*100,"%\nTotal accuracy using Perceptron = ",(per/111)*100,"%\nTotal accur
```

```
Total accuracy using Logistic regression =  0.705264014319 %
Total accuracy using KNN =  98.8482304125 %
Total accuracy using Perceptron =  99.2947359857 %
Total accuracy using XGBoost =  98.84733281 %
```

So Accuracy score for KNN and XGBoost increased by a little amount.