


**GAIZKIA ADELINE ATMAKA**

2501972493 LA05

**NO.1 UTS SPEECH AUDIO PROCESSING**LINK VIDEO : <https://www.youtube.com/playlist?list=PLgEV6IOXdLB9B6nS4YW68-9W8oyTpVG08>


```
1 from google.colab import drive
2 drive.mount('/content/drive')
```

 Mounted at /content/drive
**Import Libraries**

```
1 import os
2 import librosa
3 import librosa.display
4 import IPython.display as ipd
5 import torchaudio
6
7 import pandas as pd
8 import numpy as np
9 import matplotlib.pyplot as plt
10 import seaborn as sns
11
12 import warnings
13 from scipy.io import wavfile
14
15 from sklearn.tree import DecisionTreeClassifier
16 from sklearn.ensemble import RandomForestClassifier
17 from sklearn.model_selection import GridSearchCV
18 from sklearn.model_selection import train_test_split
19 from sklearn.metrics import classification_report, confusion_matrix
20 from sklearn.utils import resample
21
22 warnings.filterwarnings('ignore')
```


**Data Preparation**

```
1 os.chdir('/content/drive/MyDrive/Colab Notebooks/Speech Audio Processing/UTS/No1/Dataset1 dogcat audio')
2 !ls
```


 cats\_dogs train\_test\_split.csv utils.py
**✎ Exploratory Data Analysis (EDA)**

Melakukan plot pada signal cat and dog untuk mengetahui distribusi signal dari setiap audio (cat and dog) dan melakukan plot waveform untuk mengetahui perbedaan signal frequency pada audio cat and dog

```
1 signal_1, sr1 = torchaudio.load(f'./cats_dogs/train/cat/cat_1.wav')
2 signal_2, sr2 = torchaudio.load(f'./cats_dogs/train/dog/dog_barking_1.wav')
3 sr1, sr2
```

 (16000, 16000)
**Contoh Sample Audio**

```
1 ipd.Audio('./cats_dogs/train/cat/cat_1.wav')
```

 0:01 / 0:11


```
1 ipd.Audio('./cats_dogs/train/dog/dog_barking_1.wav')
```

 0:00 / 0:11


Karena datanya masih berbentuk .wav, disini saya masukkan dalam 1 variable untuk mempermudah analisa data

```
1 def load_files(directory):
2     sounds = []
3     for filename in os.listdir(directory):
4         if filename.endswith(".wav"):
5             sample_rate, data = wavfile.read(os.path.join(directory, filename))
6             sounds.append(data)
7     return sounds
```

```
1 cats_sounds = load_files('./cats_dogs/train/cat/')
2 dogs_sounds = load_files('./cats_dogs/train/dog/')
```

```
1 len(cats_sounds), len(dogs_sounds)
```

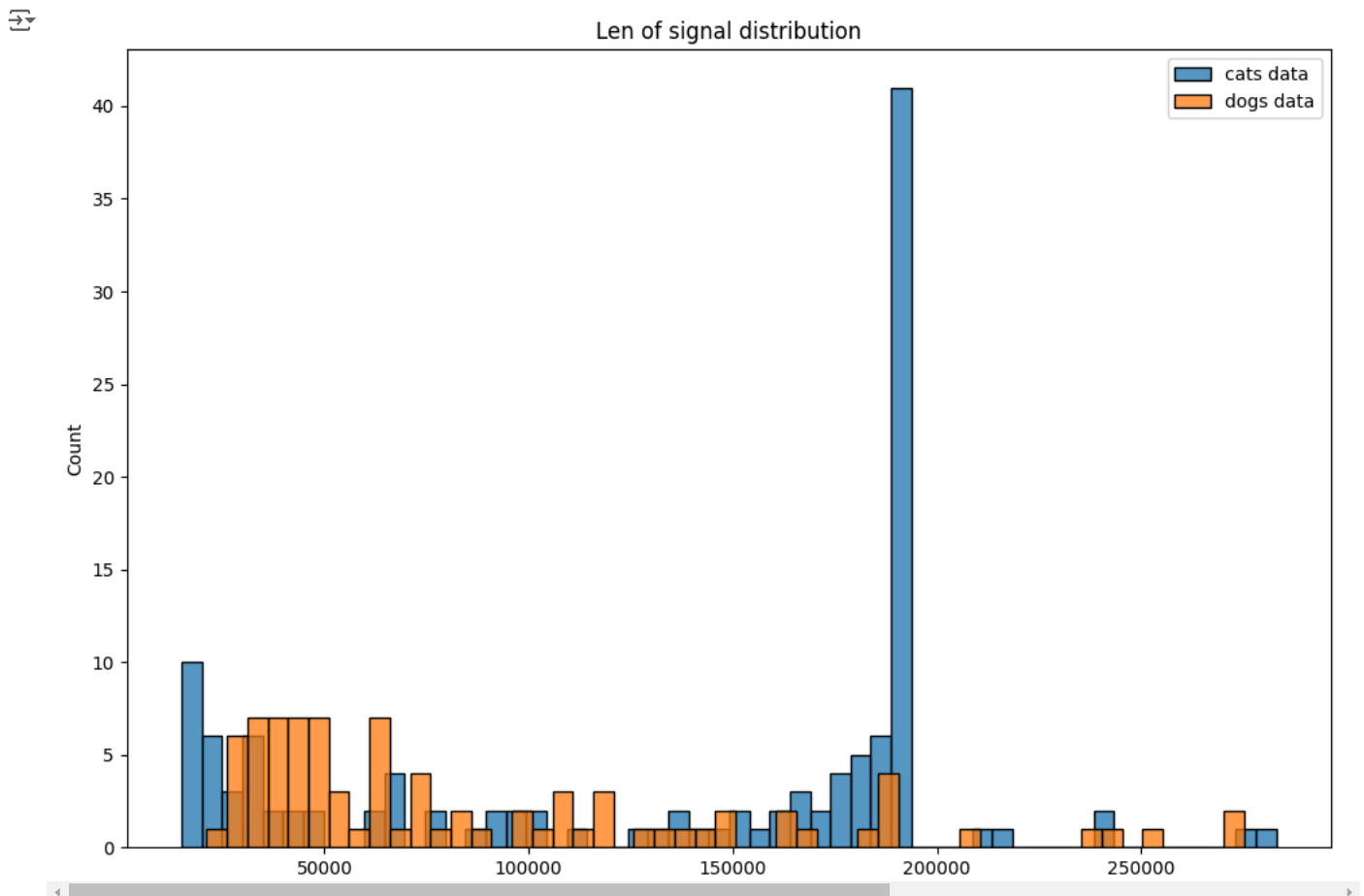
↗ (125, 85)

Jumlah / length dari audio data cat and dog

```
1 cats_sounds_len = [len(array) for array in cats_sounds]
2 dogs_sounds_len = [len(array) for array in dogs_sounds]
```

Signal distribution

```
1 plt.figure(figsize = (12, 8))
2 sns.histplot(data = cats_sounds_len, binwidth = 5000, color = 'tab:blue')
3 sns.histplot(data = dogs_sounds_len, binwidth = 5000, color = 'tab:orange')
4 plt.legend(['cats data', 'dogs data'])
5 plt.title('Len of signal distribution')
6 plt.show()
```



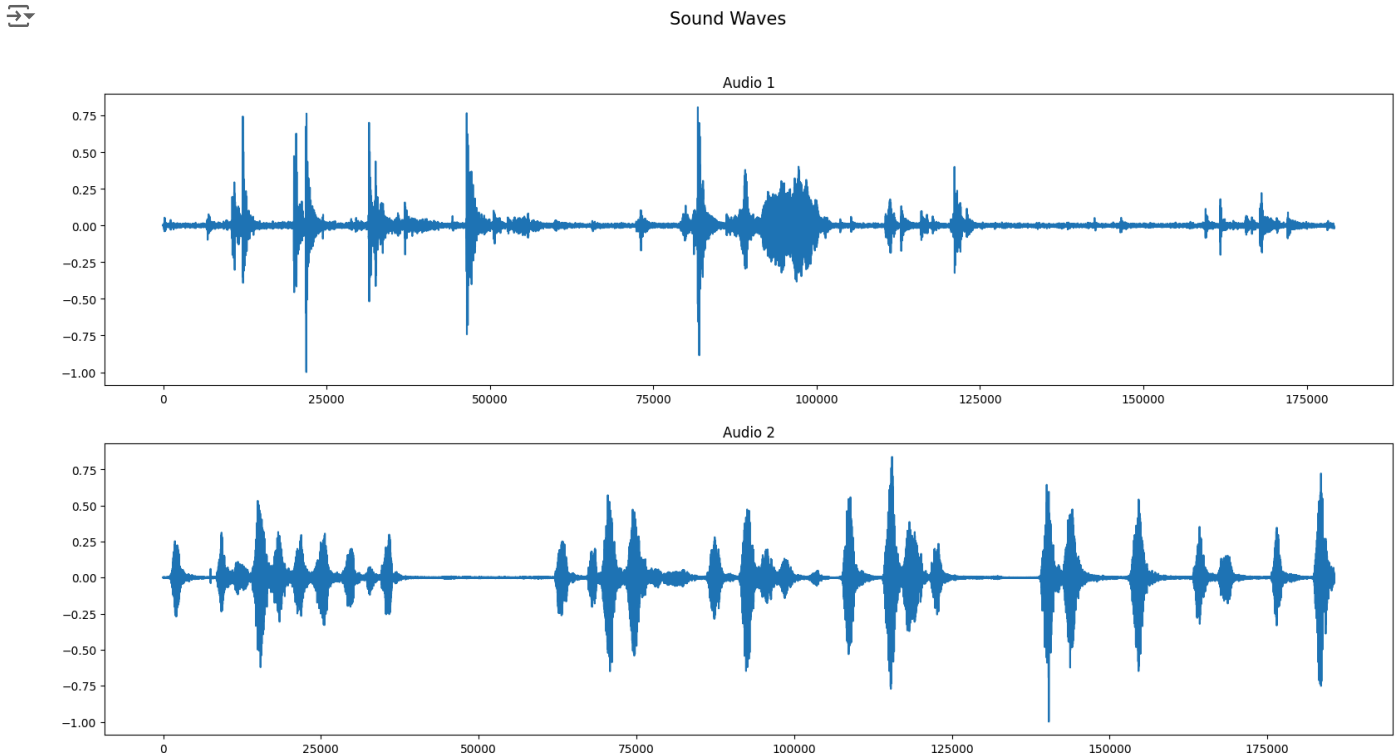
Dari distribusi diatas, terlihat bahwa data audio untuk cat cenderung memiliki panjang sinyal yang lebih panjang, dengan banyak sampel yang memiliki panjang di atas 150.000 sampel, dengan puncaknya di sekitar 200.000 sampel. Sebaliknya, data dog memiliki distribusi panjang yang lebih merata dan jarang mencapai panjang sinyal setinggi data cat.

Waveform Plot, untuk mengetahui bagaimana pergerakan suara cat and dog

```

1 fig, ax = plt.subplots(2, 1, figsize=(20, 10))
2 fig.suptitle("Sound Waves", fontsize=15)
3
4 signal_1, sr = torchaudio.load(f'./cats_dogs/train/cat/cat_1.wav')
5
6 sns.lineplot(x = np.arange(len(signal_1[0,:].detach().numpy())), y = signal_1[0,:].detach().numpy(), ax = ax[0])
7 ax[0].set_title("Audio 1")
8
9 signal_2, sr = torchaudio.load(f'./cats_dogs/train/dog/dog_barking_1.wav')
10 sns.lineplot(x = np.arange(len(signal_2[0,:].detach().numpy())), y = signal_2[0,:].detach().numpy(), ax = ax[1])
11 ax[1].set_title("Audio 2")
12
13 plt.show()

```



Kedua plot menampilkan sinyal audio dalam domain waktu, yang berarti kita bisa melihat bagaimana amplitudo suara berfluktuasi selama durasi rekaman.

Pola yang bisa ditangkap dari plot ini, yaitu:

#### Perbedaan Pola Gelombang Suara

- Audio 1 (Cat sound): Sinyal audio menunjukkan pola suara yang lebih terpisah dan memiliki jeda di antara setiap suara. Ini menunjukkan bahwa suara kucing yang memiliki jeda atau interval tiap kali dia meong.
- Audio 2 (Dog sound): Sinyal audio menunjukkan pola yang lebih sering berulang dengan amplitudo yang mirip. Ini bisa menjadi indikasi suara gonggongan yang lebih teratur dan intensitas suara yang cenderung serupa.

#### Amplitudo Sinyal

Kedua audio memiliki variasi amplitudo yang besar, yang menunjukkan fluktuasi intensitas suara. Suara dog tampak memiliki amplitudo yang lebih tinggi di beberapa bagian, menandakan suara dog yang mungkin lebih keras atau lebih kuat dibandingkan suara kucing.

## ✓ Feature Extraction

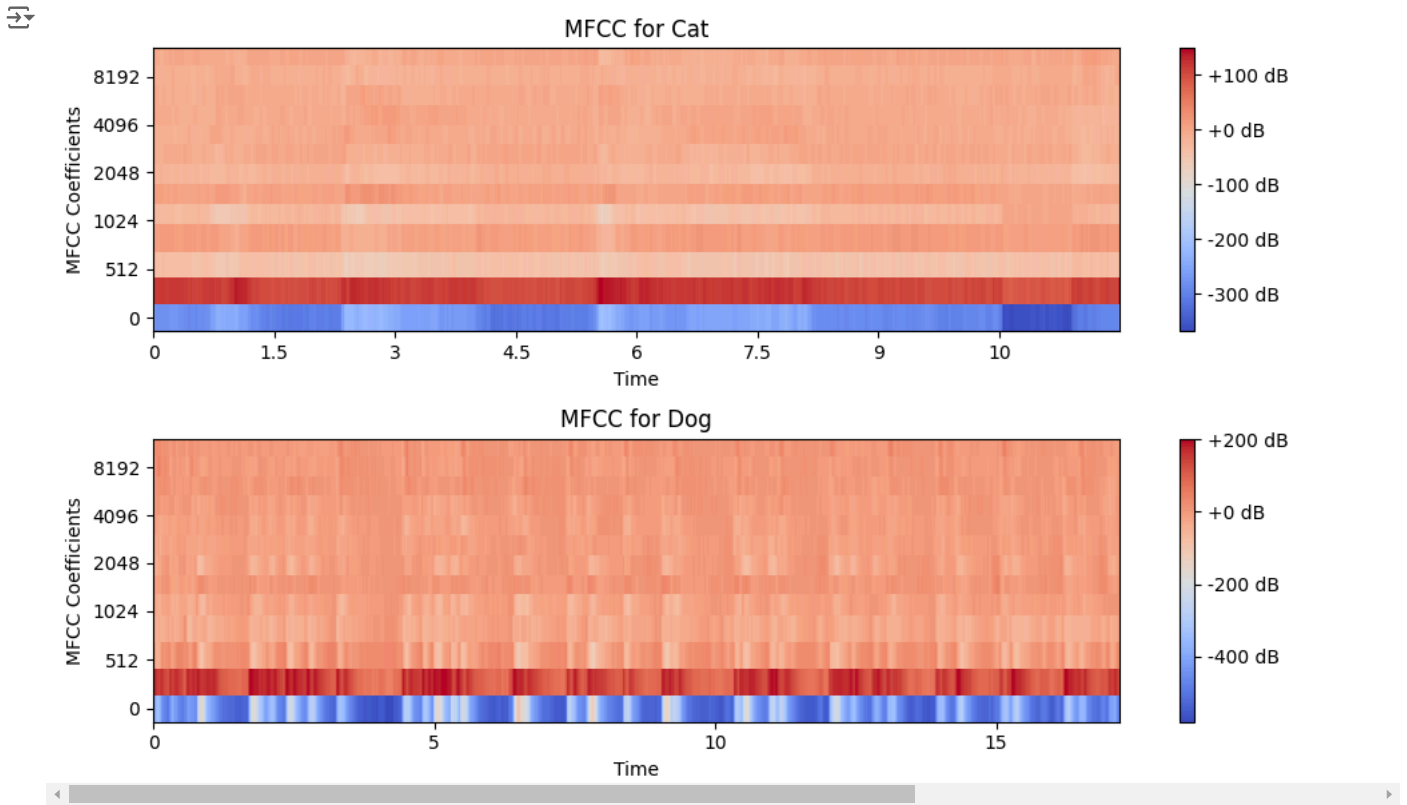
Melakukan feature extraction menggunakan MFCC (Mel-Frequency Cepstral Coefficients) sangat populer dalam pemrosesan suara atau sinyal audio, seperti voice recognition, speech recognition, dan pemrosesan sinyal audio lainnya. MFCC digunakan dalam ekstraksi fitur audio karena dapat meniru cara pendengaran manusia, mengurangi dimensi data, dan efektif untuk pengenalan suara serta analisis audio. Hal tersebut dikarenakan MFCC robust terhadap noise dan distorsi lainnya yang membuat hal tersebut cocok untuk animal audio classification.

Melakukan feature extraction menggunakan beberapa teknik ekstraksi fitur dari librosa. Fitur-fitur yang diekstrak, yaitu: RMS energy (time domain), frequency domain (f0), MFCC, spectral centroid, dan spectral bandwidth.

```

1 def extract_features(cat_dir, dog_dir):
2     cat_features, dog_features = [], []
3     cat_mfcc, dog_mfcc = [], [] # Store MFCCs for visualization
4
5     for file, label_list, mfcc_list in zip([cat_dir, dog_dir], [cat_features, dog_features], [cat_mfcc, dog_mfcc]):
6         for audio_file in os.listdir(file):
7             audio_path = os.path.join(file, audio_file)
8             waveform, sample_rate = librosa.load(audio_path)
9
10            # RMS energy (time domain)
11            rms = np.mean(librosa.feature.rms(y=waveform))
12
13            # Fundamental frequency
14            f0 = librosa.yin(waveform, fmin=50, fmax=500).mean()
15
16            # MFCCs
17            mfccs = librosa.feature.mfcc(y=waveform, sr=sample_rate, n_mfcc=13)
18            mfccs_mean = np.mean(mfccs, axis=1)
19
20            # Store MFCCs for visualization
21            mfcc_list.append(mfccs)
22
23            # Spectral centroid and bandwidth (additional features)
24            spectral_centroid = np.mean(librosa.feature.spectral_centroid(y=waveform, sr=sample_rate))
25            spectral_bandwidth = np.mean(librosa.feature.spectral_bandwidth(y=waveform, sr=sample_rate))
26
27            # Combine all features
28            features = [rms, f0, spectral_centroid, spectral_bandwidth, *mfccs_mean]
29            label_list.append(features)
30
31    return np.array(cat_features), np.array(dog_features), cat_mfcc, dog_mfcc
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```



```

1 # Calculate and print feature statistics for comparison
2 cat_rms_mean = cat_features[:, 0].mean()
3 dog_rms_mean = dog_features[:, 0].mean()
4 cat_f0_mean = cat_features[:, 1].mean()
5 dog_f0_mean = dog_features[:, 1].mean()
6
7 print(f"Average RMS (Cat): {cat_rms_mean:.2f}, Average RMS (Dog): {dog_rms_mean:.2f}")
8 print(f"Average Fundamental Frequency f0 (Cat): {cat_f0_mean:.2f}, Average f0 (Dog): {dog_f0_mean:.2f}")

```

```

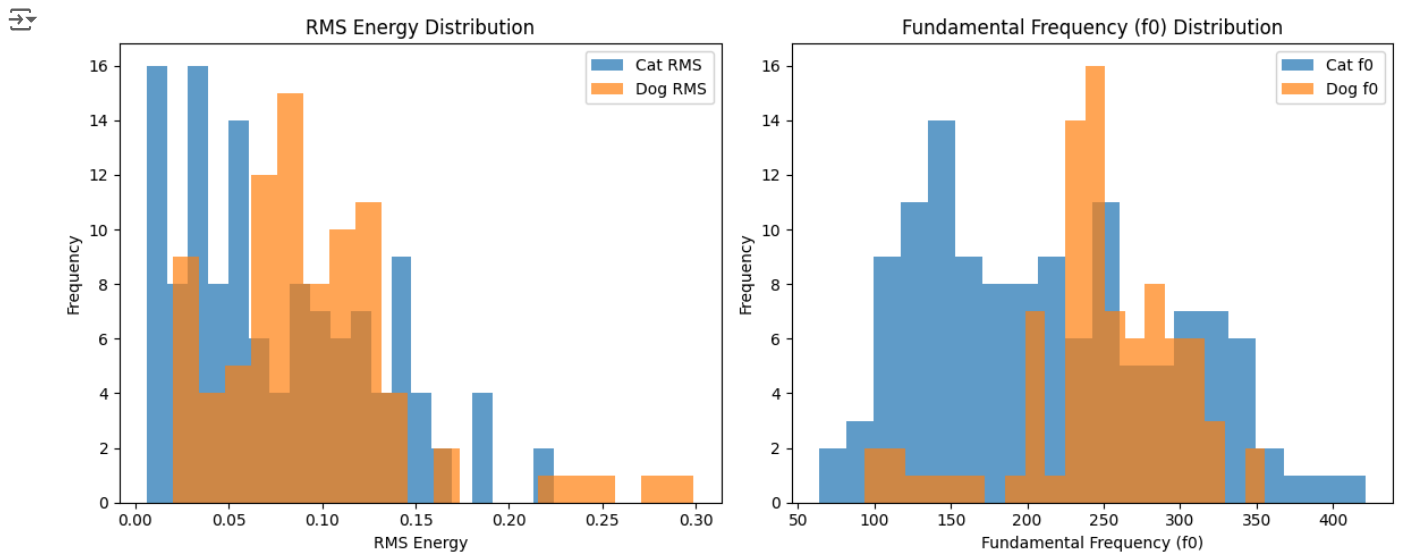
Average RMS (Cat): 0.08, Average RMS (Dog): 0.10
Average Fundamental Frequency f0 (Cat): 211.82, Average f0 (Dog): 247.27

```

```

1 # Plot distribution of RMS and f0 for cats and dogs
2 plt.figure(figsize=(12, 5))
3
4 plt.subplot(1, 2, 1)
5 plt.hist(cat_features[:, 0], bins=20, alpha=0.7, label='Cat RMS')
6 plt.hist(dog_features[:, 0], bins=20, alpha=0.7, label='Dog RMS')
7 plt.xlabel('RMS Energy')
8 plt.ylabel('Frequency')
9 plt.legend()
10 plt.title('RMS Energy Distribution')
11
12 plt.subplot(1, 2, 2)
13 plt.hist(cat_features[:, 1], bins=20, alpha=0.7, label='Cat f0')
14 plt.hist(dog_features[:, 1], bins=20, alpha=0.7, label='Dog f0')
15 plt.xlabel('Fundamental Frequency (f0)')
16 plt.ylabel('Frequency')
17 plt.legend()
18 plt.title('Fundamental Frequency (f0) Distribution')
19
20 plt.tight_layout()
21 plt.show()
22

```



## ✓ Analisis Fitur

### 1. RMS (Root Mean Square):

- **Average RMS (Cat): 0.08, Average RMS (Dog): 0.10**
- **Interpretasi:** RMS merupakan ukuran intensitas atau kekuatan sinyal suara. Nilai RMS yang lebih tinggi menunjukkan suara dengan intensitas yang lebih besar.
- Suara dog memiliki RMS rata-rata yang sedikit lebih tinggi dibandingkan suara cat. Ini menunjukkan bahwa, secara umum, suara dog mungkin cenderung lebih keras atau memiliki intensitas yang lebih besar dibandingkan suara cat.

### 2. Fundamental Frequency (f0):

- **Average f0 (Cat): 211.82 Hz, Average f0 (Dog): 247.27 Hz**
- **Interpretasi:** Fundamental frequency adalah frekuensi dasar dari sinyal suara dan sering kali dikaitkan dengan pitch (nada) dari suara. Nilai f0 yang lebih tinggi menunjukkan pitch yang lebih tinggi.
- Suara dog memiliki rata-rata f0 yang lebih tinggi daripada suara cat, yang menunjukkan bahwa suara anjing cenderung memiliki pitch yang lebih tinggi dibandingkan suara kucing.

## Memilih Fitur yang Paling Menjanjikan

Fitur yang paling menjanjikan untuk membedakan suara kucing dan anjing adalah fundamental frequency (f0). Karena:

- Pitch (f0) menunjukkan perbedaan yang lebih signifikan antara suara cat dan dog dibandingkan RMS. Nilai rata-rata f0 untuk dog lebih tinggi, yang mungkin mencerminkan pola suara dog yang berbeda secara mendasar dari suara cat.
- Pitch merupakan karakteristik yang membedakan suara hewan karena terkait langsung dengan cara hewan tersebut menghasilkan suara. Anjing biasanya memiliki nada yang lebih tinggi dalam suara mereka, terutama saat menggonggong, sementara kucing mungkin memiliki pitch yang sedikit lebih rendah dan bervariasi ketika mengeong.
- RMS dapat bervariasi tergantung pada kondisi lingkungan atau volume rekaman, sehingga cenderung lebih tidak konsisten sebagai pembeda utama antara dua jenis suara. Walaupun RMS dapat membantu membedakan suara, fitur ini mungkin kurang andal sebagai pembeda utama dibandingkan f0.

Melakukan downsampling agar jumlah data antara dua kelas (cat and dog) menjadi seimbang. Tujuannya adalah untuk mengatasi masalah ketidakseimbangan kelas dalam dataset, yang dapat memengaruhi performa model. Setelah itu saya memberikan label 0 sebagai cat dan 1 sebagai dog

```
1 # Downsampling to balance classes
2 target_size = min(len(cat_features), len(dog_features))
3 cat_features_downsampled = resample(cat_features, n_samples=target_size, random_state=42)
4 X = np.vstack((cat_features_downsampled, dog_features))
5 y = np.array([0] * len(cat_features_downsampled) + [1] * len(dog_features))

1 print("Shape of X after downsampling:", X.shape)
2 print("Shape of y:", y.shape)
```

```

↳ Shape of X after downsampling: (170, 17)
Shape of y: (170,)

```

```

1 # Splitting data
2 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

```

## ✓ Modeling

### Random Forest

```

1 # Initialize the Random Forest model
2 rf_model = RandomForestClassifier(random_state=42)
3
4 # Train the model using the training data
5 rf_model.fit(X_train, y_train)
6
7 # Make predictions using the Random Forest model on the test data
8 rf_predictions = rf_model.predict(X_test)
9
10 # Create a classification report
11 rf_report = classification_report(y_test, rf_predictions, zero_division=1)
12 print(rf_report)

```

```

↳

```

	precision	recall	f1-score	support
0	0.95	1.00	0.98	21
1	1.00	0.92	0.96	13
accuracy			0.97	34
macro avg	0.98	0.96	0.97	34
weighted avg	0.97	0.97	0.97	34

- Model Random Forest menunjukkan performa yang sangat baik dalam mengenali kedua kelas (cat dan dog) dengan akurasi 97%.
- Precision dan recall untuk kelas cat sangat tinggi (kedua nilai mendekati 1.00), yang menunjukkan bahwa model sangat baik dalam memprediksi kelas 0. Precision dan recall untuk kelas dog juga cukup baik, meskipun recall untuk kelas 1 sedikit lebih rendah (0.92), yang berarti ada sekitar 8% data kelas dog yang salah diprediksi.
- F1-score untuk kedua kelas menunjukkan keseimbangan yang sangat baik antara precision dan recall.
- Macro avg dan weighted avg menunjukkan hasil yang sangat baik, dengan model cenderung memberikan performa yang adil dan tidak bias.

Secara keseluruhan, model ini dapat dikatakan sangat efisien dan efektif dalam tugas klasifikasi antara kucing dan anjing berdasarkan fitur yang diekstraksi.

### Decision Tree

```

1 dt_model = DecisionTreeClassifier(random_state = 42)
2 dt_model.fit(X_train, y_train)
3 dt_predictions = dt_model.predict(X_test)
4
5 dt_report = classification_report(y_test, dt_predictions)
6 print(dt_report)

```

```

↳

```

	precision	recall	f1-score	support
0	0.90	0.90	0.90	21
1	0.85	0.85	0.85	13
accuracy			0.88	34
macro avg	0.88	0.88	0.88	34
weighted avg	0.88	0.88	0.88	34

- Model Decision Tree memiliki akurasi yang cukup baik, yaitu 88%, yang berarti model memprediksi dengan benar sekitar 88% dari total data uji.
- Model ini memiliki nilai precision dan recall yang relatif seimbang untuk kedua kelas, dengan nilai tertinggi pada kelas cat (0.90) dan sedikit lebih rendah pada kelas dog (0.85).
- F1-Score untuk kelas cat adalah 0.90 dan untuk kelas dog adalah 0.85. Meskipun ada sedikit penurunan, model tetap menunjukkan kinerja yang baik dalam menjaga keseimbangan antara precision dan recall untuk kedua kelas.
- Baik macro average dan weighted average menunjukkan nilai yang konsisten (0.88), yang berarti model ini memiliki kinerja yang relatif baik.

Secara keseluruhan, model ini dapat dikatakan cukup baik dalam tugas klasifikasi antara kucing dan anjing berdasarkan fitur yang diekstraksi

## ✓ Hyperparameter Tuning

Penentuan hyperparameter yang digunakan didasarkan pada cara decision tree bekerja dan untuk mengoptimalkan kinerja model sambil menghindari overfitting.

- criterion: ['gini', 'entropy']** Alasan: criterion mengontrol bagaimana kualitas split setiap node dihitung. Gini impurity cenderung lebih cepat dalam komputasi dan biasanya digunakan dalam banyak aplikasi klasifikasi. Entropy lebih sensitif terhadap distribusi kelas dan cenderung lebih stabil pada data yang tidak seimbang.
- splitter: ['best', 'random']** Alasan: splitter mengontrol bagaimana split dilakukan di setiap node
- max\_depth: [None, 2, 4, 6, 8, 10, 20, 30, 40]** Alasan: max\_depth mengontrol ukuran decision tree. Tree yang terlalu dalam dapat menyebabkan overfitting, sementara tree yang terlalu dangkal dapat menyebabkan underfitting.
- min\_samples\_split: [1, 2, 5, 10]** Alasan: min\_samples\_split menentukan jumlah sampel minimum yang dibutuhkan untuk membagi suatu node. Nilai yang lebih tinggi mencegah pembagian yang terlalu kecil, yang membantu menghindari overfitting. Memilih nilai yang lebih tinggi untuk parameter ini akan membatasi jumlah split dan regularisasi untuk mencegah split yang tidak perlu.
- min\_samples\_leaf: [1, 2, 4]** Alasan: min\_samples\_leaf menentukan jumlah minimum sampel yang harus ada di setiap leaf tree. Ini adalah bentuk regularisasi untuk mencegah pembentukan leaf yang sangat spesifik yang hanya berlaku untuk sedikit sampel. Nilai yang lebih besar memastikan bahwa model lebih generalisasi dan menghindari split yang terlalu mendetail.
- max\_features: ['auto', 'sqrt', 'log2']** Alasan: Parameter ini mengontrol jumlah fitur yang akan dipertimbangkan untuk setiap split node sqrt dan log2 sering digunakan untuk meningkatkan generalisasi dan menghindari overfitting dengan membatasi jumlah fitur yang dipertimbangkan pada setiap split.

```
1 # Decision Tree Model with Hyperparameter Tuning
2 dt_params = {
3     'criterion': ['gini', 'entropy'],
4     'splitter': ['best', 'random'],
5     'max_depth': [None, 2, 4, 6, 8, 10, 20, 30, 40],
6     'min_samples_split': [1, 2, 5, 10],
7     'min_samples_leaf': [1, 2, 4],
8     'max_features': ['auto', 'sqrt', 'log2']
9 }
10 dt_model_hyper2 = DecisionTreeClassifier(random_state=42)
11 dt_grid = GridSearchCV(dt_model_hyper2, dt_params, cv=5, scoring='accuracy')
12 dt_grid.fit(X_train, y_train)
13
14 # Evaluation
15 dt_best_model = dt_grid.best_estimator_
16 dt_predictions = dt_best_model.predict(X_test)
17 print("Best Decision Tree Hyperparameters:", dt_grid.best_params_)
18 print("Best Decision Tree Hyperparameters Estimator:", dt_grid.best_estimator_)
19 print("Decision Tree Classification Report:\n", classification_report(y_test, dt_predictions))
```

```
➤ Best Decision Tree Hyperparameters: {'criterion': 'gini', 'max_depth': None, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_split': 5}
Best Decision Tree Hyperparameters Estimator: DecisionTreeClassifier(max_features='sqrt', random_state=42)
```

```
Decision Tree Classification Report:
              precision    recall  f1-score   support

    0               1.00        0.95        0.98         21
    1               0.93        1.00        0.96         13

 accuracy               0.97
 macro avg              0.96
 weighted avg           0.97
```

- Model Decision Tree dengan best hyperparameter yang ditemukan melalui GridSearchCV memiliki performa yang sangat baik, dengan akurasi 97%.
- Model bekerja sangat baik untuk kedua kelas (kucing dan anjing) dengan precision dan recall yang tinggi, terutama untuk kelas anjing (dog), yang diprediksi dengan sempurna (recall = 1.00).
- F1-score yang tinggi untuk kedua kelas (0.98 untuk cat dan 0.96 untuk dog) menunjukkan keseimbangan yang baik antara precision dan recall.
- Metrik macro average dan weighted average yang juga tinggi menunjukkan bahwa model ini dapat mengatasi ketidakseimbangan jumlah data dengan sangat baik, menghasilkan performa yang stabil di seluruh dataset.

Terjadi peningkatan yang signifikan jika dibandingkan sebelum melakukan Hyperparameter Tuning. Hal ini berarti Hyperparameter yang dipilih memberikan dampak signifikan.



- a. `n_estimators` Alasan: Menyediakan berbagai pilihan jumlah pohon memungkinkan kita untuk mengeksplorasi trade-off antara kinerja model dan waktu komputasi.
- b. `max_depth` Alasan: Pengaturan ini memungkinkan kita untuk menemukan kedalaman yang optimal. Kedalaman yang terlalu besar bisa menyebabkan overfitting, sedangkan kedalaman yang terlalu kecil bisa menyebabkan underfitting.
- c. `min_samples_split` Alasan: `min_samples_split` menentukan jumlah sampel minimum yang dibutuhkan untuk membagi suatu node. Nilai yang lebih tinggi mencegah pembagian yang terlalu kecil, yang membantu menghindari overfitting. Memilih nilai yang lebih tinggi untuk parameter ini akan membatasi jumlah split dan regularisasi untuk mencegah split yang tidak perlu.
- d. `min_samples_leaf` Alasan: `min_samples_leaf` menentukan jumlah minimum sampel yang harus ada di setiap leaf tree. Ini adalah bentuk regularisasi untuk mencegah pembentukan leaf yang sangat spesifik yang hanya berlaku untuk sedikit sampel. Nilai yang lebih besar memastikan bahwa model lebih generalisasi dan menghindari split yang terlalu mendetail.
- e. `max_features` Alasan: Parameter ini mengontrol jumlah fitur yang akan dipertimbangkan untuk setiap split node `sqrt` dan `log2` sering digunakan untuk meningkatkan generalisasi dan menghindari overfitting dengan membatasi jumlah fitur yang dipertimbangkan pada setiap split.
- f. `criterion` Alasan: `criterion` menentukan bagaimana kualitas split pada setiap node dihitung.

```
1 # Random Forest Model with Hyperparameter Tuning
2 rf_params = {
3     'n_estimators': [50, 100, 150, 200],
4     'max_depth': [None, 5, 10, 15, 20],
5     'min_samples_split': [2, 5, 10],
6     'min_samples_leaf': [1, 2, 4],
7     'max_features': ['auto', 'sqrt', 'log2'],
8     'criterion': ['gini', 'entropy']
9 }
10 rf_model_hyper2 = RandomForestClassifier(random_state=42)
11 rf_grid = GridSearchCV(rf_model_hyper2, rf_params, cv=5, scoring='accuracy')
12 rf_grid.fit(X_train, y_train)
13
14 # Evaluation
15 rf_best_model = rf_grid.best_estimator_
16 rf_predictions = rf_best_model.predict(X_test)
17 print("Best Random Forest Hyperparameters:", rf_grid.best_params_)
18 print("Best Random Forest Hyperparameters Estimator:", rf_grid.best_estimator_)
19 print("Random Forest Classification Report:\n", classification_report(y_test, rf_predictions, zero_division=1))
```

```
Best Random Forest Hyperparameters: {'criterion': 'entropy', 'max_depth': None, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_
Best Random Forest Hyperparameters Estimator: RandomForestClassifier(criterion='entropy', min_samples_split=5,
                                random_state=42)
Random Forest Classification Report:
```

	precision	recall	f1-score	support
0	0.95	1.00	0.98	21
1	1.00	0.92	0.96	13
accuracy			0.97	34
macro avg	0.98	0.96	0.97	34
weighted avg	0.97	0.97	0.97	34

- Akurasi tinggi (97%) menunjukkan bahwa model sangat efektif dalam membedakan suara antara anjing dan kucing.
- Model lebih akurasi dalam mengenali suara kucing (precision dan recall kelas 0 lebih tinggi), namun akurasi untuk suara anjing juga sangat baik (precision kelas 1 adalah 1.00, recall 0.92).
- Precision dan recall sangat seimbang, menunjukkan bahwa model memiliki sedikit bias terhadap salah satu kelas.
- Metrik macro average dan weighted average yang juga tinggi menunjukkan bahwa model ini dapat mengatasi ketidakseimbangan jumlah data dengan sangat baik, menghasilkan performa yang stabil di seluruh dataset.

Namun hasil metrik ini tidak ada perubahan yang signifikan jika dibandingkan sebelum melakukan Hyperparameter Tuning. Hal ini bisa disebabkan Hyperparameter yang dipilih tidak memberikan dampak signifikan atau model sudah cukup kuat.

Kesimpulan: Untuk perbandingan dari kedua model, sebelum dilakukan hyperparameter tuning, Model Random Forest terbukti bekerja dengan lebih baik untuk dataset ini. Namun, Model Random Forest dan Decision Tree sama-sama memiliki akurasi 97% setelah dilakukan hyperparameter tuning.