Email Spam Classification

- Data set link: Email Spam Classification Data Set

Import Essential Libraries

In [168…
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
import warnings
warnings.filterwarnings("ignore")
```

Load data set

In [169…
```
df = pd.read_csv("email.csv")
```

In [170…
```
df.head()
```

Out[170…

	Category	Message
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...

Data Cleaning

In [171…
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5573 entries, 0 to 5572
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   Category  5573 non-null   object
 1   Message   5573 non-null   object
dtypes: object(2)
memory usage: 87.2+ KB
```

In [172…
```
# missing values
df.isnull().sum()
```

Out[172…
```
Category    0
Message     0
dtype: int64
```

In [173…
```
# check for duplicate values
df.duplicated().sum()
```

Out[173… 415

In [174…
```
# remove duplicates
df = df.drop_duplicates(keep='first')
```

In [175…
```
df.duplicated().sum()
```

Out[175… 0

In [176…
```
df.shape
```

Out[176… (5158, 2)

EDA

In [177…
```
df.head()
```

Out[177…

	Category	Message
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...

In [178…
```
df['Category'].value_counts()
```

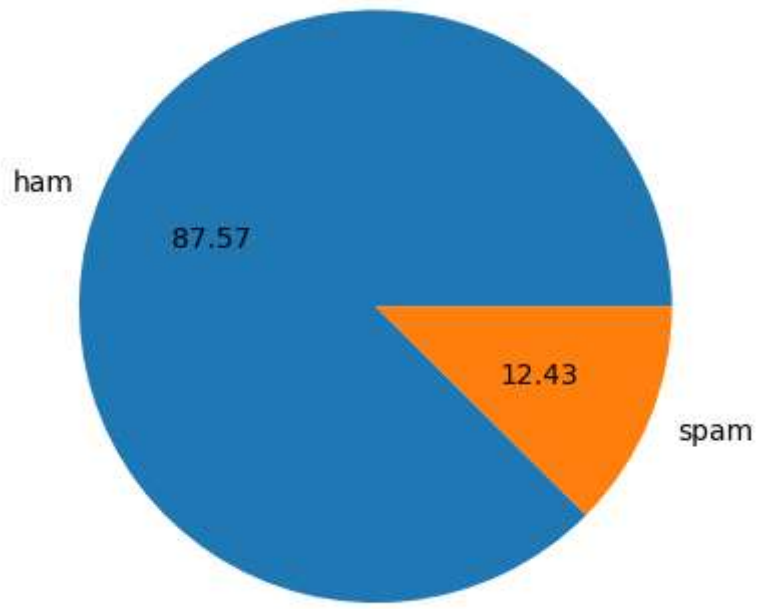
Out[178…
```
Category
ham               4516
spam               641
{"mode":"full"       1
Name: count, dtype: int64
```

In [179…
```
# Remove unwanted entry
df = df[df['Category'].isin(['ham', 'spam'])]
```

In [180…
```
df['Category'].value_counts()
```

Out[180…
```
Category
ham     4516
spam     641
Name: count, dtype: int64
```

In [181…
```
# Draw the pie chart
plt.pie(df['Category'].value_counts(), labels=['ham','spam'],autopct="%0.2f")
plt.show()
```



In [182…
```
# when doing preprocessing key-error show 103
print(103 in df.index)
```

```
False
```

In [183…
```
# Index 103 is missing
print(df.index.tolist())
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 2
3, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 4
4, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 6
5, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 8
6, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 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, 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, 208, 209, 210,
211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 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, 327, 328, 329, 330, 331,
332, 333, 334, 335, 336, 337, 338, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349,
350, 351, 352, 353, 354, 355, 356, 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, 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, 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, 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, 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, 656, 657, 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, 703, 704, 705, 706, 707, 708, 709, 710, 711, 712, 713, 714,
715, 716, 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, 770, 771, 772, 773, 774, 776, 777, 778, 779, 780, 782, 783, 784, 785, 786, 787,
788, 789, 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, 826, 827, 828, 829, 830, 831, 832, 833, 834, 835, 836, 837, 838, 839, 840,
841, 842, 843, 844, 845, 846, 847, 848, 849, 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, 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, 964, 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,
```

```
76, 5277, 5278, 5280, 5281, 5282, 5283, 5286, 5287, 5288, 5289, 5290, 5291, 5292, 52
93, 5294, 5295, 5296, 5297, 5298, 5299, 5300, 5302, 5303, 5304, 5305, 5306, 5307, 53
08, 5309, 5310, 5311, 5312, 5313, 5316, 5317, 5318, 5319, 5320, 5321, 5322, 5323, 53
24, 5325, 5326, 5327, 5328, 5329, 5330, 5331, 5332, 5333, 5334, 5335, 5336, 5337, 53
38, 5339, 5340, 5341, 5342, 5343, 5344, 5345, 5347, 5348, 5349, 5350, 5351, 5352, 53
53, 5354, 5355, 5356, 5358, 5359, 5360, 5361, 5362, 5363, 5364, 5367, 5368, 5369, 53
70, 5371, 5372, 5373, 5376, 5377, 5378, 5379, 5380, 5381, 5382, 5383, 5384, 5385, 53
87, 5388, 5390, 5391, 5392, 5393, 5394, 5395, 5396, 5397, 5398, 5399, 5400, 5401, 54
02, 5403, 5404, 5405, 5406, 5407, 5408, 5409, 5410, 5411, 5412, 5413, 5414, 5415, 54
16, 5417, 5418, 5419, 5420, 5421, 5422, 5424, 5426, 5427, 5428, 5429, 5430, 5431, 54
32, 5433, 5434, 5435, 5436, 5437, 5438, 5439, 5440, 5441, 5442, 5443, 5444, 5445, 54
46, 5447, 5448, 5449, 5450, 5451, 5452, 5453, 5454, 5455, 5456, 5459, 5461, 5462, 54
63, 5464, 5465, 5466, 5468, 5470, 5472, 5473, 5474, 5475, 5476, 5478, 5479, 5480, 54
81, 5482, 5483, 5484, 5485, 5486, 5487, 5489, 5491, 5492, 5493, 5494, 5495, 5496, 54
98, 5499, 5500, 5501, 5502, 5503, 5504, 5505, 5506, 5507, 5508, 5509, 5511, 5512, 55
13, 5514, 5515, 5516, 5517, 5518, 5519, 5520, 5521, 5522, 5523, 5525, 5526, 5527, 55
28, 5529, 5530, 5531, 5532, 5533, 5534, 5536, 5537, 5538, 5540, 5541, 5542, 5543, 55
44, 5545, 5546, 5547, 5548, 5549, 5550, 5551, 5552, 5554, 5555, 5556, 5557, 5559, 55
60, 5561, 5562, 5563, 5564, 5565, 5566, 5567, 5568, 5569, 5570, 5571]
```

In [184…

```
# Reset the index
df = df.reset_index(drop=True)
```

In [185…

```
# Again check and now it`s now its present
print(df.index.tolist())
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 2
3, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 4
4, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 6
5, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 8
6, 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,
```

```
7, 4898, 4899, 4900, 4901, 4902, 4903, 4904, 4905, 4906, 4907, 4908, 4909, 4910, 491
1, 4912, 4913, 4914, 4915, 4916, 4917, 4918, 4919, 4920, 4921, 4922, 4923, 4924, 492
5, 4926, 4927, 4928, 4929, 4930, 4931, 4932, 4933, 4934, 4935, 4936, 4937, 4938, 493
9, 4940, 4941, 4942, 4943, 4944, 4945, 4946, 4947, 4948, 4949, 4950, 4951, 4952, 495
3, 4954, 4955, 4956, 4957, 4958, 4959, 4960, 4961, 4962, 4963, 4964, 4965, 4966, 496
7, 4968, 4969, 4970, 4971, 4972, 4973, 4974, 4975, 4976, 4977, 4978, 4979, 4980, 498
1, 4982, 4983, 4984, 4985, 4986, 4987, 4988, 4989, 4990, 4991, 4992, 4993, 4994, 499
5, 4996, 4997, 4998, 4999, 5000, 5001, 5002, 5003, 5004, 5005, 5006, 5007, 5008, 500
9, 5010, 5011, 5012, 5013, 5014, 5015, 5016, 5017, 5018, 5019, 5020, 5021, 5022, 502
3, 5024, 5025, 5026, 5027, 5028, 5029, 5030, 5031, 5032, 5033, 5034, 5035, 5036, 503
7, 5038, 5039, 5040, 5041, 5042, 5043, 5044, 5045, 5046, 5047, 5048, 5049, 5050, 505
1, 5052, 5053, 5054, 5055, 5056, 5057, 5058, 5059, 5060, 5061, 5062, 5063, 5064, 506
5, 5066, 5067, 5068, 5069, 5070, 5071, 5072, 5073, 5074, 5075, 5076, 5077, 5078, 507
9, 5080, 5081, 5082, 5083, 5084, 5085, 5086, 5087, 5088, 5089, 5090, 5091, 5092, 509
3, 5094, 5095, 5096, 5097, 5098, 5099, 5100, 5101, 5102, 5103, 5104, 5105, 5106, 510
7, 5108, 5109, 5110, 5111, 5112, 5113, 5114, 5115, 5116, 5117, 5118, 5119, 5120, 512
1, 5122, 5123, 5124, 5125, 5126, 5127, 5128, 5129, 5130, 5131, 5132, 5133, 5134, 513
5, 5136, 5137, 5138, 5139, 5140, 5141, 5142, 5143, 5144, 5145, 5146, 5147, 5148, 514
9, 5150, 5151, 5152, 5153, 5154, 5155, 5156]
```

Data Pre-Processing

In [186…

```
ps = PorterStemmer()
```

In [187…

```
corpus = []
for i in range(0, len(df)):
    # Remove all the special char and keep only letter
    message = re.sub("[^a-zA-z]", " " , df['Message'][i])
    # Convert them into lower
    message = message.lower()
    # Split the text into individual words
    message = message.split()
    # Remove common stopwords and stem each word
    message = [ps.stem(word) for word in message if not word in stopwords.words('en
    # Join the words back into a single string
    message = ' '.join(message)
    corpus.append(message)
```

In [188…

```
corpus
```

Out[188… ['go jurong point crazi avail bugi n great world la e buffet cine got amor wat',
 'ok lar joke wif u oni',
 'free entri wkli comp win fa cup final tkt st may text fa receiv entri question s
td txt rate c appli',
 'u dun say earli hor u c alreadi say',
 'nah think goe usf live around though',
 'freemsg hey darl week word back like fun still tb ok xxx std chg send rcv',
 'even brother like speak treat like aid patent',
 'per request mell mell oru minnaminungint nurungu vettam set callertun caller pre
ss copi friend callertun',
 'winner valu network custom select receivea prize reward claim call claim code kl
valid hour',
 'mobil month u r entitl updat latest colour mobil camera free call mobil updat co
free',
 'gonna home soon want talk stuff anymor tonight k cri enough today',
 'six chanc win cash pound txt csh send cost p day day tsandc appli repli hl inf
o',
 'urgent week free membership prize jackpot txt word claim c www dbuk net lccltd p
obox ldnw rw',
 'search right word thank breather promis wont take help grant fulfil promis wonde
r bless time',
 'date sunday',
 'xxxmobilemovieclub use credit click wap link next txt messag click http wap xxxm
obilemovieclub com n qjkgighjjgcbl',
 'oh k watch',
 'eh u rememb spell name ye v naughti make v wet',
 'fine way u feel way gota b',
 'england v macedonia dont miss goal team news txt ur nation team eg england tri w
ale scotland txt poboxox w wq',
 'serious spell name',
 'go tri month ha ha joke',
 'pay first lar da stock comin',
 'aft finish lunch go str lor ard smth lor u finish ur lunch alreadi',
 'fffffffff alright way meet sooner',
 'forc eat slice realli hungri tho suck mark get worri know sick turn pizza lol',
 'lol alway convinc',
 'catch bu fri egg make tea eat mom left dinner feel love',
 'back amp pack car let know room',
 'ahhh work vagu rememb feel like lol',
 'wait still clear sure sarcast x want live us',
 'yeah got v apologet n fallen actin like spoilt child got caught till go badli ch
eer',
 'k tell anyth',
 'fear faint housework quick cuppa',
 'thank subscript rington uk mobil charg month pleas confirm repli ye repli char
g',
 'yup ok go home look time msg xuhui go learn nd may lesson',
 'oop let know roommat done',
 'see letter b car',
 'anyth lor u decid',
 'hello saturday go text see decid anyth tomo tri invit anyth',
 'pl go ahead watt want sure great weekend abiola',
 'forget tell want need crave love sweet arabian steed mmmmmm yummi',
 'rodger burn msg tri call repli sm free nokia mobil free camcord pleas call deliv
eri tomorrow',
 'see',

```
 'hi im relax time ever get everi day parti good night get home tomorrow ish',
 'wan come come lor din c stripe skirt',
 'xma stori peac xma msg love xma miracl jesu hav bless month ahead amp wish u mer
ri xma',
 'number',
 'chang e one next escal',
 'yetund class run water make ok pl',
 'lot happen feel quiet beth aunt charli work lot helen mo',
 'wait bu stop aft ur lect lar dun c go get car come back n pick',
 'aight thank comin',
 'heard abt tat',
 'lt gt think say syllabu',
 'umma say anyth',
 'give sec think think',
 'panason bluetoothhdset free nokia free motorola free doublemin doubletxt orang c
ontract call mobileupd call optout',
 'quit know still get hold anyon cud pick bout pm see pub',
 'poyyarikatur kolathupalayam unjalur post erod di lt gt',
 'dear hero leav qatar tonit apt opportun pl keep touch lt email gt kerala',
 'lol would mom would fit tell whole famili crazi terribl',
 'got home babe still awak',
 'dunno close oredi v fan',
 'buy pizza meat lover suprem u get pick',
 'ya told ask wat matter',
 'dear regret cudnt pick call drove frm ctla cochin home left mobil car ent style
ishtamayoo happi bakrid',
 'free st week nokia tone ur mob everi week txt nokia get txting tell ur mate www
getz co uk pobox w wq norm p tone',
 'shall send exe mail id',
 'nope watch tv home go v bore',
 'know wait check',
 'good afternoon gloriou anniversari day sweet j hope find happi content prey thin
k send teas kiss across sea coax imag fond souveni cougar pen',
 'guess somebodi know secretli fanci wanna find give us call landlin datebox essex
cm xn p min',
 'still tonight',
 'may call later pl',
 'pattern recent crap weekend',
 'sore throat scratch talk',
 ...]
```

Conclusion-1:

- Now our `corpus (message/email)` is a cleaned and preprocessed collection of text from each original message. Here's why:
    1. **Noise Removed:** Non-letter characters like punctuation and numbers were removed, so only meaningful words remain.
    2. **Uniform Case:** All words are converted to lowercase, making text case-insensitive (e.g., "Spam" and "spam" are treated the same).
    3. **Common Words Removed:** Stopwords (like "the," "is," "and") were filtered out, leaving only words that contribute more meaning.
    4. **Words Reduced to Roots:** Stemming reduced words to their root form (e.g., "running" becomes "run"), helping to treat different forms of the same word as

equivalent.

ent. nt.

Create the bags of words(BOW)

In [189…
```
from sklearn.feature_extraction.text import CountVectorizer
cv = CountVectorizer(max_features=2500,ngram_range=(1,2))
```

Independent / Input Features

In [190…
```
X = cv.fit_transform(corpus).toarray()
```

In [191…
```
X.shape
```

Out[191… (5157, 2500)

In [192…
```
cv.vocabulary_
```

Out[192…


```
{'go': 796,
 'point': 1613,
 'crazi': 438,
 'avail': 121,
 'bugi': 227,
 'great': 854,
 'world': 2431,
 'la': 1104,
 'cine': 333,
 'got': 841,
 'wat': 2342,
 'ok': 1485,
 'lar': 1115,
 'joke': 1068,
 'wif': 2396,
 'oni': 1506,
 'ok lar': 1488,
 'free': 720,
 'entri': 613,
 'wkli': 2422,
 'comp': 388,
 'win': 2400,
 'cup': 454,
 'final': 685,
 'st': 1971,
 'may': 1274,
 'text': 2084,
 'receiv': 1711,
 'question': 1675,
 'std': 1983,
 'txt': 2210,
 'rate': 1688,
 'appli': 84,
 'free entri': 725,
 'rate appli': 1690,
 'dun': 574,
 'say': 1803,
 'earli': 582,
 'alreadi': 58,
 'nah': 1392,
 'think': 2109,
 'goe': 817,
 'usf': 2280,
 'live': 1183,
 'around': 97,
 'though': 2120,
 'freemsg': 735,
 'hey': 929,
 'darl': 478,
 'week': 2366,
 'word': 2427,
 'back': 137,
 'like': 1169,
 'fun': 750,
 'still': 1985,
 'tb': 2060,
```

```
 'good night': 834,
 'congratul': 402,
 'cd': 303,
 'voucher': 2306,
 'gift': 786,
 'music': 1387,
 'tnc': 2149,
 'ldew': 1135,
 'ppmx': 1632,
 'congratul ur': 403,
 'gift guarante': 787,
 'draw txt': 563,
 'txt music': 2213,
 'music tnc': 1388,
 'tnc www': 2150,
 'www ldew': 2450,
 'ldew com': 1136,
 'com win': 376,
 'win ppmx': 2403,
 'ppmx age': 1633,
 'cal': 238,
 'hold': 954,
 'angri': 67,
 'wid': 2395,
 'dnt': 540,
 'true': 2193,
 'deep': 501,
 'care': 287,
 'ship': 1866,
 'lemm': 1151,
 'lemm know': 1152,
 'expect': 643,
 'mmm': 1326,
 'lover': 1226,
 'video': 2292,
 'handset': 904,
 'anytim': 77,
 'unlimit': 2236,
 'contact repli': 412,
 'repli offer': 1741,
 'video handset': 2293,
 'handset anytim': 905,
 'anytim network': 78,
 'network min': 1416,
 'min unlimit': 1311,
 'unlimit text': 2237,
 'camcord repli': 271,
 'repli call': 1738,
 ...}
```

In [193…

```
X.shape
```

Out[193…

```
(5157, 2500)
```

Output / Independent Features

In [194…
```
y = pd.get_dummies(df['Category'], drop_first=True).astype(int)
y = y.iloc[:, 0].values
```

In [195…
```
y
```

Out[195…
```
array([0, 0, 1, ..., 0, 0, 0])
```

In [196…
```
y.shape
```

Out[196…
```
(5157,)
```

Conclusion-2:

After applying Bag of Words (BOW) and setting `x` as the independent variable and `y` as the dependent variable, our data is now a structured format ready for classification. Here's why:

1. **Text Converted to Numerical Features**: The Bag of Words method transformed the text into a numeric matrix, where each unique word across all messages represents a feature (or column). This provides a consistent structure for model input.
2. **Binary Labels Created**: The `y` variable was created as a binary label indicating whether each message is spam (`1`) or ham (`0`). This enables a straightforward classification framewor k.
3. **Independent Variables (x)**: With BOW applied, `x` contains only relevant word frequencies, allowing the model to focus on word patterns that distinguish spam from ham.
4. **Dataset Prepared for Model Training**: The dataset is now simplified, containing only meaningful numeric data in `x` (from BOW) and binary class labels in `y` , ready for training classification algorithms like Naive Bayes or Decision Trees to identify spam.

Train Test Split

In [197…
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.20)
```

In [198…
```
y_train.shape
```

Out[198…
```
(4125,)
```

Apply the ML Model (Naive Bayes)

In [199…
```
from sklearn.naive_bayes import MultinomialNB
spam_detect_model = MultinomialNB()
spam_detect_model.fit(X_train,y_train)
```

Out[199…
```
MultinomialNB
MultinomialNB()
```

In [200…
```
y_pred = spam_detect_model.predict(X_test)
```

Classification Matrics for (Naive Bayes)

In [201…
```
from sklearn.metrics import accuracy_score,classification_report
```

In [202…
```
accuracy_score(y_test,y_pred)
```

Out[202… 0.9854651162790697

In [203…
```
score = accuracy_score(y_test,y_pred) *100
score
```

Out[203… 98.54651162790698

In [204…
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.99      0.99      0.99       893
           1       0.96      0.93      0.95       139

    accuracy                           0.99      1032
   macro avg       0.98      0.96      0.97      1032
weighted avg       0.99      0.99      0.99      1032
```

ML Model (Decision Tree)

In [205…
```
from sklearn.tree import DecisionTreeClassifier
# The criterion='entropy' is similar to the information gain measure used in J48
j48_model = DecisionTreeClassifier(criterion='entropy')
j48_model.fit(X_train, y_train)
```

Out[205…
```
DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy')
```

In [206…
```
y_pred_j48 = j48_model.predict(X_test)
```

Classification Matrics for (Decision Tree)

In [207…
```
accuracy_score(y_test,y_pred_j48)
```

Out[207… 0.9680232558139535

In [208…
```
score_dt = accuracy_score(y_test,y_pred_j48) *100
score_dt
```

Out[208… 96.80232558139535

In [209…
```
print(classification_report(y_test,y_pred_j48))
```

```
              precision    recall  f1-score   support

           0       0.98      0.98      0.98       893
           1       0.89      0.87      0.88       139

    accuracy                           0.97      1032
   macro avg       0.93      0.93      0.93      1032
weighted avg       0.97      0.97      0.97      1032
```

Test Model For New Data

In [210…
```
# Sample new email
new_email = "Congratulations! You've won a $1000 gift card. Click here to claim you
# 1. Preprocess the email
ps = PorterStemmer()
# Remove all the special characters and keep only letters
message = re.sub("[^a-zA-Z]", " ", new_email)
# Convert to lowercase
message = message.lower()
# Split the text into individual words
message = message.split()
# Remove stopwords and apply stemming
message = [ps.stem(word) for word in message if word not in stopwords.words('englis
# Join the words back into a single string
message = ' '.join(message)
# 2. Transform the preprocessed text using the trained CountVectorizer
email_bow = cv.transform([message]).toarray()
# 3. Make predictions using Naive Bayes and Decision Tree models
nb_prediction = spam_detect_model.predict(email_bow)
# Print the results
print("Naive Bayes Prediction:", "Spam" if nb_prediction[0] == 1 else "Ham")
```

```
Naive Bayes Prediction: Spam
```

Final Conclusion

Project Overview

- This project developed a system to automatically classify emails as "spam" or "ham" (non-spam), addressing the widespread issue of unwanted spam messages. The solution leverages machine learning techniques to effectively identify spam emails.

Steps Undertaken

1. **Data Collection**
    - Gathered a dataset containing both spam and non-spam messages for training and evaluation.
2. **Data Preprocessing**
    - **Noise Removal**: Removed non-letter characters like special characters and numbers.
    - **Text Standardization**: Converted all text to lowercase and removed common stopwords.
    - **Stemming**: Reduced words to their root forms to enhance data consistency.
    - **Bag of Words (BoW)**: Used `CountVectorizer` with unigrams and bigrams to transform text data into a structured numeric format, selecting up to 2500 features.
3. **Feature Selection**
    - Extracted relevant word patterns and structured them using BoW for meaningful model training.
4. **Model Selection**
    - **Algorithms Tested**:
        - **Naive Bayes (Multinomial)**: Known for high performance with text data.
        - **Decision Tree (J48)**: A robust choice for classification tasks, offering interpretability.
    - **Training and Testing**: Data was split into 70% training and 30% testing sets.
    - **Final Model Choice**:
        - Naive Bayes achieved **99% accuracy**, outperforming the Decision Tree model, which reached **96% accuracy**.
        - The Naive Bayes model was selected as the primary classifier due to its:
            - Superior accuracy on text data
            - Efficiency in handling high-dimensional BoW features
            - Faster computation, ideal for real-time filtering tasks
5. **Performance Evaluation**
    - Assessed each model using **accuracy scores** and **confusion matrices**.
    - Naive Bayes was chosen for its high accuracy and efficiency, making it optimal for text-based spam classification.

Final Results

- **Chosen Model**: Naive Bayes (Multinomial) was selected as the primary model for email classification.
- **Accuracy**: Achieved an impressive 99% accuracy with Naive Bayes on test data.

- **Conclusion**: This project successfully applied machine learning to spam detection, achieving high classification accuracy. Naive Bayes emerged as the most suitable model for this task, providing a practical and reliable solution for real-world spam filtering.