

► CONTENTS

0. Data

1. Preprocessing

2. Model selection - 3 best types & SVM

3. Predict which hosts are likely to be 'Superhosts'

0. Data

(1) Import "listing.csv.gz"

```
In [48]: import pandas as pd
import numpy as np
from sklearn.utils import resample
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt
from sklearn import metrics
import seaborn as sns
```

```
In [49]: pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
```

```
In [50]: listing = pd.read_csv('http://data.insideairbnb.com/spain/comunidad-de-madrid/madr
listing.head(2)
```

```
Out[50]:
```

	id	listing_url	scrape_id	last_scraped	source	name	desc
0	6369	https://www.airbnb.com/rooms/6369	20220911230855	2022-09-12	city scrape	Rooftop terrace room , ensuite bathroom	En con \n All and
1	21853	https://www.airbnb.com/rooms/21853	20220911230855	2022-09-12	city scrape	Bright and airy room	We ql sunr with

(2) Drop Useless Variables

we wanted to carry as many values as possible,
so we dropped some columns that has too many categorical values and are not included in test.xlsx.

```
listing_drop = listing[['host_is_superhost', 'host_response_time', 'host_response_rate', 'host_acceptance_rate',  
                        'host_listings_count', 'host_total_listings_count', 'host_has_profile_pic', 'host_identity_verified',  
                        'accommodates', 'bathrooms', 'bedrooms', 'beds', 'price',  
                        'minimum_nights', 'maximum_nights', 'minimum_minimum_nights', 'maximum_minimum_nights',  
                        'minimum_maximum_nights', 'maximum_maximum_nights', 'minimum_nights_avg_ntm', 'maximum_nights_avg_ntm',  
                        'has_availability', 'availability_30', 'availability_60', 'availability_90', 'availability_365',  
                        'calendar_last_scraped', 'number_of_reviews', 'number_of_reviews_ltm', 'number_of_reviews_l30d',  
                        'first_review', 'last_review', 'review_scores_rating', 'review_scores_accuracy',  
                        'review_scores_cleanliness', 'review_scores_checkin', 'review_scores_communication',  
                        'review_scores_location', 'review_scores_value', 'instant_bookable', 'calculated_host_listings_count',  
                        'calculated_host_listings_count_entire_homes', 'calculated_host_listings_count_private_rooms',  
                        'calculated_host_listings_count_shared_rooms', 'reviews_per_month' ]]
```

(3) Convert Categorical to Numerical

```
In [6]: listing_drop['host_is_superhost'].replace(['f', 't'],  
                                                  [0,1], inplace=True)

listing_drop['host_has_profile_pic'].replace(['f', 't'],  
                                              [0,1], inplace=True)

listing_drop['host_identity_verified'].replace(['f', 't'],  
                                               [0,1], inplace=True)

listing_drop['has_availability'].replace(['f', 't'],  
                                         [0,1], inplace=True)

listing_drop['instant_bookable'].replace(['f', 't'],  
                                         [0,1], inplace=True)

listing_drop['host_response_time'].replace(['within a day', 'within a few hours', 'within an hour', 'a few days or more'],  
                                           [0,1,2,3], inplace=True)

listing_drop.head(2)
```


(4) Change Datatype

	column name	dtypes
0	host_is_superhost	float64
1	host_response_time	float64
2	host_response_rate	object
3	host_acceptance_rate	object
4	host_listings_count	float64
5	host_total_listings_count	float64
6	host_has_profile_pic	float64
7	host_identity_verified	float64
8	accommodates	int64
9	bathrooms	float64
10	bedrooms	float64
11	beds	float64
12	price	object
13	minimum_nights	int64
14	maximum_nights	int64
15	minimum_minimum_nights	int64
16	maximum_minimum_nights	int64
17	minimum_maximum_nights	int64
18	maximum_maximum_nights	int64
19	minimum_nights_avg_ntm	float64
20	maximum_nights_avg_ntm	float64
21	has_availability	int64
22	availability_30	int64
23	availability_60	int64
24	availability_90	int64
25	availability_365	int64
26	calendar_last_scraped	object
27	number_of_reviews	int64
28	number_of_reviews_ltm	int64
29	number_of_reviews_l30d	int64
30	first_review	object
31	last_review	object
32	review_scores_rating	float64
33	review_scores_accuracy	float64
34	review_scores_cleanliness	float64
35	review_scores_checkin	float64
36	review_scores_communication	float64
37	review_scores_location	float64
38	review_scores_value	float64
39	instant_bookable	int64
40	calculated_host_listings_count	int64
41	calculated_host_listings_count_entire_homes	int64
42	calculated_host_listings_count_private_rooms	int64
43	calculated_host_listings_count_shared_rooms	int64
44	reviews_per_month	float64

object ► float



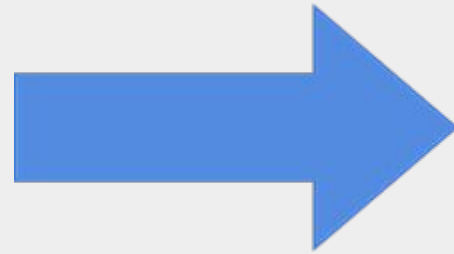
- delete (%) of 'host_response_rate', 'host_acceptance_rate'
- delete (\$) of 'price'
- convert the type of value, from datetime to float, 'calendar_last_scraped', 'first_review', and 'last_review'

	column name	dtypes
0	host_is_superhost	float64
1	host_response_time	float64
2	host_response_rate	float64
3	host_acceptance_rate	float64
4	host_listings_count	float64
5	host_total_listings_count	float64
6	host_has_profile_pic	float64
7	host_identity_verified	float64
8	accommodates	int64
9	bathrooms	float64
10	bedrooms	float64
11	beds	float64
12	price	float64
13	minimum_nights	int64
14	maximum_nights	int64
15	minimum_minimum_nights	int64
16	maximum_minimum_nights	int64
17	minimum_maximum_nights	int64
18	maximum_maximum_nights	int64
19	minimum_nights_avg_ntm	float64
20	maximum_nights_avg_ntm	float64
21	has_availability	int64
22	availability_30	int64
23	availability_60	int64
24	availability_90	int64
25	availability_365	int64
26	number_of_reviews	int64
27	number_of_reviews_ltm	int64
28	number_of_reviews_l30d	int64
29	review_scores_rating	float64
30	review_scores_accuracy	float64
31	review_scores_cleanliness	float64
32	review_scores_checkin	float64
33	review_scores_communication	float64
34	review_scores_location	float64
35	review_scores_value	float64
36	instant_bookable	int64
37	calculated_host_listings_count	int64
38	calculated_host_listings_count_entire_homes	int64
39	calculated_host_listings_count_private_rooms	int64
40	calculated_host_listings_count_shared_rooms	int64
41	reviews_per_month	float64
42	calendar_last_scraped_days_past	float64
43	first_review_days_past	float64
44	last_reviews_days_past	float64

(5) Check missing values

host_is_superhost	5
host_response_time	4889
host_response_rate	4889
host_acceptance_rate	4401
host_listings_count	3
host_total_listings_count	3
host_has_profile_pic	3
host_identity_verified	3
accommodates	0
bathrooms	20681
bedrooms	1454
beds	350
price	0
minimum_nights	0
maximum_nights	0
minimum_minimum_nights	0
maximum_minimum_nights	0
minimum_maximum_nights	0
maximum_maximum_nights	0
minimum_nights_avg_ntm	0
maximum_nights_avg_ntm	0
has_availability	0
availability_30	0
availability_60	0
availability_90	0
availability_365	0
number_of_reviews	0
number_of_reviews_ltm	0
number_of_reviews_l30d	0
review_scores_rating	4462
review_scores_accuracy	4612
review_scores_cleanliness	4610
review_scores_checkin	4611
review_scores_communication	4611
review_scores_location	4614
review_scores_value	4615
instant_bookable	0
calculated_host_listings_count	0
calculated_host_listings_count_entire_homes	0
calculated_host_listings_count_private_rooms	0
calculated_host_listings_count_shared_rooms	0
reviews_per_month	4462
calendar_last_scraped_days_past	0
first_review_days_past	4462
last_reviews_days_past	4462
dtype: int64	

fill the missing values
using the mean of each column



host_is_superhost	5
host_response_time	0
host_response_rate	0
host_acceptance_rate	0
host_listings_count	0
host_total_listings_count	0
host_has_profile_pic	0
host_identity_verified	0
accommodates	0
bathrooms	20681
bedrooms	0
beds	0
price	0
minimum_nights	0
maximum_nights	0
minimum_minimum_nights	0
maximum_minimum_nights	0
minimum_maximum_nights	0
maximum_maximum_nights	0
minimum_nights_avg_ntm	0
maximum_nights_avg_ntm	0
has_availability	0
availability_30	0
availability_60	0
availability_90	0
availability_365	0
number_of_reviews	0
number_of_reviews_ltm	0
number_of_reviews_l30d	0
review_scores_rating	0
review_scores_accuracy	0
review_scores_cleanliness	0
review_scores_checkin	0
review_scores_communication	0
review_scores_location	0
review_scores_value	0
instant_bookable	0
calculated_host_listings_count	0
calculated_host_listings_count_entire_homes	0
calculated_host_listings_count_private_rooms	0
calculated_host_listings_count_shared_rooms	0
reviews_per_month	0
calendar_last_scraped_days_past	0
first_review_days_past	0
last_reviews_days_past	0
dtype: int64	

drop null value in
"host_is_superhost"

"bathroom" has any value,
so drop it

1. Preprocessing

(1) Balancing the Data

```
listing_train["host_is_superhost"].value_counts()
```

```
0.0    16912  
1.0     3764  
Name: host_is_superhost, dtype: int64
```

```
listing_train0 = listing_train[listing_train["host_is_superhost"] == 0]  
listing_train1 = listing_train[listing_train["host_is_superhost"] == 1]
```

```
listing_train1_upsampled = resample(listing_train1, replace=True, n_samples=16912, random_state=0)
```

```
listing_train_upsampled = pd.concat([listing_train0, listing_train1_upsampled], axis=0)
```

```
listing_train_upsampled["host_is_superhost"].value_counts()
```

```
0.0    16912  
1.0    16912  
Name: host_is_superhost, dtype: int64
```

(2) Defining Y & X

```
y = listing_train.iloc[:, 0]  
x = listing_train.iloc[:, 1:]
```

(3) Scaling

```
scaler = StandardScaler()
```

```
scaler_model = scaler.fit(x)
```

```
scaled_x = scaler_model.transform(x)
```


2. Model

1) Choose optimal models : We checked the accuracy score of five models as below

```
models = []

lr = LogisticRegression()
models.append(("LR", lr))

dtc = DecisionTreeClassifier(criterion = "entropy", max_depth = 5) #or giny
models.append(("DT", dtc))

rfc = RandomForestClassifier(n_estimators = 20, random_state = 0)
models.append(("RF", rfc))

# y is binary output is one , 9+1/2 = lead into rule of thumb
nnc = MLPClassifier(activation = "relu", hidden_layer_sizes = (6), max_iter = 1200000, random_state = 0 )
models.append(("ANN", nnc))

svm = SVC(kernel = "rbf", C = 1, gamma = 1)
models.append(("SVM", svm))
```

models

```
[('LR', LogisticRegression()),
 ('DT', DecisionTreeClassifier(criterion='entropy', max_depth=5)),
 ('RF', RandomForestClassifier(n_estimators=20, random_state=0)),
 ('ANN',
  MLPClassifier(hidden_layer_sizes=6, max_iter=1200000, random_state=0)),
 ('SVM', SVC(C=1, gamma=1))]
```

```
for name, model in models:
    scores = cross_val_score(model, scaled_x, y, cv=5)
    mean = np.mean(scores)
    std = np.std(scores)
    print(name, ": ", mean, "(", std, ")")
```

```
LR : 0.8476740791700523 ( 0.026577979124396908 )
DT : 0.8713288744823503 ( 0.013741689597247937 )
RF : 0.8557054397162996 ( 0.023832185251830285 )
ANN : 0.868523086049038 ( 0.016462422461778998 )
SVM : 0.820345324027193 ( 0.0020590615877942556 )
```

► **Decision Tree model,
Random Forest model,
Artificial Neural Network model
showed good validation scores.**

► **So we made prediction much deeper using these models.**

2) Optimal Model Analysis

Then we split the train set and test set first.

```
x_train, x_test, y_train, y_test = train_test_split(scaled_x, y, test_size = 0.3, random_state = 0)
```

2) Optimal Model Analysis

(1) Decision Tree Model

we can find highest accuracy score when criterion is gini, max depth = 9.

■ criterion = "entropy"

```
nums = range(1,44)
for n in nums:
    dtc = DecisionTreeClassifier(criterion="entropy", max_depth = n)
    dtc_model = dtc.fit(x_train, y_train)
    y_pred = dtc_model.predict(x_test)
    score = metrics.accuracy_score(y_test, y_pred)
    print(n, ": ", score)
```

```
1 : 0.8181525068515234
2 : 0.8392713203288731
3 : 0.8644204417217476
4 : 0.8626471062389166
5 : 0.872803482186039
6 : 0.8790907625342577
7 : 0.8810253103337095
8 : 0.8803804610672256
9 : 0.8798968241173626
10 : 0.8803804610672256
11 : 0.8773174270514267
12 : 0.8776398516846687
13 : 0.8742543930356279
14 : 0.8694180235369983
15 : 0.8665162018378204
16 : 0.8632919555054006
17 : 0.8620022569724327
18 : 0.8618410446558117
19 : 0.8628083185555376
20 : 0.8674834757375464
21 : 0.8624858939222957
```

■ criterion = "gini"

```
nums = range(1,44)
for n in nums:
    dtc = DecisionTreeClassifier(criterion="gini", max_depth = n)
    dtc_model = dtc.fit(x_train, y_train)
    y_pred = dtc_model.predict(x_test)
    score = metrics.accuracy_score(y_test, y_pred)
    print(n, ": ", score)
```

```
1 : 0.8181525068515234
2 : 0.8479767854264065
3 : 0.8658713525713364
4 : 0.8695792358536192
5 : 0.872319845236176
6 : 0.8787683379010156
7 : 0.8800580364339835
8 : 0.8790907625342577
9 : 0.8818313719168145
10 : 0.8803804610672256
11 : 0.8771562147348058
12 : 0.8768337901015638
13 : 0.8753828792519749
14 : 0.8636143801386426
15 : 0.8645816540383685
16 : 0.8626471062389166
17 : 0.8644204417217476
18 : 0.8613574077059487
19 : 0.8591004352732549
20 : 0.8599064968563598
```


2) Optimal Model Analysis

(1) Decision Tree Model

we can find highest accuracy score when criterion is gini, max depth = 9.

CHOOSE IT

■ criterion = "entropy"

```
nums = range(1,44)
for n in nums:
    dtc = DecisionTreeClassifier(criterion="entropy", max_depth = n)
    dtc_model = dtc.fit(x_train, y_train)
    y_pred = dtc_model.predict(x_test)
    score = metrics.accuracy_score(y_test, y_pred)
    print(n, ": ", score)
```

```
1 : 0.8181525068515234
2 : 0.8392713203288731
3 : 0.8644204417217476
4 : 0.8626471062389166
5 : 0.872803482186039
6 : 0.8790907625342577
7 : 0.8810253103337095
8 : 0.8803804610672256
9 : 0.8798968241173626
10 : 0.8803804610672256
11 : 0.8773174270514267
12 : 0.8776398516846687
13 : 0.8742543930356279
14 : 0.8694180235369983
15 : 0.8665162018378204
16 : 0.8632919555054006
17 : 0.8620022569724327
18 : 0.8618410446558117
19 : 0.8628083185555376
20 : 0.8674834757375464
21 : 0.8624858939222957
```

■ criterion = "gini"

```
nums = range(1,44)
for n in nums:
    dtc = DecisionTreeClassifier(criterion="gini", max_depth = n)
    dtc_model = dtc.fit(x_train, y_train)
    y_pred = dtc_model.predict(x_test)
    score = metrics.accuracy_score(y_test, y_pred)
    print(n, ": ", score)
```

```
1 : 0.8181525068515234
2 : 0.8479767854264065
3 : 0.8658713525713364
4 : 0.8695792358536192
5 : 0.872319845236176
6 : 0.8787683379010156
7 : 0.8800580364339835
8 : 0.8790907625342577
9 : 0.8818313719168145
10 : 0.8803804610672256
11 : 0.8771562147348058
12 : 0.8768337901015638
13 : 0.8753828792519749
14 : 0.8636143801386426
15 : 0.8645816540383685
16 : 0.8626471062389166
17 : 0.8644204417217476
18 : 0.8613574077059487
19 : 0.8591004352732549
20 : 0.8599064968563598
```


(1) Decision Tree Model

- Run the best model

- criterion = "gini"
- max_depth = 9



accuracy score : 0.8811

```
dtc_high = DecisionTreeClassifier(criterion="gini", max_depth = 9)
```

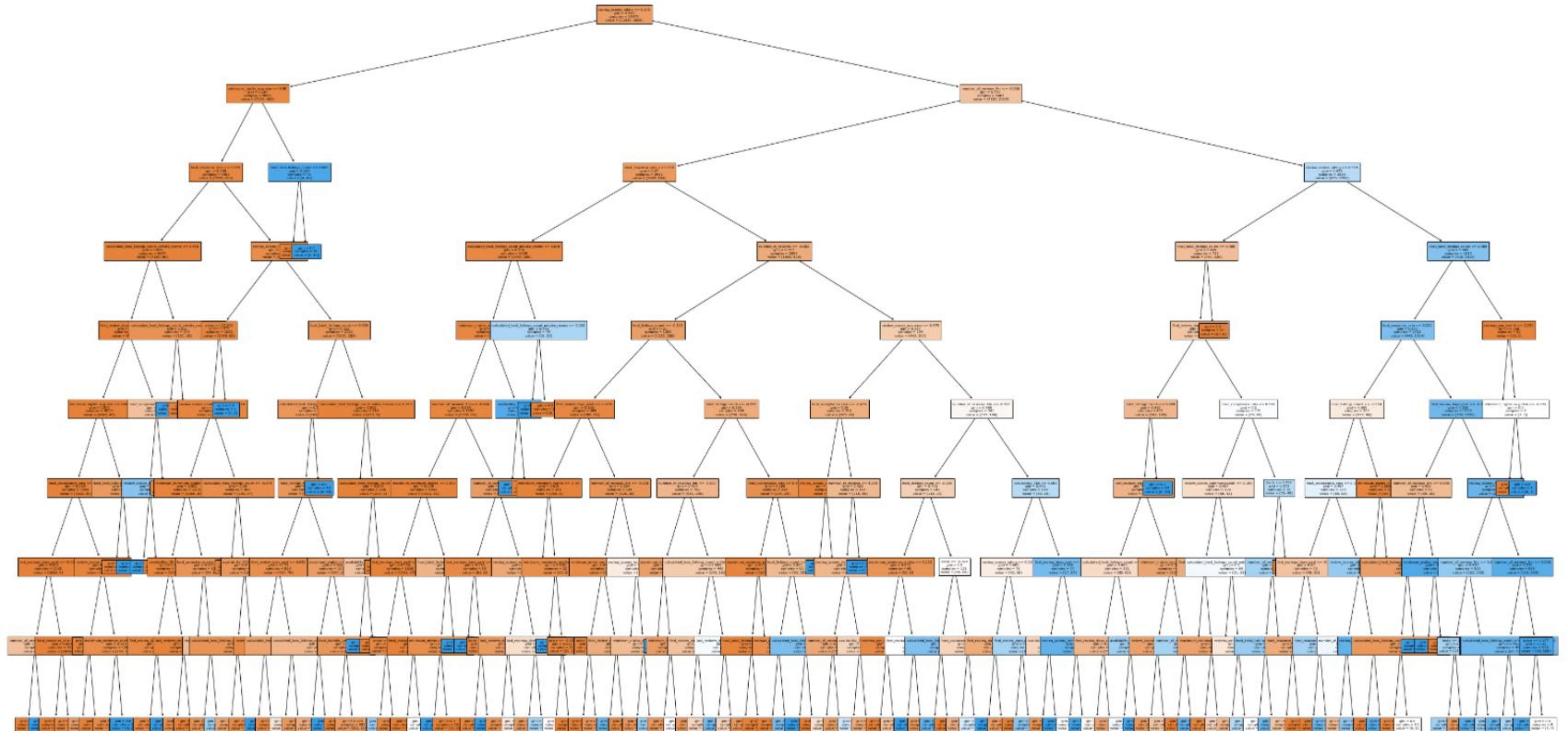
```
dtc_high_model = dtc_high.fit(x_train, y_train)
```

```
y_pred_dtc = dtc_high_model.predict(x_test)
```

```
metrics.accuracy_score(y_test, y_pred_dtc)
```

```
0.8811865226503305
```

(1) Decision Tree - Visualization



(1) Decision Tree

we can find highest accuracy score when criterion is gini, max depth = 9.

- Test : confusion matrix model

```
print(metrics.classification_report(y_test, y_pred_dtc))
```

	precision	recall	f1-score	support
0.0	0.91	0.91	0.91	5059
1.0	0.60	0.59	0.59	1143
accuracy			0.85	6202
macro avg	0.75	0.75	0.75	6202
weighted avg	0.85	0.85	0.85	6202

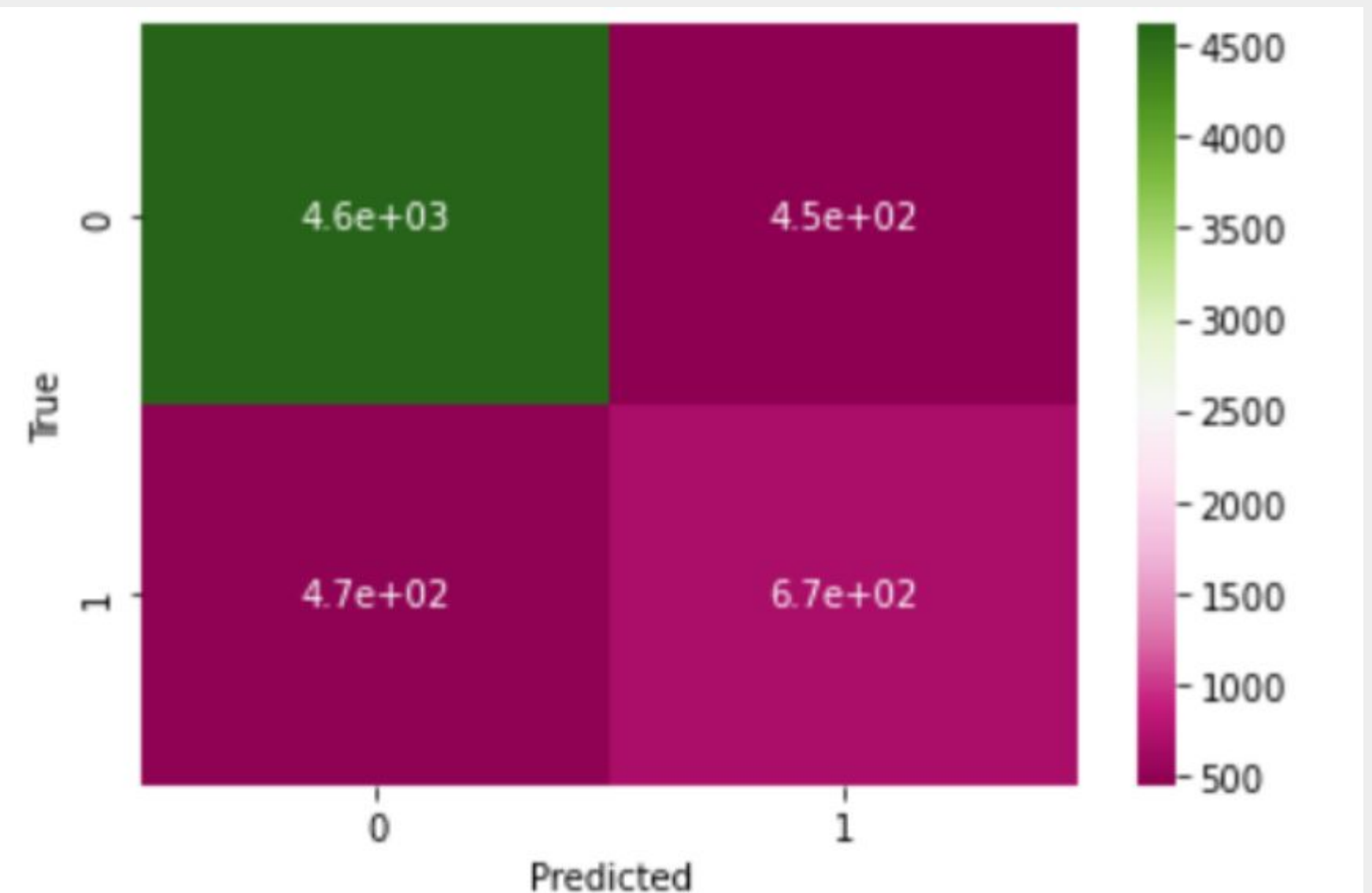
```
metrics.confusion_matrix(y_test,y_pred)
```

```
array([[4612,  447],  
       [ 471,  672]], dtype=int64)
```

```
cfm = pd.DataFrame(metrics.confusion_matrix(y_test,y_pred))
```

```
cfm.index.name = "True"  
cfm.columns.name = "Predicted"
```

- Visualization



2) Optimal Model Analysis

(2) Random Forest model : First Trying

- criterion = "entropy"
- the number of estimators = 20
- max_depth = 5



accuracy score : 0.8715

```
rfc_high = RandomForestClassifier(criterion = "entropy", n_estimators = 20, random_state = 0, max_depth = 5)
```

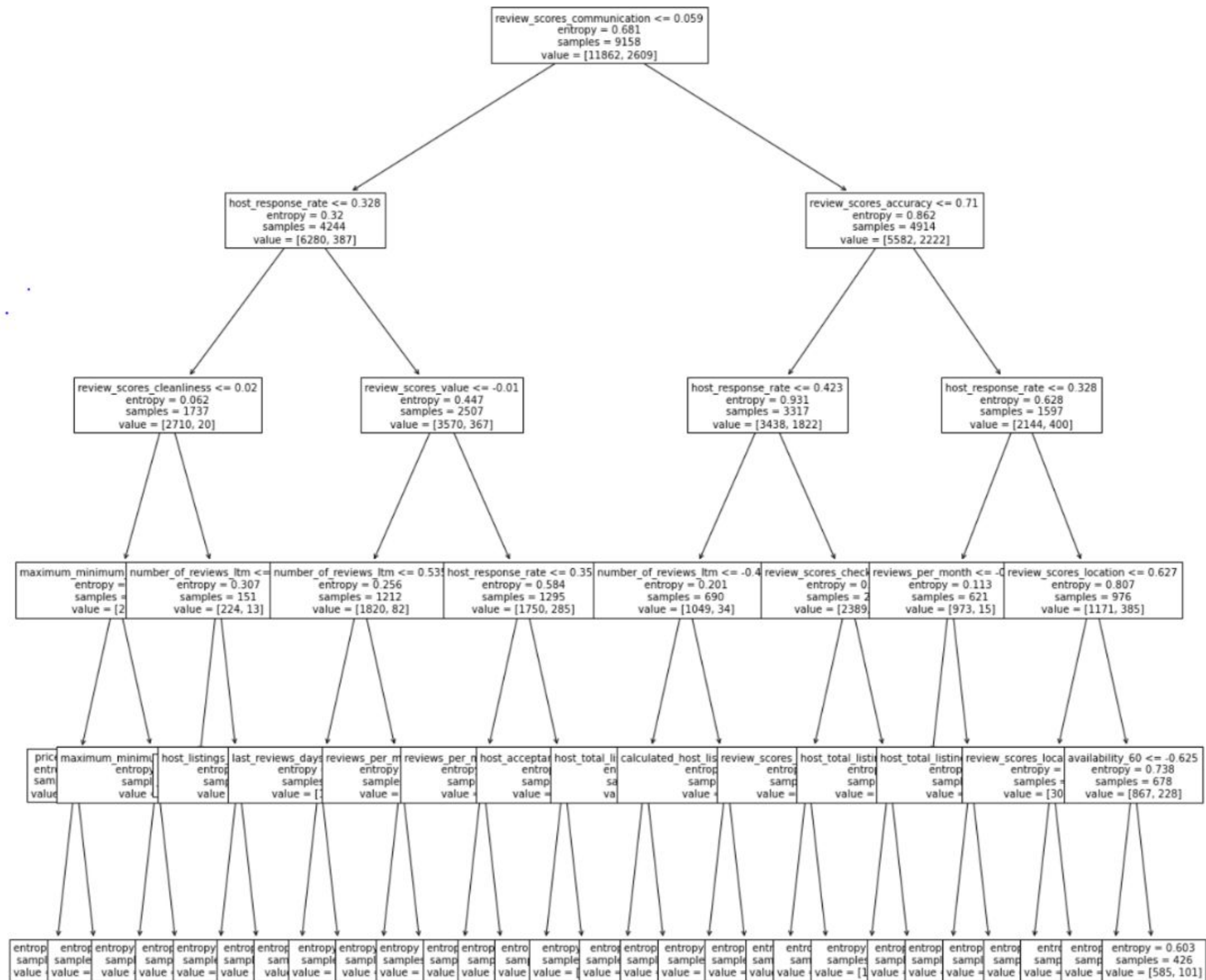
```
rfc_high_model = rfc_high.fit(x_train, y_train)
```

```
y_pred_rf = rfc_high_model.predict(x_test)
```

```
metrics.accuracy_score(y_pred_rf, y_test)
```

```
0.8715137836530711
```


(2) Random Forest model : Visualization of First Trying



(2) Random Forest model

- Check the Feature Importance


Feature Importance	
host_response_time	0.000647
host_response_rate	0.175841
host_acceptance_rate	0.065419
host_listings_count	0.005839
host_total_listings_count	0.015308
host_has_profile_pic	0.000000
host_identity_verified	0.000000
accommodates	0.000000
bedrooms	0.001301
beds	0.000000
price	0.001799
minimum_nights	0.001839
maximum_nights	0.000423
minimum_minimum_nights	0.001675
maximum_minimum_nights	0.002959
minimum_maximum_nights	0.000959
maximum_maximum_nights	0.000292
minimum_nights_avg_ntm	0.005607
maximum_nights_avg_ntm	0.001798
has_availability	0.000000
availability_30	0.001034
availability_60	0.002302
availability_90	0.000000
availability_365	0.010566

number_of_reviews	0.064006
number_of_reviews_ltm	0.040040
number_of_reviews_l30d	0.000000
review_scores_rating	0.093123
review_scores_accuracy	0.095497
review_scores_cleanliness	0.085418
review_scores_checkin	0.067935
review_scores_communication	0.088412
review_scores_location	0.011480
review_scores_value	0.059575
instant_bookable	0.000776
calculated_host_listings_count	0.007318
calculated_host_listings_count_entire_homes	0.004664
calculated_host_listings_count_private_rooms	0.005410
calculated_host_listings_count_shared_rooms	0.000000
reviews_per_month	0.002350
calendar_last_scraped_days_past	0.000000
first_review_days_past	0.001702
last_reviews_days_past	0.076685

- Find the best model

```
nums = range(1,30)
for n in nums:
    rfc = RandomForestClassifier(criterion = "entropy", n_estimators = 30, random_state = 0, max_features = n)
    rfc_model = rfc.fit(x_train,y_train)
    y_pred_rfc = rfc_model.predict(x_test)
    score = metrics.accuracy_score(y_test,y_pred_rfc)
    print(n, score)
```

1	0.8937439535633667
2	0.8958400515962592
3	0.8968074814575944
4	0.8951950983553693
5	0.8960012899064818
6	0.8960012899064818
7	0.9009996775233795
8	0.8979361496291519
9	0.9006772009029346
10	0.8961625282167043
11	0.8979361496291519
12	0.8951950983553693
13	0.8990648178007095
14	0.8989035794904869
15	0.9016446307642696
16	0.8971465419958085
17	0.8995647267451233
18	0.8984362405287764
19	0.9014992745445752
20	0.8960180557794616
21	0.9002095760116073
22	0.8984362405287764
23	0.8960180557794616
24	0.8966629050459455
25	0.8992423021118814
26	0.8971465419958085
27	0.8992423021118814
28	0.8956956311462195
29	0.8955344188295986



**#19 is best option
for max_features**

(2) Random Forest model

- Run the best model

- criterion = "entropy"
- the number of estimators = 33
- max_features = 19



Accuracy score : 0.9019

```
rfc_high2 = RandomForestClassifier(criterion = "entropy", n_estimators = 33, random_state = 0, max_features = 19)
```

```
rfc_high_model2 = rfc_high2.fit(x_train, y_train)
```

```
y_pred_rf2 = rfc_high_model2.predict(x_test)
```

```
metrics.accuracy_score(y_pred_rf2, y_test)
```

```
0.9019829114944382
```

2) Optimal Model Analysis

(3) Artificial Neural Network (ANN)

■ single layer

```
sizes = range(1,30)

for s in sizes:
    mlp = MLPClassifier(activation = "relu", hidden_layer_sizes = (s), max_iter = 3000, random_state=0)
    scores = cross_val_score(mlp,scaled_x,y, cv = 5)
    mean = np.mean(scores)
    std = np.std(scores)
    print(s, ":", mean, "(", std, ")")
```

1 : 0.8479163032369665	15 : 0.8700225516538171
2 : 0.8401268742588813	16 : 0.8675070272825286
3 : 0.8715707943505621	17 : 0.8645568497650358
4 : 0.8745703222575756	18 : 0.863493312601000
5 : 0.8664911791151211	19 : 0.8566720541111177
6 : 0.868523086049038	20 : 0.8597181111111177
7 : 0.872344863049159	21 : 0.8597181111111177
8 : 0.8727800192932177	22 : 0.8665884876001003
9 : 0.8700224697541665	23 : 0.8658622248976109
10 : 0.86257319342533	24 : 0.8677977008428058
11 : 0.8707481942589516	25 : 0.8621380722811217
12 : 0.8677972445447522	26 : 0.8663950172252516
13 : 0.8632978181348057	27 : 0.8575432678779624
14 : 0.866394841726	28 : 0.8590907500779508
	29 : 0.8609777765297831

**8 is best option
of hidden layer sizes.**

■ multiple layer (3-hidden-layer)

```
sizes2 = range(1,6)

for d in sizes2:
    for a in sizes2:
        for s in sizes2:
            mlp = MLPClassifier(activation = "relu", hidden_layer_sizes = (d,a,s), max_iter = 3000, random_state=0)
            scores = cross_val_score(mlp,scaled_x,y, cv = 5)
            mean = np.mean(scores)
            std = np.std(scores)
            print(d," ",a," ",s, ":", mean, "(", std, ")")
```

1 , 1 , 1 : 0.8179267688423308	4 , 1 , 1 : 0.8660557303723099
1 , 1 , 2 : 0.8179267688423308	4 , 1 , 2 : 0.869926296164347
1 , 1 , 3 : 0.8179267688423308	4 , 1 , 3 : 0.8740378809284378
1 , 1 , 4 : 0.8469973657562349	4 , 1 , 4 : 0.8668299862701085
1 , 1 , 5 : 0.8472876181183088	4 , 1 , 5 : 0.8678461737360692
1 , 2 , 1 : 0.830453975613794	4 , 2 , 1 : 0.8719098004047015
1 , 2 , 2 : 0.8445785648724208	4 , 2 , 2 : 0.8767953368678884
1 , 2 , 3 : 0.8468523916745495	4 , 2 , 3 : 0.8751026465872123
1 , 2 , 4 : 0.8458847122017025	4 , 2 , 4 : 0.8730702833552415
1 , 2 , 5 : 0.8466588511000002	4 , 2 , 5 : 0.8719579339994116
1 , 3 , 1 : 0.8179267688423308	4 , 3 , 1 : 0.8710869312142384
1 , 3 , 2 : 0.8489323152036757	4 , 3 , 2 : 0.8679916392156588
1 , 3 , 3 : 0.8476746290677071	4 , 3 , 3 : 0.871716096030850
1 , 3 , 4 : 0.8179267688423308	4 , 3 , 4 : 0.8728285857860
1 , 3 , 5 : 0.8457393871215139	4 , 3 , 5 : 0.863539562509
1 , 4 , 1 : 0.8453527856703691	4 , 4 , 1 : 0.8179267688423308
1 , 4 , 2 : 0.8179267688423308	4 , 4 , 2 : 0.873795820108247
1 , 4 , 3 : 0.8178784012486187	4 , 4 , 3 : 0.8730702833552415
1 , 4 , 4 : 0.848932350303526	4 , 4 , 4 : 0.8710869312142384
1 , 4 , 5 : 0.846368446638575	4 , 4 , 5 : 0.8751026465872123
1 , 5 , 1 : 0.8179267688423308	4 , 5 , 1 : 0.876262954038501
1 , 5 , 2 : 0.8179267688423308	4 , 5 , 2 : 0.877521377271326
1 , 5 , 3 : 0.8494158507413967	4 , 5 , 3 : 0.8712806355880891
1 , 5 , 4 : 0.8460298968824898	4 , 5 , 4 : 0.8698295141771221
1 , 5 , 5 : 0.8493193846528244	4 , 5 , 5 : 0.8700219666563122

**(4,5,2) is best option
of hidden layer sizes.**

(3) Artificial Neural Network (ANN)

- Run the best model

- activation = "relu"
- hidden layer sizes = (4,5,2)
- max_iter = 3000



accuracy score : 0.8805

```
mlp_high = MLPClassifier(activation = "relu", hidden_layer_sizes = (4,5,2), max_iter = 3000, random_state=0)
```

```
mlp_high_model = mlp_high.fit(x_train, y_train)
```

```
y_pred_mlp = mlp_high_model.predict(x_test)
```

```
metrics.accuracy_score(y_pred_mlp, y_test)
```


```
0.8805416733838465
```

(+) Support Vector Machine (SVM)

Regardless of the initial model accuracy score, we also checked the SVM model with the possibility of errors or variations of accuracy score in mind.


- Try the model

```
params = {  
    "C" : [0.001, 0.01, 1, 10],  
    "gamma" : [0.001, 0.01, 1, 10]  
}  
  
grid = GridSearchCV(SVC(kernel = "rbf"), params, cv = 2)  
  
grid_model = grid.fit(x_train, y_train)  
  
grid_model.best_params_  
{'C': 10, 'gamma': 0.01}  
  
grid_model.best_score_  
0.8823326617644891
```



- Run the best model

```
nlsvc = SVC(kernel = "rbf", C = 10, gamma = 0.01)  
  
nlsvc_model = nlsvc.fit(x_train, y_train)  
  
y_pred_svm = nlsvc_model.predict(x_test)  
  
metrics.accuracy_score(y_test, y_pred_svm)  
0.8798968241173626
```

- kernel = "rbf"
 - C = 10, gamma = 0.01
- 

accuracy score : 0.8798

► SUMMARY of Best Prediction Models

(1) Decision Tree Model

- criterion = "gini"
- max_depth = 9



accuracy score : 0.8811



(2) Random Forest model

- criterion = "entropy"
- the number of estimators = 33
- max_features = 19



accuracy score : 0.9019

(3) Artificial Neural Network (ANN)

- activation = "relu"
- hidden layer sizes = (4,5,2)
- max_iter = 3000



accuracy score : 0.8805

(4) Support Vector Machine (SVM)

- kernel = "rbf"
- C = 10, gamma = 0.01



accuracy score : 0.8798

► SUMMARY of Best Prediction Models



(2) Random Forest model

- criterion = "entropy"
- the number of estimators = 33
- max_features = 19



accuracy score : 0.9019

So we selected Random Forest as the best model to predict which hosts are likely to be Superhosts in the excel file, 'test.xlsx'.

► SUMMARY of Best Prediction Models

```
for name, model in models:  
    scores = cross_val_score(model, scaled_x, y, cv=5)  
    mean = np.mean(scores)  
    std = np.std(scores)  
    print(name, ": ", mean, "(" , std, ")")
```

```
LR : 0.8496811901983119 ( 0.02175474894787541 )  
DT : 0.8197010237183641 ( 0.07435945146535254 )  
RF : 0.8781203295919393 ( 0.016447052897851467 )  
ANN : 0.8727512577211567 ( 0.015221317956158605 )  
SVM : 0.8691239805500528 ( 0.017494865588051918 )
```

3. Predict which hosts are likely to be 'Superhosts'

- Apply the excel file, 'test.xlsx'

The data trimming process of the previous train data set was equally performed in the test data set as follows.

- Read the excel file and raw data set
- Drop useless columns
- Convert categorical variables to numerical ones
- Change datatype from 'object' to 'float'
- Fill the missing value using means of each column

```
test.head(2)
```

	host_is_superhost	host_response_time	host_response_rate	host_acceptance_rate	host_listings_count	host_total_listings_count	host_has_profile_pic	host_
0	NaN	2.0	100.0	72.0	1	1	1	
1	NaN	2.0	100.0	100.0	21	21	1	

3. Predict which hosts are likely to be 'Superhosts'

- Define Y & X

```
y_t = test.iloc[:, 0]  
x_t = test.iloc[:, 1:]
```

- Scale test X

```
scaler_model_t = scaler.fit(x_t)  
  
scaled_x_t = scaler_model.transform(x_t)
```

► Run the best prediction model : Random Forest

```
best = RandomForestClassifier(criterion = "entropy", n_estimators = 30, random_state = 0, max_features = 29)  
  
best_model = best.fit(scaled_x, y)  
  
y_pred_best = best_model.predict(scaled_x_t)  
  
test["host_is_superhost"] = y_pred_best
```

	host_is_superhost	host_response_time	host_response_rate	host_acceptance_rate	host_listings_count	host_total_listings_count	host_has_profile_pic	host_
0	0.0	2.0	100.0	72.0	1	1	1	
1	0.0	2.0	100.0	100.0	21	21	1	
2	1.0	2.0	100.0	100.0	6	6	1	

```
test['host_is_superhost'].replace([0, 1], ['f', 't'], inplace=True)
```

	host_is_superhost	host_response_time	host_response_rate	host_acceptance_rate	host_listings_count	host_total_listings_count	host_has_profile_pic	host_
0	f	2.0	100.0	72.0	1	1	1	
1	f	2.0	100.0	100.0	21	21	1	
2	t	2.0	100.0	100.0	6	6	1	

Finally...

we can predict which hosts are likely to be 'Superhosts' ~

f(not superhost) : 118 people
t(superhost) : 22 people

A	B	C	D	E	F	G
	host_is_superhost	host_response_time	host_response_rate	host_acceptance_rate	host_listings_count	host_total_listings_count
0	f	2	100	72	1	1
1	f	2	100	100	21	21
2	t	2	100	100	6	6
3	f	2	100	100	2	2
4	f	2	90	90	4	5
5	f	2	100	100	16	16
6	f	2	100	100	59	81
7	t	1	100	98	7	13
8	f	2	100	98	10	10
9	f	3	0	0	1	2
10	f	2	100	84	49	49
11	f	1	100	44	1	1
12	f	1.826923077	93.99038462	87.05454545	1	2
13	f	1.826923077	93.99038462	87.05454545	2	2
14	f	1.826923077	93.99038462	87.05454545	2	2
15	f	1.826923077	93.99038462	87.05454545	1	2
16	t	2	100	98	3	4
17	f	2	100	39	2	2
18	f	1.826923077	93.99038462	87.05454545	1	1
19	f	1.826923077	93.99038462	87.05454545	1	2
20	t	2	100	100	2	5
21	f	1.826923077	93.99038462	67	1	2
22	f	2	100	85	1	1
23	f	1.826923077	93.99038462	87.05454545	1	2
24	t	2	100	81	8	10
25	f	2	99	99	97	105
26	f	2	100	91	54	95
27	f	2	100	100	47	47