In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

In [ ]:
```
```

In [2]:
```
!pip install xgboost
```

```
Requirement already satisfied: xgboost in c:\users\shubham choudhary\anaconda3\lib\site-packages (1.5.2)
Requirement already satisfied: scipy in c:\users\shubham choudhary\anaconda3\lib\site-packages (from xgboost) (1.7.1)
Requirement already satisfied: numpy in c:\users\shubham choudhary\anaconda3\lib\site-packages (from xgboost) (1.20.3)
```

In [3]:
```
from sklearn.datasets import load_boston
boston = load_boston()
```

In [4]:
```
boston.keys()
```

```
dict_keys(['data', 'target', 'feature_names', 'DESCR', 'filename'])
```

In [5]:
```
print(boston.data.shape)
```

```
(506, 13)
```

In [6]:
```
print(boston.feature_names)
```

```
['CRIM' 'ZN' 'INDUS' 'CHAS' 'NOX' 'RM' 'AGE' 'DIS' 'RAD' 'TAX' 'PTRATIO'
 'B' 'LSTAT']
```

In [7]:
```
print(boston.DESCR)
```

```
.. _boston_dataset:

Boston house prices dataset
---------------------------

**Data Set Characteristics:**

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive. Median Value (attribute 14) is usually the target.

    :Attribute Information (in order):
        - CRIM     per capita crime rate by town
        - ZN       proportion of residential land zoned for lots over 25,000 sq.ft.
        - INDUS    proportion of non-retail business acres per town
        - CHAS     Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
        - NOX      nitric oxides concentration (parts per 10 million)
        - RM       average number of rooms per dwelling
        - AGE      proportion of owner-occupied units built prior to 1940
        - DIS      weighted distances to five Boston employment centres
        - RAD      index of accessibility to radial highways
        - TAX      full-value property-tax rate per $10,000
        - PTRATIO  pupil-teacher ratio by town
        - B        1000(Bk - 0.63)^2 where Bk is the proportion of black people by town
        - LSTAT    % lower status of the population
        - MEDV     Median value of owner-occupied homes in $1000's

    :Missing Attribute Values: None

    :Creator: Harrison, D. and Rubinfeld, D.L.

This is a copy of UCI ML housing dataset.
https://archive.ics.uci.edu/ml/machine-learning-databases/housing/ (https://archive.ics.uci.edu/ml/machine-learning-databases/housing/)


This dataset was taken from the StatLib library which is maintained at Carnegie Mellon University.

The Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedonic
prices and the demand for clean air', J. Environ. Economics & Management,
```

```
vol.5, 81-102, 1978.   Used in Belsley, Kuh & Welsch, 'Regression diagnostics
...', Wiley, 1980.   N.B. Various transformations are used in the table on
pages 244-261 of the latter.

The Boston house-price data has been used in many machine learning papers that address regression
problems.

.. topic:: References

   - Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influential Data and Sources of Collinearity', Wiley, 1980. 244-261.
   - Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning. In Proceedings on the Tenth International Conference of Machine
Learning, 236-243, University of Massachusetts, Amherst. Morgan Kaufmann.
```

In [8]:
```
data=pd.DataFrame(boston.data)
data
```

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33
...	...	...	...	...	...	...	...	...	...	...	...	...	...
501	0.06263	0.0	11.93	0.0	0.573	6.593	69.1	2.4786	1.0	273.0	21.0	391.99	9.67
502	0.04527	0.0	11.93	0.0	0.573	6.120	76.7	2.2875	1.0	273.0	21.0	396.90	9.08
503	0.06076	0.0	11.93	0.0	0.573	6.976	91.0	2.1675	1.0	273.0	21.0	396.90	5.64
504	0.10959	0.0	11.93	0.0	0.573	6.794	89.3	2.3889	1.0	273.0	21.0	393.45	6.48
505	0.04741	0.0	11.93	0.0	0.573	6.030	80.8	2.5050	1.0	273.0	21.0	396.90	7.88

506 rows × 13 columns

In [9]:
```
data.columns=boston.feature_names
```

In [10]:
```
data.head()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33

In [11]:
```
data['PRICE']=boston.target
```

In [12]:
```
data.head()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	PRICE
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2

In [13]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   CRIM     506 non-null    float64
 1   ZN       506 non-null    float64
 2   INDUS    506 non-null    float64
 3   CHAS     506 non-null    float64
 4   NOX      506 non-null    float64
 5   RM       506 non-null    float64
 6   AGE      506 non-null    float64
 7   DIS      506 non-null    float64
 8   RAD      506 non-null    float64
 9   TAX      506 non-null    float64
 10  PTRATIO  506 non-null    float64
 11  B        506 non-null    float64
 12  LSTAT    506 non-null    float64
 13  PRICE    506 non-null    float64
dtypes: float64(14)
memory usage: 55.5 KB
```

In [14]:
```
data.shape
```

```
(506, 14)
```

In [15]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   CRIM     506 non-null    float64
 1   ZN       506 non-null    float64
 2   INDUS    506 non-null    float64
 3   CHAS     506 non-null    float64
 4   NOX      506 non-null    float64
 5   RM       506 non-null    float64
 6   AGE      506 non-null    float64
 7   DIS      506 non-null    float64
 8   RAD      506 non-null    float64
 9   TAX      506 non-null    float64
 10  PTRATIO  506 non-null    float64
 11  B        506 non-null    float64
 12  LSTAT    506 non-null    float64
 13  PRICE    506 non-null    float64
dtypes: float64(14)
memory usage: 55.5 KB
```

In [16]:
```
data.describe()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATI
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000(
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901	3.795043	9.549407	408.237154	18.45553
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861	2.105710	8.707259	168.537116	2.164946
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.129600	1.000000	187.000000	12.6000(
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000	2.100175	4.000000	279.000000	17.4000(
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000	3.207450	5.000000	330.000000	19.0500(
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000	5.188425	24.000000	666.000000	20.2000(
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.126500	24.000000	711.000000	22.0000(

In [17]:
```
import xgboost as xgb
from sklearn.metrics import mean_squared_error
import warnings
warnings.filterwarnings('ignore')
```

In [18]:
```
X, y = data.iloc[:,:-1],data.iloc[:,-1]
```

In [19]:
```
X.head()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33

In [20]:
```
y.head()
```

```
0    24.0
1    21.6
2    34.7
3    33.4
4    36.2
Name: PRICE, dtype: float64
```

Now you will convert the dataset into an optimized data structure called Dmatrix that XGBoost supports and gives it acclaimed performance and efficiency gains

In [21]:
```
data_dmatrix=xgb.DMatrix(data=X,label=y)
```

In [22]:
```
data_dmatrix
```

```
<xgboost.core.DMatrix at 0x1628276ebb0>
```

In [23]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=123)
```

In [24]:
```
xg_reg = xgb.XGBRegressor(objective ='reg:linear', colsample_bytree = 0.3, learning_rate = 0.1,
                max_depth = 5, alpha = 10, n_estimators = 10)
```

In [25]:
```
xg_reg
```

```
XGBRegressor(alpha=10, base_score=None, booster=None, colsample_bylevel=None,
             colsample_bynode=None, colsample_bytree=0.3,
             enable_categorical=False, gamma=None, gpu_id=None,
             importance_type=None, interaction_constraints=None,
             learning_rate=0.1, max_delta_step=None, max_depth=5,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             n_estimators=10, n_jobs=None, num_parallel_tree=None,
             objective='reg:linear', predictor=None, random_state=None,
             reg_alpha=None, reg_lambda=None, scale_pos_weight=None,
             subsample=None, tree_method=None, validate_parameters=None,
             verbosity=None)
```

In [26]:
```
xg_reg.fit(X_train,y_train)
```

```
[11:18:39] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/objective/regression_obj.cu:188: reg:linear is now de
precated in favor of reg:squarederror.
```

```
XGBRegressor(alpha=10, base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=0.3, enable_categorical=False,
             gamma=0, gpu_id=-1, importance_type=None,
             interaction_constraints='', learning_rate=0.1, max_delta_step=0,
             max_depth=5, min_child_weight=1, missing=nan,
             monotone_constraints='()', n_estimators=10, n_jobs=8,
             num_parallel_tree=1, objective='reg:linear', predictor='auto',
             random_state=0, reg_alpha=10, reg_lambda=1, scale_pos_weight=1,
             subsample=1, tree_method='exact', validate_parameters=1,
             verbosity=None)
```

In [27]:
```
xgb.XGBRegressor()
```

```
XGBRegressor(base_score=None, booster=None, colsample_bylevel=None,
             colsample_bynode=None, colsample_bytree=None,
             enable_categorical=False, gamma=None, gpu_id=None,
             importance_type=None, interaction_constraints=None,
             learning_rate=None, max_delta_step=None, max_depth=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             n_estimators=100, n_jobs=None, num_parallel_tree=None,
             predictor=None, random_state=None, reg_alpha=None, reg_lambda=None,
             scale_pos_weight=None, subsample=None, tree_method=None,
             validate_parameters=None, verbosity=None)
```

In [28]:
```
preds = xg_reg.predict(X_test)
```

In [29]:
```
rmse = np.sqrt(mean_squared_error(y_test, preds))
print("RMSE: %f" % (rmse))
```

```
RMSE: 10.423243
```

In [30]:
```
params = {"objective":"reg:linear",'colsample_bytree': 0.3,'learning_rate': 0.1,
                'max_depth': 5, 'alpha': 10}
```

In [31]:
```
#number of boosting iterations
```

In [32]:
```
cv_results = xgb.cv(dtrain=data_dmatrix, params=params, nfold=3,
                    num_boost_round=50,early_stopping_rounds=10,metrics="rmse", as_pandas=True, seed=123)
```

```
[11:18:39] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/objective/regression_obj.cu:188: reg:linear is now deprecated in favor of reg:squarederror.
[11:18:39] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/objective/regression_obj.cu:188: reg:linear is now deprecated in favor of reg:squarederror.
[11:18:39] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/objective/regression_obj.cu:188: reg:linear is now deprecated in favor of reg:squarederror.
```

In [33]:
```
cv_results.head()
```

	train-rmse-mean	train-rmse-std	test-rmse-mean	test-rmse-std
0	21.750758	0.036152	21.765523	0.028849
1	19.778532	0.077649	19.830760	0.031760
2	18.052810	0.118632	18.157336	0.116038
3	16.458958	0.169189	16.623975	0.191413
4	15.074781	0.183545	15.254608	0.213612

In [34]:
```
print((cv_results["test-rmse-mean"]).tail(1))
```

```
49    3.99692
Name: test-rmse-mean, dtype: float64
```

> You can see that your RMSE for the price prediction has reduced as compared to last time and came out to be around 4.03 per 1000$. You can reach an even lower RMSE for a different set of hyper-parameters.

Visualize Boosting Trees and Feature Importance

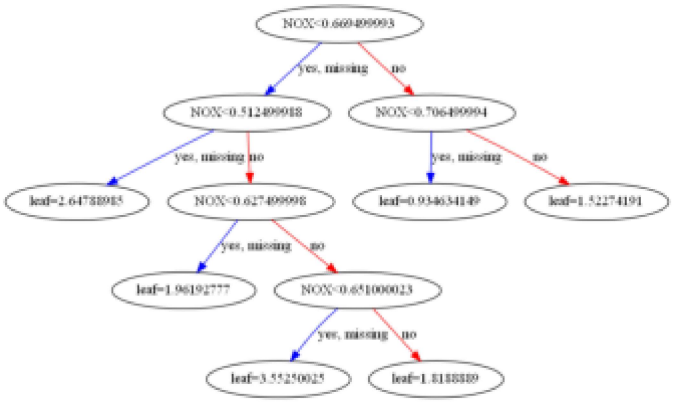
In [35]:
```
xg_reg = xgb.train(params=params, dtrain=data_dmatrix, num_boost_round=10)
```

```
[11:18:39] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/objective/regression_obj.cu:188: reg:linear is now de
precated in favor of reg:squarederror.
```

In [36]:
```
!pip install graphviz
```

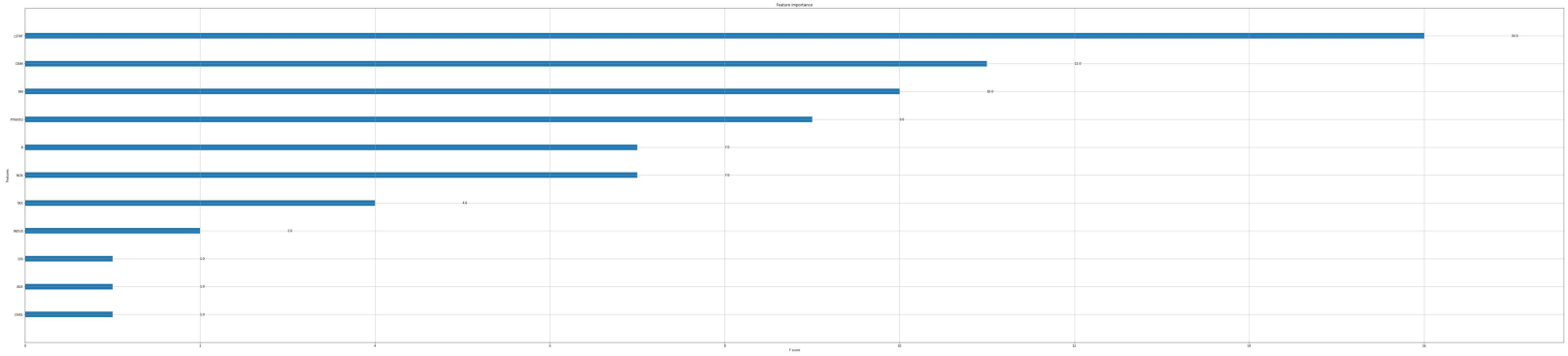
```
Requirement already satisfied: graphviz in c:\users\shubham choudhary\anaconda3\lib\site-packages (0.19.1)
```

In [37]:
```
import matplotlib.pyplot as plt

xgb.plot_tree(xg_reg,num_trees=0)#num tree give be the index of tree
plt.rcParams['figure.figsize'] = [90, 20]
plt.show()
```



In [38]:
```
# download graphviz
```

In [39]:
```
xgb.plot_importance(xg_reg)
plt.rcParams['figure.figsize'] = [10, 10]
plt.show()
```



Adaboost

In [40]:
```
df=pd.read_csv('mushrooms.csv')
```

In [41]:
```
df.head()
```

	class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	...	stalk-surface-below-ring	stalk-color-above-ring	stalk-color-below-ring	veil-type	veil-color	r r
0	p	x	s	n	t	p	f	c	n	k	...	s	w	w	p	w	o
1	e	x	s	y	t	a	f	c	b	k	...	s	w	w	p	w	o
2	e	b	s	w	t	l	f	c	b	n	...	s	w	w	p	w	o
3	p	x	y	w	t	p	f	c	n	n	...	s	w	w	p	w	o
4	e	x	s	g	f	n	f	w	b	k	...	s	w	w	p	w	o

5 rows × 23 columns

In [43]:
```
df.shape
```

(8124, 23)

In [44]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8124 entries, 0 to 8123
Data columns (total 23 columns):
 #   Column                    Non-Null Count  Dtype 
---  ------                    --------------  ----- 
 0   class                     8124 non-null   object
 1   cap-shape                 8124 non-null   object
 2   cap-surface               8124 non-null   object
 3   cap-color                 8124 non-null   object
 4   bruises                   8124 non-null   object
 5   odor                      8124 non-null   object
 6   gill-attachment           8124 non-null   object
 7   gill-spacing              8124 non-null   object
 8   gill-size                 8124 non-null   object
 9   gill-color                8124 non-null   object
 10  stalk-shape               8124 non-null   object
 11  stalk-root                8124 non-null   object
 12  stalk-surface-above-ring  8124 non-null   object
 13  stalk-surface-below-ring  8124 non-null   object
 14  stalk-color-above-ring    8124 non-null   object
 15  stalk-color-below-ring    8124 non-null   object
 16  veil-type                 8124 non-null   object
 17  veil-color                8124 non-null   object
 18  ring-number               8124 non-null   object
 19  ring-type                 8124 non-null   object
 20  spore-print-color         8124 non-null   object
 21  population                8124 non-null   object
 22  habitat                   8124 non-null   object
dtypes: object(23)
memory usage: 1.4+ MB
```

In [45]:
```
df.describe()
```

	class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	...	stalk-surface-below-ring	stalk-color-above-ring	stalk-color-below-ring	veil-type	vei col
count	8124	8124	8124	8124	8124	8124	8124	8124	8124	8124	...	8124	8124	8124	8124	812
unique	2	6	4	10	2	9	2	2	2	12	...	4	9	9	1	4
top	e	x	y	n	f	n	f	c	b	b	...	s	w	w	p	w
freq	4208	3656	3244	2284	4748	3528	7914	6812	5612	1728	...	4936	4464	4384	8124	792

4 rows × 23 columns

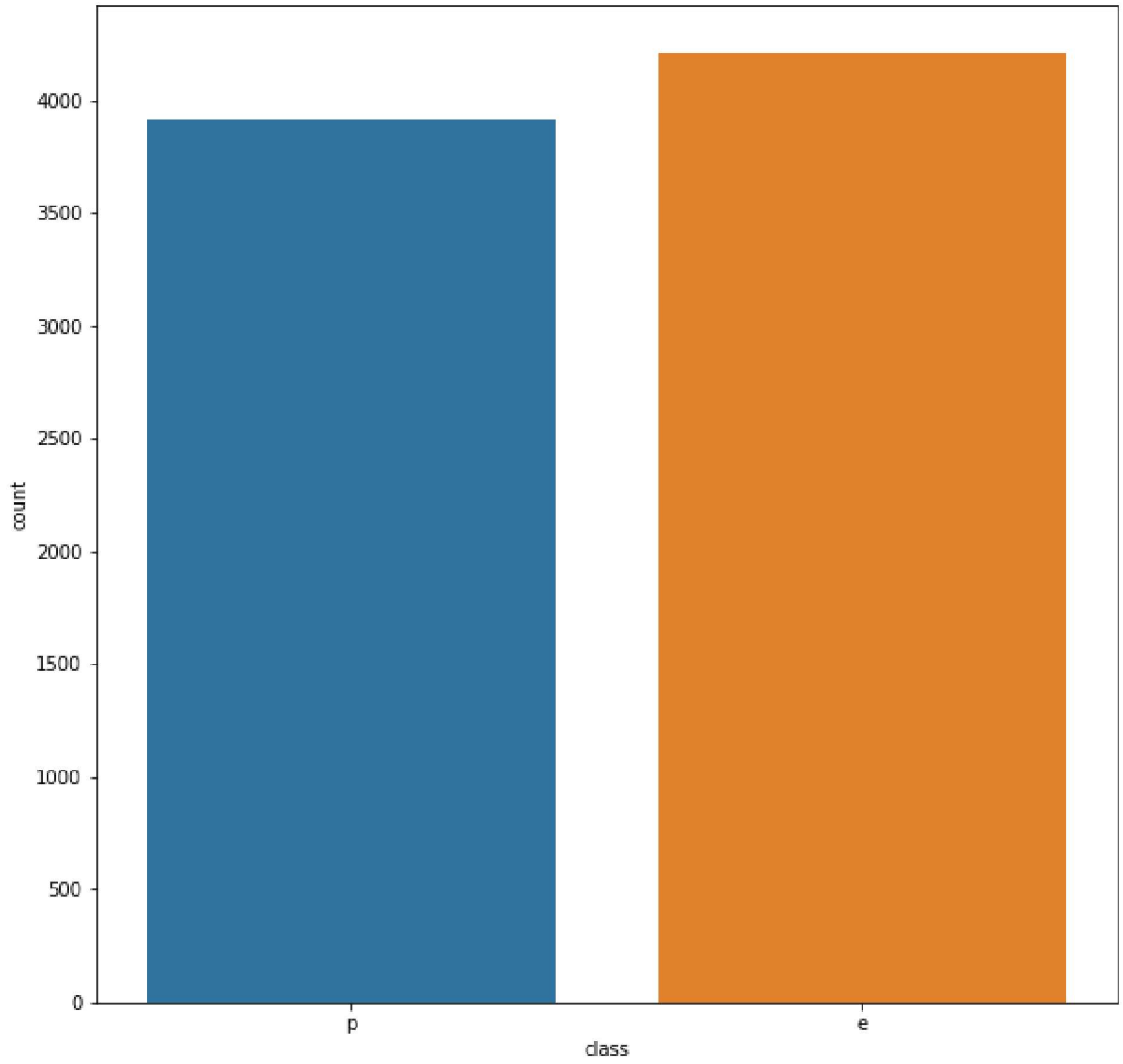
In [51]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import mean_squared_error
```

In [52]:
```
df['class'].value_counts()
```

```
e    4208
p    3916
Name: class, dtype: int64
```

In [53]:
```
sns.countplot(data=df,x='class')
```

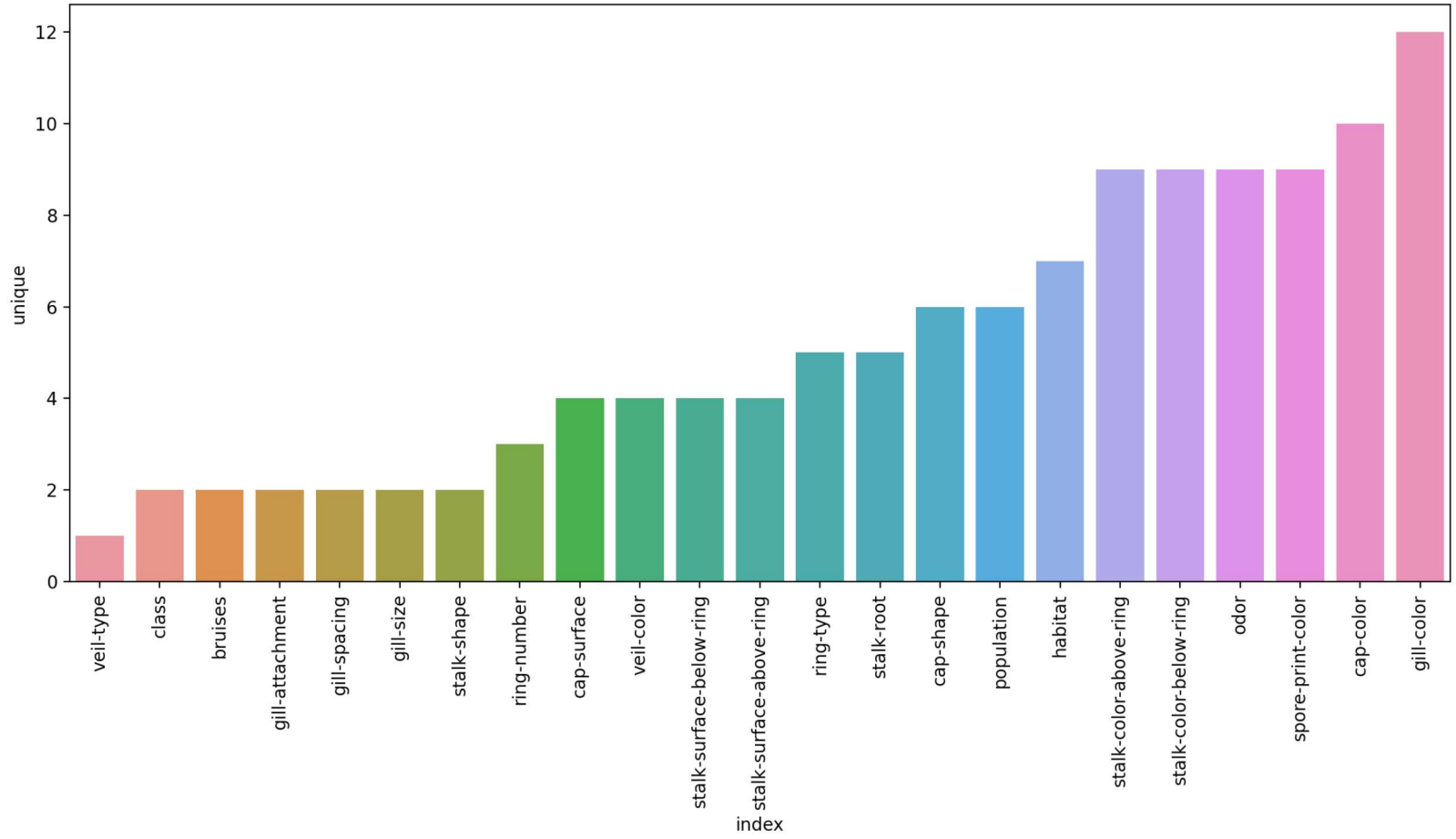
```
<AxesSubplot:xlabel='class', ylabel='count'>
```



In [54]:
```
df.describe().transpose()
```

	count	unique	top	freq
class	8124	2	e	4208
cap-shape	8124	6	x	3656
cap-surface	8124	4	y	3244
cap-color	8124	10	n	2284
bruises	8124	2	f	4748
odor	8124	9	n	3528
gill-attachment	8124	2	f	7914
gill-spacing	8124	2	c	6812
gill-size	8124	2	b	5612
gill-color	8124	12	b	1728
stalk-shape	8124	2	t	4608
stalk-root	8124	5	b	3776
stalk-surface-above-ring	8124	4	s	5176
stalk-surface-below-ring	8124	4	s	4936
stalk-color-above-ring	8124	9	w	4464
stalk-color-below-ring	8124	9	w	4384
veil-type	8124	1	p	8124
veil-color	8124	4	w	7924
ring-number	8124	3	o	7488
ring-type	8124	5	p	3968
spore-print-color	8124	9	w	2388
population	8124	6	v	4040
habitat	8124	7	d	3148

In [55]:
```
plt.figure(figsize=(14,6),dpi=200)
sns.barplot(data=df.describe().transpose().reset_index().sort_values('unique'),x='index',y='unique')
plt.xticks(rotation=90);
```



Train Test Split

In [56]:
```
X = df.drop('class',axis=1)
```

In [65]:
```
y=df['class']
```

In [66]:
```
X.head()
```

	cap-shape_c	cap-shape_f	cap-shape_k	cap-shape_s	cap-shape_x	cap-surface_g	cap-surface_s	cap-surface_y	cap-color_c	cap-color_e	...	population_n	popu
0	0	0	0	0	1	0	1	0	0	0	...	0	1
1	0	0	0	0	1	0	1	0	0	0	...	1	0
2	0	0	0	0	0	0	1	0	0	0	...	1	0
3	0	0	0	0	1	0	0	1	0	0	...	0	1
4	0	0	0	0	1	0	1	0	0	0	...	0	0

5 rows × 95 columns

In [67]:
```
X = pd.get_dummies(X,drop_first=True)
```

In [68]:
```
X.head()
```

	cap-shape_c	cap-shape_f	cap-shape_k	cap-shape_s	cap-shape_x	cap-surface_g	cap-surface_s	cap-surface_y	cap-color_c	cap-color_e	...	population_n	popu
0	0	0	0	0	1	0	1	0	0	0	...	0	1
1	0	0	0	0	1	0	1	0	0	0	...	1	0
2	0	0	0	0	0	0	1	0	0	0	...	1	0
3	0	0	0	0	1	0	0	1	0	0	...	0	1
4	0	0	0	0	1	0	1	0	0	0	...	0	0

5 rows × 95 columns

In [69]:
```
from sklearn.model_selection import train_test_split
```

In [70]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.15, random_state=101)
```

In [71]:
```
X_train.shape
```

```
(6905, 95)
```

In [72]:
```
from sklearn.ensemble import AdaBoostClassifier
```

In [73]:
```
model = AdaBoostClassifier(n_estimators=1)
```

In [74]:
```
model.fit(X_train,y_train)
```

```
AdaBoostClassifier(n_estimators=1)
```

In [77]:
```
from sklearn.metrics import classification_report,plot_confusion_matrix,accuracy_score,confusion_matrix
```

In [78]:
```
predictions = model.predict(X_test)
```

In [79]:
```
predictions
```

```
array(['p', 'e', 'p', ..., 'p', 'p', 'e'], dtype=object)
```

Evaluation

In [80]:
```
print(classification_report(y_test,predictions))
```

```
              precision    recall  f1-score   support

           e       0.96      0.81      0.88       655
           p       0.81      0.96      0.88       564

    accuracy                           0.88      1219
   macro avg       0.88      0.88      0.88      1219
weighted avg       0.89      0.88      0.88      1219
```

In [81]:
```
model.feature_importances_
```

```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

Here are all 0s and only one 1 is there.this makes sense because our model is technically a single stump which means one feature is 100% important to models capability to predict.

In [82]:
```
model.feature_importances_.argmax()
```

```
22
```

In [84]:
```
X.columns[22]
```

```
'odor_n'
```

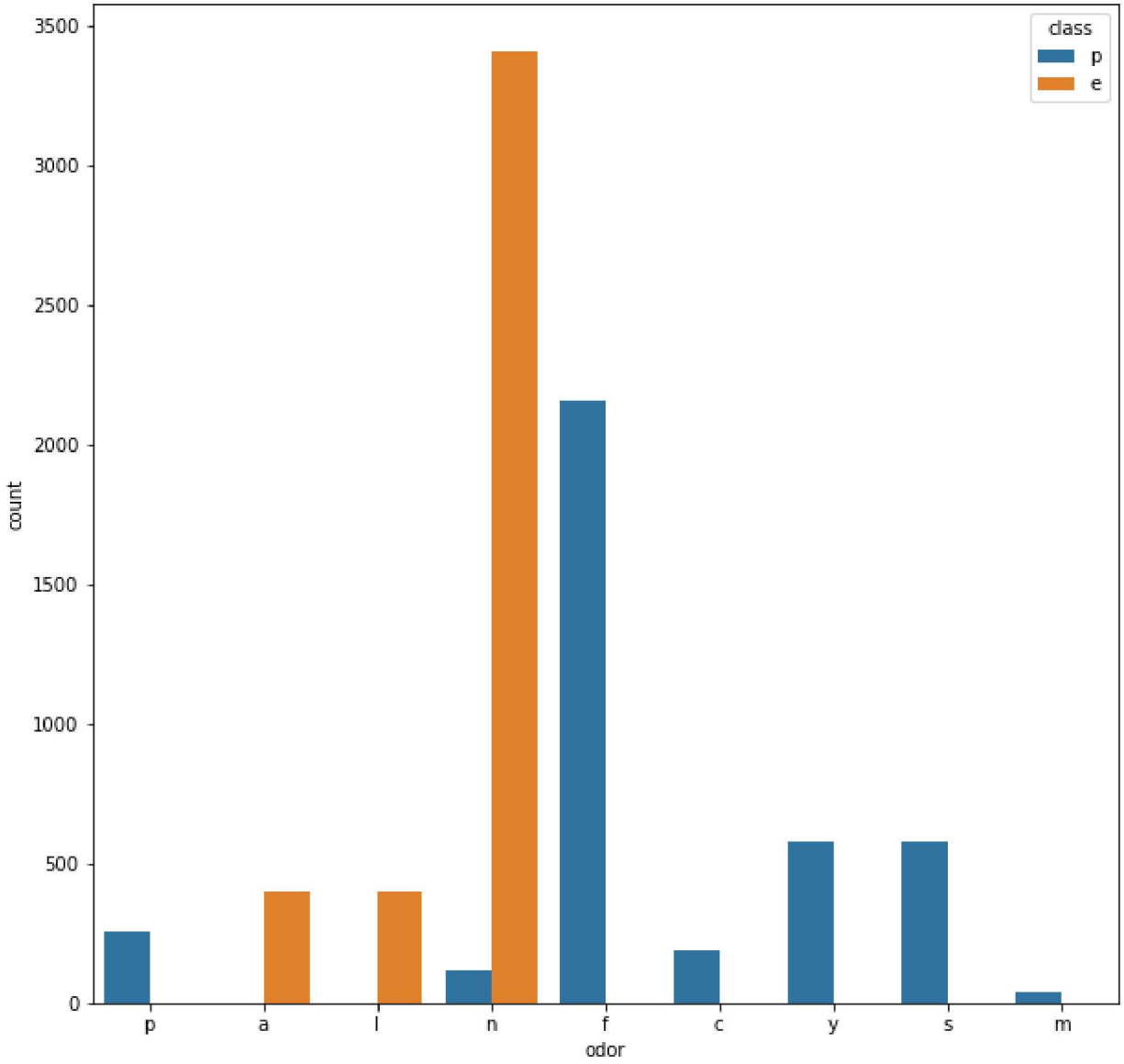
odor is the most important feature.This means according to our adaboost

classifier model with a single stump odor is the feature with which we can decide

whether the mushroom is poisionous or edible

In [86]:
```
sns.countplot(data=df,x='odor',hue='class')
```

```
<AxesSubplot:xlabel='odor', ylabel='count'>
```



Analyzing performance as more weak learners are added

In [88]:
```
len(X.columns)
```

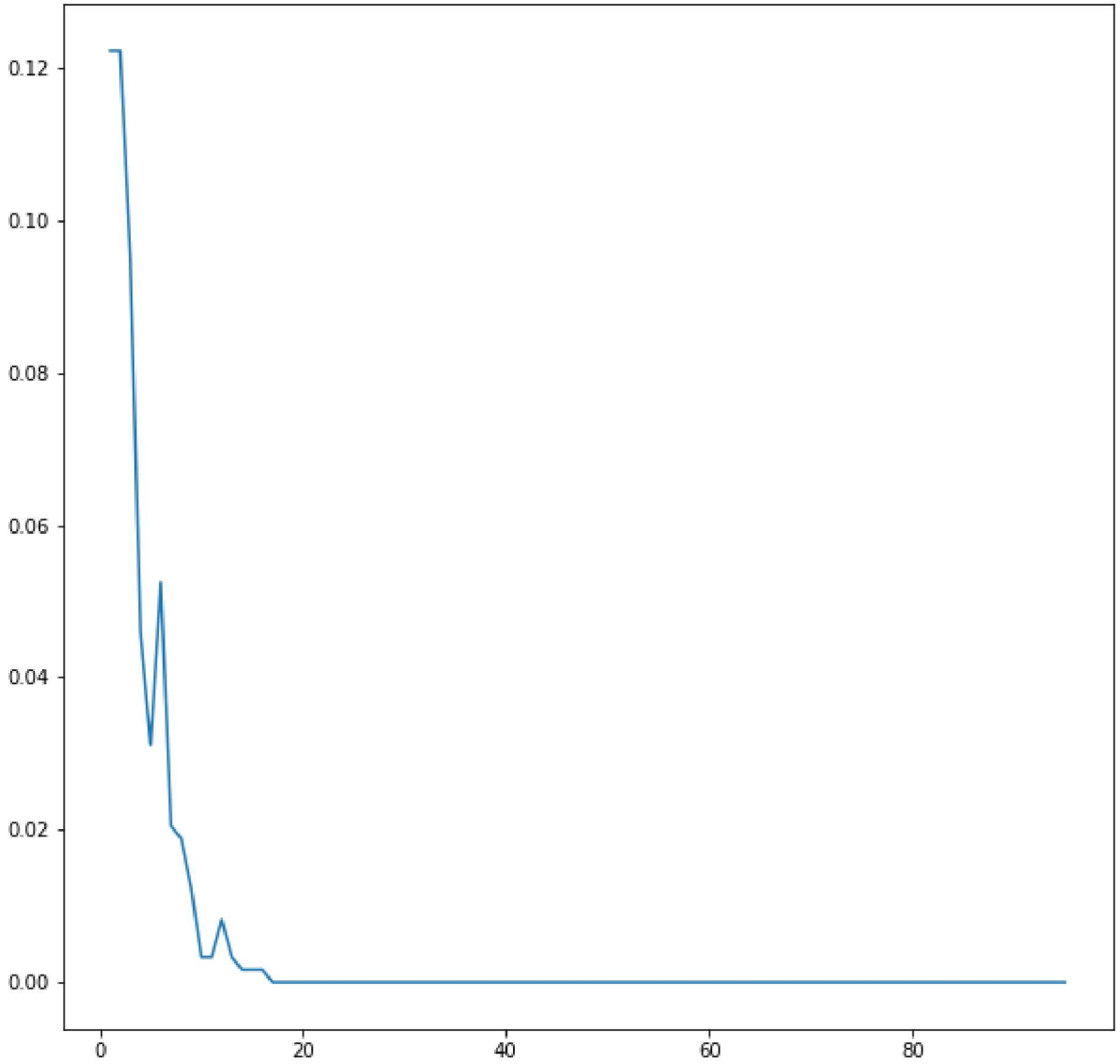
```
95
```

In [ ]:

In [89]:
```
error_rates = []

for n in range(1,96):

    model = AdaBoostClassifier(n_estimators=n)
    model.fit(X_train,y_train)
    preds = model.predict(X_test)
    err = 1 - accuracy_score(y_test,preds)

    error_rates.append(err)
```

In [90]:
```
plt.plot(range(1,96),error_rates)
```

```
[<matplotlib.lines.Line2D at 0x16285506c70>]
```



error rate starts at 0.12 because initial accuracy was 88% and it goes on

decreasing and after 20 it does not make any progress so adding 20 stumps is the

best choice for this

In [91]:
```
model
```

```
AdaBoostClassifier(n_estimators=95)
```

In [93]:
```
model.feature_importances_
```

```
array([0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.01052632, 0.        ,
       0.        , 0.01052632, 0.        , 0.        , 0.        ,
       0.01052632, 0.        , 0.05263158, 0.03157895, 0.03157895,
       0.        , 0.        , 0.06315789, 0.02105263, 0.        ,
       0.        , 0.        , 0.09473684, 0.09473684, 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.01052632, 0.01052632, 0.        , 0.        , 0.        ,
       0.06315789, 0.        , 0.        , 0.        , 0.        ,
       0.03157895, 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.06315789, 0.        , 0.        ,
       0.01052632, 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.01052632, 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.05263158, 0.        , 0.16842105, 0.        , 0.10526316,
       0.        , 0.        , 0.04210526, 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.01052632])
```

In [94]:
```
feats = pd.DataFrame(index=X.columns,data=model.feature_importances_,columns=['Importance'])
```

In [95]:
```
feats
```

	Importance
cap-shape_c	0.000000
cap-shape_f	0.000000
cap-shape_k	0.000000
cap-shape_s	0.000000
cap-shape_x	0.000000
...	...
habitat_l	0.000000
habitat_m	0.000000
habitat_p	0.000000
habitat_u	0.000000
habitat_w	0.010526

95 rows × 1 columns

In [96]:
```
imp_feats = feats[feats['Importance']>0]
```

In [97]:
```
imp_feats
```

	Importance
cap-color_c	0.010526
cap-color_n	0.010526
cap-color_w	0.010526
bruises_t	0.052632
odor_c	0.031579
odor_f	0.031579
odor_n	0.063158
odor_p	0.021053
gill-spacing_w	0.094737
gill-size_n	0.094737
stalk-shape_t	0.010526
stalk-root_b	0.010526
stalk-surface-above-ring_k	0.063158
stalk-surface-below-ring_y	0.031579
stalk-color-below-ring_n	0.063158
stalk-color-below-ring_w	0.010526
ring-number_t	0.010526
spore-print-color_r	0.052632
spore-print-color_w	0.168421
population_c	0.105263
population_v	0.042105
habitat_w	0.010526

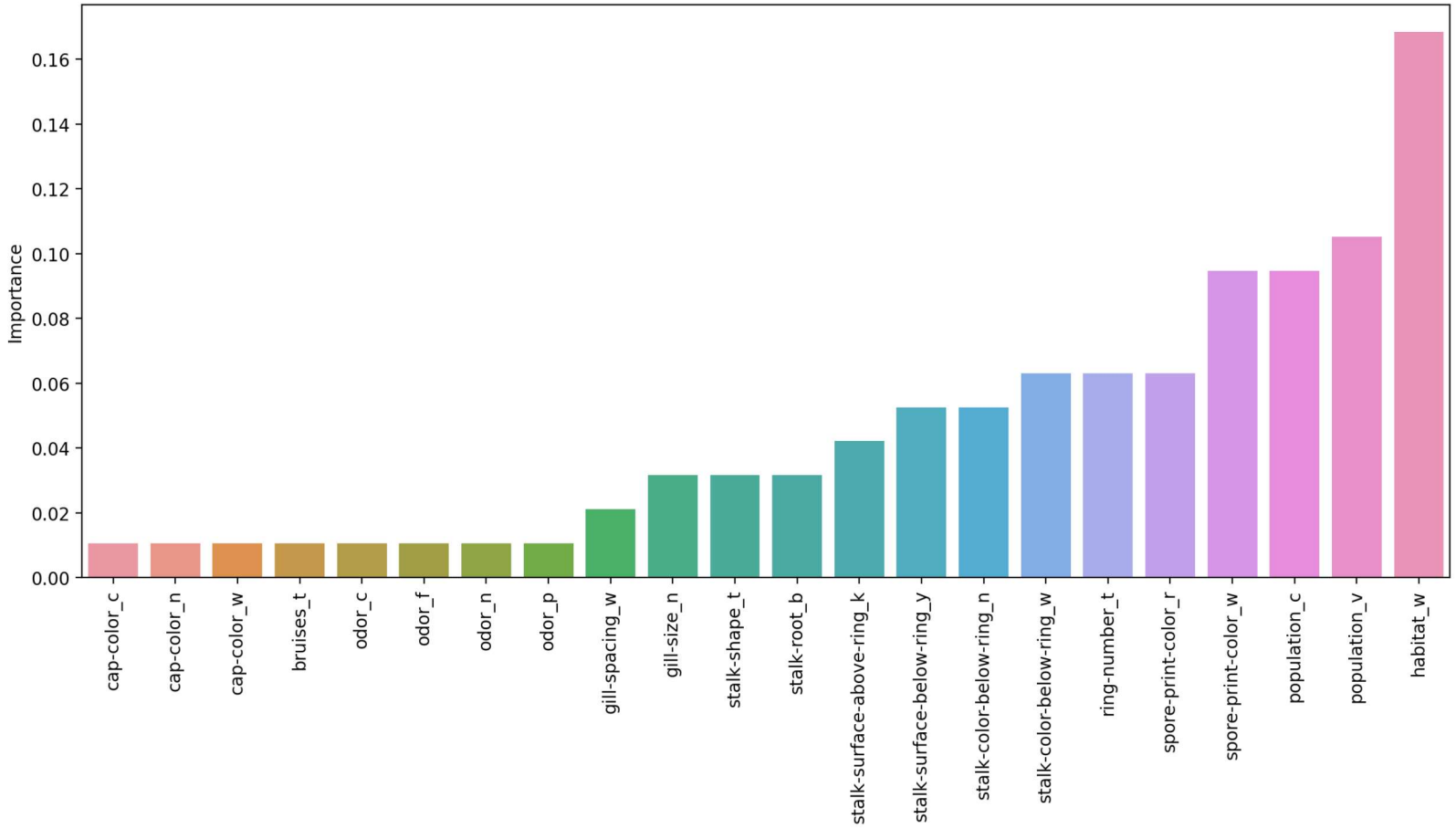
In [98]:
```
imp_feats.count()
```

```
Importance    22
dtype: int64
```

In [99]:
```
imp_feats.sort_values("Importance")
```

	Importance
cap-color_c	0.010526
ring-number_t	0.010526
stalk-color-below-ring_w	0.010526
stalk-root_b	0.010526
stalk-shape_t	0.010526
habitat_w	0.010526
cap-color_n	0.010526
cap-color_w	0.010526
odor_p	0.021053
odor_c	0.031579
odor_f	0.031579
stalk-surface-below-ring_y	0.031579
population_v	0.042105
bruises_t	0.052632
spore-print-color_r	0.052632
stalk-surface-above-ring_k	0.063158
stalk-color-below-ring_n	0.063158
odor_n	0.063158
gill-size_n	0.094737
gill-spacing_w	0.094737
population_c	0.105263
spore-print-color_w	0.168421

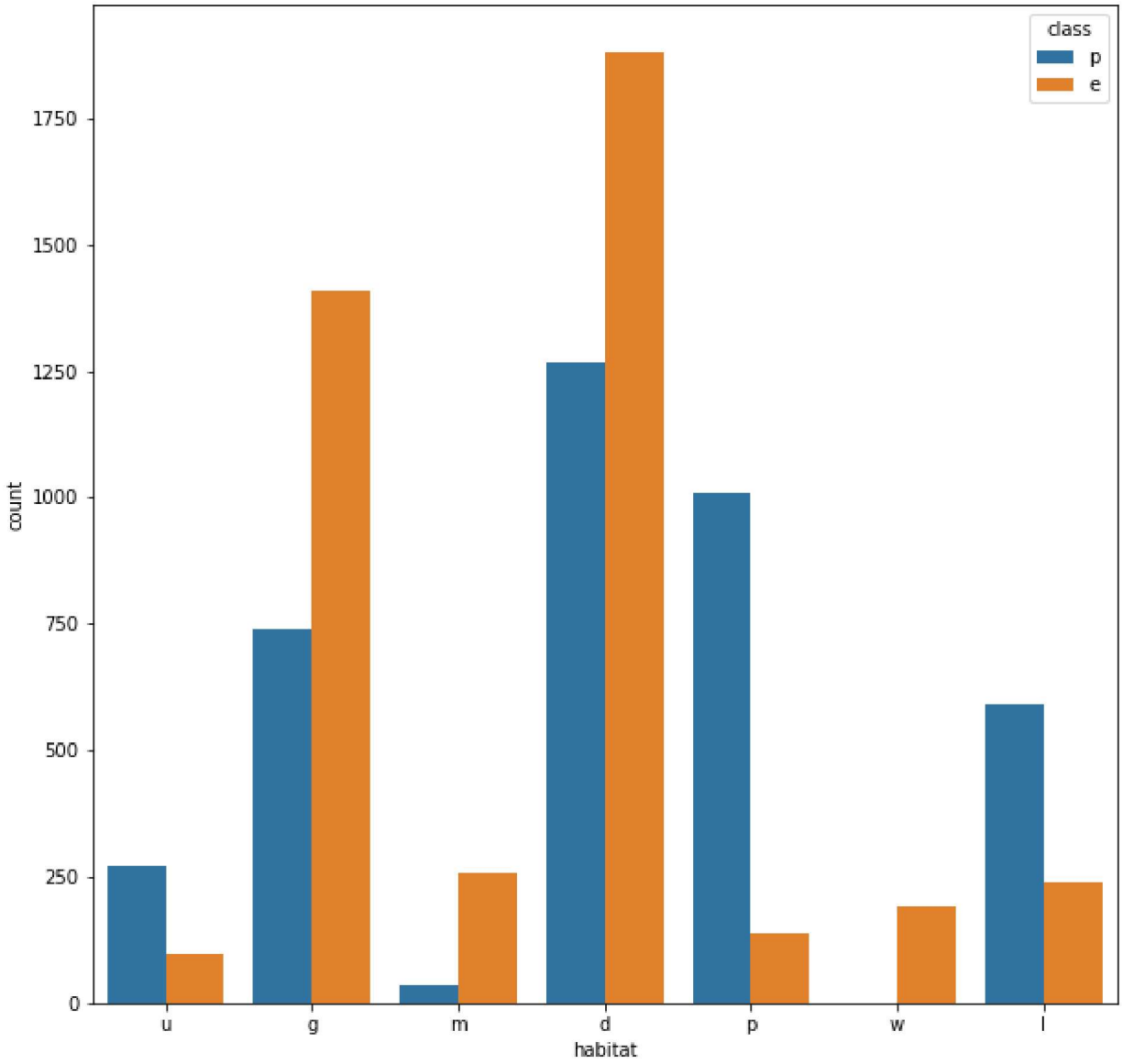
In [100]:
```
plt.figure(figsize=(14,6),dpi=200)
sns.barplot(data=imp_feats.sort_values('Importance'),x=imp_feats.index,y='Importance')

plt.xticks(rotation=90);
```



> As we add in more stumps,coefficients applied to each of those stupms i.e,eachof those subset of features is also going to change.so you should not expectodor to be the most imp as you add in more stumps because that constantly adjusted as you keep adjusting weights.

In [101]:
```
sns.countplot(data=df,x='habitat',hue='class')
```

```
<AxesSubplot:xlabel='habitat', ylabel='count'>
```



In [ ]:
```
model = AdaBoostClassifier(n_estimators=20)
model.fit(X_train,y_train)
preds = model.predict(X_test)
accuracy_score(y_test,preds)
```

In [4]:
```
!pip install nltk
```

```
Requirement already satisfied: nltk in c:\users\shubham choudhary\anaconda3\lib\site-packages (3.6.5)
Requirement already satisfied: click in c:\users\shubham choudhary\anaconda3\lib\site-packages (from nltk) (8.0.3)
Requirement already satisfied: joblib in c:\users\shubham choudhary\anaconda3\lib\site-packages (from nltk) (1.1.0)
Requirement already satisfied: regex>=2021.8.3 in c:\users\shubham choudhary\anaconda3\lib\site-packages (from nltk) (2021.8.3)
Requirement already satisfied: tqdm in c:\users\shubham choudhary\anaconda3\lib\site-packages (from nltk) (4.62.3)
Requirement already satisfied: colorama in c:\users\shubham choudhary\anaconda3\lib\site-packages (from click->nltk) (0.4.4)
```

In [5]:
```
# Natural langiage toolkit
```

In [6]:
```
import nltk
```

In [8]:
```
nltk.download_shell()
```

```
NLTK Downloader
---------------------------------------------------------------------------
    d) Download   l) List    u) Update   c) Config   h) Help   q) Quit
---------------------------------------------------------------------------
Downloader> d

Download which package (l=list; x=cancel)?
  Identifier> stopwords


    Downloading package stopwords to C:\Users\SHUBHAM
        CHOUDHARY\AppData\Roaming\nltk_data...
    Error downloading 'stopwords' from
        <https://raw.githubusercontent.com/nltk/nltk_data/gh-
        pages/packages/corpora/stopwords.zip>:   <urlopen error [Errno
        11001] getaddrinfo failed>



---------------------------------------------------------------------------
    d) Download   l) List    u) Update   c) Config   h) Help   q) Quit
---------------------------------------------------------------------------
Downloader> d

Download which package (l=list; x=cancel)?
  Identifier> stopwords


    Downloading package stopwords to C:\Users\SHUBHAM
        CHOUDHARY\AppData\Roaming\nltk_data...
      Unzipping corpora\stopwords.zip.



---------------------------------------------------------------------------
    d) Download   l) List    u) Update   c) Config   h) Help   q) Quit
```

```
---------------------------------------------------------------------------
Downloader> q
```

In [10]:
```
messages = [line.rstrip() for line in open('SMSSpamCollection')]
```

In [12]:
```
messages[0:5]
```

```
['ham\tGo until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got amore wat...',
 'ham\tOk lar... Joking wif u oni...',
 "spam\tFree entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive entry question(std txt rate)T&C's
apply 08452810075over18's",
 'ham\tU dun say so early hor... U c already then say...',
 "ham\tNah I don't think he goes to usf, he lives around here though"]
```

In [13]:
```
print(len(messages))
```

```
5574
```

In [14]:
```
messages[0]
```

```
'ham\tGo until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got amore wat...'
```

In [15]:
```
messages[50]
```

```
'ham\tWhat you thinked about me. First time you saw me in class.'
```

A collection of texts is also sometimes called "corpus". Let's print the first ten messages and number them using enumerate:

In [17]:
```
for message_no, message in enumerate(messages[:10]):
    print(message_no, message)
    print('\n')
```

```
0 ham   Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got amore wat...


1 ham   Ok lar... Joking wif u oni...


2 spam  Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive entry question(std txt rate)T&C's
apply 08452810075over18's


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 around here though


5 spam  FreeMsg Hey there darling it's been 3 week's now and no word back! I'd like some fun you up for it still? Tb ok! XxX std chgs to s
end, Â£1.50 to rcv


6 ham   Even my brother is not like to speak with me. They treat me like aids patent.


7 ham   As per your request 'Melle Melle (Oru Minnaminunginte Nurungu Vettam)' has been set as your callertune for all Callers. Press *9 t
o copy your friends Callertune


8 spam  WINNER!! As a valued network customer you have been selected to receivea Â£900 prize reward! To claim call 09061701461. Claim code
KL341. Valid 12 hours only.


9 spam  Had your mobile 11 months or more? U R entitled to Update to the latest colour mobiles with camera for Free! Call The Mobile Updat
e Co FREE on 08002986030
```

In [18]:
```
import pandas as pd
```

In [19]:
```
messages = pd.read_csv('SMSSpamCollection', sep='\t',
                           names=["label", "message"])
messages.head()
```

	label	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...

Exploratory data Analysis

messages.describe()

In [25]:
```
messages.describe()
```

	label	message
count	5572	5572
unique	2	5169
top	ham	Sorry, I'll call later
freq	4825	30

In [27]:
```
messages['label'].value_counts()
```

```
ham     4825
spam     747
Name: label, dtype: int64
```

Let's use groupby to use describe by label, this way we can begin to think about the features that separate ham and spam!

In [29]:
```
messages.groupby('label').describe()
```

	message			
	count	unique	top	freq
label				
ham	4825	4516	Sorry, I'll call later	30
spam	747	653	Please call our customer service representativ...	4

As we continue our analysis we want to start thinking about the features we are going to be using. This goes along with the general idea of feature engineering. The better your domain knowledge on the data, the better your ability to engineer more features from it. Feature engineering is a very large part of spam detection in general. I encourage you to read up on the topic!

Let's make a new column to detect how long the text messages are:

In [30]:
```
messages['length'] = messages['message'].apply(len)
messages.head()
```

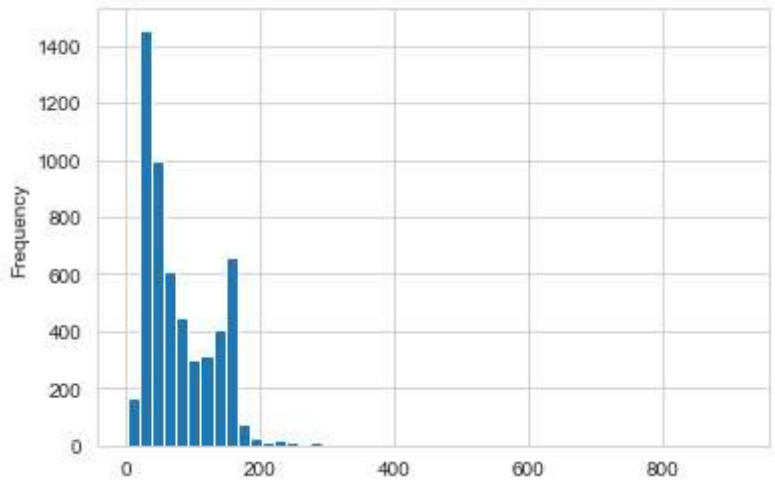
	label	message	length
0	ham	Go until jurong point, crazy.. Available only ...	111
1	ham	Ok lar... Joking wif u oni...	29
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	155
3	ham	U dun say so early hor... U c already then say...	49
4	ham	Nah I don't think he goes to usf, he lives aro...	61

In [31]:
```
import matplotlib.pyplot as plt
import seaborn as sns

%matplotlib inline
```

In [32]:
```
sns.set_style('whitegrid')
```

In [33]:
```
messages['length'].plot(bins=50, kind='hist')
```

```
<AxesSubplot:ylabel='Frequency'>
```



Let's try to explain why the x-axis goes all the way to 1000ish, this must mean that there is some really long message!

In [34]:
```
messages.length.describe()
```

```
count    5572.000000
mean       80.489950
std        59.942907
min         2.000000
25%        36.000000
50%        62.000000
75%       122.000000
max       910.000000
Name: length, dtype: float64
```

In [35]:
```
messages.head()
```

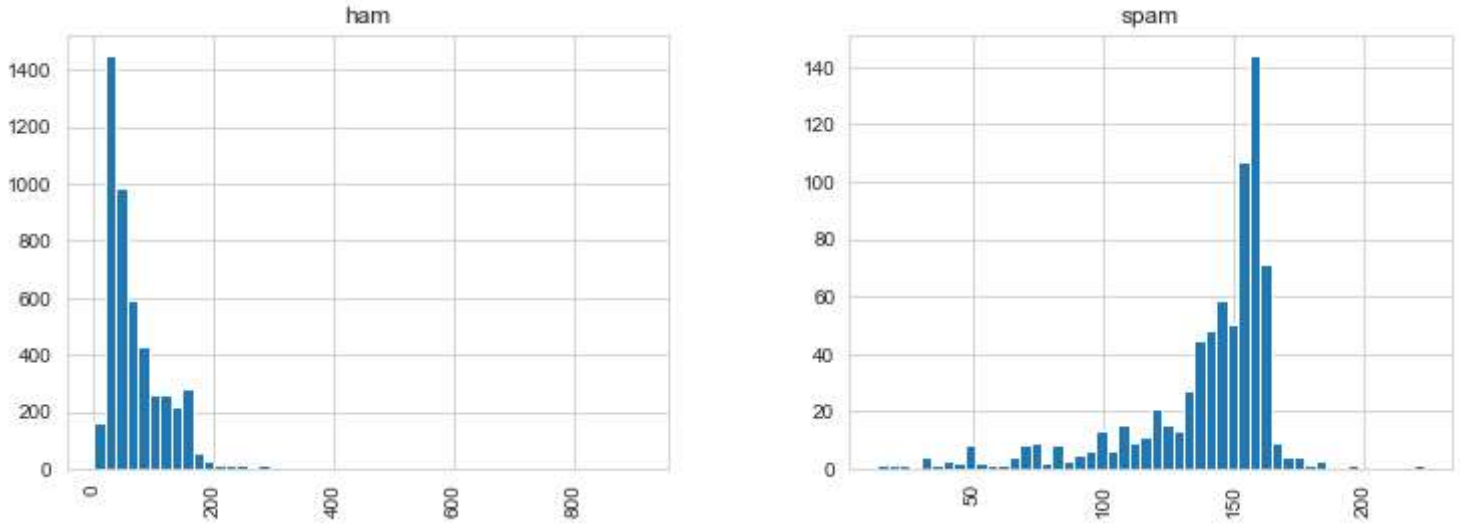
	label	message	length
0	ham	Go until jurong point, crazy.. Available only ...	111
1	ham	Ok lar... Joking wif u oni...	29
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	155
3	ham	U dun say so early hor... U c already then say...	49
4	ham	Nah I don't think he goes to usf, he lives aro...	61

In [36]:
```
messages[messages['length'] == 910]['message'].iloc[0]
```

```
"For me the love should start with attraction.i should feel that I need her every time around me.she should be the first thing which comes in my thoughts.I would start the day and end it with her.she should be there every time I dream.love will be then when my every breath has her name.my life should happen around her.my life will be named to her.I would cry for her.will give all my happiness and take all her sorrows.I will be ready to fight with anyone for her.I will be in love when I will be doing the craziest things for her.love will be when I don't have to proove anyone that my girl is the most beautiful lady on the whole planet.I will always be singing praises for her.love will be when I start up making chicken curry and end up makiing sambar.life will be the most beautiful then.will get every morning and thank god for the day because she is with me.I would like to say a lot..will tell later.."
```

In [37]:
```
messages.hist(column='length', by='label', bins=50,figsize=(12,4))
```

```
array([<AxesSubplot:title={'center':'ham'}>,
       <AxesSubplot:title={'center':'spam'}>], dtype=object)
```



Our main issue with our data is that it is all in text format (strings). The classification algorithms that we've learned about so far will need some sort of numerical feature vector in order to perform the classification task. There are actually many methods to convert a corpus to a vector format. The simplest is the the bag-of-words (http://en.wikipedia.org/wiki/Bag-of-

words_model) approach, where each unique word in a text will be represented by one number.

In this section we'll convert th

In [38]:
```
import string
```

In [39]:
```
mess = 'Sample message! Notice: it has punctuation.'

# Check characters to see if they are in punctuation
nopunc = [char for char in mess if char not in string.punctuation]

# Join the characters again to form the string.
nopunc = ''.join(nopunc)
```

In [40]:
```
print(nopunc)
```

```
Sample message Notice it has punctuation
```

Remove the stopwords

In [41]:
```
from nltk.corpus import stopwords
stopwords.words('english')
```

```
['i',
 'me',
 'my',
 'myself',
 'we',
 'our',
 'ours',
 'ourselves',
 'you',
 "you're",
 "you've",
 "you'll",
 "you'd",
 'your',
 'yours',
 'yourself',
 'yourselves',
 'he',
 'him',
 'his',
 'himself',
 'she',
 "she's",
 'her'
```

In [42]:
```
nopunc.split()
```

```
['Sample', 'message', 'Notice', 'it', 'has', 'punctuation']
```

In [ ]:

In [ ]:

Now just remove any stopwords

clean_mess = [word for word in nopunc.split() if word.lower() not in stopwords.words('english')]

In [43]:
```
# Now just remove any stopwords
clean_mess = [word for word in nopunc.split() if word.lower() not in stopwords.words('english')]
```

In [44]:
```
def text_process(mess):
    """
    Takes in a string of text, then performs the following:
    1. Remove all punctuation
    2. Remove all stopwords
    3. Returns a list of the cleaned text
    """
    # Check characters to see if they are in punctuation
    nopunc = [char for char in mess if char not in string.punctuation]

    # Join the characters again to form the string.
    nopunc = ''.join(nopunc)

    # Now just remove any stopwords
    return [word for word in nopunc.split() if word.lower() not in stopwords.words('english')]
```

In [45]:
```
messages.head()
```

	label	message	length
0	ham	Go until jurong point, crazy.. Available only ...	111
1	ham	Ok lar... Joking wif u oni...	29
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	155
3	ham	U dun say so early hor... U c already then say...	49
4	ham	Nah I don't think he goes to usf, he lives aro...	61

Now let's "tokenize" these messages. Tokenization is just the term used to describe the process of converting the normal text strings in to a list of tokens (words that we actually want).

Let's see an example output on on column:

Note: We may get some warnings or errors for symbols we didn't account for or that weren't in Unicode (like a British pound symbol)

In [47]:
```
# Check to make sure its working
messages['message'].head(5).apply(text_process)
```

```
0    [Go, jurong, point, crazy, Available, bugis, n...
1                        [Ok, lar, Joking, wif, u, oni]
2    [Free, entry, 2, wkly, comp, win, FA, Cup, fin...
3        [U, dun, say, early, hor, U, c, already, say]
4    [Nah, dont, think, goes, usf, lives, around, t...
Name: message, dtype: object
```

Continuing Normalization

There are a lot of ways to continue normalizing this text. Such as Stemming (https://en.wikipedia.org/wiki/Stemming) or distinguishing by part of speech (http://www.nltk.org/book/ch05.html).

NLTK has lots of built-in tools and great documentation on a lot of these methods. Sometimes they don't work well for text-messages due to the way a lot of people tend to use abbreviations or shorthand, For example:

```
'Nah dawg, IDK! Wut time u headin to da club?';
versus


'No dog, I don't know! What time are you heading to the club?'
```

Some text normalization methods will have trouble with this type of shorthand and so I'll leave you to explore those more advanced methods through the NLTK book online (http://www.nltk.org/book/).

For now we will just focus on using what we have to convert our list of words to an actual vector that SciKit-Learn can use.

In [1]:

In [ ]: