Kernel Ridge regression is similar to K- Nearest neighbours apart from the fact that in KNN all the data points are given equal weightage but in kernel ridge regression the weightage of the points are different so the end result differs.

Implementation/Comparion of Kernel Ridge Regression and Ridge Regression

In [1]:
```
#The dataset includes the various parameters on which a house price is predict
ed.
# Loading the necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error, make_scorer
from sklearn.kernel_ridge import KernelRidge
from sklearn import linear_model
```

In [2]:
```
# Reading the dataset
home_price=pd.read_csv('HousePrices.csv')
home_price.head()
```

Out[2]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5650
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242
2	5631500400	20150225T000000	180000.0	2	1.00	770	10000
3	2487200875	20141209T000000	604000.0	4	3.00	1960	5000
4	1954400510	20150218T000000	510000.0	3	2.00	1680	8080

5 rows × 21 columns

In [4]:
```
home_price.shape
```

Out[4]: (21613, 21)

In [5]:
```
home_price.describe()
```

Out[5]:

	id	price	bedrooms	bathrooms	sqft_living	sq
count	2.161300e+04	2.161300e+04	21613.000000	21613.000000	21613.000000	2.161300
mean	4.580302e+09	5.401822e+05	3.370842	2.114757	2079.899736	1.510697
std	2.876566e+09	3.673622e+05	0.930062	0.770163	918.440897	4.142051
min	1.000102e+06	7.500000e+04	0.000000	0.000000	290.000000	5.200000
25%	2.123049e+09	3.219500e+05	3.000000	1.750000	1427.000000	5.040000
50%	3.904930e+09	4.500000e+05	3.000000	2.250000	1910.000000	7.618000
75%	7.308900e+09	6.450000e+05	4.000000	2.500000	2550.000000	1.068800
max	9.900000e+09	7.700000e+06	33.000000	8.000000	13540.000000	1.651359

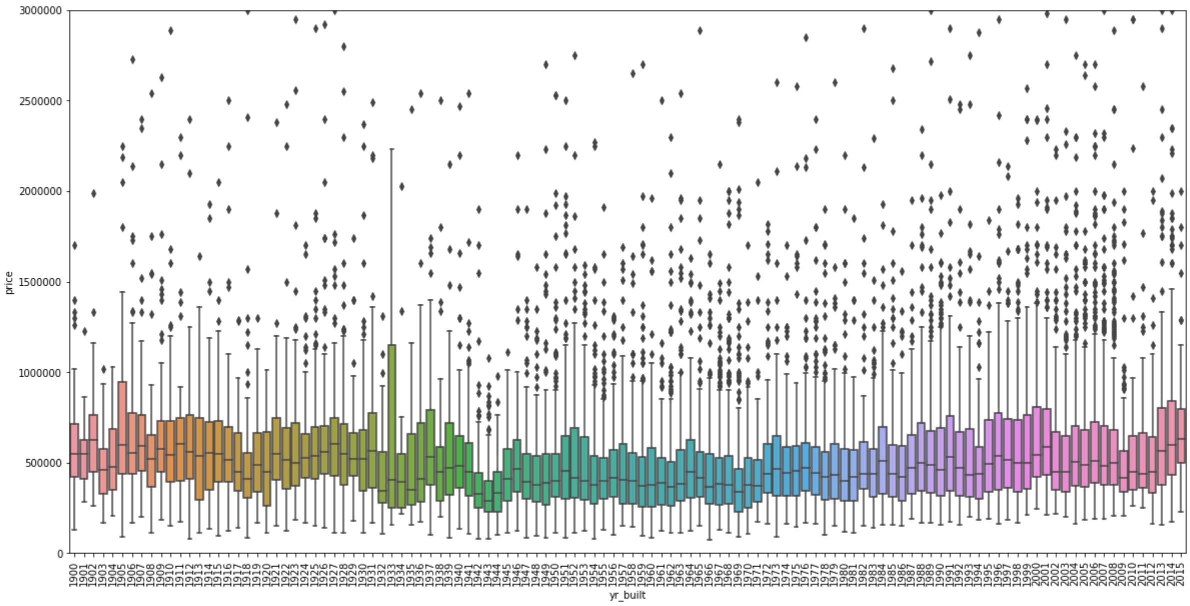
In [6]:
```
## We can see the minimum bathrooms in a house in 0. Thats strange. Missing Va
lues
```

In [7]:
```
# Dropping off id and date, its not significant
home_price.drop('id', axis = 1, inplace = True)
home_price.drop('date', axis = 1, inplace = True)
home_price.head()
```

Out[7]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	conditic
0	221900.0	3	1.00	1180	5650	1.0	0	0	3
1	538000.0	3	2.25	2570	7242	2.0	0	0	3
2	180000.0	2	1.00	770	10000	1.0	0	0	3
3	604000.0	4	3.00	1960	5000	1.0	0	0	5
4	510000.0	3	2.00	1680	8080	1.0	0	0	3

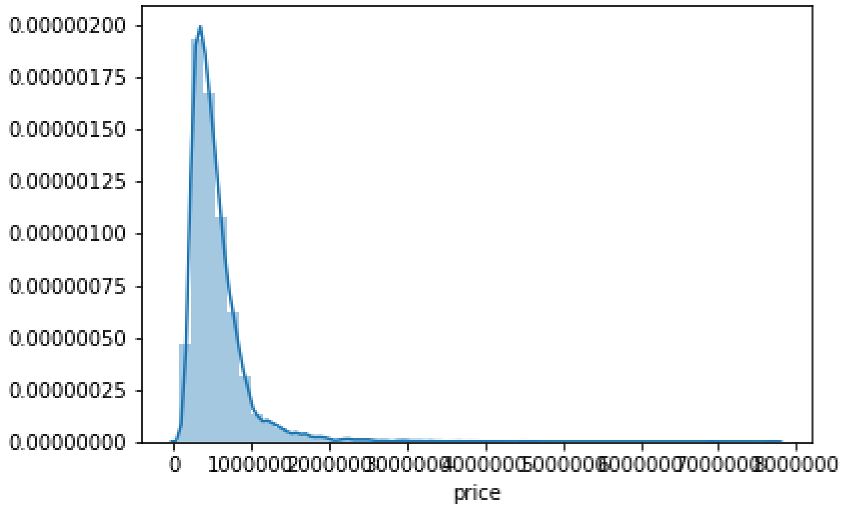
In [8]:
```
# Analysing the relationship between the response variable and various input parameters
# 1. Relationship between price and year built
data = pd.concat([home_price['price'], home_price['yr_built']], axis =1)
f, ax = plt.subplots(figsize = (20,10))
fig = sns.boxplot(x = home_price['yr_built'], y = home_price['price'], data = data)
fig.axis(ymin = 0, ymax = 3000000)
plt.xticks(rotation = 90)
plt.show()
```



Inference

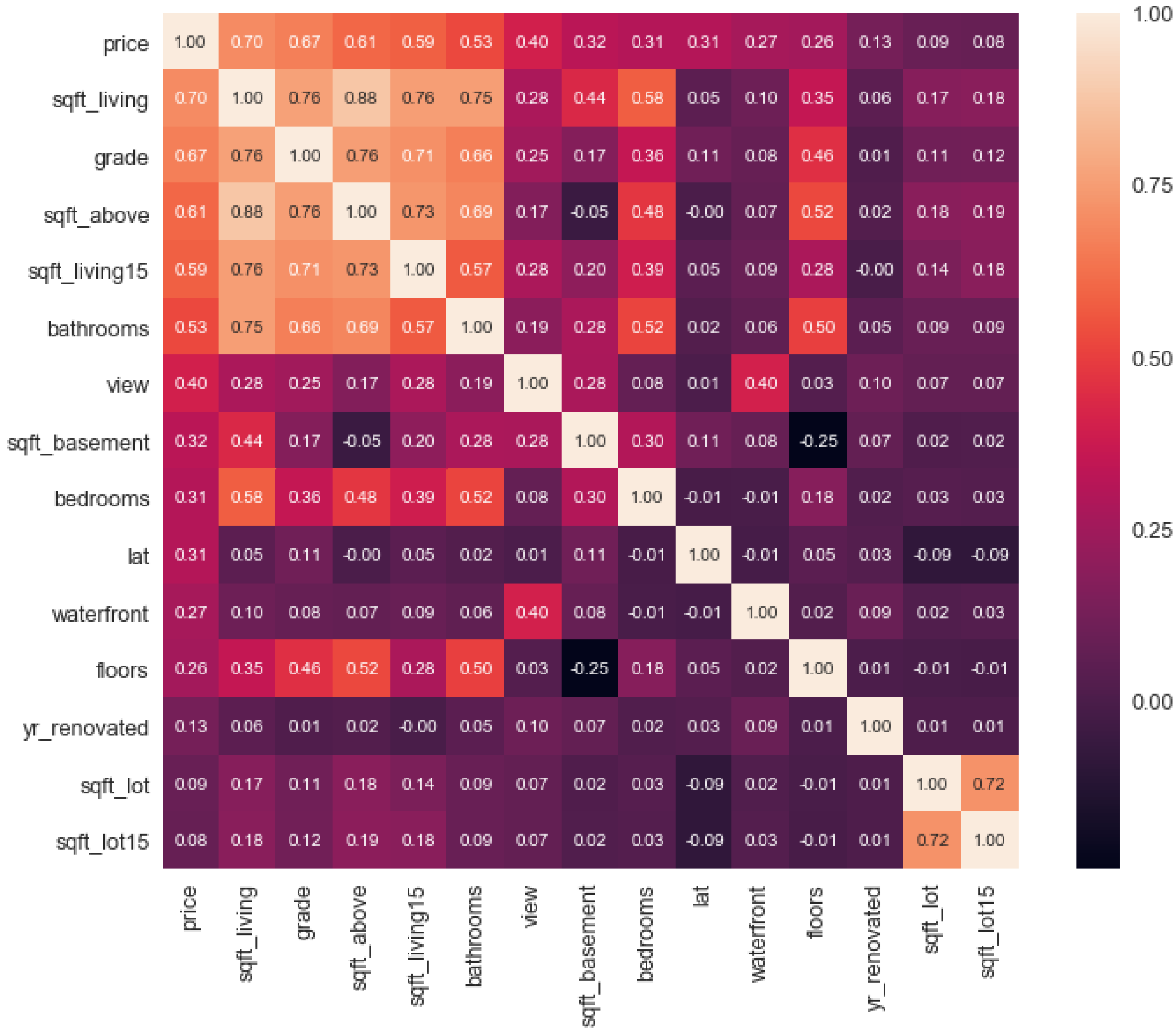
Year built is not much significant. Lots of outliers are spotted, skewness in data is observed. Nothing much inference can be drawn.

In [10]:
```
# Transformation of the dataset since skewness is observed
sns.distplot(home_price['price'])
plt.show()
```

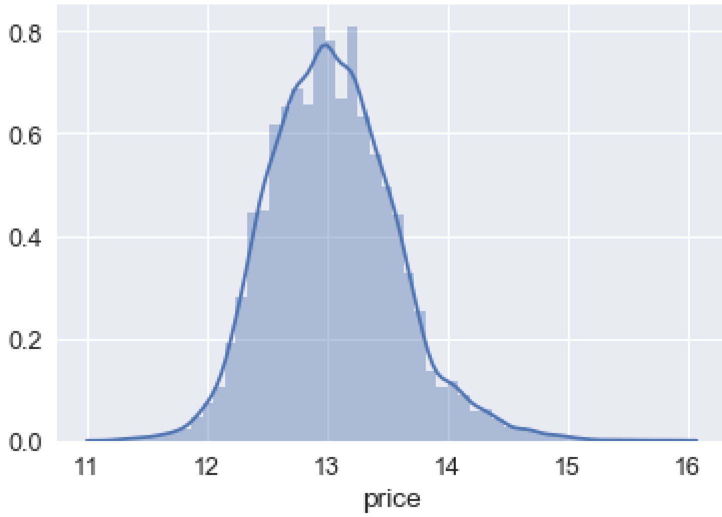


From the above plot we can confirm skewness exsists

In [12]:
```
# The best approach to find out the relationship between price and other varia
bles is by plotting a confusion matrix
#Here, I have taken the top 15 correlated features into a variable named cols.
k = 15
corrmat = home_price.corr()
cols = corrmat.nlargest(k, 'price')['price'].index
cm = np.corrcoef(home_price[cols].values.T)
f, ax = plt.subplots(figsize = (12,9)) #defining figure size
sns.set(font_scale = 1.25) #font size
hm = sns.heatmap(cm, cbar=True, annot=True, square=True, fmt='.2f', annot_kws=
{'size': 10}, yticklabels=cols.values, xticklabels=cols.values)
plt.show()
```

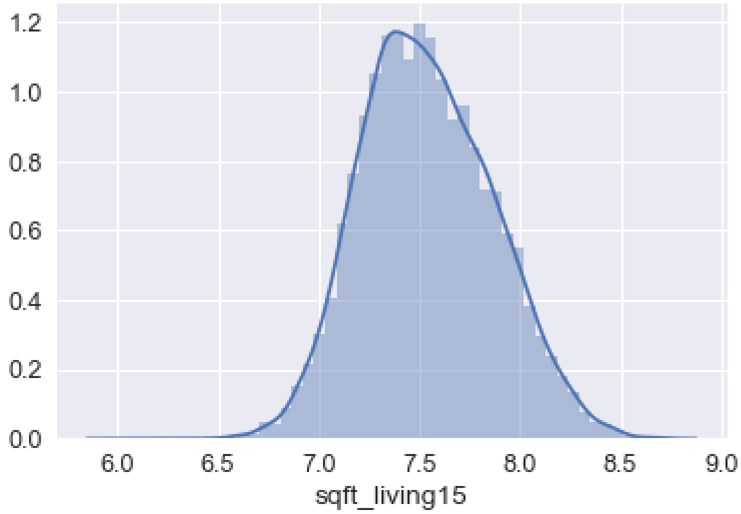


In [13]:
```
# Transformation of the variables to remove skewness

home_price['price'] = np.log(home_price['price'])
sns.distplot(home_price['price'])
plt.show()
```

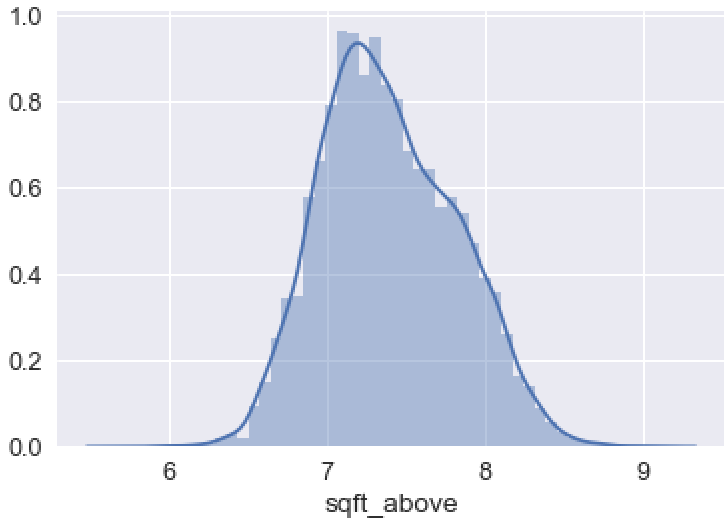


In [14]:
```
home_price['sqft_living'] = np.log(home_price['sqft_living'])
sns.distplot(home_price['sqft_living'])
plt.show()
```

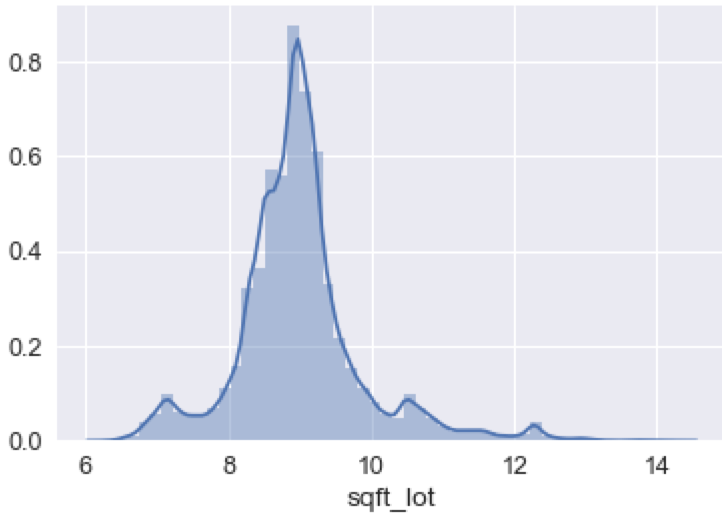
In [15]:
```
home_price['sqft_living15'] = np.log(home_price['sqft_living15'])
sns.distplot(home_price['sqft_living15'])
plt.show()
```



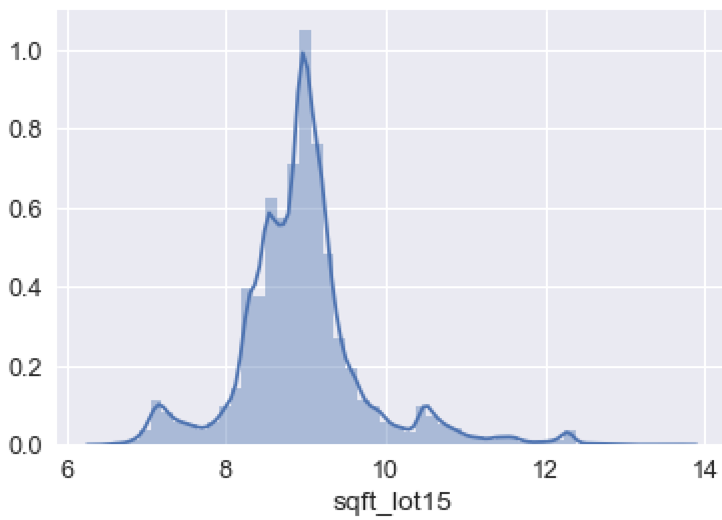
In [16]:
```
home_price['sqft_above'] = np.log(home_price['sqft_above'])
sns.distplot(home_price['sqft_above'])
plt.show()
```



In [17]:
```
home_price['sqft_lot'] = np.log(home_price['sqft_lot'])
sns.distplot(home_price['sqft_lot'])
plt.show()
```



In [18]:
```
home_price['sqft_lot15'] = np.log(home_price['sqft_lot15'])
sns.distplot(home_price['sqft_lot15'])
plt.show()
```



Defining X and y

In [19]:
```
y=home_price.iloc[:,0:1]
x=home_price.iloc[:,:]
x.drop('price',axis=1,inplace=True)
```

In [20]:
```
# Train test split

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, ran
dom_state = 1)
```

Scoring function I used make_scorer to return a score to cross_val_score for it to return the k-fold cross validation score. The greater_is_better is set to false to get sign-flipped data in scorer object. Cross validation is the technique which assesses how the results of a single analysis adapts to an independent dataset. I used cross validation to evaluate Mean Squared Error (MSE). The r2_scorer definition evaluates the r^2 score for the passed prediction.

In [3]:
```
scorer = make_scorer(mean_squared_error, greater_is_better = False)

def mse_cv_test(model):
    mse= -cross_val_score(model, x_test, y_test, scoring = scorer, cv = 5)
    return(mse)

def r2_scorer(pred):
    r2 = r2_score(y_test, pred)
    print("R2 Score: %.3f" %r2)
```

Kernalized Ridge Regression

Linear

In [22]:
```
lkrr = KernelRidge(kernel = 'linear', gamma = None, degree = None, coef0 = 1,
kernel_params = None, alpha = 1)
lkrr.fit(x_train, y_train)
lkrr_pred = lkrr.predict(x_test)
r2_scorer(lkrr_pred)
```

```
R2 Score: 0.782
```

In [23]:
```
print("Linear KRR MSE on Test set : %.3f" %(mse_cv_test(lkrr).mean()))
```

```
Linear KRR MSE on Test set : 0.063
```

Polynomial

In [ ]:
```
# lkrr = KernelRidge(kernel = 'polynomial', gamma = None, degree = 2, coef0 =
 1, kernel_params = None, alpha = 1)
# lkrr.fit(x_train, y_train)
# lkrr_pred = lkrr.predict(x_test)
# r2_scorer(lkrr_pred)
```

```
C:\Users\Anu\Anaconda3\lib\site-packages\sklearn\linear_model\ridge.py:154: U
serWarning: Singular matrix in solving dual problem. Using least-squares solu
tion instead.
  warnings.warn("Singular matrix in solving dual problem. Using "
```

In [27]:
```
ridge=linear_model.Ridge(alpha=0.1,normalize=True)
ridge.fit(x_train, y_train)
ridge_pred = ridge.predict(x_test)
r2_scorer(ridge_pred)
```

```
R2 Score: 0.778
```

In [28]:
```
print("Ridge regression MSE on Test set : %.3f" %(mse_cv_test(ridge).mean()))
```

```
Ridge regression MSE on Test set : 0.064
```

Conclusion: Based on the MSE values we can say that that Kernalized Ridge Regression performed better on the dataset with MSE of 0.063.

SVM- Linear, Polynomial, RBF(Gaussian)

In [4]:
```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.svm import SVC
```

In [5]:
```
data = load_breast_cancer()
x = data.data
y=data.target.reshape((569,1))
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, ran
dom_state = 1)
```

In [6]:
```
# Linear SVM
clf_lin = SVC(kernel='linear', C=1.0, gamma=0.1)
clf_lin.fit(x_train, y_train)
clf_lin = clf_lin.predict(x_test)
r2_scorer(clf_lin)
```

```
C:\Users\Anu\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: Dat
aConversionWarning: A column-vector y was passed when a 1d array was expecte
d. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

```
R2 Score: 0.812
```

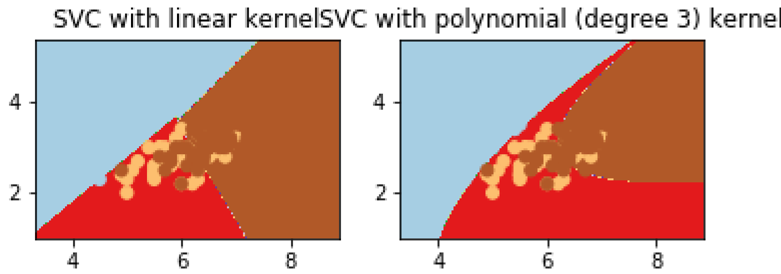
In [7]:
```
# RBF SVM
clf_rbf = SVC(kernel='rbf', C=1.0, gamma=0.1)
clf_rbf.fit(x_train, y_train)
clf_rbf = clf_rbf.predict(x_test)
r2_scorer(clf_rbf)
```

```
R2 Score: -0.583

C:\Users\Anu\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: Dat
aConversionWarning: A column-vector y was passed when a 1d array was expecte
d. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

In [ ]:
```
# # Polynomial SVM
clf_rbf = SVC(kernel='poly', gamma=2)
clf_rbf.fit(x_train, y_train)
clf_rbf = clf_rbf.predict(x_test)
r2_scorer(clf_rbf)
```

```
C:\Users\Anu\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: Dat
aConversionWarning: A column-vector y was passed when a 1d array was expecte
d. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

In [32]:
```
import numpy as np
import pylab as pl
from sklearn.datasets import load_iris
from sklearn.svm import SVC

# import some data to play with
iris = load_iris()
X = iris.data[:, :2] # we only take the first two features. We could
                     # avoid this ugly slicing by using a two-dim dataset
Y = iris.target

h=.02 # step size in the mesh

# we create an instance of SVM and fit out data. We do not scale our
# data since we want to plot the support vectors
svc     = SVC(kernel='linear').fit(X, Y)
rbf_svc = SVC(kernel='poly').fit(X, Y)
# nu_svc  = NuSVC(kernel='linear').fit(X,Y)
#lin_svc = LinearSVC().fit(X, Y)

# create a mesh to plot in
x_min, x_max = X[:,0].min()-1, X[:,0].max()+1
y_min, y_max = X[:,1].min()-1, X[:,1].max()+1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                     np.arange(y_min, y_max, h))

# title for the plots
titles = ['SVC with linear kernel',
          'SVC with polynomial (degree 3) kernel']


pl.set_cmap(pl.cm.Paired)

for i, clf in enumerate((svc, rbf_svc)):
    # Plot the decision boundary. For that, we will asign a color to each
    # point in the mesh [x_min, m_max]x[y_min, y_max].
    pl.subplot(2, 2, i+1)
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])

    # Put the result into a color plot
    Z = Z.reshape(xx.shape)
    pl.set_cmap(pl.cm.Paired)
    pl.contourf(xx, yy, Z)
    pl.axis('tight')

    # Plot also the training points
    pl.scatter(X[:,0], X[:,1], c=Y)

    pl.title(titles[i])


pl.show()
```


SVC with linear kernelSVC with polynomial (degree 3) kernel
4
2
4
6
8
4
2
4
6
8