In [1]:
```
#scikit learning is a package in python used for effective implementation of machine learning models
```

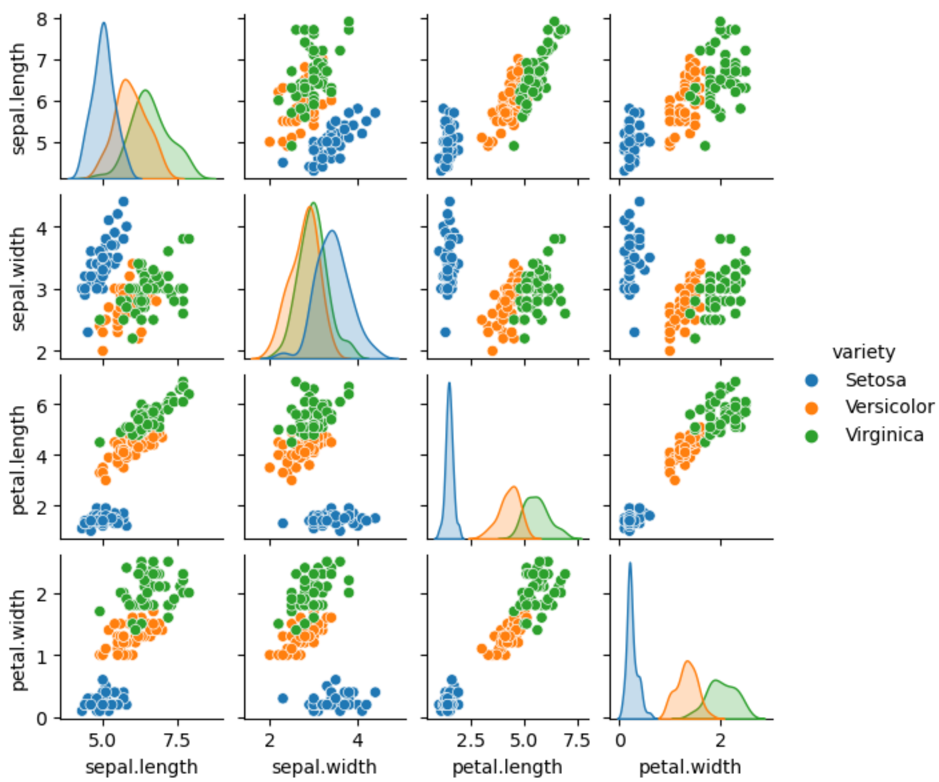
In [2]:
```
#data representation in scikit learn
#data as table
import seaborn as sns
import pandas as pd
iris = pd.read_csv("iris.csv")
iris.head()
#rows of the matrix are refered as samples and number of rows as n_samples
#columns of the matrix are refered as features and number of columns as n_features
#information can be thought of a 2 dimensional matrix which we call feature matrix represented by X
#feature matrix is assumed to be 2-dimensional with shapes[n_samples,n_features]
#samples are indivual objects described by the dataset
#features are distinct observations that describe each sample in a qualitative manner
```

Out[2]:

	sepal.length	sepal.width	petal.length	petal.width	variety
0	5.1	3.5	1.4	0.2	Setosa
1	4.9	3.0	1.4	0.2	Setosa
2	4.7	3.2	1.3	0.2	Setosa
3	4.6	3.1	1.5	0.2	Setosa
4	5.0	3.6	1.4	0.2	Setosa

In [3]:
```
#target array convention is y
#target array is one dimensional with length n_samples is usually contained in a numpy array or pandas Series
#target array is the qunatity we want to predict from the data,it is the dependent variable
```

In [4]:
```
%matplotlib inline
import seaborn as sns
sns.pairplot(iris, hue='variety',height=1.5);
```



In [5]:
```
X_iris=iris.drop("variety",axis=1)
print(X_iris.shape)
y_iris=iris["variety"]
print(y_iris.shape)
```

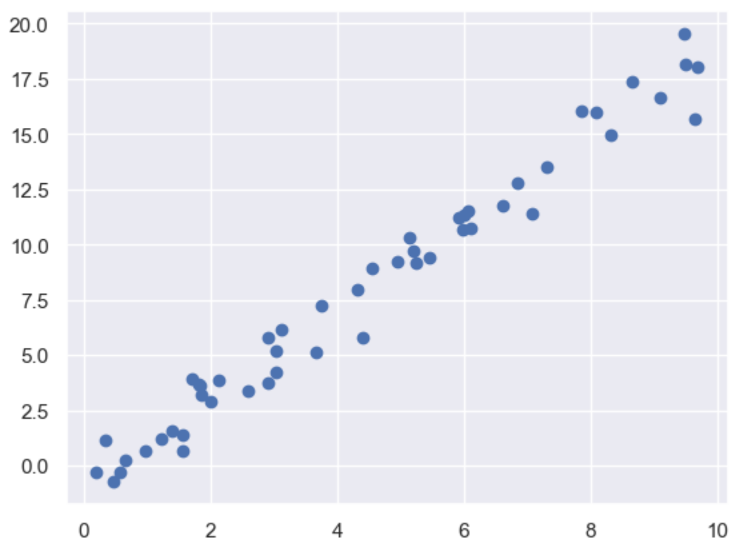
```
(150, 4)
(150,)
```

In [6]:
```
#fundamental principles of scikit-Learn
#Consistency-all objects share a common interface with limited methods with same documentations
#Insepction-All specified parameters are specified as public attributes
#Limited object hierachy-Only alogrithms represented by python classes,
#datasets are represented in standard formats(numpy,pandas or scipy sparse matrices)
#paramters names use standard python strings
#composition-many machine learning takss are expressed as seqeunces of more fundamental algorithms
#sensible defaults-when models require user-specified parameters ,the library defines an appropriate default value
```

In [7]:
```
#steps for using scikit learn API
#choose a class of model by importing appropriate estimator class from scikit-learn
#choose model hyperparamters by instantiating this class with desired values
#arrange data into features matrix and target array
#Fit model into data by calling the fit() method of the model instance
#apply model to new data
#--->for supervised learning we predict labels for unknown data using predict method
#--->for unsupervised learning we often transform or infer properties from data using transform() or predict() method
```

In [8]:
```
import matplotlib.pyplot as plt
import numpy as np
rng=np.random.RandomState(42)
x=10*rng.rand(50)
y=2*x-1+rng.randn(50)
sns.set()
plt.scatter(x,y)
```

Out[8]: <matplotlib.collections.PathCollection at 0x1ce2630f4c0>



In [9]:
```
#choosing class model by importing appropriate API
from sklearn.linear_model import LinearRegression
```

In [10]:
```
#creating an instance of model by specifying hyperparamters
model=LinearRegression(fit_intercept=True)
model
```

Out[10]: LinearRegression()

In [11]:
```
#we need to make feature matrix and target array,y is already in correct shape
#we need to reshape x to a one dimensional array
X=x.reshape((50,1))
```

In [12]:
```
#fitting model to existing data
model.fit(X,y)
```

Out[12]: LinearRegression()

In [13]:
```
model.coef_#slope
```

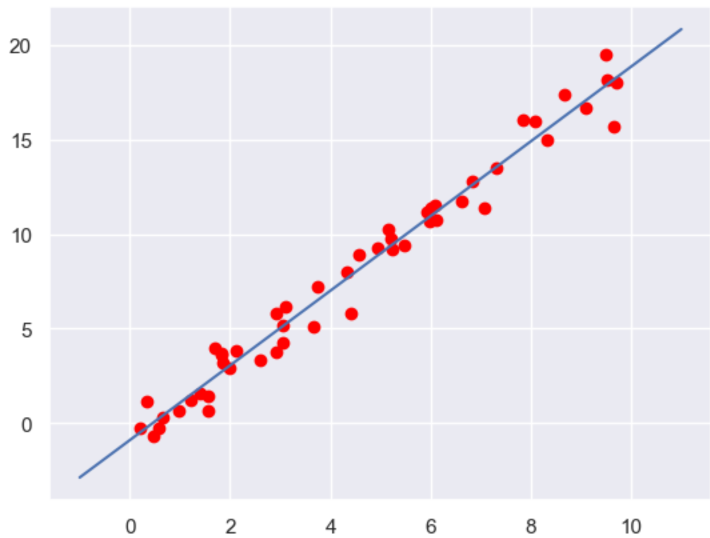
Out[13]: array([1.9776566])

In [14]:
```
model.intercept_
```

Out[14]:
```
-0.9033107255311146
```

In [15]:
```
#predicting labels for unknown data
xfit=np.linspace(-1,11)
Xfit=xfit[:,np.newaxis]#reshaping input dataset
yfit=model.predict(Xfit)
```

In [16]:
```
#visualisation by plotting both original dataset and machine driven output
plt.scatter(x,y,c="red")
plt.plot(xfit,yfit);
```



In [17]:
```
#supervised learning example-iris classification
#we will use guassian naive byes calssification which uses no hyperparamters
#we can split data into training and testing set using import train_test_split
from sklearn.model_selection import train_test_split#cross_validation is not usable in newer version
Xtrain,Xtest,Ytrain,Ytest=train_test_split(X_iris,y_iris,random_state=1)
```

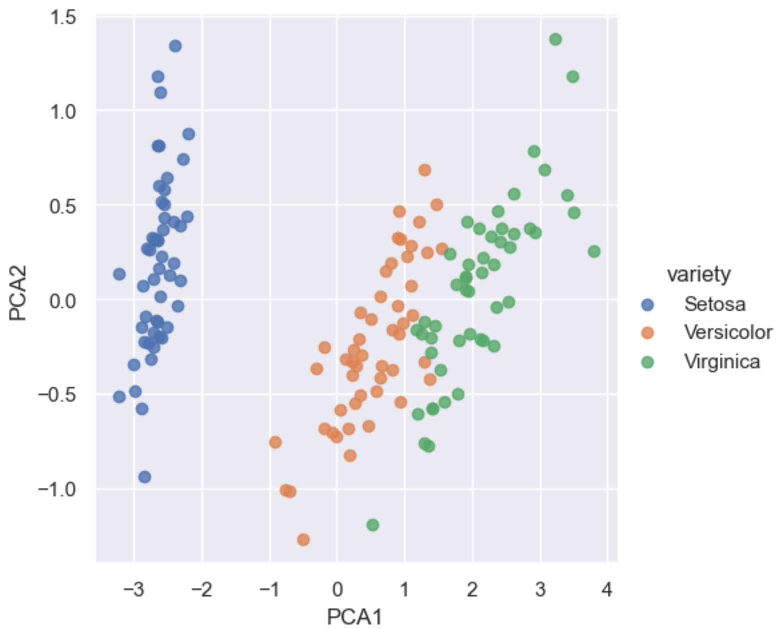
In [18]:
```
from sklearn.naive_bayes import GaussianNB #choose model class
model=GaussianNB() #instantiate model
model.fit(Xtrain,Ytrain)
y_model=model.predict(Xtest)
```

In [19]:
```
#finding the level of accuracy in data predicted
from sklearn.metrics import accuracy_score
accuracy_score(Ytest,y_model)
```

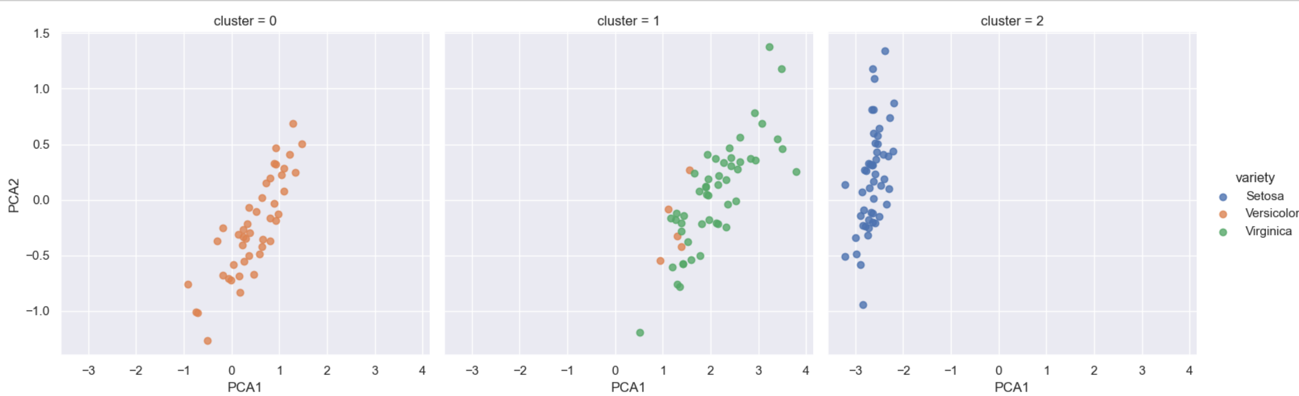
Out[19]:
```
0.9736842105263158
```

In [20]:
```
#unsupervised learning example:Iris dimensionality
#to reduce dimenisons of data in this example we use PCA(principal component analysis)
from sklearn.decomposition import PCA
model=PCA(n_components=2)#asking model to return 2 components
model.fit(X_iris)
X_2D=model.transform(X_iris)#transform method is used in dimensionality reduction
iris["PCA1"]=X_2D[:, 0]
iris["PCA2"]=X_2D[:, 1]
sns.lmplot(x="PCA1",y="PCA2",hue="variety",data=iris,fit_reg=False)
```

Out[20]: <seaborn.axisgrid.FacetGrid at 0x1ce233f5d90>



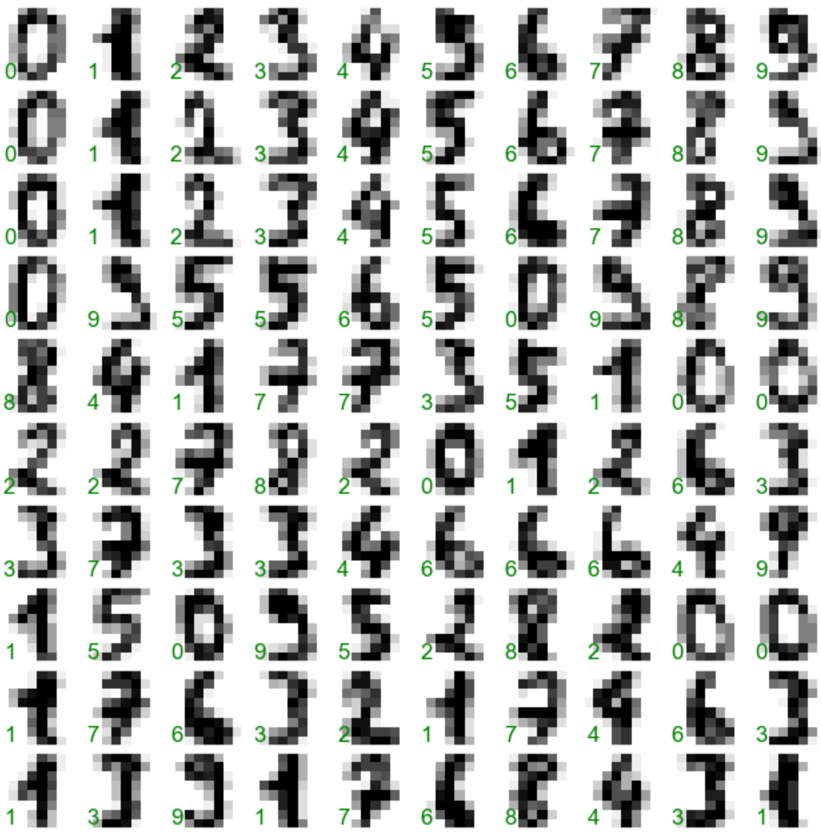
In [21]:
```
#unsupervised leanring-Iris clustering
#in this example we will use Guassian mixture model
from sklearn.mixture import GaussianMixture
model=GaussianMixture(n_components=3,covariance_type="full")
model.fit(X_iris)
y_model=model.predict(X_iris)
iris["cluster"]=y_model#creating new column for predicted cluster label
sns.lmplot(x="PCA1",y="PCA2",data=iris,hue="variety",col="cluster",fit_reg=False);#plotting using seaborn
```



In [26]:
```
#exploring hand written digits
from sklearn.datasets import load_digits
digits = load_digits()
digits.images.shape
```

Out[26]: (1797, 8, 8)

In [28]:
```
import matplotlib.pyplot as plt
fig,axes=plt.subplots(10, 10, figsize=(8, 8),subplot_kw={'xticks':[], 'yticks':[]},gridspec_kw=dict(hspace=0.1, wspace=0.1))
for i, ax in enumerate(axes.flat):
    ax.imshow(digits.images[i], cmap='binary', interpolation='nearest')
    ax.text(0.05, 0.05, str(digits.target[i]),
            transform=ax.transAxes, color='green')
```



In [30]:
```
X=digits.data
X.shape#feature matrix
```

Out[30]: (1797, 64)

In [31]:
```
y=digits.target
y.shape#target array
```

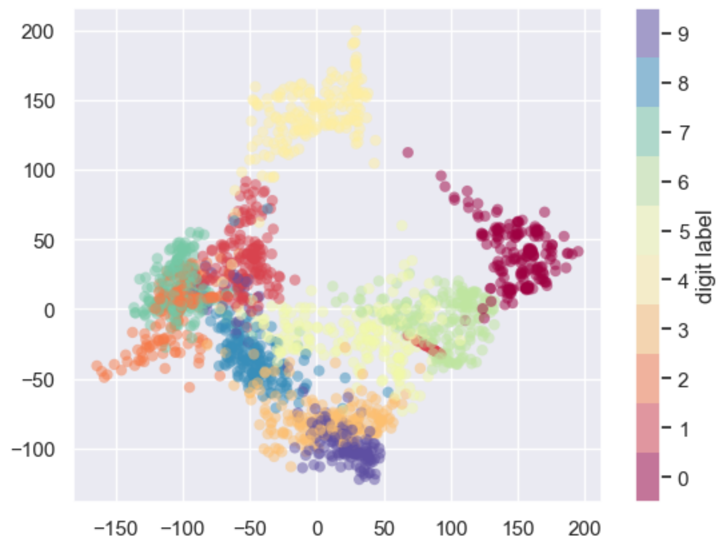
Out[31]: (1797,)

In [34]:
```
#unsupervised learning
from sklearn.manifold import Isomap
iso=Isomap(n_components=2)
iso.fit(digits.data)
data_projected=iso.transform(digits.data)
data_projected.shape
```

```
C:\Users\kdmag\anaconda3\lib\site-packages\sklearn\manifold\_isomap.py:304: UserWarning: The number of connected components of
the neighbors graph is 2 > 1. Completing the graph to fit Isomap might be slow. Increase the number of neighbors to avoid this
issue.
  self._fit_transform(X)
C:\Users\kdmag\anaconda3\lib\site-packages\scipy\sparse\_index.py:103: SparseEfficiencyWarning: Changing the sparsity structure
of a csr_matrix is expensive. lil_matrix is more efficient.
  self._set_intXint(row, col, x.flat[0])
```

Out[34]: (1797, 2)

In [38]:
```
plt.scatter(data_projected[:,0],data_projected[:,1],c=digits.target,edgecolor="none",
            alpha=0.5,cmap=plt.cm.get_cmap("Spectral",10))
plt.colorbar(label="digit label",ticks=range(10))
plt.clim(-0.5,9.5)
```



In [47]:
```
Xtrain,Xtest,ytrain,ytest=train_test_split(X,y,random_state=0)
from sklearn.naive_bayes import GaussianNB
model=GaussianNB()
model.fit(Xtrain,ytrain)
y_model=model.predict(Xtest)
```

In [48]:
```
from sklearn.metrics import accuracy_score
accuracy_score(ytest,y_model)
```

Out[48]:
```
0.8333333333333334
```

In [50]:
```
#to check where machine classification has gone wrong we have to use confusion matrix
from sklearn.metrics import confusion_matrix
mat=confusion_matrix(ytest,y_model)
sns.heatmap(mat,square=True,annot=True,cbar=False)
plt.xlabel("predicted value")
plt.ylabel("true value")
```

Out[50]:
```
Text(110.44999999999997, 0.5, 'true value')
```