```
In [0]: from sklearn.datasets import load_iris

        iris_data = load_iris()
```

```
In [0]: iris_data.keys()
Out[0]: dict_keys(['data', 'target', 'target_names', 'DESCR', 'feature_names', 'filename'])
```

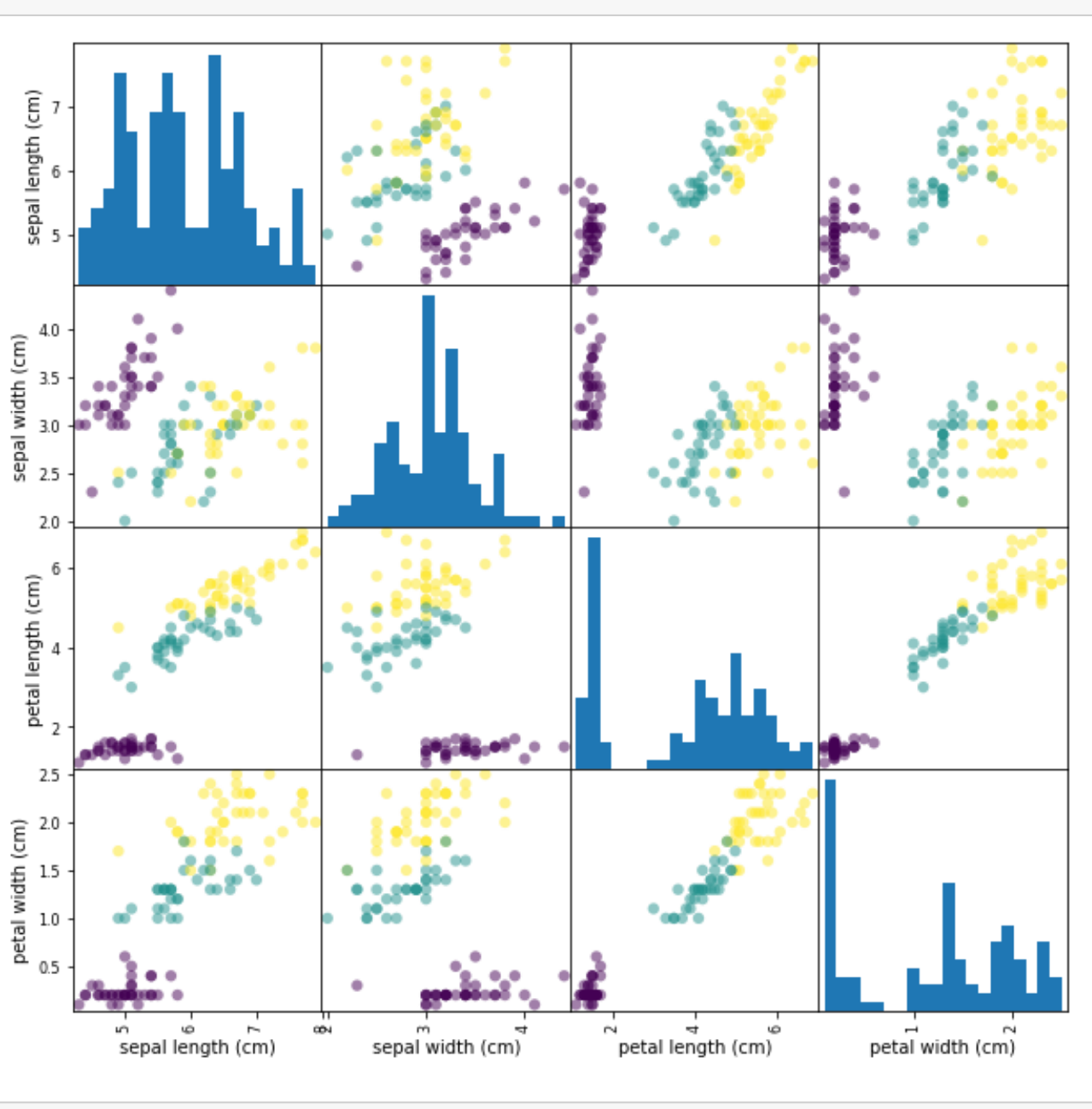
```
In [0]: # print(iris_data.DESCR)  # for details of dataset
```

```
In [0]: iris_data.feature_names
Out[0]: ['sepal length (cm)',
         'sepal width (cm)',
         'petal length (cm)',
         'petal width (cm)']
```

```
In [0]: from sklearn.model_selection import train_test_split

        X_train, X_test, y_train, y_test = train_test_split(iris_data['data'], iris_data['target'], test_siz
        e=0.25, random_state=0)
```

```
In [0]: y_train
Out[0]: array([1, 1, 2, 0, 2, 0, 0, 1, 2, 2, 2, 2, 1, 2, 1, 1, 2, 2, 2, 2, 1, 2,
               1, 0, 2, 1, 1, 1, 1, 2, 0, 0, 2, 1, 0, 0, 1, 0, 2, 1, 0, 1, 2, 1,
               0, 2, 2, 2, 2, 0, 0, 2, 2, 0, 2, 0, 2, 2, 0, 0, 2, 0, 0, 0, 1, 2,
               2, 0, 0, 0, 1, 1, 0, 0, 1, 0, 2, 1, 2, 1, 0, 2, 0, 2, 0, 0, 2, 0,
               2, 1, 1, 1, 2, 2, 1, 1, 0, 1, 2, 2, 0, 1, 1, 1, 1, 0, 0, 0, 2, 1,
               2, 0])
```

```
In [0]: import pandas as pd
        from pandas import plotting as pdplt

        iris_df = pd.DataFrame(X_train,columns=iris_data['feature_names'])
        vis = pdplt.scatter_matrix(iris_df, c=y_train, diagonal='hist', figsize=[10,10], marker='O', alpha=
        0.5, s=40, hist_kwds={'bins':20})
```



```
In [0]: from sklearn.neighbors import KNeighborsClassifier

        knn = KNeighborsClassifier(n_neighbors=1)
        knn.fit(X_train,y_train)

        perc_accu = knn.score(X_test,y_test) * 100
        print(f"perc accu : {perc_accu}")

        perc accu : 97.36842105263158
```

So using all 4 features we get 97% accuracy which is great but now let's reduce the feature space from 4D to 2D and let's see how well we do

as the above scatter matrix clearly shows that if we just take petal's length and width then the model is very easily separable

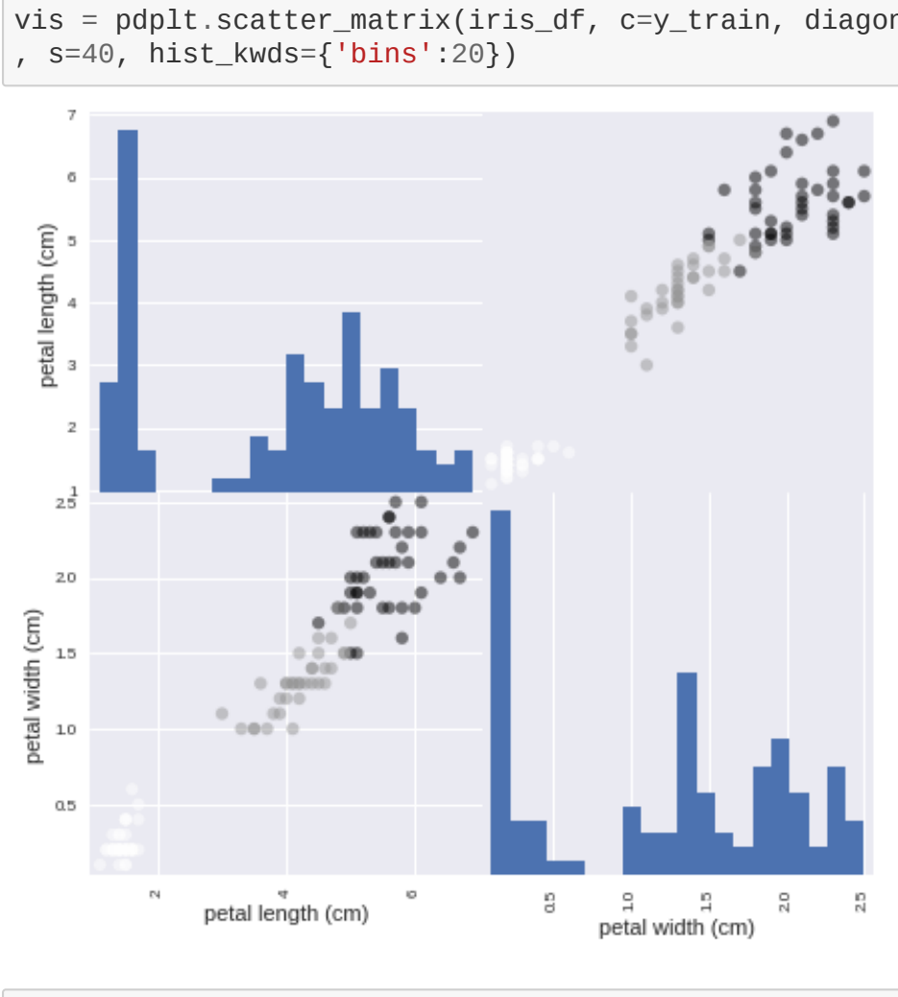
```
In [0]: iris_data = load_iris()
```

```
In [0]: iris_data.data = iris_data.data[:, 2:]
        iris_data.feature_names = iris_data.feature_names[2:]
```

```
In [0]: iris_data.feature_names
Out[0]: ['petal length (cm)', 'petal width (cm)']
```

```
In [0]: X_train, X_test, y_train, y_test = train_test_split(iris_data.data,iris_data.target,test_size=0.25,r
        andom_state=0)
```

```
In [0]: iris_df = pd.DataFrame(X_train,columns=iris_data['feature_names'])
        vis = pdplt.scatter_matrix(iris_df, c=y_train, diagonal='hist', figsize=[7,7], marker='O', alpha=0.5
        , s=40, hist_kwds={'bins':20})
```



```
In [0]: from sklearn.neighbors import KNeighborsClassifier

        knn = KNeighborsClassifier(n_neighbors=1)
        knn.fit(X_train,y_train)

        perc_accu = knn.score(X_test,y_test) * 100
        print(f"perc accu : {perc_accu}")

        perc accu : 97.36842105263158
```

So using just 2D feature space we get max accuracy of 97% which is exact same as 4D feature space gives us, so this tells us that the other two features are just redundant and adding just nothing to our model

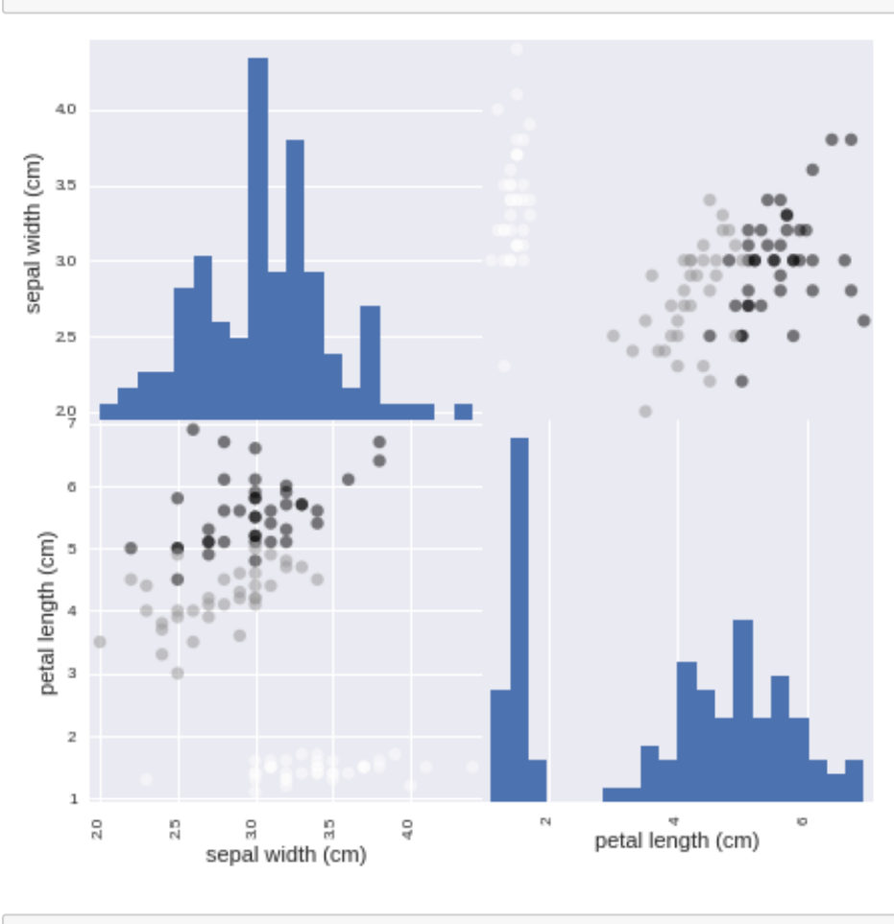
```
In [0]: iris_data = load_iris()
```

```
In [0]: iris_data.data = iris_data.data[:, 1:3]
        iris_data.feature_names = iris_data.feature_names[1:3]
```

```
In [0]: iris_data.feature_names
Out[0]: ['sepal width (cm)', 'petal length (cm)']
```

```
In [0]: X_train, X_test, y_train, y_test = train_test_split(iris_data.data,iris_data.target,test_size=0.25,r
        andom_state=0)
```

```
In [0]: iris_df = pd.DataFrame(X_train,columns=iris_data['feature_names'])
        vis = pdplt.scatter_matrix(iris_df, c=y_train, diagonal='hist', figsize=[7,7], marker='O', alpha=0.5
        , s=40, hist_kwds={'bins':20})
```



```
In [0]: from sklearn.neighbors import KNeighborsClassifier

        knn = KNeighborsClassifier(n_neighbors=1)
        knn.fit(X_train,y_train)

        perc_accu = knn.score(X_test,y_test) * 100
        print(f"perc accu : {perc_accu}")

        perc accu : 89.47368421052632
```

similarly we can also take petal length and sepal length

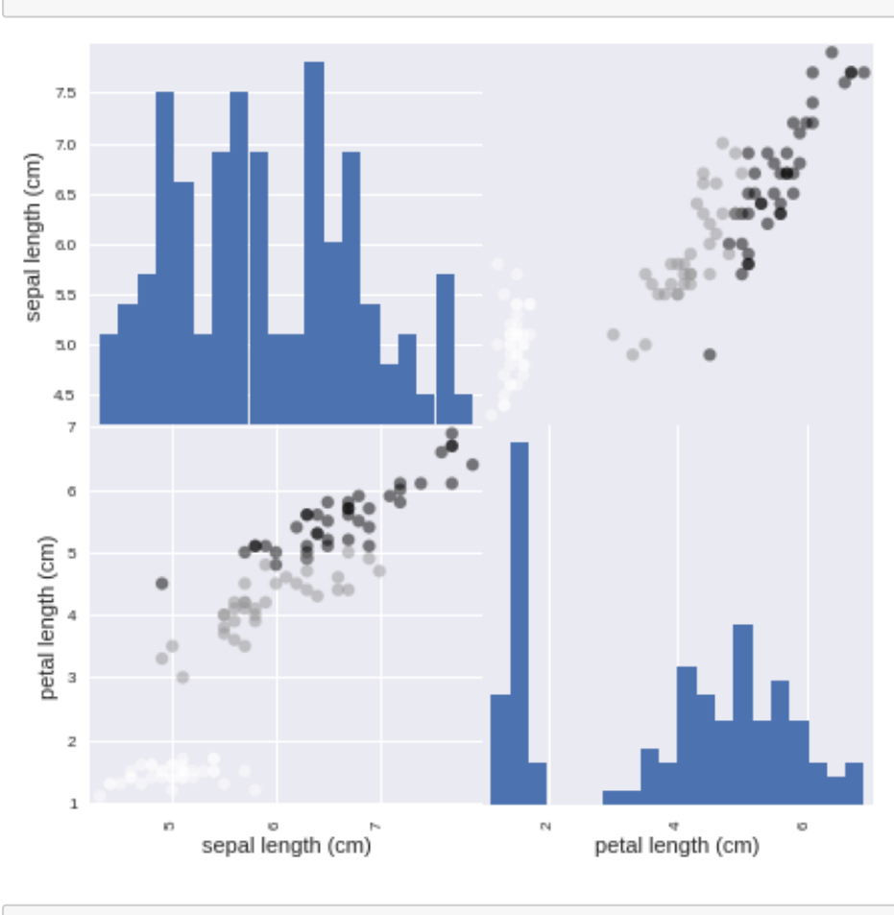
```
In [0]: iris_data=load_iris()
```

```
In [0]: iris_data.data = iris_data.data[:, [0,2]]
        iris_data.feature_names = [iris_data.feature_names[i] for i in range(4) if i%2 == 0]
```

```
In [0]: iris_data.feature_names
Out[0]: ['sepal length (cm)', 'petal length (cm)']
```

```
In [0]: X_train, X_test, y_train, y_test = train_test_split(iris_data.data,iris_data.target,test_size=0.25,r
        andom_state=0)
```

```
In [0]: iris_df = pd.DataFrame(X_train,columns=iris_data['feature_names'])
        vis = pdplt.scatter_matrix(iris_df, c=y_train, diagonal='hist', figsize=[7,7], marker='O', alpha=0.5
        , s=40, hist_kwds={'bins':20})
```



```
In [0]: from sklearn.neighbors import KNeighborsClassifier

        knn = KNeighborsClassifier(n_neighbors=1)
        knn.fit(X_train,y_train)

        perc_accu = knn.score(X_test,y_test) * 100
        print(f"perc accu : {perc_accu}")

        perc accu : 94.73684210526315
```

similarly we can also take sepal's length and width

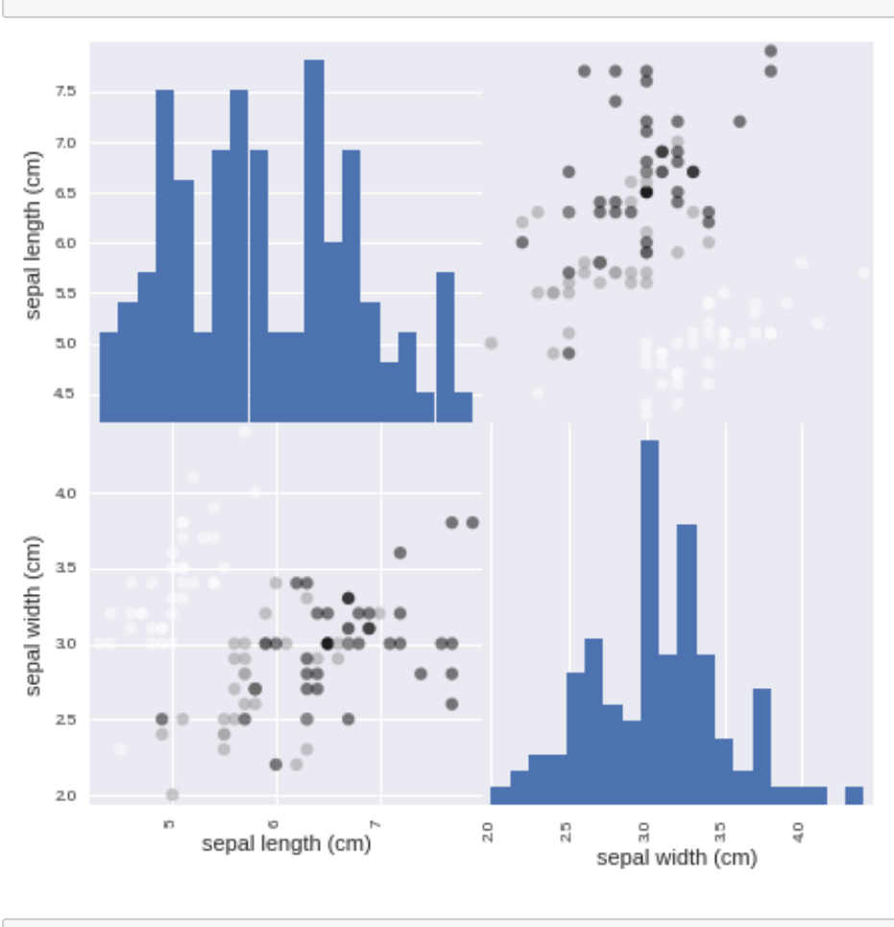
```
In [0]: iris_data = load_iris()
```

```
In [0]: iris_data.data = iris_data.data[:, [0,1]]
        iris_data.feature_names = [iris_data.feature_names[i] for i in range(4) if i <= 1]
```

```
In [0]: iris_data.feature_names
Out[0]: ['sepal length (cm)', 'sepal width (cm)']
```

```
In [0]: X_train, X_test, y_train, y_test = train_test_split(iris_data.data,iris_data.target,test_size=0.25,r
        andom_state=0)
```

```
In [0]: iris_df = pd.DataFrame(X_train,columns=iris_data['feature_names'])
        vis = pdplt.scatter_matrix(iris_df, c=y_train, diagonal='hist', figsize=[7,7], marker='O', alpha=0.5
        , s=40, hist_kwds={'bins':20})
```



```
In [0]: from sklearn.neighbors import KNeighborsClassifier

        knn = KNeighborsClassifier(n_neighbors=1)
        knn.fit(X_train,y_train)

        perc_accu = knn.score(X_test,y_test) * 100
        print(f"perc accu : {perc_accu}")

        perc accu : 71.05263157894737
```

Now as from the top most scatter matrix it is clearly evident that sepal's length and sepal's width are the worst pair to classify the dataset so when we use that as the feature than we get worse results

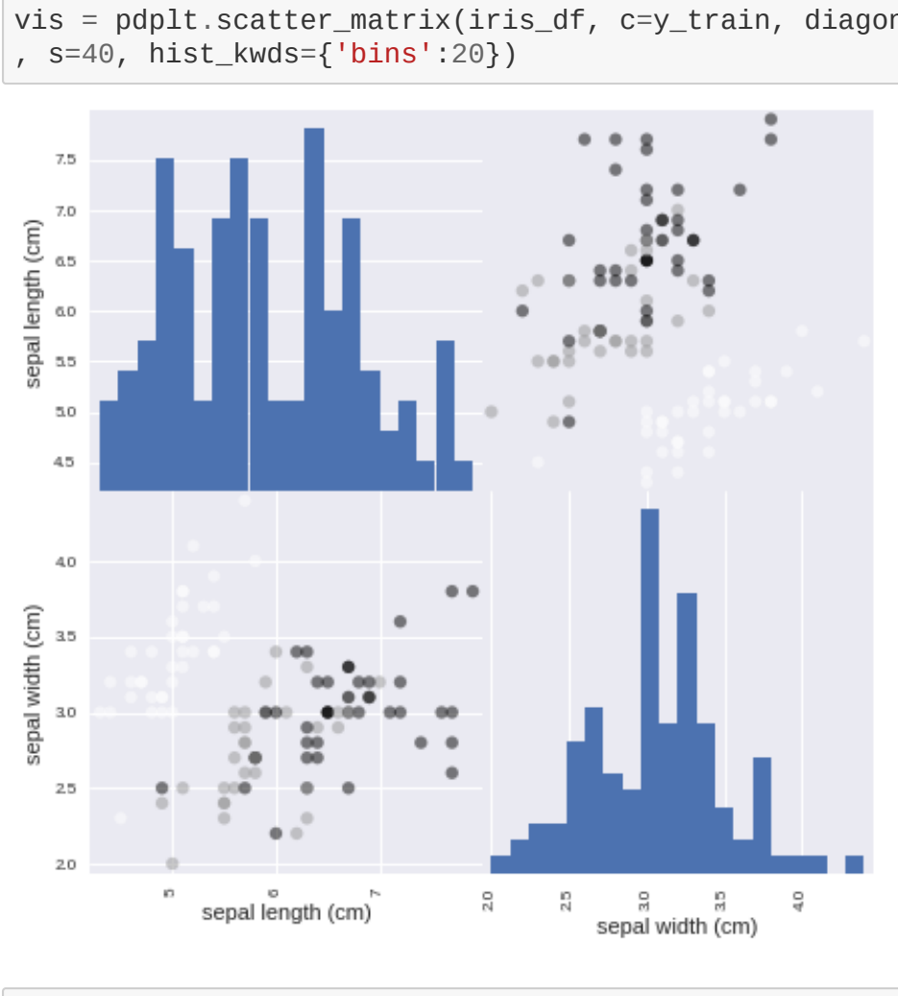
```
In [0]: iris_data = load_iris()
```

```
In [0]: iris_data.data = iris_data.data[:, :2]
        iris_data.feature_names = iris_data.feature_names[:2]
```

```
In [0]: iris_data.feature_names
Out[0]: ['sepal length (cm)', 'sepal width (cm)']
```

```
In [0]: X_train, X_test, y_train, y_test = train_test_split(iris_data.data,iris_data.target,test_size=0.25,r
        andom_state=0)
```

```
In [0]: iris_df = pd.DataFrame(X_train,columns=iris_data['feature_names'])
        vis = pdplt.scatter_matrix(iris_df, c=y_train, diagonal='hist', figsize=[7,7], marker='O', alpha=0.5
        , s=40, hist_kwds={'bins':20})
```



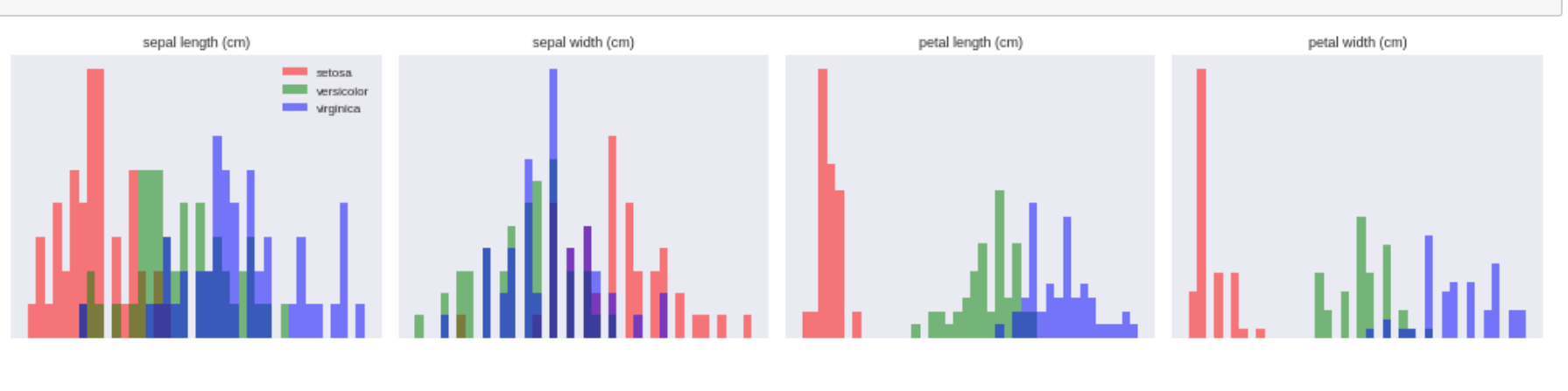
```
In [0]: from sklearn.neighbors import KNeighborsClassifier

        knn = KNeighborsClassifier(n_neighbors=1)
        knn.fit(X_train,y_train)

        perc_accu = knn.score(X_test,y_test) * 100
        print(f"perc accu : {perc_accu}")

        perc accu : 71.05263157894737
```

Until now we are dealing with 2D feature space and are succefully get great results but now try to reduce the feature space further i.e., further from 2D to 1D

```
In [0]: iris_data=load_iris()
```

```
In [0]: iris_data.target_names
Out[0]: array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

```
In [0]: # %matplotlib qt
        %matplotlib inline
        import matplotlib.pyplot as plt
        import numpy as np

        fig,axes =plt.subplots(1,4, figsize=(14, 3)) # 1 column containing 4 figures of all 4 features

        setosa=iris_data.data[iris_data.target==0]
        versicolor=iris_data.data[iris_data.target==1]
        virginica=iris_data.data[iris_data.target==2]

        ax=axes.ravel()# flat axes with numpy ravel

        for i in range(4):

            _,bins=np.histogram(iris_data.data[:,i],bins=40)

            ax[i].hist(setosa[:,i],bins=bins,color='r',alpha=0.5)
            ax[i].hist(versicolor[:,i],bins=bins,color='g',alpha=0.5)
            ax[i].hist(virginica[:,i],bins=bins,color='b',alpha=0.5)

            ax[i].set_title(iris_data.feature_names[i],fontsize=9)
            ax[i].axes.get_xaxis().set_visible(False) # the x-axis co-ordinates are not so useful, as we jus
        t want to look how well separated the histograms are
            ax[i].set_yticks(())

        ax[0].legend(['setosa','versicolor','virginica'],loc='best',fontsize=8)
        plt.tight_layout()

        plt.show()
```



Now from the above histograms it is very clear that petal's width and length are very good features to classify and sepal's both dimensions are very bad for classfying the flowers

Let's first take petal's length

```
In [0]: iris_data.data = iris_data.data[:, 2:3]
        iris_data.feature_names = iris_data.feature_names[2:3]
```

```
In [0]: iris_data.feature_names
Out[0]: ['petal length (cm)']
```

```
In [0]: X_train, X_test, y_train, y_test = train_test_split(iris_data.data,iris_data.target,test_size=0.25,r
        andom_state=0)
```

```
In [0]: X_train.shape
Out[0]: (112, 1)
```

```
In [0]: from sklearn.neighbors import KNeighborsClassifier

        knn = KNeighborsClassifier(n_neighbors=1)
        knn.fit(X_train,y_train)

        perc_accu = knn.score(X_test,y_test) * 100
        print(f"perc accu : {perc_accu}")

        perc accu : 89.47368421052632
```

we got 90% accuracy which is not bad as we are dealing with just 1 feature

Let's first take petal's width

```
In [0]: iris_data=load_iris()
```

```
In [0]: iris_data.data = iris_data.data[:, 3:4]
        iris_data.feature_names = iris_data.feature_names[-1]
```

```
In [0]: X_train, X_test, y_train, y_test = train_test_split(iris_data.data,iris_data.target,test_size=0.25,r
        andom_state=0)
```

```
In [0]: X_train.shape
Out[0]: (112, 1)
```

```
In [0]: from sklearn.neighbors import KNeighborsClassifier

        knn = KNeighborsClassifier(n_neighbors=1)
        knn.fit(X_train,y_train)

        perc_accu = knn.score(X_test,y_test) * 100
        print(f"perc accu : {perc_accu}")

        perc accu : 78.94736842105263
```

So in this case we got around 80% accuracy although it is indeed low and we will not classify our model using this feauture because it is 10% lower than previous one but if we still think intiutively this feature still doing some good job in classfying the model because if we make a classifier which randomly picks a class from three classes then the accuracy will be 33.33%, 1/3 using basic probability, so as compared to random selection this classifer is way more than double of that accuracy so it's not as bad as it seems to

Let's first take sepal's length

```
In [0]: iris_data=load_iris()
```

```
In [0]: iris_data.data = iris_data.data[:, 0:1]
        iris_data.feature_names = iris_data.feature_names[0:1]
```

```
In [0]: X_train, X_test, y_train, y_test = train_test_split(iris_data.data,iris_data.target,test_size=0.25,r
        andom_state=0)
```

```
In [0]: X_train.shape
Out[0]: (112, 1)
```

```
In [0]: from sklearn.neighbors import KNeighborsClassifier

        knn = KNeighborsClassifier(n_neighbors=1)
        knn.fit(X_train,y_train)

        perc_accu = knn.score(X_test,y_test) * 100
        print(f"perc accu : {perc_accu}")

        perc accu : 47.368421052631575
```

Now this is extremly bad and nothing more than a random selection of 1 from three (whose accuracy is 33.33%) and this is what we expect as the histogram clearly shows that this feature is bad in classfying

Let's first take sepal's width

```
In [0]: iris_data=load_iris()
```

```
In [0]: iris_data.data = iris_data.data[:, 1:2]
        iris_data.feature_names = iris_data.feature_names[1]
```

```
In [0]: iris_data.feature_names
Out[0]: 'sepal width (cm)'
```

```
In [0]: X_train, X_test, y_train, y_test = train_test_split(iris_data.data,iris_data.target,test_size=0.25,r
        andom_state=0)
```

```
In [0]: X_train.shape
Out[0]: (112, 1)
```

```
In [0]: from sklearn.neighbors import KNeighborsClassifier

        knn = KNeighborsClassifier(n_neighbors=1)
        knn.fit(X_train,y_train)

        perc_accu = knn.score(X_test,y_test) * 100
        print(f"perc accu : {perc_accu}")

        perc accu : 50.0
```

and in this case also we get expected results

Until now we are just making our hands dirty on our own visualization using scatter_matrix and histograms but now it's time for using some magic and the magic we our going to use is PCA and SVD

LET'S FIRST LOOK AT PCA

```
In [0]: iris_data = load_iris()
```

```
In [0]: # Variance ratio of the four features
        ex_variance=np.var(iris_data.data,axis=0)
        ex_variance_ratio = ex_variance/np.sum(ex_variance)

        print( ex_variance_ratio )

        [0.14994532 0.04154411 0.68145793 0.12705264]
```

```
In [0]: X_train, X_test, y_train, y_test = train_test_split(iris_data.data,iris_data.target,test_size=0.25,r
        andom_state=0)
```

```
In [0]: from sklearn.decomposition import PCA

        """
        initial 4 features will be combined to make the best 3 principal components
        these are not features but are principal compnents generated from the original 4 features
        these principal components are some complex mixture of those 4 features
        """

        pca = PCA(n_components=3)
        pca.fit(X_train)
        X_pca = pca.transform(X_train)
        X_test = pca.transform(X_test)

        print("shape of X_pca", X_pca.shape)

        shape of X_pca (112, 3)
```

```
In [0]: pca.explained_variance_ratio_
Out[0]: array([0.92743115, 0.05245721, 0.01557205])
```

the following matrix is the exact composition of features to form the PCs

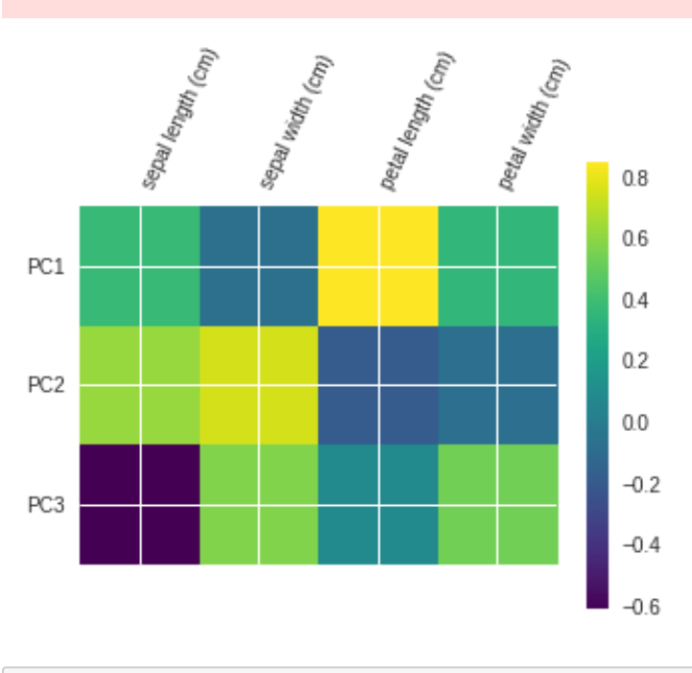
```
In [0]: pca.components_
Out[0]: array([[ 0.37649644, -0.06637905,  0.85134571,  0.35924188],
               [ 0.6240207 ,  0.75538031, -0.18479376, -0.07648543],
               [-0.60667794,  0.57674603,  0.08522779,  0.54040922]])
```

```
In [0]: df = pd.DataFrame(pca.components_, columns=iris_data.feature_names)
        df
Out[0]:
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	0.376496	-0.066379	0.851346	0.359242
1	0.624021	0.755380	-0.184794	-0.076485
2	-0.606678	0.576746	0.085228	0.540409

The following plot shows that how the PC's are composed with the given features as shown in above table, this is a magical composition of all features in different proportion which gives us these PC's which suited the model best

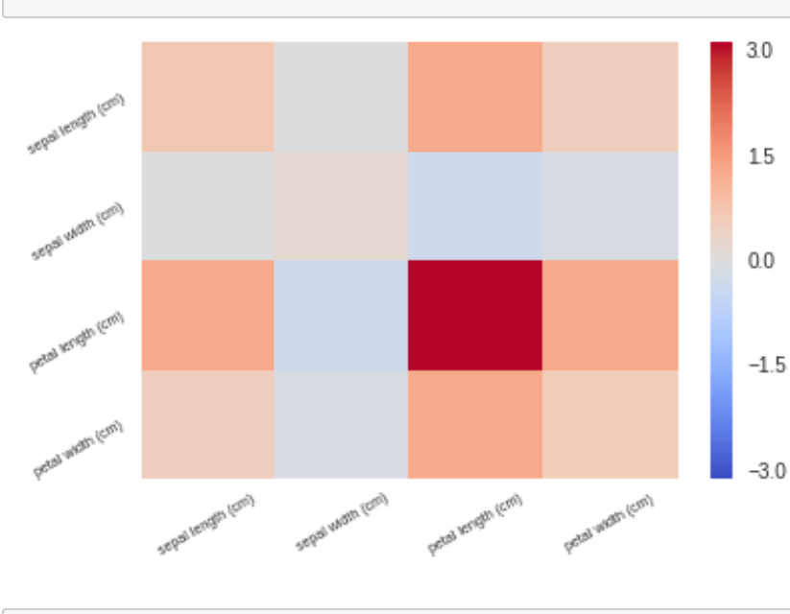
```
In [0]: """
        these 3 principal components are some complex mixture of those 4 features
        heat-plot to see how the features mixed up to create the components.
        """

        plt.matshow(pca.components_,cmap='viridis')
        plt.yticks([0,1,2],['PC1','PC2','PC3'],fontsize=10)
        plt.colorbar()
        plt.xticks(range(len(iris_data.feature_names)),iris_data.feature_names,rotation=65,ha='left')
        plt.tight_layout()
        plt.show()

        /usr/local/lib/python3.6/dist-packages/matplotlib/figure.py:2366: UserWarning: This figure includes A
        xes that are not compatible with tight_layout, so results might be incorrect.
          warnings.warn("This figure includes Axes that are not compatible "
```



```
In [0]: pca.singular_values_
Out[0]: array([22.23515693,  5.28812784,  2.88119225])
```

Below is a covariance heatmap, diagonals are the variance of that indexed feature more this variance is the more powerful is that individual feature for classifying data and the off-digonal are the relation/variance between the features i.e., how one feature vary with other feautre lower this covariance is better those features are as those feature are more likely to be orthogonal and both contributes individually

```
In [0]: import seaborn as sns
        """
         the corvariation heatmap of features
        """

        df_iris = pd.DataFrame(iris_data.data,columns=iris_data.feature_names)
        s=sns.heatmap(df_iris.cov(),cmap='coolwarm')
        s.set_yticklabels(s.get_yticklabels(),rotation=30,fontsize=7)
        s.set_xticklabels(s.get_xticklabels(),rotation=30,fontsize=7)

        plt.show()
```



```
In [0]: pca.get_covariance()
Out[0]: array([[ 0.75918838, -0.02083172,  1.39139318,  0.57112452],
               [-0.02083172,  0.19026705, -0.27999356, -0.10247426],
               [ 1.39139318, -0.27999356,  3.24250965,  1.36125483],
               [ 0.57112452, -0.10247426,  1.36125483,  0.61062741]])
```

As the most darkest spot on the digonal is the petal's length hence this is the feature which dominates our model i.e., most of the the data is varied due to petal's length and it also being verified by our previous results because using petal's length as the only predictor for the model we get around 90% accuracy.

Then in the off-diagonal petal length and petal width covar is highest which tells that they are highly related i.e., if one increases then other increases and same for decrease and also this pair has the highest covar hence classify data better than any other pair and we also saw that at the very begining

last PC contributes less so we take first two

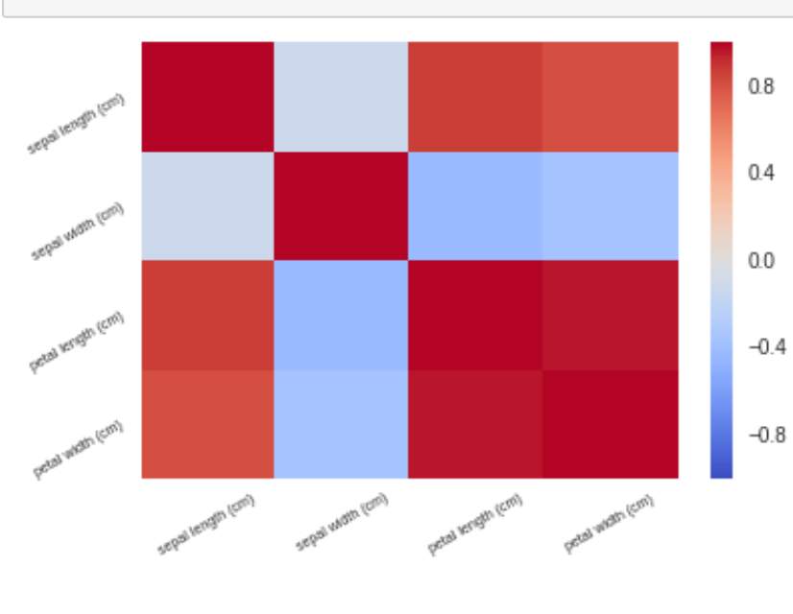
```
In [0]: X_pca = X_pca[:, 0:2]
        X_test = X_test[:, 0:2]
```

```
In [0]: from sklearn.neighbors import KNeighborsClassifier

        knn = KNeighborsClassifier(n_neighbors=4)
        knn.fit(X_pca,y_train)

        perc_accu = knn.score(X_test,y_test) * 100
        print(f"perc accu : {perc_accu}")

        perc accu : 97.36842105263158
```

The below plot clearly shows that how the first two PC's seperates the three classes in different slots which makes a clear decision boundary between them leads to an easy and efficient classification

```
In [0]: fig = plt.figure()
        ax1 = fig.add_subplot(111)

        f1x=list()
        f1y=list()
        f2x=list()
        f2y=list()
        f3x=list()
        f3y=list()

        Xax=X_pca[:,0]
        Yax=X_pca[:,1]
        # Yax = [i for i in range(len(Xax))]

        for (i,j) in zip(Xax,Yax):

            var = knn.predict([[i,j]])

            if var==0:
                f1x.append(i)
                f1y.append(j)

            elif var==1:
                f2x.append(i)
                f2y.append(j)

            else:
                f3x.append(i)
                f3y.append(j)

        ax1.scatter(f1y, f1x, s=40, alpha=0.4, c='b', marker="*", label='setosa')
        ax1.scatter(f2y, f2x, s=40, alpha=0.4, c='r', marker="o", label='versicolor')
        ax1.scatter(f3y, f3x, s=40, alpha=0.4, c='g', marker="+", label='virginica')

        plt.show()
```

```
In [0]: import seaborn as sns
        """
         the correlation heatmap of features
        """

        df_iris = pd.DataFrame(iris_data.data,columns=iris_data.feature_names)
        s=sns.heatmap(df_iris.corr(),cmap='coolwarm')
        s.set_yticklabels(s.get_yticklabels(),rotation=30,fontsize=7)
        s.set_xticklabels(s.get_xticklabels(),rotation=30,fontsize=7)

        plt.show()
```



The heatmap shows that petal length and petal width are the highest correlated pair of distinct features and this was also the case in above cells and petal length and sepal width is the least correlated or closest to orthogonal pair

USING SVD

```
In [0]: iris_data = load_iris()
```

A = USV^T

```
In [0]: A = iris_data.data
```

```
In [0]: A.shape
Out[0]: (150, 4)
```

```
In [0]: U , S , VT = np.linalg.svd(A)
```

```
In [0]: U.shape
Out[0]: (150, 150)
```

```
In [0]: VT.shape
Out[0]: (4, 4)
```

```
In [0]: S.shape
Out[0]: (4,)
```

```
In [0]: S = np.diag(S)
        Sig = np.zeros((A.shape), dtype=float)
        Sig[:S.shape[0], :S.shape[1]] = S

        Sig.shape
Out[0]: (150, 4)
```

```
In [0]: S
Out[0]: array([[95.95991387,  0.        ,  0.        ,  0.        ],
               [ 0.        , 17.76103366,  0.        ,  0.        ],
               [ 0.        ,  0.        ,  3.46093093,  0.        ],
               [ 0.        ,  0.        ,  0.        ,  1.88482631]])
```

As the first two singular values have very large values as compared to the last two, so we take them and remove the last two from our data

```
In [0]: Sig = Sig[:, :2].reshape(150,2)

        Sig.shape
Out[0]: (150, 2)
```

```
In [0]: A_new = np.matmul(U,Sig)
```

```
In [0]: A_new.shape
Out[0]: (150, 2)
```

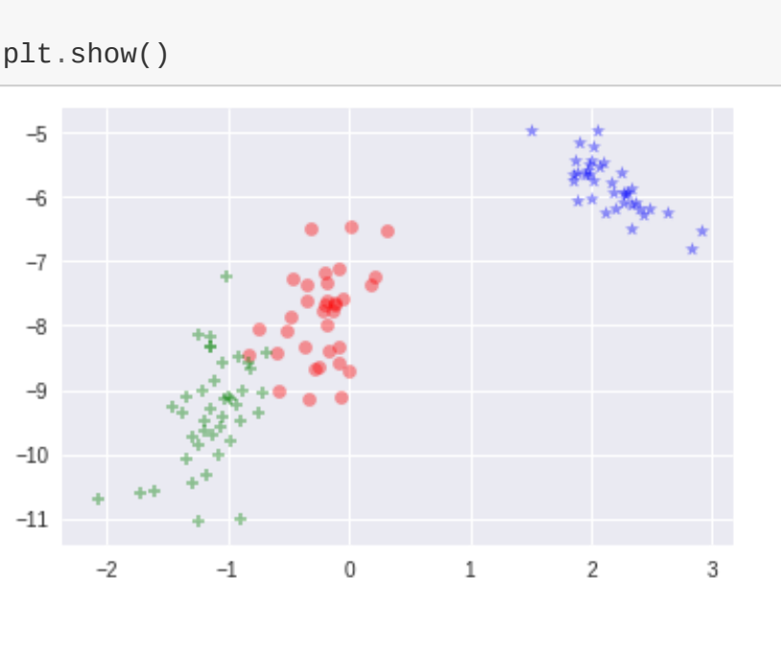
```
In [0]: X_train, X_test, y_train, y_test = train_test_split(A_new,iris_data.target,test_size=0.25,random_sta
        te=0)
```

```
In [0]: X_train.shape
Out[0]: (112, 2)
```

```
In [0]: from sklearn.neighbors import KNeighborsClassifier

        knn = KNeighborsClassifier(n_neighbors=1)
        knn.fit(X_train,y_train)

        perc_accu = knn.score(X_test,y_test) * 100
        print(f"perc accu : {perc_accu}")

        perc accu : 97.36842105263158
```

Now we visualize how well the top 2 singular values separates the classes

```
In [0]: fig = plt.figure()
        ax1 = fig.add_subplot(111)

        f1x=list()
        f1y=list()
        f2x=list()
        f2y=list()
        f3x=list()
        f3y=list()

        Xax=X_train[:,0]
        Yax=X_train[:,1]
        # Yax = [i for i in range(len(Xax))]

        for (i,j) in zip(Xax,Yax):

            var = knn.predict([[i,j]])

            if var==0:
                f1x.append(i)
                f1y.append(j)

            elif var==1:
                f2x.append(i)
                f2y.append(j)

            else:
                f3x.append(i)
                f3y.append(j)

        ax1.scatter(f1y, f1x, s=40, alpha=0.4, c='b', marker="*", label='setosa')
        ax1.scatter(f2y, f2x, s=40, alpha=0.4, c='r', marker="o", label='versicolor')
        ax1.scatter(f3y, f3x, s=40, alpha=0.4, c='g', marker="+", label='virginica')

        plt.show()
```



As it is very clear that our data is very well separates by these top 2 singular values especially setosa

Hence we get 97% accuracy using the two largest singular values. Now let's take only the first singular value

```
In [0]: Sig = Sig[:, :1].reshape(150,1)

        Sig.shape
Out[0]: (150, 1)
```

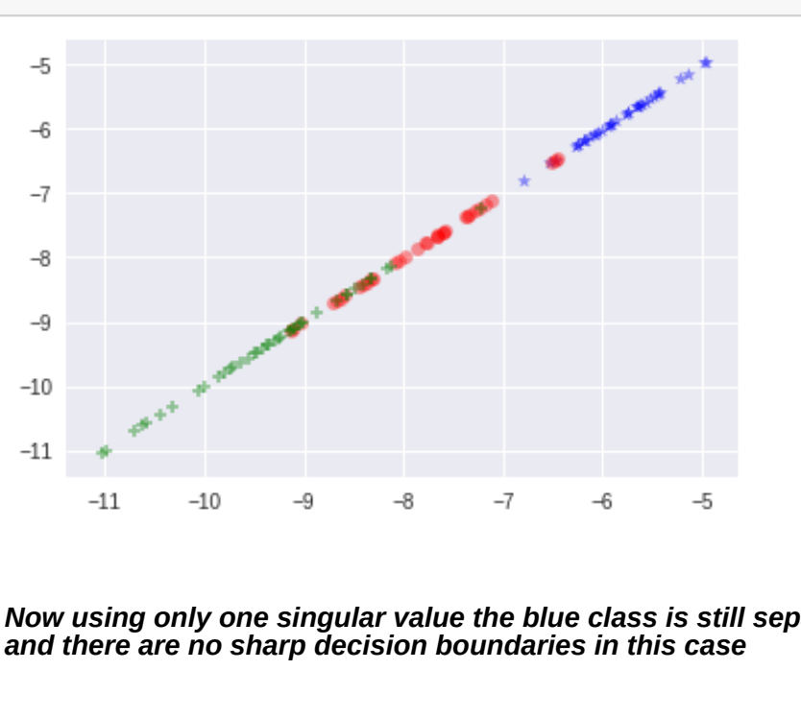
```
In [0]: A_new = np.matmul(U,Sig)
```

```
In [0]: A_new.shape
Out[0]: (150, 1)
```

```
In [0]: X_train, X_test, y_train, y_test = train_test_split(A_new,iris_data.target,test_size=0.25,random_sta
        te=0)
```

```
In [0]: from sklearn.neighbors import KNeighborsClassifier

        knn = KNeighborsClassifier(n_neighbors=1)
        knn.fit(X_train,y_train)

        perc_accu = knn.score(X_test,y_test) * 100
        print(f"perc accu : {perc_accu}")

        perc accu : 76.31578947368422
```

```
In [0]: fig = plt.figure()
        ax1 = fig.add_subplot(111)

        f1x=list()
        f1y=list()
        f2x=list()
        f2y=list()
        f3x=list()
        f3y=list()

        Xax=X_train[:,0]
        Yax=X_train[:,0]
        # Yax = [i for i in range(len(Xax))]

        for (i,j) in zip(Xax,Yax):

            var = knn.predict([[i]])

            if var==0:
                f1x.append(i)
                f1y.append(j)

            elif var==1:
                f2x.append(i)
                f2y.append(j)

            else:
                f3x.append(i)
                f3y.append(j)

        ax1.scatter(f1y, f1x, s=40, alpha=0.4, c='b', marker="*", label='setosa')
        ax1.scatter(f2y, f2x, s=40, alpha=0.4, c='r', marker="o", label='versicolor')
        ax1.scatter(f3y, f3x, s=40, alpha=0.4, c='g', marker="+", label='virginica')

        plt.show()
```



Now using only one singular value the blue class is still separable but there is very high intermixing of red and green classes and there are no sharp decision boundaries in this case

Hence we got an expected decrease in the accuracy as we avoid the second singular value which indeed is small than first but contributes resonable due to its high magnitude as compared to the remaining last two

```
In [0]:
```

```
In [0]:
```

```
In [0]:
```