Support Vector Machines Project

We will be analyzing the famous iris data set!

The Data

For this series of lectures, we will be using the famous Iris flower data set.

The Iris flower data set or Fisher's Iris data set is a multivariate data set introduced by Sir Ronald Fisher in the 1936 as an example of discriminant analysis.

The data set consists of 50 samples from each of three species of Iris (Iris setosa, Iris virginica and Iris versicolor), so 150 total samples. Four features were measured from each sample: the length and the width of the sepals and petals, in centimeters.

Here's a picture of the three different Iris types:

```
# The Iris Setosa
from IPython.display import Image
url =
'http://upload.wikimedia.org/wikipedia/commons/5/56/Kosaciec_szczecinkowaty_Iris_setosa.jpg'
Image(url,width=300, height=300)
```

```
# The Iris Versicolor
from IPython.display import Image
url =
'http://upload.wikimedia.org/wikipedia/commons/4/41/Iris_versicolor_3.
```

```
jpg'
Image(url,width=300, height=300)
```

```
# The Iris Virginica
from IPython.display import Image
url =
'http://upload.wikimedia.org/wikipedia/commons/9/9f/Iris_virginica.jpg
'
Image(url,width=300, height=300)
```

The iris dataset contains measurements for 150 iris flowers from three different species.

The three classes in the Iris dataset:

```
Iris-setosa (n=50)
Iris-versicolor (n=50)
Iris-virginica (n=50)
```

The four features of the Iris dataset:

```
sepal length in cm
sepal width in cm
petal length in cm
petal width in cm
```

Get the data

Use seaborn to get the iris data by using: iris = sns.load_dataset('iris')

```
import seaborn as sns

# Load the Iris dataset
iris = sns.load_dataset('iris')

# Display the first few rows of the dataset
print(iris.head())
```

```
   sepal_length  sepal_width  petal_length  petal_width species
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
```

Let's visualize the data and get you started!

Exploratory Data Analysis

Time to put your data viz skills to the test! Try to recreate the following plots, make sure to import the libraries you'll need!

Import some libraries you think you'll need.

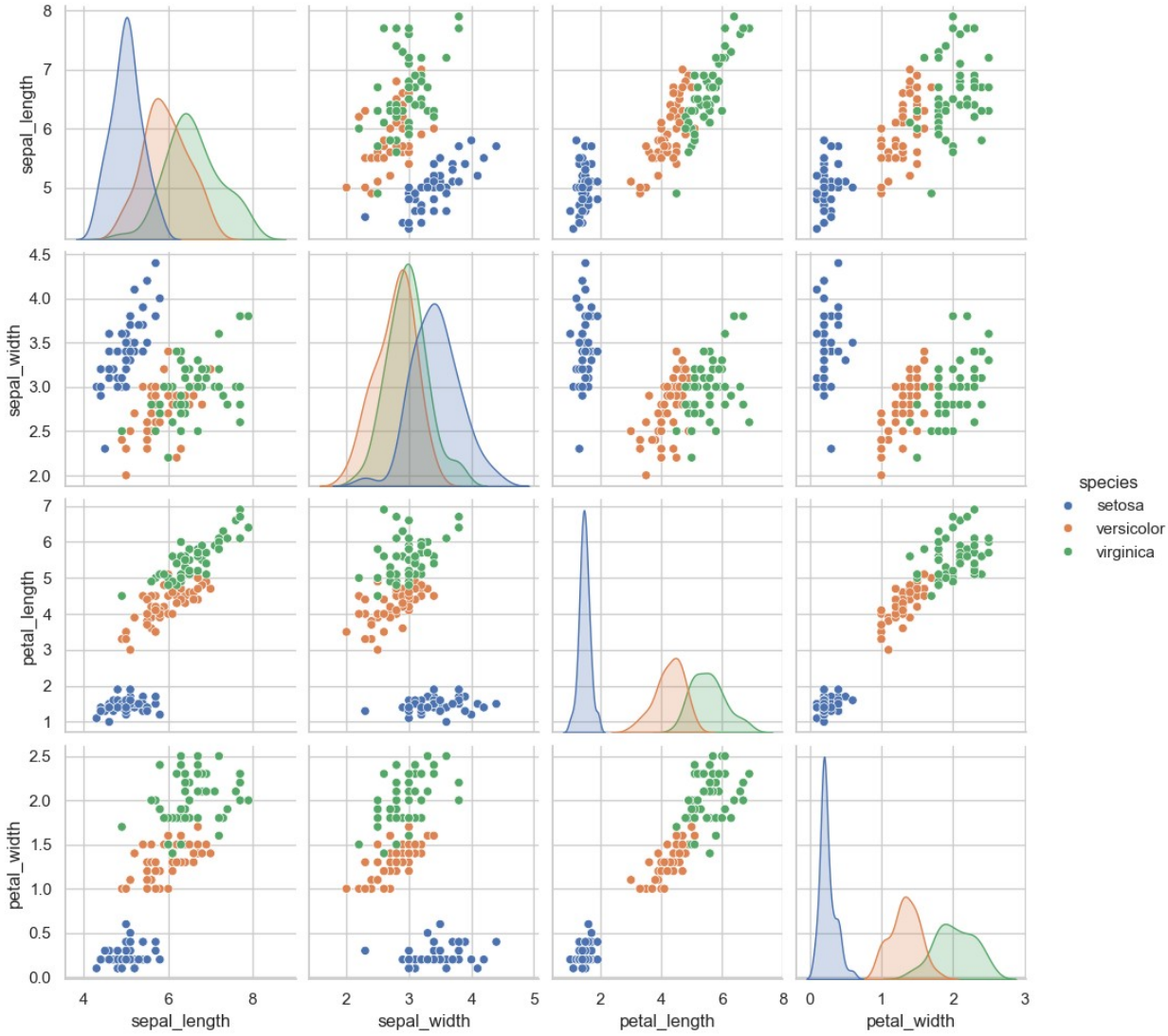
```
import seaborn as sns
import matplotlib.pyplot as plt

# Set the style for the plots
sns.set(style="whitegrid")

# Optional: To display the plots in Jupyter Notebook
%matplotlib inline
```

** Create a pairplot of the data set. Which flower species seems to be the most separable?**

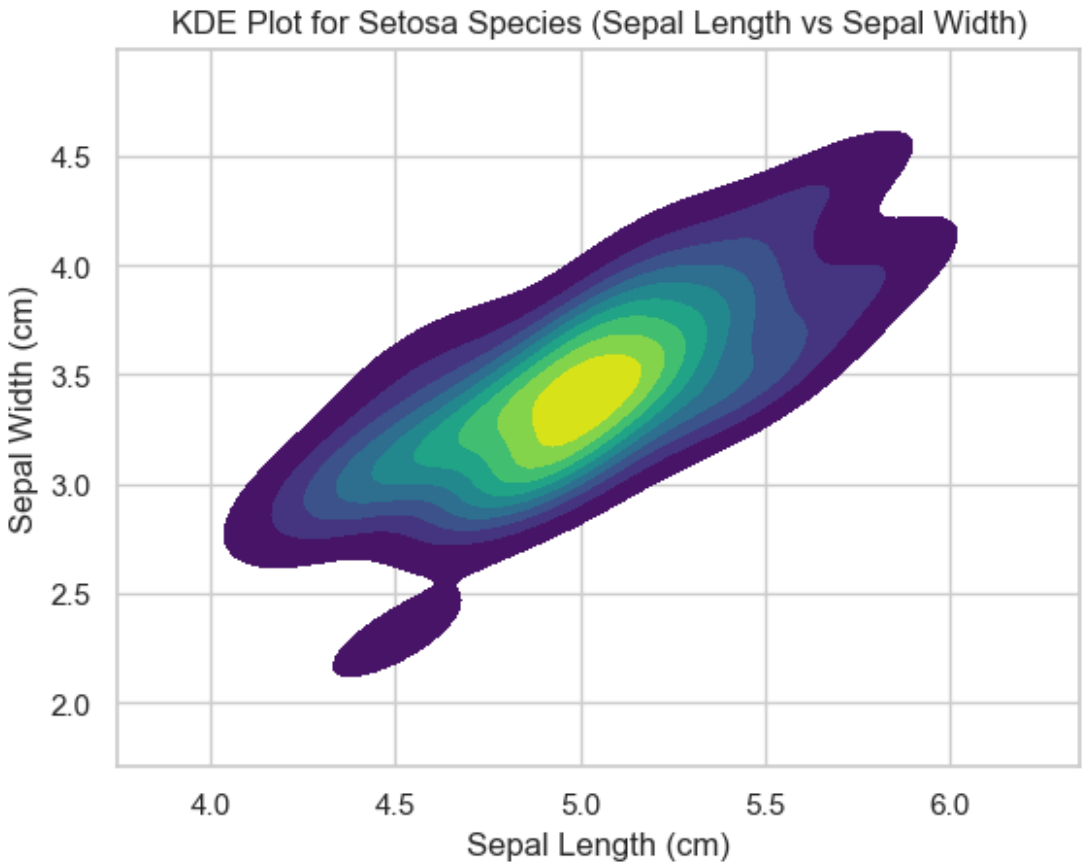
```
# Create a pairplot
sns.pairplot(iris, hue='species')
plt.show()
```



Create a kde plot of sepal_length versus sepal width for setosa species of flower.

```
# Filter the data for Setosa species
setosa_data = iris[iris['species'] == 'setosa']

# Create a KDE plot
sns.kdeplot(x=setosa_data['sepal_length'],
y=setosa_data['sepal_width'], cmap='viridis', fill=True)
plt.title('KDE Plot for Setosa Species (Sepal Length vs Sepal Width)')
plt.xlabel('Sepal Length (cm)')
plt.ylabel('Sepal Width (cm)')
plt.show()
```



Train Test Split

** Split your data into a training set and a testing set.**

```
from sklearn.model_selection import train_test_split

# Features (X) and Target (y)
X = iris.drop('species', axis=1)
y = iris['species']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)

# Display the shapes of the resulting sets
print("Training set shape:", X_train.shape, y_train.shape)
print("Testing set shape:", X_test.shape, y_test.shape)
```

```
Training set shape: (105, 4) (105,)
Testing set shape: (45, 4) (45,)
```

Train a Model

Now its time to train a Support Vector Machine Classifier.

Call the SVC() model from sklearn and fit the model to the training data.

```
from sklearn.svm import SVC

# Create an SVM classifier
svm_model = SVC()

# Fit the model to the training data
svm_model.fit(X_train, y_train)
```

```
SVC()
```

Model Evaluation

Now get predictions from the model and create a confusion matrix and a classification report.

```
from sklearn.metrics import confusion_matrix, classification_report

from sklearn.svm import SVC

# Create an SVM classifier with specific parameters
svm_model = SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
                decision_function_shape='ovr',  # Set this parameter
to 'ovo' or 'ovr'
                degree=3, gamma='auto', kernel='rbf',
                max_iter=-1, probability=False, random_state=None,
shrinking=True,
                tol=0.001, verbose=False)

# Fit the model to the training data
svm_model.fit(X_train, y_train)

# Get predictions from the model
predictions = svm_model.predict(X_test)

# Create a confusion matrix
conf_matrix = confusion_matrix(y_test, predictions)

# Create a classification report
class_report = classification_report(y_test, predictions)

# Display the results
print("Confusion Matrix:\n", conf_matrix)
print("\nClassification Report:\n", class_report)
```

```
Confusion Matrix:
 [[19  0  0]
 [ 0 13  0]
 [ 0  0 13]]

Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        19
  versicolor       1.00      1.00      1.00        13
   virginica       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45
```

```
# Display the results
print("Confusion Matrix:\n", conf_matrix)
print("\nClassification Report:\n", class_report)
```

```
Confusion Matrix:
 [[19  0  0]
 [ 0 13  0]
 [ 0  0 13]]

Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        19
  versicolor       1.00      1.00      1.00        13
   virginica       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45
```

Wow! You should have noticed that your model was pretty good! Let's see if we can tune the parameters to try to get even better (unlikely, and you probably would be satisfied with these results in real like because the data set is quite small, but I just want you to practice using GridSearch.

Gridsearch Practice

** Import GridsearchCV from SciKit Learn.**

```
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix, classification_report
```

Create a dictionary called param_grid and fill out some parameters for C and gamma.

```
# Define the parameter grid
param_grid = {'C': [0.1, 1, 10, 100],
              'gamma': [1, 0.1, 0.01, 0.001],
              'kernel': ['linear', 'rbf', 'poly']}

# Create an SVM classifier
svm_model = SVC()
```

** Create a GridSearchCV object and fit it to the training data.**

```
# Create a GridSearchCV object
grid_search = GridSearchCV(svm_model, param_grid, cv=3, verbose=2,
n_jobs=-1)

# Fit the GridSearchCV object to the training data
grid_search.fit(X_train, y_train)

# Get the best parameters and the best model
best_params = grid_search.best_params_
best_model = grid_search.best_estimator_

# Use the best model for predictions
best_predictions = best_model.predict(X_test)

# Create a confusion matrix
best_conf_matrix = confusion_matrix(y_test, best_predictions)

# Create a classification report
best_class_report = classification_report(y_test, best_predictions)

# Display the results
print("Best Parameters:\n", best_params)
print("\nBest Confusion Matrix:\n", best_conf_matrix)
print("\nBest Classification Report:\n", best_class_report)
```

```
Fitting 3 folds for each of 48 candidates, totalling 144 fits
Best Parameters:
 {'C': 100, 'gamma': 0.01, 'kernel': 'rbf'}

Best Confusion Matrix:
 [[19  0  0]
 [ 0 13  0]
 [ 0  0 13]]

Best Classification Report:
              precision    recall  f1-score   support
```

```
      setosa       1.00      1.00      1.00        19
  versicolor       1.00      1.00      1.00        13
   virginica       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45
```

** Now take that grid model and create some predictions using the test set and create classification reports and confusion matrices for them. Were you able to improve?**

```
# Use the best model for predictions on the test set
grid_predictions = best_model.predict(X_test)

# Create a confusion matrix for the GridSearchCV model
grid_conf_matrix = confusion_matrix(y_test, grid_predictions)

# Create a classification report for the GridSearchCV model
grid_class_report = classification_report(y_test, grid_predictions)

# Display the results for the GridSearchCV model
print("Confusion Matrix (GridSearchCV Model):\n", grid_conf_matrix)
print("\nClassification Report (GridSearchCV Model):\n",
grid_class_report)
```

```
Confusion Matrix (GridSearchCV Model):
 [[19  0  0]
 [ 0 13  0]
 [ 0  0 13]]

Classification Report (GridSearchCV Model):
               precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        19
  versicolor       1.00      1.00      1.00        13
   virginica       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45
```

You should have done about the same or exactly the same, this makes sense, there is basically just one point that is too noisey to grab, which makes sense, we don't want to have an overfit model that would be able to grab that.