Notebook Setup

```
!which python3
```

```
/Library/Frameworks/Python.framework/Versions/3.11/bin/python3
```

```
%load_ext autoreload
%autoreload 2
%matplotlib inline
# these are just couple extensions to help with certain things
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

```
 # Standard imports
import os

# Third-party imports
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Local imports

sns.set() # this will make the notebook use seaborn plotting styles
```

Load Data

```
data = pd.read_csv('IRIS.csv')
```

```
data.columns
```

```
Index(['sepal_length', 'sepal_width', 'petal_length', 'petal_width',
       'species'],
      dtype='object')
```

```
df = data.drop(columns=["species"])
df.head()
```

	sepal_length	sepal_width	petal_length	petal_width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

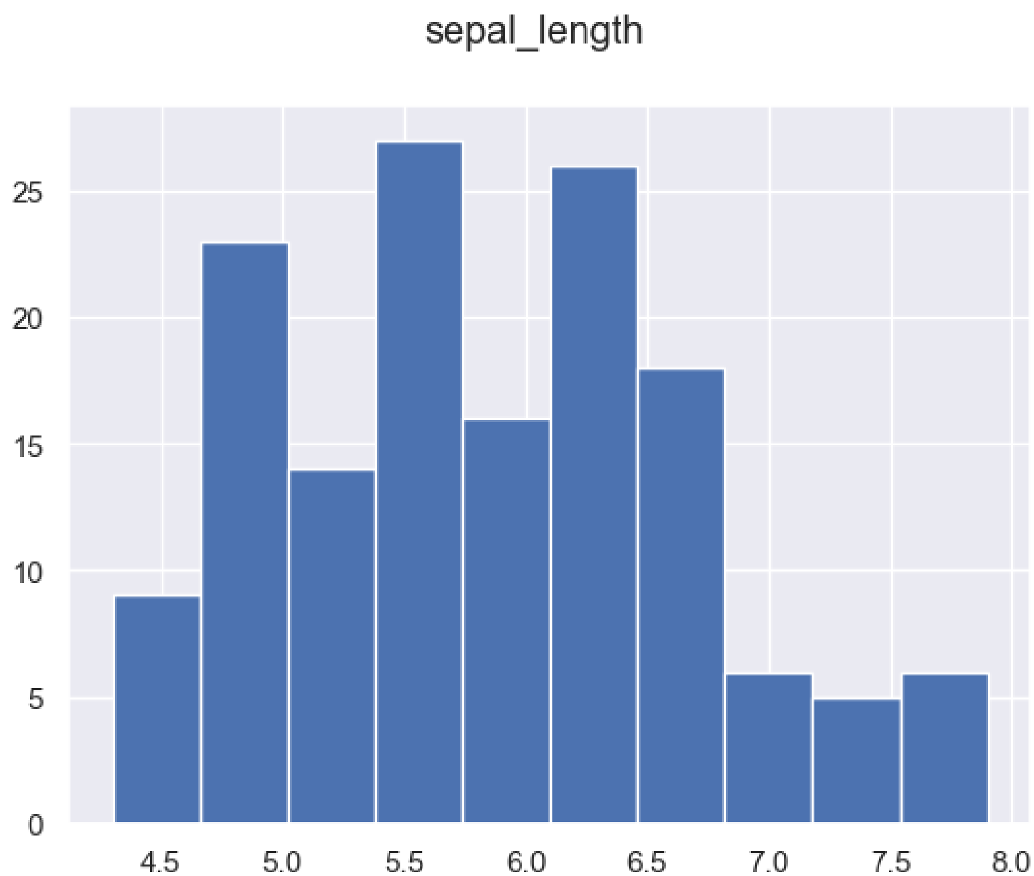
```python
df["target"] = pd.Categorical(data["species"]).codes
# df["target"] = data["species"].map({"name":0}) # this way will work too write the name and its relative value you want to use
df
```

```
     sepal_length  sepal_width  petal_length  petal_width  target
0             5.1          3.5           1.4          0.2       0
1             4.9          3.0           1.4          0.2       0
2             4.7          3.2           1.3          0.2       0
3             4.6          3.1           1.5          0.2       0
4             5.0          3.6           1.4          0.2       0
..            ...          ...           ...          ...     ...
145           6.7          3.0           5.2          2.3       2
146           6.3          2.5           5.0          1.9       2
147           6.5          3.0           5.2          2.0       2
148           6.2          3.4           5.4          2.3       2
149           5.9          3.0           5.1          1.8       2

[150 rows x 5 columns]
```

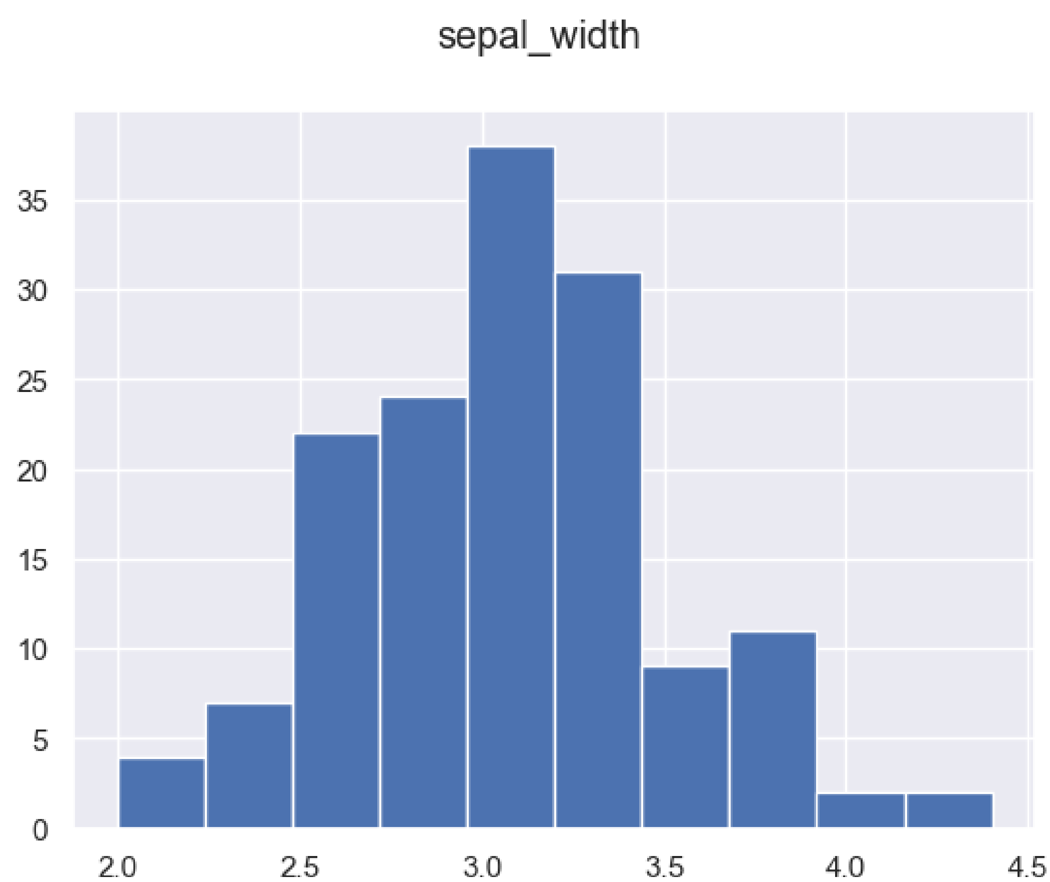
```python
df.describe()
```

```
       sepal_length  sepal_width  petal_length  petal_width
target
count    150.000000   150.000000    150.000000   150.000000
150.000000
mean       5.843333     3.054000      3.758667     1.198667
1.000000
std        0.828066     0.433594      1.764420     0.763161
0.819232
min        4.300000     2.000000      1.000000     0.100000
0.000000
25%        5.100000     2.800000      1.600000     0.300000
0.000000
50%        5.800000     3.000000      4.350000     1.300000
1.000000
75%        6.400000     3.300000      5.100000     1.800000
2.000000
max        7.900000     4.400000      6.900000     2.500000
2.000000
```

```python
df["sepal_length"].hist()
plt.suptitle("sepal_length")
plt.show()
```



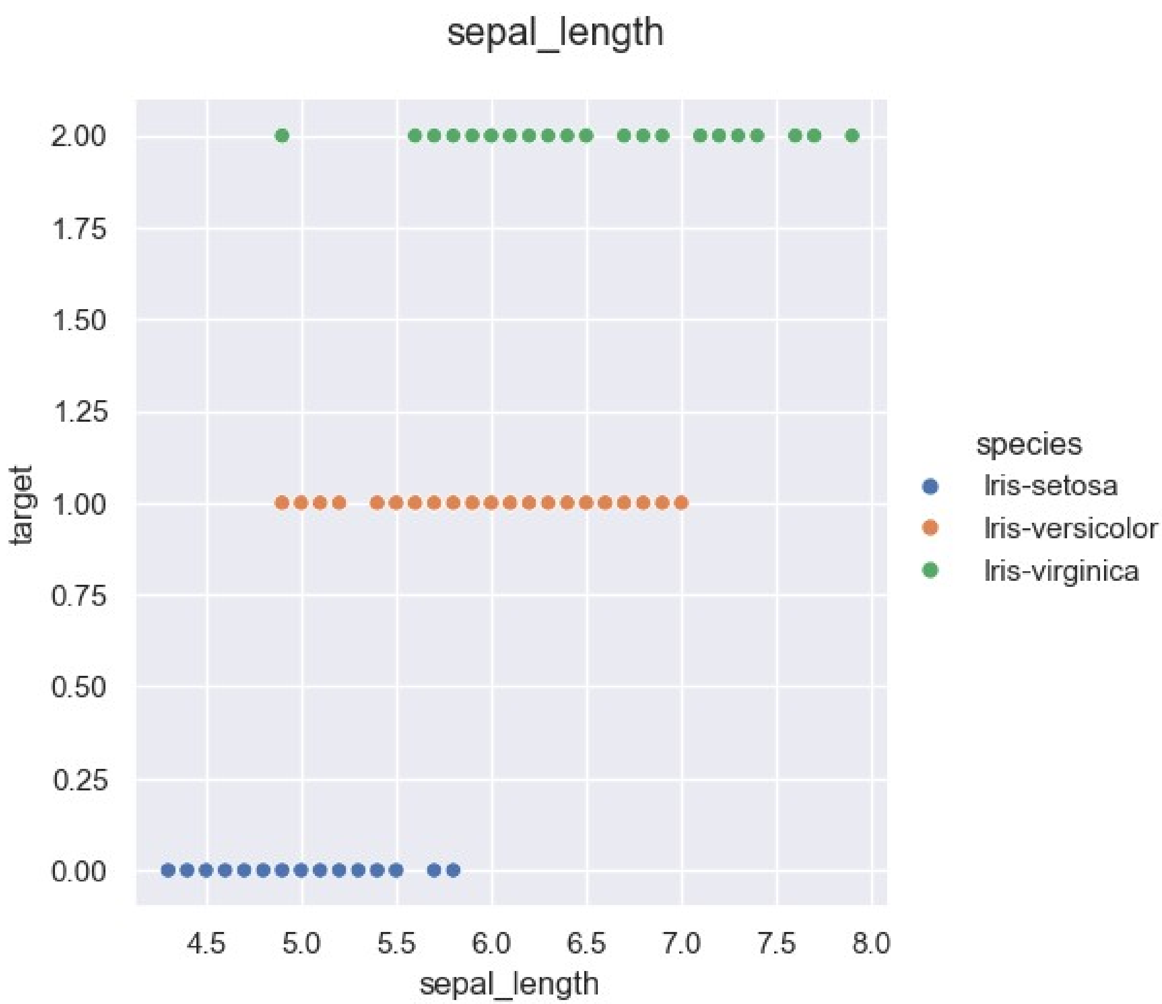

```
df["sepal_width"].hist()
plt.suptitle("sepal_width")
plt.show()
```



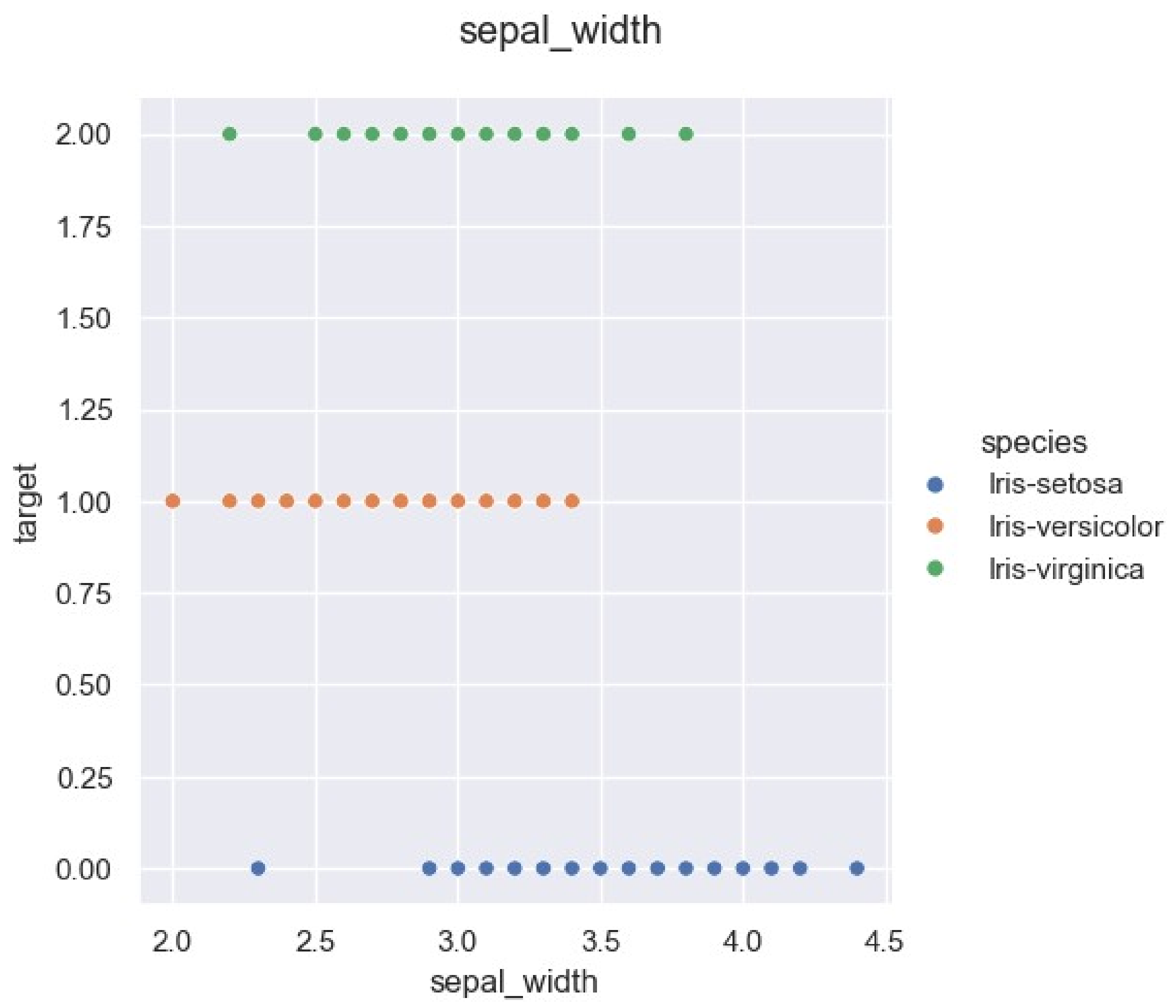

```python
# Assuming df contains your dataset
features = ["sepal_length", "sepal_width", "petal_length",
"petal_width"]

fig, axes = plt.subplots(2, 2, figsize=(10, 8))  # Create a 2x2 grid
fig.suptitle("Histograms of Features")  # Set the overall title

# Loop through features and axes
for ax, feature in zip(axes.ravel(), features):
    df[feature].hist(ax=ax)  # Plot histogram
    ax.set_title(feature)  # Set title for each subplot

plt.show()
```


```
df["target"]

0      0
1      0
2      0
3      0
4      0
      ..
145    2
146    2
147    2
148    2
149    2
Name: target, Length: 150, dtype: int8
```

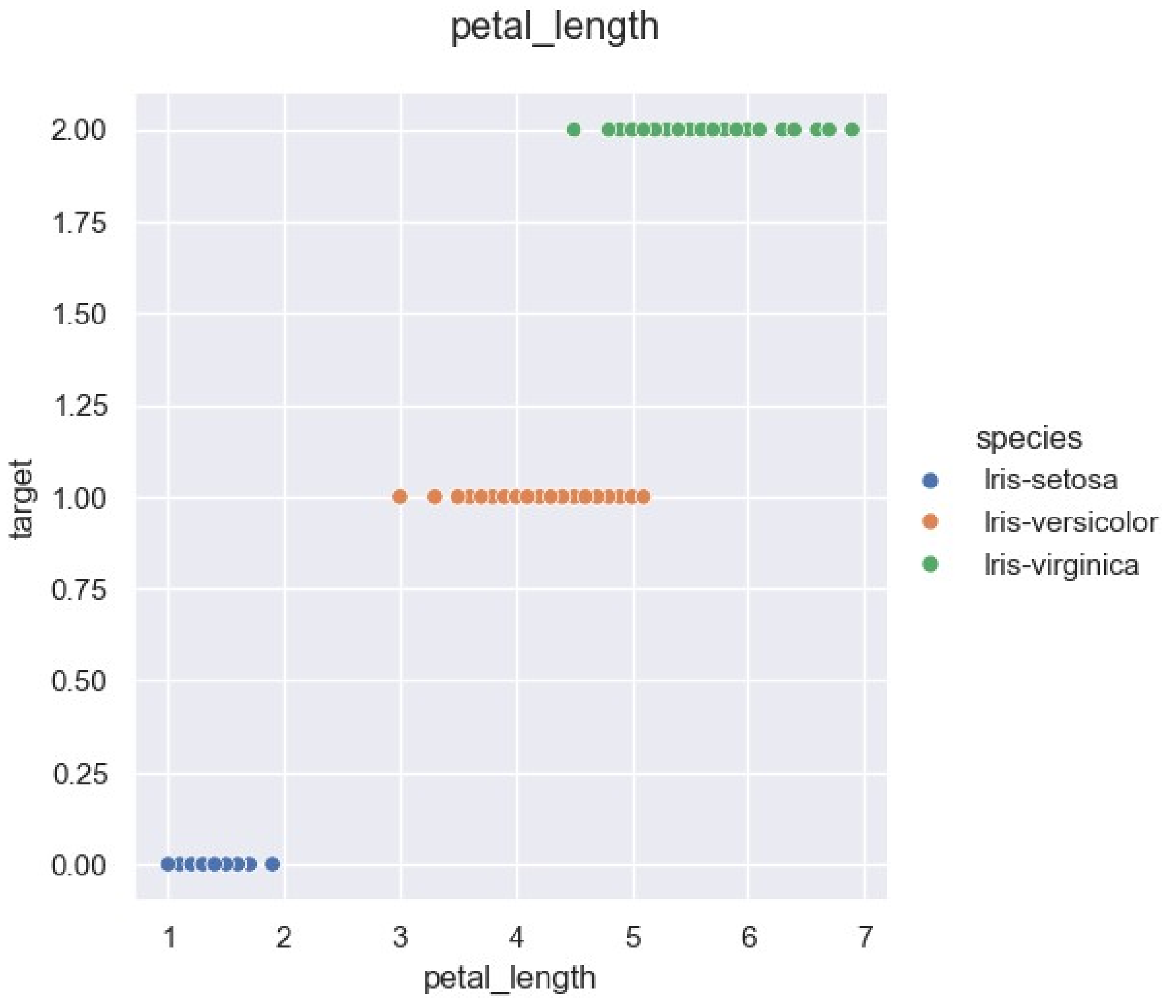
```
# df["sepal_length"].hist()
# plt.suptitle("sepal_length")
# plt.show()
col = "sepal_length"
sns.relplot(x=col, y="target", hue=data["species"], data=df )
plt.suptitle(col, y=1.05)
plt.show()
```



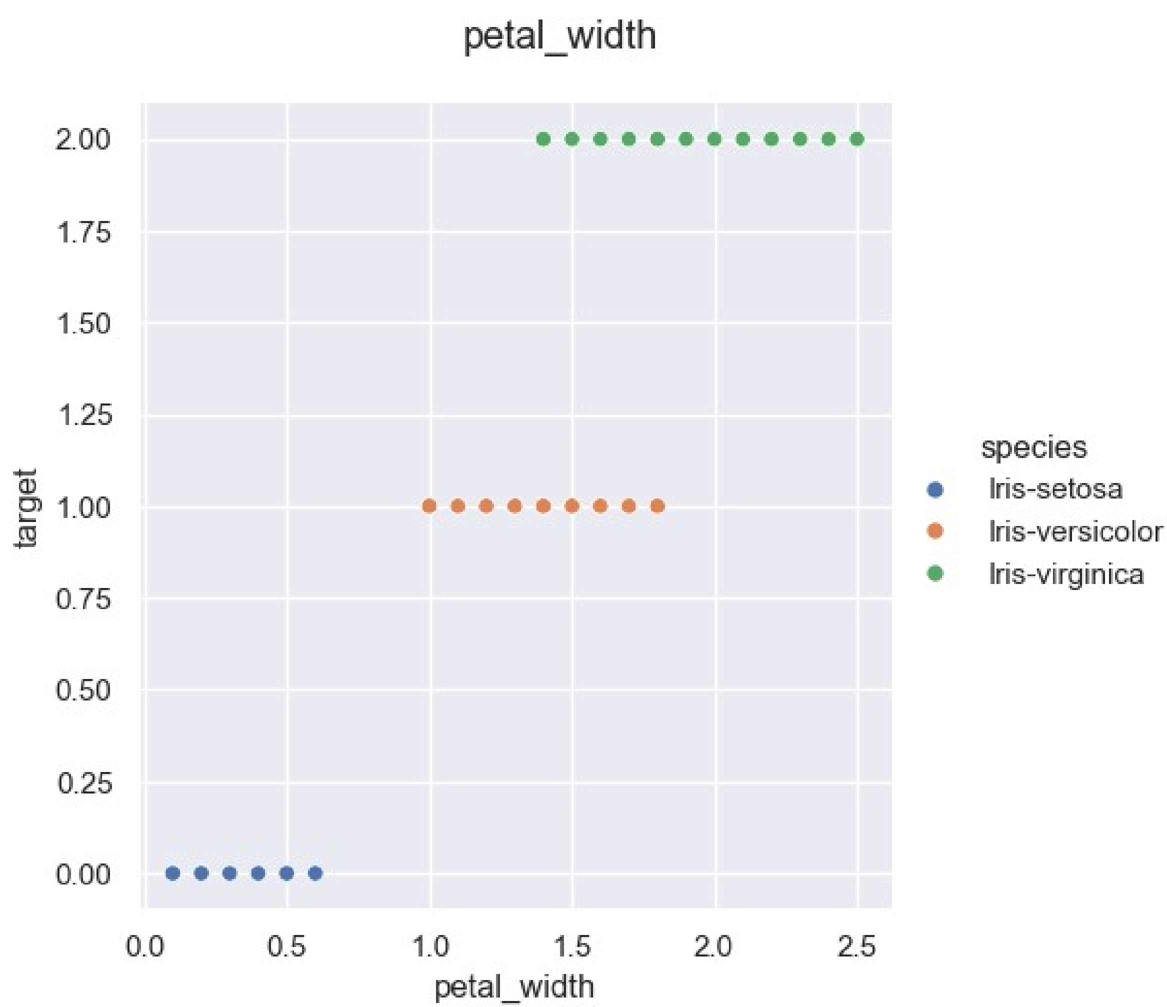

```
col = "sepal_width"
sns.relplot(x=col, y="target", hue=data["species"], data=df )
plt.suptitle(col, y=1.05)
plt.show()
```




```
col = "petal_length"
sns.relplot(x=col, y="target", hue=data["species"], data=df )
plt.suptitle(col, y=1.05)
plt.show()
```



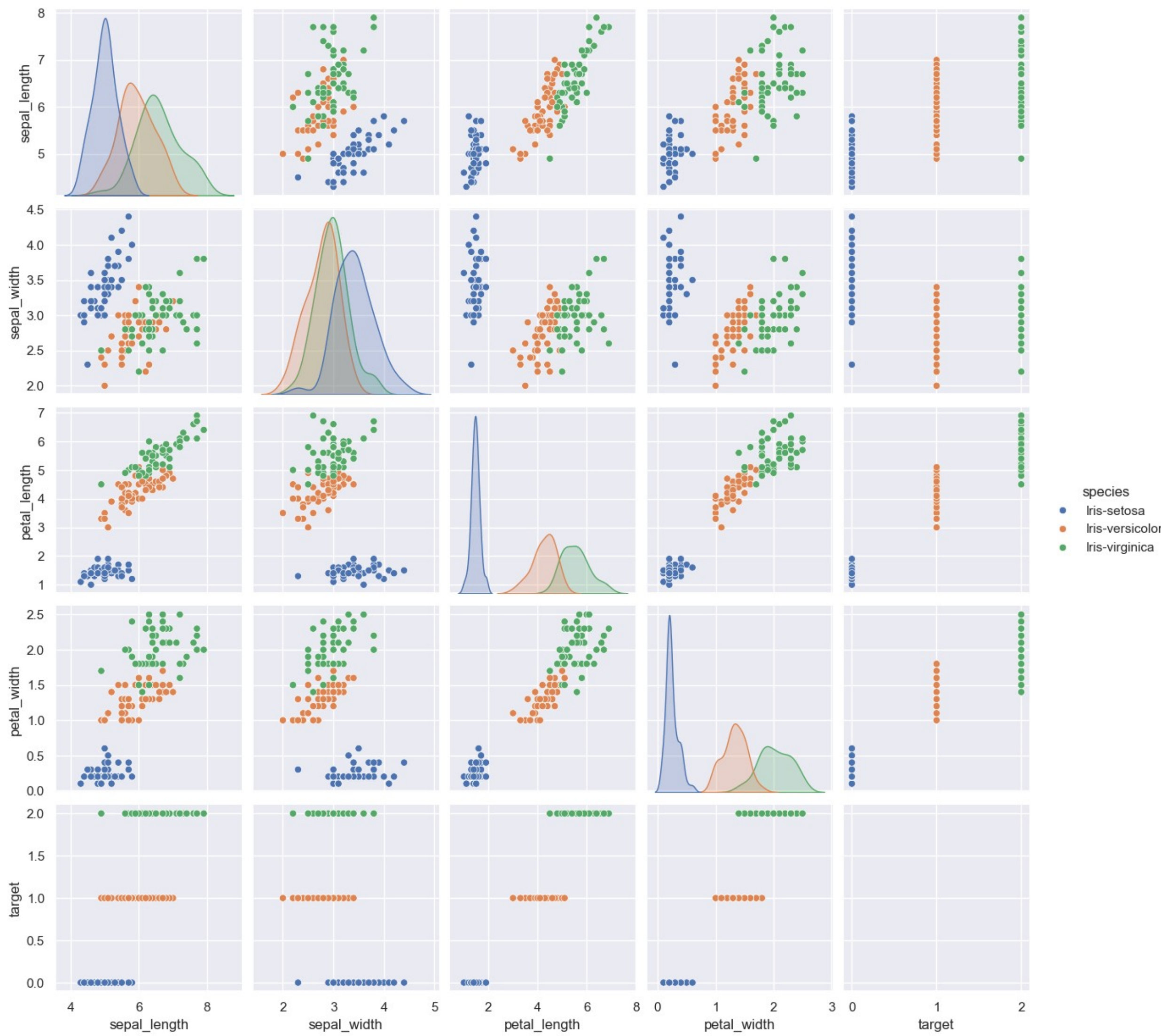

```
col = "petal_width"
sns.relplot(x=col, y="target", hue=data["species"], data=df )
plt.suptitle(col, y=1.05)
plt.show()
```



EDA (Pair Plots):

```
df["species"] = data["species"]
sns.pairplot(df, hue="species")

# sns.pairplot(df)
```

```
<seaborn.axisgrid.PairGrid at 0x2965c1650>
```



Train test split

You always want to evaluate your final model on a test set that has not been used at all in the training process. So we split off a test set here

(Note: When using cross-validation we could technically use the same data but its best practice it used separate data for testing)

```
from sklearn.model_selection import train_test_split

df_train, df_test = train_test_split(df, test_size=0.25)

df_train.shape
```

```
(112, 6)
```

```
df_test.shape
```

```
(38, 6)
```

Preparing our data for modeling

This involves splitting the data back into plain NumPy arrays

```
X_train = df_train.drop(columns=[ "species", "target"]).to_numpy() # so we drop these two columns and converts to the numpy array for model to train on
y_train = df_train["target"].to_numpy()
```

```
X_train.shape
```

```
(112, 4)
```

Modeling - What is our baseline?

What is the simplest model we can think of?

In this case, if our baseline model is just randomly guesssing the species of flower, or guesssing a single species for every data point, we would expect to have a model accuracy of 0.33 or 33%, since we have 3 different classes which are evenly balanced (50 data points each).

So our models should atleast beat 33% accuracy.

Modeling - Simple Manual Model

Let's manually look at our data and decide some cutoff points for classification.

```
data["species"].unique()
```

```
array(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'], dtype=object)
```

```
def single_feature_prediction(petal_length):
    """Predicts the Iris species given the petal length."""
    if petal_length < 2.5:
        return 0
    elif petal_length < 4.8:
        return 1
    else:
        return 2
```

```
# This is kindof very basic view of what decision trees ml model do in
the backend bunch of if else statements

df_train.columns

Index(['sepal_length', 'sepal_width', 'petal_length', 'petal_width',
'target',
       'species'],
      dtype='object')

X_train[:,2] # ':' is to select all the rows 2 is the 'petal_length'
column

array([4.5, 6.4, 5.9, 1.4, 1.4, 4.7, 1.6, 5.3, 4.9, 5.1, 1.7, 6.3,
1.5,
       5.2, 3.5, 1.4, 4.2, 1.5, 3.9, 6. , 1.3, 4.2, 4. , 5.1, 1.6,
1.4,
       1.3, 3.6, 1.4, 5. , 1.5, 5.5, 4.6, 1.6, 4.5, 4.9, 1.6, 5. ,
4.5,
       1.5, 3.3, 1.2, 1.4, 1.6, 1.4, 1.2, 1.7, 6. , 1.9, 4. , 4.4,
3.8,
       5.7, 3.5, 3.7, 4.3, 4.5, 4.4, 6.1, 1.5, 1.5, 5.6, 3.3, 4.1,
6.1,
       1.3, 4.5, 1.5, 4. , 1.4, 1.3, 5.6, 5.7, 3. , 6.9, 1.5, 6.6,
5.1,
       4.7, 6.7, 1.4, 1.5, 1.5, 4.8, 5.8, 4.5, 4.8, 5.6, 4. , 4.2,
5.3,
       1.4, 1.5, 5.9, 4.3, 5.1, 4.1, 4.7, 1.3, 1.1, 5. , 3.9, 5.6,
4.9,
       5.6, 5.1, 5.8, 4.4, 1.3, 5.7, 5.6, 5. ])

manual_y_predictions = np.array([single_feature_prediction(val) for
val in X_train[:,2] ])

manual_y_predictions == y_train  # this basically is gonna give true
whenever the prediction is correct else false

array([ True,  True,  True,  True,  True,  True,  True,  True,  True,
       False,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True, False, False,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True, False,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True, False,  True,  True, False,  True,  True,  True,  True,
        True,  True,  True,  True])
```

```python
manual_model_accuracy = float(np.mean(manual_y_predictions ==
y_train)) # this way we can find accuracy all true values as 1 and
false as 0 and takes it average

model_accuracies=[]
model_accuracies.append([manual_model_accuracy,"Manual"])

print(f"Manual model accuracy {manual_model_accuracy * 100:.2f}% ")
```

```
Manual model accuracy 94.64%
```

Modeling - Logistic Regression

```python
from sklearn.linear_model import LogisticRegression
```

Using a validation set to evaluate our model

This is different from the original test data which we split we will use that later to test all our models here we split a portion of the training dataset

```python
Xt, Xv, yt, yv = train_test_split(X_train, y_train, test_size=0.25)
# Xt is X_train and Xv is X_validation

Xt.shape
```

```
(84, 4)
```

```python
Xv.shape
```

```
(28, 4)
```

```python
model = LogisticRegression()

# model.fit(X_train, y_train)
model.fit(Xt, yt)
```

```
LogisticRegression()
```

```python
y_pred = model.predict(Xv)
np.mean(y_pred == yv)
```

```
1.0
```

```python
model.score(Xv,yv)
# both this and above doing the same thing to show that how it
calculates the score
```

```
1.0
```

```python
# model.score(X_train, y_train) # This right here is wrong as you never wanna evaluate your model on the same data that was used for training
```

Using Cross-Validation to evaluate our model

```python
from sklearn.model_selection import cross_val_score, cross_val_predict

model = LogisticRegression(max_iter  = 200)

accuracies = cross_val_score(estimator = model , X = X_train, y = y_train, cv = 5, scoring = "accuracy") # you dont really have to do estimator = X = y =
# as those 3 its already expecting as the first 3 values

accuracies
```

```
array([0.95652174, 0.91304348, 1.        , 0.95454545, 0.95454545])
```

```python
# this gives the scores from all those 5 splits where 1/5 of data is tested and 4/5 data is used as training

np.mean(accuracies)
```

```
0.9557312252964426
```

Where are we misclassifying points?

```python
y_pred = cross_val_predict(model, X_train, y_train, cv = 5)

predicted_correctly_mask = y_pred == y_train # basically like above y_pred == y_train will give us boolean array of where data was right and wrong

# this is using cross validation so basically training on 4/5 and testing on 1/5 data and each of this y_pred the value is when it was not tested on that

# like example it will for first 1/5 data check on portion when other 4/5 was used to train and so on

# so basically it meshes together the different predictions for different parts

X_train[predicted_correctly_mask] # see here we see the data points where predictions were correct
X_train[~predicted_correctly_mask] # see here we see the data points where predictions were incorrect
```

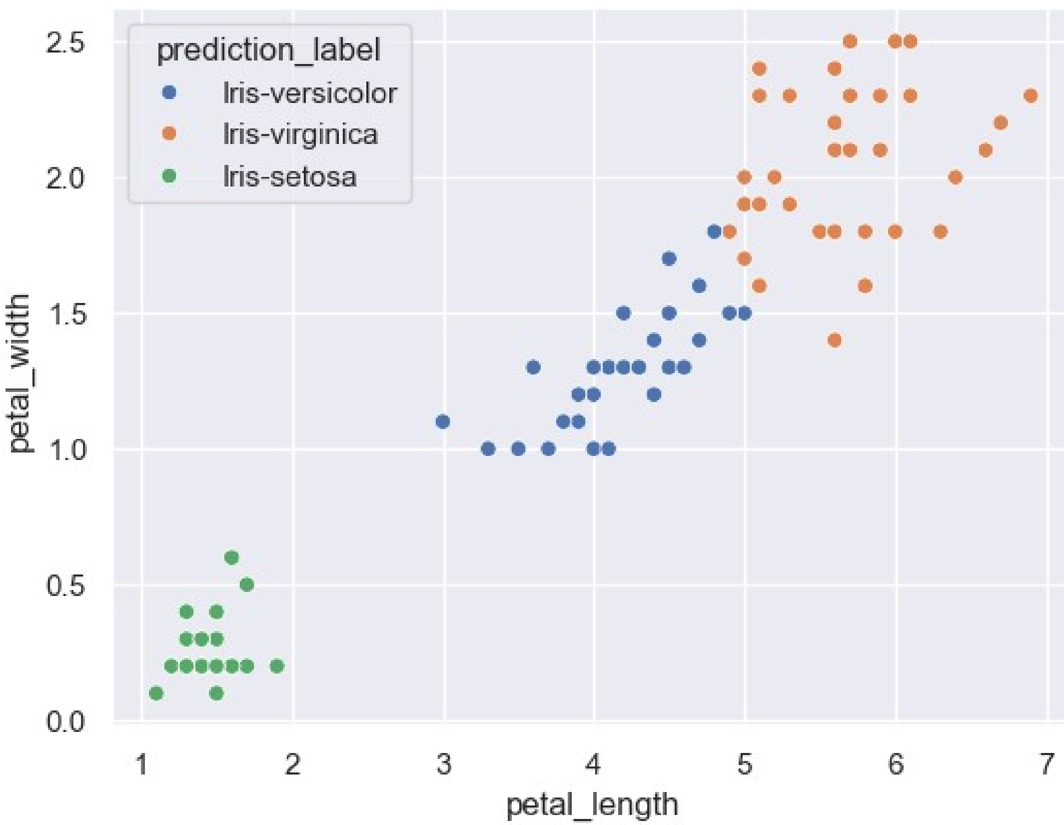
```
array([[6. , 2.7, 5.1, 1.6],
       [6. , 2.2, 5. , 1.5],
       [4.9, 2.5, 4.5, 1.7],
       [6. , 3. , 4.8, 1.8],
       [6.7, 3. , 5. , 1.7]])
```

```python
df_predictions = df_train.copy()

df_predictions["correct_prediction"] = predicted_correctly_mask

df_predictions["prediction"] = y_pred

df_predictions["prediction_label"] =
df_predictions["prediction"].map({0:"Iris-setosa", 1:"Iris-
versicolor", 2:"Iris-virginica"})

df_predictions.head()
```

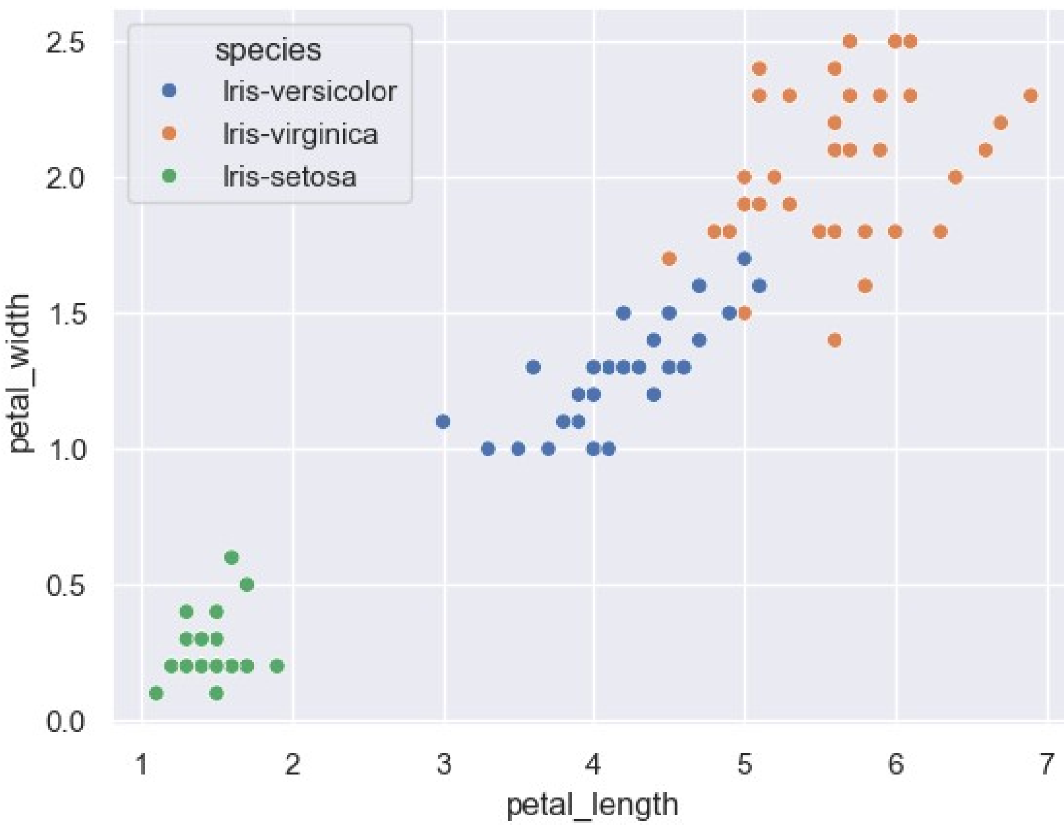
```
     sepal_length  sepal_width  petal_length  petal_width  target  \
55            5.7          2.8           4.5          1.3       1
131           7.9          3.8           6.4          2.0       2
143           6.8          3.2           5.9          2.3       2
49            5.0          3.3           1.4          0.2       0
17            5.1          3.5           1.4          0.3       0

             species  correct_prediction  prediction prediction_label

55   Iris-versicolor                True           1  Iris-versicolor

131   Iris-virginica                True           2   Iris-virginica

143   Iris-virginica                True           2   Iris-virginica

49       Iris-setosa                True           0      Iris-setosa

17       Iris-setosa                True           0      Iris-setosa
```

```python
sns.scatterplot(x="petal_length", y="petal_width",
hue="prediction_label", data=df_predictions)
```

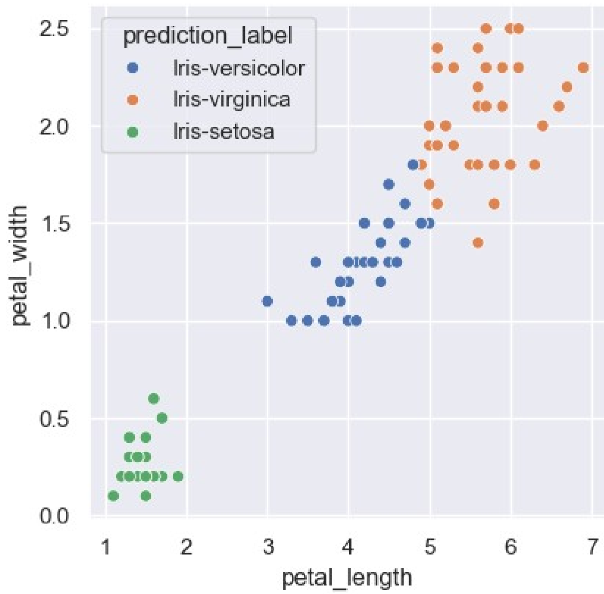
```
<Axes: xlabel='petal_length', ylabel='petal_width'>
```

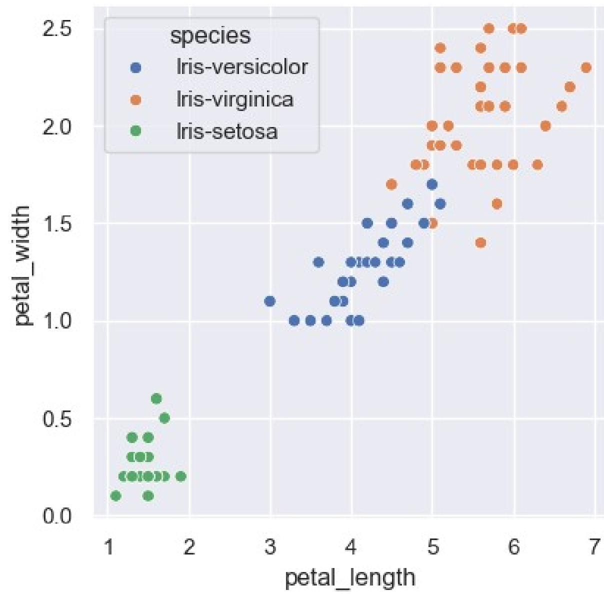


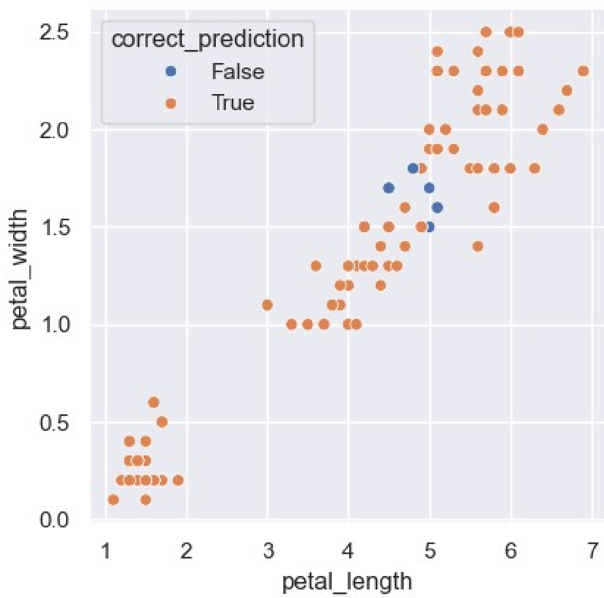

```
sns.scatterplot(x="petal_length", y="petal_width", hue="species",
data=df_predictions)

<Axes: xlabel='petal_length', ylabel='petal_width'>
```




```
def plot_incorrect_predictions(df_predictions, x_axis_feature,
y_axis_feature):
    fig, axs = plt.subplots(2, 2, figsize=(10,10))
    axs = axs.flatten()
    sns.scatterplot(x=x_axis_feature, y=y_axis_feature,
hue="prediction_label", data=df_predictions, ax=axs[0])
    sns.scatterplot(x=x_axis_feature, y=y_axis_feature, hue="species",
data=df_predictions, ax=axs[1])
    sns.scatterplot(x=x_axis_feature, y=y_axis_feature,
hue="correct_prediction", data=df_predictions, ax=axs[2])
    axs[3].set_visible(False)

    plt.show()

plot_incorrect_predictions(df_predictions, "petal_length",
"petal_width")
```







Model Tuning

What is model tuning?

Model tuning is trying to determine the parameters of your model (these are also known as "hyperparameters") that maximize the model performance.

```
for reg_para in (0.1,0.2,0.3,0.9,1,1.3,1.9,2,5):
    print(reg_para)
```

```
    # model = LogisticRegression(max_iter=200, C =1)

    model = LogisticRegression(max_iter=200, C =reg_para)

    # less C is more restricted model do does not go after patterns
too much and more C is less restrictive going towards overfitting
    # if we don't want our model to like memorize some data patterns
too much put less C else put more C

    # If your model is overfitting with high C, try lowering it. If
it's underfitting, increasing C might help.

    accuracies = cross_val_score(model, X_train, y_train, cv = 5,
scoring = "accuracy")
    print(f"Accuracy: {np.mean(accuracies) * 100:.2f}%")
    # change and see at which C the accuracy is the best

# we see which is best then lets say 1 is best then you could start
from one and add higher or end at 1 and go lower

# we did for bunch of values in a for look we can do one value as well
we can also use gridsearchcv to do this
```

```
0.1
Accuracy: 93.75%
0.2
Accuracy: 94.66%
0.3
Accuracy: 95.57%
0.9
Accuracy: 95.57%
1
Accuracy: 95.57%
1.3
Accuracy: 95.57%
1.9
Accuracy: 96.48%
2
Accuracy: 96.48%
5
Accuracy: 97.39%
```

Final Model

```
model = LogisticRegression(max_iter=200, C=1.9) # choose whichever you
think is best as use that as the final model here
```

How well does our model do on the Test Set?

```
X_test = df_test.drop(columns=[ "species", "target"]).to_numpy() # so we drop these two columns and converts to the numpy array for model to train on
y_test = df_test["target"].to_numpy()

X_test.shape
```

```
(38, 4)
```

```
 y_test
```

```
array([2, 0, 1, 0, 2, 0, 2, 2, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 2,
       1, 2, 1, 2, 0, 0, 2, 2, 0, 2, 1, 0, 2, 2, 2, 2], dtype=int8)
```

Train our final model using our full Training Dataset

```
model.fit(X_train, y_train)
# print(model.get_params())
```

```
LogisticRegression(C=1.9, max_iter=200)
```

```
y_test_pred = model.predict(X_test)

test_set_correctly_classified = y_test_pred == y_test
test_set_accuracy = np.mean(test_set_correctly_classified)

# test_set_accuracy = model.score(X_test,y_test) # this line is also doing the same thing which we did manually

# we are doing manually as we want to generate the graph so it is giving use predictions which we can put in the dataframe and generated graphs

model_accuracies.append([test_set_accuracy,"Logistic Regression"])
print(f"Test set accuracy: {test_set_accuracy * 100:.2f}%")
# if our training dataset was higher score than the test dataset meaning we are like overfitting
```

```
Test set accuracy: 97.37%
```

Final model with Cross validation

```
final_model = LogisticRegression(max_iter=200, C=1.9)
cv_accuracies = cross_val_score(final_model, X_train, y_train, cv=5,
scoring="accuracy")

print(f"Cross-validation Accuracy: {np.mean(cv_accuracies) * 100:.2f}
%")
```

```
Cross-validation Accuracy: 96.48%
```

```
test_set_correctly_classified
```

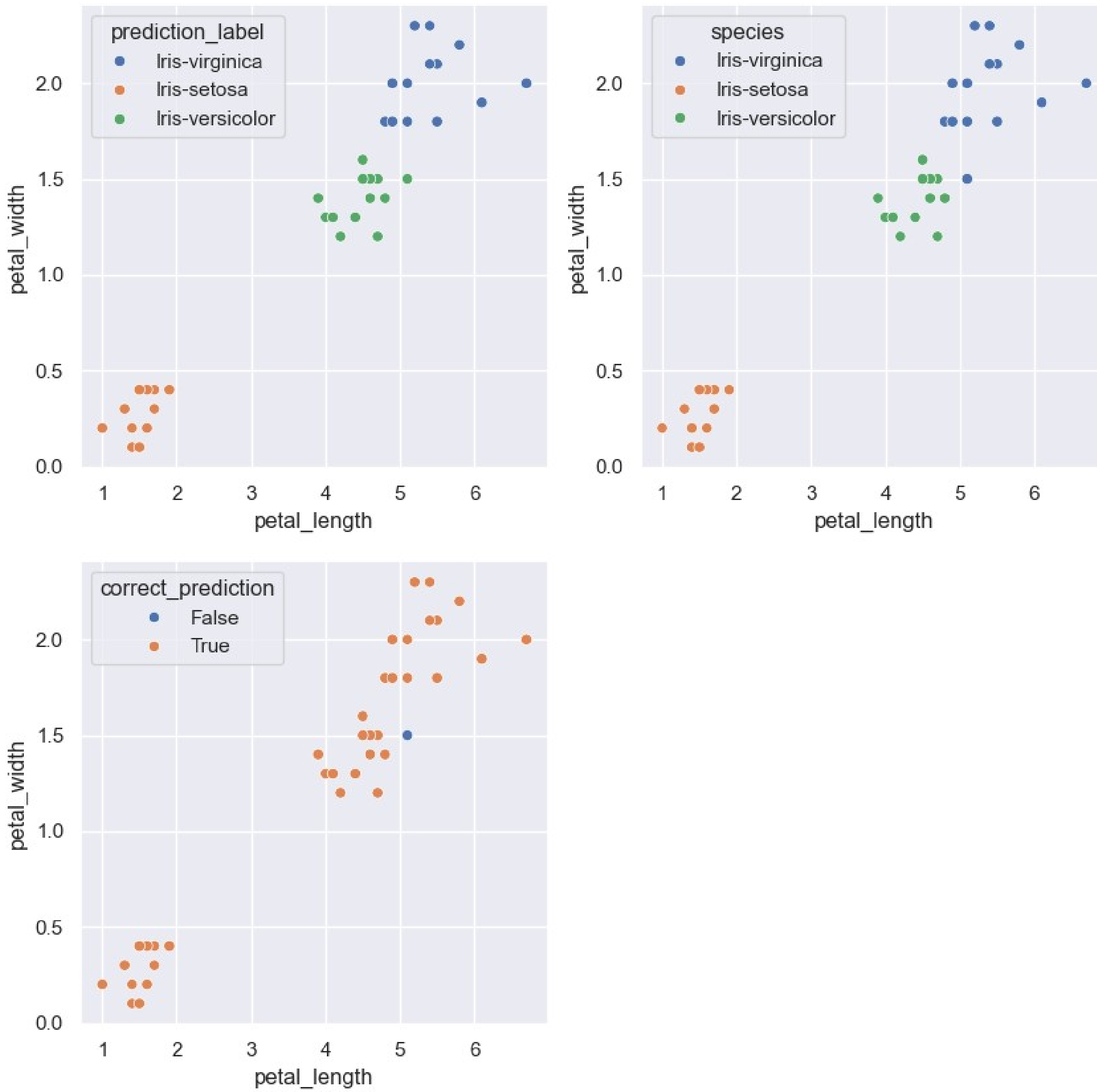
```
array([ True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True, False,
        True,  True])
```

```
df_predictions_test = df_test.copy()
df_predictions_test["correct_prediction"] =
test_set_correctly_classified
df_predictions_test["prediction"] = y_test_pred
df_predictions_test["prediction_label"] =
df_predictions_test["prediction"].map({0:"Iris-setosa", 1:"Iris-
versicolor", 2:"Iris-virginica"})
df_predictions_test.head()
```

```
     sepal_length  sepal_width  petal_length  petal_width  target  \
121           5.6          2.8           4.9          2.0       2
41            4.5          2.3           1.3          0.3       0
76            6.8          2.8           4.8          1.4       1
25            5.0          3.0           1.6          0.2       0
130           7.4          2.8           6.1          1.9       2

             species  correct_prediction  prediction prediction_label

121   Iris-virginica                True           2   Iris-virginica

41       Iris-setosa                True           0       Iris-setosa

76   Iris-versicolor                True           1  Iris-versicolor

25       Iris-setosa                True           0       Iris-setosa

130   Iris-virginica                True           2   Iris-virginica
```

```
# plot_incorrect_predictions(df_predictions_test,
x_axis_feature="petal_length", y_axis_feature="petal_width") # we
created this function earlier
```

```python
plot_incorrect_predictions(df_predictions_test, "petal_length",
"petal_width")
```



Modeling - Random Forest Regression

```python
from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier()
# we use classifier instead of regressor as we need to find a
categorical data and not finding a value like we did in house
prediction
```

```
Xt, Xv, yt, yv = train_test_split(X_train, y_train, test_size=0.25)
# we split the training dataset as we will test on testing dataset
towards the end till then we use this split to test the dataset


model.fit(Xt,yt)
```

```
RandomForestClassifier()
```

```
y_pred = model.predict(Xv)
np.mean(y_pred == yv)
```

```
1.0
```

```
model.score(Xv, yv)
```

```
1.0
```

Scaling our data

Basically we find out that in this case scaling the data did not really imporve the result just usually we should scale the data

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

Xts = scaler.fit_transform(Xt) # this first fits so scaler knows how
to transform the data then scales Xt
Xvs = scaler.transform(Xv) # this is scaling Xv for basically the
texting data

model = RandomForestClassifier()

model.fit(Xts, yt)
model.score(Xvs, yv)
# about the same accuracy as we got without scaling the data
```

```
1.0
```

Using Cross-Validation to evaluate our Model

```
from sklearn.model_selection import cross_val_score, cross_val_predict
model = RandomForestClassifier()
accuracies = cross_val_score(estimator = model , X = X_train, y =
y_train, cv = 5, scoring = "accuracy")
```

```
# here we just put the training data and it splits it here cv = 5 so 1/5 to test 4/5 to train and it does on all different data using each portion to test

accuracies
```

```
array([0.95652174, 0.91304348, 1.        , 0.95454545, 0.90909091])
```

```
np.mean(accuracies)
```

```
0.9466403162055336
```

What are the misclassifying points in the Random Forest model?

```
y_pred = cross_val_predict(model, X_train, y_train, cv = 5) # its already expecting those values as initial so don't really need to specify
y_pred
```

```
array([1, 2, 2, 0, 0, 1, 0, 2, 2, 2, 0, 2, 0, 2, 1, 0, 1, 0, 1, 2, 0, 1,
       1, 2, 0, 0, 0, 1, 0, 1, 0, 2, 1, 0, 1, 1, 0, 2, 1, 0, 1, 0, 0, 0,
       0, 0, 0, 2, 0, 1, 1, 1, 2, 1, 1, 1, 1, 1, 2, 0, 0, 2, 1, 1, 2, 0,
       1, 0, 1, 0, 0, 2, 2, 1, 2, 0, 2, 2, 1, 2, 0, 0, 0, 2, 1, 1, 2, 2,
       1, 1, 2, 0, 0, 2, 1, 2, 1, 1, 0, 0, 2, 1, 1, 1, 2, 2, 2, 1, 0, 2,
       2, 2], dtype=int8)
```

```
predicted_correctly_mask = y_pred==y_train
predicted_correctly_mask
```

```
array([ True,  True,  True,  True,  True,  True,  True,  True,  True,
       False,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True, False,  True,  True,  True,  True, False,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True, False, False,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True, False,  True, False,  True,  True,  True,  True,  True,
        True,  True,  True,  True])
```

```
X_train[predicted_correctly_mask][:5] # here displaying first 5 rows
of dataset where its doing correct prediction
```

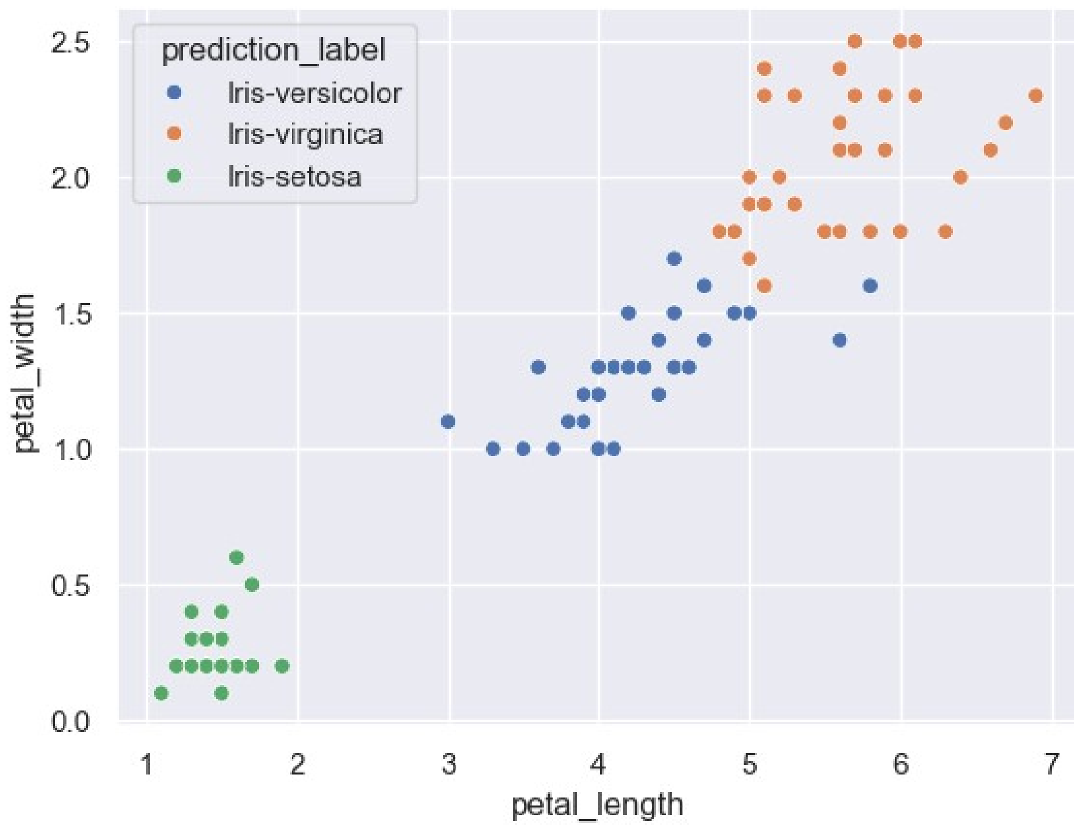
```
array([[5.7, 2.8, 4.5, 1.3],
       [7.9, 3.8, 6.4, 2. ],
       [6.8, 3.2, 5.9, 2.3],
       [5. , 3.3, 1.4, 0.2],
       [5.1, 3.5, 1.4, 0.3]])
```

```
df_predictions = df_train.copy() # so we don't make changes to the
df_train dataset

df_predictions['correct_prediction'] = predicted_correctly_mask

df_predictions['prediction'] = y_pred

df_predictions['prediction_label'] =
df_predictions['prediction'].map({0:"Iris-setosa", 1:"Iris-
versicolor", 2:"Iris-virginica"})
df_predictions.head()
```

```
     sepal_length  sepal_width  petal_length  petal_width  target  \
55            5.7          2.8           4.5          1.3       1
131           7.9          3.8           6.4          2.0       2
143           6.8          3.2           5.9          2.3       2
49            5.0          3.3           1.4          0.2       0
17            5.1          3.5           1.4          0.3       0

             species  correct_prediction  prediction prediction_label

55   Iris-versicolor                True           1  Iris-versicolor

131   Iris-virginica                True           2   Iris-virginica

143   Iris-virginica                True           2   Iris-virginica

49       Iris-setosa                True           0      Iris-setosa

17       Iris-setosa                True           0      Iris-setosa
```

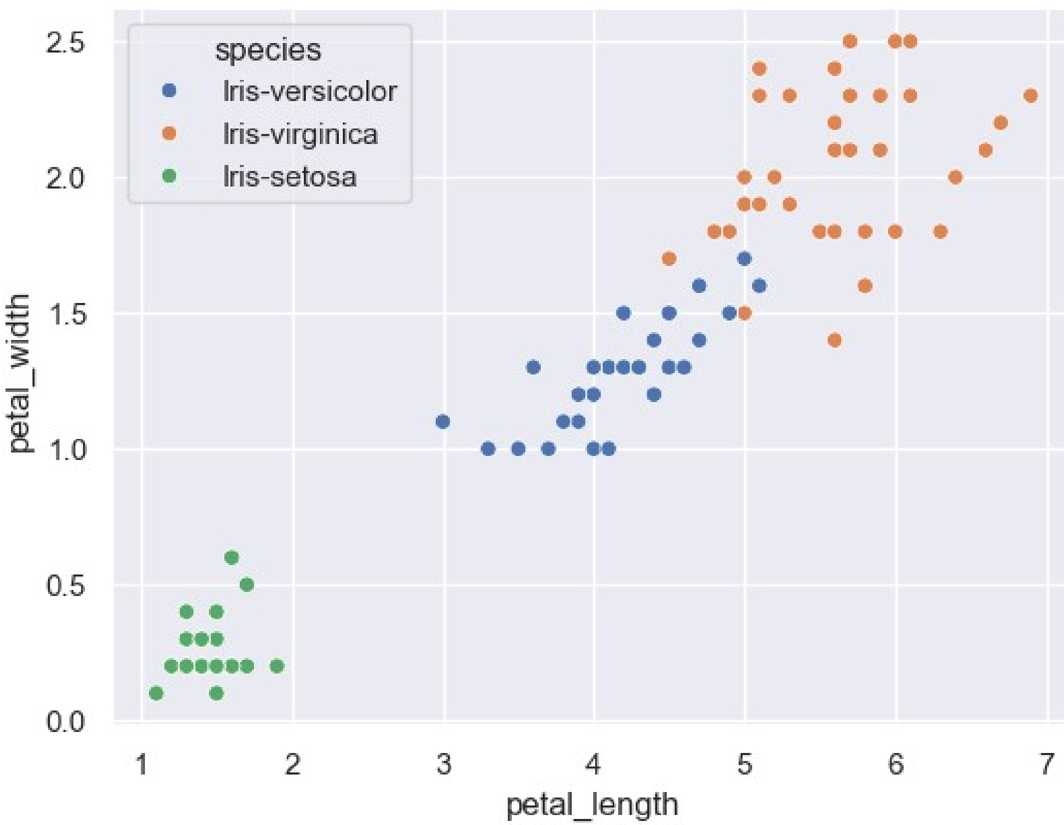
```
sns.scatterplot(x="petal_length", y="petal_width",
hue="prediction_label", data=df_predictions)
```

```
<Axes: xlabel='petal_length', ylabel='petal_width'>
```

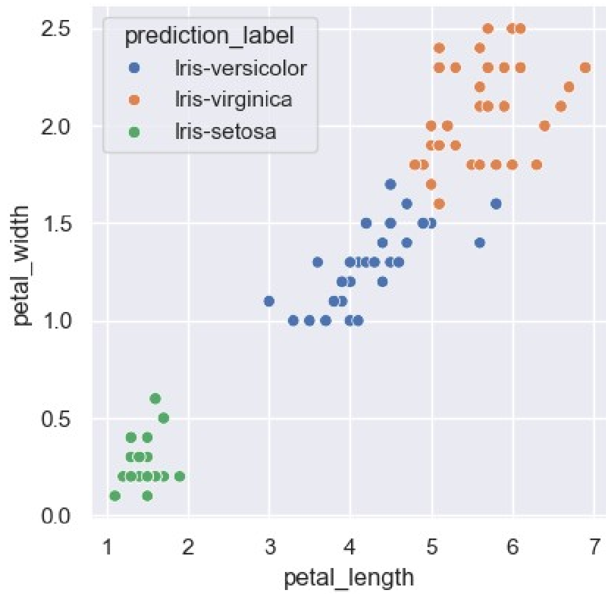


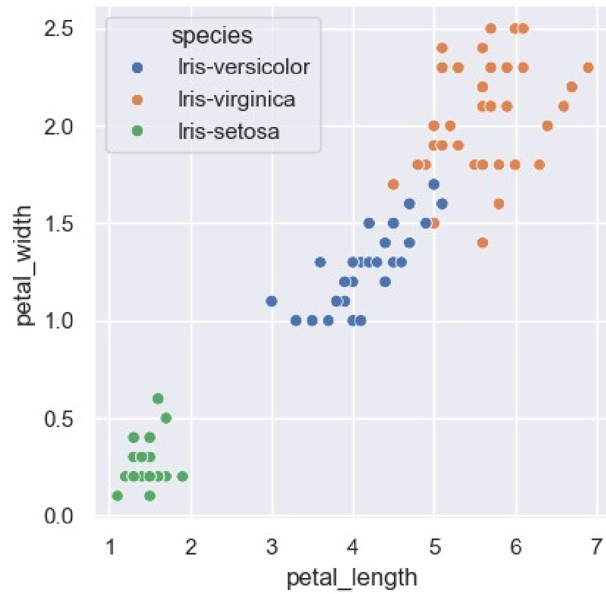

```
sns.scatterplot(x="petal_length", y="petal_width", hue="species",
data=df_predictions)
```

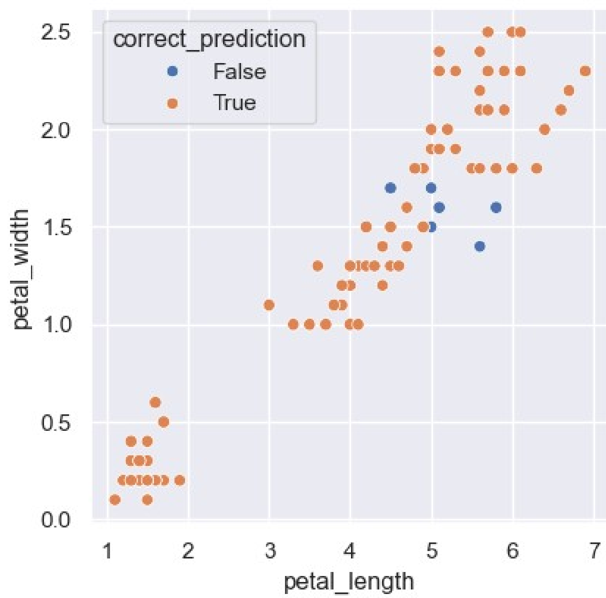
```
<Axes: xlabel='petal_length', ylabel='petal_width'>
```




```
plot_incorrect_predictions(df_predictions, "petal_length",
"petal_width")
# These plots are good for visualizing the data
```







Hyperparameter tuning

```
from sklearn.model_selection import GridSearchCV

forest = RandomForestClassifier() # this is to get a fresh untrained
model again

param_grid = {
    "n_estimators": [40, 50, 100],  # No need for very high values
    "min_samples_split": [2, 3],  # Small dataset, so minor tuning
    "max_depth": [2, 3, 4]  # Best for avoiding overfitting
}
```

```
# putting too much values will case it slower as it checks each
combination

grid_search = GridSearchCV(forest, param_grid, cv= 5, scoring =
"accuracy", return_train_score=True)
# we did neg-mse as the bigger the values the better mse will be the
lower the better

grid_search.fit(Xts, yt)

GridSearchCV(cv=5, estimator=RandomForestClassifier(),
             param_grid={'max_depth': [2, 3, 4], 'min_samples_split':
[2, 3],
                         'n_estimators': [40, 50, 100]},
             return_train_score=True, scoring='accuracy')

best_forest = grid_search.best_estimator_ # we save the best
parameters
print(best_forest)
# the things its not showing meaning best values for them is the
default

RandomForestClassifier(max_depth=2, min_samples_split=3,
n_estimators=50)

print(f"Accuracy: {best_forest.score(Xvs, yv) * 100:.2f}%")

Accuracy: 100.00%
```

In this case hyperparameter tuning did not really improve the accuracy but it still good practice to test

Final Model

```
model = RandomForestClassifier(n_estimators=50, max_depth=2) # Using
the best values obtained from GridSearchCV rest values are the
defaults
model.get_params()

{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': 2,
 'max_features': 'sqrt',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_samples_leaf': 1,
```

```
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'monotonic_cst': None,
 'n_estimators': 50,
 'n_jobs': None,
 'oob_score': False,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}
```

Train our final model using our full Training Dataset

```
model.fit(X_train, y_train)
```

```
RandomForestClassifier(max_depth=2, n_estimators=50)
```

```
y_test_pred = model.predict(X_test)
```

```
test_set_correctly_classified = y_test_pred == y_test
test_set_accuracy = np.mean(test_set_correctly_classified)
print(f"Test set accuracy: {test_set_accuracy * 100:.2f}%")
```

```
Test set accuracy: 100.00%
```

Final model with Cross validation

```
final_model = RandomForestClassifier(n_estimators=50, max_depth=2)
cv_accuracies = cross_val_score(final_model, X_train, y_train, cv=5,
scoring="accuracy")

model_accuracies.append([np.mean(cv_accuracies),"Random Forest"])
print(f"Cross-validation Accuracy: {np.mean(cv_accuracies) * 100:.2f}
%")
```

```
Cross-validation Accuracy: 93.79%
```

```
test_set_correctly_classified
```

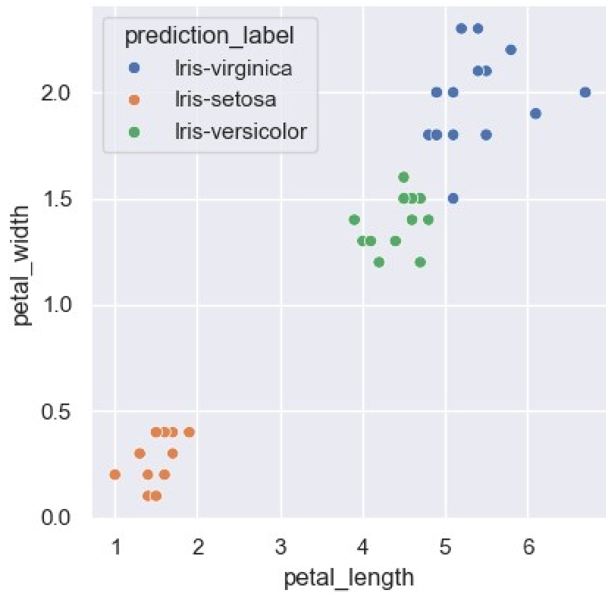
```
array([ True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True])
```

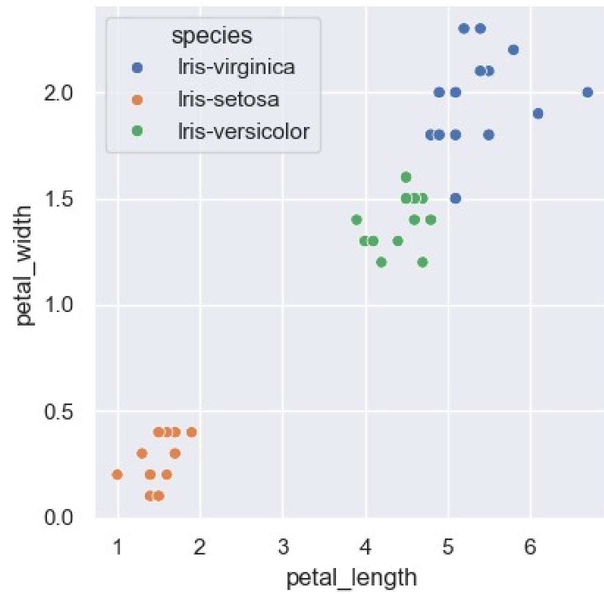
```
df_predictions_test = df_test.copy()
df_predictions_test["correct_prediction"] =
```

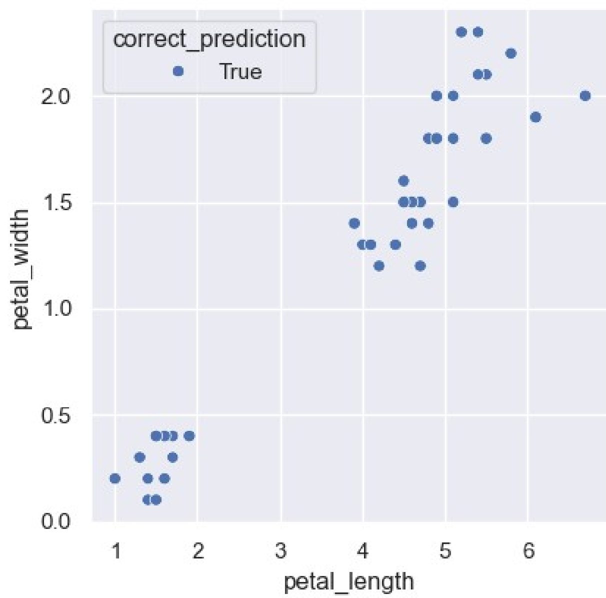
```
test_set_correctly_classified
df_predictions_test["prediction"] = y_test_pred
df_predictions_test["prediction_label"] =
df_predictions_test["prediction"].map({0:"Iris-setosa", 1:"Iris-
versicolor", 2:"Iris-virginica"})
df_predictions_test.head()
```

```
     sepal_length  sepal_width  petal_length  petal_width  target  \
121           5.6          2.8           4.9          2.0       2
41            4.5          2.3           1.3          0.3       0
76            6.8          2.8           4.8          1.4       1
25            5.0          3.0           1.6          0.2       0
130           7.4          2.8           6.1          1.9       2

             species  correct_prediction  prediction prediction_label

121   Iris-virginica                True           2   Iris-virginica

41       Iris-setosa                True           0      Iris-setosa

76   Iris-versicolor                True           1  Iris-versicolor

25       Iris-setosa                True           0      Iris-setosa

130   Iris-virginica                True           2   Iris-virginica
```

```
# plot_incorrect_predictions(df_predictions_test,
x_axis_feature="petal_length", y_axis_feature="petal_width") # we
created this function earlier
plot_incorrect_predictions(df_predictions_test, "petal_length",
"petal_width")
```







Scaling our Test and Train dataset for the models below

```
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Modeling - Support Vector Machine

```
from sklearn.svm import SVC

def check(para):
    svm_model = SVC(kernel='rbf', C=para, gamma='scale')
    svm_model.fit(Xv, yv)
    accuracy = svm_model.score(Xt,yt)
    print(f"SVM Model Accuracy (C={para:.1f}): {accuracy:.2f}")

# values = [i for i in frange(0.1, 2.1, 0.1)]
for i in np.arange(0.1, 2.1, 0.1): # as range does not work with
floating points either use this or make a list comprehension then
iterate on that list
    check(i)
```

```
SVM Model Accuracy (C=0.1): 0.29
SVM Model Accuracy (C=0.2): 0.42
SVM Model Accuracy (C=0.3): 0.89
SVM Model Accuracy (C=0.4): 0.89
SVM Model Accuracy (C=0.5): 0.90
SVM Model Accuracy (C=0.6): 0.90
SVM Model Accuracy (C=0.7): 0.90
SVM Model Accuracy (C=0.8): 0.90
SVM Model Accuracy (C=0.9): 0.90
SVM Model Accuracy (C=1.0): 0.90
SVM Model Accuracy (C=1.1): 0.90
SVM Model Accuracy (C=1.2): 0.90
SVM Model Accuracy (C=1.3): 0.90
SVM Model Accuracy (C=1.4): 0.90
SVM Model Accuracy (C=1.5): 0.90
SVM Model Accuracy (C=1.6): 0.90
SVM Model Accuracy (C=1.7): 0.90
SVM Model Accuracy (C=1.8): 0.90
SVM Model Accuracy (C=1.9): 0.90
SVM Model Accuracy (C=2.0): 0.90
```

Cross Validating the SVM model

```
def check2(para):
    svm_model = SVC(kernel='rbf',C=para) # gamma='scale' is default so
we don't need to specify that
    accuracy = cross_val_score(svm_model,X=Xt, y=yt, cv=5,
scoring="accuracy")
    print(f"SVM Model Accuracy (C={para:.1f}):
{np.mean(accuracy):.2f}")
```

```python
for i in np.arange(0.1,2.5,0.1):
    check2(i)
```

```
SVM Model Accuracy (C=0.1): 0.71
SVM Model Accuracy (C=0.2): 0.87
SVM Model Accuracy (C=0.3): 0.91
SVM Model Accuracy (C=0.4): 0.95
SVM Model Accuracy (C=0.5): 0.93
SVM Model Accuracy (C=0.6): 0.94
SVM Model Accuracy (C=0.7): 0.94
SVM Model Accuracy (C=0.8): 0.95
SVM Model Accuracy (C=0.9): 0.96
SVM Model Accuracy (C=1.0): 0.96
SVM Model Accuracy (C=1.1): 0.96
SVM Model Accuracy (C=1.2): 0.96
SVM Model Accuracy (C=1.3): 0.96
SVM Model Accuracy (C=1.4): 0.96
SVM Model Accuracy (C=1.5): 0.96
SVM Model Accuracy (C=1.6): 0.96
SVM Model Accuracy (C=1.7): 0.96
SVM Model Accuracy (C=1.8): 0.96
SVM Model Accuracy (C=1.9): 0.96
SVM Model Accuracy (C=2.0): 0.96
SVM Model Accuracy (C=2.1): 0.96
SVM Model Accuracy (C=2.2): 0.96
SVM Model Accuracy (C=2.3): 0.96
SVM Model Accuracy (C=2.4): 0.96
```

Final Model:

```python
svm_model = SVC(kernel='rbf', C=2.0) # Choosing the best C from the give options
svm_model.get_params()
```

```
{'C': 2.0,
 'break_ties': False,
 'cache_size': 200,
 'class_weight': None,
 'coef0': 0.0,
 'decision_function_shape': 'ovr',
 'degree': 3,
 'gamma': 'scale',
 'kernel': 'rbf',
 'max_iter': -1,
 'probability': False,
 'random_state': None,
 'shrinking': True,
```

```
 'tol': 0.001,
 'verbose': False}
```

Train our final model using our full Training Dataset

```
svm_model.fit(X_train, y_train)
y_pred = svm_model.predict(X_test)

test_set_correctly_classified = y_pred == y_test
test_set_accuracy = np.mean(test_set_correctly_classified)
print(f"Test set accuracy: {test_set_accuracy * 100:.2f}%")
```

```
Test set accuracy: 97.37%
```

Final model with Cross validation

```
svm_model = SVC(kernel='rbf', C=2.0)

accuracies = cross_val_score(svm_model, X_train, y_train, cv = 5,
scoring="accuracy")

model_accuracies.append([np.mean(cv_accuracies),"Support Vector
Machine"])
print(f"Cross-validation Accuracy: {np.mean(cv_accuracies) * 100:.2f}
%")
```

```
Cross-validation Accuracy: 93.79%
```

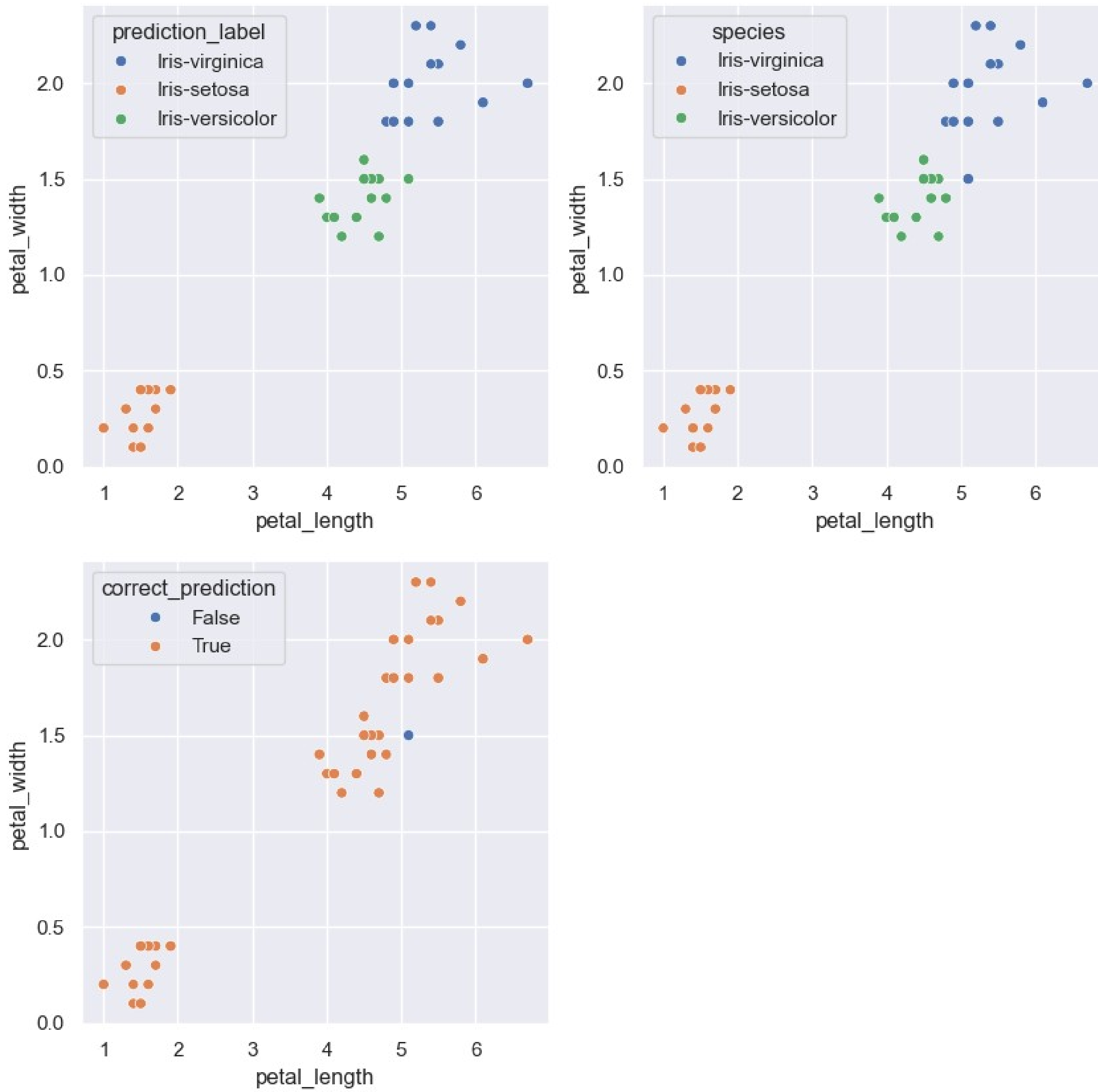
```
test_set_correctly_classified
```

```
array([ True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True, False,
        True,  True])
```

```
df_predictions_test = df_test.copy()
df_predictions_test['correct_prediction'] =
test_set_correctly_classified
df_predictions_test['prediction'] = y_pred
df_predictions_test['prediction_label'] =
df_predictions_test['prediction'].map({0:"Iris-setosa", 1:"Iris-
versicolor", 2:"Iris-virginica"})
df_predictions_test.head()
```

```
     sepal_length  sepal_width  petal_length  petal_width  target  \
121           5.6          2.8           4.9          2.0       2
41            4.5          2.3           1.3          0.3       0
76            6.8          2.8           4.8          1.4       1
25            5.0          3.0           1.6          0.2       0
130           7.4          2.8           6.1          1.9       2

             species  correct_prediction  prediction prediction_label

121   Iris-virginica                True           2   Iris-virginica

41       Iris-setosa                True           0      Iris-setosa

76   Iris-versicolor                True           1  Iris-versicolor

25       Iris-setosa                True           0      Iris-setosa

130   Iris-virginica                True           2   Iris-virginica
```

```
# plot_incorrect_predictions(df_predictions_test,
x_axis_feature="petal_length", y_axis_feature="petal_width") # we
created this function earlier
plot_incorrect_predictions(df_predictions_test, "petal_length",
"petal_width")
```



K - Nearest Neighbors

```python
from sklearn.neighbors import KNeighborsClassifier as knn

model_knn = knn(n_neighbors=5)
# model_knn.fit(Xv,yv)
model_knn.fit(Xts, yt)

# accuracy = model_knn.score(Xt,yt)
accuracy = model_knn.score(Xvs, yv)
accuracy
# we are getting better accuracy when scaling the dataset
```

```
0.9642857142857143
```

Hypertuning the model

```
values = [i for i in range(1,21)]

model = knn()

para_grid={
    'n_neighbors' : values,
    'weights' : ['uniform', 'distance'],
    'metric' : ['euclidean', 'manhattan']
}

grid_search = GridSearchCV(model, para_grid, cv=5, scoring='accuracy')
grid_search.fit(Xts, yt)
```

```
GridSearchCV(cv=5, estimator=KNeighborsClassifier(),
             param_grid={'metric': ['euclidean', 'manhattan'],
                         'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9,
10, 11, 12,
                                         13, 14, 15, 16, 17, 18, 19,
20],
                         'weights': ['uniform', 'distance']},
             scoring='accuracy')
```

```
grid_search.best_estimator_
```

```
KNeighborsClassifier(metric='manhattan', n_neighbors=9)
```

Final Model

```
best_knn = grid_search.best_estimator_
# print(best_knn)
best_knn.get_params()
```

```
{'algorithm': 'auto',
 'leaf_size': 30,
 'metric': 'manhattan',
 'metric_params': None,
 'n_jobs': None,
 'n_neighbors': 9,
 'p': 2,
 'weights': 'uniform'}
```

```
print(f"Accuracy: {best_knn.score(Xvs, yv) * 100:.2f}%")
```

```
Accuracy: 96.43%
```

Train our final model using our full Training Dataset

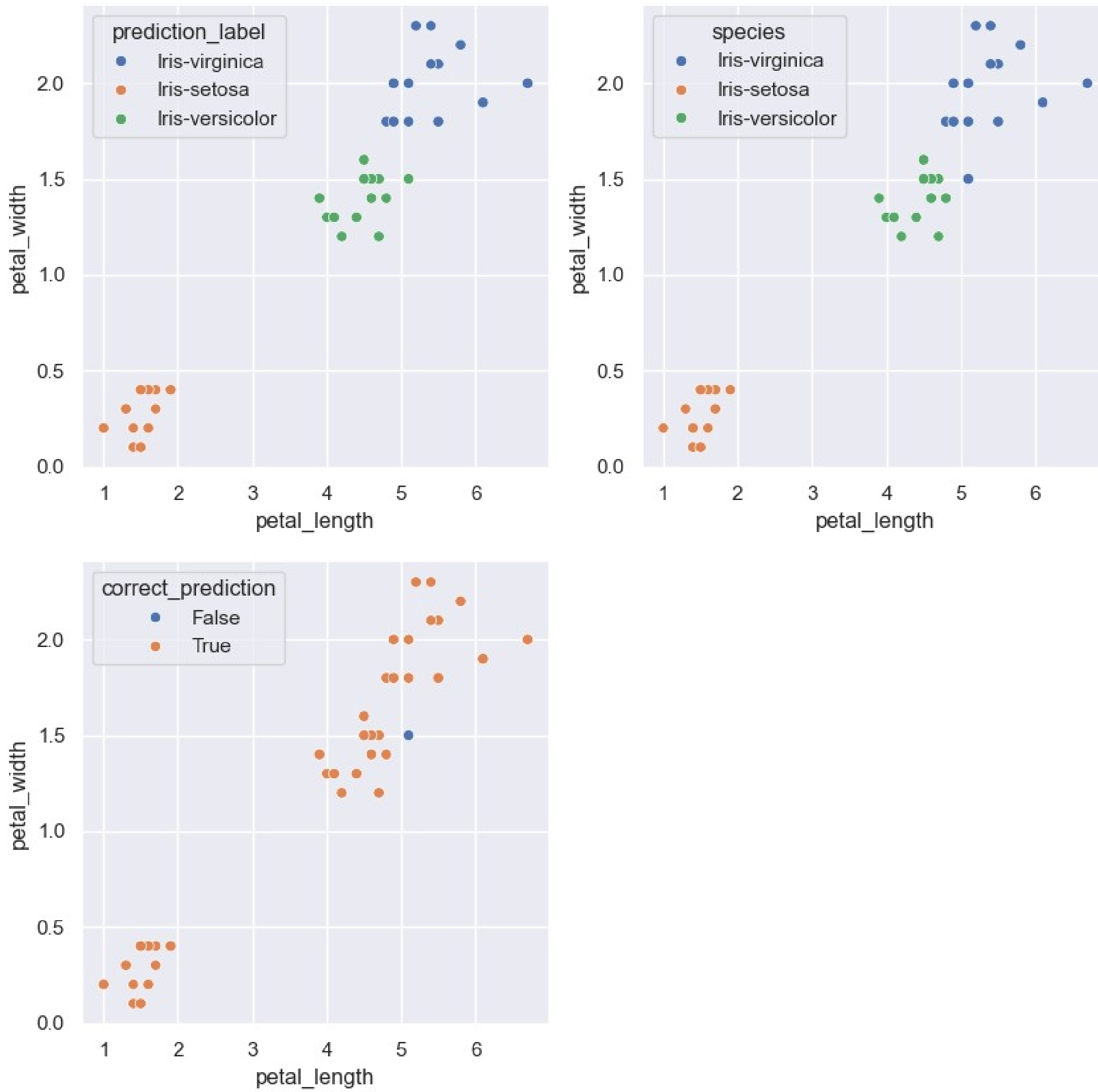
```
best_knn.fit(X_train, y_train)
y_pred = best_knn.predict(X_test)

test_set_correctly_classified = y_pred == y_test
test_set_accuracy = np.mean(test_set_correctly_classified)
print(f"Test set accuracy: {test_set_accuracy * 100:.2f}%")

Test set accuracy: 97.37%
```

Final model with Cross validation

```
best_knn = grid_search.best_estimator_

accuracies = cross_val_score(best_knn, X_train, y_train, cv = 5,
scoring="accuracy")

model_accuracies.append([np.mean(cv_accuracies),"K-Nearest
Neighbors"])
print(f"Cross-validation Accuracy: {np.mean(cv_accuracies) * 100:.2f}
%")

Cross-validation Accuracy: 93.79%

test_set_correctly_classified

array([ True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True, False,
        True,  True])

df_predictions_test = df_test.copy()
df_predictions_test['correct_prediction'] =
test_set_correctly_classified
df_predictions_test['prediction'] = y_pred
df_predictions_test['prediction_label'] =
df_predictions_test['prediction'].map({0:"Iris-setosa", 1:"Iris-
versicolor", 2:"Iris-virginica"})
df_predictions_test.head()
```

```
     sepal_length  sepal_width  petal_length  petal_width  target  \
121           5.6          2.8           4.9          2.0       2
41            4.5          2.3           1.3          0.3       0
76            6.8          2.8           4.8          1.4       1
25            5.0          3.0           1.6          0.2       0
130           7.4          2.8           6.1          1.9       2

             species  correct_prediction  prediction prediction_label

121   Iris-virginica                True           2   Iris-virginica

41       Iris-setosa                True           0      Iris-setosa

76   Iris-versicolor                True           1  Iris-versicolor

25       Iris-setosa                True           0      Iris-setosa

130   Iris-virginica                True           2   Iris-virginica
```

```
# plot_incorrect_predictions(df_predictions_test,
x_axis_feature="petal_length", y_axis_feature="petal_width") # we
created this function earlier
plot_incorrect_predictions(df_predictions_test, "petal_length",
"petal_width")
```



Accuracy of all the models

```
print("Accuracies of different models:")
for accu,name in model_accuracies:
    print(f"Accuracy of {name} Model : {accu * 100:.2f}%")

model_accuracies.sort(reverse=True)
print("*************************")
print(f"{model_accuracies[0][1]} Model has the best accuracy of
{model_accuracies[0][0]* 100:.2f}%")
print("*************************")
```

```
Accuracies of different models:
Accuracy of Logistic Regression Model : 97.37%
Accuracy of Manual Model : 94.64%
Accuracy of Support Vector Machine Model : 93.79%
Accuracy of Random Forest Model : 93.79%
Accuracy of K-Nearest Neighbors Model : 93.79%
*************************
Logistic Regression Model has the best accuracy of 97.37%
*************************
```

Conclusion:

Logistic Regression Model:

We achieve a 97% accuracy on the test dataset using a Logistic Regression model with these model parameters:

{'C': 1.9, 'class_weight': None, 'dual': False, 'fit_intercept': True, 'intercept_scaling': 1, 'l1_ratio': None, 'max_iter': 200, 'multi_class': 'deprecated', 'n_jobs': None, 'penalty': 'l2', 'random_state': None, 'solver': 'lbfgs', 'tol': 0.0001, 'verbose': 0, 'warm_start': False}

Random Forest Model:

We achieve a 94% accuracy on the test dataset using a Random Forest model with these model parameters:

{'bootstrap': True,'ccp_alpha': 0.0,'class_weight': None,'criterion': 'gini','max_depth': 2,'max_features': 'sqrt','max_leaf_nodes': None,'max_samples': None,'min_impurity_decrease': 0.0,'min_samples_leaf': 1,'min_samples_split': 2,'min_weight_fraction_leaf': 0.0,'monotonic_cst': None,'n_estimators': 50,'n_jobs': None,'oob_score': False,'random_state': None,'verbose': 0,'warm_start': False}

Support Vector Machines:

We achieve a 95% accuracy on the test dataset using Support Vector Machines model with these parameters:

{'C': 2.0,'break_ties': False,'cache_size': 200,'class_weight': None,'coef0': 0.0,'decision_function_shape': 'ovr','degree': 3,'gamma': 'scale','kernel': 'rbf','max_iter': -1,'probability': False,'random_state': None,'shrinking': True,'tol': 0.001,'verbose': False}

K - Nearest Neighbours:

We achieve a 96% accuracy on the test dataset using Support Vector Machines model with these parameters:

{'algorithm': 'auto','leaf_size': 30,'metric': 'manhattan','metric_params': None,'n_jobs': None,'n_neighbors': 13,'p': 2,'weights': 'distance'}

Observation:

The Logistic Regression model outperforms the other models in this case, achieving a higher accuracy. However, Random Forest might, Support Vector Machines and K - Nearest Neighbours still be useful in handling more complex patterns or larger datasets. Further tuning or feature selection could improve performance.