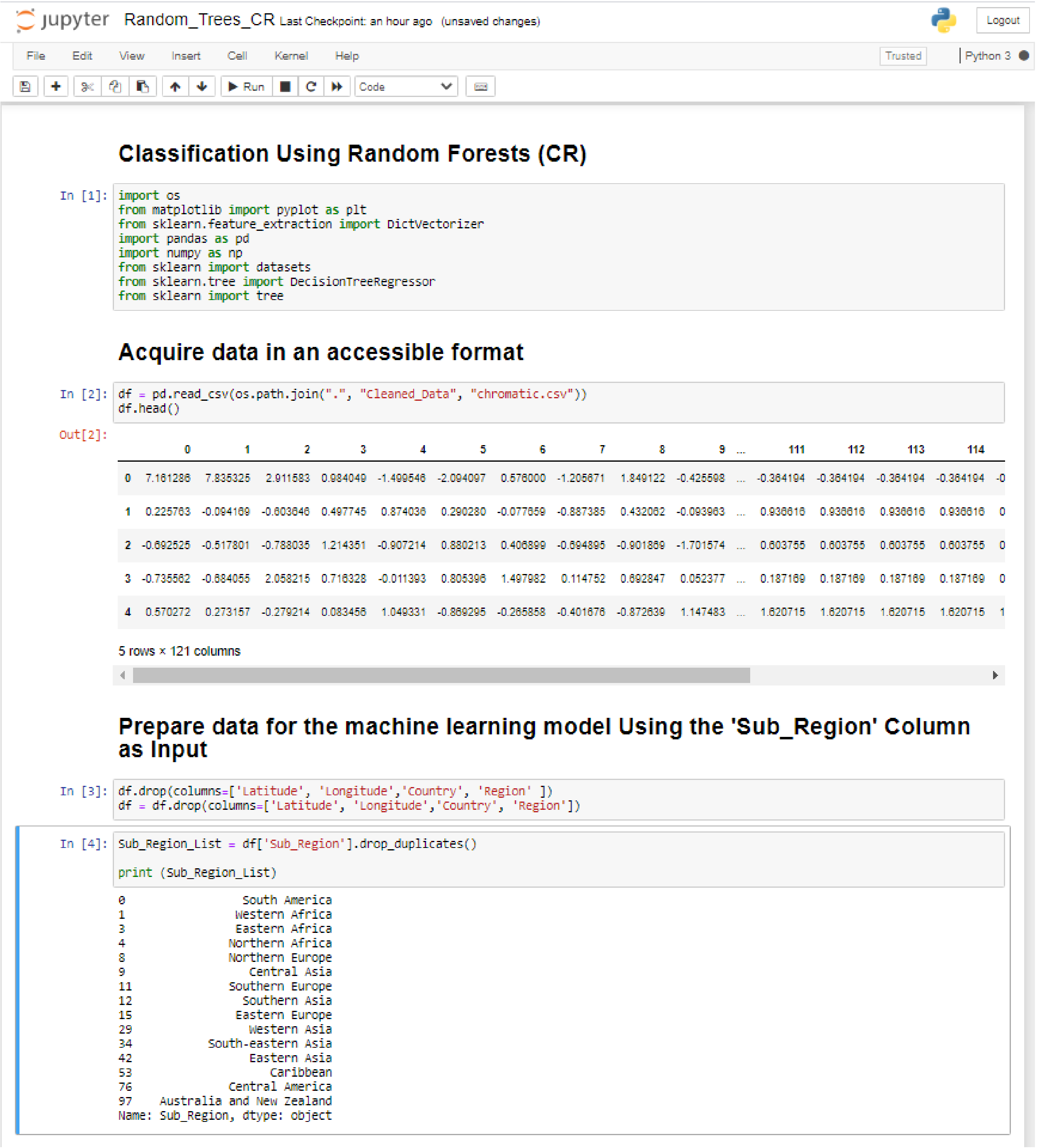


Classification Using Random Forests (CR)

In [1]:
```
import os
from matplotlib import pyplot as plt
from sklearn.feature_extraction import DictVectorizer
import pandas as pd
import numpy as np
from sklearn import datasets
from sklearn.tree import DecisionTreeRegressor
from sklearn import tree
```

Acquire data in an accessible format

In [2]:
```
df = pd.read_csv(os.path.join(".", "Cleaned_Data", "chromatic.csv"))
df.head()
```

Out[2]:

	0	1	2	3	4	5	6	7	8	9	...	111	112	113	114	
0	7.161286	7.835325	2.911583	0.984049	-1.499546	-2.094097	0.576000	-1.205671	1.849122	-0.425598	...	-0.364194	-0.364194	-0.364194	-0.364194	-0
1	0.225763	-0.094169	-0.603646	0.497745	0.874036	0.290280	-0.077659	-0.887385	0.432062	-0.093963	...	0.936616	0.936616	0.936616	0.936616	0
2	-0.692525	-0.517801	-0.788035	1.214351	-0.907214	0.880213	0.406899	-0.694895	-0.901869	-1.701574	...	0.603755	0.603755	0.603755	0.603755	0
3	-0.735562	-0.684055	2.058215	0.716328	-0.011393	0.805396	1.497982	0.114752	0.692847	0.052377	...	0.187169	0.187169	0.187169	0.187169	0
4	0.570272	0.273157	-0.279214	0.083456	1.049331	-0.869295	-0.265858	-0.401676	-0.872639	1.147483	...	1.620715	1.620715	1.620715	1.620715	1

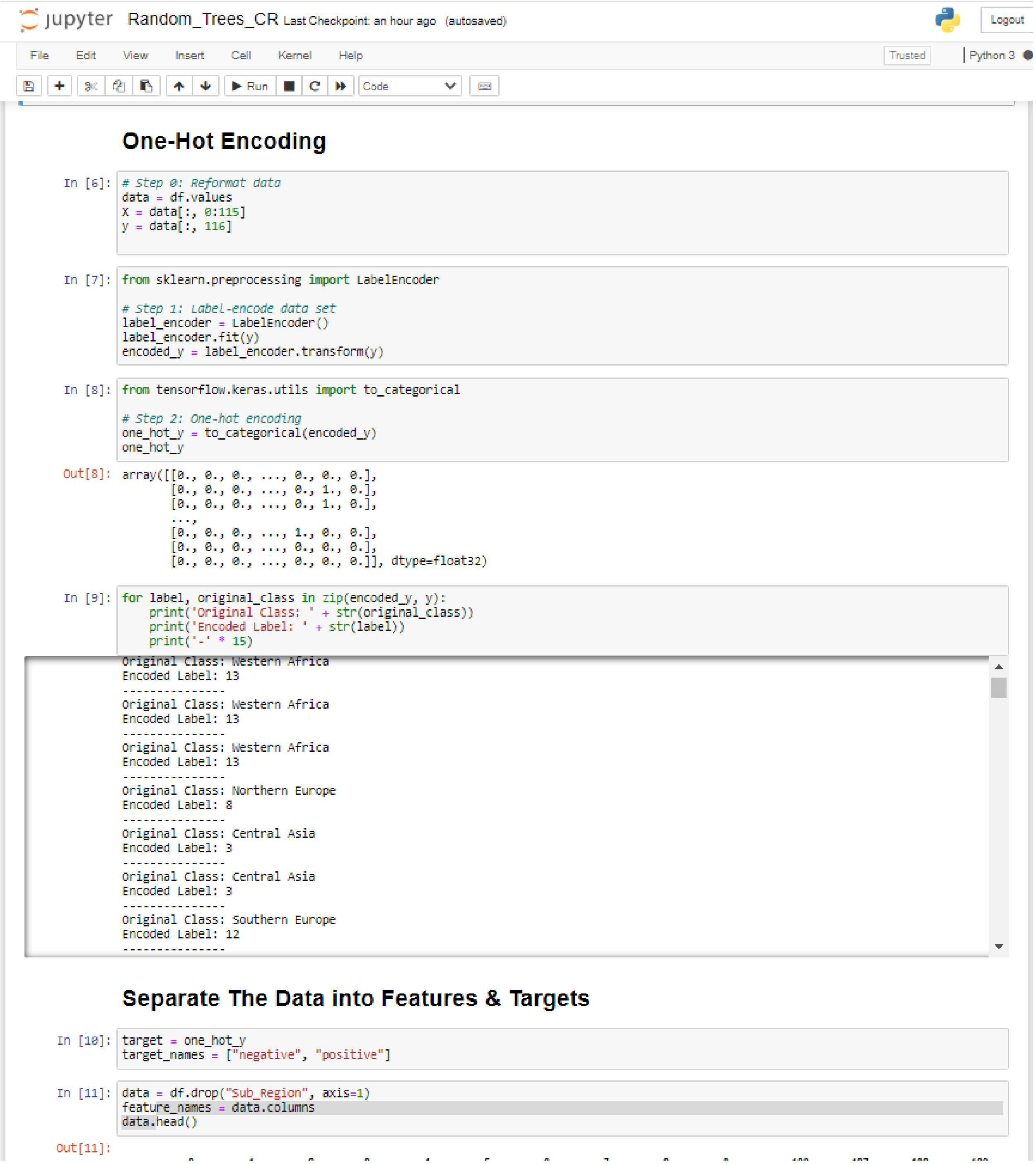
5 rows × 121 columns

Prepare data for the machine learning model Using the 'Sub_Region' Column as Input

In [3]:
```
df.drop(columns=['Latitude', 'Longitude','Country', 'Region' ])
df = df.drop(columns=['Latitude', 'Longitude','Country', 'Region'])
```

In [4]:
```
Sub_Region_List = df['Sub_Region'].drop_duplicates()

print (Sub_Region_List)
```

```
0                 South America
1                Western Africa
3                Eastern Africa
4               Northern Africa
8               Northern Europe
9                  Central Asia
11              Southern Europe
12                Southern Asia
15               Eastern Europe
29                 Western Asia
34           South-eastern Asia
42                 Eastern Asia
53                    Caribbean
76              Central America
97    Australia and New Zealand
Name: Sub_Region, dtype: object
```


jupyter Random_Trees_CR Last Checkpoint: an hour ago (autosaved)
Logout
File
Edit
View
Insert
Cell
Kernel
Help
Trusted
Python 3
Run
Code

One-Hot Encoding

```
In [6]: # Step 0: Reformat data
        data = df.values
        X = data[:, 0:115]
        y = data[:, 116]
```

```
In [7]: from sklearn.preprocessing import LabelEncoder

        # Step 1: Label-encode data set
        label_encoder = LabelEncoder()
        label_encoder.fit(y)
        encoded_y = label_encoder.transform(y)
```

```
In [8]: from tensorflow.keras.utils import to_categorical

        # Step 2: One-hot encoding
        one_hot_y = to_categorical(encoded_y)
        one_hot_y
```

```
Out[8]: array([[0., 0., 0., ..., 0., 0., 0.],
               [0., 0., 0., ..., 0., 1., 0.],
               [0., 0., 0., ..., 0., 1., 0.],
               ...,
               [0., 0., 0., ..., 1., 0., 0.],
               [0., 0., 0., ..., 0., 0., 0.],
               [0., 0., 0., ..., 0., 0., 0.]], dtype=float32)
```

```
In [9]: for label, original_class in zip(encoded_y, y):
            print('Original Class: ' + str(original_class))
            print('Encoded Label: ' + str(label))
            print('-' * 15)
```

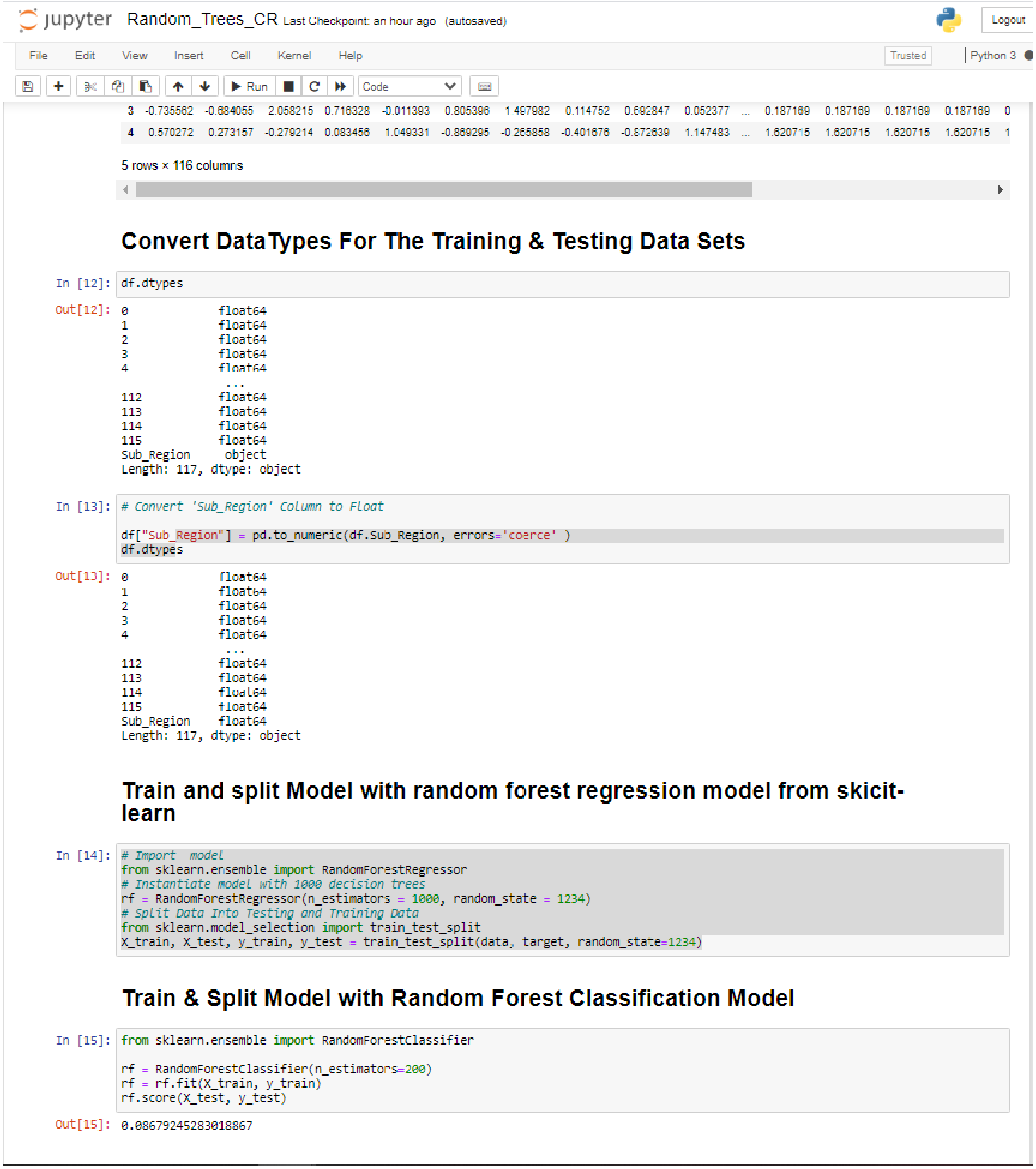
```
Original Class: Western Africa
Encoded Label: 13
---------------
Original Class: Western Africa
Encoded Label: 13
---------------
Original Class: Western Africa
Encoded Label: 13
---------------
Original Class: Northern Europe
Encoded Label: 8
---------------
Original Class: Central Asia
Encoded Label: 3
---------------
Original Class: Central Asia
Encoded Label: 3
---------------
Original Class: Southern Europe
Encoded Label: 12
---------------
```

Separate The Data into Features & Targets

```
In [10]: target = one_hot_y
         target_names = ["negative", "positive"]
```

```
In [11]: data = df.drop("Sub_Region", axis=1)
         feature_names = data.columns
         data.head()
```

```
Out[11]:
```



3	-0.735562	-0.684055	2.058215	0.716328	-0.011393	0.805396	1.497982	0.114752	0.692847	0.052377	...	0.187169	0.187169	0.187169	0.187169	0
4	0.570272	0.273157	-0.279214	0.083456	1.049331	-0.869295	-0.265858	-0.401676	-0.872639	1.147483	...	1.620715	1.620715	1.620715	1.620715	1

5 rows × 116 columns

Convert DataTypes For The Training & Testing Data Sets

In [12]:

```
df.dtypes
```

Out[12]:

```
0             float64
1             float64
2             float64
3             float64
4             float64
               ...
112           float64
113           float64
114           float64
115           float64
Sub_Region     object
Length: 117, dtype: object
```

In [13]:

```
# Convert 'Sub_Region' Column to Float

df["Sub_Region"] = pd.to_numeric(df.Sub_Region, errors='coerce' )
df.dtypes
```

Out[13]:

```
0             float64
1             float64
2             float64
3             float64
4             float64
               ...
112           float64
113           float64
114           float64
115           float64
Sub_Region    float64
Length: 117, dtype: object
```

Train and split Model with random forest regression model from skicit-learn

In [14]:

```
# Import  model
from sklearn.ensemble import RandomForestRegressor
# Instantiate model with 1000 decision trees
rf = RandomForestRegressor(n_estimators = 1000, random_state = 1234)
# Split Data Into Testing and Training Data
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(data, target, random_state=1234)
```

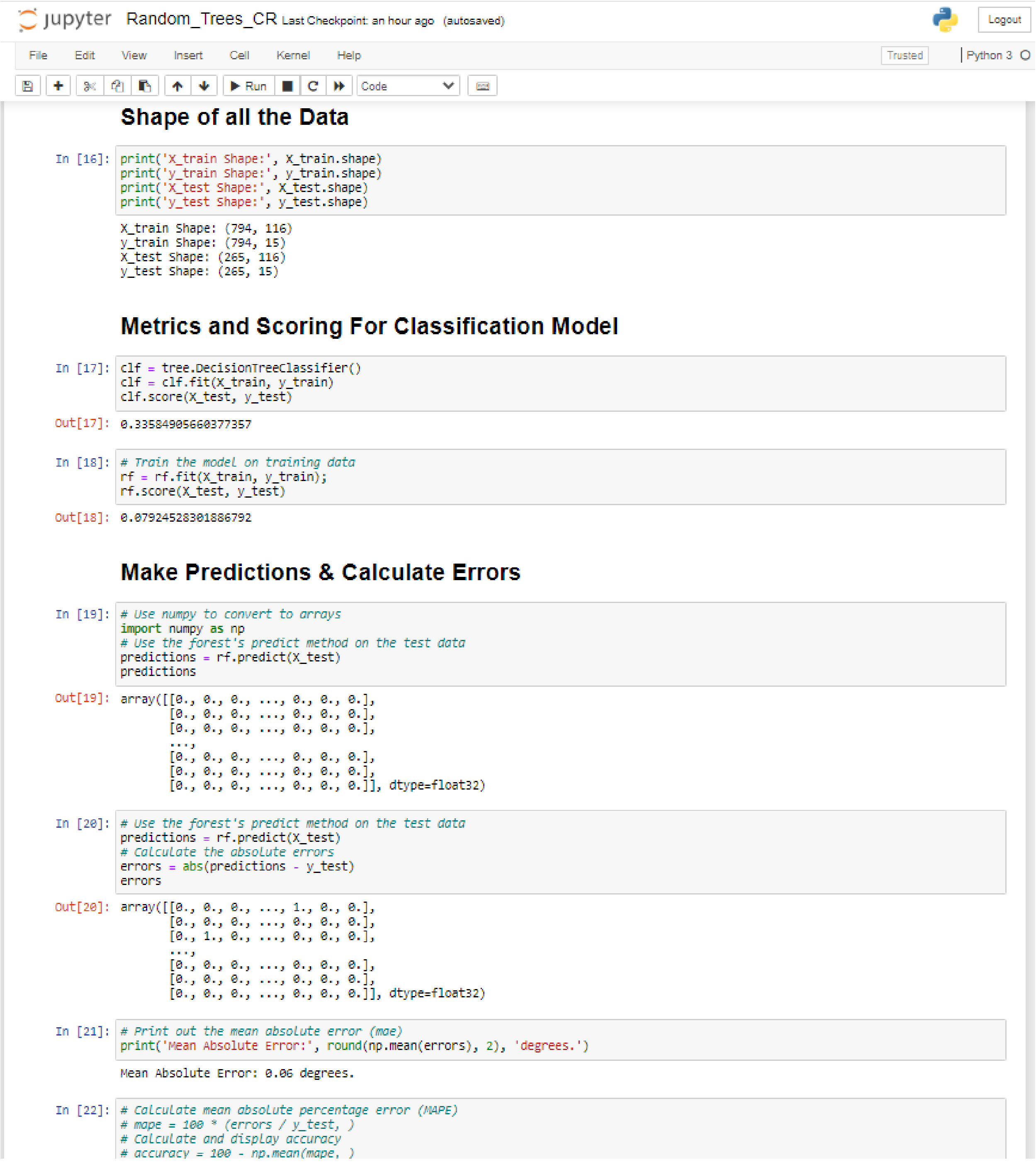
Train & Split Model with Random Forest Classification Model

In [15]:

```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(n_estimators=200)
rf = rf.fit(X_train, y_train)
rf.score(X_test, y_test)
```

Out[15]:

```
0.08679245283018867
```



Shape of all the Data

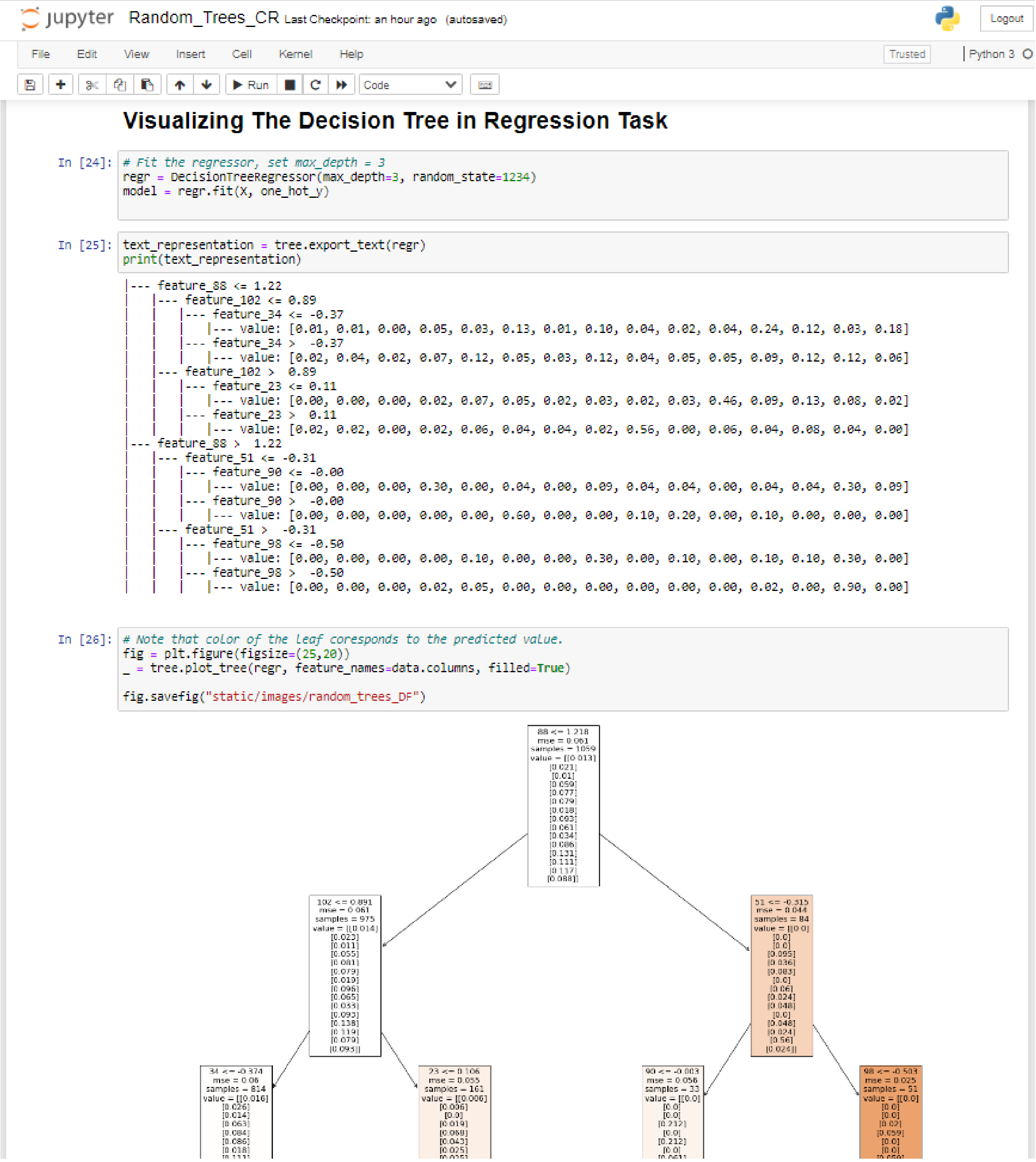
```
In [16]: print('X_train Shape:', X_train.shape)
         print('y_train Shape:', y_train.shape)
         print('X_test Shape:', X_test.shape)
         print('y_test Shape:', y_test.shape)

         X_train Shape: (794, 116)
         y_train Shape: (794, 15)
         X_test Shape: (265, 116)
         y_test Shape: (265, 15)
```

Metrics and Scoring For Classification Model

```
In [17]: clf = tree.DecisionTreeClassifier()
         clf = clf.fit(X_train, y_train)
         clf.score(X_test, y_test)

Out[17]: 0.33584905660377357

In [18]: # Train the model on training data
         rf = rf.fit(X_train, y_train);
         rf.score(X_test, y_test)

Out[18]: 0.07924528301886792
```

Make Predictions & Calculate Errors

```
In [19]: # Use numpy to convert to arrays
         import numpy as np
         # Use the forest's predict method on the test data
         predictions = rf.predict(X_test)
         predictions

Out[19]: array([[0., 0., 0., ..., 0., 0., 0.],
                [0., 0., 0., ..., 0., 0., 0.],
                [0., 0., 0., ..., 0., 0., 0.],
                ...,
                [0., 0., 0., ..., 0., 0., 0.],
                [0., 0., 0., ..., 0., 0., 0.],
                [0., 0., 0., ..., 0., 0., 0.]], dtype=float32)

In [20]: # Use the forest's predict method on the test data
         predictions = rf.predict(X_test)
         # Calculate the absolute errors
         errors = abs(predictions - y_test)
         errors

Out[20]: array([[0., 0., 0., ..., 1., 0., 0.],
                [0., 0., 0., ..., 0., 0., 0.],
                [0., 1., 0., ..., 0., 0., 0.],
                ...,
                [0., 0., 0., ..., 0., 0., 0.],
                [0., 0., 0., ..., 0., 0., 0.],
                [0., 0., 0., ..., 0., 0., 0.]], dtype=float32)

In [21]: # Print out the mean absolute error (mae)
         print('Mean Absolute Error:', round(np.mean(errors), 2), 'degrees.')

         Mean Absolute Error: 0.06 degrees.

In [22]: # Calculate mean absolute percentage error (MAPE)
         # mape = 100 * (errors / y_test, )
         # Calculate and display accuracy
         # accuracy = 100 - np.mean(mape, )
```



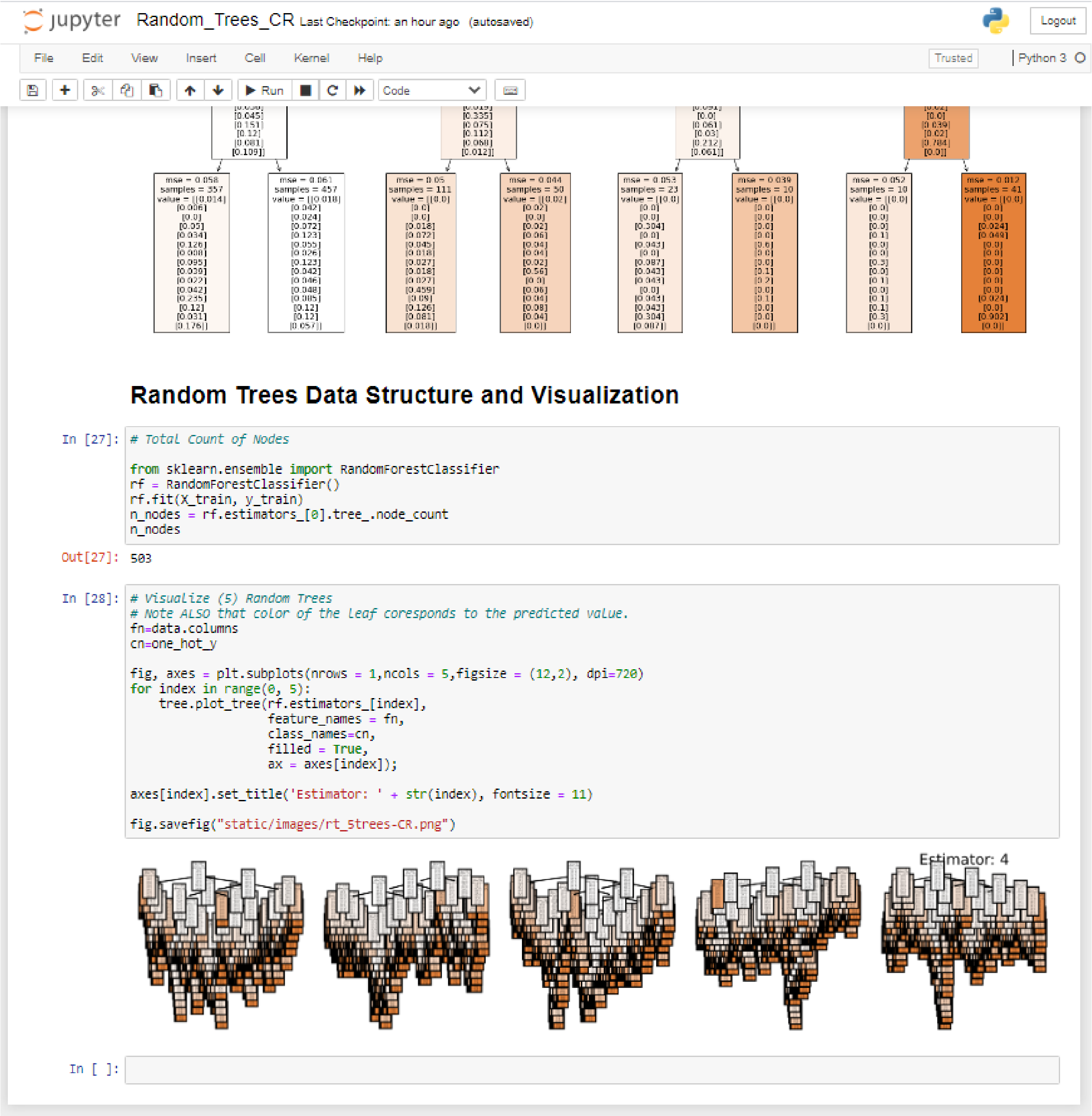
Visualizing The Decision Tree in Regression Task

In [24]:
```
# Fit the regressor, set max_depth = 3
regr = DecisionTreeRegressor(max_depth=3, random_state=1234)
model = regr.fit(X, one_hot_y)
```

In [25]:
```
text_representation = tree.export_text(regr)
print(text_representation)
```

```
|--- feature_88 <= 1.22
|   |--- feature_102 <= 0.89
|   |   |--- feature_34 <= -0.37
|   |   |   |--- value: [0.01, 0.01, 0.00, 0.05, 0.03, 0.13, 0.01, 0.10, 0.04, 0.02, 0.04, 0.24, 0.12, 0.03, 0.18]
|   |   |--- feature_34 >  -0.37
|   |   |   |--- value: [0.02, 0.04, 0.02, 0.07, 0.12, 0.05, 0.03, 0.12, 0.04, 0.05, 0.05, 0.09, 0.12, 0.12, 0.06]
|   |--- feature_102 >  0.89
|   |   |--- feature_23 <= 0.11
|   |   |   |--- value: [0.00, 0.00, 0.00, 0.02, 0.07, 0.05, 0.02, 0.03, 0.02, 0.03, 0.46, 0.09, 0.13, 0.08, 0.02]
|   |   |--- feature_23 >  0.11
|   |   |   |--- value: [0.02, 0.02, 0.00, 0.02, 0.06, 0.04, 0.04, 0.02, 0.56, 0.00, 0.06, 0.04, 0.08, 0.04, 0.00]
|--- feature_88 >  1.22
|   |--- feature_51 <= -0.31
|   |   |--- feature_90 <= -0.00
|   |   |   |--- value: [0.00, 0.00, 0.00, 0.30, 0.00, 0.04, 0.00, 0.09, 0.04, 0.04, 0.00, 0.04, 0.04, 0.30, 0.09]
|   |   |--- feature_90 >  -0.00
|   |   |   |--- value: [0.00, 0.00, 0.00, 0.00, 0.00, 0.60, 0.00, 0.00, 0.10, 0.20, 0.00, 0.10, 0.00, 0.00, 0.00]
|   |--- feature_51 >  -0.31
|   |   |--- feature_98 <= -0.50
|   |   |   |--- value: [0.00, 0.00, 0.00, 0.00, 0.10, 0.00, 0.00, 0.30, 0.00, 0.10, 0.00, 0.10, 0.10, 0.30, 0.00]
|   |   |--- feature_98 >  -0.50
|   |   |   |--- value: [0.00, 0.00, 0.00, 0.02, 0.05, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.02, 0.00, 0.90, 0.00]
```

In [26]:
```
# Note that color of the leaf coresponds to the predicted value.
fig = plt.figure(figsize=(25,20))
_ = tree.plot_tree(regr, feature_names=data.columns, filled=True)

fig.savefig("static/images/random_trees_DF")
```



Random Trees Data Structure and Visualization

In [27]:

```
# Total Count of Nodes

from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier()
rf.fit(X_train, y_train)
n_nodes = rf.estimators_[0].tree_.node_count
n_nodes
```

Out[27]:

```
503
```

In [28]:

```
# Visualize (5) Random Trees
# Note ALSO that color of the leaf coresponds to the predicted value.
fn=data.columns
cn=one_hot_y

fig, axes = plt.subplots(nrows = 1,ncols = 5,figsize = (12,2), dpi=720)
for index in range(0, 5):
    tree.plot_tree(rf.estimators_[index],
                   feature_names = fn,
                   class_names=cn,
                   filled = True,
                   ax = axes[index]);

axes[index].set_title('Estimator: ' + str(index), fontsize = 11)

fig.savefig("static/images/rt_5trees-CR.png")
```