**CODE**

**In[ ]**

# Pandas is used for data manipulation

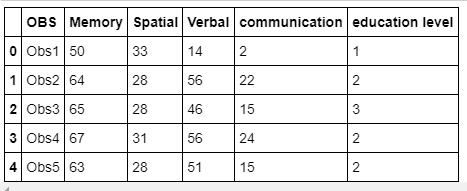
**import pandas as pd**

# Read in data as pandas dataframe and display first 5 rows

features = pd.read\_csv('Manova.csv')

features.head(5)

Out[]



**In [ ]:**

print('The shape of our features is:', features.shape)

**Out[ ]**

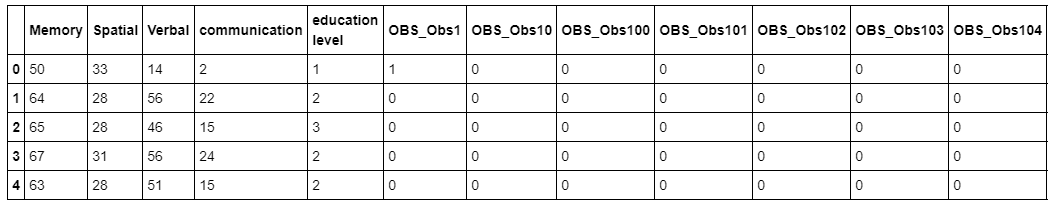
The shape of our features is: (150, 6)

**In [ ]:**

features = pd.get\_dummies(features)

features.head(5)

**Out[ ]:**



155 columns

**In [ ]:**

print('Shape of features after one-hot encoding:', features.shape)

**Out[ ]**

Shape of features after one-hot encoding: (150, 155)

**In[ ]**

# Use numpy to convert to arrays

**import** **numpy** **as** **np**

# Labels are the values we want to predict

labels = np.array(features['Verbal'])

# Remove the labels from the features

# axis 1 refers to the columns

features= features.drop('Verbal', axis = 1)

# Saving feature names for later use

feature\_list = list(features.columns)

# Convert to numpy array

features = np.array(features)

**In [ ]:**

# Using Skicit-learn to split data into training and testing sets

**from** **sklearn.model\_selection** **import** train\_test\_split

# Split the data into training and testing sets

train\_features, test\_features, train\_labels, test\_labels = train\_test\_split(features, labels, test\_size = 0.25,random\_state = 42)

**In [ ]:**

print('Training Features Shape:', train\_features.shape)

print('Training Labels Shape:', train\_labels.shape)

print('Testing Features Shape:', test\_features.shape)

print('Testing Labels Shape:', test\_labels.shape)

**Out[ ]**

Training Features Shape: (112, 154)

Training Labels Shape: (112,)

Testing Features Shape: (38, 154)

Testing Labels Shape: (38,)

**In [ ]:**

# Import the model we are using

**from** **sklearn.ensemble** **import** RandomForestRegressor

# Instantiate model

rf = RandomForestRegressor(n\_estimators= 1000, random\_state=42)

# Train the model on training data

rf.fit(train\_features, train\_labels);

**In [ ]:**

rf\_new = RandomForestRegressor(n\_estimators = 100, criterion = 'mse', max\_depth = **None**,

min\_samples\_split = 2, min\_samples\_leaf = 1)

**In [ ]:**

# Use the forest's predict method on the test data

predictions = rf.predict(test\_features)

# Calculate the absolute errors

errors = abs(predictions - test\_labels)

# Print out the mean absolute error (mae)

print('Mean Absolute Error:', round(np.mean(errors), 2), 'degrees.')

**Out[ ]**

Mean Absolute Error: 2.34 degrees.

**In [ ]:**

# Calculate mean absolute percentage error (MAPE)

mape = 100 \* (errors / test\_labels)

# Calculate and display accuracy

accuracy = 100 - np.mean(mape)

print('Accuracy:', round(accuracy, 2), '%.')

**Out[ ]**

Accuracy: 93.25 %.

**In [ ]:**

# Import tools needed for visualization

**from** **sklearn.tree** **import** export\_graphviz

**import** **pydot**

# Pull out one tree from the forest

tree = rf.estimators\_[5]

# Export the image to a dot file

export\_graphviz(tree, out\_file = 'tree.dot', feature\_names = feature\_list, rounded = **True**, precision = 1)

# Use dot file to create a graph

(graph, ) = pydot.graph\_from\_dot\_file('tree.dot')

# Write graph to a png file

graph.write\_png('tree.png');

**In [ ]:**

print('The depth of this tree is:', tree.tree\_.max\_depth)

**Out[ ]**

The depth of this tree is: 12

**In [ ]:**

# Limit depth of tree to 2 levels

rf\_small = RandomForestRegressor(n\_estimators=10, max\_depth = 3, random\_state=42)

rf\_small.fit(train\_features, train\_labels)

# Extract the small tree

tree\_small = rf\_small.estimators\_[5]

# Save the tree as a png image

export\_graphviz(tree\_small, out\_file = 'small\_tree.dot', feature\_names = feature\_list, rounded = **True**, precision = 1)

(graph, ) = pydot.graph\_from\_dot\_file('small\_tree.dot')

graph.write\_png('small\_tree.png')

**In [ ]:**

# Get numerical feature importances

importances = list(rf.feature\_importances\_)

# List of tuples with variable and importance

feature\_importances = [(feature, round(importance, 2)) **for** feature, importance **in** zip(feature\_list, importances)]

# Sort the feature importances by most important first

feature\_importances = sorted(feature\_importances, key = **lambda** x: x[1], reverse = **True**)

# Print out the feature and importances

[print('Variable: **{:20}** Importance: **{}**'.format(\*pair)) **for** pair **in** feature\_importances];

**In [ ]:**

# New random forest with only the two most important variables

rf\_most\_important = RandomForestRegressor(n\_estimators= 1000, random\_state=42)

# Extract the two most important features

important\_indices = [feature\_list.index('Spatial'), feature\_list.index('Memory')]

train\_important = train\_features[:, important\_indices]

test\_important = test\_features[:, important\_indices]

# Train the random forest

rf\_most\_important.fit(train\_important, train\_labels)

# Make predictions and determine the error

predictions = rf\_most\_important.predict(test\_important)

errors = abs(predictions - test\_labels)

# Display the performance metrics

print('Mean Absolute Error:', round(np.mean(errors), 2), 'degrees.')

mape = np.mean(100 \* (errors / test\_labels))

accuracy = 100 - mape

print('Accuracy:', round(accuracy, 2), '%.')

**Out[ ]**

Mean Absolute Error: 3.67 degrees.

Accuracy: 90.47 %.

**In [ ]:**

**from** **IPython.display** **import** Image

Image('tree.png')

**In [ ]:**

**from** **IPython.display** **import** Image

Image('small\_tree.png')

**In [ ]:**

**import** **matplotlib.pyplot** **as** **plt**

%matplotlib inline

# Set the style

plt.style.use('fivethirtyeight')

# list of x locations for plotting

x\_values = list(range(len(importances)))

# Make a bar chart

plt.bar(x\_values, importances, orientation = 'vertical')

# Tick labels for x axis

plt.xticks(x\_values, feature\_list, rotation='vertical')

# Axis labels and title

plt.ylabel('Importance'); plt.xlabel('Variable'); plt.title('Variable Importances');