Assignment-15-Random Forest (Company Data)

Random Forest

Assignment

About the data:Let's consider a company dataset with around 10 variables and 400 records.

Sales -- Unit sales (in thousands) at each location Competitor Price -- Price charged by competitor at each location Income -- Community income level (in thousands of dollars) Advertising -- Local advertising budget for company at each location (in thousands of dollars) Population -- Population size in region (in thousands) Price -- Price company charges for car seats at each site Shelf Location at stores -- A factor with levels Bad, Good and Medium indicating the quality of the shelving location for the car seats at each site Age -- Average age of the local population Education -- Education level at each location Urban -- A factor with levels No and Yes to indicate whether the store is in an urban or rural location US -- A factor with levels No and Yes to indicate whether the store is in the US or not The company dataset looks like this:

Problem Statement:

A cloth manufacturing company is interested to know about the segment or attributes causes high sale. Approach - A Random Forest can be built with target variable Sales (we will first convert it in categorical variable) & all other variable will be independent in the analysis.

Import Libraries

In [131]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as pd
import seaborn as sns
%matplotlib inline #for encoding
from sklearn.preprocessing import LabelEncoder  # for train test splitting,
from sklearn.model_selection import train_test_split  # for decision tree object
from sklearn.tree import DecisionTreeClassifier  # for checking testing results
from sklearn.metrics import classification_report,confusion_matrix  # for visualizing tree
from sklearn.tree import plot_tree
import warnings
warnings.filterwarnings('ignore')
```

```
UsageError: unrecognized arguments: #for encoding
```

Import Data

In [132]:
```
# Pandas is used for data manipulation
import pandas as pd
# Read in data and display first 5 rows
features = pd.read_csv('Company_Data.csv')
features.head(5)
```

Out[132]:

	Sales	CompPrice	Income	Advertising	Population	Price	ShelveLoc	Age	Education	Urban	US
0	9.50	138	73	11	276	120	Bad	42	17	Yes	Yes
1	11.22	111	48	16	260	83	Good	65	10	Yes	Yes
2	10.06	113	35	10	269	80	Medium	59	12	Yes	Yes
3	7.40	117	100	4	466	97	Medium	55	14	Yes	Yes
4	4.15	141	64	3	340	128	Bad	38	13	Yes	No

In [133]:
```
# getting information of dataset
features.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Sales        400 non-null    float64
 1   CompPrice    400 non-null    int64
 2   Income       400 non-null    int64
 3   Advertising  400 non-null    int64
 4   Population   400 non-null    int64
 5   Price        400 non-null    int64
 6   ShelveLoc    400 non-null    object
 7   Age          400 non-null    int64
 8   Education    400 non-null    int64
 9   Urban        400 non-null    object
 10  US           400 non-null    object
dtypes: float64(1), int64(7), object(3)
memory usage: 34.5+ KB
```

In [134]:
```
print('The Shape of our features is:',features.shape)
```

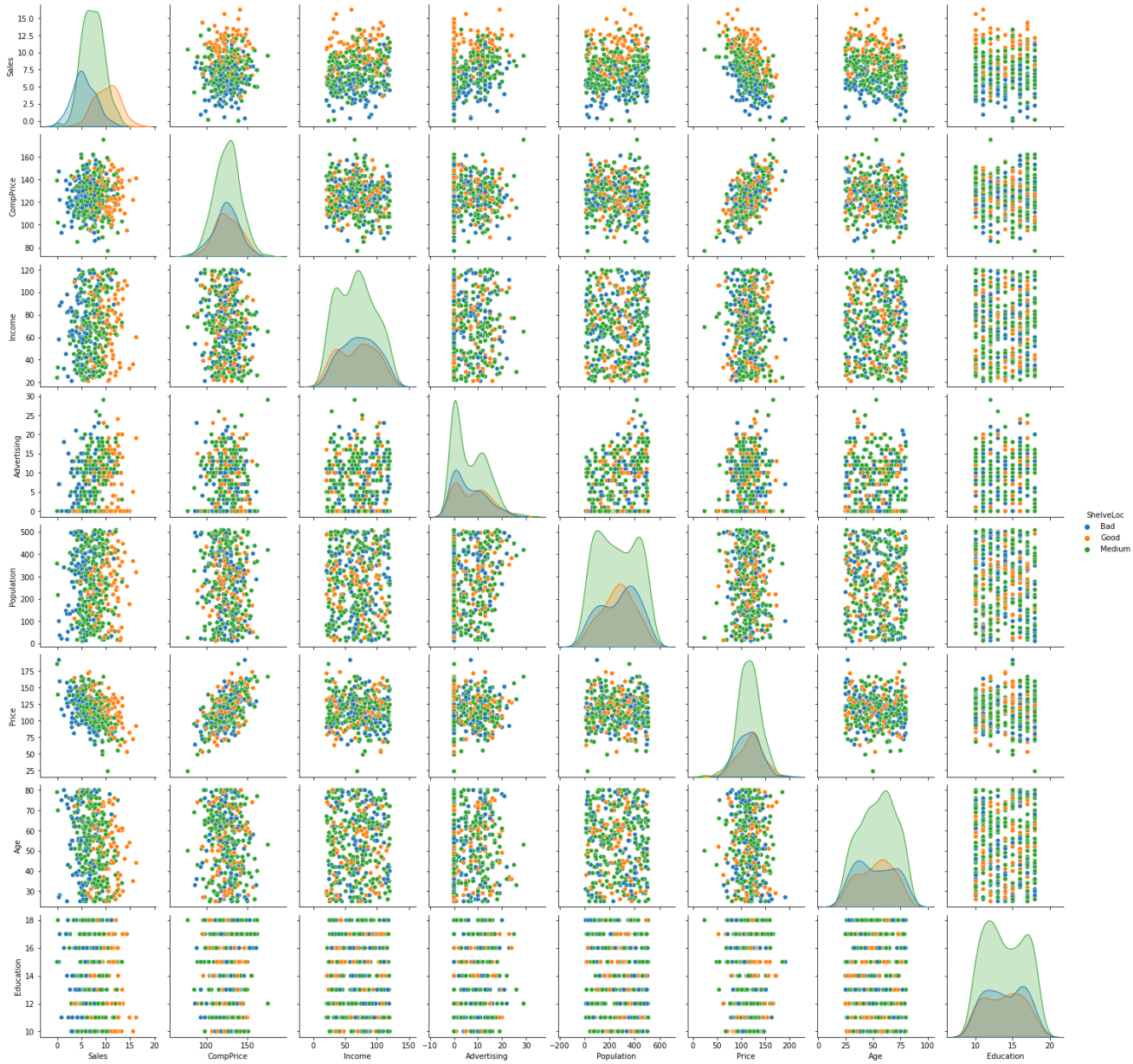
```
The Shape of our features is: (400, 11)
```

In [135]:
```
features.isnull().any()
```

Out[135]:
```
Sales          False
CompPrice      False
Income         False
Advertising    False
Population     False
Price          False
ShelveLoc      False
Age            False
Education      False
Urban          False
US             False
dtype: bool
```

In [136]:
```
# lets plot pair plot to visualise the attributes all at once.
sns.pairplot(data=features, hue = 'ShelveLoc')
```

Out[136]: <seaborn.axisgrid.PairGrid at 0x227371a58b0>



In [137]:
```
# Creating dummy variables dropping first dummy variable
df=pd.get_dummies(features,columns=['Urban','US'], drop_first=True)
```

In [138]:
```
print(df.head())
```

```
   Sales  CompPrice  Income  Advertising  Population  Price ShelveLoc  Age  \
0   9.50        138      73           11         276    120       Bad   42
1  11.22        111      48           16         260     83      Good   65
2  10.06        113      35           10         269     80    Medium   59
3   7.40        117     100            4         466     97    Medium   55
4   4.15        141      64            3         340    128       Bad   38

   Education  Urban_Yes  US_Yes
0         17          1       1
1         10          1       1
2         12          1       1
3         14          1       1
4         13          1       0
```

In [139]:
```
from sklearn.metrics import f1_score
from sklearn.model_selection import train_test_split
```

In [140]:
```
df['ShelveLoc']=df['ShelveLoc'].map({'Good':1,'Medium':2,'Bad':3})
```

In [141]:
```
print(df.head())
```

```
   Sales  CompPrice  Income  Advertising  Population  Price  ShelveLoc  Age  \
0   9.50        138      73           11         276    120          3   42
1  11.22        111      48           16         260     83          1   65
2  10.06        113      35           10         269     80          2   59
3   7.40        117     100            4         466     97          2   55
4   4.15        141      64            3         340    128          3   38

   Education  Urban_Yes  US_Yes
0         17          1       1
1         10          1       1
2         12          1       1
3         14          1       1
4         13          1       0
```

In [142]:
```
x=df.iloc[:,0:6]
y=df['ShelveLoc']
x
```

Out[142]:

	Sales	CompPrice	Income	Advertising	Population	Price
0	9.50	138	73	11	276	120
1	11.22	111	48	16	260	83
2	10.06	113	35	10	269	80
3	7.40	117	100	4	466	97
4	4.15	141	64	3	340	128
...	...	...	...	...	...	...
395	12.57	138	108	17	203	128
396	6.14	139	23	3	37	120
397	7.41	162	26	12	368	159
398	5.94	100	79	7	284	95
399	9.71	134	37	0	27	120

400 rows × 6 columns

In [143]:
```
y
```

Out[143]:
```
0      3
1      1
2      2
3      2
4      3
      ..
395    1
396    2
397    2
398    3
399    1
Name: ShelveLoc, Length: 400, dtype: int64
```

In [144]:
```
df['ShelveLoc'].unique()
```

Out[144]:
```
array([3, 1, 2], dtype=int64)
```

In [145]:
```
df.ShelveLoc.value_counts()
```

Out[145]:
```
2    219
3     96
1     85
Name: ShelveLoc, dtype: int64
```

In [146]:
```
colnames = list(df.columns)
colnames
```

Out[146]:
```
['Sales',
 'CompPrice',
 'Income',
 'Advertising',
 'Population',
 'Price',
 'ShelveLoc',
 'Age',
 'Education',
 'Urban_Yes',
 'US_Yes']
```

In [147]:
```
# Descriptive statistics for each column
df.describe()
```

Out[147]:

	Sales	CompPrice	Income	Advertising	Population	Price	ShelveLoc	Age	Education	Urban_Yes	US_Yes
count	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000
mean	7.496325	124.975000	68.657500	6.635000	264.840000	115.795000	2.027500	53.322500	13.900000	0.705000	0.645000
std	2.824115	15.334512	27.986037	6.650364	147.376436	23.676664	0.672961	16.200297	2.620528	0.456614	0.479113
min	0.000000	77.000000	21.000000	0.000000	10.000000	24.000000	1.000000	25.000000	10.000000	0.000000	0.000000
25%	5.390000	115.000000	42.750000	0.000000	139.000000	100.000000	2.000000	39.750000	12.000000	0.000000	0.000000
50%	7.490000	125.000000	69.000000	5.000000	272.000000	117.000000	2.000000	54.500000	14.000000	1.000000	1.000000
75%	9.320000	135.000000	91.000000	12.000000	398.500000	131.000000	2.000000	66.000000	16.000000	1.000000	1.000000
max	16.270000	175.000000	120.000000	29.000000	509.000000	191.000000	3.000000	80.000000	18.000000	1.000000	1.000000

In [148]:
```
df.head()
```

Out[148]:

	Sales	CompPrice	Income	Advertising	Population	Price	ShelveLoc	Age	Education	Urban_Yes	US_Yes
0	9.50	138	73	11	276	120	3	42	17	1	1
1	11.22	111	48	16	260	83	1	65	10	1	1
2	10.06	113	35	10	269	80	2	59	12	1	1
3	7.40	117	100	4	466	97	2	55	14	1	1
4	4.15	141	64	3	340	128	3	38	13	1	0

In [149]:
```
# Labels are the values we want to predict
labels = np.array(df['Income'])
# Remove the labels from the features
# axis 1 refers to the columns
features=df.drop('Income', axis = 1)
# Saving feature names for later use
features_list = list(df.columns)
# Convert to numpy array
features = np.array(df)
```

In [150]:
```
# 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
```

In [151]:
```
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)
```

```
Training Features Shape: (300, 11)
Training Labels Shape: (300,)
Testing Features Shape: (100, 11)
Testing Labels Shape: (100,)
```

Establish Baseline

In [152]:
```
# The baseline predictions are the historical averages
baseline_preds = test_features[:, features_list.index('Sales')]
# Baseline errors, and display average baseline error
baseline_errors = abs(baseline_preds - test_labels)
print('Average baseline error: ', round(np.mean(baseline_errors), 2))
```

```
Average baseline error:  65.26
```

In [153]:
```
# Import the model we are using
from sklearn.ensemble import RandomForestRegressor
# Instantiate model with 1000 decision trees
rf = RandomForestRegressor(n_estimators = 1000, random_state = 42)
# Train the model on training data
rf.fit(train_features, train_labels);
```

In [154]:
```
# 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('Meam Absolute Error:', round(np.mean(errors), 2), 'degrees.')
```

```
Meam Absolute Error: 0.27 degrees.
```

Determine Performance Metrics

In [155]:
```
# 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), '%.')
```

```
Accuracy: 99.58 %.
```

In [156]:
```
!pip install pydot
```

```
Requirement already satisfied: pydot in c:\users\admin\anaconda3\lib\site-packages (1.4.2)
Requirement already satisfied: pyparsing>=2.1.4 in c:\users\admin\anaconda3\lib\site-packages (from pydot) (3.0.4)
```

In [157]:
```
!pip install graphviz
```

```
Requirement already satisfied: graphviz in c:\users\admin\anaconda3\lib\site-packages (0.20.1)
```

In [171]:
```
graph.write_png
```

Out[171]:
```
<function pydot.Dot.__init__.<locals>.new_method(path, f='png', prog='dot', encoding=None)>
```

In [169]:

```
# Import tools needed for visualization
from sklearn.tree import export_graphviz
import pydot
# Pull out one tree from the forest
tree = rf.estimators_[5]
# 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 = features_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')
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
~\anaconda3\lib\site-packages\pydot.py in create(self, prog, format, encoding)
   1922         try:
-> 1923             stdout_data, stderr_data, process = call_graphviz(
   1924                 program=prog,

~\anaconda3\lib\site-packages\pydot.py in call_graphviz(program, arguments, working_dir, **kwargs)
    131
--> 132     process = subprocess.Popen(
    133         program_with_args,

~\anaconda3\lib\subprocess.py in __init__(self, args, bufsize, executable, stdin, stdout, stderr, preexec_fn, close_fds, shell,
cwd, env, universal_newlines, startupinfo, creationflags, restore_signals, start_new_session, pass_fds, user, group, extra_grou
ps, encoding, errors, text, umask)
    950
--> 951             self._execute_child(args, executable, preexec_fn, close_fds,
    952                                 pass_fds, cwd, env,

~\anaconda3\lib\subprocess.py in _execute_child(self, args, executable, preexec_fn, close_fds, pass_fds, cwd, env, startupinfo,
creationflags, shell, p2cread, p2cwrite, c2pread, c2pwrite, errread, errwrite, unused_restore_signals, unused_gid, unused_gids,
unused_uid, unused_umask, unused_start_new_session)
   1419             try:
-> 1420                 hp, ht, pid, tid = _winapi.CreateProcess(executable, args,
   1421                                          # no special security

FileNotFoundError: [WinError 2] The system cannot find the file specified

During handling of the above exception, another exception occurred:

FileNotFoundError                         Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_6664/2350536375.py in <module>
     14 (graph, ) = pydot.graph_from_dot_file('tree.dot')
     15 # Write graph to a png file
---> 16 graph.write_png('tree.png')

~\anaconda3\lib\site-packages\pydot.py in new_method(path, f, prog, encoding)
   1741                     encoding=None):
   1742                 """Refer to docstring of method `write.`"""
-> 1743                 self.write(
   1744                     path, format=f, prog=prog,
   1745                     encoding=encoding)

~\anaconda3\lib\site-packages\pydot.py in write(self, path, prog, format, encoding)
   1826                 f.write(s)
   1827         else:
-> 1828             s = self.create(prog, format, encoding=encoding)
   1829             with io.open(path, mode='wb') as f:
   1830                 f.write(s)

~\anaconda3\lib\site-packages\pydot.py in create(self, prog, format, encoding)
   1931                 args[1] = '"{prog}" not found in path.'.format(
   1932                     prog=prog)
-> 1933                 raise OSError(*args)
   1934             else:
   1935                 raise

FileNotFoundError: [WinError 2] "dot" not found in path.
```

In [166]:

```
# Import tools needed for visualization
from sklearn.tree import export_graphviz
import pydot
import graphviz
# Pull out one tree from the forest
tree = rf.estimators_[5]
# 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 = features_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')
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
~\anaconda3\lib\site-packages\pydot.py in create(self, prog, format, encoding)
   1922         try:
-> 1923             stdout_data, stderr_data, process = call_graphviz(
   1924                 program=prog,

~\anaconda3\lib\site-packages\pydot.py in call_graphviz(program, arguments, working_dir, **kwargs)
    131
--> 132     process = subprocess.Popen(
    133         program_with_args,

~\anaconda3\lib\subprocess.py in __init__(self, args, bufsize, executable, stdin, stdout, stderr, preexec_fn, close_fds, shell,
cwd, env, universal_newlines, startupinfo, creationflags, restore_signals, start_new_session, pass_fds, user, group, extra_grou
ps, encoding, errors, text, umask)
    950
--> 951             self._execute_child(args, executable, preexec_fn, close_fds,
    952                                 pass_fds, cwd, env,

~\anaconda3\lib\subprocess.py in _execute_child(self, args, executable, preexec_fn, close_fds, pass_fds, cwd, env, startupinfo,
creationflags, shell, p2cread, p2cwrite, c2pread, c2pwrite, errread, errwrite, unused_restore_signals, unused_gid, unused_gids,
unused_uid, unused_umask, unused_start_new_session)
   1419             try:
-> 1420                 hp, ht, pid, tid = _winapi.CreateProcess(executable, args,
   1421                                          # no special security

FileNotFoundError: [WinError 2] The system cannot find the file specified

During handling of the above exception, another exception occurred:

FileNotFoundError                         Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_6664/2479579253.py in <module>
     15 (graph, ) = pydot.graph_from_dot_file('tree.dot')
     16 # Write graph to a png file
---> 17 graph.write_png('tree.png')

~\anaconda3\lib\site-packages\pydot.py in new_method(path, f, prog, encoding)
   1741                     encoding=None):
   1742                 """Refer to docstring of method `write.`"""
-> 1743                 self.write(
   1744                     path, format=f, prog=prog,
   1745                     encoding=encoding)

~\anaconda3\lib\site-packages\pydot.py in write(self, path, prog, format, encoding)
   1826                 f.write(s)
   1827         else:
-> 1828             s = self.create(prog, format, encoding=encoding)
   1829             with io.open(path, mode='wb') as f:
   1830                 f.write(s)

~\anaconda3\lib\site-packages\pydot.py in create(self, prog, format, encoding)
   1931                 args[1] = '"{prog}" not found in path.'.format(
   1932                     prog=prog)
-> 1933                 raise OSError(*args)
   1934             else:
   1935                 raise

FileNotFoundError: [WinError 2] "dot" not found in path.
```

In [159]:
```
# Limit depth of tree to 3 levels
rf_small = RandomForestRegressor(n_estimators=10, max_depth = 3)
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 = features_list, rounded = True, precision = 1)
(graph, ) = pydot.graph_from_dot_file('small_tree.dot')
graph.write_png('small_tree.png');
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
~\anaconda3\lib\site-packages\pydot.py in create(self, prog, format, encoding)
   1922         try:
-> 1923             stdout_data, stderr_data, process = call_graphviz(
   1924                 program=prog,

~\anaconda3\lib\site-packages\pydot.py in call_graphviz(program, arguments, working_dir, **kwargs)
    131
--> 132     process = subprocess.Popen(
    133         program_with_args,

~\anaconda3\lib\subprocess.py in __init__(self, args, bufsize, executable, stdin, stdout, stderr, preexec_fn, close_fds, shell,
cwd, env, universal_newlines, startupinfo, creationflags, restore_signals, start_new_session, pass_fds, user, group, extra_grou
ps, encoding, errors, text, umask)
    950
--> 951             self._execute_child(args, executable, preexec_fn, close_fds,
    952                                 pass_fds, cwd, env,

~\anaconda3\lib\subprocess.py in _execute_child(self, args, executable, preexec_fn, close_fds, pass_fds, cwd, env, startupinfo,
creationflags, shell, p2cread, p2cwrite, c2pread, c2pwrite, errread, errwrite, unused_restore_signals, unused_gid, unused_gids,
unused_uid, unused_umask, unused_start_new_session)
   1419             try:
-> 1420                 hp, ht, pid, tid = _winapi.CreateProcess(executable, args,
   1421                                          # no special security

FileNotFoundError: [WinError 2] The system cannot find the file specified

During handling of the above exception, another exception occurred:

FileNotFoundError                         Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_6664/1441242784.py in <module>
      7 export_graphviz(tree_small, out_file = 'small_tree.dot', feature_names = features_list, rounded = True, precision = 1)
      8 (graph, ) = pydot.graph_from_dot_file('small_tree.dot')
----> 9 graph.write_png('small_tree.png');

~\anaconda3\lib\site-packages\pydot.py in new_method(path, f, prog, encoding)
   1741                     encoding=None):
   1742                 """Refer to docstring of method `write.`"""
-> 1743                 self.write(
   1744                     path, format=f, prog=prog,
   1745                     encoding=encoding)

~\anaconda3\lib\site-packages\pydot.py in write(self, path, prog, format, encoding)
   1826                 f.write(s)
   1827         else:
-> 1828             s = self.create(prog, format, encoding=encoding)
   1829             with io.open(path, mode='wb') as f:
   1830                 f.write(s)

~\anaconda3\lib\site-packages\pydot.py in create(self, prog, format, encoding)
   1931                 args[1] = '"{prog}" not found in path.'.format(
   1932                     prog=prog)
-> 1933                 raise OSError(*args)
   1934             else:
   1935                 raise

FileNotFoundError: [WinError 2] "dot" not found in path.
```

In [161]:
```
# 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(features_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];
```

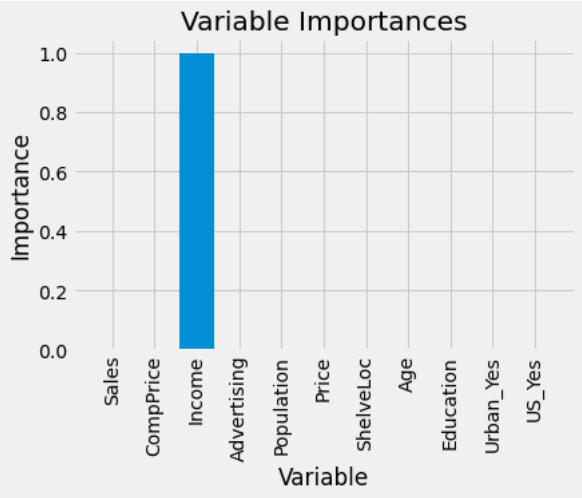
```
Variable: Income               Importance: 1.0
Variable: Sales                Importance: 0.0
Variable: CompPrice            Importance: 0.0
Variable: Advertising          Importance: 0.0
Variable: Population           Importance: 0.0
Variable: Price                Importance: 0.0
Variable: ShelveLoc            Importance: 0.0
Variable: Age                  Importance: 0.0
Variable: Education            Importance: 0.0
Variable: Urban_Yes            Importance: 0.0
Variable: US_Yes               Importance: 0.0
```

In [163]:
```
# 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 = [features_list.index('Sales'), features_list.index('Income')]
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), '%.')
```

```
Mean Absolute Error: 0.16 degrees.
Accuracy: 99.75 %.
```

In [165]:
```
# Import matplotlib for plotting and use magic command for Jupyter Notebooks
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, features_list, rotation='vertical')
# Axis labels and title
plt.ylabel('Importance'); plt.xlabel('Variable'); plt.title('Variable Importances')
```

Out[165]: Text(0.5, 1.0, 'Variable Importances')



In [ ]:
```
1
```

In [ ]:
```
1
```