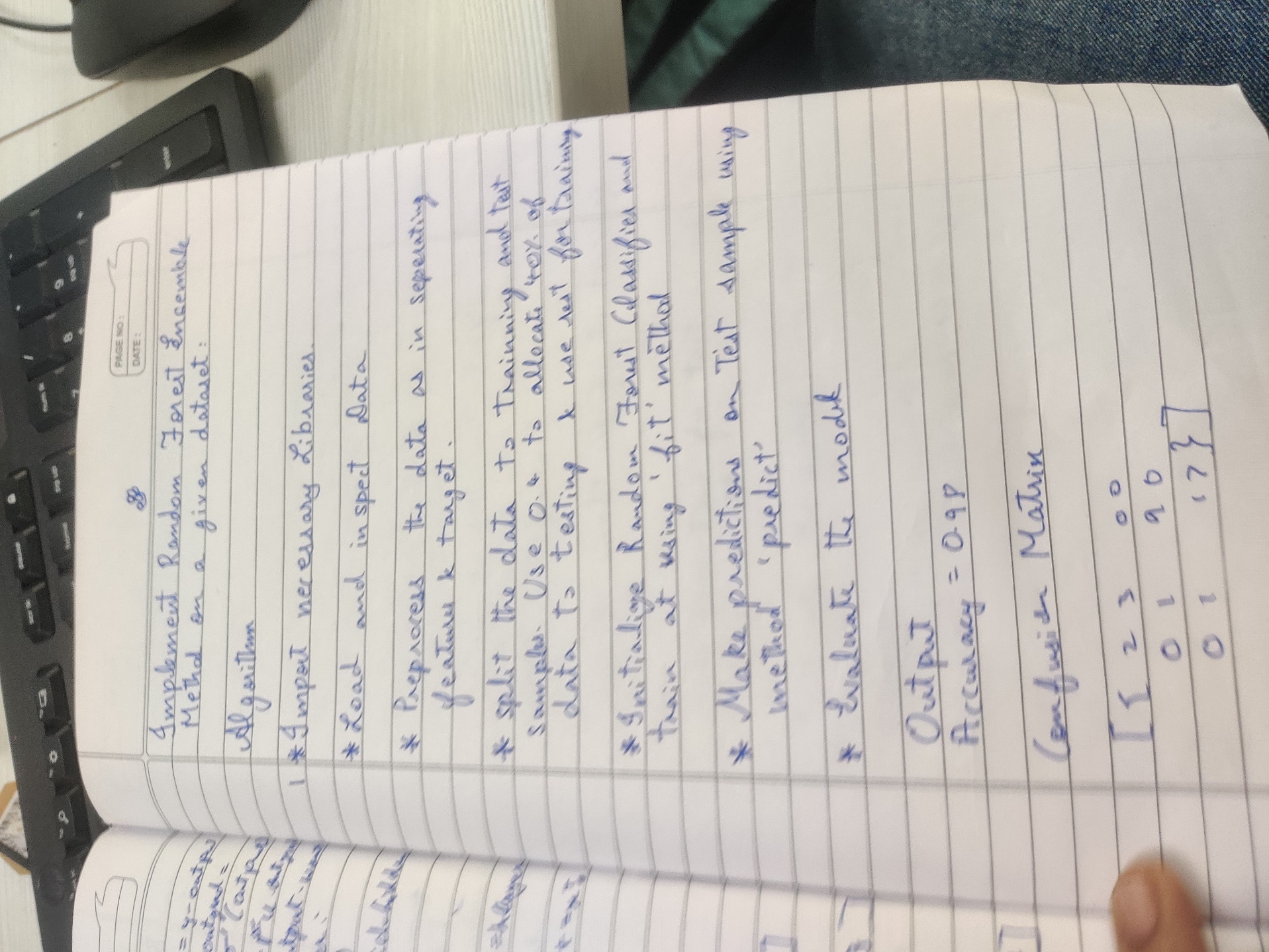
Implement Random forest ensemble method on a given dataset.



*Code:*

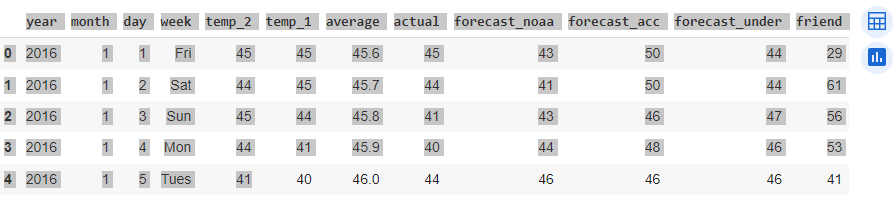
*# Pandas is used for data manipulation*

import pandas as pd

*# Read in data and display first 5 rows*

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

features.head(5)



Out[1]:

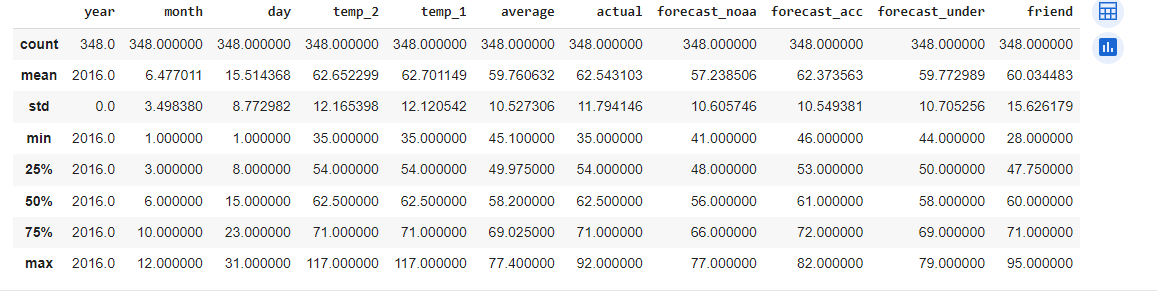
In [2]:

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

*# Descriptive statistics for each column*

features.describe()

The shape of our features is: (348, 12)

Out[3]:

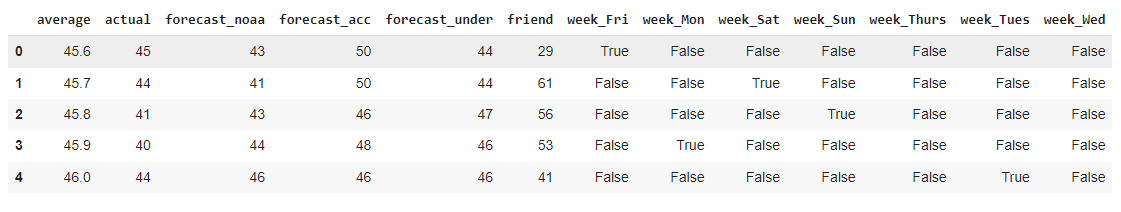
In [4]:

*# One-hot encode the data using pandas get\_dummies*

features = pd.get\_dummies(features)

*# Display the first 5 rows of the last 12 columns*

features.iloc[:,5:].head(5)



*# Use numpy to convert to arrays*

import numpy as np

*# Labels are the values we want to predict*

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

*# Remove the labels from the features*

*# axis 1 refers to the columns*

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

*# Saving feature names for later use*

feature\_list = list(features.columns)

*# Convert to numpy array*

features = np.array(features)

In [6]:

*# 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 [9]:

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: (261, 17)

Training Labels Shape: (261,)

Testing Features Shape: (87, 17)

Testing Labels Shape: (87,)

In [10]:

*# The baseline predictions are the historical averages*

baseline\_preds = test\_features[:, feature\_list.index('average')]

*# Baseline errors, and display average baseline error*

baseline\_errors = abs(baseline\_preds - test\_labels)

print('Average baseline error: ', round(np.mean(baseline\_errors), 2))

*# 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);

Average baseline error: 5.06

In [11]:

*# 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.')

*# 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), '%.')

Mean Absolute Error: 3.87 degrees.

Accuracy: 93.93 %.

In [12]:

*# 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 = 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 [13]:

*# 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 = feature\_list, rounded = True, precision = 1)

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

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

In [14]:

*# 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];

Variable: temp\_1 Importance: 0.66

Variable: average Importance: 0.15

Variable: forecast\_noaa Importance: 0.05

Variable: forecast\_acc Importance: 0.03

Variable: day Importance: 0.02

Variable: temp\_2 Importance: 0.02

Variable: forecast\_under Importance: 0.02

Variable: friend Importance: 0.02

Variable: month Importance: 0.01

Variable: year Importance: 0.0

Variable: week\_Fri Importance: 0.0

Variable: week\_Mon Importance: 0.0

Variable: week\_Sat Importance: 0.0

Variable: week\_Sun Importance: 0.0

Variable: week\_Thurs Importance: 0.0

Variable: week\_Tues Importance: 0.0

Variable: week\_Wed Importance: 0.0

In [16]:

*# 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('temp\_1'), feature\_list.index('average')]

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: 3.92 degrees.

Accuracy: 93.76 %.

In [17]:

*# 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*

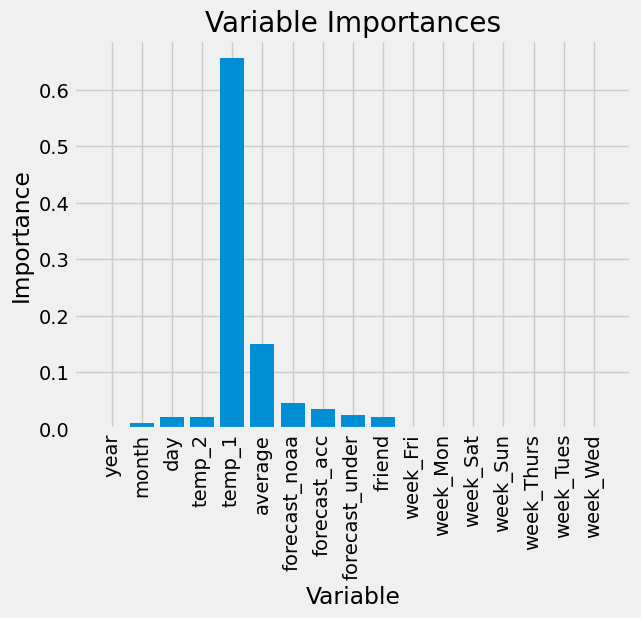
plt.bar(x\_values, importances, orientation = 'vertical')

*# Tick labels for x axis*

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

*# Axis labels and title*

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



In [18]:

*# Use datetime for creating date objects for plotting*

import datetime

*# Dates of training values*

months = features[:, feature\_list.index('month')]

days = features[:, feature\_list.index('day')]

years = features[:, feature\_list.index('year')]

*# List and then convert to datetime object*

dates = [str(int(year)) + '-' + str(int(month)) + '-' + str(int(day)) for year, month, day in zip(years, months, days)]

dates = [datetime.datetime.strptime(date, '%Y-%m-%d') for date in dates]

*# Dataframe with true values and dates*

true\_data = pd.DataFrame(data = {'date': dates, 'actual': labels})

*# Dates of predictions*

months = test\_features[:, feature\_list.index('month')]

days = test\_features[:, feature\_list.index('day')]

years = test\_features[:, feature\_list.index('year')]

*# Column of dates*

test\_dates = [str(int(year)) + '-' + str(int(month)) + '-' + str(int(day)) for year, month, day in zip(years, months, days)]

*# Convert to datetime objects*

test\_dates = [datetime.datetime.strptime(date, '%Y-%m-%d') for date in test\_dates]

*# Dataframe with predictions and dates*

predictions\_data = pd.DataFrame(data = {'date': test\_dates, 'prediction': predictions})

*# Plot the actual values*

plt.plot(true\_data['date'], true\_data['actual'], 'b-', label = 'actual')

*# Plot the predicted values*

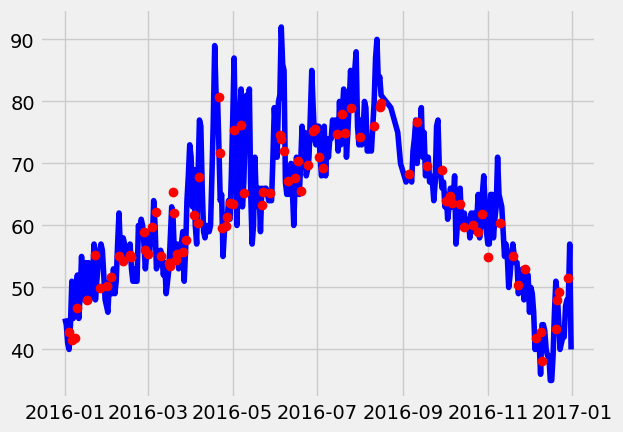
plt.plot(predictions\_data['date'], predictions\_data['prediction'], 'ro', label = 'prediction')

plt.xticks(rotation = '60');

plt.legend()

*# Graph labels*

plt.xlabel('Date'); plt.ylabel('Maximum Temperature (F)'); plt.title('Actual and Predicted Values');



In [19]:

*# Make the data accessible for plotting*

true\_data['temp\_1'] = features[:, feature\_list.index('temp\_1')]

true\_data['average'] = features[:, feature\_list.index('average')]

true\_data['friend'] = features[:, feature\_list.index('friend')]

*# Plot all the data as lines*

plt.plot(true\_data['date'], true\_data['actual'], 'b-', label = 'actual', alpha = 1.0)

plt.plot(true\_data['date'], true\_data['temp\_1'], 'y-', label = 'temp\_1', alpha = 1.0)

plt.plot(true\_data['date'], true\_data['average'], 'k-', label = 'average', alpha = 0.8)

plt.plot(true\_data['date'], true\_data['friend'], 'r-', label = 'friend', alpha = 0.3)

*# Formatting plot*

plt.legend(); plt.xticks(rotation = '60');

*# Lables and title*

plt.xlabel('Date'); plt.ylabel('Maximum Temperature (F)'); plt.title('Actual Max Temp and Variables');

