Import libraries

In [2]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.ensemble import RandomForestRegressor
```

In [3]:

```
df = pd.read_csv('temps.csv')
df.head()
```

Out[3]:

	year	month	day	week	temp_2	temp_1	average	actual	forecast_noaa	forecast_acc	forecast_under	friend
0	2016	1	1	Fri	45	45	45.6	45	43	50	44	29
1	2016	1	2	Sat	44	45	45.7	44	41	50	44	61
2	2016	1	3	Sun	45	44	45.8	41	43	46	47	56
3	2016	1	4	Mon	44	41	45.9	40	44	48	46	53
4	2016	1	5	Tues	41	40	46.0	44	46	46	46	41

In [4]:

```
df.shape
```

Out[4]:

```
(348, 12)
```

In [5]:
```
df.describe()
```

Out[5]:

	year	month	day	temp_2	temp_1	average	actual	forecast_noaa	forecast_acc	forecast_under	fri
count	348.0	348.000000	348.000000	348.000000	348.000000	348.000000	348.000000	348.000000	348.000000	348.000000	348.000
mean	2016.0	6.477011	15.514368	62.652299	62.701149	59.760632	62.543103	57.238506	62.373563	59.772989	60.034
std	0.0	3.498380	8.772982	12.165398	12.120542	10.527306	11.794146	10.605746	10.549381	10.705256	15.626
min	2016.0	1.000000	1.000000	35.000000	35.000000	45.100000	35.000000	41.000000	46.000000	44.000000	28.000
25%	2016.0	3.000000	8.000000	54.000000	54.000000	49.975000	54.000000	48.000000	53.000000	50.000000	47.750
50%	2016.0	6.000000	15.000000	62.500000	62.500000	58.200000	62.500000	56.000000	61.000000	58.000000	60.000
75%	2016.0	10.000000	23.000000	71.000000	71.000000	69.025000	71.000000	66.000000	72.000000	69.000000	71.000
max	2016.0	12.000000	31.000000	117.000000	117.000000	77.400000	92.000000	77.000000	82.000000	79.000000	95.000

In [6]:
```
# convert categorical to numerical features

df = pd.get_dummies(df)
df.head()
```

Out[6]:

	year	month	day	temp_2	temp_1	average	actual	forecast_noaa	forecast_acc	forecast_under	friend	week_Fri	week_Mon	week_Sat
0	2016	1	1	45	45	45.6	45	43	50	44	29	1	0	0
1	2016	1	2	44	45	45.7	44	41	50	44	61	0	0	1
2	2016	1	3	45	44	45.8	41	43	46	47	56	0	0	0
3	2016	1	4	44	41	45.9	40	44	48	46	53	0	1	0
4	2016	1	5	41	40	46.0	44	46	46	46	41	0	0	0

In [7]:
```
df.shape
```

Out[7]:
```
(348, 18)
```

```
In [8]: df.columns
```

```
Out[8]: Index(['year', 'month', 'day', 'temp_2', 'temp_1', 'average', 'actual',
               'forecast_noaa', 'forecast_acc', 'forecast_under', 'friend', 'week_Fri',
               'week_Mon', 'week_Sat', 'week_Sun', 'week_Thurs', 'week_Tues',
               'week_Wed'],
              dtype='object')
```

```
In [10]: df['day'].value_counts()
```

```
Out[10]: 16    12
         7     12
         12    12
         11    12
         10    12
         9     12
         8     12
         23    12
         6     12
         5     12
         4     12
         3     12
         28    12
         15    12
         14    11
         2     11
         13    11
         1     11
         17    11
         18    11
         19    11
         20    11
         21    11
         22    11
         24    11
         25    11
         26    11
         27    11
         30    10
         29    10
         31     6
         Name: day, dtype: int64
```

In [13]:

```
df['month'].value_counts()
```

Out[13]:

```
12    31
7     31
5     31
3     31
1     31
11    30
10    30
6     30
4     30
9     28
2     26
8     19
Name: month, dtype: int64
```

In [14]:

```
# drop 'year' features
df.drop(['year'],axis = 1,inplace = True)
df.head()
```

Out[14]:

	month	day	temp_2	temp_1	average	actual	forecast_noaa	forecast_acc	forecast_under	friend	week_Fri	week_Mon	week_Sat	weel
0	1	1	45	45	45.6	45	43	50	44	29	1	0	0	
1	1	2	44	45	45.7	44	41	50	44	61	0	0	1	
2	1	3	45	44	45.8	41	43	46	47	56	0	0	0	
3	1	4	44	41	45.9	40	44	48	46	53	0	1	0	
4	1	5	41	40	46.0	44	46	46	46	41	0	0	0	

In [18]:

```
df.shape
```

Out[18]:

```
(348, 17)
```

In [15]:

```
# seperate out features and target value from dataset

X = df.drop(['actual'],axis = 1).values
y = df['actual'].values
```

In [16]:

```
# split the data in training and testing set

X_train, X_test, y_train,y_test = train_test_split(X,y, test_size = 0.25, random_state = 42)
```

In [17]:

```
print("X_train shape : " , X_train.shape)
print("X_test shape : " , X_test.shape)
print("y_train shape : " , y_train.shape)
print("y_test shape : " , y_test.shape)
```

```
X_train shape :  (261, 16)
X_test shape :  (87, 16)
y_train shape :  (261,)
y_test shape :  (87,)
```

In [19]:

```
# RF model

rf = RandomForestRegressor(n_estimators=1000,random_state=42)

#fit model

rf.fit(X_train,y_train)
```

Out[19]:

```
RandomForestRegressor(n_estimators=1000, random_state=42)
```

In [20]:
```
# prediction
y_pred = rf.predict(X_test)
y_pred
```

Out[20]:
```
array([69.894, 61.311, 51.838, 61.331, 66.474, 70.284, 78.954, 75.945,
       62.044, 74.06 , 63.679, 72.146, 38.642, 62.558, 71.664, 55.993,
       60.951, 57.006, 56.676, 76.123, 63.684, 54.362, 66.548, 62.506,
       58.657, 53.029, 66.651, 46.469, 62.18 , 80.157, 73.759, 64.273,
       55.326, 82.128, 74.137, 61.627, 53.678, 51.405, 68.91 , 42.386,
       70.363, 57.358, 75.855, 42.474, 61.107, 73.991, 52.664, 81.469,
       53.237, 42.449, 46.478, 42.242, 64.18 , 65.781, 74.088, 61.41 ,
       55.166, 59.937, 54.497, 59.633, 65.539, 50.212, 60.757, 70.168,
       60.099, 59.281, 71.771, 69.866, 76.804, 41.387, 76.789, 56.868,
       60.416, 50.491, 54.489, 63.883, 43.877, 74.416, 47.341, 52.38 ,
       53.485, 68.207, 73.444, 72.496, 63.22 , 57.148, 45.948])
```

In [21]:
```
# calculate RMSE

rmse = np.sqrt(metrics.mean_squared_error(y_test,y_pred))
print(rmse)
```

```
5.091044648124332
```

In [ ]:
```

```

In [24]:

```
# merge predicted and actual value in one dataframe

y_pred_df = pd.DataFrame(y_pred)
y_pred_df['Actual'] = y_test
y_pred_df.columns = ['Predicted','Actual']
y_pred_df
```

Out[24]:

```
     Predicted  Actual
  0     69.894      66
  1     61.311      61
  2     51.838      52
  3     61.331      66
  4     66.474      70
...        ...     ...
 82     73.444      81
 83     72.496      67
 84     63.220      66
 85     57.148      57
 86     45.948      45

87 rows × 2 columns
```

In [25]:

```
# error

error = abs(y_pred-y_test)
```

In [26]:
```
error
```

Out[26]:
```
array([ 3.894,  0.311,  0.162,  4.669,  3.526, 11.716,  6.046,  8.055,
        2.956, 17.94 ,  2.679, 12.854,  5.358,  2.442,  2.664,  6.007,
        3.049,  1.006,  3.676,  2.877,  0.684,  2.638,  0.452,  0.506,
        0.343,  2.971,  1.651,  5.531,  3.82 ,  4.157,  9.759,  3.273,
        7.326,  1.128,  2.863,  4.627,  4.322,  4.405,  0.91 ,  8.614,
        6.637,  1.358,  2.855,  8.526,  2.107, 13.009,  5.336,  0.469,
        4.763,  0.449,  2.522,  2.242,  0.82 ,  1.781,  4.088,  3.59 ,
        2.166,  2.937,  1.497,  0.367,  1.461,  1.212,  7.757,  1.168,
        5.099,  8.719,  4.229,  0.866,  1.196,  5.387,  2.789,  0.132,
        8.584,  0.491,  1.511,  3.117,  4.123,  5.584,  0.659,  3.38 ,
        3.515,  8.793,  7.556,  5.496,  2.78 ,  0.148,  0.948])
```

In [27]:
```
# mean absolute error
mse = np.mean(error)
print("MSE : ",mse)
```

```
MSE :  3.8630574712643666
```

```
In [28]: # MAPE : mean absolute percentage error

         mape = 100*(error/y_test)
         mape
```

```
Out[28]: array([ 5.9       ,  0.50983607,  0.31153846,  7.07424242,  5.03714286,
               14.28780488,  7.11294118,  9.58928571,  4.54769231, 19.5       ,
                4.39180328, 15.12235294, 12.17727273,  3.75692308,  3.86086957,
                9.68870968,  4.7640625 ,  1.79642857,  6.93584906,  3.64177215,
                1.08571429,  4.62807018,  0.67462687,  0.81612903,  0.58135593,
                5.30535714,  2.54      , 10.63653846,  5.78787879,  5.46973684,
               15.2484375 ,  5.36557377, 15.2625    ,  1.39259259,  3.71818182,
                8.11754386,  7.45172414,  9.37234043,  1.33823529, 16.89019608,
                8.61948052,  2.425     ,  3.9109589 , 16.71764706,  3.57118644,
               14.95287356,  9.2       ,  0.57901235,  8.21206897,  1.06904762,
                5.14693878,  5.605     ,  1.26153846,  2.7828125 ,  5.84      ,
                5.52307692,  4.08679245,  5.15263158,  2.8245283 ,  0.61166667,
                2.18059701,  2.47346939, 14.63584906,  1.69275362,  9.27090909,
               12.82205882,  5.56447368,  1.25507246,  1.53333333, 14.96388889,
                3.76891892,  0.23157895, 12.44057971,  0.982     ,  2.69821429,
                4.65223881,  8.58958333,  6.98      ,  1.37291667,  6.89795918,
                6.16666667, 11.41948052,  9.32839506,  8.20298507,  4.21212121,
                0.25964912,  2.10666667])
```

```
In [29]: # accuracy

         acc = 100- np.mean(mape)
         print("Accuracy : ",acc)

         Accuracy :  93.94846113730775
```

```
In [ ]:
```

feature importance

In [58]:
```
importance = list(rf.feature_importances_)
print(importance)
```

```
[0.010322601403532283, 0.02111366527196822, 0.021110269148599918, 0.6555698213907486, 0.149480391207536
28, 0.04601374758544169, 0.03517063342900543, 0.02318439434529083, 0.02054749470049068, 0.0034748931143
383653, 0.00252835863556795, 0.003593012624168681, 0.0022740438819965005, 0.0012834507255272789, 0.0023
2653540191857, 0.0020066871338688596]
```

In [59]:
```
df.columns
```

Out[59]:
```
Index(['month', 'day', 'temp_2', 'temp_1', 'average', 'actual',
       'forecast_noaa', 'forecast_acc', 'forecast_under', 'friend', 'week_Fri',
       'week_Mon', 'week_Sat', 'week_Sun', 'week_Thurs', 'week_Tues',
       'week_Wed'],
      dtype='object')
```

In [60]:
```
features = ['month', 'day', 'temp_2', 'temp_1', 'average',
       'forecast_noaa', 'forecast_acc', 'forecast_under', 'friend', 'week_Fri',
       'week_Mon', 'week_Sat', 'week_Sun', 'week_Thurs', 'week_Tues',
       'week_Wed']
```

In [61]:
```
feature_importance = [(feature, round(importance, 2)) for feature, importance in zip(features, importanc
e)]
```

```
In [62]: feature_importance
```

```
Out[62]: [('month', 0.01),
          ('day', 0.02),
          ('temp_2', 0.02),
          ('temp_1', 0.66),
          ('average', 0.15),
          ('forecast_noaa', 0.05),
          ('forecast_acc', 0.04),
          ('forecast_under', 0.02),
          ('friend', 0.02),
          ('week_Fri', 0.0),
          ('week_Mon', 0.0),
          ('week_Sat', 0.0),
          ('week_Sun', 0.0),
          ('week_Thurs', 0.0),
          ('week_Tues', 0.0),
          ('week_Wed', 0.0)]
```

```
In [64]: feature_importance[0][1]
```

```
Out[64]: 0.01
```

In [66]:

```
# sorting

feature_importance_sorted = sorted(feature_importance, key = lambda x: x[1],reverse = True)
feature_importance_sorted
```

Out[66]:

```
[('temp_1', 0.66),
 ('average', 0.15),
 ('forecast_noaa', 0.05),
 ('forecast_acc', 0.04),
 ('day', 0.02),
 ('temp_2', 0.02),
 ('forecast_under', 0.02),
 ('friend', 0.02),
 ('month', 0.01),
 ('week_Fri', 0.0),
 ('week_Mon', 0.0),
 ('week_Sat', 0.0),
 ('week_Sun', 0.0),
 ('week_Thurs', 0.0),
 ('week_Tues', 0.0),
 ('week_Wed', 0.0)]
```

```
In [70]: x_value
```

```
Out[70]: ['month',
          'day',
          'temp_2',
          'temp_1',
          'average',
          'forecast_noaa',
          'forecast_acc',
          'forecast_under',
          'friend',
          'week_Fri',
          'week_Mon',
          'week_Sat',
          'week_Sun',
          'week_Thurs',
          'week_Tues',
          'week_Wed']
```

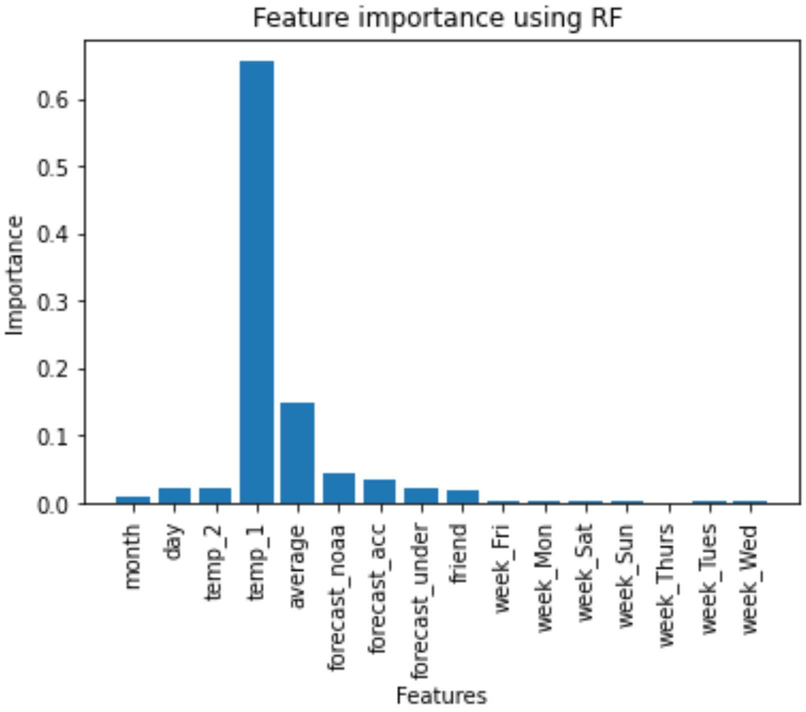
```
In [71]: y
```

```
Out[71]: [0.010322601403532283,
          0.02111366527196822,
          0.021110269148599918,
          0.6555698213907486,
          0.14948039120753628,
          0.04601374758544169,
          0.03517063342900543,
          0.02318439434529083,
          0.02054749470049068,
          0.0034748931143383653,
          0.00252835863556795,
          0.0035930126241686681,
          0.0022740438819965005,
          0.0012834507255272789,
          0.00232653540191857,
          0.0020066871338688596]
```

In [69]:

```
x_value = features
y = importance

plt.bar(x_value,y)
plt.xticks(x_value,rotation = 'vertical')
plt.title("Feature importance using RF ")
plt.xlabel("Features")
plt.ylabel("Importance")
plt.show()
```



In [72]:

```
# we can eleminate the features with least importance and can
# rebuild model again considering importance features
```

In [ ]: