1. Pregunta Problema ? **Cual va a ser la temperatura máxima mañana en Seattle ?**

```
In [8]: df.head()
Out[8]:
   year  month  day  week  ...  forecast_noaa  forecast_acc  forecast_under  friend
0  2016      1    1   Fri  ...             43            50              44      29
1  2016      1    2   Sat  ...             41            50              44      61
2  2016      1    3   Sun  ...             43            46              47      56
3  2016      1    4   Mon  ...             44            48              46      53
4  2016      1    5  Tues  ...             46            46              46      41

[5 rows x 12 columns]
```

```
In [10]: df.tail()
Out[10]:
     year  month  day  ... forecast_acc  forecast_under  friend
343  2016     12   27  ...            50              47      47
344  2016     12   28  ...            49              44      58
345  2016     12   29  ...            50              45      65
346  2016     12   30  ...            46              44      42
347  2016     12   31  ...            48              47      57

[5 rows x 12 columns]
```

Se va a predecir la temperatura del 1 de Enero de 2017

2. Qué tipo de datos se necesitan ? **Se necesita una base de datos histórica de temperaturas en Seattle, la cual se obtuvo de la pagina de la NOAA. NOAA = National Oceanic and Atmospheric Administration**

3. Adquirir los datos en un formato accesible. **Los datos se obtuvieron en formato *.csv y fueron cargados a un DataFrame (nombre df). Las variables (columnas en DataFrame) son:**

year año u años de la medición de temperatura
month numero del mes de la medición de temperatura
day número del día de la medición de temperatura
Week nombre del día de la semana, de la medición de temperatura como un string.
temp_2 Temperatura máxima dos días antes de la fecha
temp_1 Temperatura máxima un día antes de la fecha
average Promedio histórico de temperatura máxima
actual Máxima temperatura medida en el día
friend Variable para suavización exponencial (parámetro Alpha) si Alpha cercano a cero, hay prioridad por datos antiguos. Si Alpha cercano a uno, hay prioridad por datos nuevos.

4. Identificar y corregir puntos de datos faltantes / anomalías según sea necesario

year para determinar cuántos años hay en la base de datos se usa:
df.year.unique()

```
In [7]: df.year.unique()
Out[7]: array([2016], dtype=int64)
```

Únicamente hay datos del año 2016.

month numero del mes

```
In [13]: df.month.unique() # determina cuantos años hay en database
Out[13]: array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12], dtype=int64)
```

Se observa que los valores van del 1 al 12, lo cual está bien y muestra que no hay datos anómalos para esta variable

day número del día

```
In [15]: df.day.unique() # determina cuantos años hay en database
Out[15]:
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
       18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31],
      dtype=int64)

In [16]: len(df.day.unique())
Out[16]: 31
```

Se observa que los valores van del 1 al 31, lo cual está bien y muestra que no hay datos anómalos para esta variable

Week nombre del día de la semana, como un string

```
In [17]: df.week.unique()   # determina nombres de los dia de la semana en database
Out[17]: array(['Fri', 'Sat', 'Sun', 'Mon', 'Tues', 'Wed', 'Thurs'], dtype=object)

In [18]: len(df.week.unique())
Out[18]: 7
```

Se observa que los valores van del lunes a domingo, lo cual está bien y muestra que no hay datos anómalos para esta variable

temp_2 Temperatura máxima dos días antes de la fecha

```
In [19]: df['temp_2'].isnull().sum()
Out[19]: 0
```

No hay datos nulos para temp_2

temp_1: Temperatura máxima un día antes de la fecha

```
In [21]: df['temp_1'].isnull().sum()
Out[21]: 0
```

No hay datos nulos para temp_1

average

```
In [22]: df['average'].isnull().sum()
Out[22]: 0
```

No hay datos nulos para average

actual

```
In [23]: df['actual'].isnull().sum()
Out[23]: 0
```

No hay datos nulos para actual

friend

```
In [24]: df['friend'].isnull().sum()
Out[24]: 0
```

No hay datos nulos para friend

Se ejecuta instrucción describe, así:

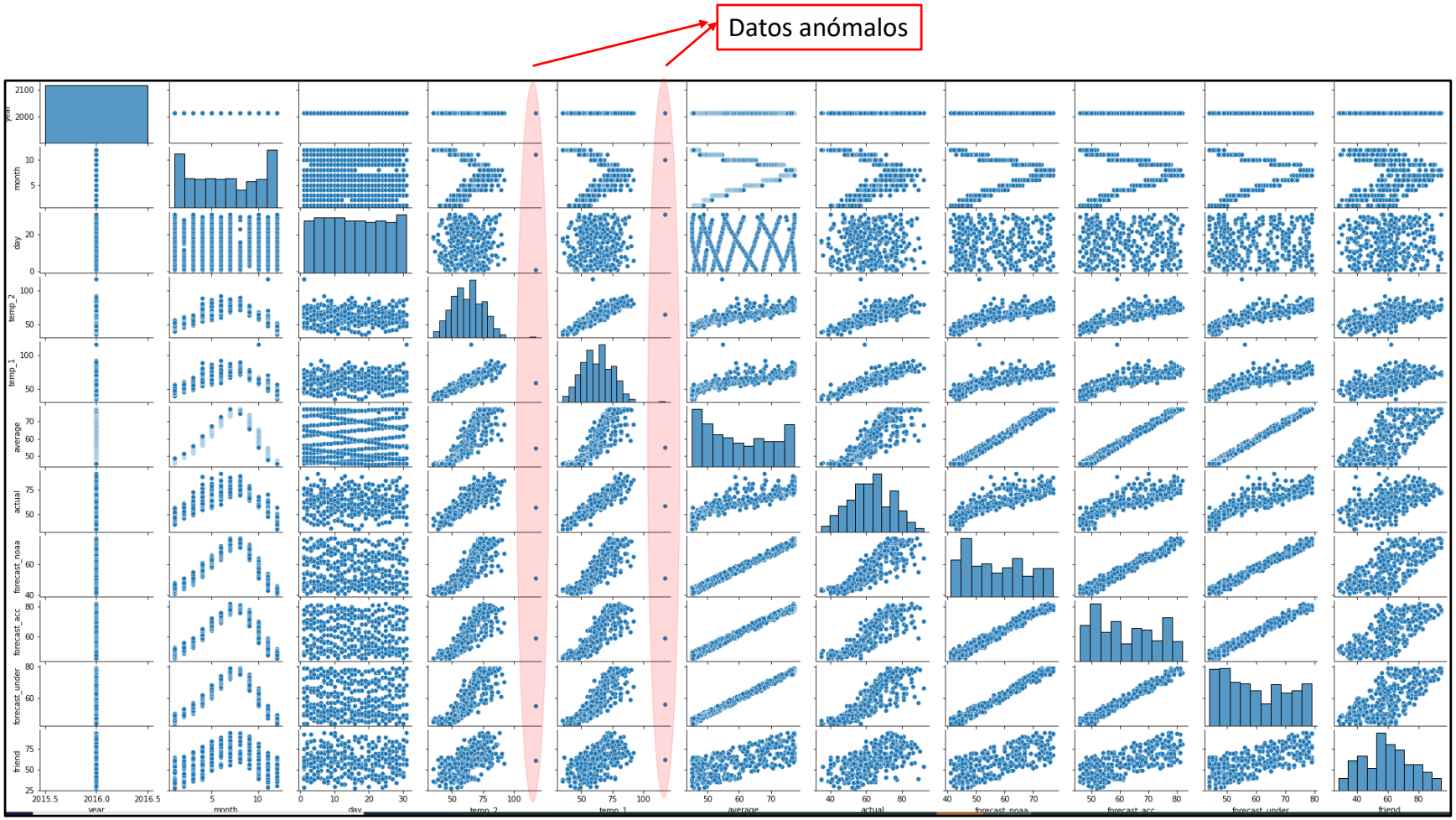
```
df.describe()
df.describe().to_excel("describe.xlsx")
```

Obteniendo:

	year	month	day	temp_2	temp_1	average	actual	forecast_noaa	forecast_acc	forecast_under	friend
count	348.00	348.00	348.00	348.00	348.00	348.00	348.00	348.00	348.00	348.00	348.00
mean	2016.00	6.48	15.51	62.65	62.70	59.76	62.54	57.24	62.37	59.77	60.03
std	0.00	3.50	8.77	12.17	12.12	10.53	11.79	10.61	10.55	10.71	15.63
min	2016.00	1.00	1.00	35.00	35.00	45.10	35.00	41.00	46.00	44.00	28.00
25%	2016.00	3.00	8.00	54.00	54.00	49.98	54.00	48.00	53.00	50.00	47.75
50%	2016.00	6.00	15.00	62.50	62.50	58.20	62.50	56.00	61.00	58.00	60.00
75%	2016.00	10.00	23.00	71.00	71.00	69.03	71.00	66.00	72.00	69.00	71.00
max	2016.00	12.00	31.00	117.00	117.00	77.40	92.00	77.00	82.00	79.00	95.00

Parece haber un **valor anómalo** máximo de 117, para temp_2 y temp_1, el cual no coincide con los máximos de **average y actual**

Se ejecuta keyword **sns.pairplot(df)** con los siguientes resultados:


Datos anómalos
year
month
day
temp_2
temp_1
average
actual
forecast_noaa
forecast_acc
forecast_under
friend

Se corre instrucción:

```
df.loc[(df["temp_2"] == 117) ]
```

con el siguiente resultado

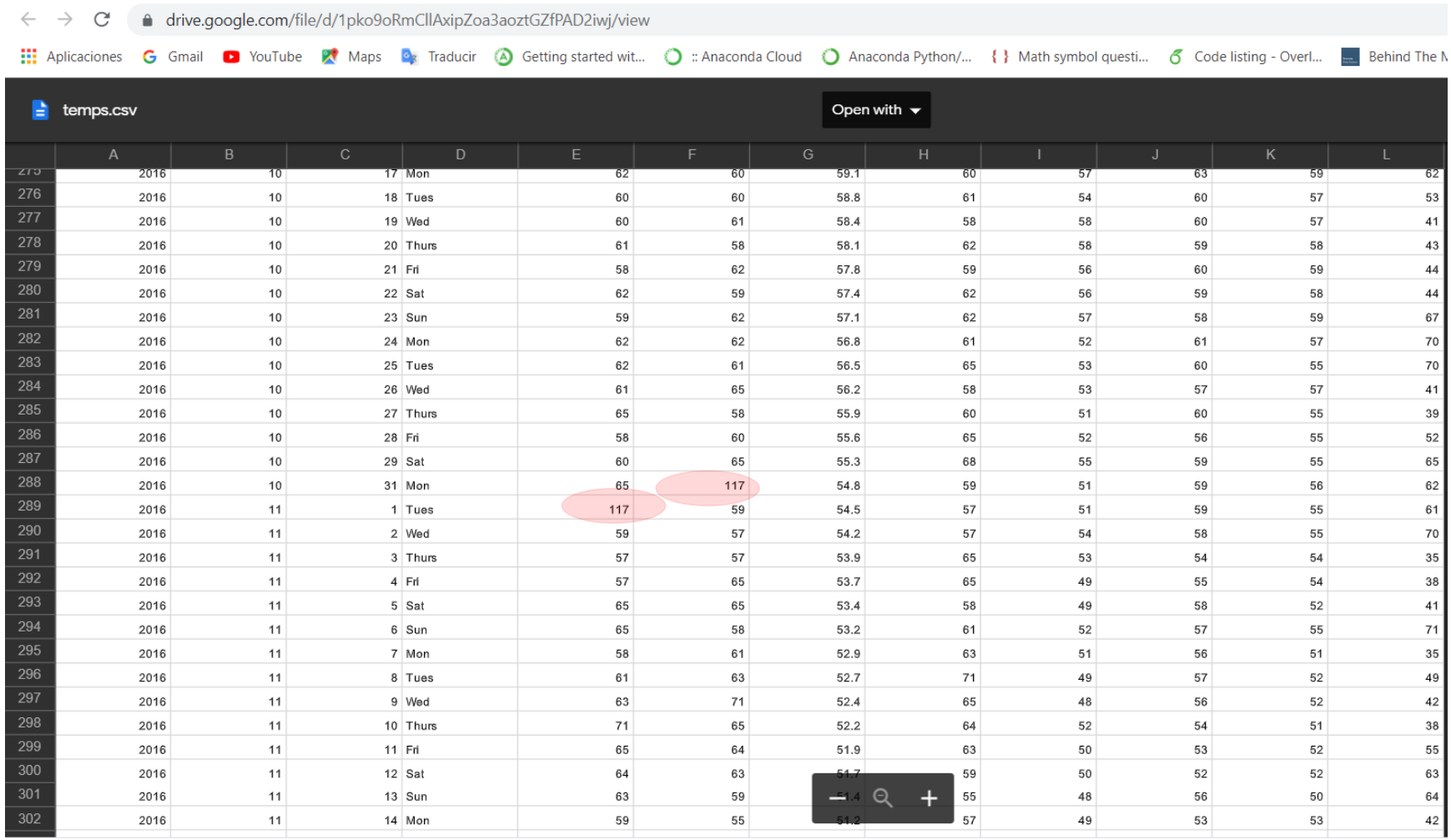
```
In [27]: df.loc[(df["temp_2"] == 117) ]
Out[27]:
     year  month  day  ... forecast_acc  forecast_under  friend
287  2016     11    1  ...           59              55      61

[1 rows x 12 columns]
```

El dato anómalo corresponde al 1 de noviembre de 2016, se hace lo mismo con temp_1

```
In [28]: df.loc[(df["temp_1"] == 117) ]
Out[28]:
     year  month  day week  ...  forecast_noaa  forecast_acc  forecast_under  friend
286  2016     10   31  Mon  ...             51            59              56      62

[1 rows x 12 columns]
```

Donde el dato anómalo corresponde a 31 de octubre de 2016

Se verifica el dato anómalo en la fuente original (link dado en clase)
https://drive.google.com/file/d/1pko9oRmCllAxipZoa3aoztGZfPAD2iwj/view



	A	B	C	D	E	F	G	H	I	J	K	L
275	2016	10	17	Mon	62	60	59.1	60	57	63	59	62
276	2016	10	18	Tues	60	60	58.8	61	54	60	57	53
277	2016	10	19	Wed	60	61	58.4	58	58	60	57	41
278	2016	10	20	Thurs	61	58	58.1	62	58	59	58	43
279	2016	10	21	Fri	58	62	57.8	59	56	60	59	44
280	2016	10	22	Sat	62	59	57.4	62	56	59	58	44
281	2016	10	23	Sun	59	62	57.1	62	57	58	59	67
282	2016	10	24	Mon	62	62	56.8	61	52	61	57	70
283	2016	10	25	Tues	62	61	56.5	65	53	60	55	70
284	2016	10	26	Wed	61	65	56.2	58	53	57	57	41
285	2016	10	27	Thurs	65	58	55.9	60	51	60	55	39
286	2016	10	28	Fri	58	60	55.6	65	52	56	55	52
287	2016	10	29	Sat	60	65	55.3	68	55	59	55	65
288	2016	10	31	Mon	65	117	54.8	59	51	59	56	62
289	2016	11	1	Tues	117	59	54.5	57	51	59	55	61
290	2016	11	2	Wed	59	57	54.2	57	54	58	55	70
291	2016	11	3	Thurs	57	57	53.9	65	53	54	54	35
292	2016	11	4	Fri	57	65	53.7	65	49	55	54	38
293	2016	11	5	Sat	65	65	53.4	58	49	58	52	41
294	2016	11	6	Sun	65	58	53.2	61	52	57	55	71
295	2016	11	7	Mon	58	61	52.9	63	51	56	51	35
296	2016	11	8	Tues	61	63	52.7	71	49	57	52	49
297	2016	11	9	Wed	63	71	52.4	65	48	56	52	42
298	2016	11	10	Thurs	71	65	52.2	64	52	54	51	38
299	2016	11	11	Fri	65	64	51.9	63	50	53	52	55
300	2016	11	12	Sat	64	63	51.7	59	50	52	52	63
301	2016	11	13	Sun	63	59	51.4	55	48	56	50	64
302	2016	11	14	Mon	59	55	51.2	57	49	53	53	42

Se toma la decisión de reemplazar los valores anómalos máximos de temp_1 y temp_2 con el máximo de actual = 92, de acuerdo a la tabla de describe

df.loc[(df.temp_2 == 117),'temp_2']= 92
df.loc[(df.temp_1 == 117),'temp_1']= 92

df.describe().to_excel("describe2.xlsx")

	year	month	day	temp_2	temp_1	average	actual	recast_no	recast_ac	recast_und	friend
count	348.00	348.00	348.00	348.00	348.00	348.00	348.00	348.00	348.00	348.00	348.00
mean	2016.00	6.48	15.51	62.58	62.63	59.76	62.54	57.24	62.37	59.77	60.03
std	0.00	3.50	8.77	11.91	11.87	10.53	11.79	10.61	10.55	10.71	15.63
min	2016.00	1.00	1.00	35.00	35.00	45.10	35.00	41.00	46.00	44.00	28.00
25%	2016.00	3.00	8.00	54.00	54.00	49.98	54.00	48.00	53.00	50.00	47.75
50%	2016.00	6.00	15.00	62.50	62.50	58.20	62.50	56.00	61.00	58.00	60.00
75%	2016.00	10.00	23.00	71.00	71.00	69.03	71.00	66.00	72.00	69.00	71.00
max	2016.00	12.00	31.00	92.00	92.00	77.40	92.00	77.00	82.00	79.00	95.00

Ahora hay 348 registros versus 366 que debería haber por ser año 2016. Comparando los datos de Temp_2 (2 días antes), Temp_1 (1 día antes) y actual vemos que la información es muy similar, por lo que se puede asumir que no es necesario llenar 18 registros faltantes.

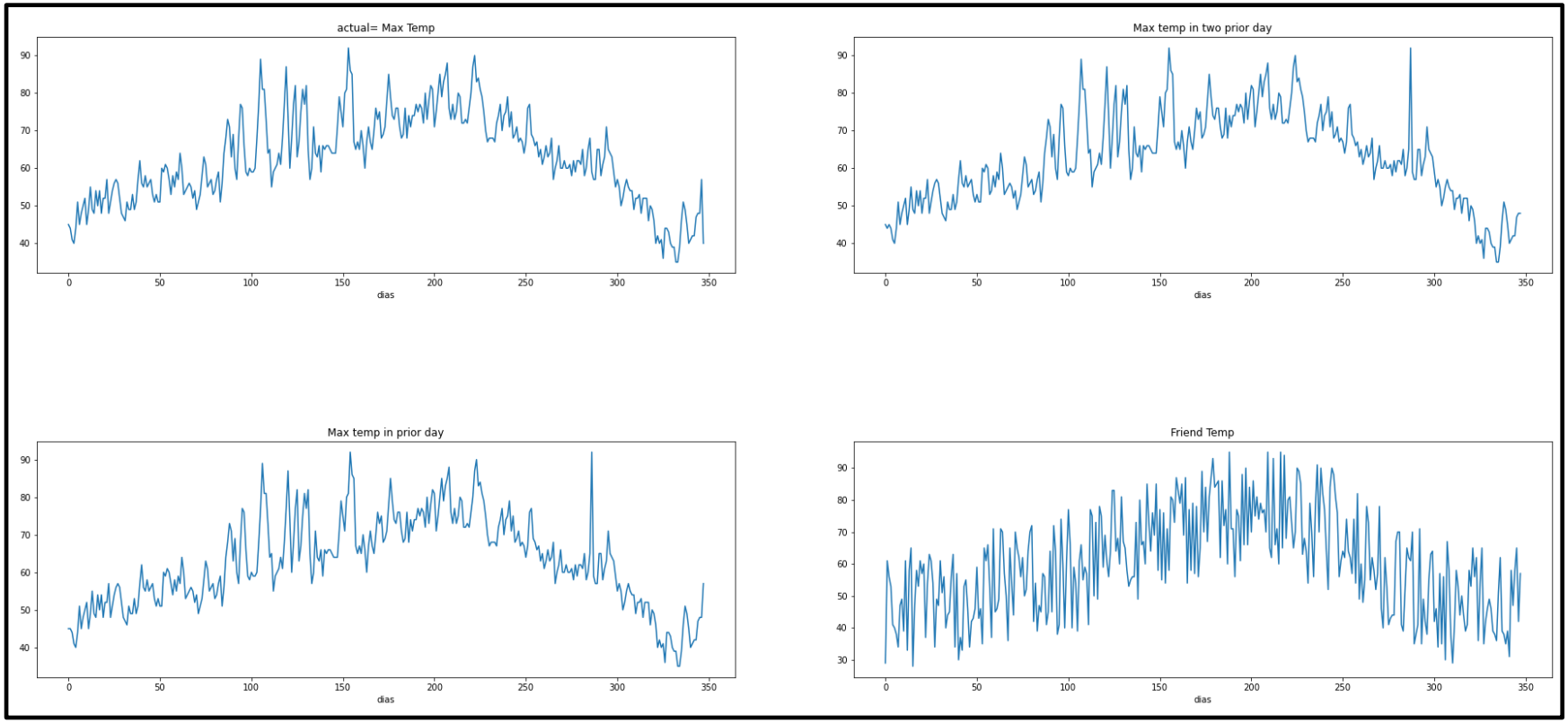


Grafico de las 4 variables de temperatura por día

Los gráficos de las 4 variables tienen la misma tendencia, por lo cual la base de datos se ve razonable y se puede continuar.

5. Prepare los datos para el modelo de aprendizaje automático.

   La variable week es categórica, por lo que para que se pueda usar en modelos de regresión debe ser transformada a numérica usando get dummies.

```
#%%
df2 = df.copy()
df2= pd.get_dummies(df2)
```

   # borrando las columnas:

   # forecast_noaa
   # forecast_acc
   # forecast_under
   #
   # display de las 5 primeras filas y las ultimas doce columnas
   # para verificar le resultado del get-dummies

   df2 = df2.drop(['forecast_noaa', 'forecast_acc', 'forecast_under'], axis=1)
   df2.iloc[:,5:].head(5).to_excel("dummies.xlsx")

	average	actual	friend	week_Fri	week_Mon	week_Sat	week_Sun	week_Thurs	week_Tues	week_Wed
0	45.6	45	29	1	0	0	0	0	0	0
1	45.7	44	61	0	0	1	0	0	0	0
2	45.8	41	56	0	0	0	1	0	0	0
3	45.9	40	53	0	1	0	0	0	0	0
4	46	44	41	0	0	0	0	0	1	0

   Adicionalmente se definió la variable objetivo o dependiente (y) como array: **y = np.array(df2['actual'])**
   Se removió variable objetivo del df2. **df2 = df2.drop('actual', axis = 1)**

Se creó lista con nombre de variables independientes. **x_list = list(df2.columns)**
Se convirtió df a array de variables independientes. **df2 = np.array(df2)**
Se dividió el conjunto de datos en dos: training y testing (65 y 35% respectivamente)

from sklearn.model_selection import train_test_split
train_x, test_x, train_y, test_y = train_test_split(df2, y, test_size = 0.35, random_state = 50)

6. Modelo Base

La línea base de predicciones son los promedios históricos x día 'average'
linea_base = test_x[:, x_list.index('average')]

error = average - y (test). y=actual
error_linea_base = abs(linea_base - test_y)

print('Error_linea_base promedio: ', round(np.mean(error_linea_base), 3), 'grados.')
 # Error_linea_base promedio: 4.905 grados

7. Entendiendo el modelo

from sklearn.ensemble import RandomForestRegressor
regressor = RandomForestRegressor(n_estimators = 1000, random_state = 50)
regressor.fit(train_x, train_y)
print(regressor.score(train_x, train_y))

```
In [70]: print(regressor.score(train_x, train_y))
0.9769685564308977
```

8. Hacer predicciones sobre los datos de prueba

```
# predicción con los datos de testing
forecast = regressor.predict(test_x)

# Calculo del valor absoluto del error
errors = abs(forecast - test_y)

# Error Promedio
print('Mean Absolute Error:', round(np.mean(errors), 3), 'grados.')
```

```
In [72]: print('Mean Absolute Error:', round(np.mean(errors), 3), 'grados.')
Mean Absolute Error: 3.575 grados.
```

El error promedio de las predicciones de 3.575 grados es menor al error base usando los promedios días de 4.90 grados, lo cual es bueno.

9. Métrica de rendimiento

```
# Calculo de vector de errores relativos del valor absoluto del error
error_mean = 100 * (errors / test_y)
# Calculo de la precisión del modelo
accuracy = 100 - np.mean(error_mean)
print('Accuracy:', round(accuracy, 3), '%.')
```

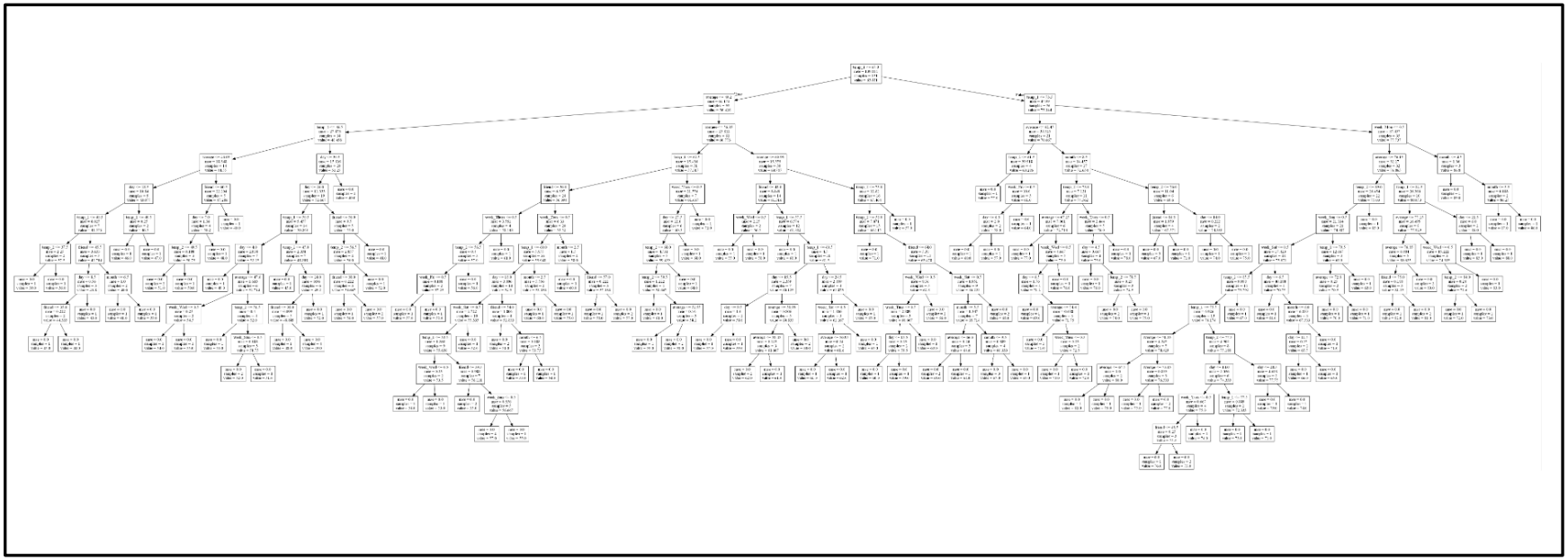
```
In [75]: print('Accuracy:', round(accuracy, 3), '%.')
Accuracy: 94.334 %.
```

La temperatura del 1 de enero de 2017 se puede calcular con una precisión del 94.3%

10. Análisis de sensibilidad

Se generó todo el árbol generado para el ajuste de datos. La siguiente grafica lo muestra.

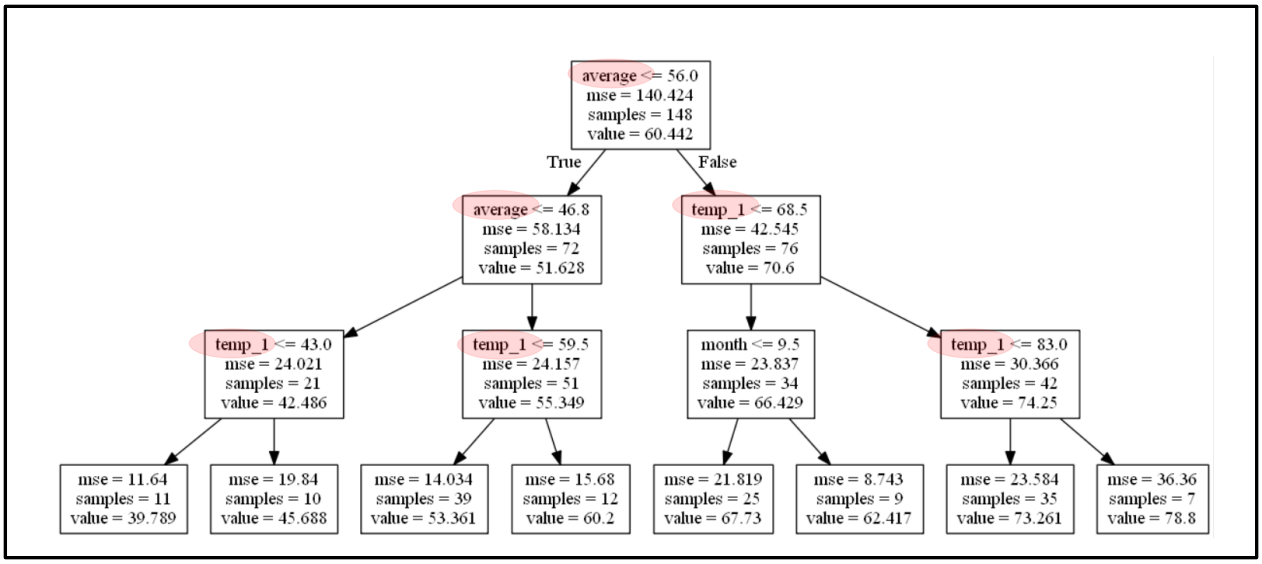
```
# conda install python-graphviz
import pydotplus
# Importando librerías para visualización
from sklearn.tree import export_graphviz

arbol = regressor.estimators_[15]
# definiendo ruta para graphviz
os.environ['PATH'] = os.environ['PATH']+';' +
r'C:\Users\lpren\anaconda3\Library\bin\graphviz'

# Exportando archivo *.dot
export_graphviz(arbol, out_file="tree2" + ".dot", feature_names =  x_list)
# Generando archivo *.dot
dot_data = export_graphviz(arbol, out_file=None, feature_names = x_list)
graph = pydotplus.graphviz.graph_from_dot_data(dot_data)
# Generando archivo *.png para visualizar todo el arbol
graph.write_png("tree2" + "_gv.png")
```



Generamos una poda para extraer árbol mas pequeño y determinar las variables de importancia.

Limitando profundidad del arbol a 3 niveles
regressor_small = RandomForestRegressor(n_estimators=10, max_depth = 3)
regressor_small.fit(train_x, train_y)
Extrayendo el arbol pequeño
arbol_small = regressor_small.estimators_[5]

# Exportando archivo *.dot
export_graphviz(arbol, out_file="small-tree" + ".dot", feature_names = x_list)
# Generando *.dot
dot_data = export_graphviz(arbol_small, out_file=None, feature_names = x_list)
graph = pydotplus.graphviz.graph_from_dot_data(dot_data)
# Generando *.png
graph.write_png("small-tree" + "_gv.png")



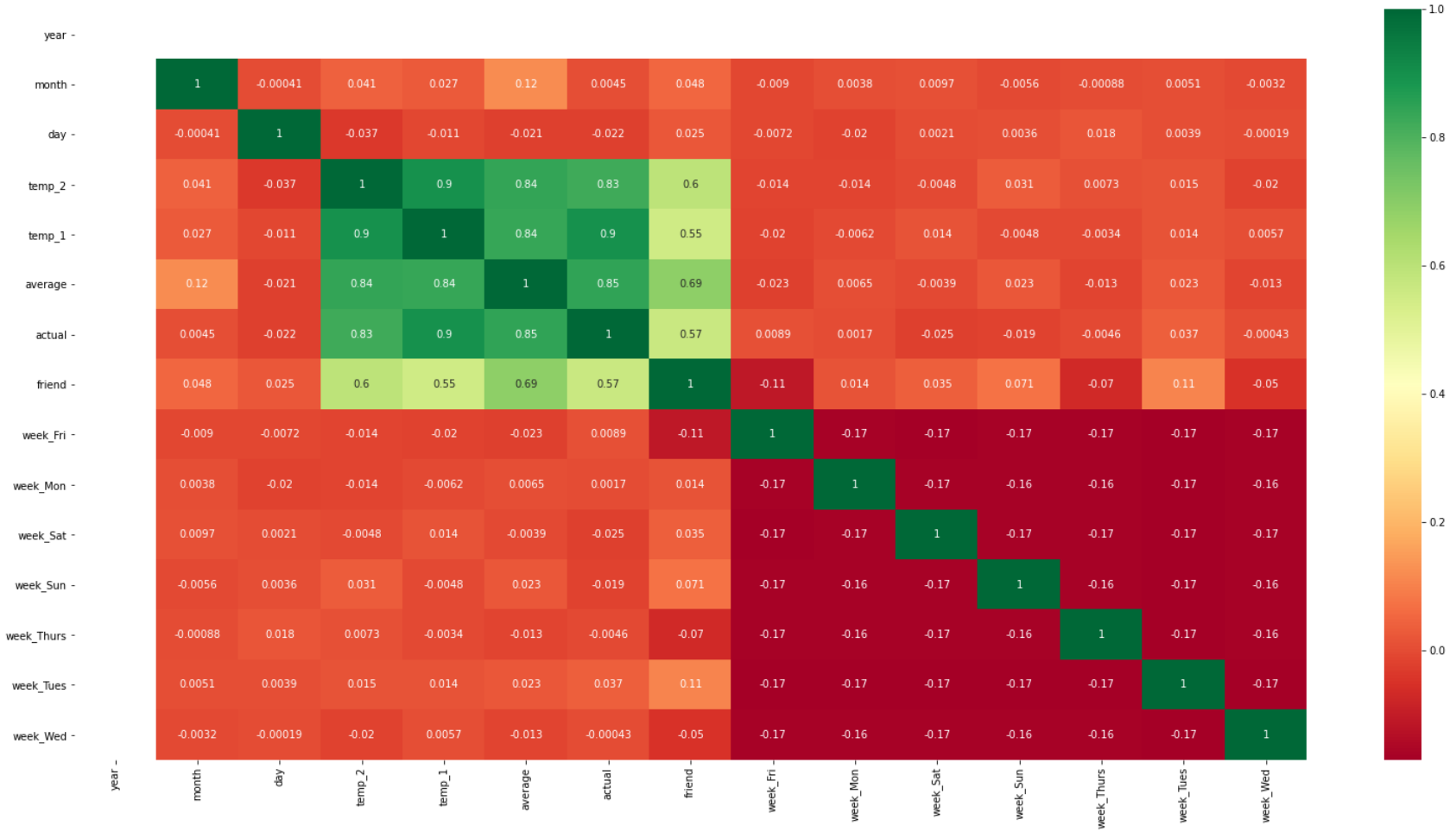
De acuerdo al árbol menor las variables de importancia sugeridas son: average y temp_1

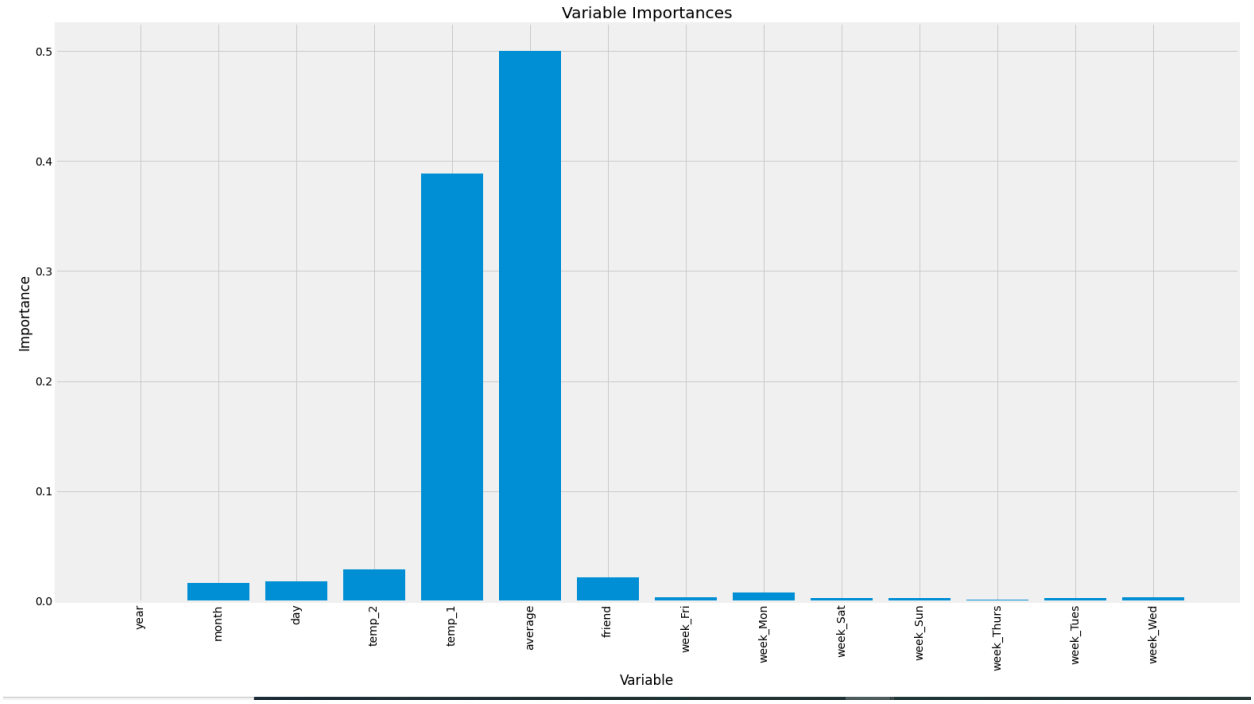
obtención de variables independientes más importantes /influyentes en el modelo

```
importances = list(regressor.feature_importances_)
# Listado de una tupla que contiene Var. Ind y su grado de importancia
feature_importances = [(feature, round(importance, 2)) \
                       for feature, importance in zip(x_list, importances)]
# ordenamiento de variables de acuerdo a importancia
feature_importances = sorted(feature_importances, key = lambda x: x[1], reverse = True)
# imprimiendo Var. Ind. y su importancia
[print('Variable: {:20} Importance: {}'.format(*pair)) \
 for pair in feature_importances];
```

```
Variable: average              Importance: 0.5
Variable: temp_1               Importance: 0.39
Variable: temp_2               Importance: 0.03
Variable: month                Importance: 0.02
Variable: day                  Importance: 0.02
Variable: friend               Importance: 0.02
Variable: week_Mon             Importance: 0.01
Variable: year                 Importance: 0.0
Variable: week_Fri             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
```

Se comprueba que average y temp_1 son las variables que mas influyen en actual. La matriz de correlación (mapa de color) muestra también que la variable actual correlación mejor con temp_1 y average


year
month
day
temp_2
temp_1
average
actual
friend
week_Fri
week_Mon
week_Sat
week_Sun
week_Thurs
week_Tues
week_Wed
1 -0.00041 0.041 0.027 0.12 0.0045 0.048 -0.009 0.0038 0.0097 -0.0056 -0.00088 0.0051 -0.0032
-0.00041 1 -0.037 -0.011 -0.021 -0.022 0.025 -0.0072 -0.02 0.0021 0.0036 0.018 0.0039 -0.00019
0.041 -0.037 1 0.9 0.84 0.83 0.6 -0.014 -0.014 -0.0048 0.031 0.0073 0.015 -0.02
0.027 -0.011 0.9 1 0.84 0.9 0.55 -0.02 -0.0062 0.014 -0.0048 -0.0034 0.014 0.0057
0.12 -0.021 0.84 0.84 1 0.85 0.69 -0.023 0.0065 -0.0039 0.023 -0.013 0.023 -0.013
0.0045 -0.022 0.83 0.9 0.85 1 0.57 0.0089 0.0017 -0.025 -0.019 -0.0046 0.037 -0.00043
0.048 0.025 0.6 0.55 0.69 0.57 1 -0.11 0.014 0.035 0.071 -0.07 0.11 -0.05
-0.009 -0.0072 -0.014 -0.02 -0.023 0.0089 -0.11 1 -0.17 -0.17 -0.17 -0.17 -0.17 -0.17
0.0038 -0.02 -0.014 -0.0062 0.0065 0.0017 0.014 -0.17 1 -0.17 -0.16 -0.16 -0.17 -0.16
0.0097 0.0021 -0.0048 0.014 -0.0039 -0.025 0.035 -0.17 -0.17 1 -0.17 -0.17 -0.17 -0.17
-0.0056 0.0036 0.031 -0.0048 0.023 -0.019 0.071 -0.17 -0.16 -0.17 1 -0.16 -0.17 -0.16
-0.00088 0.018 0.0073 -0.0034 -0.013 -0.0046 -0.07 -0.17 -0.16 -0.17 -0.16 1 -0.17 -0.16
0.0051 0.0039 0.015 0.014 0.023 0.037 0.11 -0.17 -0.17 -0.17 -0.17 -0.17 1 -0.17
-0.0032 -0.00019 -0.02 0.0057 -0.013 -0.00043 -0.05 -0.17 -0.16 -0.17 -0.16 -0.16 -0.17 1
1.0
0.8
0.6
0.4
0.2
0.0



Haciendo análisis de sensibilidad, corriendo el modelo únicamente con temp_1 y average

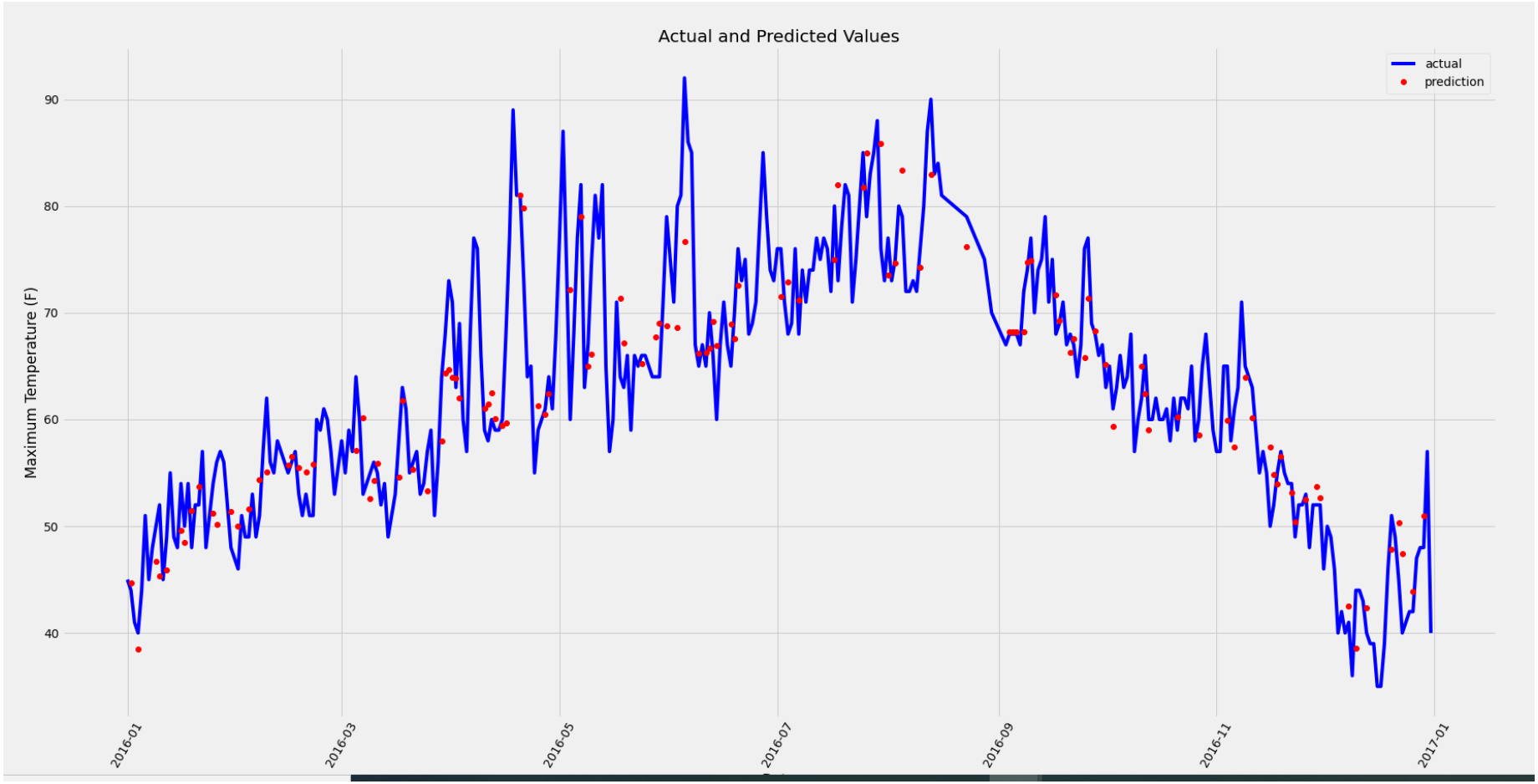
```
In [10]:
    ...:
    ...: regressor_most_important = RandomForestRegressor(n_estimators= 1000, random_state=50)
    ...: # Sacando de lista de var X, a temp_1 y average
    ...: important_indices = [x_list.index('temp_1'), x_list.index('average')]
    ...: train_important = train_x[:, important_indices]
    ...: test_important = test_x[:, important_indices]
    ...: # Entrenando RF con temp_1 y Average
    ...: regressor_most_important.fit(train_important, train_y)
    ...: # Realizando predicciones y calculando el error
    ...: predictions = regressor_most_important.predict(test_important)
    ...: errors_mi = abs(predictions - test_y)
    ...: # Mostrando metricas de predicción
    ...: print('Mean Absolute Error:', round(np.mean(errors_mi), 2), 'degrees.')
    ...: mape = np.mean(100 * (errors_mi / test_y))
    ...: accuracy = 100 - mape
    ...: print('Accuracy:', round(accuracy, 2), '%.')
Mean Absolute Error: 3.56 degrees.
Accuracy: 94.32 %.
```

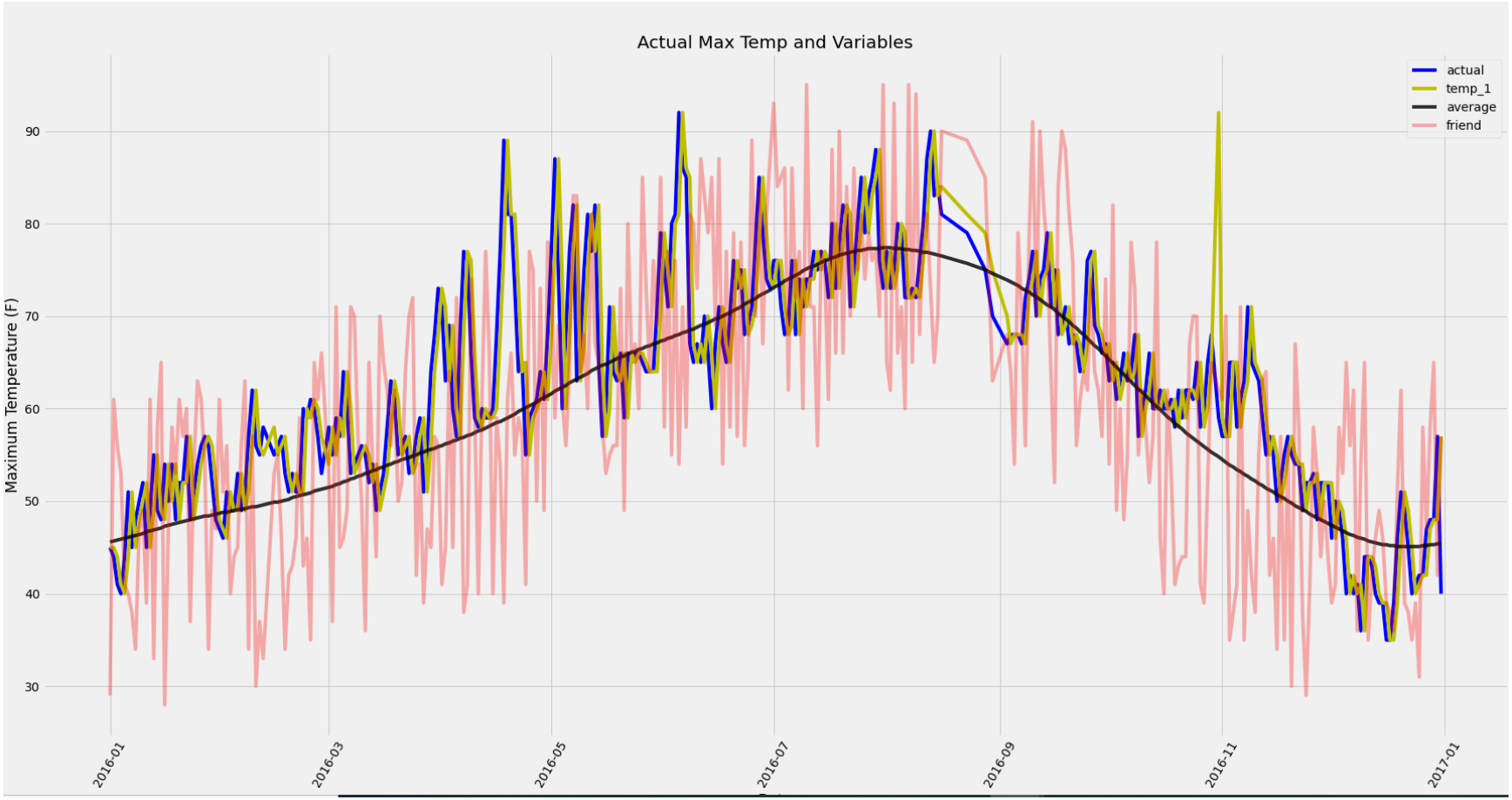
La mejora no es significativa el error promedio paso de 3.57 a 3.56 y la precisión fue la misma 94%, sin embargo, en términos de memoria y recursos de maquina se optimizan tiempos de cómputo y el modelo es más eficiente.

11. Interpretar el modelo e informar los resultados de forma visual y numérica

Para predecir la variable actual, solo es necesario usar temp_1 y average.

Gráficos para comparar actual versus los valores predichos y ver desempeño del modelo


Actual and Predicted Values
actual
prediction
Maximum Temperature (F)
90
80
70
60
50
40
2016-01
2016-03
2016-05
2016-07
2016-09
2016-11
2017-01


Actual Max Temp and Variables
actual
temp_1
average
friend
Maximum Temperature (F)
90
80
70
60
50
40
30
2016-01
2016-03
2016-05
2016-07
2016-09
2016-11
2017-01

Se corrieron modelos adicionales como: svr(rbf), svr(poly=3), svr(poly=5), svr(lineal), decision_tree_regressor y regresión multilineal.

El mejor modelo fue el random_forest, a continuación, tabla comparativa con análisis ANOVA

Param. / Modelo	Random Forest	rbf	Pol grado 3	Pol grado 5	svr-linear	DTR	Multilineal
STCC	48268.4	48268.4	48268.4	48268.4	48268.4	48268.4	48268.4
SCE	1125.06	5956.99	6310.48	4844.3	7761.73	0	7596.09
SCR	47143.3	42311.4	41957.9	43424.1	40506.6	48268.4	40672.3
n	348	348	348	348	348	348	348
k	1	1	1	1	1	1	1
S^2	3.25163	17.2167	18.2384	14.0009	22.4327	0	21.954
F	14498.4	2457.57	2300.52	3101.53	1805.69	inf	1852.61
p-Value	1.11022e-16	1.11022e-16	1.11022e-16	1.11022e-16	1.11022e-16	0	1.11022e-16
r^2	0.976691	0.876586	0.869262	0.899638	0.839196	1	0.842628