In [1]:
```
# Import libraries and dataset

import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

import os
import pandas as pd
pd.set_option('display.max_columns', None)
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
%matplotlib inline
plt.rcParams["figure.figsize"]=(20,10)
import seaborn as sns
from scipy import stats

import sklearn
from sklearn.model_selection import train_test_split
#models
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.neural_network import MLPRegressor
from sklearn.neighbors import KNeighborsRegressor
#model testing
from sklearn import metrics
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import cross_val_predict
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_validate
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import GridSearchCV
```

In [2]:
```
notebook_path = os.path.abspath("Notebook.ipynb")
dengAI_features_path="/Users/nataliecarlson/Desktop/MachineLearning/A3
df = pd.read_csv(dengAI_features_path)
dengAI_labels_path="/Users/nataliecarlson/Desktop/MachineLearning/A3/C
y = pd.read_csv(dengAI_labels_path)
```

In [3]:
```

# Merge data frames
```

In [4]:
```
df.columns
```

Out[4]:
```
Index(['city', 'year', 'weekofyear', 'week_start_date', 'ndvi_ne', 'n
dvi_nw',
       'ndvi_se', 'ndvi_sw', 'precipitation_amt_mm', 'reanalysis_air_
temp_k',
       'reanalysis_avg_temp_k', 'reanalysis_dew_point_temp_k',
       'reanalysis_max_air_temp_k', 'reanalysis_min_air_temp_k',
       'reanalysis_precip_amt_kg_per_m2',
       'reanalysis_relative_humidity_percent', 'reanalysis_sat_precip
_amt_mm',
       'reanalysis_specific_humidity_g_per_kg', 'reanalysis_tdtr_k',
       'station_avg_temp_c', 'station_diur_temp_rng_c', 'station_max_
temp_c',
       'station_min_temp_c', 'station_precip_mm'],
      dtype='object')
```

In [5]:
```
y.columns
```

Out[5]:
```
Index(['city', 'year', 'weekofyear', 'total_cases'], dtype='object')
```

In [6]:
```
#add output of y to main data frame
df['total_cases']=y['total_cases']
df
```

Out[6]:

	city	year	weekofyear	week_start_date	ndvi_ne	ndvi_nw	ndvi_se	ndvi_sw	precipi
0	sj	1990	18	1990-04-30	0.122600	0.103725	0.198483	0.177617	
1	sj	1990	19	1990-05-07	0.169900	0.142175	0.162357	0.155486	
2	sj	1990	20	1990-05-14	0.032250	0.172967	0.157200	0.170843	
3	sj	1990	21	1990-05-21	0.128633	0.245067	0.227557	0.235886	
4	sj	1990	22	1990-05-28	0.196200	0.262200	0.251200	0.247340	
...	...	...	...	...	...	...	...	...	
1451	iq	2010	21	2010-05-28	0.342750	0.318900	0.256343	0.292514	
1452	iq	2010	22	2010-06-04	0.160157	0.160371	0.136043	0.225657	
1453	iq	2010	23	2010-06-11	0.247057	0.146057	0.250357	0.233714	
1454	iq	2010	24	2010-06-18	0.333914	0.245771	0.278886	0.325486	
1455	iq	2010	25	2010-06-25	0.298186	0.232971	0.274214	0.315757	

1456 rows × 25 columns

In [7]:
```
#view feature data types
df.dtypes
```

Out[7]:
```
city                                      object
year                                       int64
weekofyear                                 int64
week_start_date                           object
ndvi_ne                                  float64
ndvi_nw                                  float64
ndvi_se                                  float64
ndvi_sw                                  float64
precipitation_amt_mm                     float64
reanalysis_air_temp_k                    float64
reanalysis_avg_temp_k                    float64
reanalysis_dew_point_temp_k              float64
reanalysis_max_air_temp_k                float64
reanalysis_min_air_temp_k                float64
reanalysis_precip_amt_kg_per_m2          float64
reanalysis_relative_humidity_percent     float64
reanalysis_sat_precip_amt_mm             float64
reanalysis_specific_humidity_g_per_kg    float64
reanalysis_tdtr_k                        float64
station_avg_temp_c                       float64
station_diur_temp_rng_c                  float64
station_max_temp_c                       float64
station_min_temp_c                       float64
station_precip_mm                        float64
total_cases                                int64
dtype: object
```

In [8]:
```
#Convert city data type to catergory
df.city = pd.Categorical(df.city)
#Convert category to int code
df['city']=df.city.cat.codes
```

In [9]:
```
#remove "week_start_date" (redundant to "year" and "weekofyear" and in
df=df.drop(['week_start_date'],axis=1) #drop
```

In [10]:
```

# Assess and manipulate null values

```

In [11]:
```
# #remove completely empty records
df.dropna(how = 'all')
# #remove records with no output
df = df[df['total_cases'].notna()]
```

In [12]:
```
#view count of empty cells by column
df.isnull().sum()
```

Out[12]:
```
city                                       0
year                                       0
weekofyear                                 0
ndvi_ne                                  194
ndvi_nw                                   52
ndvi_se                                   22
ndvi_sw                                   22
precipitation_amt_mm                      13
reanalysis_air_temp_k                     10
reanalysis_avg_temp_k                     10
reanalysis_dew_point_temp_k               10
reanalysis_max_air_temp_k                 10
reanalysis_min_air_temp_k                 10
reanalysis_precip_amt_kg_per_m2           10
reanalysis_relative_humidity_percent      10
reanalysis_sat_precip_amt_mm              13
reanalysis_specific_humidity_g_per_kg     10
reanalysis_tdtr_k                         10
station_avg_temp_c                        43
station_diur_temp_rng_c                   43
station_max_temp_c                        20
station_min_temp_c                        14
station_precip_mm                         22
total_cases                                0
dtype: int64
```

In [13]:
```
#view general statistics
df.describe()
```

Out[13]:

	city	year	weekofyear	ndvi_ne	ndvi_nw	ndvi_se	ndv
count	1456.000000	1456.000000	1456.000000	1262.000000	1404.000000	1434.000000	1434.00
mean	0.642857	2001.031593	26.503434	0.142294	0.130553	0.203783	0.20
std	0.479322	5.408314	15.019437	0.140531	0.119999	0.073860	0.08
min	0.000000	1990.000000	1.000000	-0.406250	-0.456100	-0.015533	-0.06
25%	0.000000	1997.000000	13.750000	0.044950	0.049217	0.155087	0.14
50%	1.000000	2002.000000	26.500000	0.128817	0.121429	0.196050	0.18
75%	1.000000	2005.000000	39.250000	0.248483	0.216600	0.248846	0.24
max	1.000000	2010.000000	53.000000	0.508357	0.454429	0.538314	0.54

In [14]:
```
#fill missing values with data from data in row above (the week before
# Note: Week one is fully filled for each city, i.e. there is a starti
df = df.ffill(axis = 0)
```

In [15]:
```
#view updated count of empty cells by column
df.isnull().sum()
```

Out[15]:
```
city                                     0
year                                     0
weekofyear                               0
ndvi_ne                                  0
ndvi_nw                                  0
ndvi_se                                  0
ndvi_sw                                  0
precipitation_amt_mm                     0
reanalysis_air_temp_k                    0
reanalysis_avg_temp_k                    0
reanalysis_dew_point_temp_k              0
reanalysis_max_air_temp_k                0
reanalysis_min_air_temp_k                0
reanalysis_precip_amt_kg_per_m2          0
reanalysis_relative_humidity_percent     0
reanalysis_sat_precip_amt_mm             0
reanalysis_specific_humidity_g_per_kg    0
reanalysis_tdtr_k                        0
station_avg_temp_c                       0
station_diur_temp_rng_c                  0
station_max_temp_c                       0
station_min_temp_c                       0
station_precip_mm                        0
total_cases                              0
dtype: int64
```

In [16]:
```
#view updated general statistics and visually compare to prior
df.describe()
```

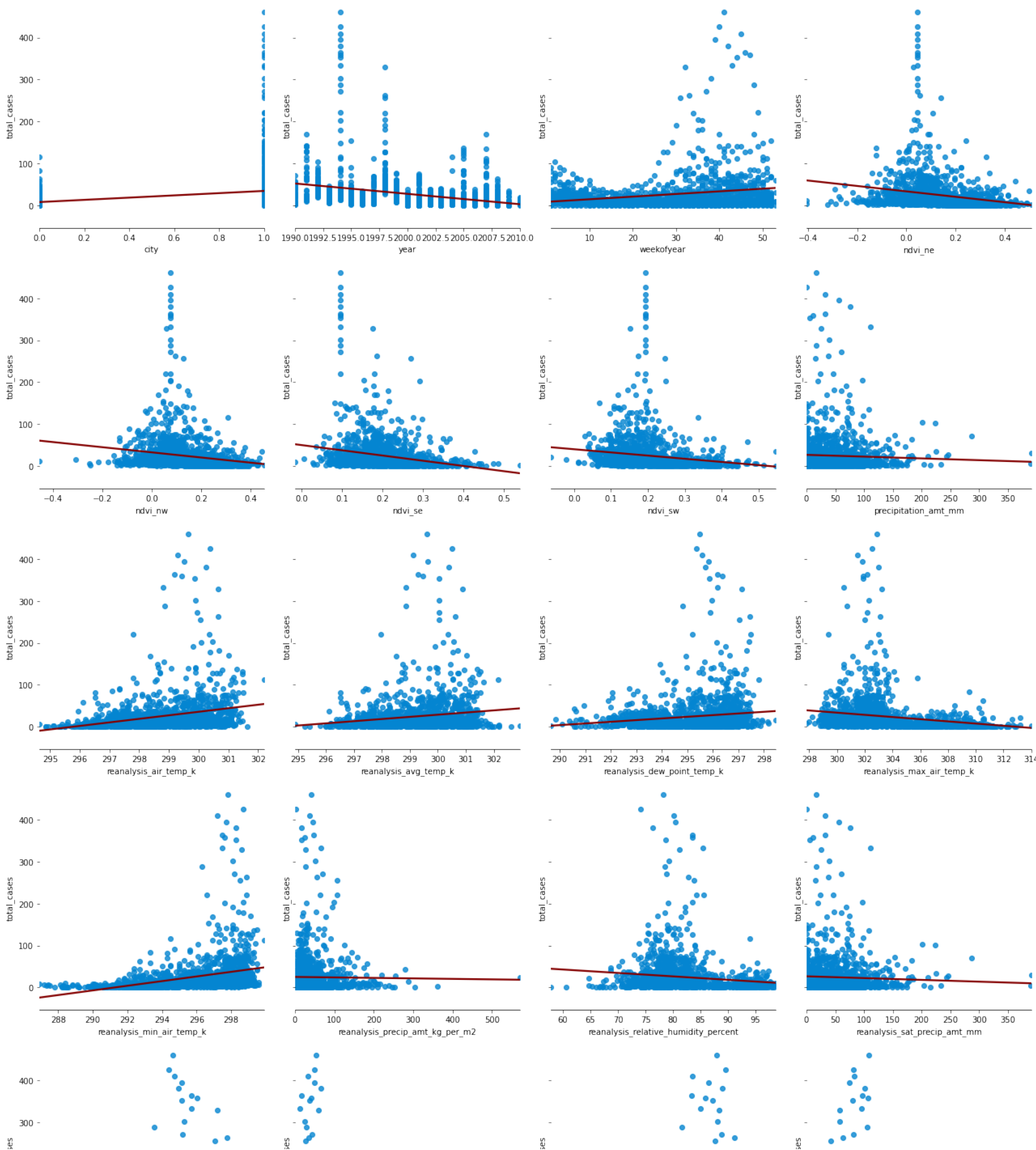
Out[16]:

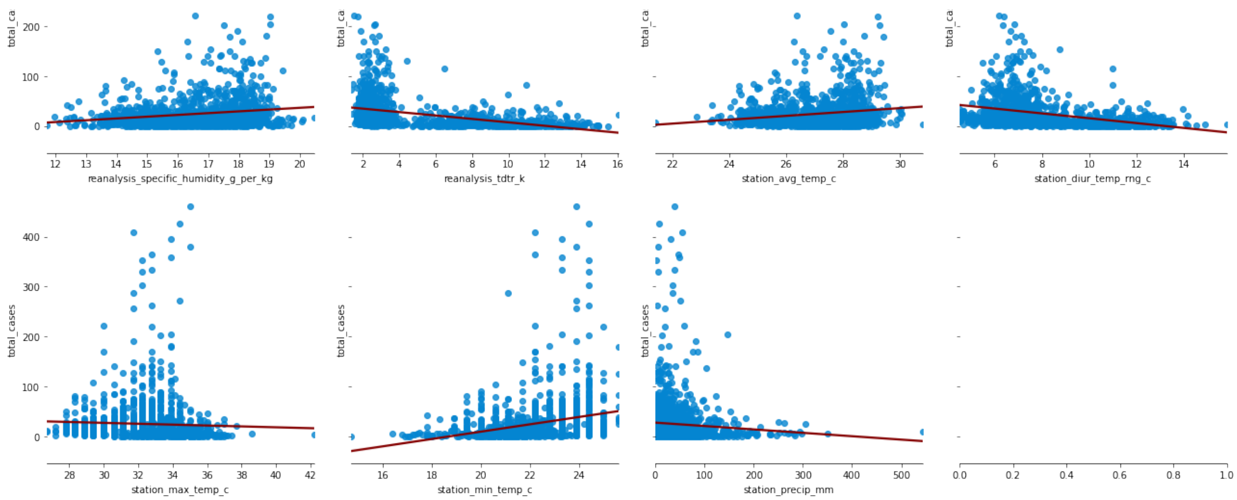
	city	year	weekofyear	ndvi_ne	ndvi_nw	ndvi_se	ndv
count	1456.000000	1456.000000	1456.000000	1456.000000	1456.000000	1456.000000	1456.00
mean	0.642857	2001.031593	26.503434	0.131271	0.128068	0.202606	0.20
std	0.479322	5.408314	15.019437	0.138527	0.119561	0.074409	0.08
min	0.000000	1990.000000	1.000000	-0.406250	-0.456100	-0.015533	-0.06
25%	0.000000	1997.000000	13.750000	0.039100	0.048250	0.152795	0.14
50%	1.000000	2002.000000	26.500000	0.113900	0.115926	0.195664	0.19
75%	1.000000	2005.000000	39.250000	0.232018	0.213429	0.247461	0.24
max	1.000000	2010.000000	53.000000	0.508357	0.454429	0.538314	0.54

In [17]:
```

#Visulaize the data
```

In [18]:
```
# View linear regression of total cases per attribute

xvlist = list(df.drop(['total_cases'],axis=1))

fcol = 4 # limit four graphs per row for easy visualization
frow = int(np.ceil(len(xvlist)/fcol)) # number of rows in your subplot
fhgt = frow*4.5 # height

# Set up the matplotlib figure
f, axes = plt.subplots(frow, fcol, figsize=(18, fhgt), sharey=True)
sns.despine(left=True)

# make a list of items to iterate over to produce graph
axes_list = [item for sublist in axes for item in sublist]

for k, xvar in enumerate(xvlist):

    sns.regplot(
        x=xvar,
        y='total_cases',
        data=df,
        ax=axes_list[k],
        ci = None, # set the confidence interval to none, so no resamp
        logx=False,
        scatter_kws={'color': 'xkcd:cerulean'}, # using xkcd color coc
        line_kws={'color': '#840000'} # using hex color codes
```

```
        line_kws={'color': '#810000'} # using hex color codes
    )

plt.tight_layout()
```




total_ca
total_cases
reanalysis_specific_humidity_g_per_kg
reanalysis_tdtr_k
station_avg_temp_c
station_diur_temp_rng_c
station_max_temp_c
station_min_temp_c
station_precip_mm

In [19]:
```
#View columns with extreme outliers
#Note: first array == row, second array == column
#column name                                number
# city                                          1
# year                                          2
# weekofyear                                    3
# ndvi_ne                                       4
# ndvi_nw                                       5
# ndvi_se                                       6
# ndvi_sw                                       7
# precipitation_amt_mm                          8
# reanalysis_air_temp_k                         9
# reanalysis_avg_temp_k                         10
# reanalysis_dew_point_temp_k                   11
# reanalysis_max_air_temp_k                     12
# reanalysis_min_air_temp_k                     13
# reanalysis_precip_amt_kg_per_m2               14
# reanalysis_relative_humidity_percent          15
# reanalysis_sat_precip_amt_mm                  16
# reanalysis_specific_humidity_g_per_kg         17
# reanalysis_tdtr_k                             18
# station_avg_temp_c                            19
# station_diur_temp_rng_c                       20
# station_max_temp_c                            21
# station_min_temp_c                            22
# station_precip_mm                             23
# total_cases                                   24

z = np.abs(stats.zscore(df))
print(np.where(z>5))
```

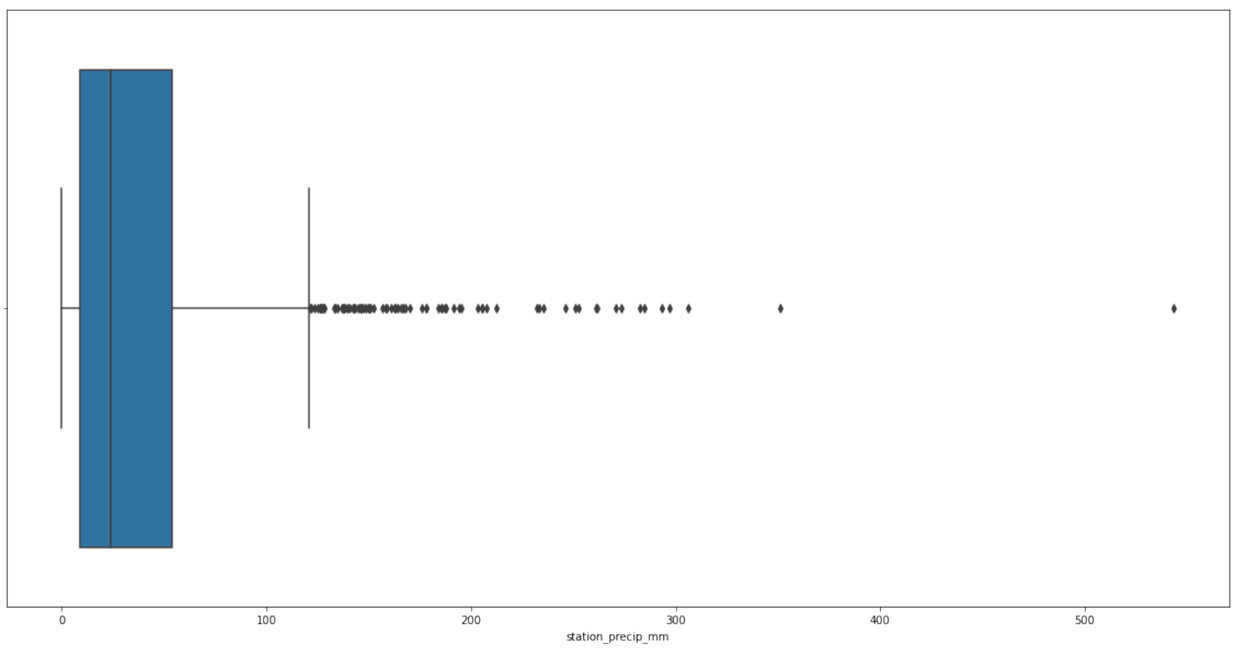
```
(array([  24,  107,  228,  229,  230,  231,  232,  233,  234,  235,
236,
        237,  238,  239,  332,  429,  430,  431,  600,  600,  675,  6
75,
        705,  705,  974, 1033, 1077, 1112, 1138, 1177, 1321, 1338, 14
38]), array([13, 13, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23,
22, 23, 23,
       23,  7, 15,  7, 15,  7, 15, 13, 22, 13, 22, 22, 22, 22, 22, 13
]))
```

In [20]:
```
#outlier counts by column number
oc = pd.DataFrame({'col': [13, 13, 23, 23, 23, 23, 23, 23, 23, 23, 23,
       23,  7, 15,  7, 15,  7, 15, 13, 22, 13, 22, 22, 22, 22, 22, 13]
#frequency count
count = oc['col'].value_counts()
print(count)
```

```
23    15
22     7
13     5
15     3
7      3
Name: col, dtype: int64
```

In [21]:
```
sns.boxplot(x=df['station_precip_mm']) #column 23
```

Out[21]: <matplotlib.axes._subplots.AxesSubplot at 0x124ef8fd0>

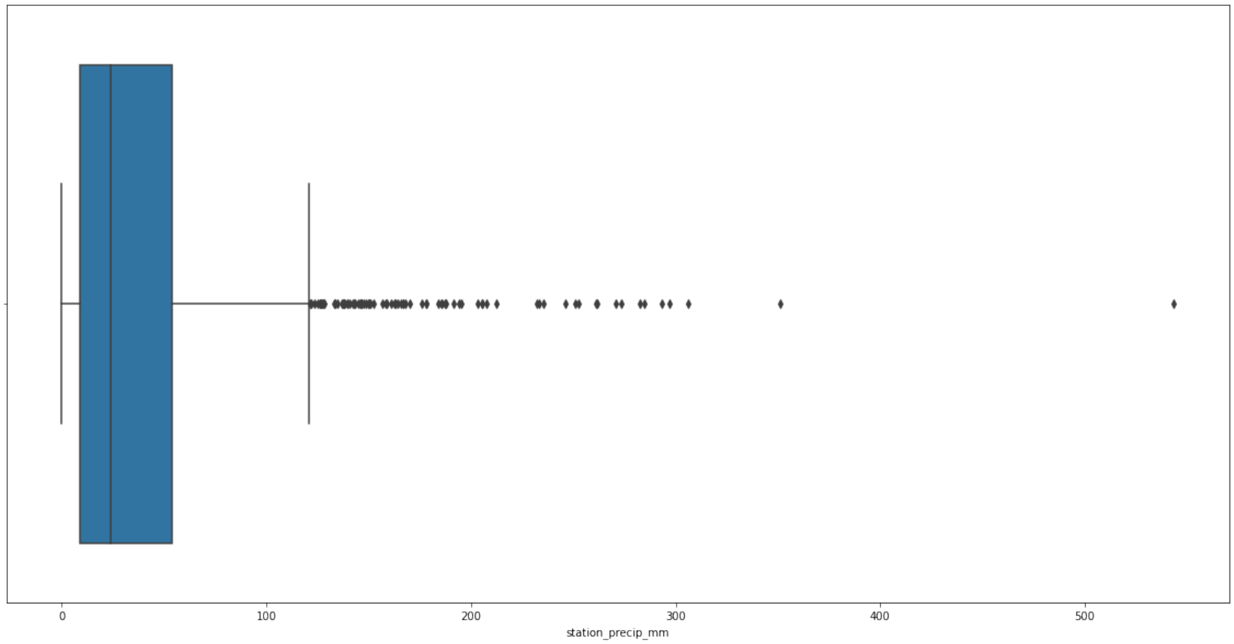


In [22]:
```
# Get outlier record numbers
station_precip_mm_outliers = df[ df['station_precip_mm'] > 300 ].index
station_precip_mm_outliers
```

Out[22]: Int64Index([332, 1033, 1177], dtype='int64')

In [23]:
```
sns.boxplot(x=df['station_precip_mm']) #column 22
```

Out[23]: <matplotlib.axes._subplots.AxesSubplot at 0x125e94880>

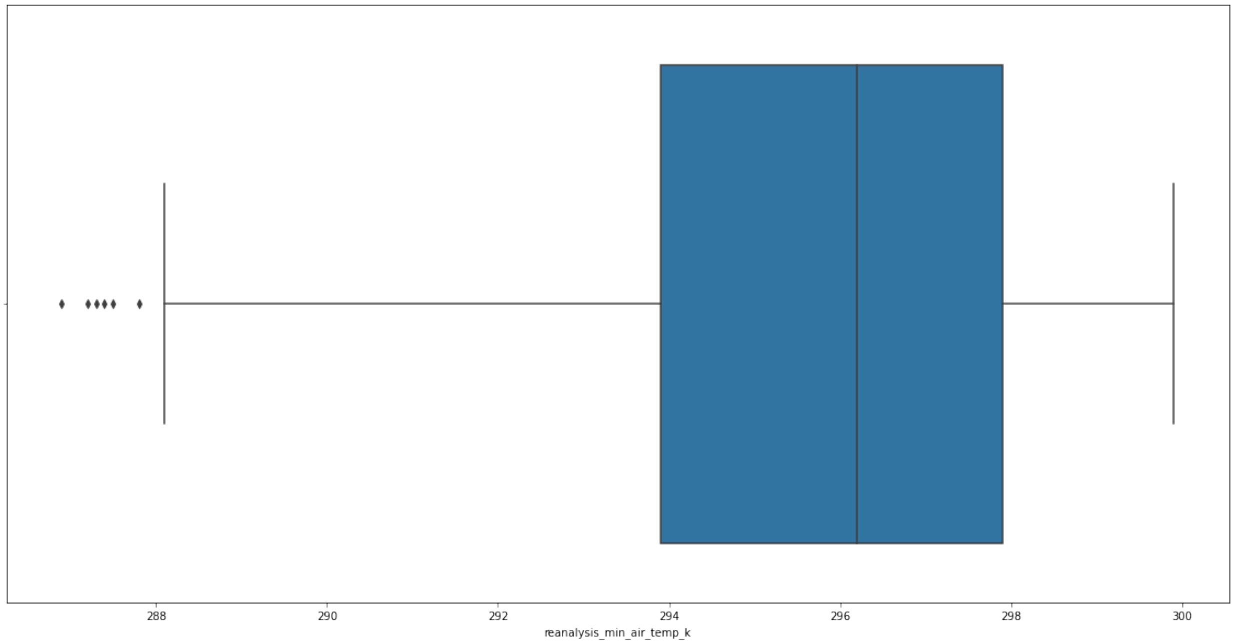


In [24]:
```
# Get outlier record numbers
station_min_temp_c_outliers = df[ df['station_min_temp_c'] < 17 ].inde
station_min_temp_c_outliers
```

Out[24]: Int64Index([939, 1103, 1208, 1304], dtype='int64')

In [25]:
```
sns.boxplot(x=df['reanalysis_min_air_temp_k']) #column 13
```

Out[25]: <matplotlib.axes._subplots.AxesSubplot at 0x1220ede20>

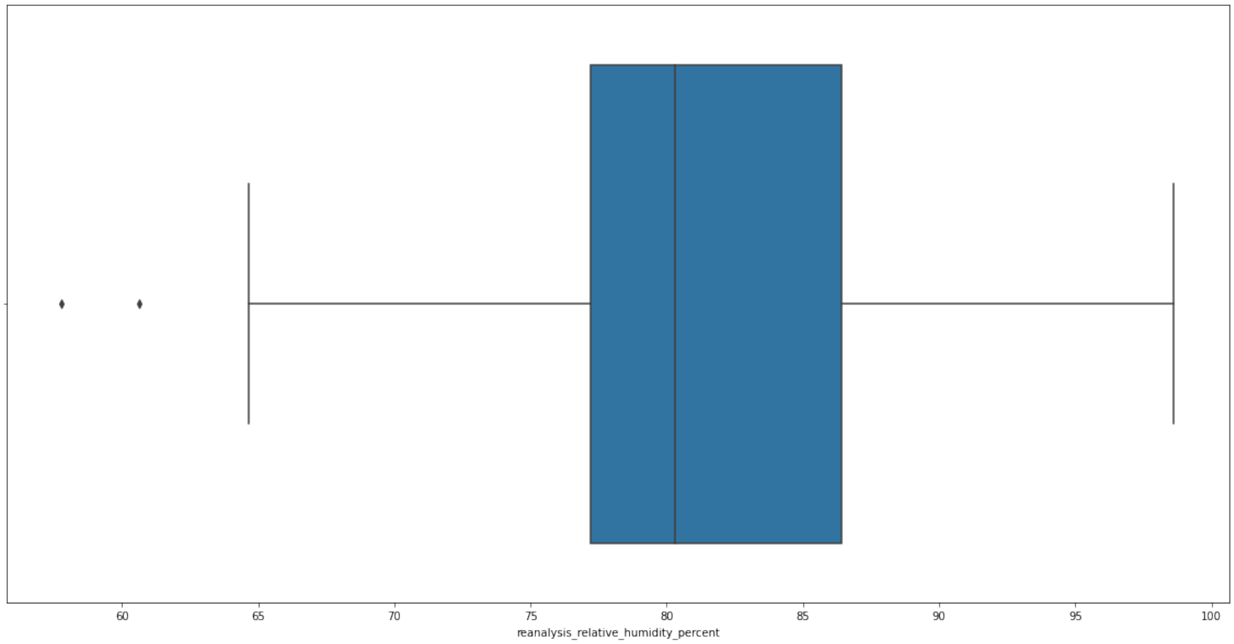


In [26]:
```
# Get outlier record numbers
reanalysis_min_air_temp_k_outliers = df[ df['reanalysis_min_air_temp_k
reanalysis_min_air_temp_k_outliers
```

Out[26]: Int64Index([1040, 1094, 1199, 1242, 1295, 1355], dtype='int64')

In [27]:
```
sns.boxplot(x=df['reanalysis_relative_humidity_percent']) #column 15
```

Out[27]: <matplotlib.axes._subplots.AxesSubplot at 0x122ad6ca0>

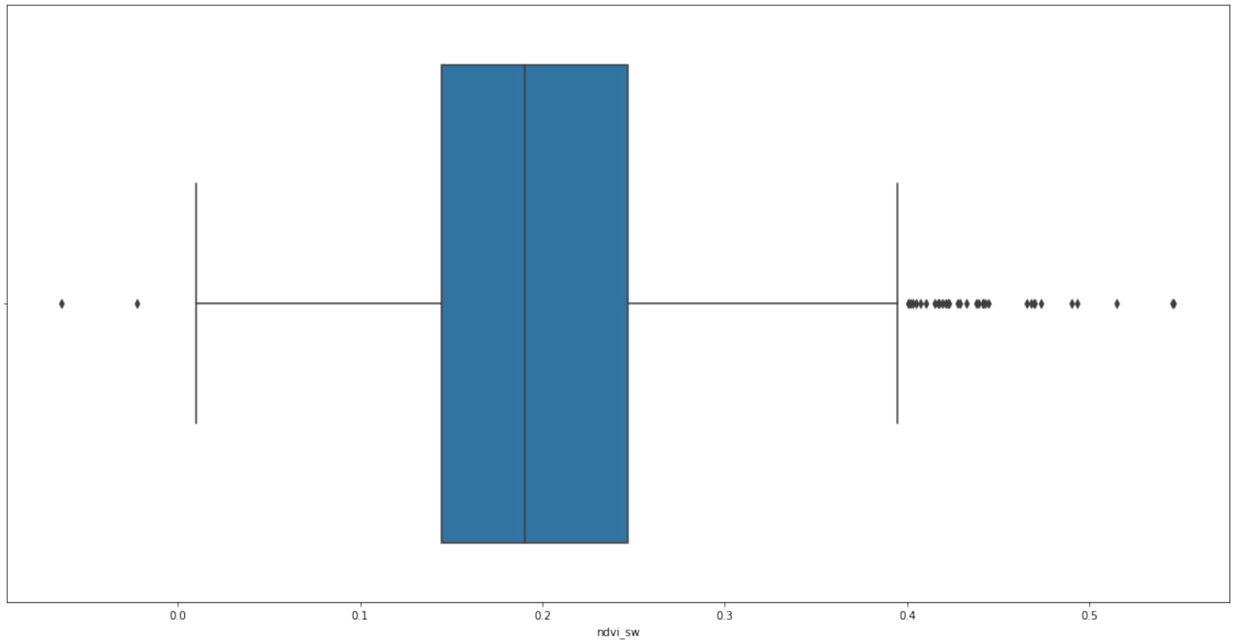


In [28]:
```
# Get outlier record numbers
reanalysis_relative_humidity_percent_outliers = df[ df['reanalysis_rel
reanalysis_relative_humidity_percent_outliers
```

Out[28]: Int64Index([946, 961, 1051, 1203], dtype='int64')

In [29]:
```
sns.boxplot(x=df['ndvi_sw']) #column 7
```

Out[29]: <matplotlib.axes._subplots.AxesSubplot at 0x122b40820>



In [30]:
```
# Get outlier record numbers
reanalysis_min_air_temp_k_outliers = df[ df['ndvi_sw'] > 0.5 ].index
reanalysis_min_air_temp_k_outliers
```

Out[30]: Int64Index([1002, 1370, 1413], dtype='int64')

In [31]:
```
#view records with feature extreme outliers

#record num   feature with outlier
# 332:        station_precip_mm
# 939:        station_min_temp_c
# 946:        reanalysis_relative_humidity_percent
# 961:        reanalysis_relative_humidity_percent
# 1002:       ndvi_sw
# 1033:       station_precip_mm
# 1040:       reanalysis_min_air_temp_k
# 1051:       reanalysis_relative_humidity_percent
# 1094:       reanalysis_min_air_temp_k
# 1103:       station_min_temp_c
# 1177:       station_precip_mm
# 1199:       reanalysis_min_air_temp_k
# 1203:       reanalysis_relative_humidity_percent
# 1208:       station_min_temp_c
# 1242:       reanalysis_min_air_temp_k
# 1295:       reanalysis_min_air_temp_k
# 1304:       station_min_temp_c
# 1355:       reanalysis_min_air_temp_k
```

```
# 1370:       ndvi_sw
# 1413:       ndvi_sw
df.loc[[332,939,946,961,1002,1033,1040,1051,1094,1103,1177,1199,1203,1
```

Out[31]:

	city	year	weekofyear	ndvi_ne	ndvi_nw	ndvi_se	ndvi_sw	precipitation_amt_mm	r
332	1	1996	38	0.104300	0.028450	0.150429	0.107817	243.55	
939	0	2000	29	0.227729	0.145429	0.254200	0.200314	5.60	
946	0	2000	36	0.295586	0.295683	0.312214	0.265929	23.12	
961	0	2000	51	0.265014	0.169057	0.234867	0.302243	3.90	
1002	0	2001	41	0.420286	0.381957	0.443800	0.546017	58.75	
1033	0	2002	20	0.133800	0.109633	0.219614	0.141700	46.79	
1040	0	2002	27	0.101543	0.126600	0.119357	0.076243	3.27	
1051	0	2002	38	0.327414	0.347257	0.338671	0.383871	11.49	
1094	0	2003	29	0.193571	0.168850	0.159086	0.231471	65.20	
1103	0	2003	38	0.364486	0.260986	0.316457	0.385157	55.73	
1177	0	2005	7	0.199500	0.202457	0.218900	0.180686	125.35	
1199	0	2005	29	0.168029	0.134329	0.186557	0.147257	27.33	
1203	0	2005	33	0.266986	0.293929	0.336000	0.292486	17.85	
1208	0	2005	38	0.184567	0.195257	0.154386	0.212483	2.56	
1242	0	2006	20	0.339286	0.348286	0.315829	0.374571	1.52	
1295	0	2007	22	0.090057	0.082229	0.111057	0.064743	76.68	
1304	0	2007	31	0.417229	0.369929	0.366971	0.489871	70.88	
1355	0	2008	30	0.327017	0.209117	0.283767	0.309550	52.36	
1370	0	2008	45	0.501029	0.445000	0.427686	0.545729	57.85	
1413	0	2009	36	0.508357	0.454429	0.538314	0.514829	83.02	

In [32]:

```
#save original unaltered dataframe
dfOriginal = df
```

In [33]:
```
#update df to exclude feature extreme outliers
df = dfOriginal.drop(df.index[[332,939,946,961,1002,1033,1040,1051,109
```

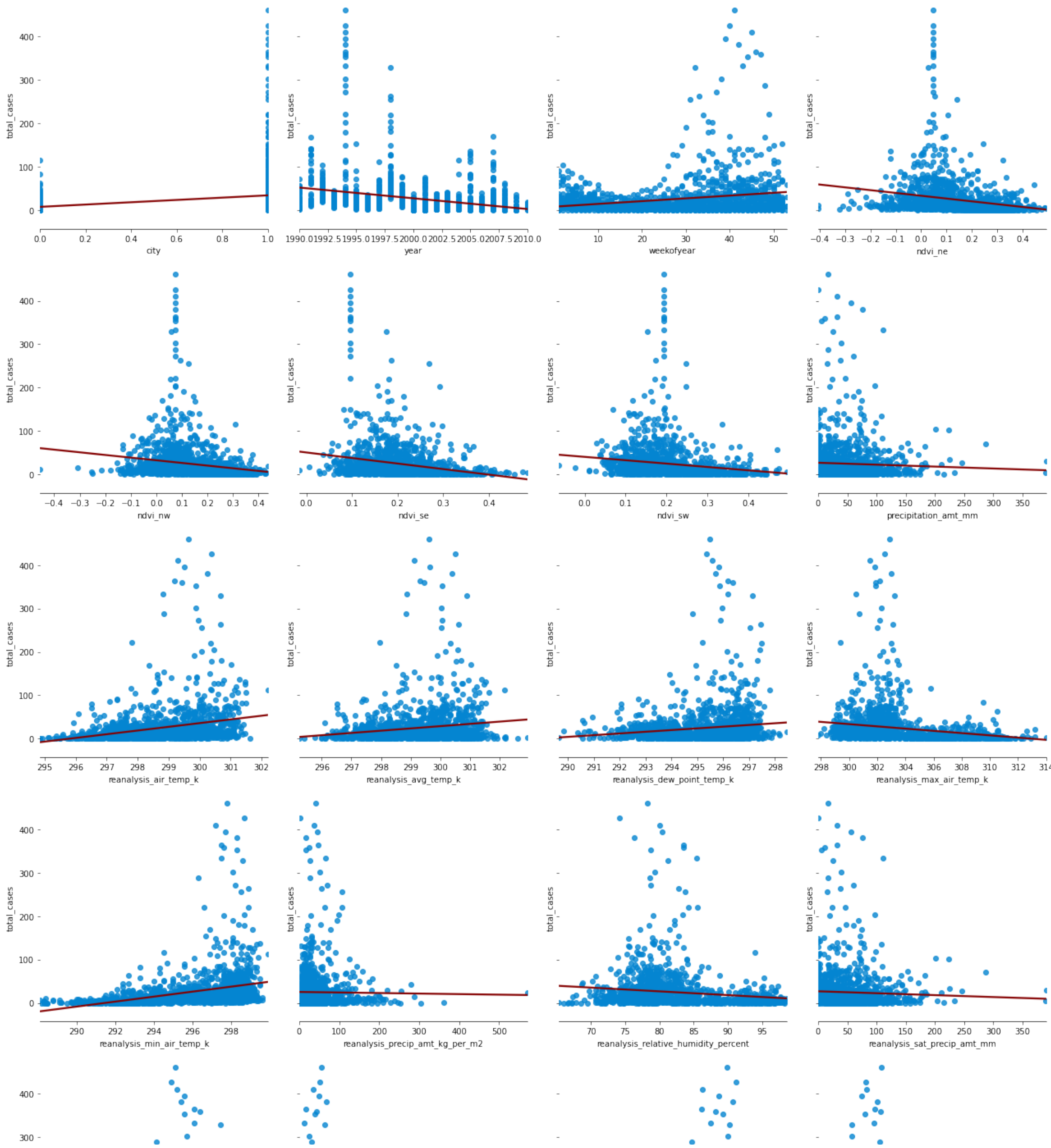
In [34]:
```
#view updated df general statistics
df.describe()
```

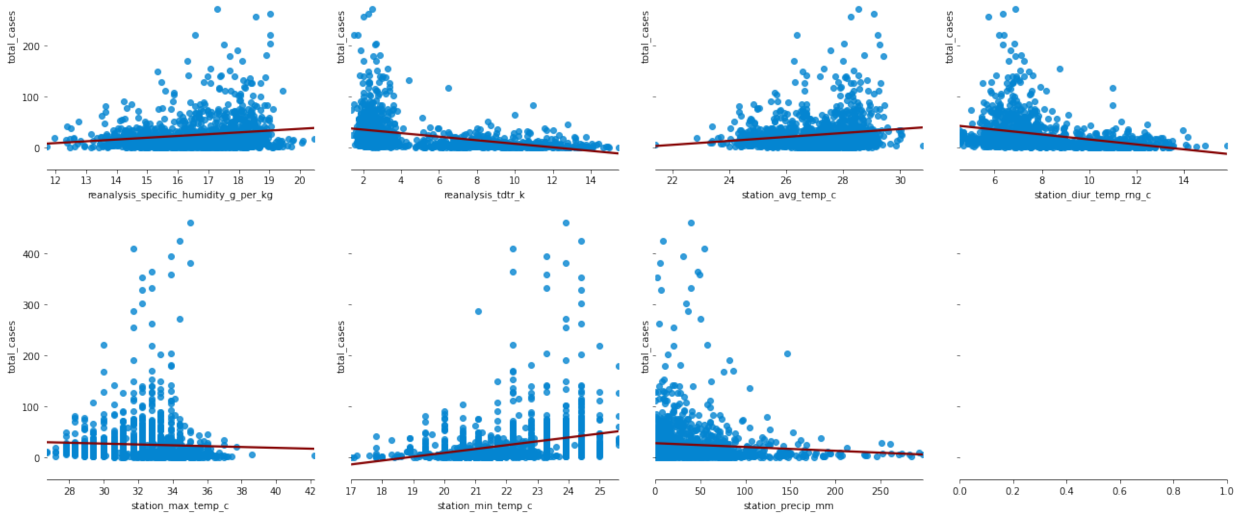
Out[34]:

	city	year	weekofyear	ndvi_ne	ndvi_nw	ndvi_se	ndv
count	1436.000000	1436.000000	1436.000000	1436.000000	1436.000000	1436.000000	1436.00
mean	0.651114	2000.994429	26.428273	0.129314	0.126530	0.201606	0.20
std	0.476784	5.422752	15.067227	0.137697	0.118830	0.073245	0.08
min	0.000000	1990.000000	1.000000	-0.406250	-0.456100	-0.015533	-0.06
25%	0.000000	1997.000000	13.000000	0.037437	0.047767	0.152700	0.14
50%	1.000000	2002.000000	26.000000	0.112200	0.114633	0.195086	0.18
75%	1.000000	2005.000000	40.000000	0.230127	0.212168	0.246482	0.24
max	1.000000	2010.000000	53.000000	0.493400	0.437100	0.484286	0.49

In [35]:
```
# View linear regression of total cases per attribute

xvlist = list(df.drop(['total_cases'],axis=1))

fcol = 4 # limit four graphs per row for easy visualization
frow = int(np.ceil(len(xvlist)/fcol)) # number of rows in your subplot
fhgt = frow*4.5 # height

# Set up the matplotlib figure
f, axes = plt.subplots(frow, fcol, figsize=(18, fhgt), sharey=True)
sns.despine(left=True)

# make a list of items to iterate over to produce graph
axes_list = [item for sublist in axes for item in sublist]

for k, xvar in enumerate(xvlist):

    sns.regplot(
        x=xvar,
        y='total_cases',
        data=df,
        ax=axes_list[k],
        ci = None, # set the confidence interval to none, so no resamp
        logx=False,
        scatter_kws={'color': 'xkcd:cerulean'}, # using xkcd color coc
```

```
        line_kws={'color': '#840000'} # using hex color codes
    )

plt.tight_layout()
```





In [36]:
```
# Train Data
```

In [37]:
```
#Set X and Y variables
Y=df.total_cases
X=df.iloc[:, :-1]
```

In [38]:
```
#Split
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=.2
```

In [39]:
```
#Scale
scaler=StandardScaler().fit(x_train)
print(scaler.scale_)
print("\n")
print(scaler.mean_)
```

```
[ 0.47591597  5.42284013 15.18363844  0.13683864  0.11923098  0.07261
706
  0.08197002 43.97413739  1.3505885   1.25183431  1.50150792  3.17366
27
  2.46736251 38.73920471  7.03553824 43.97413739  1.51979123  3.46582
41
  1.26267291  2.07867185  1.96360501  1.50711766 44.91602668]


[6.53310105e-01 2.00095296e+03 2.63092334e+01 1.28336290e-01
 1.24822602e-01 2.00852745e-01 1.99756125e-01 4.59379878e+01
 2.98711142e+02 2.99225977e+02 2.95274689e+02 3.03334408e+02
 2.95802352e+02 3.87889460e+01 8.22158885e+01 4.59379878e+01
 1.67724017e+01 4.80093330e+00 2.71805687e+01 8.03133379e+00
 3.24062718e+01 2.21417247e+01 3.87328397e+01]
```

In [ ]:
```
#Fit models
```

In [ ]:
```
#fit linear regression
lin_reg = LinearRegression(n_jobs=-1)
lin_reg.fit(scaler.transform(x_train), y_train)
```

In [ ]:
```
#fit decision tree
tree = DecisionTreeRegressor()
tree.fit(scaler.transform(x_train), y_train)
```

In [ ]:
```
#fit random forest
forest = RandomForestRegressor(n_jobs=-1)
forest.fit(scaler.transform(x_train), y_train)
```

In [ ]:
```
#fit Multilayer Perceptron (MLP)
percept = MLPRegressor()
percept.fit(scaler.transform(x_train), y_train)
```

In [ ]:
```
#fit k-Nearest Neighbour Regression (KNNR)
neighbor = KNeighborsRegressor(n_neighbors=2, n_jobs=-1)
neighbor.fit(scaler.transform(x_train), y_train)
```

In [ ]:
```
#Evalaute Models
models= [('Linear Regression', lin_reg), ('Random Forest', forest), ('
```

In [ ]:
```
#RMSE(Root Mean Squared Error)
#Difference between actual and predicted output outputs
#Note: Lower is better
for i, model in models:
    predictions = model.predict(scaler.transform(x_train))
    MSE = metrics.mean_squared_error(y_train, predictions)
    RMSE = np.sqrt(MSE)
    msg = "%s = %.2f" % (i, round(RMSE, 2))
    print('RMSE of', msg)
```

In [ ]:
```
#MAE(Mean Absolute Error)
#Average sum of absolute errors
#Note: Lower is better
for i, model in models:
    predictions = model.predict(scaler.transform(x_train))
    MAE = metrics.mean_absolute_error(y_train, predictions)
    msg = "%s= %.2f"% (i, round(MAE, 2))
    print('MAE of', msg)
```

In [ ]:
```
#R2 (R-Squared Score)
#How much data is explained by the model
#Note: 0 - 1 Higher is better
for i, model in models:
    predictions = model.predict(scaler.transform(x_train))
    R2 = metrics.r2_score(y_train, predictions)
    msg = "%s= %.2f"% (i, round(R2, 2))
    print('R2 of', msg)
```

In [ ]:
```
#Selected model: Random Forest
print("Random Forest:")

#Evalaute Model
prediction= forest.predict(scaler.transform(x_train))

MSE = metrics.mean_squared_error(y_train, prediction)
RMSE = np.sqrt(MSE)
msg = "%.2f" % (round(RMSE, 2))
print('RMSE =', msg)

MAE = metrics.mean_absolute_error(y_train, prediction)
msg = "%.2f"% (round(MAE, 2))
print('MAE =', msg)

R2 = metrics.r2_score(y_train, prediction)
msg = "%.2f"% (round(R2, 2))
print('R2 =', msg)
```

In [ ]:
```
# Fine-tune Random Forest

#n_estimators = n of trees
#max_features = max number of features considered for splitting a node
#max_depth = max number of levels in each decision tree
#min_samples_split = min number of data points placed in a node before
#min_samples_leaf = min number of data points allowed in a leaf node
#bootstrap = method for sampling data points (with or without replacem
```

In [ ]:
```
#Grid Search

n_estimators = [10, 25]
max_features = [5, 10]
max_depth = [10, 50, None]
bootstrap = [True, False]

param_grid=[{'n_estimators': n_estimators, 'max_features': max_feature

grid_search_forest=GridSearchCV(forest, param_grid, cv=10, scoring='ne
grid_search_forest.fit(scaler.transform(x_train), y_train)
```

In [ ]:
```
#MTS (Mean test score)
cvres = grid_search_forest.cv_results_
for mean_score, params in zip(cvres["mean_test_score"], cvres["params"
    MTS = np.sqrt(-mean_score)
    msg = "%s = %.2f"% (params, round(MTS, 2))
    print('MTS of', msg)
```

In [ ]:
```
print("Grid Search:")
#find the best model of grid search
grid_best= grid_search_forest.best_estimator_
grid_best_params= grid_search_forest.best_params_
print("best paramaters: ", grid_best_params)


#Evalaute Model
prediction= grid_best.predict(scaler.transform(x_train))

MSE = metrics.mean_squared_error(y_train, prediction)
RMSE = np.sqrt(MSE)
msg = "%.2f" % (round(RMSE, 2))
print('RMSE =', msg)

MAE = metrics.mean_absolute_error(y_train, prediction)
msg = "%.2f"% (round(MAE, 2))
print('MAE =', msg)

R2 = metrics.r2_score(y_train, prediction)
msg = "%.2f"% (round(R2, 2))
print('R2 =', msg)
```

In [ ]:
```
#Randomized Search

n_estimators = [int(x) for x in np.linspace(start = 20, stop = 200, nu
print("n_estimators: ",n_estimators)
max_features = ['auto', 'sqrt']
max_depth = [int(x) for x in np.linspace(1, 45, num = 3)]
print("max_depth: ",max_depth)
min_samples_split = [5, 10]

param_grid=[{'n_estimators': n_estimators, 'max_features': max_feature
rand_search_forest=RandomizedSearchCV(estimator = forest, param_distri
rand_search_forest.fit(scaler.transform(x_train), y_train)
```

In [ ]:
```
#MTS (Mean test score)
cvres = rand_search_forest.cv_results_
for mean_score, params in zip(cvres["mean_test_score"], cvres["params"
    MTS = np.sqrt(-mean_score)
    msg = "%s = %.2f"% (params, round(MTS, 2))
    print('MTS of', msg)
```

In [ ]:
```
print("Rand Search:")
#find the best model of rand search
rand_best= rand_search_forest.best_estimator_
rand_best_params= rand_search_forest.best_params_
print("best paramaters: ", rand_best_params)

#Evalaute Model
prediction= rand_best.predict(scaler.transform(x_train))

MSE = metrics.mean_squared_error(y_train, prediction)
RMSE = np.sqrt(MSE)
msg = "%.2f" % (round(RMSE, 2))
print('RMSE =', msg)

MAE = metrics.mean_absolute_error(y_train, prediction)
msg = "%.2f"% (round(MAE, 2))
print('MAE =', msg)

R2 = metrics.r2_score(y_train, prediction)
msg = "%.2f"% (round(R2, 2))
print('R2 =', msg)
```

In [ ]:
```
#Test set

print("Random Forest:")
#fit random forest
forest = RandomForestRegressor(n_jobs=-1)
forest.fit(scaler.transform(x_test), y_test)
#Evalaute Model
prediction= forest.predict(scaler.transform(x_test))
MSE = metrics.mean_squared_error(y_test, prediction)
RMSE = np.sqrt(MSE)
msg = "%.2f" % (round(RMSE, 2))
print('RMSE =', msg)
MAE = metrics.mean_absolute_error(y_test, prediction)
msg = "%.2f"% (round(MAE, 2))
print('MAE =', msg)
R2 = metrics.r2_score(y_test, prediction)
msg = "%.2f"% (round(R2, 2))
print('R2 =', msg)

print("\nRandom Forest with best grid parameters:")
#fit random forest with grid
param_grid=[{'bootstrap': [False], 'max_depth': [50], 'max_features':
grid_search_forest=GridSearchCV(forest, param_grid, cv=10, scoring='ne
grid_search_forest.fit(scaler.transform(x_test), y_test)
#Evalaute Model
prediction= grid_search_forest.predict(scaler.transform(x_test))
MSE = metrics.mean_squared_error(y_test, prediction)
RMSE = np.sqrt(MSE)
msg = "%.2f" % (round(RMSE, 2))
print('RMSE =', msg)
MAE = metrics.mean_absolute_error(y_test, prediction)
msg = "%.2f"% (round(MAE, 2))
print('MAE =', msg)
R2 = metrics.r2_score(y_test, prediction)
msg = "%.2f"% (round(R2, 2))
print('R2 =', msg)

print("\nRandom Forest with best random parameters:")
#fit random forest with random
param_grid=[{'n_estimators': [155], 'min_samples_split': [10], 'max_fe
rand_search_forest=RandomizedSearchCV(estimator = forest, param_distri
rand_search_forest.fit(scaler.transform(x_test), y_test)
#Evalaute Model
prediction= rand_search_forest.predict(scaler.transform(x_test))
MSE = metrics.mean_squared_error(y_test, prediction)
RMSE = np.sqrt(MSE)
msg = "%.2f" % (round(RMSE, 2))
print('RMSE =', msg)
MAE = metrics.mean_absolute_error(y_test, prediction)
msg = "%.2f"% (round(MAE, 2))
```

```
print('MAE =', msg)
R2 = metrics.r2_score(y_test, prediction)
msg = "%.2f"% (round(R2, 2))
print('R2 =', msg)
```

In [ ]:

```
# Feature adjustment
```

In [ ]:

```
#view correlation of features
f = plt.figure()
plt.matshow(df.corr(), fignum=f.number)
plt.xticks(range(df.shape[1]), df.columns, fontsize=14, rotation=45)
plt.yticks(range(df.shape[1]), df.columns, fontsize=14)
cb = plt.colorbar()
cb.ax.tick_params(labelsize=14)

plt.show()
```

In [ ]:
```
#Feature Importance

features = list(X.columns)

#get feature rating
rand_best_features = rand_best.feature_importances_
grid_best_features = grid_best.feature_importances_

#create tuples
rand_feature_importance= sorted(zip(rand_best_features, features), rev
grid_feature_importance= sorted(zip(grid_best_features, features), rev

# #create dataframe
df_grid_feature_importance = pd.DataFrame(grid_feature_importance, col
df_best_features = pd.DataFrame(rand_feature_importance, columns=['ran
df_best_features['grid_importance']=df_grid_feature_importance['grid_b
df_best_features['grid_feature']=df_grid_feature_importance['features'

#View feature importance comparison
df_best_features
```

In [ ]:
```
df_top_four = pd.DataFrame(df, columns=['year', 'weekofyear','ndvi_se'
df_top_four
```

In [ ]:
```
#view correlation of features
f = plt.figure()
plt.matshow(df_top_four.corr(), fignum=f.number)
plt.xticks(range(df_top_four.shape[1]), df_top_four.columns, fontsize=
plt.yticks(range(df_top_four.shape[1]), df_top_four.columns, fontsize=
cb = plt.colorbar()
cb.ax.tick_params(labelsize=14)

plt.show()
```

In [ ]:

In [ ]:
```
#Set X and Y variables
tfY=df_top_four.total_cases
tfX=df_top_four.iloc[:, :-1]
```

In [ ]:
```
#Split
tfx_train, tfx_test, tfy_train, tfy_test = train_test_split(tfX, tfY,
```

In [ ]:
```
#Scale
tfscaler=StandardScaler().fit(tfx_train)
print(tfscaler.scale_)
print("\n")
print(tfscaler.mean_)
```

In [ ]:
```
#fit random forest
tfforest = RandomForestRegressor(n_jobs=-1)
tfforest.fit(tfscaler.transform(tfx_train), tfy_train)
```

In [ ]:
```
#Selected model: Random Forest
print("Random Forest:")

#Evalaute Model
prediction= tfforest.predict(tfscaler.transform(tfx_train))

MSE = metrics.mean_squared_error(tfy_train, prediction)
RMSE = np.sqrt(MSE)
msg = "%.2f" % (round(RMSE, 2))
print('RMSE =', msg)

MAE = metrics.mean_absolute_error(tfy_train, prediction)
msg = "%.2f"% (round(MAE, 2))
print('MAE =', msg)

R2 = metrics.r2_score(tfy_train, prediction)
msg = "%.2f"% (round(R2, 2))
print('R2 =', msg)
```

In [ ]: