In [10]:
```
# Exploration and Modeling of Covid 19
```

In [2]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline
plt.style.use('bmh')
```

In [137]:
```
# Importing the dataset from my PC
# Reading different US states coronavirus information
df = pd.read_csv("/Users/local/Desktop/Covid_Folder/owid-covid-data.csv")
df.head()
```

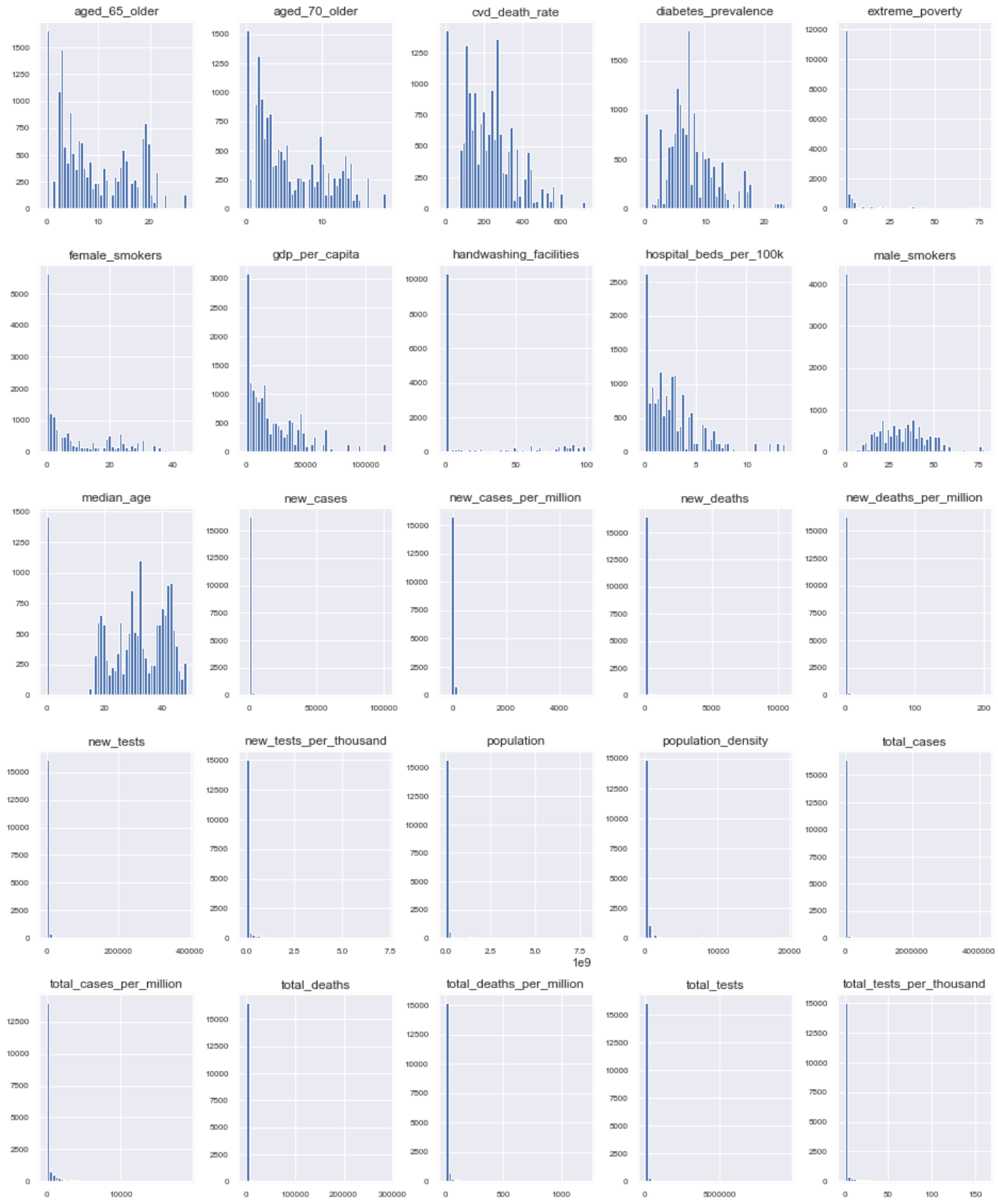
	iso_code	location	date	total_cases	new_cases	total_deaths	new_deaths	total_cases_per_million	new_cases_per_millior
0	ABW	Aruba	2020-03-13	2	2	0	0	18.733	18.733
1	ABW	Aruba	2020-03-20	4	2	0	0	37.465	18.733
2	ABW	Aruba	2020-03-24	12	8	0	0	112.395	74.930
3	ABW	Aruba	2020-03-25	17	5	0	0	159.227	46.831
4	ABW	Aruba	2020-03-26	19	2	0	0	177.959	18.733

In [138]:
```
# Pandas option to display maximum number of rows.
# Describe the dataset
pd.set_option("display.max_rows", None, "display.max_columns", None)
#print("Number of NAN values in the dataset:",df.isnull().sum())
# Filling in missing values with 0 instead of deleting the whole rows.
df=df.fillna(0)
df.describe()
```

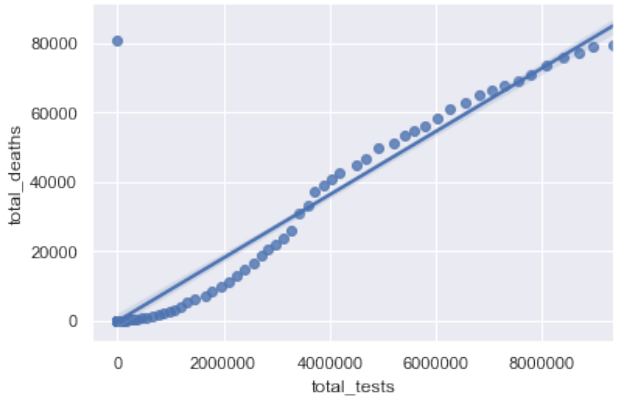
	total_cases	new_cases	total_deaths	new_deaths	total_cases_per_million	new_cases_per_million	total_deaths_per_m
count	1.680100e+04	16801.000000	16801.000000	16801.000000	16801.000000	16801.000000	16801.000000
mean	1.353186e+04	492.493661	898.069222	34.017023	385.188978	12.340580	16.597701
std	1.415289e+05	4524.373965	9877.320053	334.552798	1249.929573	64.845631	80.809005
min	0.000000e+00	-2461.000000	0.000000	0.000000	0.000000	-265.189000	0.000000
25%	3.000000e+00	0.000000	0.000000	0.000000	0.151000	0.000000	0.000000
50%	5.800000e+01	2.000000	1.000000	0.000000	12.201000	0.137000	0.026000
75%	7.950000e+02	36.000000	16.000000	1.000000	183.213000	4.569000	2.430000
max	4.137193e+06	101533.000000	285760.000000	10520.000000	18769.521000	4944.376000	1208.085000

In [139]:
```
# Removing all the non integer values from the dataset
df_num = df.select_dtypes(include= ['float64', 'int64'])
```

In [140]:
```
# Univariate analysis of the dataset
df_num = df_num.hist(figsize=(16, 20), bins=50, xlabelsize=8, ylabelsize=8);
```

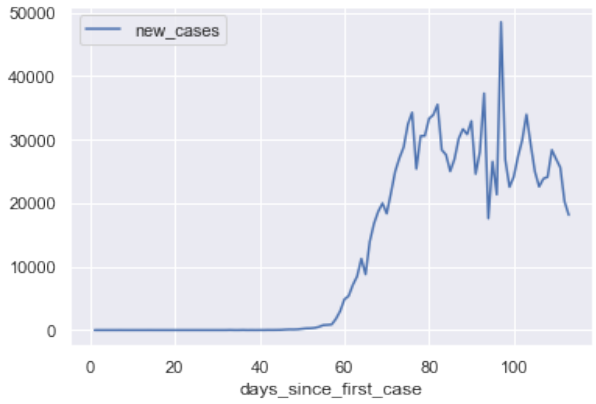


In [141]:
```
# Bivariate analysis between total tests and total deaths
sns.regplot(x=df_us['total_tests'],y=df_us['total_deaths']);
```



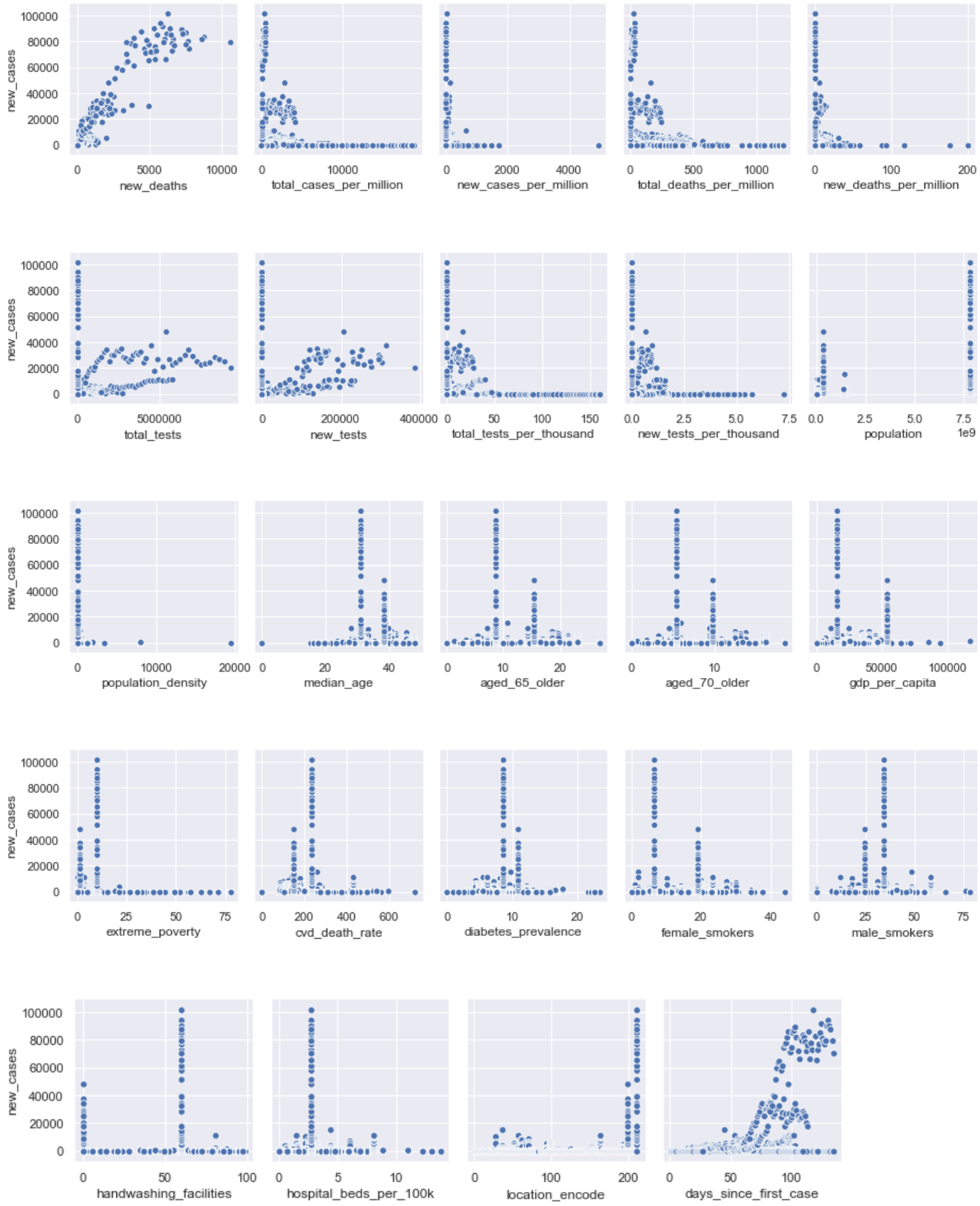
In [142]:
```
# Function to convert date string to "number of days since the first case has been detecte
d"
# Create a list to add new column to the dataset to store "Days" feature
days_list = []
# Shadow variables
location_tracker = ""
location_encode = 0
num_days = 0
for i in df.index:
    if df.at[i,'location']!=location_tracker:
            # Code to encode the country names
            location_tracker = df.at[i,'location']
            location_encode = location_encode + 1
            df.at[i,'location_encode'] = location_encode
            # Num_days code
            num_days = 0
            previous_cases = 0
            if df.at[i,'total_cases'] > 0:
                num_days = num_days + 1
                previous_cases = 1
                df.at[i, 'days_since_first_case'] = num_days
    else:
        df.at[i,'location_encode'] = location_encode
        if df.at[i,'total_cases'] > 0 or previous_cases == 1:
                num_days = num_days + 1
                previous_cases = 1
                df.at[i, 'days_since_first_case'] = num_days
```

In [143]:
```
df = df.dropna()
#Creating a new dataframe as a place holder for further time series analysis
df_main_X = df
# New data frame df_us to explore US cases
df_us=df.loc[df['location'] == "United States"]
df_us.plot(y='new_cases', x='days_since_first_case');
```

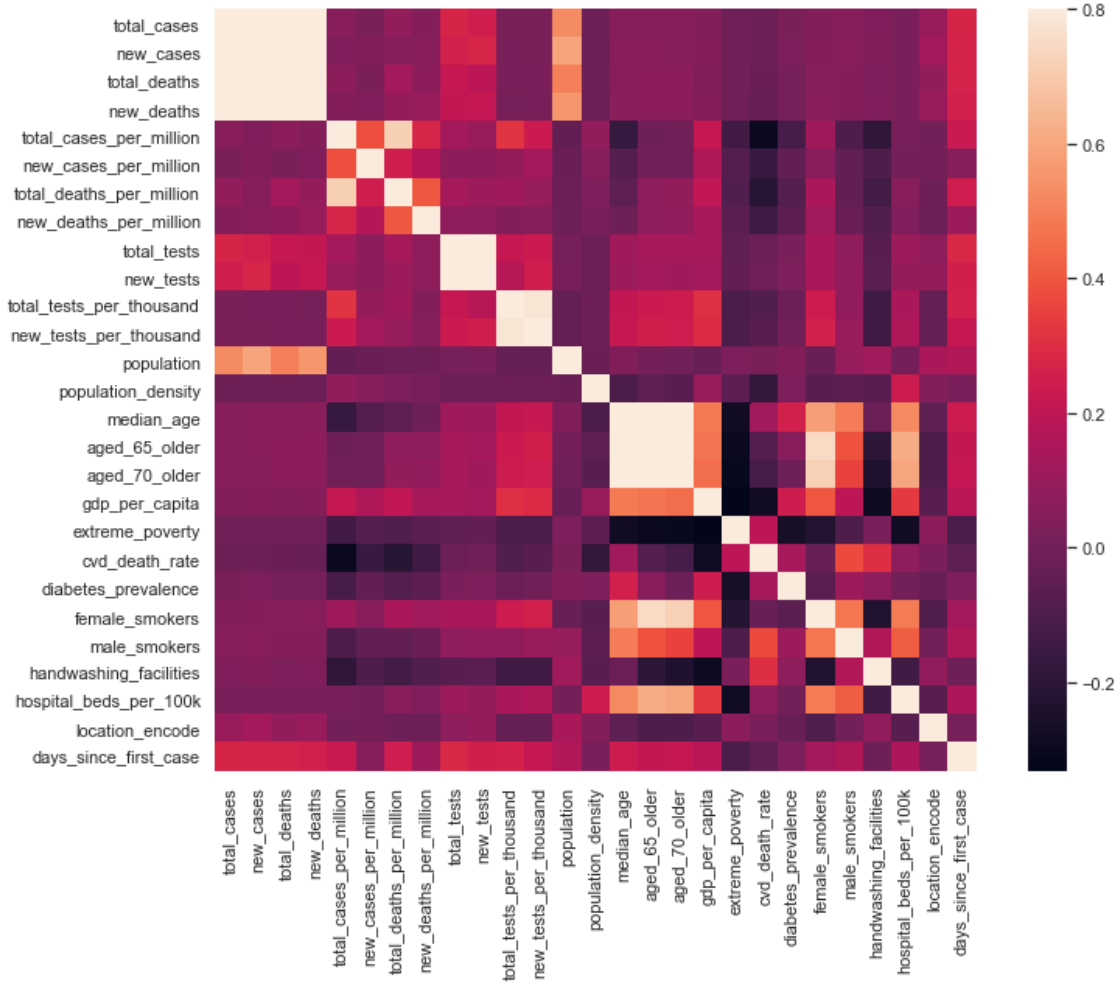


In [25]:
```
# As part of dataset cleanup, remove any negitive values in new cases column
df['new_cases'].describe()
df = df.query("~(new_cases < 0)")
```

In [26]:
```
# Multivariate Analysis of the dataset
sns.set()
df_num = df.select_dtypes(include= ['float64', 'int64'])
for i in range(3, len(df_num.columns), 5):
    sns.pairplot(data = df_num, x_vars=df_num.columns[i:i+5], y_vars = ['new_cases'])
```


new_cases
new_deaths
total_cases_per_million
new_cases_per_million
total_deaths_per_million
new_deaths_per_million
total_tests
new_tests
total_tests_per_thousand
new_tests_per_thousand
population
1e9
population_density
median_age
aged_65_older
aged_70_older
gdp_per_capita
extreme_poverty
cvd_death_rate
diabetes_prevalence
female_smokers
male_smokers
handwashing_facilities
hospital_beds_per_100k
location_encode
days_since_first_case

In [27]:
```
# Correlation matrix to find out the important variables
corrmat = df_num.corr()
f, ax = plt.subplots(figsize=(12, 9))
sns.heatmap(corrmat, vmax=0.8, square=True);
```



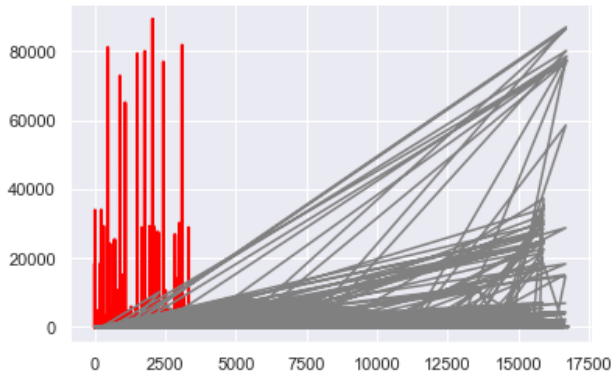
In [28]:
```
# Implement the baseline K nearest neighbors algorithm
from sklearn import neighbors, datasets, preprocessing
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import mean_absolute_error, mean_squared_error, mean_squared_log_erro
r
```

In [29]:
```
X, y = df_num.loc[:, df_num.columns != 'new_cases'],df_num['new_cases']
X_train,X_test,y_train,y_test=train_test_split(X,y,shuffle=True,random_state=10)
scaler = preprocessing.StandardScaler().fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
knn = neighbors.KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
print ("Accuracy:",accuracy_score(y_test, y_pred))
print("MAE:",mean_absolute_error(y_test, y_pred))
print("MSE:",mean_squared_error(y_test, y_pred))
print("log error:",np.sqrt(mean_squared_log_error(y_test, y_pred)))
```

```
Accuracy: 0.30430984274898076
MAE: 158.73966220151428
MSE: 1365195.4571927781
log error: 1.2143243733544964
```

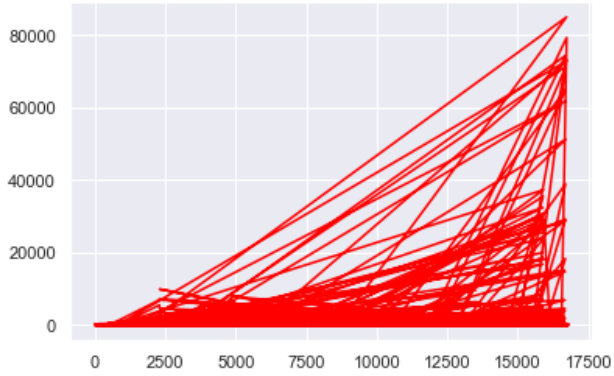
In [32]:
```
# Function to calculate the result metric root mean squared log error
def rmsle(ytrue, ypred):
    return np.sqrt(mean_squared_log_error(ytrue, ypred))
```

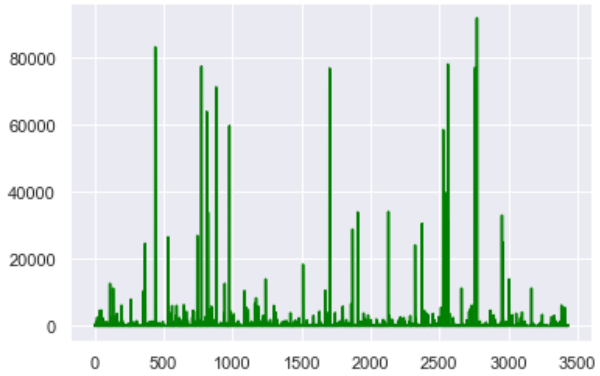
In [33]:
```
# Not doing well on both lower or higher values.
df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
plt.plot( y_pred, color='red', linewidth=2)
plt.plot(y_test,  color='gray')
plt.show()
```



In [34]:
```
# Try different algorithms with subset important features
df_truck = df_num[['total_tests','population','median_age','aged_65_older','aged_70_older'
,'gdp_per_capita','location_encode','days_since_first_case','new_cases']]
df_truck.describe()
X, y = df_num.loc[:, df_num.columns != 'new_cases'],df_num['new_cases']
```

In [41]:
```
X_train,X_test,y_train,y_test=train_test_split(X,y,shuffle=True)
scaler = preprocessing.StandardScaler().fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
from sklearn import linear_model
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestClassifier
# Classifier Model
clf = RandomForestClassifier(n_estimators=50)
clf = clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
# Linear Regression Model
regressor = LinearRegression()
regressor.fit(X_train, y_train)
y_pred_reg = regressor.predict(X_test)
# Lasso Model
reg = linear_model.Lasso(alpha=0.1)
reg.fit(X_train, y_train)
y_pred_Las = reg.predict(X_test)
plt.plot(y_test, color='red')
plt.show()
plt.plot(y_pred, color='green')
plt.show()
print("MAE of Classifier:",mean_absolute_error(y_test, y_pred))
print("MSE of Classifier:",mean_squared_error(y_test, y_pred))
print("log error of Classifier:",rmsle(y_test, y_pred))
print("MAE of Linear Regression:",mean_absolute_error(y_test, y_pred_reg))
print("MSE of Linear Regression:",mean_squared_error(y_test, y_pred_reg))
df_temp = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred_reg})
df_temp = df_temp.query("~(Predicted < 0)")
print("log error of Linear Regression:",rmsle(df_temp['Actual'],df_temp['Predicted']))
print("MAE of Lasso:",mean_absolute_error(y_test, y_pred_Las))
print("MSE of Lasso:",mean_squared_error(y_test, y_pred_Las))
df_temp = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred_Las})
df_temp = df_temp.query("~(Predicted < 0)")
print("log error of Lasso:",rmsle(df_temp['Actual'],df_temp['Predicted']))
```






```
MAE of Classifier: 99.67821782178218
MSE of Classifier: 1483195.135993011
log error of Classifier: 0.5038526809155315
MAE of Linear Regression: 258.08190958720536
MSE of Linear Regression: 1164962.3836631647
log error of Linear Regression: 2.391942784957338
MAE of Lasso: 257.4192710486816
MSE of Lasso: 1164901.731802805
log error of Lasso: 2.382607438678546
```

In [42]:
```
# Random implementations of the COVID dataset yeilded poor results. Moving to Time Series implementation
```

In [98]:
```
!pip install lightgbm
import pandas as pd
import numpy as np
import random
%matplotlib inline
from sklearn.metrics import mean_squared_log_error
from sklearn.ensemble import RandomForestRegressor
from lightgbm import LGBMRegressor
```

```
Looking in indexes: https://pypi.python.org/simple, https://pypi.apple.com/simple
Requirement already satisfied: lightgbm in /Users/local/opt/anaconda3/lib/python3.7/site-packages (2.3.1)
Requirement already satisfied: scikit-learn in /Users/local/opt/anaconda3/lib/python3.7/site-packages (from lightgbm) (0.22.1)
Requirement already satisfied: scipy in /Users/local/opt/anaconda3/lib/python3.7/site-packages (from lightgbm) (1.4.1)
Requirement already satisfied: numpy in /Users/local/opt/anaconda3/lib/python3.7/site-packages (from lightgbm) (1.18.1)
Requirement already satisfied: joblib>=0.11 in /Users/local/opt/anaconda3/lib/python3.7/site-packages (from scikit-learn->lightgbm)
(0.14.1)
```

In [144]:
```
# Cleaning the dataset further to remove the datapoints of the World as they miss the rest
# of the features.
df_main = df_main_X
indexNames = df_main[ df_main['location'] == 'World' ].index
df_main.drop(indexNames , inplace=True)
```

In [145]:
```
# Further data cleaning to group the dataset by the location and shuffle the data by group
s.
df_main=df_main.drop(['iso_code', 'location','date'], axis=1)
df_main = df_main.query("~(new_cases < 0)")
groups = [df_main for _, df_main in df_main.groupby('location_encode')]
random.seed(1)
random.shuffle(groups)
df_main = pd.concat(groups).reset_index(drop=True)
#df_main.to_csv('/Users/local/Desktop/groupby.csv')
# Below we can see how unbalenced the data set is between cases > 8000
seriesObj = df_main.apply(lambda x: True if x['new_cases'] > 8000 else False , axis=1)
numOfRows = len(seriesObj[seriesObj == True].index)
numOfCol  = len(seriesObj[seriesObj == False].index)
print('Number of Rows in dataframe in which new_class > 8000 : ', numOfRows)
print('Number of Rows in dataframe in which new_class < 8000 : ', numOfCol)
#series_name_Obj = df_main.apply(lambda x:x['location'] if x['new_cases'] > 8000 else Fals
e, axis=1)
#print(series_name_Obj)
```

```
Number of Rows in dataframe in which new_class > 8000 :  72
Number of Rows in dataframe in which new_class < 8000 :  13530
```

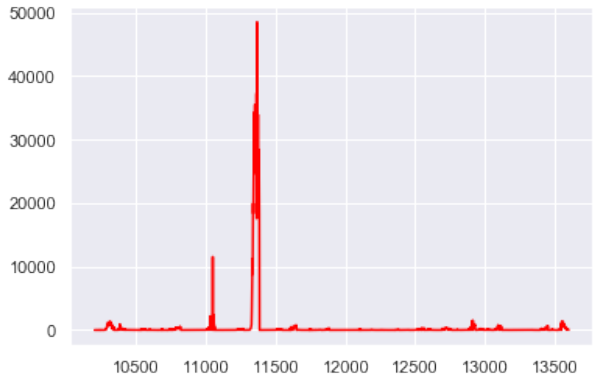
In [146]:
```
# Adding/creating new features to help with the predictions
df_main_2 = df_main.copy()
df_main_2 = df_main_2.select_dtypes(include= ['float64', 'int64'])
df_main_2 = df_main_2.dropna()
```

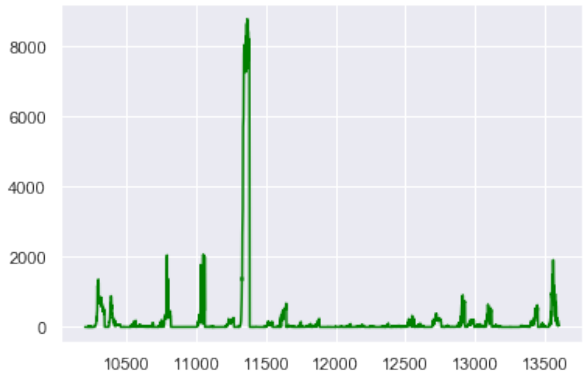
In [147]:
```
# Notice the shuffle being false as we continue with our Time series implementations
X, y = df_main_2.loc[:, df_main_2.columns != 'new_cases'],df_main_2['new_cases']
X_train,X_test,y_train,y_test=train_test_split(X,y,shuffle=False)
```

In [148]:
```
# Random Forest Regressor Algorithm
mdl = RandomForestRegressor(n_estimators=1000, n_jobs=-1, random_state=0)
mdl.fit(X_train, y_train)
p = mdl.predict(X_test)
error = rmsle(y_test, p)
print("log error:",error)
```

```
log error: 0.4901838912346516
```

In [149]:
```
df_comp = pd.DataFrame({'Actual': y_test, 'Predicted': p})
#df_comp=df_comp.iloc[2000:3000]
plt.plot(df_comp['Actual'], color='red')
plt.show()
plt.plot(df_comp['Predicted'], color='green')
plt.show()
```





In [150]:
```
# Adding additional features to help with the prediction
df_main_2['Last_Day_Deaths'] = df_main_2.groupby(['location_encode'])['new_deaths'].shift(
)
df_main_2['Last_Day_Death_Diff'] = df_main_2.groupby(['location_encode'])['Last_Day_Deaths
'].diff()
df_main_2['Last_Day_Cases'] = df_main_2.groupby(['location_encode'])['new_cases'].shift()
df_main_2['Last_Day_Case_Diff'] = df_main_2.groupby(['location_encode'])['Last_Day_Cases']
.diff()
df_main_2 = df_main_2.dropna()
```

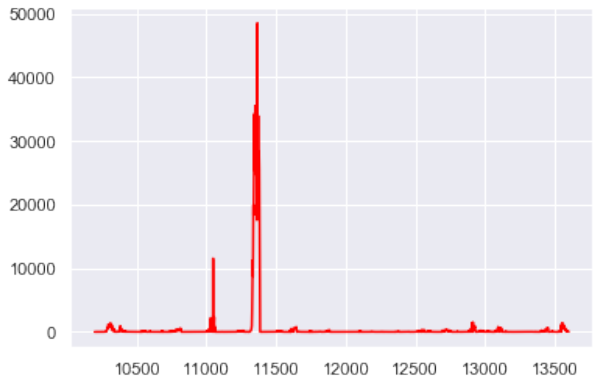
In [151]:
```
X, y = df_main_2.loc[:, df_main_2.columns != 'new_cases'],df_main_2['new_cases']
X_train,X_test,y_train,y_test=train_test_split(X,y,shuffle=False)
mdl = RandomForestRegressor(n_estimators=1000, n_jobs=-1, random_state=0)
mdl.fit(X_train, y_train)
p = mdl.predict(X_test)
error = rmsle(y_test, p)
print("log error after new features:",error)
```

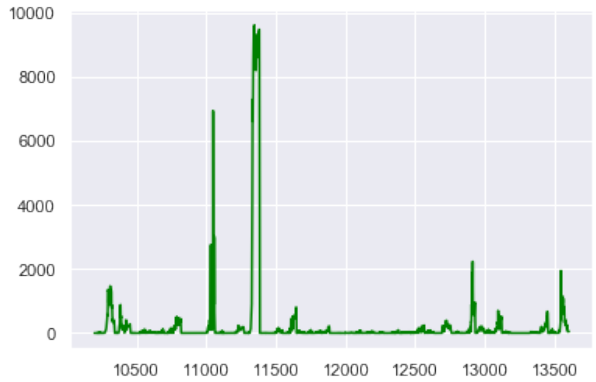
```
log error after new features: 0.5806170256010894
```

In [125]:
```
# The same algorithm but we take the exponential of the p value to be on the same scale
mdl = RandomForestRegressor(n_estimators=1000, n_jobs=-1, random_state=0)
mdl.fit(X_train, np.log1p(y_train))
p = np.expm1(mdl.predict(X_test))
error = rmsle(y_test, p)
print("log error:",error)
```

```
log error: 0.38644041111049854
```

In [152]:
```
df_comp = pd.DataFrame({'Actual': y_test, 'Predicted': p})
plt.plot(df_comp['Actual'], color='red')
plt.show()
plt.plot(df_comp['Predicted'], color='green')
plt.show()
```





In [153]:
```
# LGBM Regressor with Time series implementation
mdl = LGBMRegressor(n_estimators=1000, learning_rate=0.01)
mdl.fit(X_train, np.log1p(y_train))
p = np.expm1(mdl.predict(X_test))
```

In [154]:
```
df_comp = pd.DataFrame({'Actual': y_test, 'Predicted': p})
df_comp = df_comp.query("~(Predicted < 0)")
error = rmsle(df_comp['Actual'],df_comp['Predicted'])
print("log error:", error)
```

```
log error: 0.24316492925822691
```

In [155]:
```
# Due to the unbalance nature caused by the US case numbers, let us remove and see what
# the model will look like
indexNames = df_main_2[ df_main_2['location_encode'] == 198 ].index
df_main_2.drop(indexNames , inplace=True)
X, y = df_main_2.loc[:, df_main_2.columns != 'new_cases'],df_main_2['new_cases']
X_train,X_test,y_train,y_test=train_test_split(X,y,shuffle=False)
mdl = LGBMRegressor(n_estimators=1000, learning_rate=0.01)
mdl.fit(X_train, np.log1p(y_train))
p = np.expm1(mdl.predict(X_test))
df_comp = pd.DataFrame({'Actual': y_test, 'Predicted': p})
df_comp = df_comp.query("~(Predicted < 0)")
error = rmsle(df_comp['Actual'],df_comp['Predicted'])
print("log error:", error)
```

```
log error: 0.19315616682622
```

In [ ]: