In [411…

```
import os
import zipfile
import kaggle
from statsmodels.tsa import arima

DATA_PATH = r"datasets\vaccines"
IMAGES_PATH = r"images"
kaggle.api.authenticate()
kaggle.api.dataset_download_files('gpreda/covid-world-vaccination-progress')


def fetch_data(data_path=DATA_PATH):
    if not os.path.isdir(data_path):
        os.mkdir(data_path)
    zf = zipfile.ZipFile('covid-world-vaccination-progress.zip')
    zf.extractall(DATA_PATH)


fetch_data()
```

In [412…

```
# Retreive the data from the file, fill the null values with 0 and convert date string to date

import pandas as pd
import seaborn as sns
import plotly.express as px
import plotly.graph_objects as go
import matplotlib.pyplot as plt
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score
from sklearn import tree
from statsmodels.tsa.stattools import adfuller
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
import statsmodels.api as sm
from statsmodels.tsa.arima.model import ARIMA
import pmdarima as pm
```

```
df = pd.read_csv(r"datasets\vaccines\country_vaccinations.csv")
df['date'] = pd.to_datetime(df['date'], format='%Y-%m-%d')
df = df.fillna(0)
df
```

Out[412…

	country	iso_code	date	total_vaccinations	people_vaccinated	people_fully_vaccinated	daily_vaccinations_raw	daily_vaccinations	total_v
0	Afghanistan	AFG	2021-02-22	0.0	0.0	0.0	0.0	0.0	
1	Afghanistan	AFG	2021-02-23	0.0	0.0	0.0	0.0	1367.0	
2	Afghanistan	AFG	2021-02-24	0.0	0.0	0.0	0.0	1367.0	
3	Afghanistan	AFG	2021-02-25	0.0	0.0	0.0	0.0	1367.0	
4	Afghanistan	AFG	2021-02-26	0.0	0.0	0.0	0.0	1367.0	
...	...	...	...	...	...	...	...	...	
11359	Zimbabwe	ZWE	2021-04-08	193677.0	166543.0	27134.0	14260.0	12624.0	
11360	Zimbabwe	ZWE	2021-04-09	206205.0	178237.0	27968.0	12528.0	11636.0	
11361	Zimbabwe	ZWE	2021-04-10	222733.0	193936.0	28797.0	16528.0	12831.0	
11362	Zimbabwe	ZWE	2021-04-11	223492.0	194594.0	28898.0	759.0	12085.0	
11363	Zimbabwe	ZWE	2021-04-12	234579.0	205275.0	29304.0	11087.0	12724.0	

11364 rows × 15 columns

Project1 - Second Deliverable

Data Analysis

We will begin by checking the level of correlation between the data fields from the table. This helps us find redundant columns, which are highly correlated to columns which are of interests to us. Correlation describes how variables are connected to each other. That is, how changing the current state of a variable impacts another one. High correlation (of 1 or close to that value) may disturb the creation of prediction models and interfere with the output:

1. In the case of **Linear Regression**, for example, slight changes of highly correlated data can result in erratic responses over small data changes.
2. When it comes to **Random Forests**, while they are great at detecting interactions between different features, correlated data might hide these interactions.

Below is the heat map of correlation of the data from the table.

In [413…

```
fig,ax = plt.subplots(figsize=(10,6))
sns.heatmap(df.corr(), annot=True, cmap="Blues")
ax.set_title('correlations')
plt.show()
```

As it can be seen from the heat map, there are several columns which have a slightly high degree of correlation:

1. **daily_vaccinations** - total_vaccinations
2. **daily_vaccinations_raw** - total_vaccinations and daily_vaccinations
3. **total_vaccinations_per_hundred** - people_vaccinated_per_hundred

We can conclude that we can drop several columns, as below:

In [414…

```
df.drop(columns=['total_vaccinations', 'people_vaccinated_per_hundred', 'daily_vaccinations_raw'])
print('Size of data frame: ', df.size)
print('NaN values')
df.isna().sum()
```

```
Size of data frame:  170460
NaN values
```

Out[414…

```
country                              0
iso_code                             0
date                                 0
total_vaccinations                   0
people_vaccinated                    0
people_fully_vaccinated              0
daily_vaccinations_raw               0
daily_vaccinations                   0
total_vaccinations_per_hundred       0
people_vaccinated_per_hundred        0
people_fully_vaccinated_per_hundred  0
daily_vaccinations_per_million       0
vaccines                             0
source_name                          0
source_website                       0
dtype: int64
```

In [ ]:

Since we are interested in the daily vaccination progress, we grouped the data by date and summed the values of **daily_vaccinations** to get the total number of daily vaccinations.

However, other models did not accept the date format as training and testing data, therefore we introduced a new field, called **date_difference**. This field is very similar to the classic index, performing a count across the dataset. Such, the first entry has the date_difference 0, and the last 119.

In [415…

```
cov_data = df.groupby('date')['daily_vaccinations'].sum().reset_index()

cov_data['date_difference'] = np.nan
```

```
count = 0
for index, row in cov_data.iterrows():
    cov_data.loc[index, 'date_difference'] = count
    count += 1

cov_data
```

Out[415…

	date	daily_vaccinations	date_difference
0	2020-12-14	0.0	0.0
1	2020-12-15	718.0	1.0
2	2020-12-16	192366.0	2.0
3	2020-12-17	193256.0	3.0
4	2020-12-18	193649.0	4.0
...	...	...	...
115	2021-04-08	16630277.0	115.0
116	2021-04-09	16785255.0	116.0
117	2021-04-10	17340369.0	117.0
118	2021-04-11	17298612.0	118.0
119	2021-04-12	16004359.0	119.0

120 rows × 3 columns

This is the general form of model fitting and prediction, along with a helper function for printing the decision trees below. We have also printed the predictions for day 100, for exactly one year later, and the regression score. The latter will be discussed further later.

In [416…

```
def prediction(model, x_train, y_train, x_test, y_test, title):
    model.fit(x_train, y_train)
    pred = model.predict(x_test)

    plt.title(title)
    plt.plot(x_train, y_train, color='black')
    plt.plot(x_test, y_test, color='green')
    plt.plot(x_test, pred, color='blue')
```

```
    plt.show()
    print('prediction for day 100: ', int(model.predict([[100]])))
    print('prediction after a year: ', int(model.predict([[365]])))
    print('regression score: ', r2_score(y_test, pred))
    return r2_score(y_test, pred)
```

In [417…

```
def print_tree(tree_name):
    fig = plt.figure(figsize=(25,20))
    fig = tree.plot_tree(tree_name, filled=True)
```

We will begin to analyze the models by testing their prediction capabilities on the same data that they were trained on.

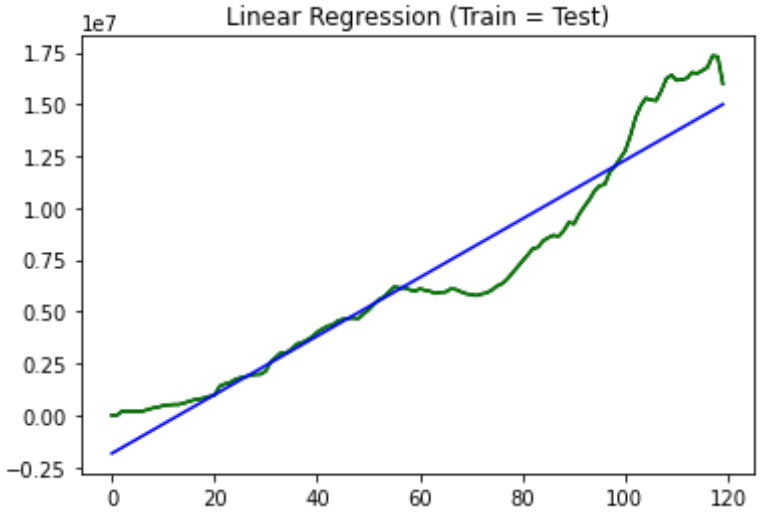
In [418…

```
x_train = x_test = cov_data[['date_difference']]
y_train = y_test = cov_data['daily_vaccinations']
```

Linear Regression

In [419…

```
lreg = LinearRegression()
lreg_score1 = prediction(lreg, x_train, y_train, x_test, y_test, 'Linear Regression (Train = Test)')
```



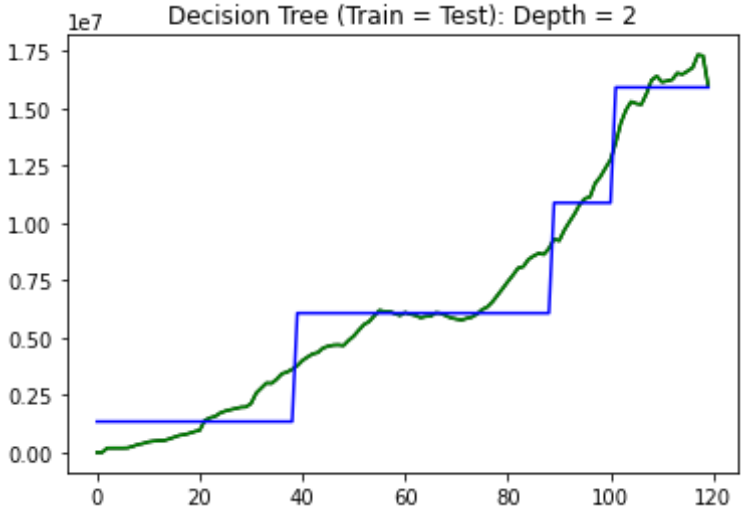
```
prediction for day 100:  12311856
prediction after a year:  49825001
regression score:  0.9239323895658387
```

Decision Tree

max depth = 2

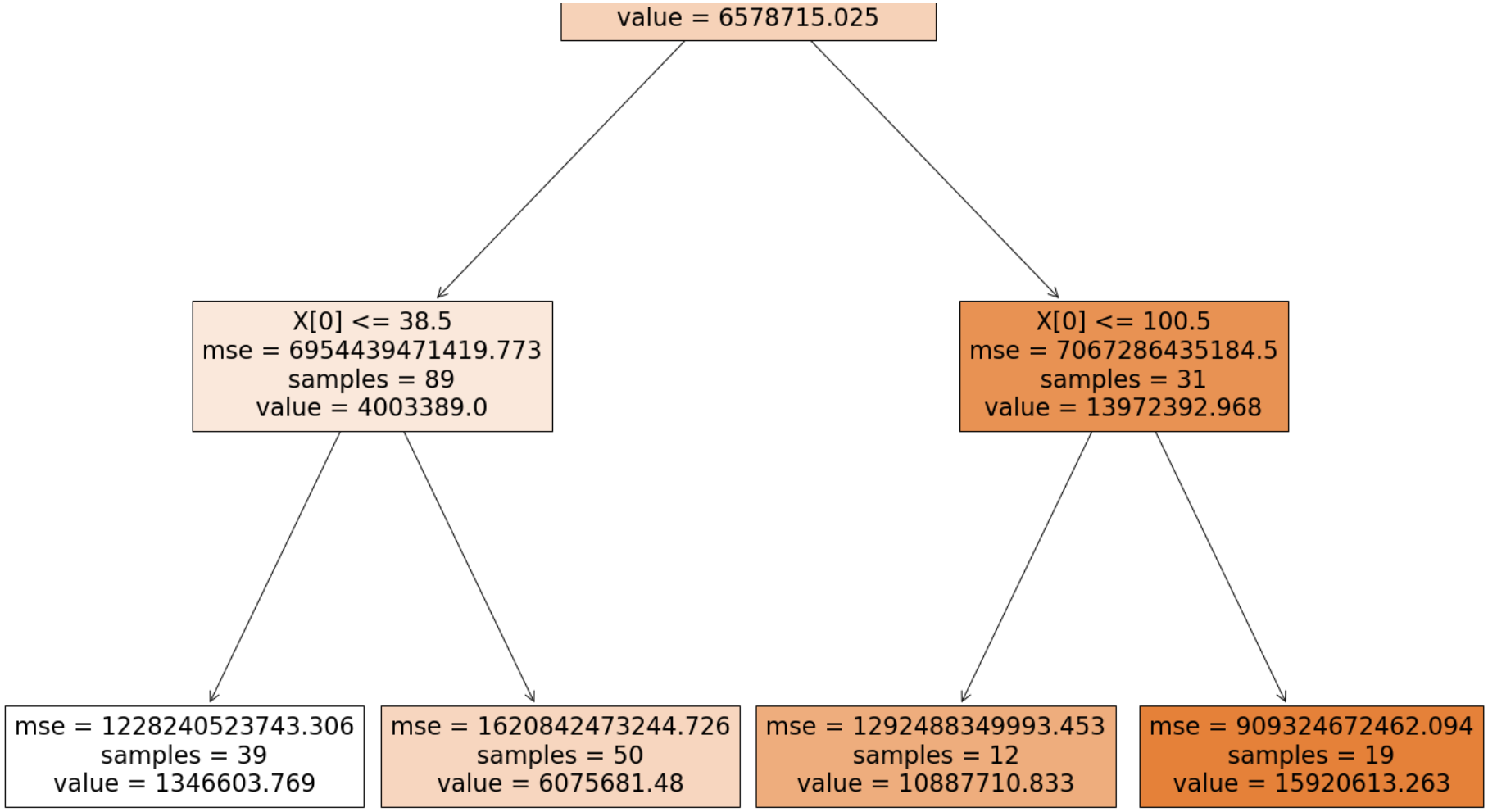
In [420…

```
dtree1 = DecisionTreeRegressor(max_depth=2)
dtree_score11 = prediction(dtree1, x_train, y_train, x_test, y_test, 'Decision Tree (Train = Test): Depth = 2')
print_tree(dtree1)
```




```
prediction for day 100:  10887710
prediction after a year:  15920613
regression score:  0.9482125342504327
```

X[0] <= 88.5
mse = 26024722830137.414
samples = 120

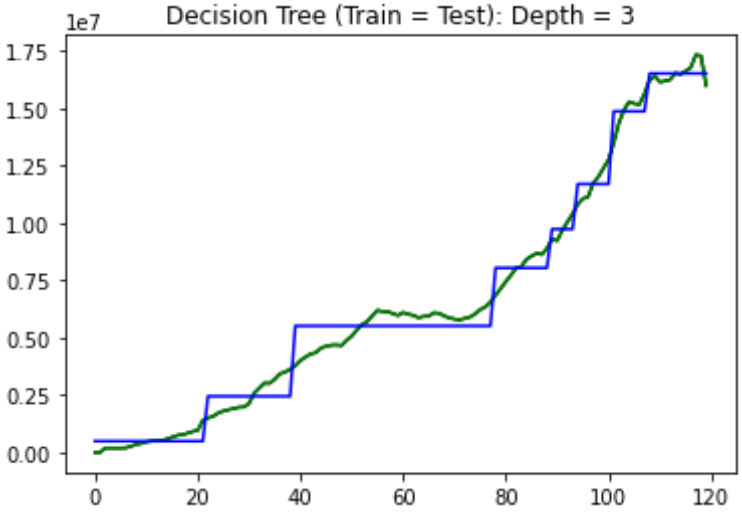


max depth = 3

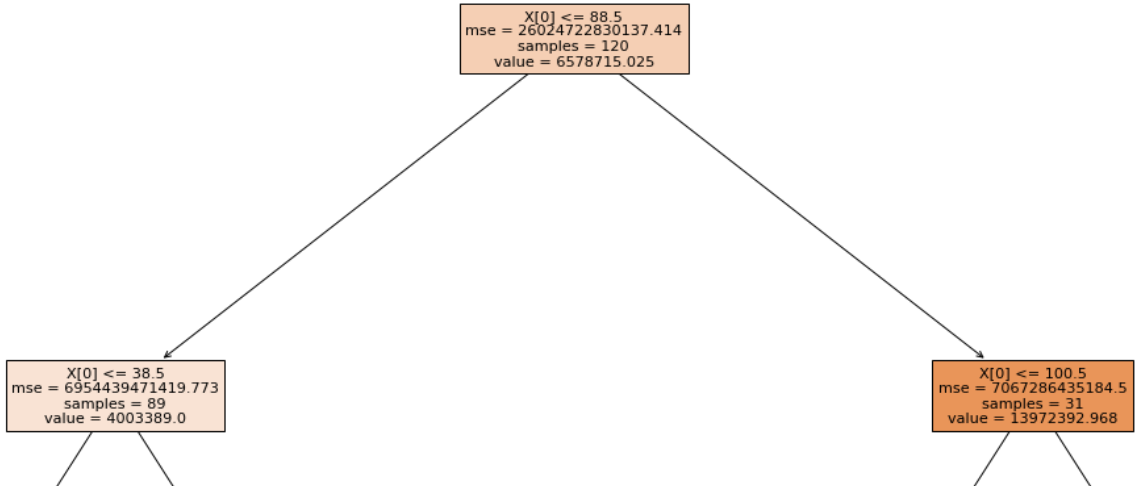
In [421…

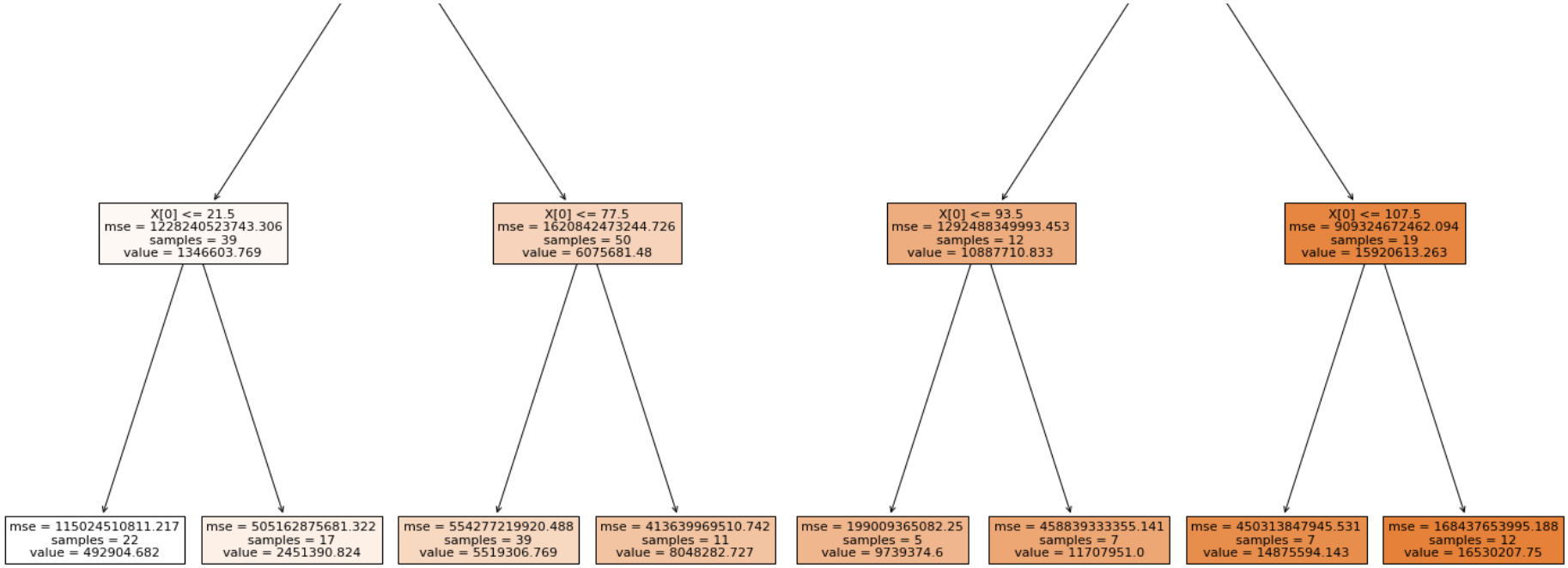
```
dtree2 = DecisionTreeRegressor(max_depth=3)
```

```
dtree_score21 = prediction(dtree2, x_train, y_train, x_test, y_test, 'Decision Tree (Train = Test): Depth = 3')
print_tree(dtree2)
```




```
prediction for day 100:  11707951
prediction after a year:  16530207
regression score:  0.9850573079384972
```



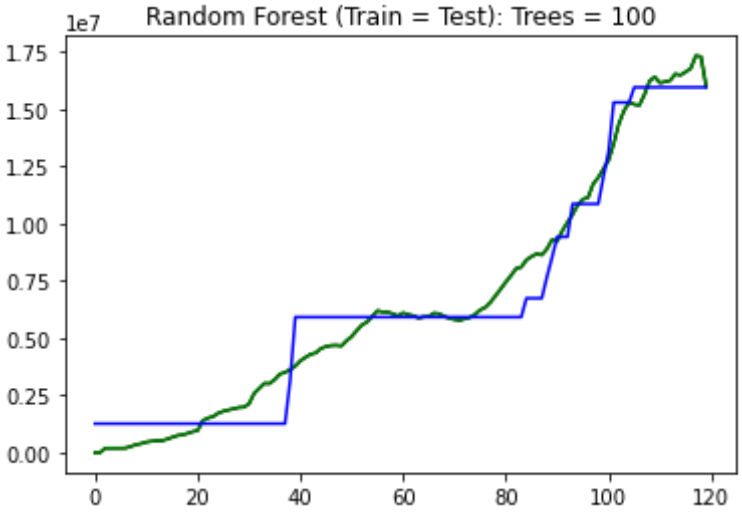


Random Forest

number of trees = 5, max depth = 2

In [422…

```
rf1 = RandomForestRegressor(n_estimators=5, max_depth=2)
rf_score1 = prediction(rf1, x_train, y_train, x_test, y_test, 'Random Forest (Train = Test): Trees = 100')
```

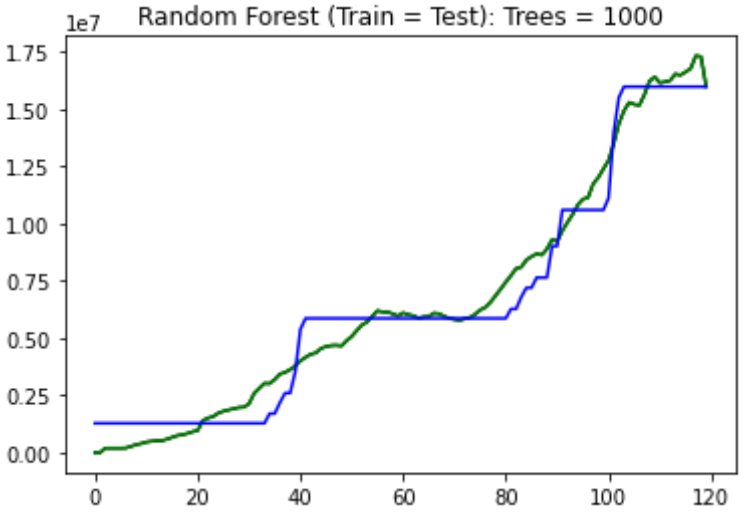



```
prediction for day 100:  13141776
prediction after a year:  15961338
regression score:  0.9621711161136298
```

number of trees = 10, max depth = 2

In [423…

```
rf2 = RandomForestRegressor(n_estimators=10, max_depth=2)
rf_score2 = prediction(rf2, x_train, y_train, x_test, y_test, 'Random Forest (Train = Test): Trees = 1000')
```




```
prediction for day 100:  11121334
prediction after a year:  15987372
regression score:  0.9697827252846372
```

Next, we will analyze the forecasting capabilities of the models. We will train the model on the first 90 entries from the dataframe and then perform predictions / forecats on the rest of the entries.

It is important to note that since the time series is related to the anti-covid vaccination progress, the the data is rather scarce. A larger amount of training data would have meant a more accurate prediction, but that is not currently available.

In [424…

```
x_train = cov_data[['date_difference']][:90]
y_train = cov_data['daily_vaccinations'][:90]

x_test = cov_data[['date_difference']][90:]
y_test = cov_data[['daily_vaccinations']][90:]
```

Linear Regression

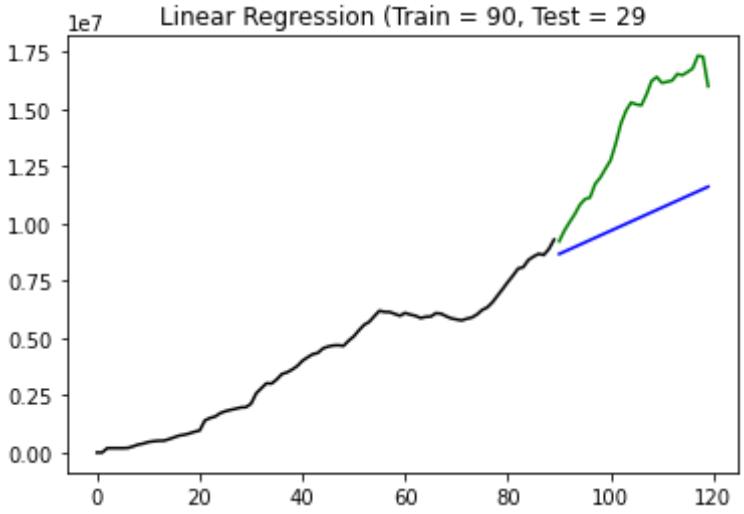
Training Set <> Testing Set

We can notice the fact that the prediction is not accurate. That probably results from the shape of the curve up until somewhere around day 75. We can notice that there was a sudden increase of daily vaccination numbers, which does not belong to the training set.

If we take a look at the shape of the training set (black) and the resulting prediction, we can notice that they are quite similar. The model could not have predicted that vaccination spike.

In [425…

```
lreg_score2 = prediction(lreg, x_train, y_train, x_test, y_test, 'Linear Regression (Train = 90, Test = 29')
```




```
prediction for day 100:  9690117
prediction after a year:  36561741
regression score:  -1.8819501865861508
```

Decision Tree

max depth = 3

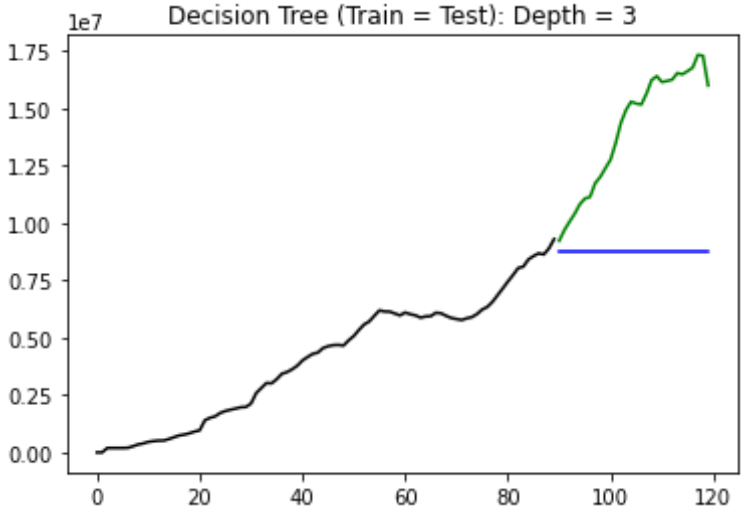
Training Set <> Testing Set

In terms of forecasting this model, decision trees are not the ideal choice either. Decision trees are great for prediction and approximation on the same entries that they were trained on. However, when it comes to forecasting, the model seems to remain constant at the last trained maximum

value throughout the entire testing model.

In [426…

```
dtree2 = DecisionTreeRegressor(max_depth=3)
dtree_score12 = prediction(dtree2, x_train, y_train, x_test, y_test, 'Decision Tree (Train = Test): Depth = 3')
```




```
prediction for day 100:  8749826
prediction after a year:  8749826
regression score:  -4.414492061125101
```

ARIMA

In order to perform a full computation of the ARIMA model, we had to take stationarity into consideration. A stationary series is a series whose **mean average** and **standard deviation** remain as unchanged in time as possible. We performed the mean average on a window of 14 data entries (i.e. 14 days).

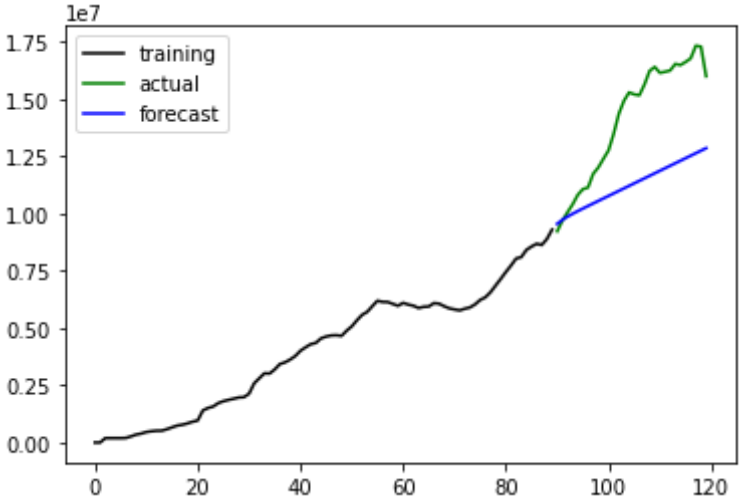
We also performed the **Dickey-Fuller** test to calculate the stationarity of the series. The test is based on the null hypothesis, which can be rejected. If it is not rejcted, then the series is dependent on time and is non-stationary. If it is rejected, however, i results that it is not time-dependent and it is stationary.

We are mostly interested in the p-value. By definition, in order for the null hypothesis to be rejected, the p-value should be < 0.05 (5%).

In [427…

```
train_model = cov_data[:90]
test_model = cov_data[90:]

model = pm.auto_arima(train_model.daily_vaccinations, enforce_stationarity=False, enforce_invertibility=False)
print(model.summary())
model_fit = model.fit(train_model.daily_vaccinations)
fc = model.predict(30)

fc_series = pd.DataFrame({'daily_vaccinations' : fc}, index=test_model.index)

arima_score3 = r2_score(test_model.daily_vaccinations, fc_series.daily_vaccinations)

plt.plot(train_model.daily_vaccinations, label='training', color='black')
plt.plot(test_model.daily_vaccinations, label='actual', color='green')
plt.plot(fc_series.daily_vaccinations, label='forecast', color='blue')
plt.legend(loc='best')
plt.show()
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                   90
Model:               SARIMAX(1, 1, 0)   Log Likelihood               -1158.913
Date:                Thu, 15 Apr 2021   AIC                           2323.827
Time:                        14:10:38   BIC                           2331.293
Sample:                             0   HQIC                          2326.836
                                  - 90
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
intercept   5.811e+04    1.6e+04      3.641      0.000    2.68e+04    8.94e+04
ar.L1          0.4709      0.097      4.837      0.000       0.280       0.662
sigma2      1.206e+10      0.078   1.55e+11      0.000    1.21e+10    1.21e+10
===================================================================================
Ljung-Box (L1) (Q):                   0.34   Jarque-Bera (JB):                10.87
Prob(Q):                              0.56   Prob(JB):                         0.00
Heteroskedasticity (H):               1.78   Skew:                             0.75
Prob(H) (two-sided):                  0.12   Kurtosis:                         3.82
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
[2] Covariance matrix is singular or near-singular, with condition number 2.95e+26. Standard errors may be unstable.
```



In [428…

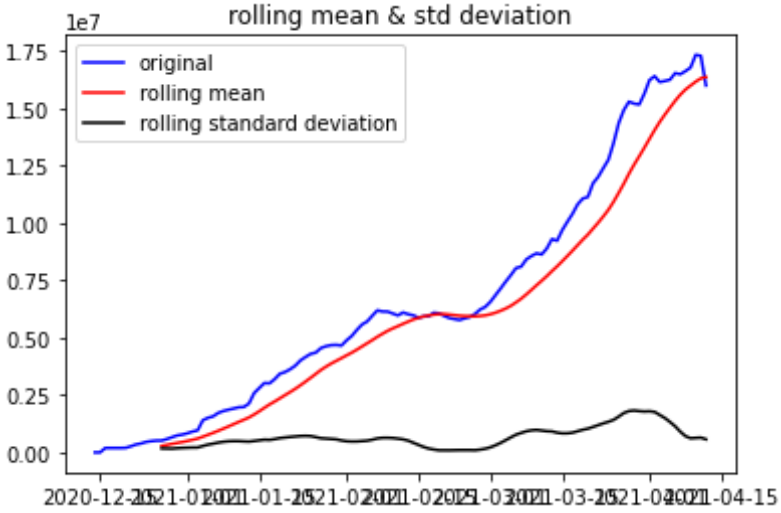
```
def test_stationarity(timeseries):
    moving_average = timeseries.rolling(window=14).mean()
    moving_stddev = timeseries.rolling(window=14).std()

    plt.plot(timeseries, color='blue', label='original')
    plt.plot(moving_average, color='red', label='rolling mean')
    plt.plot(moving_stddev, color='black', label='rolling standard deviation')
    plt.legend(loc='best')
    plt.title('rolling mean & std deviation')
    plt.show(block=False)

    print('Dickey-Fuller Test:')
    dftest = adfuller(timeseries['daily_vaccinations'])
    print('ADF Statistic: %f' %dftest[0])
    print('p-value (should be smaller than 0.05): %s' %dftest[1])
    print('Critical values: ')
    for key, value in dftest[4].items():
            print('\t%s: %.3f' % (key, value))
```

In terms of the initial dataset, we can observe that the rolling mean and the rolling standard deviation are clearly not constant. The p-value is also quite large, therefore the series is non-stationary.

In [429…

```
cov_data_stationarity = cov_data[['date', 'daily_vaccinations']].copy().set_index('date')
```

```
cov_data_stationarity.index = pd.DatetimeIndex(cov_data_stationarity.index.values, freq=cov_data_stationarity.index.i
test_stationarity(cov_data_stationarity)
```




```
Dickey-Fuller Test:
ADF Statistic: -1.589114
p-value (should be smaller than 0.05): 0.4890598293984345
Critical values:
        1%: -3.494
        5%: -2.889
        10%: -2.582
```

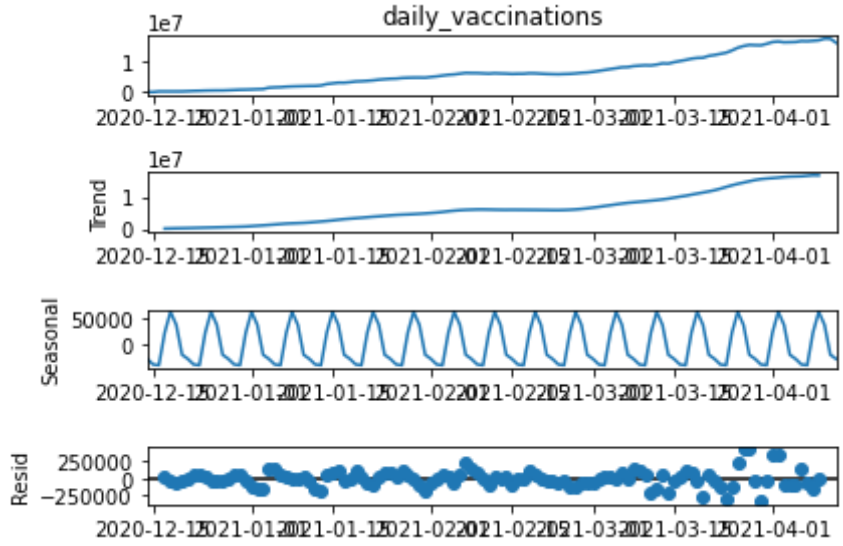
Below is a representation of the seasonal decomposition of the dataset. The decomposition contains three elements: the trend, the seasonal componend and the residual component.

1. trend - increasing / decreasing value of the series
2. seasonality - repeating cycle of the series
3. noise - random variation

For our series, we an notice that the trend is for the most part ascending throughout the entire dataset. The series does have some seasonality. In terms of noise, we can notice that there have been spikes and sudden changes in values at latter dates.

In [430…

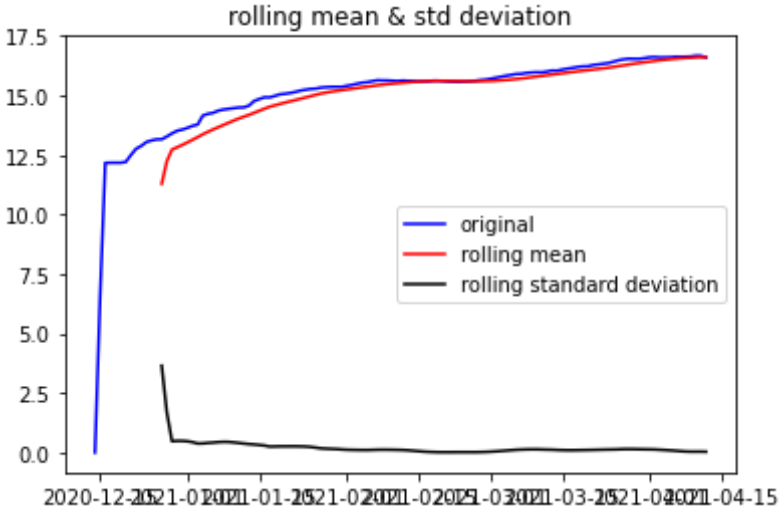
```
sm.tsa.seasonal_decompose(cov_data_stationarity.daily_vaccinations).plot()
plt.show()
```



Logarithmic Transformation

In [431…

```
cov_data_log = np.log(cov_data_stationarity)
cov_data_log.replace([np.inf, -np.inf], np.nan, inplace=True)
cov_data_log.fillna(0, inplace=True)
test_stationarity(cov_data_log)
```




```
Dickey-Fuller Test:
ADF Statistic: -3.484655
p-value (should be smaller than 0.05): 0.008390046841046767
Critical values:
        1%: -3.487
        5%: -2.886
        10%: -2.580
```

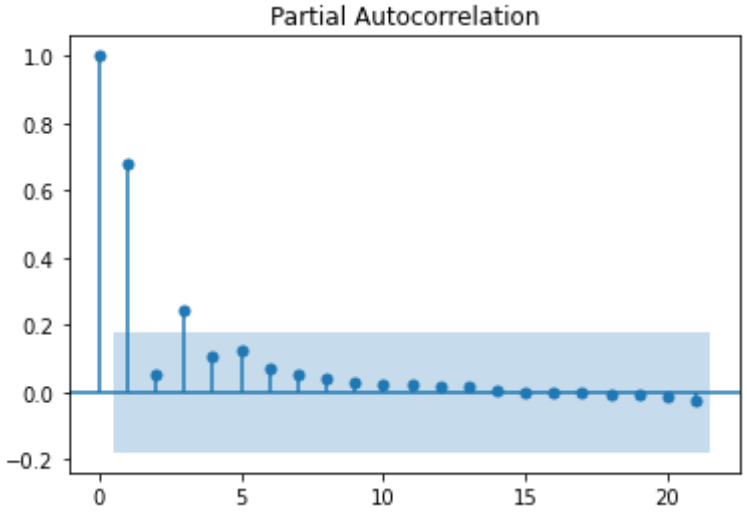
The following plots, PACF and ACF, help us choose the AR and the MA terms.

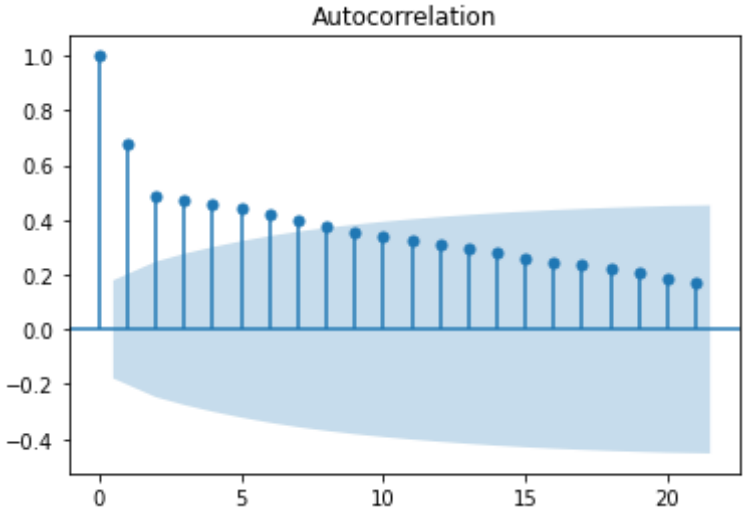
The **PACF** (partial autocorrelation of lag) plot presents the correlation between the series and its lag. The AR term is the number of lags that cross the significance limit (approx. 3).

The **ACF** essentially states how many MA terms are necessariy to avoid autocorrelation (approx. 7).

In [432…

```
plot_pacf(cov_data_log)
plot_acf(cov_data_log)
plt.show()
```





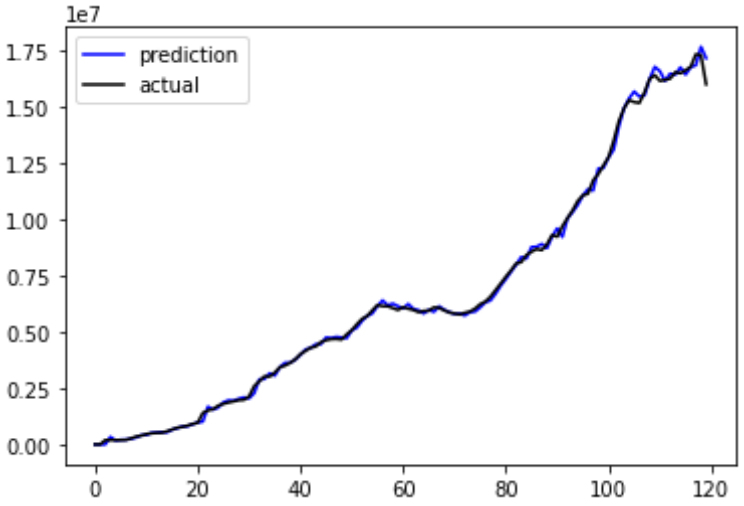
Therefore, from the previous computations we can deduce that we require the following orders:

1. we only applied the logarithmic transformation once => **d=1**
2. lags from the PACF above the significance limit => **p=2**
3. lags from the ACF above the significance limit => **q=5**

We can notice that the prediction on the trained dataset is completely accurate.

In [433…

```
model = ARIMA(cov_data.daily_vaccinations, order=(1,2,5))
model_fit = model.fit()

pred = model_fit.predict()

plt.plot(pred, label='prediction', color='blue')
plt.plot(cov_data.daily_vaccinations, label='actual', color='black')
plt.legend(loc='best')
plt.show()

arima_score11 = r2_score(cov_data.daily_vaccinations, pred)
print('regression score: ', r2_score(cov_data.daily_vaccinations, pred))
```



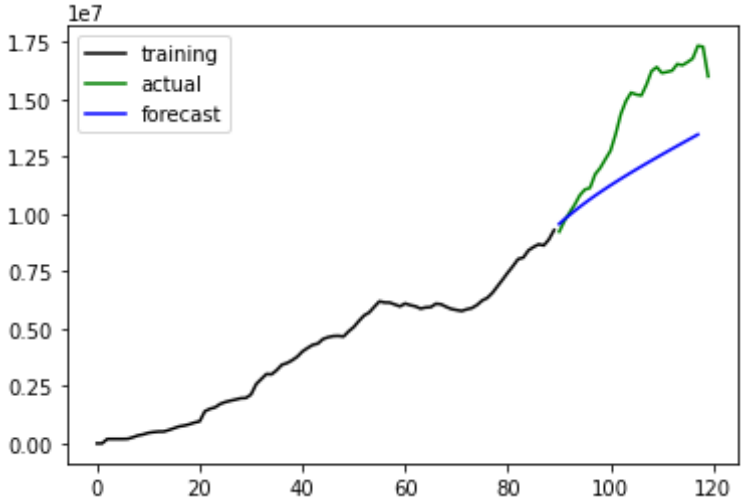

```
regression score:  0.9984953385895642
```

Below we have the forecast for the last 28 days from the dataset, based on the training set composed of the first 90 entries. This is the most accurate prediction out of what we have seen so far.

In [434…

```
train_model = cov_data[:90]
test_model = cov_data[90:]
model = ARIMA(train_model.daily_vaccinations, order=(1,2,5))
model_fit = model.fit()
```

```
fc = model_fit.forecast(28)
fc_series = pd.DataFrame({'daily_vaccinations' : fc}, index=test_model.index)

plt.plot(train_model.daily_vaccinations, label='training', color='black')
plt.plot(test_model.daily_vaccinations, label='actual', color='green')
plt.plot(fc_series.daily_vaccinations, label='forecast', color='blue')
plt.legend(loc='best')
plt.show()

arima_score12 = r2_score(test_model.daily_vaccinations[:-2], fc_series.daily_vaccinations[:-2])
print('regression score: ', r2_score(test_model.daily_vaccinations[:-2], fc_series.daily_vaccinations[:-2]))
```



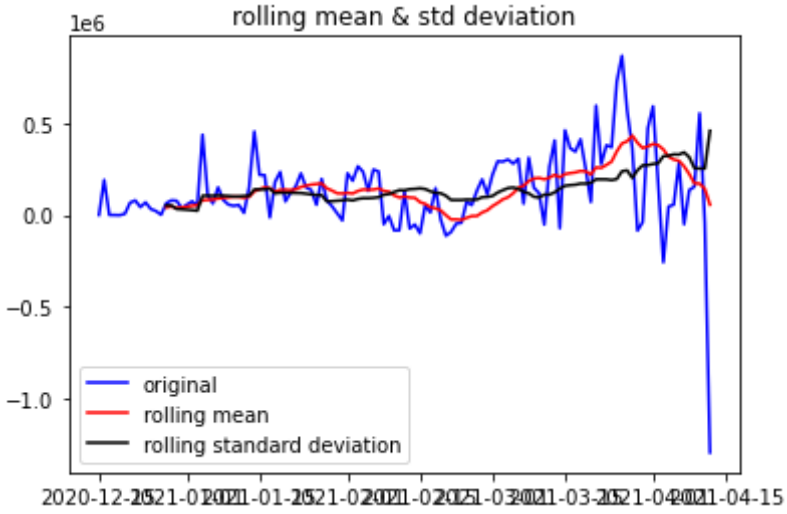

```
regression score:  -0.11929929447876497
```

Difference Transformation

We performed three different difference transformation to analyze the outputs. We can notice that one transformation is not enough to obtain the required p-value.

In [435…

```
cov_data_diff = cov_data_stationarity.diff().dropna()
test_stationarity(cov_data_diff)
```

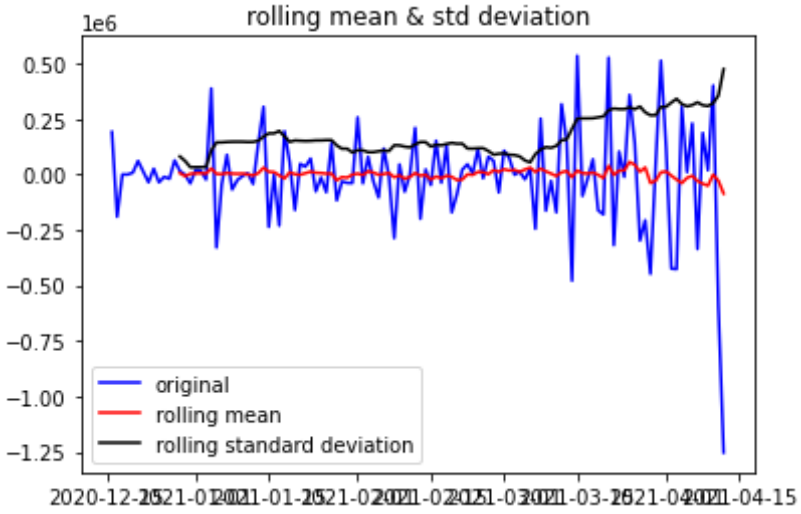



```
Dickey-Fuller Test:
ADF Statistic: -1.530893
p-value (should be smaller than 0.05): 0.5181776938129292
Critical values:
        1%: -3.494
        5%: -2.889
        10%: -2.582
```

Two transformations are still not ideal, but they are closer to the 0.05 limit.

In [436…

```
cov_data_diff = cov_data_stationarity.diff().diff().dropna()
test_stationarity(cov_data_diff)
```

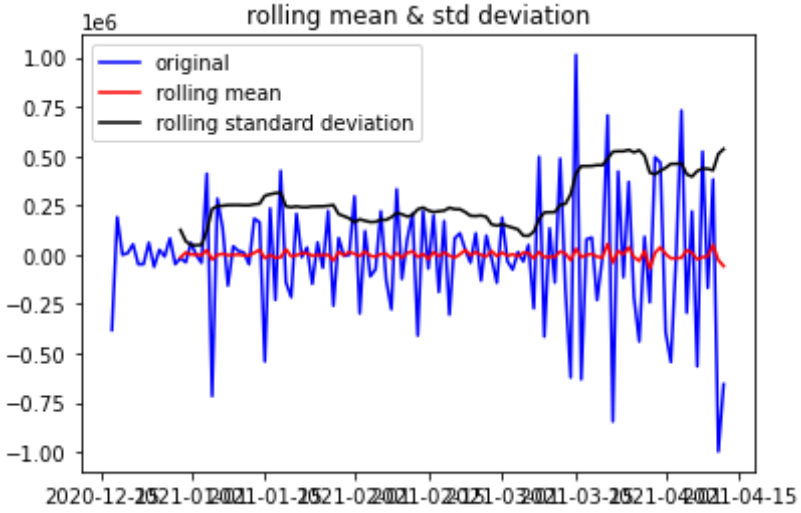



```
Dickey-Fuller Test:
ADF Statistic: -2.323838
p-value (should be smaller than 0.05): 0.1643950783776551
Critical values:
        1%: -3.495
        5%: -2.890
        10%: -2.582
```

The third transformation results into what seems to be an over-differenced series. For this reason, we will use the two difference transform for the ARIMA model.

In [437…

```
cov_data_diff2 = cov_data_stationarity.diff().diff().diff().dropna()
test_stationarity(cov_data_diff2)
```



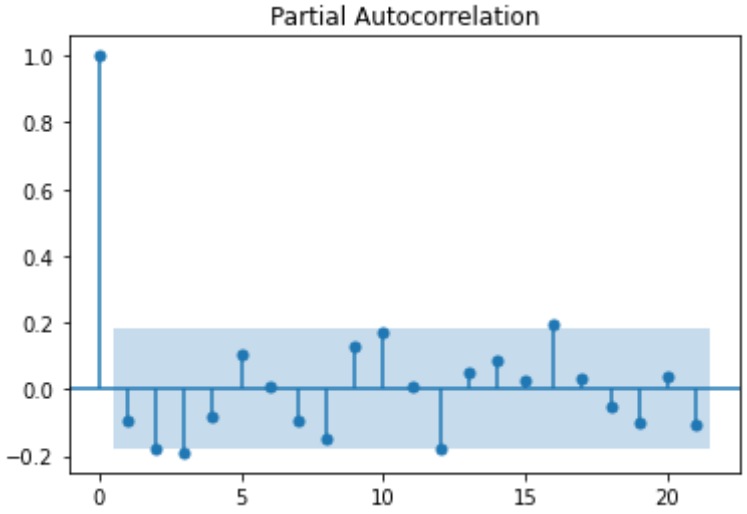

```
Dickey-Fuller Test:
ADF Statistic: -5.735911
p-value (should be smaller than 0.05): 6.43927266057013e-07
Critical values:
        1%: -3.495
        5%: -2.890
        10%: -2.582
```

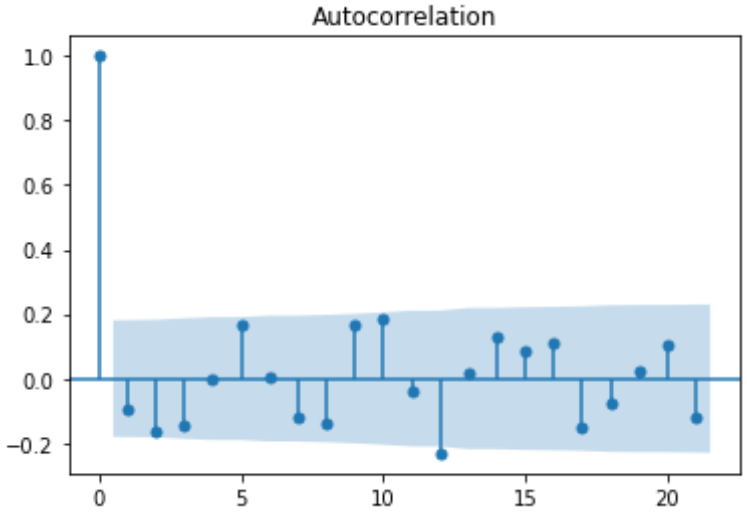
As above, we can notice that:

1. **d=2**
2. **p=3**
3. **q=2**

In [438…

```
plot_pacf(cov_data_diff)
plot_acf(cov_data_diff)
plt.show()
```

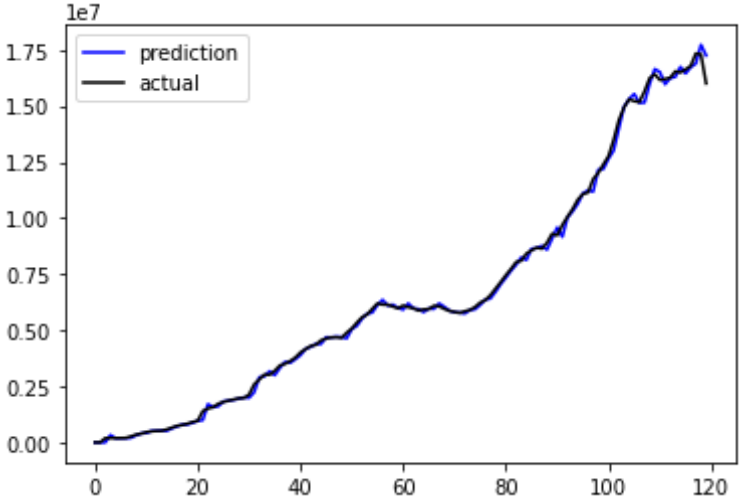




In [439…

```
model = ARIMA(cov_data.daily_vaccinations, order=(2,1,0))
model_fit = model.fit()

pred = model_fit.predict(dynamic=False)

plt.plot(pred, label='prediction', color='blue')
plt.plot(cov_data.daily_vaccinations, label='actual', color='black')
```

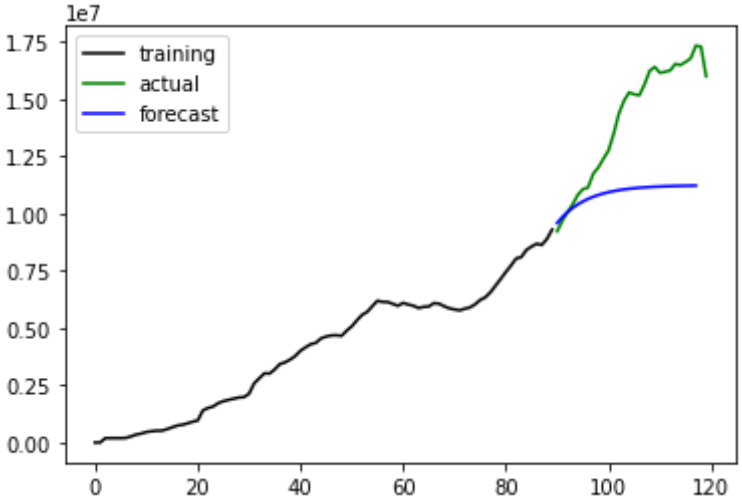
```
plt.legend(loc='best')
plt.show()

arima_score21 =  r2_score(cov_data.daily_vaccinations, pred)
print('regression score: ', r2_score(cov_data.daily_vaccinations, pred))
```




```
regression score:  0.9983302521906136
```

In [440…

```
train_model = cov_data[:90]
test_model = cov_data[90:]
model = ARIMA(train_model.daily_vaccinations, order=(2,1,0))
model_fit = model.fit()

fc = model_fit.forecast(28)
fc_series = pd.DataFrame({'daily_vaccinations' : fc}, index=test_model.index)

plt.plot(train_model.daily_vaccinations, label='training', color='black')
plt.plot(test_model.daily_vaccinations, label='actual', color='green')
plt.plot(fc_series.daily_vaccinations, label='forecast', color='blue')
plt.legend(loc='best')
plt.show()

arima_score22 = r2_score(test_model.daily_vaccinations[:-2], fc_series.daily_vaccinations[:-2])
print('regression score: ', r2_score(test_model.daily_vaccinations[:-2], fc_series.daily_vaccinations[:-2]))
```




```
regression score:  -1.1690253264476138
```

SARIMA

The SARIMA model is an enhanced form of the ARIMA model which performs better on time series which have seasonality, like in our case. This is done by introducing a new tuple, which is the seasonal components, which are convey the seasonality orders of the d, p, q components.

In order to obtain a model that is as accurate as possible, we calculated the AIC score of each of the (order, seasonal_order) tuples. We are looking for the lowest AIC score to determine combination which is most likely to give a satisfacatory result.

In [441…

```
from statsmodels.tsa.statespace.sarimax import SARIMAX
import itertools

dictionary = dict()
p = d = q = range(0, 2)
pdq = list(itertools.product(p, d, q))

seasonal_pdq = [(x[0], x[1], x[2], 12) for x in list(itertools.product(p, d, q))]

for params in pdq:
    for params_seasonal in seasonal_pdq:
        model = sm.tsa.statespace.SARIMAX(cov_data.daily_vaccinations,
                                          order=params,
```

```
                                            seasonal_order=params_seasonal,
                                            enforce_stationarity=False,
                                            enforce_invertibility=False)
        model_aic = model.fit()
        dictionary.update({(params, params_seasonal):model_aic.aic})

{k: v for k, v in sorted(dictionary.items(), key=lambda item: item[1])}
#((1, 1, 1), (1, 1, 1, 12)): 2573.671869615992
#print(dictionary)
```

```
c:\users\iulia\anaconda3\envs\lab_is\lib\site-packages\statsmodels\base\model.py:566: ConvergenceWarning:

Maximum Likelihood optimization failed to converge. Check mle_retvals
```

Out[441…

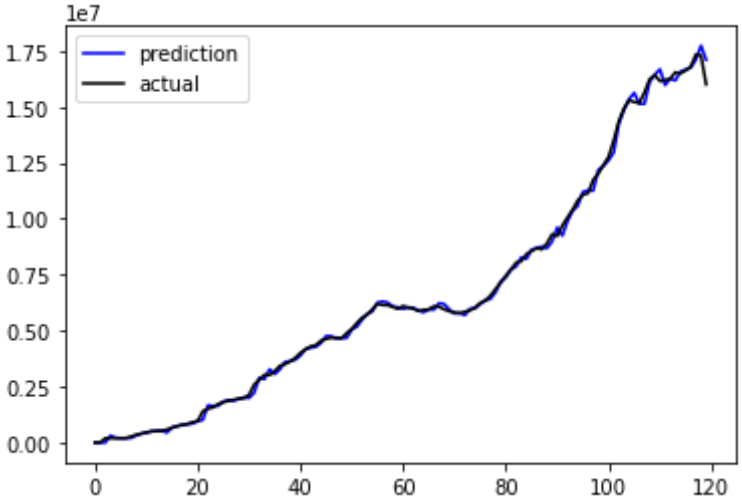
```
{((1, 1, 1), (1, 1, 1, 12)): 2573.671869615992,
 ((1, 1, 1), (0, 1, 1, 12)): 2577.289026810623,
 ((0, 1, 1), (1, 1, 1, 12)): 2577.994989414253,
 ((0, 1, 1), (0, 1, 1, 12)): 2578.5218963502725,
 ((1, 1, 0), (1, 1, 1, 12)): 2599.3514372334184,
 ((1, 1, 0), (0, 1, 1, 12)): 2602.924012660668,
 ((1, 1, 0), (1, 1, 0, 12)): 2611.538285972629,
 ((1, 1, 1), (1, 1, 0, 12)): 2613.2395552060866,
 ((1, 0, 1), (1, 1, 1, 12)): 2615.193138693079,
 ((0, 1, 0), (1, 1, 1, 12)): 2623.4994369086626,
 ((0, 1, 0), (0, 1, 1, 12)): 2630.3976790101333,
 ((0, 1, 1), (1, 1, 0, 12)): 2640.2125615900477,
 ((1, 0, 1), (1, 1, 0, 12)): 2642.0984763249694,
 ((1, 0, 0), (1, 1, 1, 12)): 2649.7682245641013,
 ((0, 1, 0), (1, 1, 0, 12)): 2656.0114437551997,
 ((1, 0, 0), (1, 1, 0, 12)): 2659.8509990947905,
 ((1, 1, 1), (1, 0, 1, 12)): 2883.2250311039156,
 ((1, 1, 1), (0, 0, 1, 12)): 2885.10358784698,
 ((1, 0, 1), (1, 0, 1, 12)): 2892.0978203316513,
 ((0, 1, 1), (1, 0, 1, 12)): 2894.803570299413,
 ((0, 1, 1), (0, 0, 1, 12)): 2900.0844848569704,
 ((1, 1, 0), (1, 0, 0, 12)): 2907.5495480268287,
 ((1, 1, 0), (1, 0, 1, 12)): 2908.6170774800503,
 ((1, 1, 1), (1, 0, 0, 12)): 2908.958552135025,
 ((1, 1, 0), (0, 0, 1, 12)): 2909.972258175247,
 ((1, 0, 0), (1, 0, 1, 12)): 2931.6498822271033,
 ((0, 0, 1), (1, 1, 1, 12)): 2938.412955636303,
 ((1, 0, 1), (1, 0, 0, 12)): 2939.857390941271,
 ((0, 0, 1), (0, 1, 1, 12)): 2943.513301567542,
```

```
((0, 1, 1), (0, 1, 0, 12)): 2943.715191764104,
((0, 1, 0), (1, 0, 1, 12)): 2945.4618619624152,
((1, 1, 1), (0, 1, 0, 12)): 2945.7148205225467,
((0, 1, 1), (1, 0, 0, 12)): 2953.2231878653442,
((0, 0, 0), (1, 1, 1, 12)): 2959.208671242231,
((0, 1, 0), (0, 0, 1, 12)): 2959.489997616264,
((1, 0, 0), (1, 0, 0, 12)): 2962.243095091133,
((1, 0, 1), (0, 1, 0, 12)): 2971.9628173211922,
((1, 1, 0), (0, 1, 0, 12)): 2973.7668171572423,
((0, 1, 0), (1, 0, 0, 12)): 2981.8310205586095,
((0, 1, 0), (0, 1, 0, 12)): 2985.3299625885807,
((0, 0, 1), (1, 1, 0, 12)): 3001.163920289835,
((0, 0, 0), (1, 1, 0, 12)): 3004.877956980975,
((1, 0, 0), (0, 1, 0, 12)): 3013.089955081767,
((0, 0, 0), (0, 1, 1, 12)): 3027.7676099833106,
((1, 1, 1), (0, 0, 0, 12)): 3206.006307996419,
((0, 1, 1), (0, 0, 0, 12)): 3217.8956265963466,
((1, 1, 0), (0, 0, 0, 12)): 3231.373606192966,
((1, 0, 1), (0, 0, 0, 12)): 3232.6656368429512,
((1, 0, 0), (0, 0, 0, 12)): 3281.685548238968,
((0, 1, 0), (0, 0, 0, 12)): 3283.9724921254833,
((0, 0, 0), (1, 0, 1, 12)): 3316.1015040717975,
((0, 0, 0), (1, 0, 0, 12)): 3344.8800415951014,
((0, 0, 0), (0, 1, 0, 12)): 3436.779550192775,
((0, 0, 1), (0, 1, 0, 12)): 3489.4844364215437,
((0, 0, 1), (1, 0, 1, 12)): 3581.981491382398,
((0, 0, 1), (0, 0, 1, 12)): 3587.369193082728,
((0, 0, 1), (1, 0, 0, 12)): 3647.4052582162535,
((0, 0, 0), (0, 0, 1, 12)): 3669.3369859746113,
((0, 0, 0), (0, 0, 0, 12)): 4133.181063838229,
((1, 0, 0), (0, 1, 1, 12)): 4200.80806130816,
((0, 0, 1), (0, 0, 0, 12)): 4730.630384886412,
((1, 0, 1), (0, 1, 1, 12)): 10855.666841228382,
((1, 0, 1), (0, 0, 1, 12)): 41799.085891088456,
((1, 0, 0), (0, 0, 1, 12)): 48163.11358114551}
```

In [442…

```
model = sm.tsa.statespace.SARIMAX(cov_data.daily_vaccinations,
                                  order=(1,1,1),
                                  seasonal_order=(1,1,1,12),
                                  enforce_stationarity=False,
                                  enforce_invertibility=False)

model_fit = model.fit()

pred = model_fit.predict()
```

```
plt.plot(pred, label='prediction', color='blue')
plt.plot(cov_data.daily_vaccinations, label='actual', color='black')
plt.legend(loc='best')
plt.show()

sarima_score1 = r2_score(cov_data.daily_vaccinations, pred)
print('regression score: ', r2_score(cov_data.daily_vaccinations, pred))
print(model_fit.aic)
```

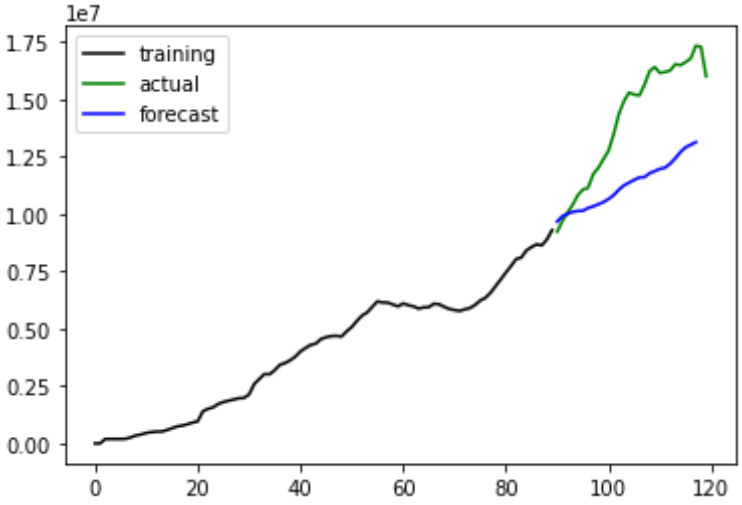



```
regression score:  0.9984936865891029
2573.671869615992
```

In [443…

```
train_model = cov_data[:90]
test_model = cov_data[90:]
model = sm.tsa.statespace.SARIMAX(train_model.daily_vaccinations,
                                  order=(1,1,0),
                                  seasonal_order=(1,1,1,12))

model_fit = model.fit()

fc = model_fit.forecast(28)
fc_series = pd.DataFrame({'daily_vaccinations' : fc}, index=test_model.index)

plt.plot(train_model.daily_vaccinations, label='training', color='black')
plt.plot(test_model.daily_vaccinations, label='actual', color='green')
```

```
plt.plot(fc_series.daily_vaccinations, label='forecast', color='blue')
plt.legend(loc='best')
plt.show()

sarima_score2 = r2_score(test_model.daily_vaccinations[:-2], fc_series.daily_vaccinations[:-2])
print('regression score: ', r2_score(test_model.daily_vaccinations[:-2], fc_series.daily_vaccinations[:-2]))
print(model_fit.aic)
```




```
regression score:  -0.5016216488488419
2039.6325323271847
```

Comparison between linear regression, ARIMA and Decision Trees

General overview of training set - testing set comparison

As we have seen in the graphs above, for the most part, as it natural, the model which tests on the set that it was trained on performs really well, while training on a different set might generate some issues.

This might be generated because of two reasons:

1. The size of the dataset. The set that we tested on this rather small and contains only a few months of data, with abrupt changes which cannot be detected that easily when trying to predict. That is, if the training is done on a subset with a smaller increasing rate, if the testing is performed on a subset with a large increasing rate, the results will be inaccurate.
2. The nature of the model. For example, the linear regression model, as well as the decision tree model do not perform well when it comes to forecasting. The linear regression attempts an approximation but by it's nature it is not that efficient. The decision tree gets capped at the maximum value and it will predict that constant value throughout the entire testing set.

ARIMA and SARIMA, however, perform better. We wanted to compute the ARIMA model even if it does not perform as well on seasonal datasets in order to analyze the computation algorithm for it. In case of SARIMA, while it has the potential of generating accurate data, it lacks the training set necessary to do so.

For the comparisons below, we used the r2 score to draw conclusions. The r2 score represents how much of the output predictions can be deduced from the input. In our case, the function performs a comparison between the predicted output and the expected output to draw this conclusion.

Comparsion between predictions on same set

Linear, Decision Trees, Random Forests, ARIMA (log and diff), SARIMA

In [444…

```
comp1 = np.array([['index', 'name', 'score'], [1, 'lreg', lreg_score1], [2, 'dtree_depth2', dtree_score11], [3, 'dtre
data = pd.DataFrame(data=comp1[1:,1:], index=comp1[1:,0], columns=comp1[0,1:])
data
```

Out[444…

	name	score
1	lreg	0.9239323895658387
2	dtree_depth2	0.9482125342504327
3	dtree_depth3	0.9850573079384972
4	rf_n5	0.9621711161136298
5	rf_n10	0.9697827252846372
6	arima_log	0.9984953385895642
7	arima_diff	0.9983302521906136

	name	score
8	sarima	0.9984936865891029

As it can be seen in the bar chart below, both the ARIMA log and diff and SARIMA have near perfect scores, as it was expected.

The random forests approach with 5 and 10 trees respectively have almost the same score, which is worse than the approach with a decision tree of depth 3 and better than the approach with a decision tree of depth 2.

Lastly, the linear regression approach yielded the worst score in this scenario. Nontheless, this score was still above 0.92.

In [445…

```
fig = px.bar(data, x='name', y='score')
fig.update_layout(yaxis_title = 'Scores', yaxis_type = 'linear', yaxis_range = [0.9, 1])
fig.update_layout(xaxis_title = 'Name')
fig.show()
```

Comparsion between predictions on different set

Linear, Decision Trees, ARIMA (log and diff), SARIMA

In [446…

```
comp2 = np.array([['index', 'name', 'score'], [1, 'lreg', lreg_score2], [2, 'dtree', dtree_score12], [3, 'arima_log',
data = pd.DataFrame(data=comp2[1:,1:], index=comp2[1:,0], columns=comp2[0,1:])
data
```

Out[446…

	name	score
1	lreg	-1.8819501865861508
2	dtree	-4.414492061125101
3	arima_log	-0.11929929447876497
4	arima_diff	-1.1690253264476138
5	sarima	-0.5016216488488419
6	auto_arima	-0.6737033653810394

We can observe that the models all have negative scores, which indicate a non-satisfiable prediction.

SARIMA and auto_ARIMA are perform slightly better, along with the logairthmic transformation ARIMA model.

In [447…

```
fig = px.bar(data, x='name', y='score')
fig.update_layout(yaxis_title = 'Scores', yaxis_type = 'linear', yaxis_range = [-5, 1.1])
```

```
fig.update_layout(xaxis_title = 'Name')
fig.show()
```

References

Information sources

The main source of information for the linear regression, decision trees and random forests was Hands-On Machine LEarning with Scikit-Learn & TensorFlow, **Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems, Aurélien Géron**

For the ARIMA model we used several sources from the internet, among which: https://people.duke.edu/~rnau/411arim.htm

For model descriptions and specifications, along with python implementations

https://www.analyticsvidhya.com/blog/2018/08/auto-arima-time-series-modeling-python-r/ https://www.machinelearningplus.com/time-series/arima-model-time-series-forecasting-python/ https://machinelearningmastery.com/time-series-data-stationary-python/ https://www.kaggle.com/antoinekrajnc/simplest-time-series-using-linear-regression https://www.scribbr.com/statistics/p-value/