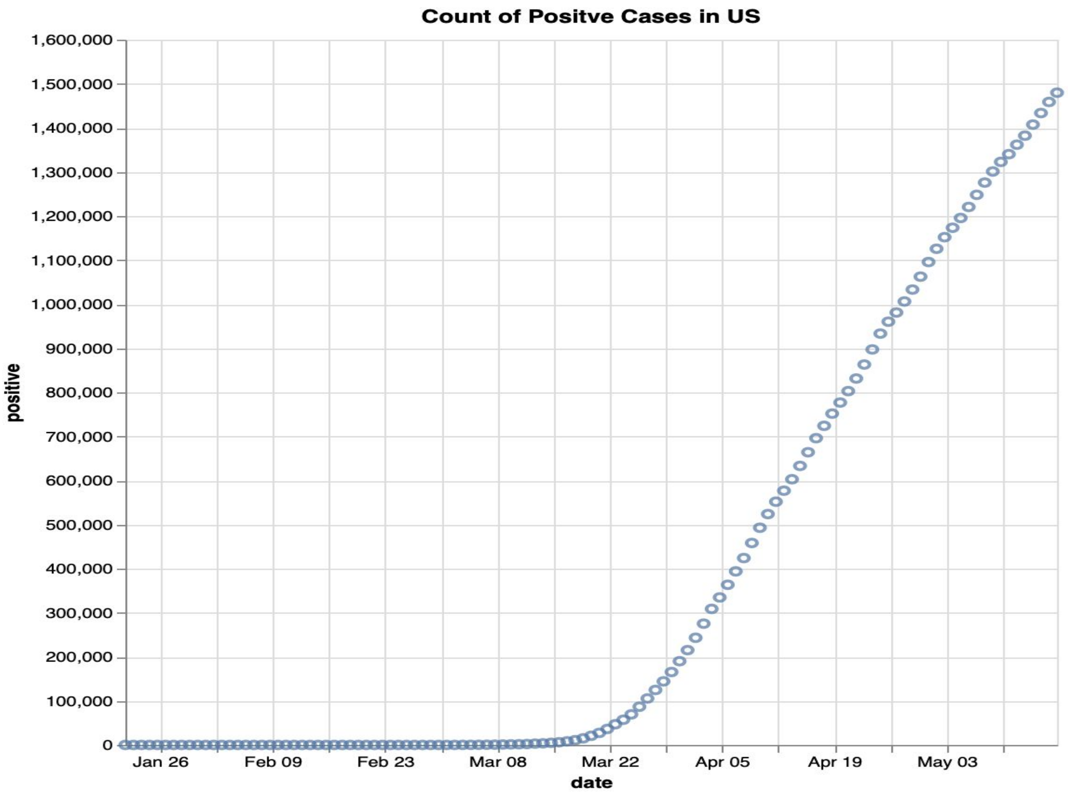
Investigating and Predicting COVID-19 Confirmed Cases in U.S. : Analysis and Conclusion

Team #5:
Kiran Brar
Olivia Alexander
Kimberly Segura

Number of Confirmed COVID-19 Cases in U.S.



Output Generation and Analysis

Preprocessing Techniques

- **Power transformation**: BoxCox() is configurable data transform method that evaluate a suite of transforms automatically and select a best fit (optimizes lambda).

- `lambda` = -1.0 is a reciprocal transform.
- `lambda` = -0.5 is a reciprocal square root transform.
- `lambda` = 0.0 is a log transform.
- `lambda` = 0.5 is a square root transform.
- `lambda` = 1.0 is no transform.

```
us_Y = us_df["positive"]

us_box_total, lam = boxcox(us_Y)
print('Lambda: %f' % lam)
```

```
Lambda: 0.046134
```

Preprocessing Techniques

- **Standardization**: Transform data to mean of 0 and standard deviation of 1-- Gaussian transform.

```
#Standardize
scaler = StandardScaler()

scaled_train_labels = scaler.fit_transform(train_labels.reshape(-1, 1))
scaled_test_labels = scaler.transform(test_labels.reshape(-1, 1))
```

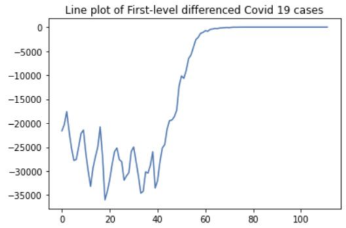
- **Normalization**: Scale data to different/minimized range.

```
#Scale/Normalize Input
minmax_scaler = MinMaxScaler()

scaled_train_labels = minmax_scaler.fit_transform(train_labels.reshape(-1, 1))
scaled_test_labels = minmax_scaler.transform(test_labels.reshape(-1, 1))
```

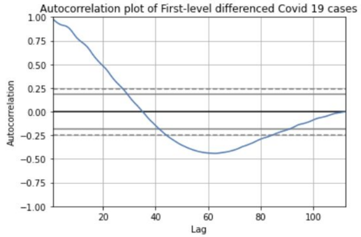
Preprocessing Techniques

- **Differencing:** Removes trends and seasonality from a time series dataset.

```
raw_cases = us_df["positive"]

first_level =  difference(raw_cases)
plt.plot(first_level);
plt.title("Line plot of First-level differenced Covid 19 cases");
```

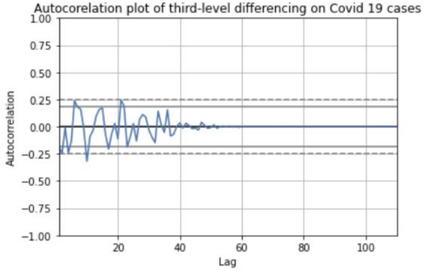



```
autocorrelation_plot(first_level)
plt.title("Autocorrelation plot of First-level differenced Covid 19 cases");
```




```
third_level_diff = difference(second_level)
autocorrelation_plot(third_level_diff)
plt.title("Autocorelation plot of third-level differencing on Covid 19 cases");
print("Data is now stationary!")
```

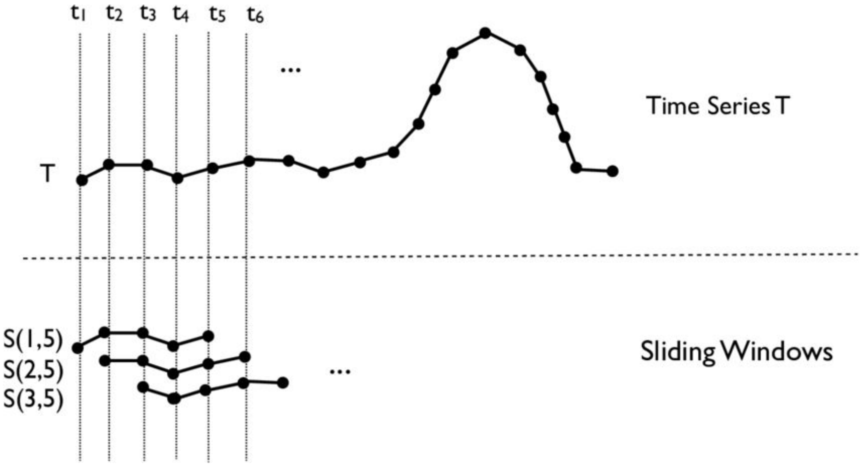
```
Data is now stationary!
```



Third-level Differencing removed the autocorrelation from our dataset

Converting Time Series into Supervised Learning

Sliding Window / Lagged Values: Using a previous number of values (rather than time) to predict the following value




```
def create_dataset(X, y, time_steps=1):
    Xs, ys = [], []
    for i in range(len(X) - time_steps):
        v = X.iloc[i:(i + time_steps)].values
        Xs.append(v)
        ys.append(y.iloc[i + time_steps])
    return np.array(Xs), np.array(ys)

timesteps =7
train_x, train_y = create_dataset(train_data , train_data , timesteps)
test_x, test_y = create_dataset(test_data , test_data , timesteps)
```

Models:

Linear Regression

- Used sliding window approach
- **Best Linear Regression model:**

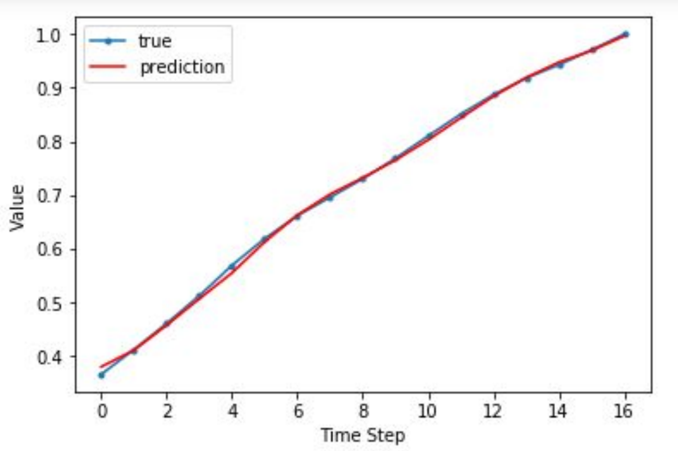
  Lagged Values: 6 days

  Box Cox: Yes

  Differencing: No

  Scaling/Normalizing: MinMaxScaler()

Linear Regression: Prediction Plots



MSE	0.0000402
SMAPE	0.84284873

Linear Regression Model: Different Lagged Values

	Linear Regression with Different Preprocessing Techniques									
	Model 0	**Model 1.1**	**Model 1.2**	**Model 1.3**	**Model 1.4**	**Model 1.5**	**Model 1.6**	**Model 1.7**	**Model 1.8**	**Model 2**
Preprocessing										
Lagged Value	None	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	6
Box Cox /Log	None	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Scaling/Normalizing	None	Minmax()	Minmax()	Minmax()	Minmax()	Minmax()	Minmax()	Minmax()	Minmax()	**Minmax(-1,1)**
MSE (Test)	49347904776	0.001348057	0.000220618	0.000107409	8.45E-05	5.36E-05	4.02E-05	6.44E-05	6.61E-05	0.000160831
SMAPE (Test)	120.5941151	16.37057734	3.737763698	2.002087713	1.512598561	1.055856231	0.84284873	0.987350005	0.97504241	7.03589239
Notes:	*LR on raw set*						*lowest error*	*error increasing again*		*auto minmax is better*

Linear SVR

- Used sliding window approach
- **Best Linear SVR model:**

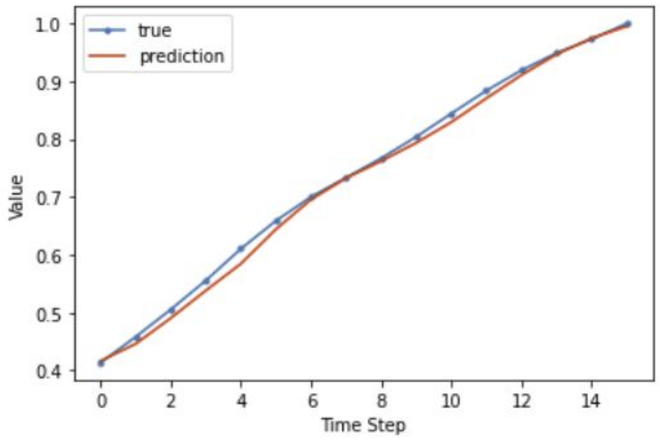
  Lagged Values: 7 days

  Box Cox: Yes

  Differencing: No

  Scaling/Normalizing: MinMaxScaler()

Linear SVR: Prediction Plots



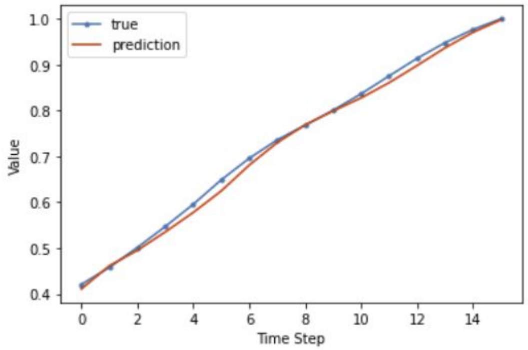
MSE	**0.00009108727842**
SMAPE	1.145
RMSE	0.009544

Nonlinear SVR with RBF Kernel

Nonlinear SVR model (with lowest MSE):

- Lagged Values: 7 days
- Box Cox: Yes
- Differencing: No
- Scaling/Normalizing: MinMaxScaler()

Nonlinear SVR: Prediction Plots



MSE	0.000135
SMAPE	1.455
RMSE	0.011619

Linear SVR: Effect of Lag Value

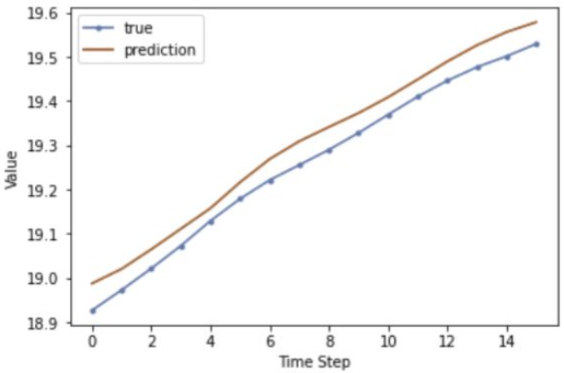
	Linear SVR: Performance for Different Lag Values			
	Model 3.1	**Model 3.2**	**Model 3.3**	**Model 3.4**
Features:				
Preprocessing				
Lagged Value	**3**	**5**	**10**	**8**
Box Cox /Log	Yes	Yes	Yes	Yes
Differencing	No	No	No	No
Scaling/Normaliz	Minmax(0,1)	Minmax(0,1)	Minmax(0,1)	Minmax(0,1)
Model				
Objective	Linear	Linear	Linear	Linear
Kernel	-	-	-	-
Hyperparamters				
C	100	1000	100	1
Epsilon	0.0005	0.0001	0.0001	0.001
Gamma	-	-	-	-
MSE	0.001026	0.000804	0.0001094	0.000114
SMAPE	5.068	4.284	1.072	1.2952
RMSE	0.0320312	0.0283549	0.0104594	0.0106771

Linear SVR: Effect of Scaling vs. Standardizing

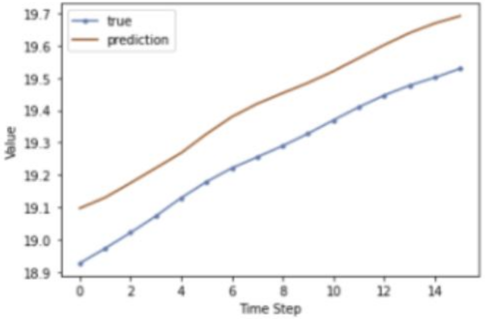
	LinearSVR Model Performances		
	Model 5	**Model 5.1**	**Model 5.5**
Features:			
Preprocessing			
Lagged Value	7	7	7
Box Cox /Log	No	No	No
Differencing	No	No	No
Scaling/Normali	**Minmax(0,1)**	**Minmax(-1,1)**	**StandardScaler**
Hyperparamters			
C	1000	100	1000
Epsilon	0.001	0.0001	0.0001
MSE	0.000276	0.00049046	0.0033577
SMAPE	2.419158	9.68705	11.82743
RMSE	0.0166132	0.0221463	0.0579457

	Support Vector Regression: Performance Results					
	Model 1	**Model 2**	**Model 5.5**	**Model 6**	**Model 7**	**Model 8**
Features:						
Preprocessing						
Lagged Value	7	7	7	7	7	7
Box Cox /Log	Yes	Yes	No	Yes	Yes	**No**
Differencing	No	No	No	No	No	**No**
Scaling/Normaliz	Minmax(0,1)	Minmax(0,1)	**StandardScaler**	**None**	*None*	**None**
Model						
Objective	Linear	**Non-linear**	Linear	Linear	**Nonlinear**	Linear
Kernel	-	**RBF**	-	-	rbf	-
Hyperparamters						
C	1000	1000	1000	1	1000	1
Epsilon	0.0001	0.0001	0.0001	0.0005	0.001	0.5
Gamma	-	0.005	-		0.001	
MSE	**0.00009108727**	0.000135	0.0033577	0.0246214	0.00213	307124112.3
SMAPE	1.145	1.455	11.82743	0.810457	**0.236**	1.256246775
RMSE	0.009544	0.011619	0.0579457	0.1569121	0.0461519	17524.95684

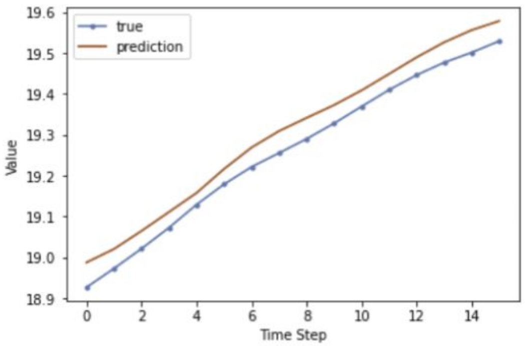
Model 5.5



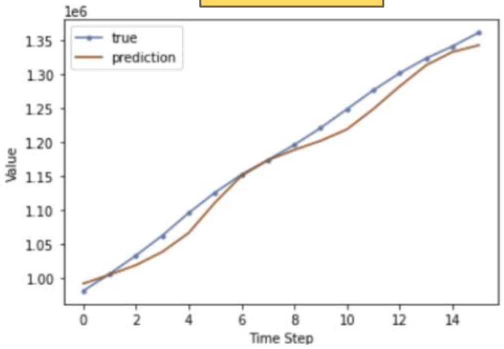
Model 6



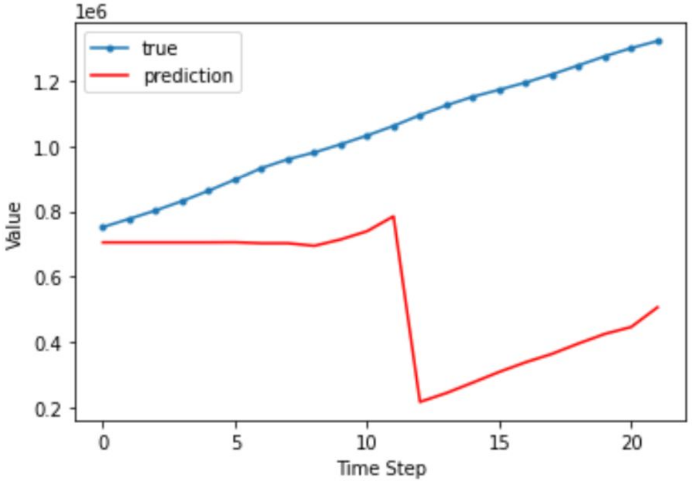
Model 7



Model 8



XGBoost :
Approach 1 (Use month, day as features)



	XGBoost: Using Date Features		
	Model 0	**Model 0.5**	**Model 1**
Features:	Month, day	Month, day	Month, day
Preprocessing			
Box Cox /Log	No	Yes	Yes
Scaling/Normali	Minmax	Minmax	StandardScaler
MSE (Test)	883237648.7	168	2.199
SMAPE (Test)	69	21.56	6.789
RMSE(Test)	29719.3144	12.96148	1.4829

XGBoost :
Approach 2 (Lagged Values as Input)

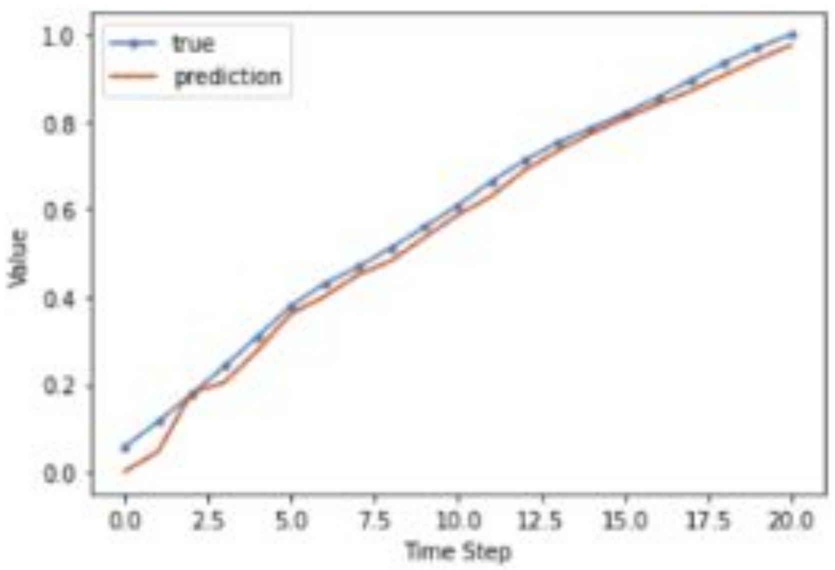
- Used sliding window approach
- Best model

  Lagged Values: 1

  Box Cox: Yes

  Differencing: No

  Scaling/Normalizing: MinMaxScaler()



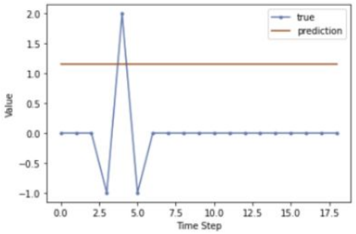
XGBoost : Effect of Lagged Values

- From our experiments, we found that the smaller lag values increased performance for XGBoost.

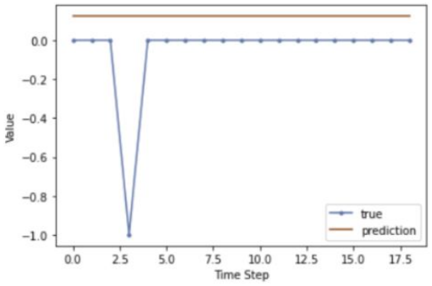
	XGBoost Model Performances with Lagged Values Features and No Differencing							
	Model 2	**Model 2.1**	**Model 2.3***	**Model 2.5**	**Model. 2.6**	**Model. 2.7**	**Model 2.8**	**Model 2.9**
Features:	LagVal	LagVal	LagVal	LagVal	LagVal	LagVal	LagVal	LagVal
Preprocessing								
Lagged Value	**7**	7	7	**10**	**3**	**5**	**2**	**1**
Box Cox /Log (0	Yes	Yes	Yes	Yes	Yes	yes	Yes	Yes
Scaling/Normali	Minmax [0,1]	**Minmax[-1,1]**	Minmax [0,1]	Minmax [0,1]	Minmax [0,1]	Minmax [0,1]	Minmax [0,1]	Minmax [0,1]
MSE (Test)	0.02119	0.022	0.01072	0.04252	0.0049	0.011	0.002751	**0.0009066**
SMAPE (Test)	22	23.53	17.649	30.882	12.370	18.46	**13.54425**	17.22109
RMSE	0.1455679	0.148324	0.1035374	0.2062038	0.07	0.1048809	0.05245	0.0301098

XGBoost: Effect of Differencing

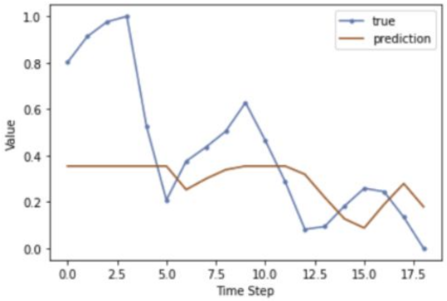
Model 3



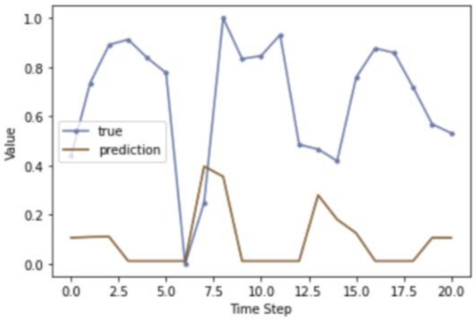
Model 3.1



Model 3.5



Model 3.8



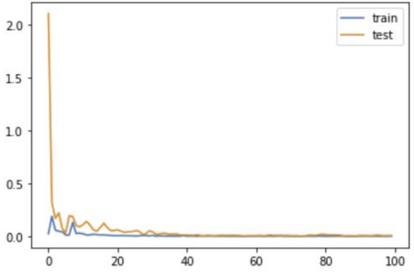
XGBoost: Effect of Differencing

	XGBoost Models using Differencing Preprocessing Techniques						
	Model 3	**Model 3.1**	**Model 3.5**	**Model 3.6**	**Model 3.7**	**Model 3.8**	**Model 3.9**
Features:	LagVal	LagVal	LagVal	LagVal	LagVal	LagVal	LagVal
Preprocessing							
Lagged Value	3	3	3	3	1	1	1
Box Cox /Log	None	None	Yes	Yes	Yes	Yes	Yes
Differencing Lev	3	1	1	2	2	3	3
Scaling/Normali	None	None	Minmax [0,1]	Minmax [0,1]	Minmax [0,1]	Minmax [0,1]	StandardScaler
MSE (Test)	192.291	200	68.064	132.445	154.858	**46.12**	174.962
SMAPE (Test)	1.6506	0.08078	0.0883	0.35	0.425	**0.082**	2.047
RMSE	13.8669168	14.1421356	8.2500909	11.5084751	12.4441954	**6.7911707**	13.2273202

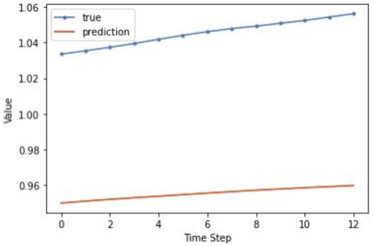
LSTM Best Model

```
] model = keras.Sequential()
  model.add(keras.layers.LSTM(50, input_shape=(X_train.shape[1], X_train.shape[2]), return_sequences = True))
  model.add(keras.layers.Dropout(0.1))
  model.add(keras.layers.LSTM(units = 50, return_sequences = True))
  model.add(keras.layers.Dropout(0.1))
  model.add(keras.layers.Dense(1))
  model.compile(loss='mean_squared_error', optimizer = keras.optimizers.Adam(0.1))
```

```
#Diagnostic Line Plots for Loss and Validation loss of the data
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='test')
plt.legend();
```




```
Evaluation on test data: MSE
1/1 [==============================] - 0s 970us/step - loss: 0.0082
0.0081513412296772

Evaluation on test data: SMAPE
117.20786382367955
```



LSTM: Effect of Lag Value and Differencing

	LSTM Performance of Different Lag Values (No Differencing)				LSTM Performance of Different Lag Values (with Differencing)			
Trial	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Features:								
Preprocessing								
Lagged Value	3	3	5	7	3	5	7	10
Box Cox /Log	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Differencing	No	No	No	No	Yes	Yes	Yes	Yes
Scaling/Normali	MinMax(-1,1)	MinMax(0,1)	MinMax(0,1)	MinMax(0,1)	MinMax(0,1)	MinMax(0,1)	MinMax(0,1)	MinMax(0,1)
MSE (Test)	0.161956	0.043926	0.023220	0.044450	0.030001	0.010664	0.009857	**0.008716**
SMAPE (Test)	917.383506	448.378769	283.922014	359.388216	363.644981	187.775853	159.572204	**121.370556**
RMSE	0.026230	0.001929	0.000539	0.001976	0.000900	0.000114	0.000097	**0.000076**

Compare output against Hypothesis

Hypothesis: We predicted long short term networks (LSTM) to yield the best accuracy, followed by XGBoost, SVR, and linear regression.

Output:

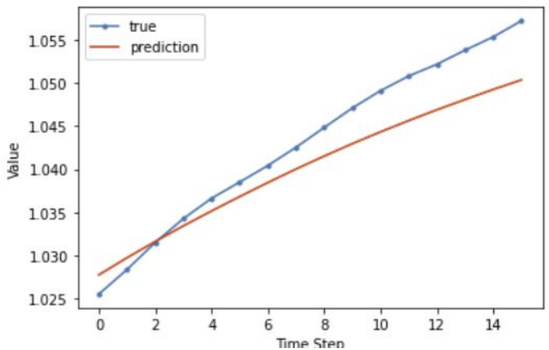
- Linear Regression and SVR had overall best performance(with preprocessing techniques).
- SVR had better performance than XGBoost and LSTM on ***raw*** data (in general)
- LSTM had the worst predictions

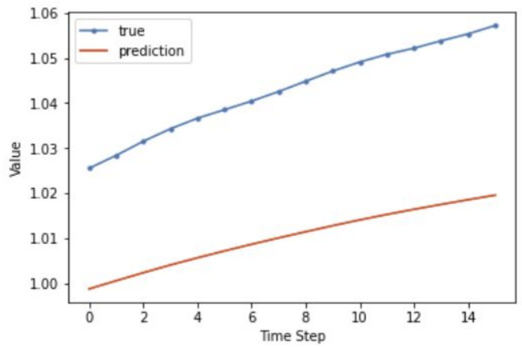
Abnormal Case Explanation

- Differencing did not always improve performance (only slightly helped for LSTM)
- SVR had surprisingly accurate predictions on untransformed data, in comparison to other models.
- Smaller lag values improved performance for XGBoost, whereas larger lag values improved performance for LR, SVR, and LSTM
- Minmax Scaler better performance than StandardScaler

Discussion

- We found lagged values, and power transformations (Box Cox) to be incredibly helpful to improve model performance.
- We also found LSTM had poor and unstable performance. This can be due to the small dataset or lack of time to tune the number of layers/neurons
    - Same model yielding different results:





Conclusion/ Recommendation

Summary/Conclusion

- Don't underestimate the importance of preprocessing or feature engineering!

- Important to test simple models first!

- For deep learning, you need a lot of data and it’s very time consuming to pick and choose different number of layers and neurons.

Recommendations for Future Studies

- Our work can be furthered by predicting number of deaths or number of recovered.

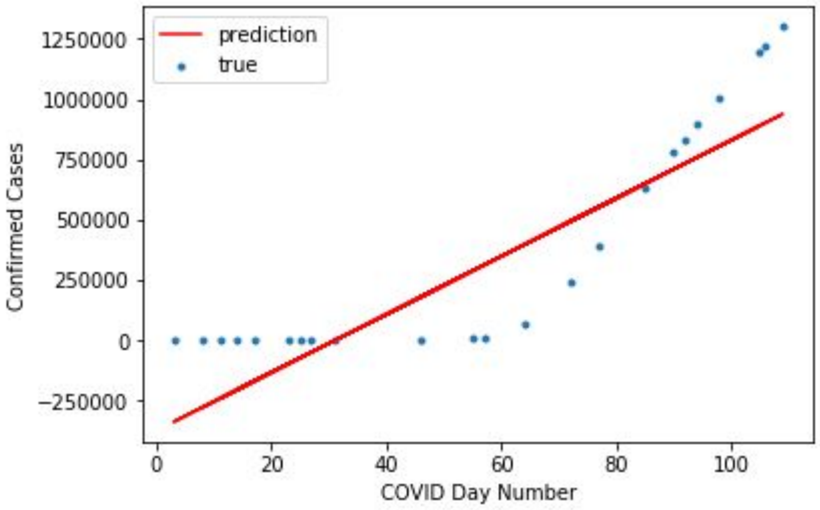
- Our models can be extended to take into consideration the health capacity as well as social restrictions for each in order to make better future forecasting.

Demonstration:

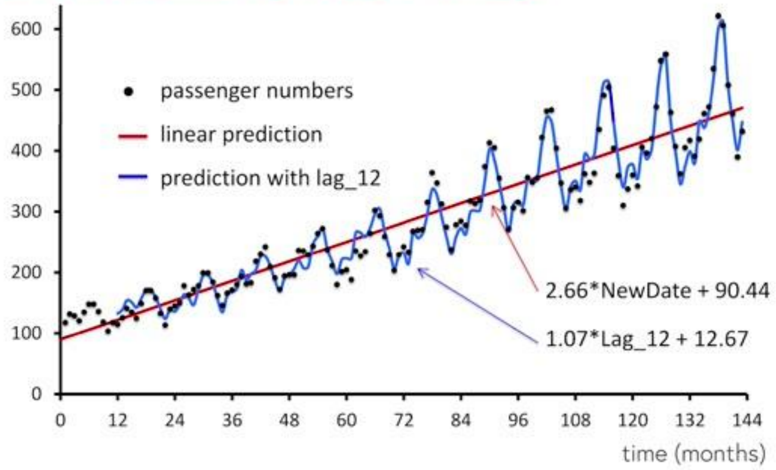
- Model Training and Evaluation
- EDA

Linear Regression Model: Nonlinear Prediction

Linear Prediction vs Nonlinear Prediction: Transform predictor X from time to sliding window. Nonlinear functional form but model still linear in parameters.



Time series: Linear regression with lags



Extra Reference Slides - LSTM approach 1

MODEL

```
[ ] #
    lstm_model = Sequential()
    lstm_model.add(LSTM(units = 50, return_sequences = True, input_shape = (n_input, n_features)))
    lstm_model.add(Dropout(0.2))
    lstm_model.add(LSTM(units = 50, return_sequences = True))
    lstm_model.add(Dropout(0.2))
    lstm_model.add(LSTM(units = 50))
    lstm_model.add(Dropout(0.2))
    lstm_model.add(Dense(units = 1))
    lstm_model.compile(optimizer = 'adam', loss = 'mean_squared_error')
    history = lstm_model.fit(generator, epochs = 100)
```

```
Epoch 14/100
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 10, 50)            10400
_________________________________________________________________
dropout_1 (Dropout)          (None, 10, 50)            0
_________________________________________________________________
lstm_2 (LSTM)                (None, 10, 50)            20200
_________________________________________________________________
dropout_2 (Dropout)          (None, 10, 50)            0
_________________________________________________________________
lstm_3 (LSTM)                (None, 50)                20200
_________________________________________________________________
dropout_3 (Dropout)          (None, 50)                0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 51
=================================================================
Total params: 50,851
Trainable params: 50,851
Non-trainable params: 0
_________________________________________________________________
```

Extra Reference Slides - LSTM approach 1

TimeseriesGenerator

```
] #LAG preprocessing to frame a sequence as a supervised learning problem
  #returns a sequence of overlapping windows
  #batch_size = # of samples to return on each iteration
  from tensorflow.keras.preprocessing.sequence import TimeseriesGenerator

  n_input =10 # lag
  n_features = 1
  generator = TimeseriesGenerator(scaled_train_data, scaled_train_data, length = n_input, batch_size = 1)
  for i in range(len(generator)):
      x, y = generator[i]
      #print('%s => %s' % (x, y))
```