Data Science Project1 (2021/2)
PM2.5 Kaggle (Redo)

Team: One Night Miracle

Member: 6230556021 Sirawit Makoo
6230563321 Suchakree Maneepakorn

YOUR RECENT SUBMISSION

testing_submission.csv

Submitted by Fact Maneepakorn · Submitted 2 days ago

Score: 9.10378

↓ Jump to your leaderboard position

RMSE

```
for sb in [sb_bkk, sb_cnx, sb_ray, sb_sara, sb_kkc, sb_surat]:
    sb[sb['Predict']<0] = 0
    diff = np.sqrt((np.array(sb['Predict'] - sb['Actual'])**2).mean())
    print(diff)
```

```
6.24731247042558
9.744932835576
7.602580886955378
11.75068202216614
10.763007935018155
7.2878734947065915
```

```
np.sqrt((np.array(final_answer['Predict'] - final_answer['Actual'])**2).mean())
```

```
9.103784095360165
```

Minor Mistake

2

Minor Mistake

Compose an answer

ในขั้นตอนเตรียม test data ได้ใช้ ffill() 1 ครั้ง ตามด้วย bfill() อีกหนึ่งครั้ง
ข้อผิดพลาดเกิดที่การ bfill() 1 ครั้ง เนื่องจากได้ทำการเติมค่า exogeneous value
ณ เวลา 07/01/20 0:00 ใน test data จากอนาคตมา 1 ค่า โดยดึงมาจาก เวลา 07/01/20 1:00

จะทำการแก้ไขโดยการใช้ข้อมูล exogeneous value จาก valid data ไปทำการ pad ณ จุดๆ นั้นแทนการทำ bfill() ดังกล่าว
เพื่อไม่ใช้ข้อมูลในอนาคตจากเวลา 07/01/20 1:00 มาแทนที่เวลา 07/01/20 0:00

Minor Mistake

./custom_function/csv.df

```
temp_train_df = pd.read_csv('datasci_dataset_2022/'+temp_train_filename, names=['Time', 'Temp'], skiprows=1)
temp_train_df['Time'] = pd.to_datetime(temp_train_df['Time'])
temp_df = pd.read_csv('datasci_dataset_2022/'+temp_filename, names=['Time', 'Temp'], skiprows=1)
temp_df['Time'] = pd.to_datetime(temp_df['Time'])

temp_df = temp_df.append(temp_train_df.iloc[-1]).reset_index()
temp_df = temp_df.drop(columns=['index'])
temp_df = temp_df.sort_values(by=['Time'])

temp_df.set_index(temp_df['Time'], inplace=True)
temp_df.drop(columns={'Time'}, inplace=True)
temp_df.columns = ['Temp']
temp_df = temp_df.resample('h').ffill()
# temp_df = temp_df.bfill()
```

Minor Mistake

./custom_function/csv.df

```
wind_train_df = pd.read_csv('datasci_dataset_2022/'+wind_train_filename, names=['Time', 'WindSpeed', 'WindDir'], skiprows=1)
wind_train_df['Time'] = pd.to_datetime(wind_train_df['Time'])
wind_df = pd.read_csv('datasci_dataset_2022/'+wind_filename, names=['Time', 'WindSpeed', 'WindDir'], skiprows=1)
wind_df['Time'] = pd.to_datetime(wind_df['Time'])

wind_df = wind_df.append(wind_train_df.iloc[-1]).reset_index()
wind_df = wind_df.drop(columns=['index'])
wind_df = wind_df.sort_values(by=['Time'])

wind_df.set_index(wind_df['Time'], inplace=True)
wind_df.drop(columns={'Time'}, inplace=True)
wind_df.columns = ['WindSpeed', 'WindDir']
# forward filling
wind_df = wind_df.resample('h').ffill()
# wind_df = wind_df.bfill()
# wind_df.index = pd.DatetimeIndex(wind_df.index)
```

Parameter transformation : wind direction

```
train_set = pd.read_csv('6Hsampled_data_set/CNY/train_set.csv')
train_set['WindDirSin'] = np.sin(train_set['WindDir'])*10 + 10
train_set['WindDirCos'] = np.cos(train_set['WindDir'])*10 + 10
train_set['Time'] = pd.to_datetime(train_set['Time'])
train_set = train_set.set_index('Time')

valid_set = pd.read_csv('6Hsampled_data_set/CNY/valid_set.csv')
valid_set['WindDirSin'] = np.sin(valid_set['WindDir'])*10 + 10
valid_set['WindDirCos'] = np.cos(valid_set['WindDir'])*10 + 10
valid_set['Time'] = pd.to_datetime(valid_set['Time'])
valid_set = valid_set.set_index('Time')

test_set = pd.read_csv('6Hsampled_data_set/CNY/test_set.csv')
test_set['WindDirSin'] = np.sin(test_set['WindDir'])*10 + 10
test_set['WindDirCos'] = np.cos(test_set['WindDir'])*10 + 10
test_set['Time'] = pd.to_datetime(test_set['Time'])
test_set = test_set.set_index('Time')
```

```
train_set = pd.read_csv('6Hsampled_data_set/BKK/train_set.csv')
train_set['WindDirSin'] = np.sin(train_set['WindDir'])*10 + 10
train_set['WindDirCos'] = np.cos(train_set['WindDir'])*10 + 10
train_set['Time'] = pd.to_datetime(train_set['Time'])
train_set = train_set.set_index('Time')

valid_set = pd.read_csv('6Hsampled_data_set/BKK/valid_set.csv')
valid_set['WindDirSin'] = np.sin(valid_set['WindDir'])*10 + 10
valid_set['WindDirCos'] = np.cos(valid_set['WindDir'])*10 + 10
valid_set['Time'] = pd.to_datetime(valid_set['Time'])
valid_set = valid_set.set_index('Time')

test_set = pd.read_csv('6Hsampled_data_set/BKK/test_set.csv')
test_set['WindDirSin'] = np.sin(test_set['WindDir'])*10 + 10
test_set['WindDirCos'] = np.cos(test_set['WindDir'])*10 + 10
test_set['Time'] = pd.to_datetime(test_set['Time'])
test_set = test_set.set_index('Time')
```

```
train_set = pd.read_csv('6Hsampled_data_set/KKC/train_set.csv')
train_set['WindDirSin'] = np.sin(train_set['WindDir'])*10 + 10
train_set['WindDirCos'] = np.cos(train_set['WindDir'])*10 + 10
train_set['Time'] = pd.to_datetime(train_set['Time'])
train_set = train_set.set_index('Time')

valid_set = pd.read_csv('6Hsampled_data_set/KKC/valid_set.csv')
valid_set['WindDirSin'] = np.sin(valid_set['WindDir'])*10 + 10
valid_set['WindDirCos'] = np.cos(valid_set['WindDir'])*10 + 10
valid_set['Time'] = pd.to_datetime(valid_set['Time'])
valid_set = valid_set.set_index('Time')

test_set = pd.read_csv('6Hsampled_data_set/KKC/test_set.csv')
test_set['WindDirSin'] = np.sin(test_set['WindDir'])*10 + 10
test_set['WindDirCos'] = np.cos(test_set['WindDir'])*10 + 10
test_set['Time'] = pd.to_datetime(test_set['Time'])
test_set = test_set.set_index('Time')
```

```
train_set = pd.read_csv('6Hsampled_data_set/RAY/train_set.csv')
train_set['WindDirSin'] = np.sin(train_set['WindDir'])*10 + 10
train_set['WindDirCos'] = np.cos(train_set['WindDir'])*10 + 10
train_set['Time'] = pd.to_datetime(train_set['Time'])
train_set = train_set.set_index('Time')

valid_set = pd.read_csv('6Hsampled_data_set/RAY/valid_set.csv')
valid_set['WindDirSin'] = np.sin(valid_set['WindDir'])*10 + 10
valid_set['WindDirCos'] = np.cos(valid_set['WindDir'])*10 + 10
valid_set['Time'] = pd.to_datetime(valid_set['Time'])
valid_set = valid_set.set_index('Time')

test_set = pd.read_csv('6Hsampled_data_set/RAY/test_set.csv')
test_set['WindDirSin'] = np.sin(test_set['WindDir'])*10 + 10
test_set['WindDirCos'] = np.cos(test_set['WindDir'])*10 + 10
test_set['Time'] = pd.to_datetime(test_set['Time'])
test_set = test_set.set_index('Time')
```

```
train_set = pd.read_csv('6Hsampled_data_set/SURAT/train_set.csv')
train_set['WindDirSin'] = np.sin(train_set['WindDir'])*10 + 10
train_set['WindDirCos'] = np.cos(train_set['WindDir'])*10 + 10
train_set['Time'] = pd.to_datetime(train_set['Time'])
train_set = train_set.set_index('Time')

valid_set = pd.read_csv('6Hsampled_data_set/SURAT/valid_set.csv')
valid_set['WindDirSin'] = np.sin(valid_set['WindDir'])*10 + 10
valid_set['WindDirCos'] = np.cos(valid_set['WindDir'])*10 + 10
valid_set['Time'] = pd.to_datetime(valid_set['Time'])
valid_set = valid_set.set_index('Time')

test_set = pd.read_csv('6Hsampled_data_set/SURAT/test_set.csv')
test_set['WindDirSin'] = np.sin(test_set['WindDir'])*10 + 10
test_set['WindDirCos'] = np.cos(test_set['WindDir'])*10 + 10
test_set['Time'] = pd.to_datetime(test_set['Time'])
test_set = test_set.set_index('Time')
```

```
train_set = pd.read_csv('6Hsampled_data_set/SARA/train_set.csv')
train_set['WindDirSin'] = np.sin(train_set['WindDir'])*10 + 10
train_set['WindDirCos'] = np.cos(train_set['WindDir'])*10 + 10
train_set['Time'] = pd.to_datetime(train_set['Time'])
train_set = train_set.set_index('Time')

valid_set = pd.read_csv('6Hsampled_data_set/SARA/valid_set.csv')
valid_set['WindDirSin'] = np.sin(valid_set['WindDir'])*10 + 10
valid_set['WindDirCos'] = np.cos(valid_set['WindDir'])*10 + 10
valid_set['Time'] = pd.to_datetime(valid_set['Time'])
valid_set = valid_set.set_index('Time')

test_set = pd.read_csv('6Hsampled_data_set/SARA/test_set.csv')
test_set['WindDirSin'] = np.sin(test_set['WindDir'])*10 + 10
test_set['WindDirCos'] = np.cos(test_set['WindDir'])*10 + 10
test_set['Time'] = pd.to_datetime(test_set['Time'])
test_set = test_set.set_index('Time')
```

```python
def sarimax_randomsearch(y_train, y_test, pdq, PDQs, y_val = None, x_train = None, x_test = None, x_val = None, model_exog=None, verbose=0, n_rand=10):
    '''
    Input:
        y_train: training data
        y_test: test data
        pdq : ARIMA combinations
        pdqs : seasonal ARIMA combinations
        x_train: exogenous training data
        x_test: exogenous test data

    Return:
        Returns dataframe of parameter combinations with the least RMSE
    '''

    ans_df = pd.DataFrame(columns=['pdq', 'pdqs', 'rmse'])

    save_comb = set()

    i = 0
    while i!=n_rand:
        comb = random.sample(pdq, 1)[0]
        combs = random.sample(PDQs, 1)[0]

        if (comb+combs) in save_comb:
            continue

        save_comb.add(comb+combs)

        p, d, q = comb[0], comb[1], comb[2]
        P, D, Q = combs[0], combs[1], combs[2]
        if (d <= 1) and (D <= 1) and (P <= 1) and (Q <= 1):
            model = MinimalSARIMAX(y_train, comb, combs, exog=x_train)
            model.fit(lr=1e-6, lr_decay=0.999, verbose=0)

            if (y_val is None):
                y_pred, err = model.predict(y_test, y_exog=x_test, verbose=verbose)
                rmse = model.scoring(y_pred, y_test)

            else:
                Result = model.predict_step(y_val, y_test, val_X_exog=x_val, y_exog=x_test,
                                            model_exog=model_exog, lr=np.array([1e-6, 1e-6, 1e-6, 1e-6, 1e-6]), lr_decay=0.9995,
                                            learn=True, verbose=verbose, verbose_rmse=0)

                _, y_pred_sav, _ = Result

                rmse = model.scoring(y_pred_sav.iloc[:,[1]], y_pred_sav.iloc[:,[2]])

            print(f"ITER#{i} {comb} {combs} {rmse}") ; i=i+1

            ans_df = ans_df.append({'pdq':comb, 'pdqs':combs, 'rmse':rmse}, ignore_index=True)
        else: continue

    # Sort and return a combination with the lowest RMSE
    ans_df = ans_df.sort_values(by=['rmse'],ascending=True)

    return ans_df
```

Model Train:

custom randomizedsearch

Model Train: Training Step

STEP1: Fill Tuning Parameters (from randomizedsearch)

Tuning Parameters

```
order = (2, 0, 2)
seasonal_order = (1, 1, 1, 1461)

exog_order = {}
exog_order['Temp'] = (2, 0, 1)
exog_order['WindSpeed'] = (2, 0, 2)
exog_order['WindDirSin'] = (0, 1, 1)
exog_order['WindDirCos'] = (2, 0, 0)

exog_seasonal_order = {}
exog_seasonal_order['Temp'] = (1, 0, 0, 1461)
exog_seasonal_order['WindSpeed'] = (0, 0, 1, 1461)
exog_seasonal_order['WindDirSin'] = (1, 0, 1, 1461)
exog_seasonal_order['WindDirCos'] = (0, 1, 1, 1461)

exog_columns = ['Temp', 'WindSpeed', 'WindDirSin', 'WindDirCos']
```

Model Train: Training Step

STEP2: Initialize model

```
model = {}
model_exog = {}

model = MinimalSARIMAX(train_set[['PM25']],
                       order,
                       seasonal_order,
                       exog=train_set[exog_columns])
model.fit(lr=1e-6, lr_decay=0.999 ,verbose=0)

model_exog = {}
for exog in exog_columns:
    model_exog[exog] = MinimalSARIMAX(train_set[[exog]],
                                      exog_order[exog],
                                      exog_seasonal_order[exog])
```

Model Train: Training Step

STEP2: Initialize model

```
model = {}
model_exog = {}

model = MinimalSARIMAX(train_set[['PM25']],
                       order,
                       seasonal_order,
                       exog=train_set[exog_columns])
model.fit(lr=1e-6, lr_decay=0.999 ,verbose=0)

model_exog = {}
for exog in exog_columns:
    model_exog[exog] = MinimalSARIMAX(train_set[[exog]],
                                      exog_order[exog],
                                      exog_seasonal_order[exog])
```

Model Train: Training Step

STEP3: Train exogenous model first!

```
model.fit(lr=1e-6, lr_decay=0.999 ,verbose=0)

for exog in exog_columns:
        model_exog[exog].fit(lr=1e-6, lr_decay=0.999, verbose=0)
        n_iter = 1
        if exog=='WindSpeed': n_iter=10
        _ = model_exog[exog].predict_step(train_set[[exog]],
                                          valid_set[[exog]],
                                          lr=np.array([1e-6]), lr_decay=0.999875, lr_decay_iter=0.95,
                                          step=12, n_iter=n_iter, learn=True, verbose=1)
```

```
exog_columns = ['Temp', 'WindSpeed', 'WindDirSin', 'WindDirCos']
```

Model Train: Training Step

STEP3: Train exogenous model first! (cont.)

```
100%|██████████| 4383/4383 [04:02<00:00, 18.08it/s]
ITER#0  RMSE:5.421074495132013
100%|██████████| 4383/4383 [04:19<00:00, 16.86it/s]
ITER#0  RMSE:15.038015894104312
100%|██████████| 4383/4383 [04:46<00:00, 15.32it/s]
ITER#1  RMSE:14.644124674690037
100%|██████████| 4383/4383 [04:20<00:00, 16.82it/s]
ITER#2  RMSE:14.32056132829562
100%|██████████| 4383/4383 [04:21<00:00, 16.79it/s]
ITER#3  RMSE:14.118306378020314
100%|██████████| 4383/4383 [03:58<00:00, 18.41it/s]
ITER#4  RMSE:13.940105546971603
100%|██████████| 4383/4383 [03:49<00:00, 19.09it/s]
ITER#5  RMSE:13.76131440183912
```

```
ITER#6  RMSE:13.575419109702153
100%|██████████| 4383/4383 [04:18<00:00, 16.98it/s]
ITER#7  RMSE:13.377829303675995
100%|██████████| 4383/4383 [04:15<00:00, 17.12it/s]
ITER#8  RMSE:13.16839470977157
100%|██████████| 4383/4383 [04:18<00:00, 16.99it/s]
ITER#9  RMSE:12.94523483497169
100%|██████████| 4383/4383 [04:41<00:00, 15.57it/s]
ITER#0  RMSE:10.305962807024063
100%|██████████| 4383/4383 [05:40<00:00, 12.89it/s]
ITER#0  RMSE:9.347175594155463
```

Model Train: Training Step

STEP4: Train PM2.5 model with exogenous model 10 iters

```
Result_train = model.predict_step(train_set[['PM25']],
                                  valid_set[['PM25']],
                                  val_X_exog=train_set[exog_columns],
                                  y_exog=valid_set[exog_columns],
                                  model_exog=model_exog,
                                  lr=np.array([5e-6, 5e-7, 5e-7, 5e-7, 5e-7]), lr_decay=0.999, lr_decay_iter=0.95,
                                  step=12, n_iter=10, learn=True, verbose=1)

train_pred_sav, val1_pred_sav, Error_save = Result_train
```

Model Train: Training Step

STEP4: Train PM2.5 model with exogenous model 10 iters (cont.)

```
100%|██████████| 4383/4383 [15:22<00:00,  4.75it/s]
ITER#0  RMSE:16.911281303234457
100%|██████████| 4383/4383 [13:24<00:00,  5.45it/s]
ITER#1  RMSE:16.530480118586787
100%|██████████| 4383/4383 [13:25<00:00,  5.44it/s]
ITER#2  RMSE:15.733736727622643
100%|██████████| 4383/4383 [13:26<00:00,  5.44it/s]
ITER#3  RMSE:14.98472105122573
100%|██████████| 4383/4383 [13:26<00:00,  5.43it/s]
ITER#4  RMSE:14.191440874527773
```

```
100%|██████████| 4383/4383 [13:25<00:00,  5.44it/s]
ITER#5  RMSE:13.464046897119115
100%|██████████| 4383/4383 [13:30<00:00,  5.41it/s]
ITER#6  RMSE:12.955801859695306
100%|██████████| 4383/4383 [13:29<00:00,  5.41it/s]
ITER#7  RMSE:12.713139714664603
100%|██████████| 4383/4383 [13:35<00:00,  5.38it/s]
ITER#8  RMSE:12.638095028669687
100%|██████████| 4383/4383 [13:31<00:00,  5.40it/s]
ITER#9  RMSE:12.621512650333633
```

Model Train: Training Step

STEP5: Predict test set

```
Result_test = model.predict_step(valid_set[['PM25']],
                                 test_set[['PM25']],
                                 val_X_exog=valid_set[exog_columns],
                                 y_exog=test_set[exog_columns],
                                 model_exog=model_exog,
                                 lr=np.array([5e-7, 5e-7, 5e-7, 5e-7, 5e-7]), lr_decay=0.999875,
                                 step=12, n_iter=1, learn=True, verbose=1)

val2_pred_sav, test_pred_sav, Error_save = Result_test
```

```
100%|██████████| 2779/2779 [07:46<00:00,  5.95it/s]

ITER#0  RMSE:11.40674019718906
```

Model Train: Training Step

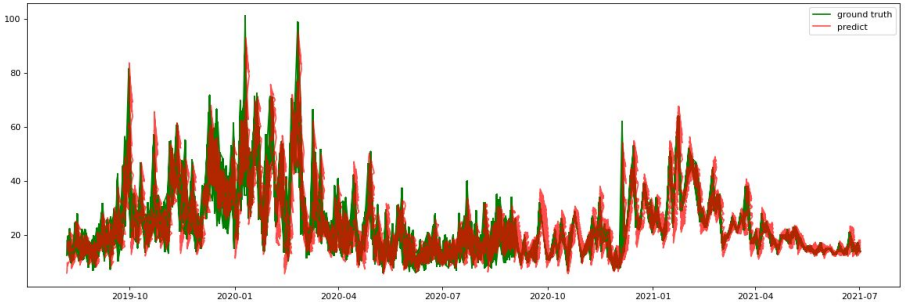
STEP5: Predict test set

```
Result_test = model.predict_step(valid_set[['PM25']],
                                 test_set[['PM25']],
                                 val_X_exog=valid_set[exog_columns],
                                 y_exog=test_set[exog_columns],
                                 model_exog=model_exog,
                                 lr=np.array([5e-7, 5e-7, 5e-7, 5e-7, 5e-7]), lr_decay=0.999875,
                                 step=12, n_iter=1, learn=True, verbose=1)

val2_pred_sav, test_pred_sav, Error_save = Result_test
```

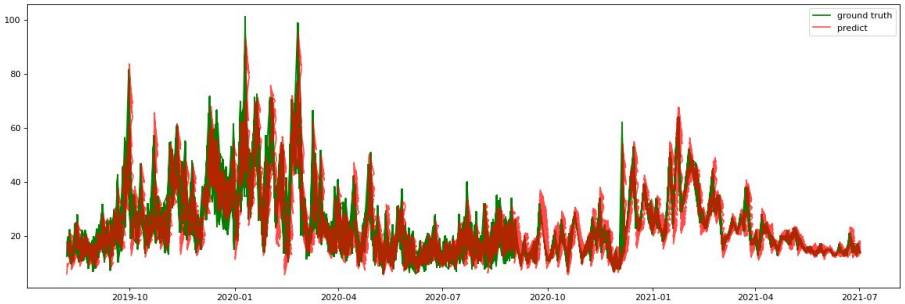
```
100%|██████████| 2779/2779 [07:46<00:00,  5.95it/s]

ITER#0  RMSE:11.40674019718906
```

Model Train: MinimalSARIMAX

Bangkok
7.1726

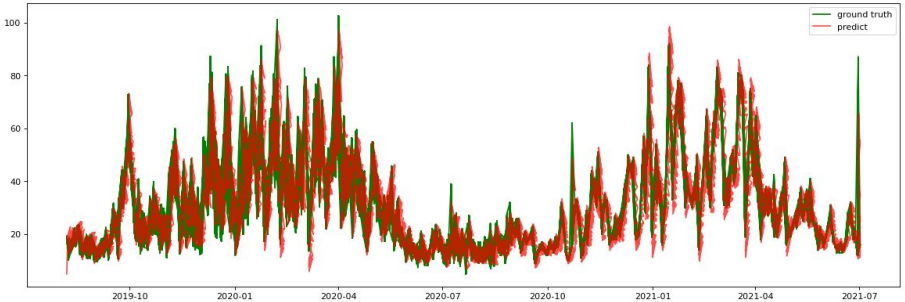


Chiangmai
11.8930

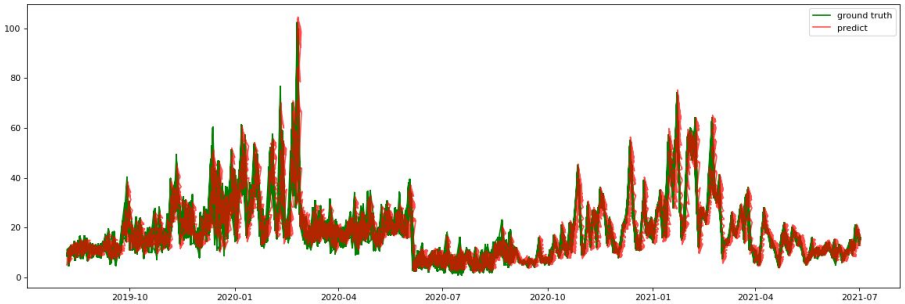


Model Train: MinimalSARIMAX

Khonkaen
9.5040

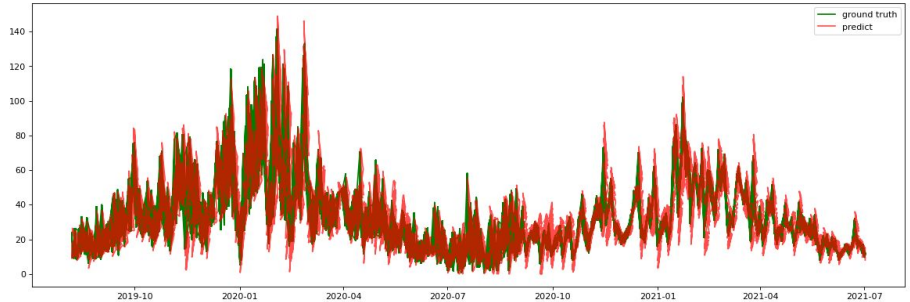


Rayong
6.7187

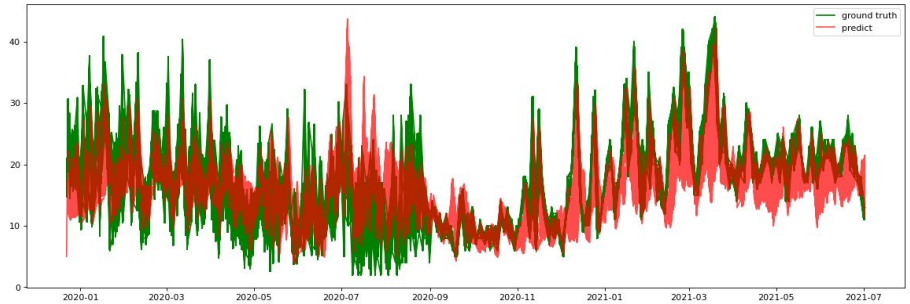


Model Train: MinimalSARIMAX

Saraburi
11.4067



Surat
6.6751



YOUR RECENT SUBMISSION

testing_submission.csv

Submitted by Fact Maneepakorn · Submitted 2 days ago

Score: 9.10378

↓ Jump to your leaderboard position

RMSE

```
for sb in [sb_bkk, sb_cnx, sb_ray, sb_sara, sb_kkc, sb_surat]:
    sb[sb['Predict']<0] = 0
    diff = np.sqrt((np.array(sb['Predict'] - sb['Actual'])**2).mean())
    print(diff)
```

```
6.24731247042558
9.744932835576
7.602580886955378
11.75068202216614
10.763007935018155
7.2878734947065915
```

```
np.sqrt((np.array(final_answer['Predict'] - final_answer['Actual'])**2).mean())
```

```
9.103784095360165
```