arima_lstm_tryout

September 7, 2019

[49]:
```
import gc
gc.collect()
```

[49]:
```
43350
```

[ ]:
```
# ARIMA as input for LSTM

# https://www.kaggle.com/muonneutrino/wikipedia-traffic-data-exploration
```

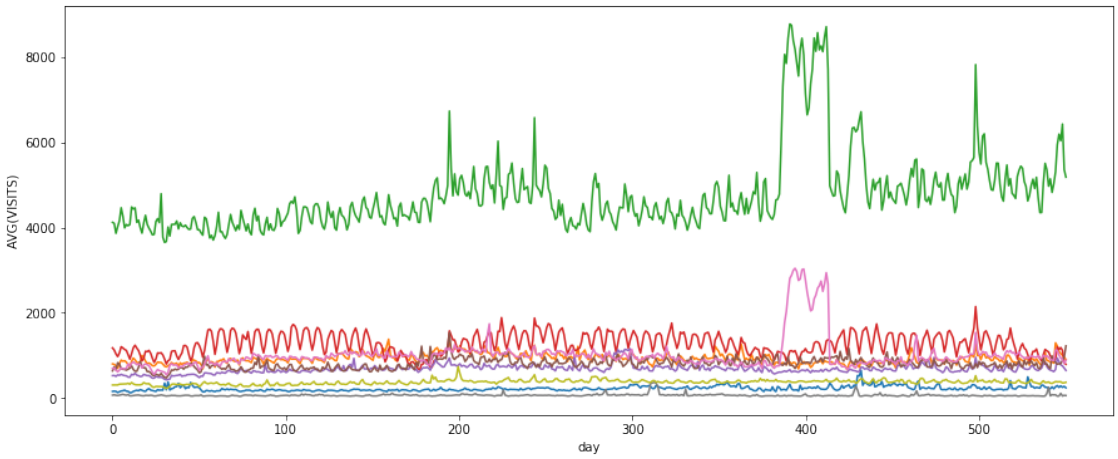
[62]:
```
%matplotlib inline
from matplotlib import pyplot as plt

import numpy as np
import pandas as pd

from statsmodels.tsa.arima_model import ARIMA
from statsmodels.tsa.stattools import pacf, acf

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler

from keras.models import Sequential
from keras.layers import LSTM, Dense
```

[14]:
```
# Load datasets

train_df = pd.read_csv('data/train_1.csv', header=0)
```

[15]:
```
# Replace 'Page' with 'Language' feature

train_splitted_df = train_df['Page'].str.rsplit( pat='_', n=3, expand=True )
train_splitted_df.columns = [ 'name', 'project', 'access', 'agent' ]

# reformatting:
train_df['Page'] = train_splitted_df['project'].str[:2]
train_df.rename( columns={'Page': 'lang'}, inplace=True )

# clean up
del train_splitted_df
```

[20]:
```
# Group data by languages

language_groupby = train_df.groupby( by='lang' )
```

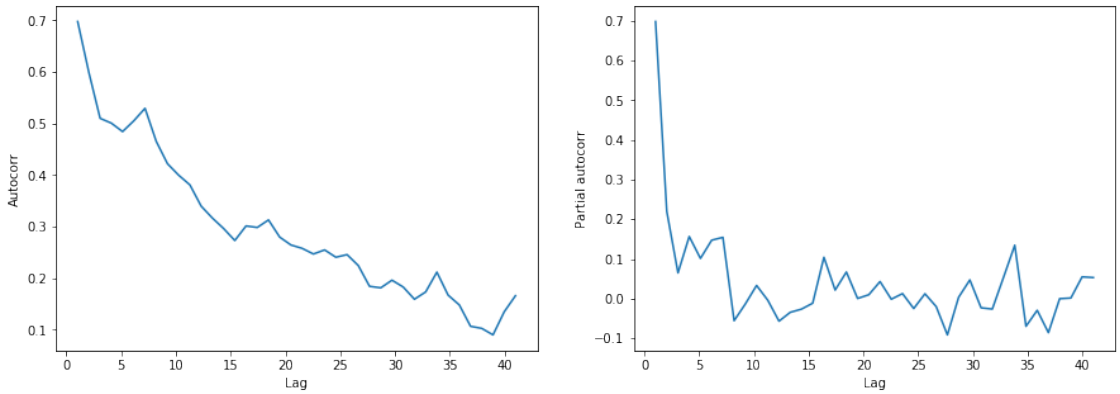
[28]:
```
# Plot daily views by language

days = np.linspace(0, 550, 550)

fig = plt.figure( figsize=(15, 6) )

for name, group in language_groupby:
    plt.plot( days, group.iloc[:, 1:].mean(axis=0), label=name )

plt.ylabel('AVG(VISITS)')
plt.xlabel('day')
plt.show()
```



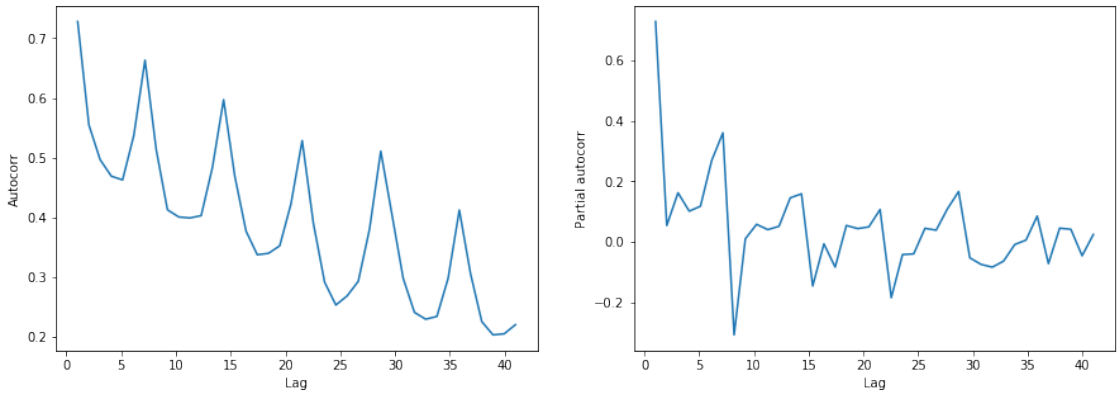
[43]:
```
for name, group in language_groupby:
    fig, [ax_0, ax_1] = plt.subplots( 1, 2, figsize=(15, 5) )

    mean_values = group.iloc[:, 1:].mean(axis=0)

    autocorr_values = acf(mean_values, fft=True)
    partial_autocorr_values = pacf(mean_values)

    x_values = np.linspace(0, len(partial_autocorr_values),␣
 ↪len(partial_autocorr_values))

    ax_0.plot( x_values[1:], autocorr_values[1:] )
    ax_1.plot( x_values[1:], partial_autocorr_values[1:] )

    ax_0.set_xlabel('Lag'); ax_0.set_ylabel('Autocorr')
```

```
    ax_1.set_xlabel('Lag'); ax_1.set_ylabel('Partial autocorr')

    display('Autocorr / Partial Autocorr for {0}'.format(name)); plt.show()
```

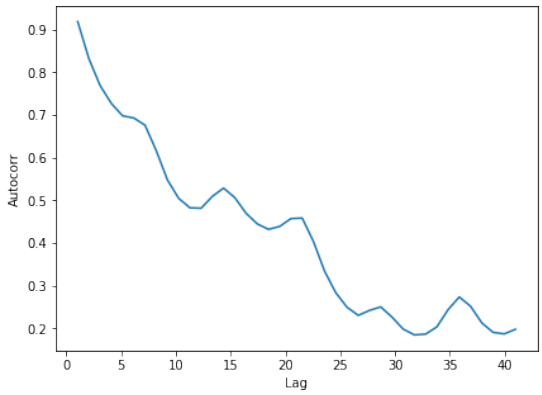
'Autocorr / Partial Autocorr for co'

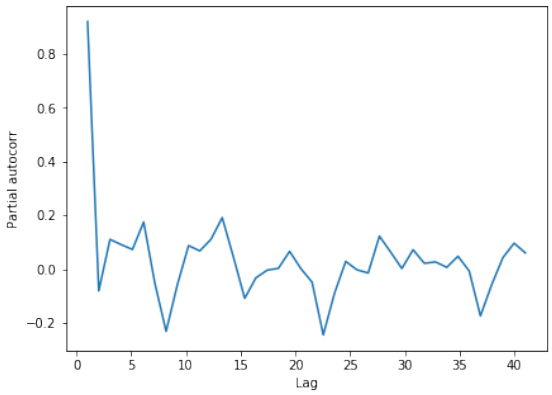


'Autocorr / Partial Autocorr for de'

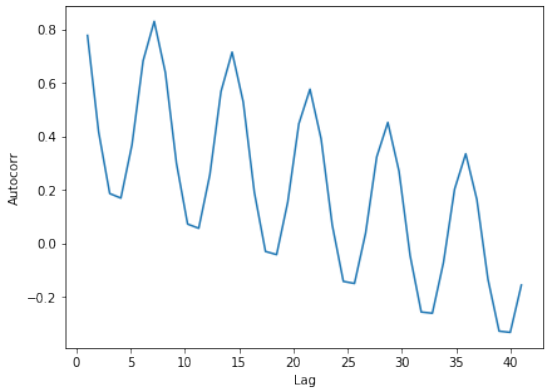


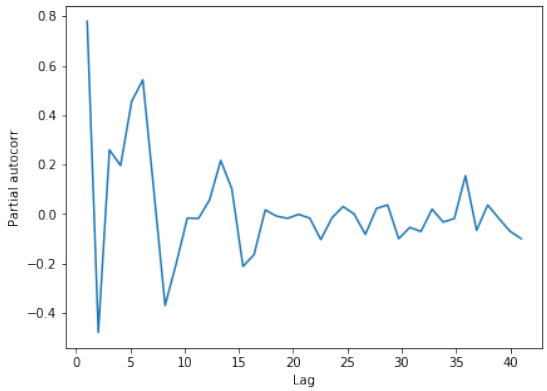
'Autocorr / Partial Autocorr for en'



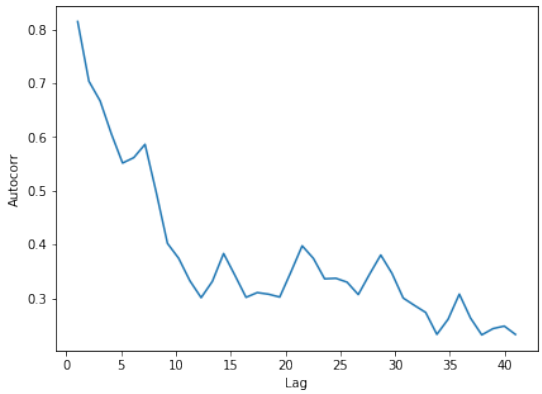


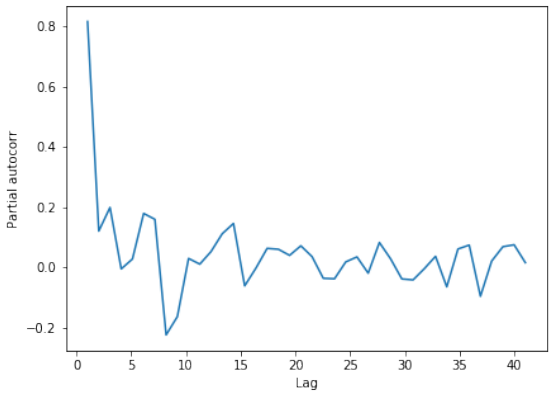
'Autocorr / Partial Autocorr for es'



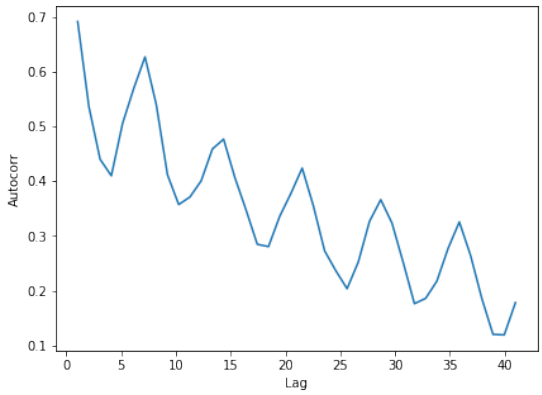


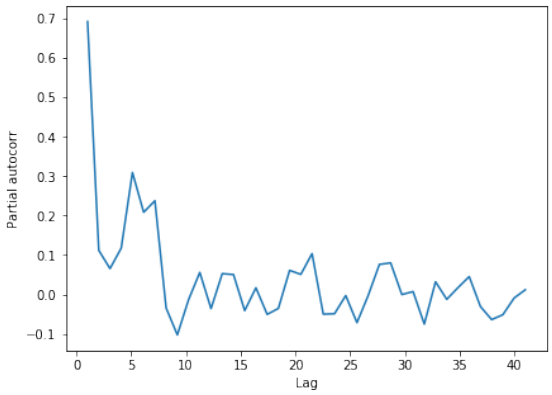
'Autocorr / Partial Autocorr for fr'



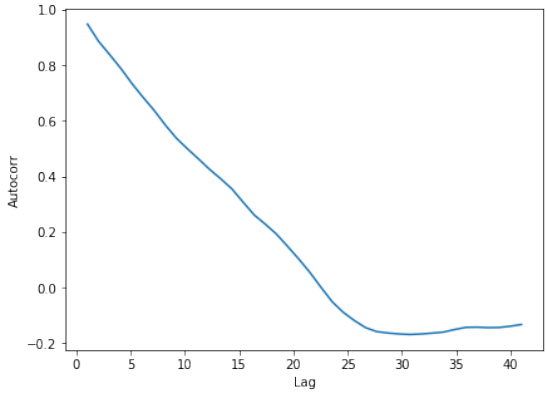


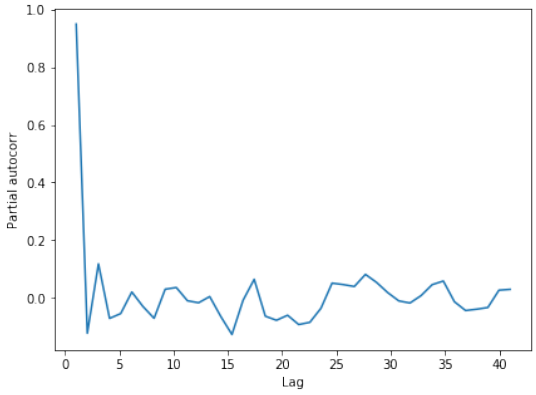
'Autocorr / Partial Autocorr for ja'



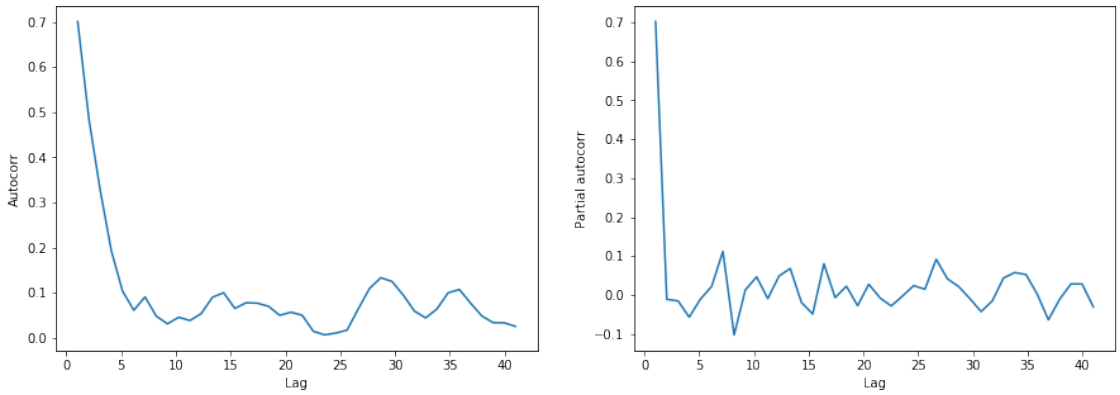


'Autocorr / Partial Autocorr for ru'

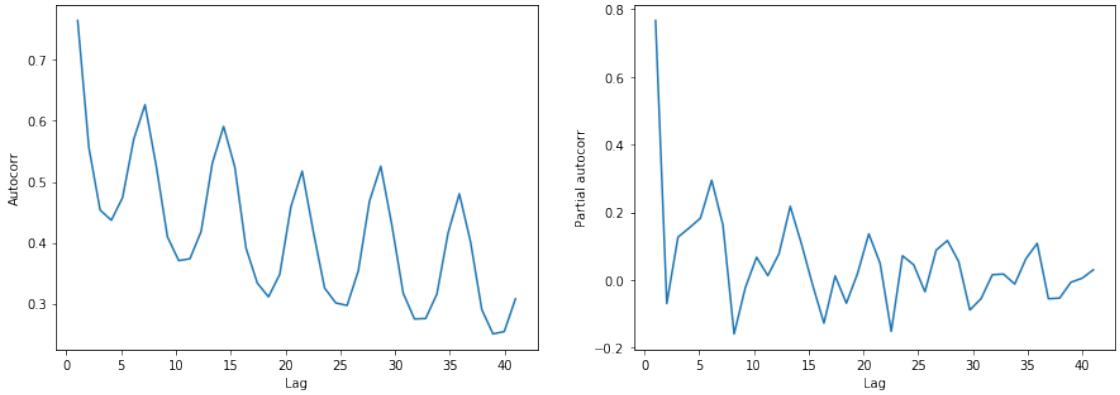




'Autocorr / Partial Autocorr for ww'



'Autocorr / Partial Autocorr for zh'

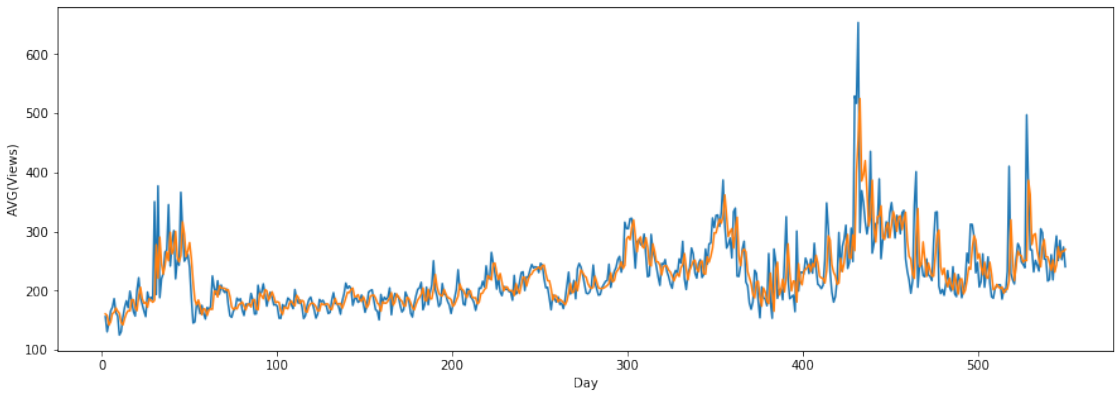


[47]:
```
# TONOTE from these graphs:
# 1. de, es, ja, zh -> have lag=7
```

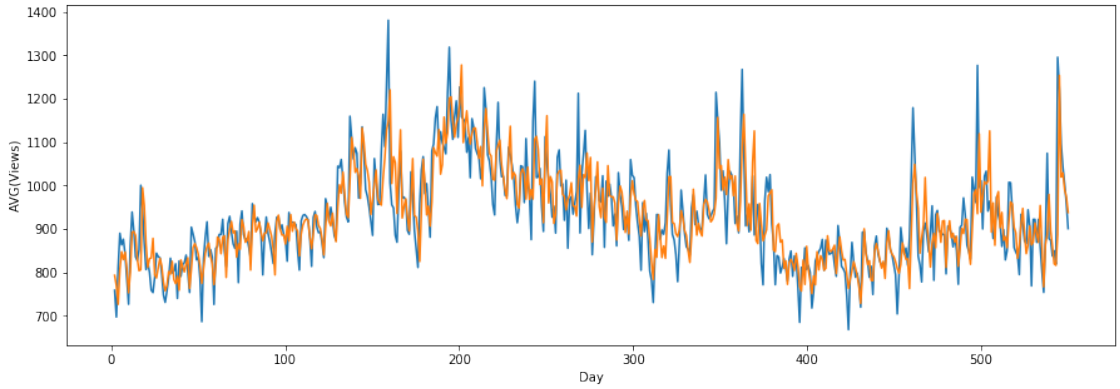
[59]:
```
# For now, use MA=1

# https://machinelearningmastery.com/
 ↪arima-for-time-series-forecasting-with-python/
# The parameters of the ARIMA model are defined as follows:
# p: The number of lag observations included in the model, also called the lag␣
 ↪order.
# d: The number of times that the raw observations are differenced, also called␣
 ↪the degree of differencing.
# q: The size of the moving average window, also called the order of moving␣
 ↪average.

arima_lang_params = {
```

```
    'de': [7, 1, 1], 'es': [7, 1, 1], 'ja': [7, 1, 1], 'zh': [7, 1, 1],
    'en': [4, 1, 0], 'na': [4, 1, 0], 'fr': [4, 1, 0], 'ru': [4, 1, 0],
    'ww': [4, 1, 0], 'co': [4, 1, 0]
}

for name, group in language_groupby:
    fig = plt.figure( figsize=(15, 5) )

    x_values = np.linspace(0, 550, 550)  # 600? 700? 1,000? 5,000?

    mean_values = group.iloc[:, 1:].mean(axis=0)

    arima_model = ARIMA( mean_values, arima_lang_params[name] )
    arima_model_fit = arima_model.fit( disp=False )

    views_pred = arima_model_fit.predict(2, 549, typ='levels')

    plt.plot( x_values[2:len(mean_values)], mean_values[2:], label='real data'␣
 ↪)  # real data
    plt.plot( x_values[2:], views_pred, label='arima data') # predicted data

    plt.xlabel('Day'); plt.ylabel('AVG(Views)')
    display('ARIMA for "{0}" language:'.format(name)); plt.show()
```

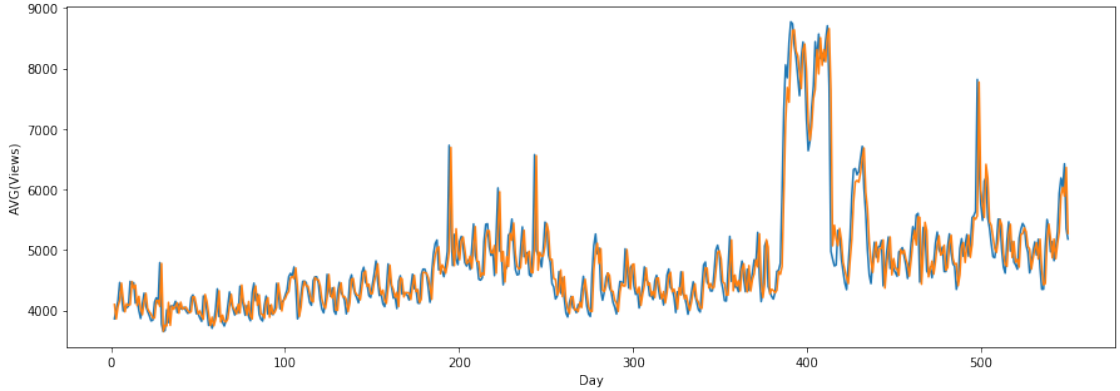
'ARIMA for "co" language:'



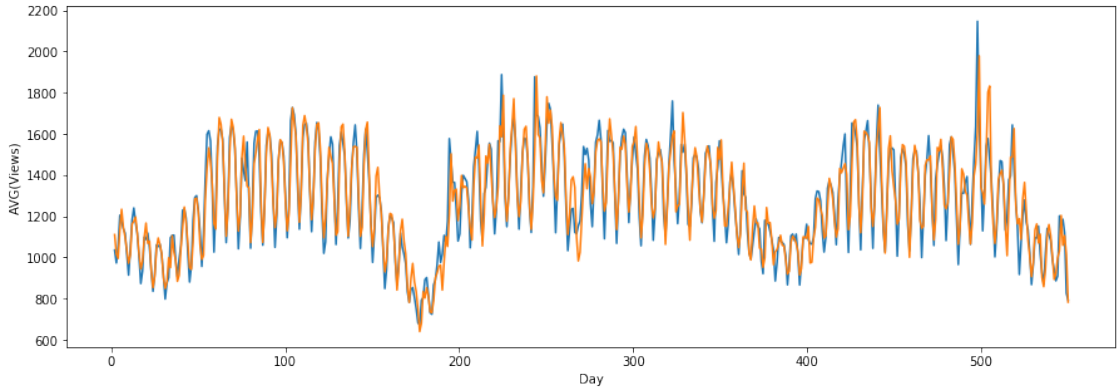
'ARIMA for "de" language:'



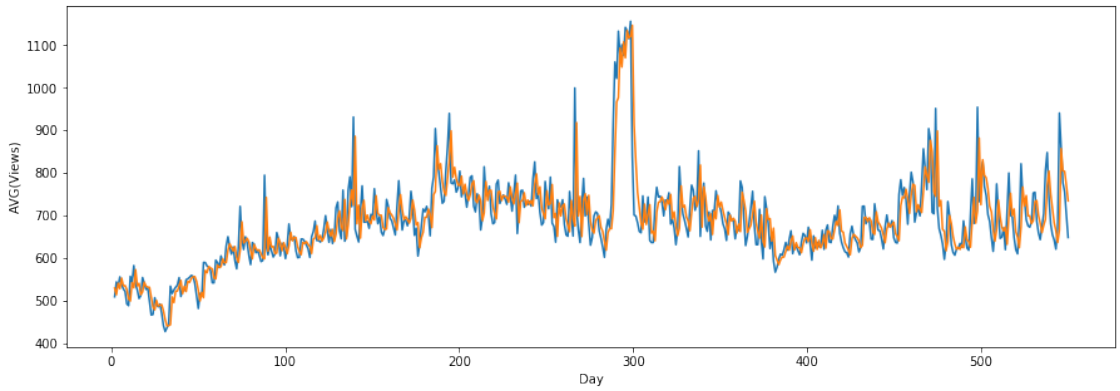
'ARIMA for "en" language:'



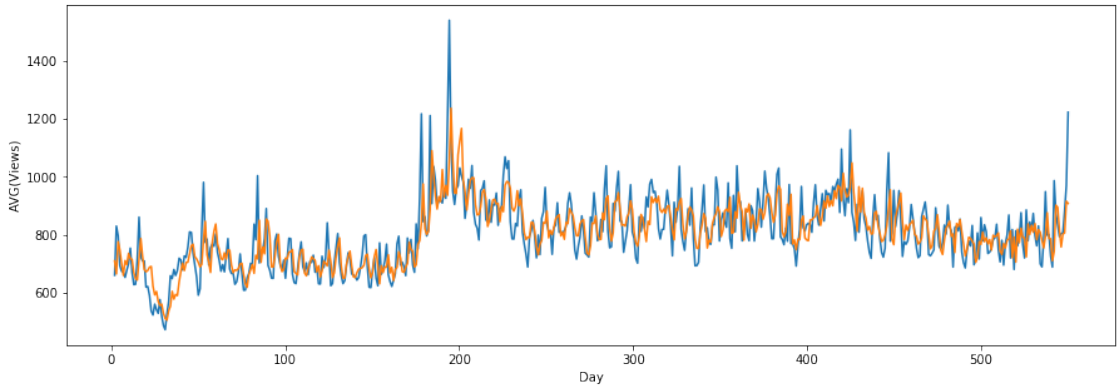
'ARIMA for "es" language:'



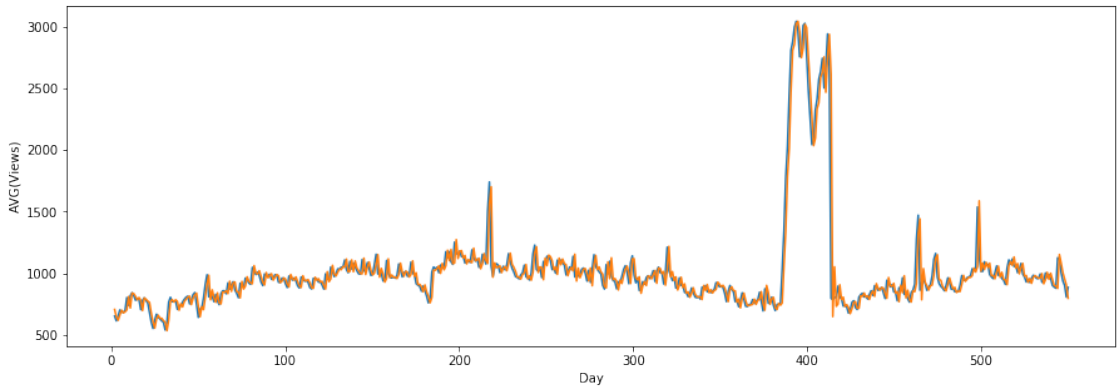
'ARIMA for "fr" language:'



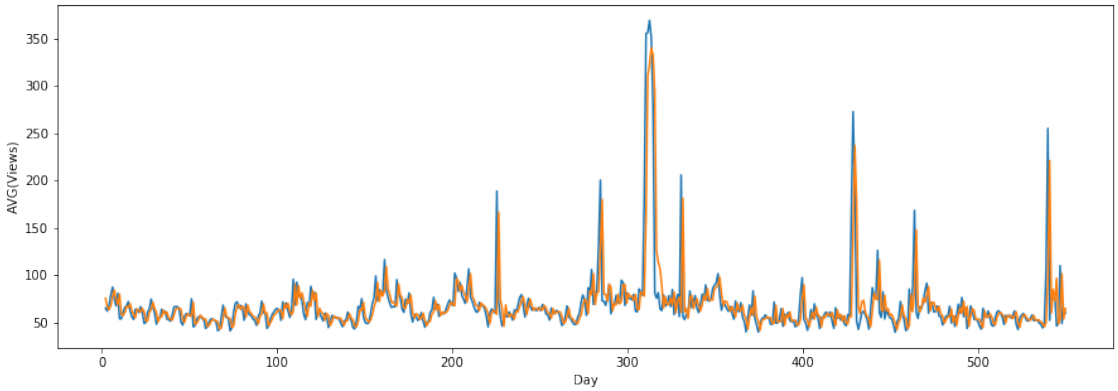
'ARIMA for "ja" language:'



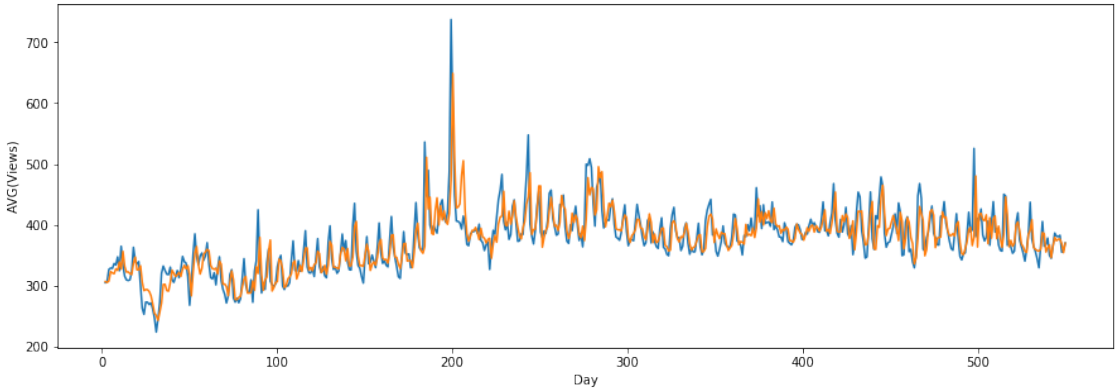
'ARIMA for "ru" language:'



'ARIMA for "ww" language:'



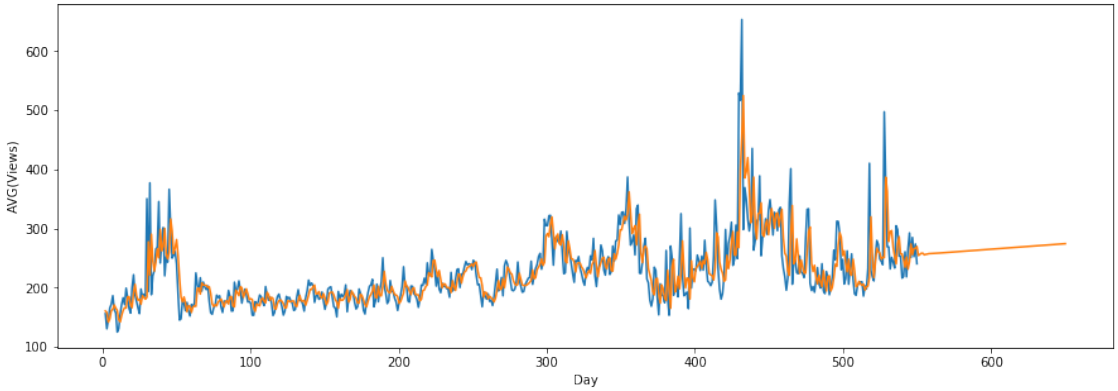
'ARIMA for "zh" language:'



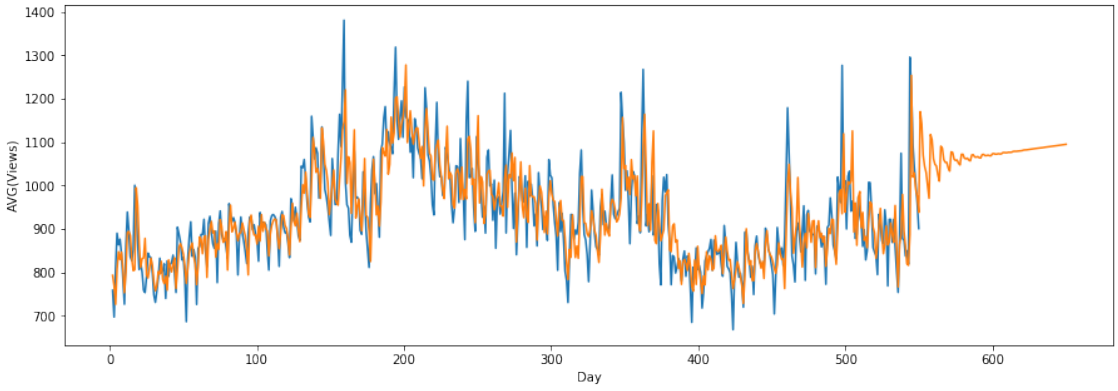
[60]:
```
# Try out ARIMA for LONGER distances (+ 100 points)

for name, group in language_groupby:
    fig = plt.figure( figsize=(15, 5) )

    x_values = np.linspace(0, 650, 650)

    mean_values = group.iloc[:, 1:].mean(axis=0)

    arima_model = ARIMA( mean_values, arima_lang_params[name] )
    arima_model_fit = arima_model.fit( disp=False )
```

```
    views_pred = arima_model_fit.predict(2, 649, typ='levels')

    plt.plot( x_values[2:len(mean_values)], mean_values[2:], label='real data'␣
↪)  # real data
    plt.plot( x_values[2:], views_pred, label='arima data') # predicted data

    plt.xlabel('Day'); plt.ylabel('AVG(Views)')
    display('ARIMA for "{0}" language:'.format(name)); plt.show()
```

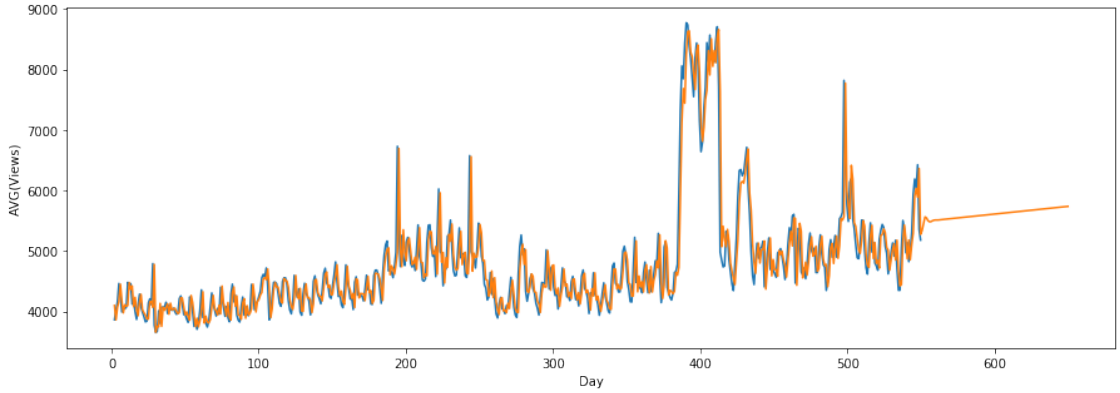
'ARIMA for "co" language:'



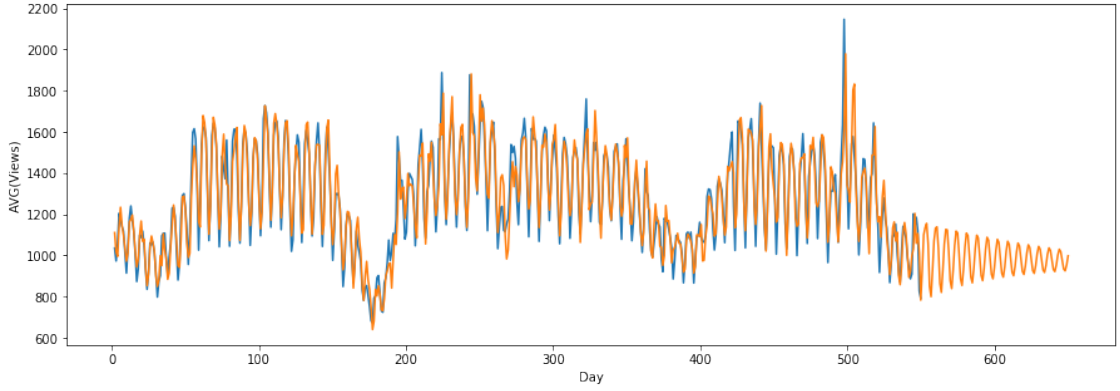
'ARIMA for "de" language:'



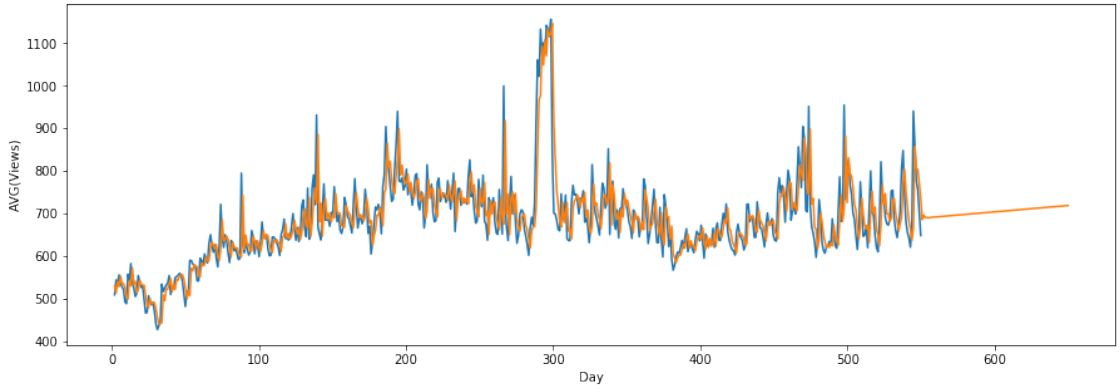
'ARIMA for "en" language:'



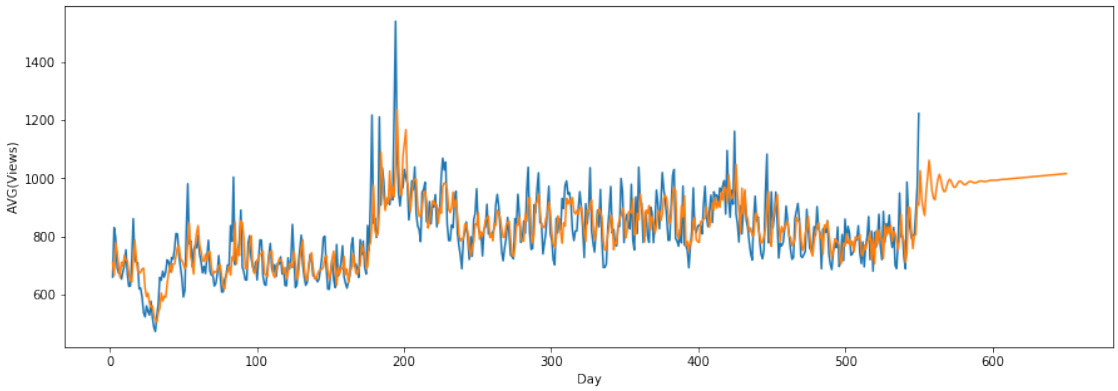
'ARIMA for "es" language:'



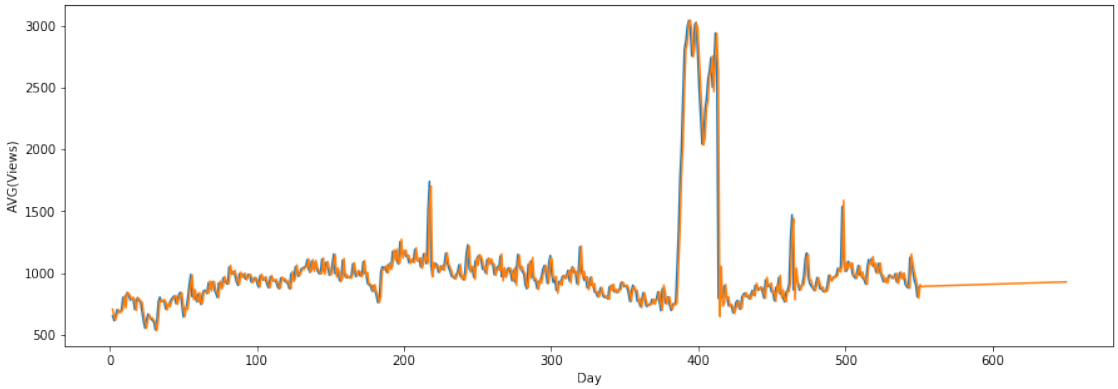
'ARIMA for "fr" language:'



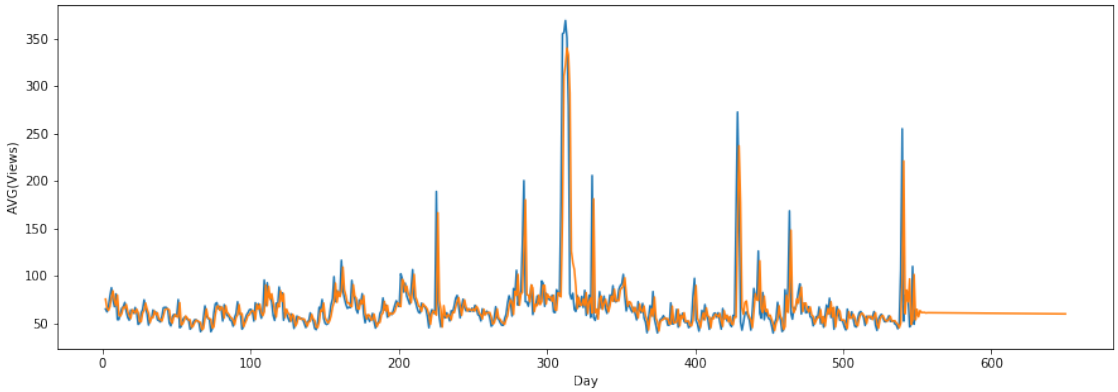
'ARIMA for "ja" language:'



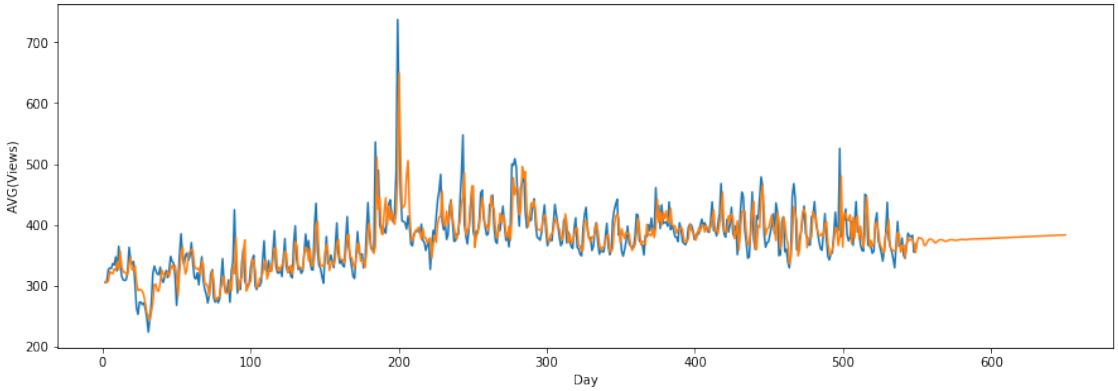
'ARIMA for "ru" language:'



'ARIMA for "ww" language:'



```
'ARIMA for "zh" language:'
```



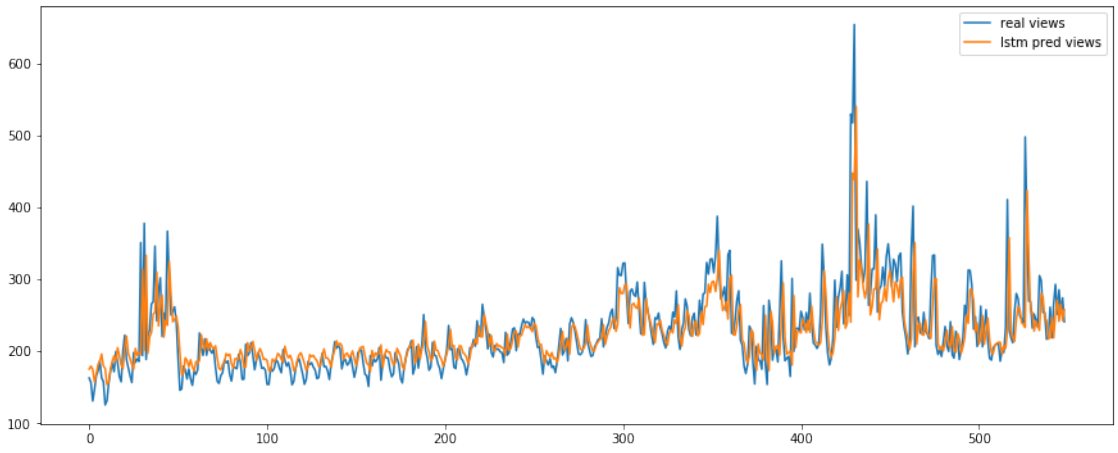
[66]:
```python
def create_lstm_model():
    nn = Sequential()

    nn.add( LSTM(units=16, activation='relu', input_shape=(None, 1)) )  # input
    nn.add( Dense(units=1) )  # output

    nn.compile(optimizer='rmsprop', loss='mean_squared_error')

    return nn
```

[93]:
```python
# For each language, try out building LSTM and predicting views amount

for name, group in language_groupby:

    mean_values = np.array( group.iloc[:, 1:].mean(axis=0) )

    X = mean_values[0:549]
    y = mean_values[1:550]

    X_tr, X_val, y_tr, y_val = train_test_split(X, y, test_size=0.25)

    X_tr = np.reshape(X_tr, (-1, 1))
    y_tr = np.reshape(y_tr, (-1, 1))

    sc = MinMaxScaler()
    X_tr = sc.fit_transform(X_tr)
    y_tr = sc.fit_transform(y_tr)

    # lstm
```

```
    print('lstm...')
    X_tr = np.reshape(X_tr, (411, 1, 1))
    nn = create_lstm_model()
    nn.fit( X_tr, y_tr, batch_size=8, epochs=100, verbose=0 )

    # predict
    print('predicting...')
    inputs = X
    inputs = np.reshape(inputs,(-1,1))
    inputs = sc.transform(inputs)
    inputs = np.reshape(inputs, (549,1,1))
    y_pred = nn.predict(inputs)
    y_pred = sc.inverse_transform(y_pred)

    # plot
    display('Language "{0}"'.format(name))
    fig = plt.figure( figsize=(15, 6) )
    plt.plot(y, label='real views')
    plt.plot(y_pred, label='lstm pred views')
    plt.legend()
    plt.show()
```

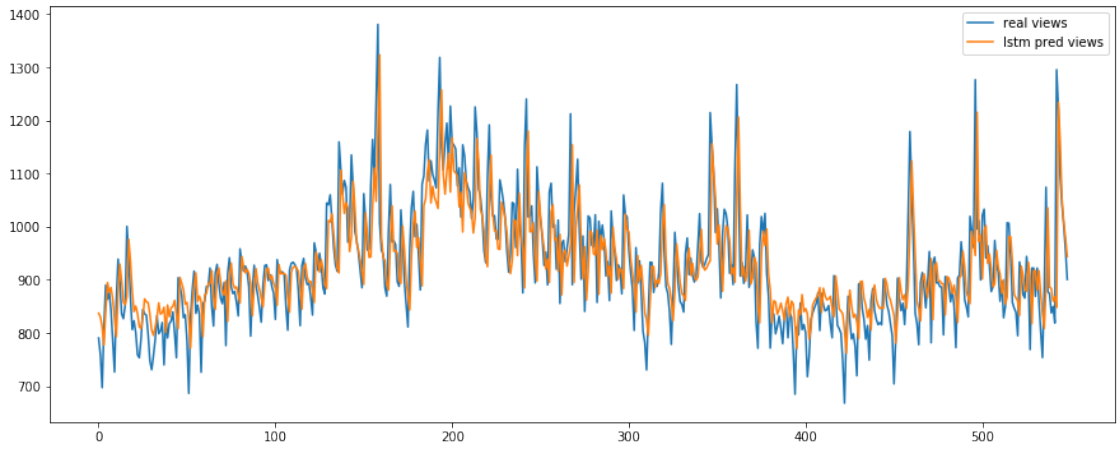
```
lstm...
predicting...
```

```
'Language "co"'
```

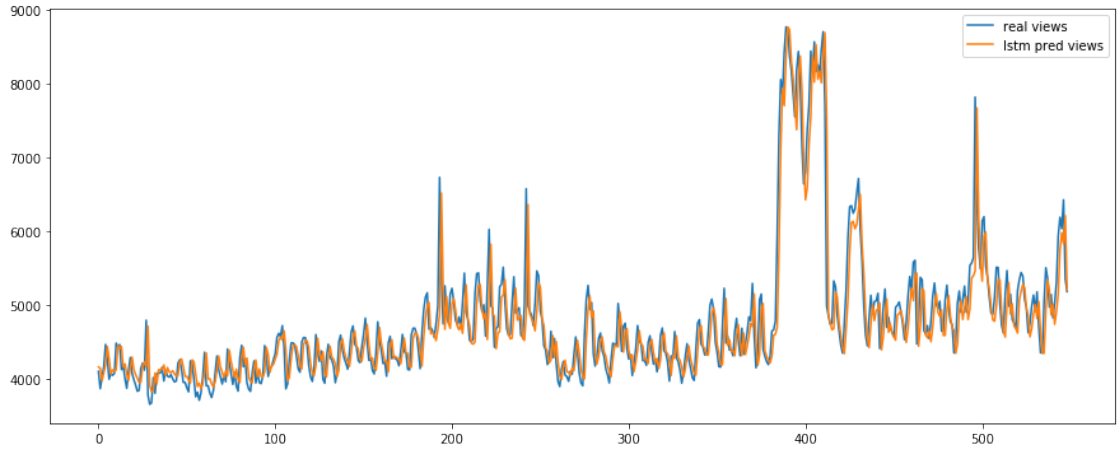



```
lstm...
predicting...
```

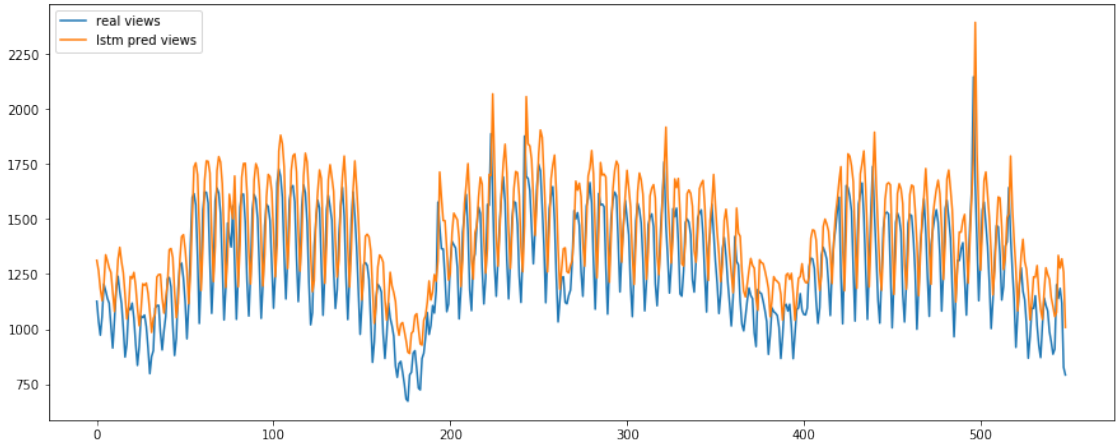
```
'Language "de"'
```



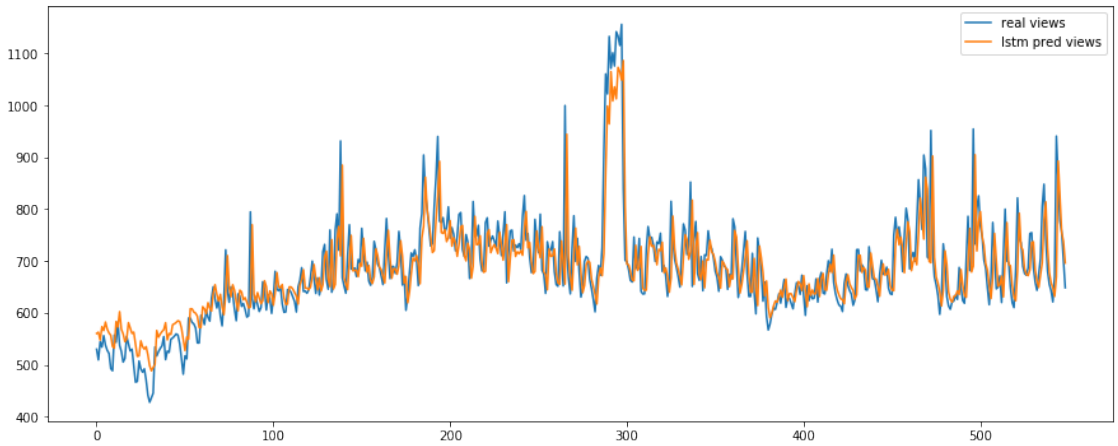

```
lstm...
predicting...

'Language "en"'
```



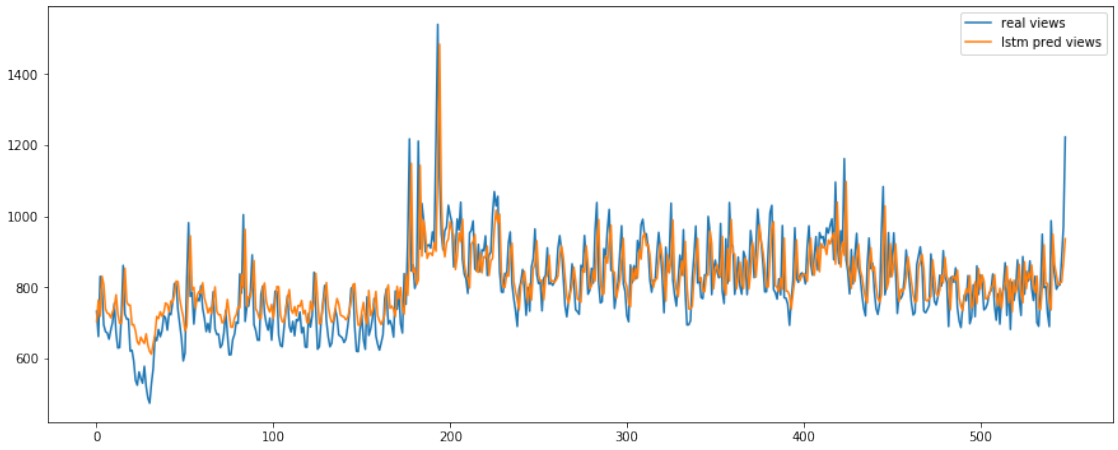

```
lstm...
predicting...

'Language "es"'
```



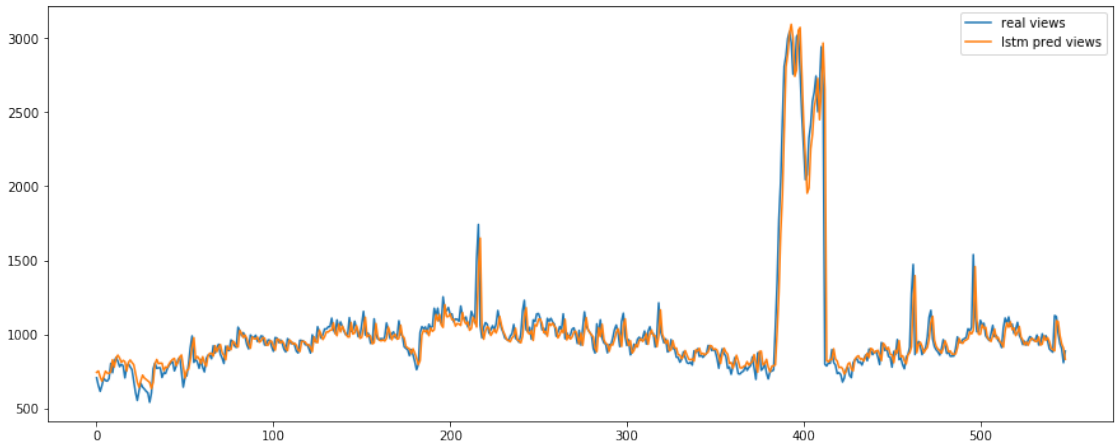

```
lstm...
predicting...

'Language "fr"'
```



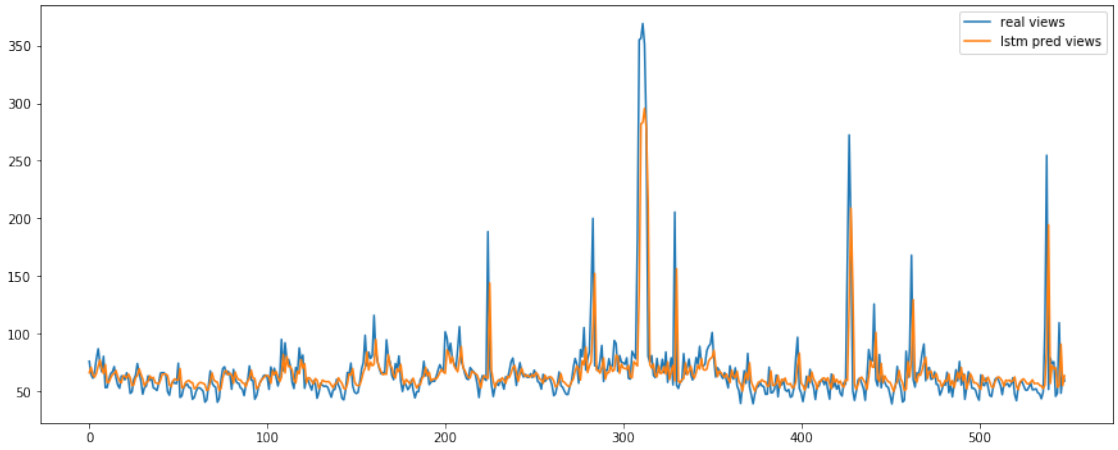

```
lstm...
predicting...

'Language "ja"'
```




```
lstm...
predicting...

'Language "ru"'
```




```
lstm...
predicting...

'Language "ww"'
```




```
lstm...
predicting...

'Language "zh"'
```