davis_xgb

May 16, 2024

[ ]:
```
!pip install xgboost
```

[13]:
```
import seaborn as sns
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import warnings
from sklearn.model_selection import train_test_split
import xgboost as xgb
```

1 Attempt #1 - no additional feature engineering

[15]:
```
df = pd.read_csv('Davis.csv', parse_dates=['date'])
df.set_index('date', inplace=True)
#drop the columns that are not needed, unnamed:0 and hospital
df = df.drop(['Unnamed: 0', 'hospital'], axis=1)
```

[16]:
```
df.dtypes
```

[16]:
```
year              int64
monthday          int64
month             int64
day               int64
attendences     float64
min             float64
max             float64
aver            float64
Hosp_ID           int64
Time_ID           int64
DAT             float64
ThreeDAT        float64
EHIaccl         float64
dow               int64
Sun               int64
Mon               int64
Tue               int64
Wed               int64
```

```
Thu                 int64
Fri                 int64
Sat                 int64
Jan                 int64
Feb                 int64
Mar                 int64
Apr                 int64
May                 int64
Jun                 int64
Jul                 int64
Aug                 int64
Sep                 int64
Oct                 int64
Nov                 int64
Dec                 int64
Year_1              int64
Year_2              int64
Year_3              int64
Year_4              int64
Year_5              int64
Year_6              int64
Year_7              int64
Year_8              int64
dtype: object
```

[17]:
```
#separate out the features and target variable
X, y = df.drop('attendences', axis=1), df[['attendences']]
```

[18]:
```
#Split data into train/test split
X_train, X_test, y_train, y_test = X[:'2014'], X['2015':], y[:'2014'], y['2015':
 ↪]
```

[19]:
```
# Create regression matrices
dtrain_reg = xgb.DMatrix(X_train, y_train, enable_categorical=True)
dtest_reg = xgb.DMatrix(X_test, y_test, enable_categorical=True)
```

[23]:
```
# Define hyperparameters
params = {"objective": "reg:squarederror", "tree_method": "hist"}

n = 100
model = xgb.train(
   params=params,
   dtrain=dtrain_reg,
   num_boost_round=n,
)
```

[24]:
```python
from sklearn.metrics import mean_squared_error

preds = model.predict(dtest_reg)
```

[25]:
```python
rmse = mean_squared_error(y_test, preds, squared=False)

print(f"RMSE of the base model: {rmse:.3f}")
```

```
RMSE of the base model: 21.290
```

```
c:\Users\kentm\Documents\Jupyter Notebooks\ed visit timeseries\.conda\Lib\site-
packages\sklearn\metrics\_regression.py:483: FutureWarning: 'squared' is
deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean
squared error, use the function'root_mean_squared_error'.
  warnings.warn(
```

[26]:
```python
evals = [(dtrain_reg, "train"), (dtest_reg, "validation")]
```

[27]:
```python
evals = [(dtrain_reg, "train"), (dtest_reg, "validation")]

model = xgb.train(
   params=params,
   dtrain=dtrain_reg,
   num_boost_round=n,
   evals=evals,
)
```

```
[0]     train-rmse:20.57698     validation-rmse:35.38845
[1]     train-rmse:17.89705     validation-rmse:31.04664
[2]     train-rmse:16.11103     validation-rmse:28.23228
[3]     train-rmse:14.94763     validation-rmse:26.22315
[4]     train-rmse:14.07073     validation-rmse:23.80828
[5]     train-rmse:13.50216     validation-rmse:23.01773
[6]     train-rmse:13.04559     validation-rmse:22.37125
[7]     train-rmse:12.75375     validation-rmse:22.09961
[8]     train-rmse:12.36247     validation-rmse:21.64580
[9]     train-rmse:12.17684     validation-rmse:21.39410
[10]    train-rmse:11.91217     validation-rmse:21.36551
[11]    train-rmse:11.80576     validation-rmse:21.07718
[12]    train-rmse:11.53037     validation-rmse:20.94231
[13]    train-rmse:11.32419     validation-rmse:20.94428
[14]    train-rmse:11.16781     validation-rmse:20.82353
[15]    train-rmse:10.90405     validation-rmse:21.12225
[16]    train-rmse:10.69685     validation-rmse:21.15110
[17]    train-rmse:10.38318     validation-rmse:21.24322
[18]    train-rmse:10.26230     validation-rmse:20.97955
[19]    train-rmse:10.00594     validation-rmse:20.88005
[20]    train-rmse:9.94340      validation-rmse:20.89322
[21]    train-rmse:9.90405      validation-rmse:20.91943
```

```
[22]    train-rmse:9.85032     validation-rmse:20.59603
[23]    train-rmse:9.81836     validation-rmse:20.60465
[24]    train-rmse:9.68540     validation-rmse:20.49280
[25]    train-rmse:9.42245     validation-rmse:20.47224
[26]    train-rmse:9.40610     validation-rmse:20.46674
[27]    train-rmse:9.31659     validation-rmse:20.47315
[28]    train-rmse:9.17599     validation-rmse:20.45866
[29]    train-rmse:9.02224     validation-rmse:20.49002
[30]    train-rmse:8.96467     validation-rmse:20.79847
[31]    train-rmse:8.91975     validation-rmse:20.81242
[32]    train-rmse:8.70874     validation-rmse:20.81000
[33]    train-rmse:8.59723     validation-rmse:20.79737
[34]    train-rmse:8.43756     validation-rmse:20.83933
[35]    train-rmse:8.36136     validation-rmse:20.85988
[36]    train-rmse:8.34698     validation-rmse:20.87487
[37]    train-rmse:8.32844     validation-rmse:20.85109
[38]    train-rmse:8.15004     validation-rmse:20.85886
[39]    train-rmse:8.04151     validation-rmse:20.85571
[40]    train-rmse:7.96766     validation-rmse:20.88116
[41]    train-rmse:7.83688     validation-rmse:20.90447
[42]    train-rmse:7.74672     validation-rmse:20.91494
[43]    train-rmse:7.67471     validation-rmse:20.80084
[44]    train-rmse:7.52446     validation-rmse:20.82732
[45]    train-rmse:7.46895     validation-rmse:20.81068
[46]    train-rmse:7.40479     validation-rmse:20.81407
[47]    train-rmse:7.27238     validation-rmse:20.81378
[48]    train-rmse:7.18302     validation-rmse:20.83357
[49]    train-rmse:7.08652     validation-rmse:20.98890
[50]    train-rmse:7.00587     validation-rmse:21.01872
[51]    train-rmse:6.92945     validation-rmse:21.02215
[52]    train-rmse:6.87135     validation-rmse:21.03175
[53]    train-rmse:6.73948     validation-rmse:21.04085
[54]    train-rmse:6.59420     validation-rmse:21.07513
[55]    train-rmse:6.51885     validation-rmse:21.06905
[56]    train-rmse:6.47816     validation-rmse:21.07362
[57]    train-rmse:6.36690     validation-rmse:21.17923
[58]    train-rmse:6.36098     validation-rmse:21.18134
[59]    train-rmse:6.22542     validation-rmse:21.18277
[60]    train-rmse:6.19454     validation-rmse:21.20757
[61]    train-rmse:6.09971     validation-rmse:21.16663
[62]    train-rmse:6.04597     validation-rmse:21.17911
[63]    train-rmse:5.92140     validation-rmse:21.18289
[64]    train-rmse:5.91505     validation-rmse:21.18653
[65]    train-rmse:5.85327     validation-rmse:21.18679
[66]    train-rmse:5.78370     validation-rmse:21.18500
[67]    train-rmse:5.70560     validation-rmse:21.18758
[68]    train-rmse:5.63868     validation-rmse:21.18658
[69]    train-rmse:5.54738     validation-rmse:21.35656
```

```
[70]	train-rmse:5.43029	validation-rmse:21.39749
[71]	train-rmse:5.35898	validation-rmse:21.39642
[72]	train-rmse:5.23461	validation-rmse:21.40526
[73]	train-rmse:5.19625	validation-rmse:21.40676
[74]	train-rmse:5.15821	validation-rmse:21.39932
[75]	train-rmse:5.14638	validation-rmse:21.33115
[76]	train-rmse:5.13535	validation-rmse:21.33361
[77]	train-rmse:5.10735	validation-rmse:21.34454
[78]	train-rmse:5.08941	validation-rmse:21.35428
[79]	train-rmse:5.03167	validation-rmse:21.30998
[80]	train-rmse:4.94863	validation-rmse:21.30703
[81]	train-rmse:4.90116	validation-rmse:21.30902
[82]	train-rmse:4.82021	validation-rmse:21.33733
[83]	train-rmse:4.78846	validation-rmse:21.32348
[84]	train-rmse:4.72223	validation-rmse:21.31695
[85]	train-rmse:4.63709	validation-rmse:21.31840
[86]	train-rmse:4.54898	validation-rmse:21.30781
[87]	train-rmse:4.51128	validation-rmse:21.40188
[88]	train-rmse:4.44278	validation-rmse:21.39443
[89]	train-rmse:4.41445	validation-rmse:21.40290
[90]	train-rmse:4.38960	validation-rmse:21.38219
[91]	train-rmse:4.33664	validation-rmse:21.37649
[92]	train-rmse:4.29016	validation-rmse:21.29750
[93]	train-rmse:4.25880	validation-rmse:21.29425
[94]	train-rmse:4.16962	validation-rmse:21.31368
[95]	train-rmse:4.14030	validation-rmse:21.31482
[96]	train-rmse:4.06991	validation-rmse:21.31194
[97]	train-rmse:4.00219	validation-rmse:21.32143
[98]	train-rmse:3.94368	validation-rmse:21.34343
[99]	train-rmse:3.86686	validation-rmse:21.28979
```

[35]:

```
evals = [(dtrain_reg, "train"), (dtest_reg, "validation")]
n = 10000


model = xgb.train(
   params=params,
   dtrain=dtrain_reg,
   num_boost_round=n,
   evals=evals,
   verbose_eval=50,
   # Activate early stopping
   early_stopping_rounds=100
)
```

```
[0]	train-rmse:20.57698	validation-rmse:35.38845
[50]	train-rmse:7.00587	validation-rmse:21.01872
[100]	train-rmse:3.82457	validation-rmse:21.28313
```

```
[127]   train-rmse:2.90429      validation-rmse:21.40159
```

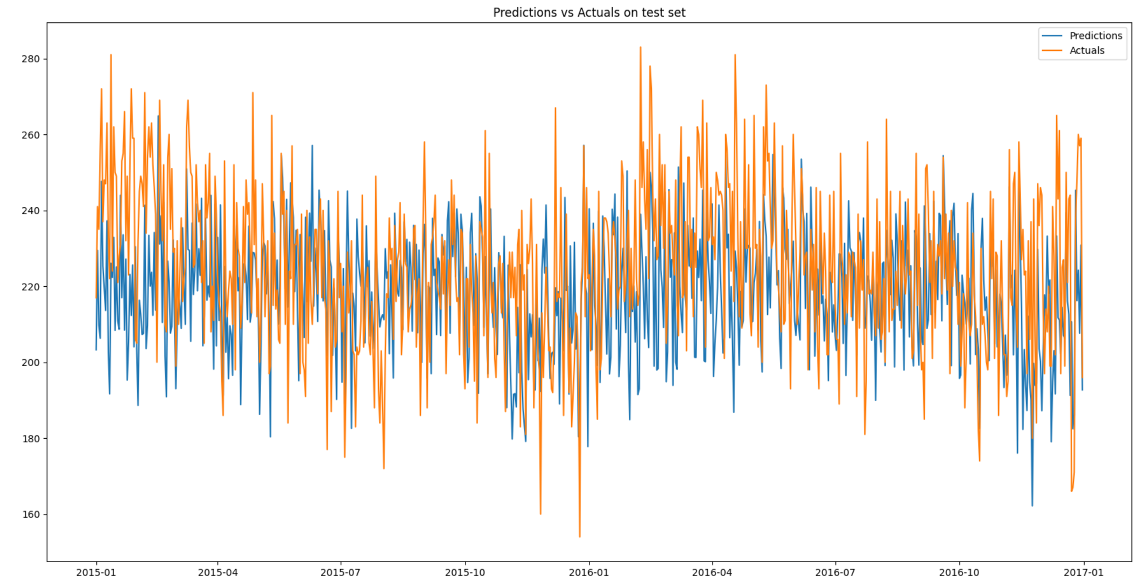
[31]:
```
params = {"objective": "reg:squarederror", "tree_method": "hist"}
n = 1000

results = xgb.cv(
   params, dtrain_reg,
   num_boost_round=n,
   nfold=5,
   early_stopping_rounds=100
)
```

[34]:
```
best_rmse = results['test-rmse-mean'].min()
best_rmse
```

[34]:
```
15.688509272020877
```

[39]:
```
#apply model to the test data
dtest_reg = xgb.DMatrix(X_test)
preds_test = model.predict(dtest_reg)

#apply model to the train data
dtrain_reg = xgb.DMatrix(X_train)
preds_train = model.predict(dtrain_reg)
```

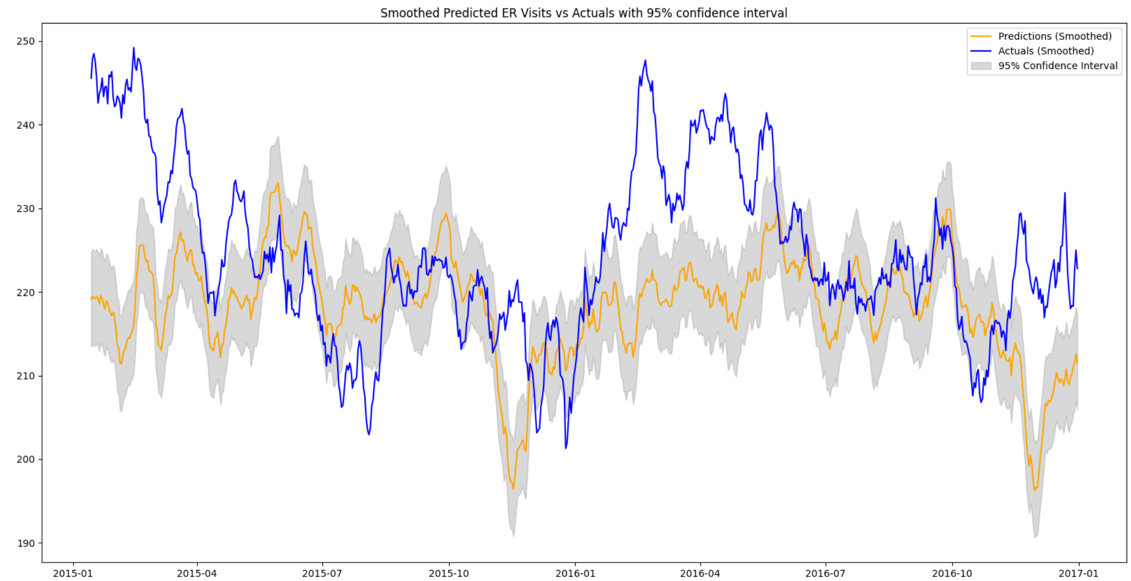
[43]:
```
plt.figure(figsize=(20,10))
#graph preds and actuals
plt.plot(y_test.index, preds_test, label='Predictions')
plt.plot(y_test.index, y_test, label='Actuals')
plt.legend()
plt.title('Predictions vs Actuals on test set')
plt.show()
```



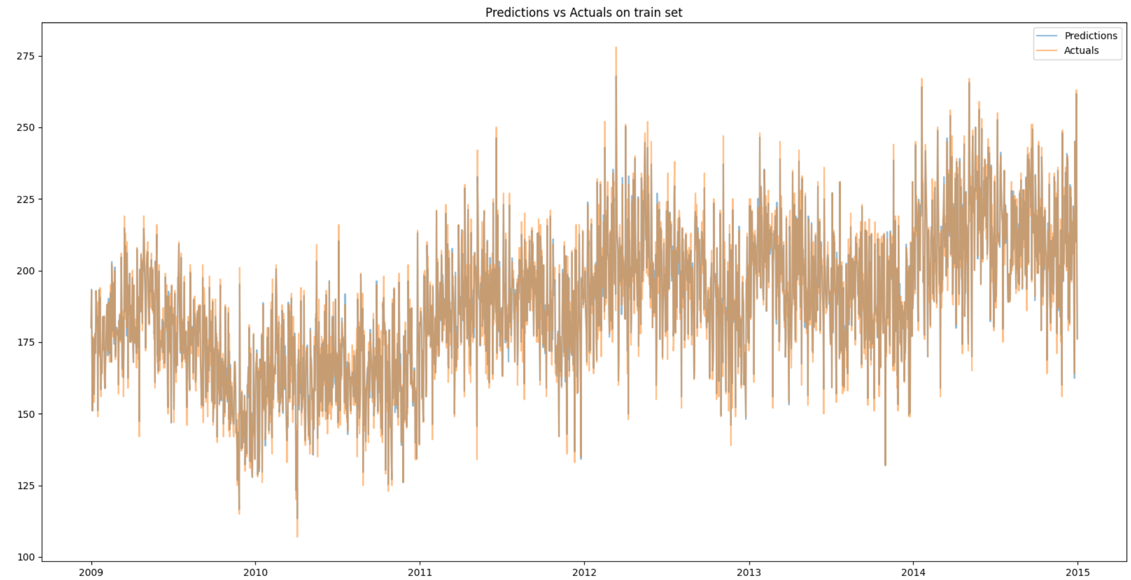
This graph is hard to look at… let's smooth it out a bit and add confidence intervals

[54]:
```
# Calculate rolling averages
window_size = 14  # 7-day rolling window
y_test_smoothed = y_test.rolling(window=window_size).mean()
preds_test_smoothed = pd.Series(preds_test, index=y_test.index).
  ↪rolling(window=window_size).mean()

# Calculate residuals on training data
residuals = y_train['attendences'] - model.predict(xgb.DMatrix(X_train))

# Calculate the standard deviation of these residualsa
error_std = np.std(residuals)

# Generate upper and lower confidence bounds
confidence_interval = 1.96 * error_std  # 95% confidence interval
upper_bound = preds_test + confidence_interval
lower_bound = preds_test - confidence_interval
#smooth upper and lower bounds
upper_bound_smoothed = pd.Series(upper_bound, index=y_test.index).
  ↪rolling(window=window_size).mean()
lower_bound_smoothed = pd.Series(lower_bound, index=y_test.index).
  ↪rolling(window=window_size).mean()

plt.figure(figsize=(20,10))
plt.plot(y_test_smoothed.index, preds_test_smoothed, label='Predictions␣
  ↪(Smoothed)', color='orange')
```

```
plt.plot(y_test_smoothed.index, y_test_smoothed, label='Actuals (Smoothed)',␣
↪color='blue')
plt.fill_between(y_test.index, lower_bound_smoothed, upper_bound_smoothed,␣
↪color='gray', alpha=0.3, label='95% Confidence Interval')
plt.legend()
plt.title('Smoothed Predicted ER Visits vs Actuals with 95% confidence␣
↪interval')
plt.show()
```



[41]:
```
#graph preds and actuals on train data
plt.figure(figsize=(20,10))
plt.plot(y_train.index, preds_train, label='Predictions', alpha=0.5)
plt.plot(y_train.index, y_train, label='Actuals', alpha=0.5)
plt.legend()
plt.title('Predictions vs Actuals on train set')
plt.show()
```



As expected, the predicted values for the training data line up amazingly well.

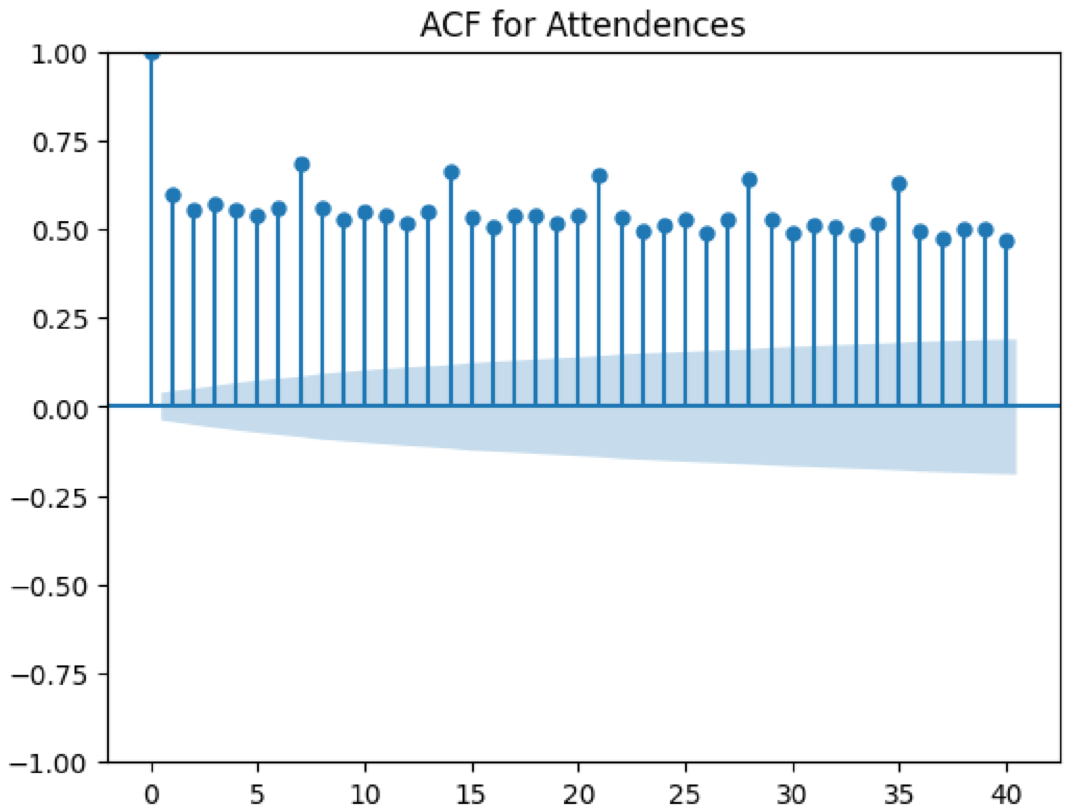
2 Attempt #2: adding lags and rolling averages

[79]:
```
df = pd.read_csv('Davis.csv', parse_dates=['date'])
df.set_index('date', inplace=True)
#drop the columns that are not needed, unnamed:0 and hospital
df = df.drop(['Unnamed: 0', 'hospital'], axis=1)
```

Autocorrelation graphing to determine which lags are most relevant:

[77]:
```
from statsmodels.graphics.tsaplots import plot_acf

plt.figure(figsize=(12, 6))
plot_acf(df['attendences'].dropna(), lags=40)  # Adjust lags as necessary
plt.title('ACF for Attendences')
plt.show()
```

```
<Figure size 1200x600 with 0 Axes>
```



It looks like there is pretty heavy autocorrelation accross all 40, so I'll pick a few short term ones and weekly values to try to capture these trends without overcomplicating the model or leading to overfitting:

[102]:
```
df = pd.read_csv('Davis.csv', parse_dates=['date'])
df.set_index('date', inplace=True)
#drop the columns that are not needed, unnamed:0 and hospital
df = df.drop(['Unnamed: 0', 'hospital'], axis=1)

#Creating lag features
df['lag_1'] = df['attendences'].shift(1)
df['lag_2'] = df['attendences'].shift(2)
df['lag_3'] = df['attendences'].shift(3)
df['lag_7'] = df['attendences'].shift(7)
df['lag_14'] = df['attendences'].shift(14)
df['lag_28'] = df['attendences'].shift(28)
df['lag_365'] = df['attendences'].shift(365)

#Adding somewhat arbitrary rolling averages
df['roll_avg_3'] = df['attendences'].rolling(window=3).mean()
#df['roll_avg_7'] = df['attendences'].rolling(window=7).mean()
```

```
#df['roll_avg_14'] = df['attendences'].rolling(window=14).mean()
#df['roll_avg_30'] = df['attendences'].rolling(window=30).mean()

#drop rows with any missing values in the lag or rolling avg columns
df = df.dropna()
```

[105]:
```
df.head(5)
```

[105]:
```
            year  monthday  month  day  attendences  min   max  aver  Hosp_ID  \
date
2010-01-01  2010       101      1    1        156.0  7.0  13.0  11.0        6
2010-01-02  2010       102      1    2        168.0  7.0  14.0  11.0        6
2010-01-03  2010       103      1    3        168.0  3.0  10.0   7.0        6
2010-01-04  2010       104      1    4        171.0  3.0  12.0   7.0        6
2010-01-05  2010       105      1    5        165.0  4.0   8.0   6.0        6

            Time_ID  …  Year_7  Year_8  lag_1  lag_2  lag_3  lag_7  lag_14  \
date                 …
2010-01-01      366  …       0       0  134.0  157.0  142.0  128.0   181.0
2010-01-02      367  …       0       0  156.0  134.0  157.0  144.0   145.0
2010-01-03      368  …       0       0  168.0  156.0  134.0  146.0   131.0
2010-01-04      369  …       0       0  168.0  168.0  156.0  178.0   173.0
2010-01-05      370  …       0       0  171.0  168.0  168.0  142.0   133.0

            lag_28  lag_365  roll_avg_3
date
2010-01-01   150.0    180.0  149.000000
2010-01-02   151.0    193.0  152.666667
2010-01-03   142.0    171.0  164.000000
2010-01-04   143.0    151.0  169.000000
2010-01-05   156.0    177.0  168.000000

[5 rows x 49 columns]
```

[103]:
```
#separate out the features and target variable
X, y = df.drop('attendences', axis=1), df[['attendences']]
#Split data into train/test split
X_train, X_test, y_train, y_test = X[:'2014'], X['2015':], y[:'2014'], y['2015':
 ↪]
# Create regression matrices
dtrain_reg = xgb.DMatrix(X_train, y_train, enable_categorical=True)
dtest_reg = xgb.DMatrix(X_test, y_test, enable_categorical=True)
#train model
evals = [(dtrain_reg, "train"), (dtest_reg, "validation")]
n = 10000
model = xgb.train(
   params=params,
```

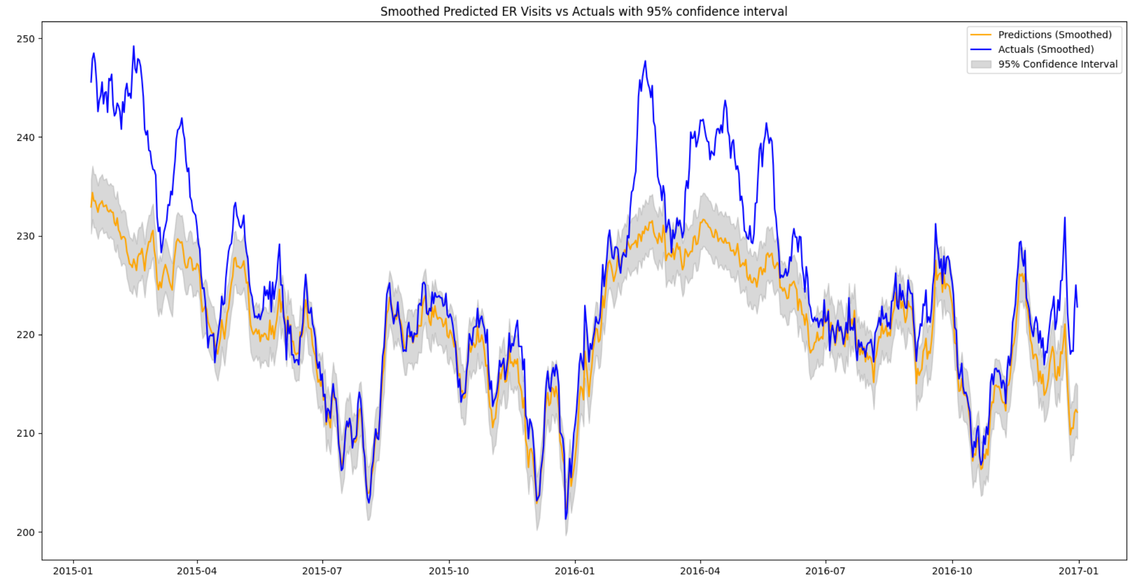
```
    dtrain=dtrain_reg,
    num_boost_round=n,
    evals=evals,
    verbose_eval=50,
    # Activate early stopping
    early_stopping_rounds=5
)
```

```
[0]     train-rmse:19.17805     validation-rmse:31.01858
[34]    train-rmse:1.37975      validation-rmse:11.39988
```

[104]:

```
#apply model to the test data
dtest_reg = xgb.DMatrix(X_test)
preds_test = model.predict(dtest_reg)

# Calculate rolling averages
window_size = 14  # 7-day rolling window
y_test_smoothed = y_test.rolling(window=window_size).mean()
preds_test_smoothed = pd.Series(preds_test, index=y_test.index).
 ↪rolling(window=window_size).mean()

# Calculate residuals on training data
residuals = y_train['attendences'] - model.predict(xgb.DMatrix(X_train))

# Calculate the standard deviation of these residualsa
error_std = np.std(residuals)

# Generate upper and lower confidence bounds
confidence_interval = 1.96 * error_std  # 95% confidence interval
upper_bound = preds_test + confidence_interval
lower_bound = preds_test - confidence_interval
#smooth upper and lower bounds
upper_bound_smoothed = pd.Series(upper_bound, index=y_test.index).
 ↪rolling(window=window_size).mean()
lower_bound_smoothed = pd.Series(lower_bound, index=y_test.index).
 ↪rolling(window=window_size).mean()

plt.figure(figsize=(20,10))
plt.plot(y_test_smoothed.index, preds_test_smoothed, label='Predictions␣
 ↪(Smoothed)', color='orange')
plt.plot(y_test_smoothed.index, y_test_smoothed, label='Actuals (Smoothed)',␣
 ↪color='blue')
plt.fill_between(y_test.index, lower_bound_smoothed, upper_bound_smoothed,␣
 ↪color='gray', alpha=0.3, label='95% Confidence Interval')
plt.legend()
plt.title('Smoothed Predicted ER Visits vs Actuals with 95% confidence␣
 ↪interval')
```

```
plt.show()
```



3 Attempt #3: Using above augmented data with grid search hyperparameter optimization

[106]:
```
df = pd.read_csv('Davis.csv', parse_dates=['date'])
df.set_index('date', inplace=True)
#drop the columns that are not needed, unnamed:0 and hospital
df = df.drop(['Unnamed: 0', 'hospital'], axis=1)

#Creating lag features
df['lag_1'] = df['attendences'].shift(1)
df['lag_2'] = df['attendences'].shift(2)
df['lag_3'] = df['attendences'].shift(3)
df['lag_7'] = df['attendences'].shift(7)
df['lag_14'] = df['attendences'].shift(14)
df['lag_28'] = df['attendences'].shift(28)
df['lag_365'] = df['attendences'].shift(365)

#Adding somewhat arbitrary rolling averages
df['roll_avg_3'] = df['attendences'].rolling(window=3).mean()
#df['roll_avg_7'] = df['attendences'].rolling(window=7).mean()
#df['roll_avg_14'] = df['attendences'].rolling(window=14).mean()
#df['roll_avg_30'] = df['attendences'].rolling(window=30).mean()

#drop rows with any missing values in the lag or rolling avg columns
```

```
df = df.dropna()
```

[107]:
```
from sklearn.model_selection import GridSearchCV
from xgboost import XGBRegressor

#separate out the features and target variable
X, y = df.drop('attendences', axis=1), df[['attendences']]
#Split data into train/test split
X_train, X_test, y_train, y_test = X[:'2014'], X['2015':], y[:'2014'], y['2015':
↪]
# Create regression matrices
dtrain_reg = xgb.DMatrix(X_train, y_train, enable_categorical=True)
dtest_reg = xgb.DMatrix(X_test, y_test, enable_categorical=True)

# Define the model
model = XGBRegressor()

# Define the parameter grid
param_grid = {
    'max_depth': [3, 4, 5],
    'learning_rate': [0.01, 0.1, 0.2],
    'n_estimators': [100, 200],
    'subsample': [0.7, 0.8, 0.9]
}

# Setup the grid search
grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=3,␣
↪scoring='neg_mean_squared_error', verbose=1)

# Fit the grid search to the data
grid_search.fit(X_train, y_train)

# Get the best parameters
print("Best parameters:", grid_search.best_params_)
```

```
Fitting 3 folds for each of 54 candidates, totalling 162 fits
Best parameters: {'learning_rate': 0.1, 'max_depth': 4, 'n_estimators': 200,
'subsample': 0.8}
```

[114]:
```
#train model
evals = [(dtrain_reg, "train"), (dtest_reg, "validation")]
n = 10000
model = xgb.train(
   params=grid_search.best_params_,
   dtrain=dtrain_reg,
   num_boost_round=n,
   evals=evals,
```

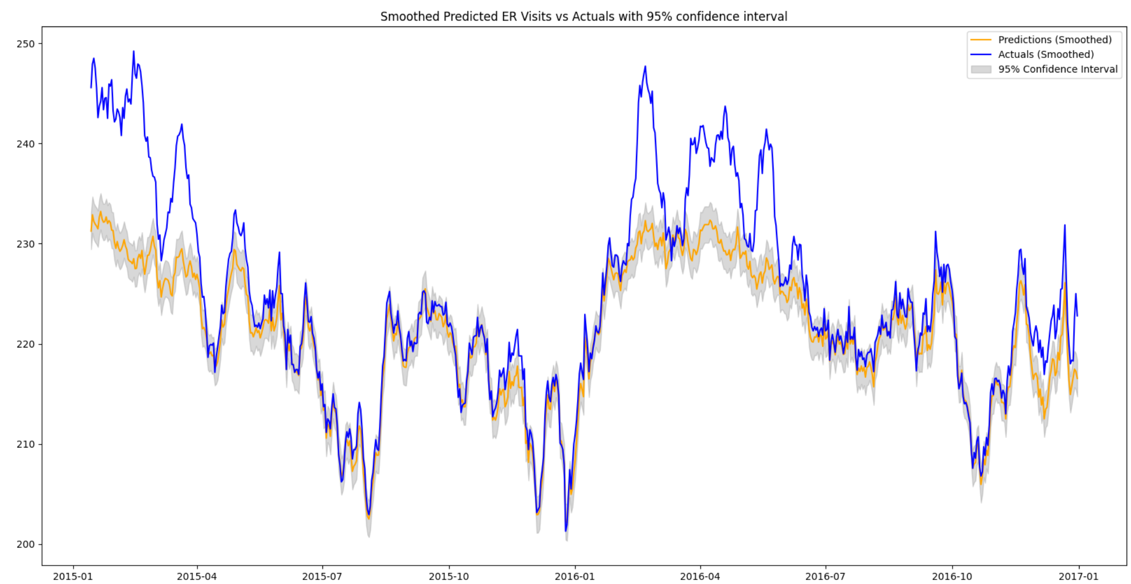
```
    verbose_eval=50,
    # Activate early stopping
    early_stopping_rounds=25
)
```

```
[0]     train-rmse:23.26391     validation-rmse:36.72013
[50]    train-rmse:6.04741      validation-rmse:12.85865
[100]   train-rmse:3.43751      validation-rmse:11.24115

c:\Users\kentm\Documents\Jupyter Notebooks\ed visit timeseries\.conda\Lib\site-
packages\xgboost\core.py:160: UserWarning: [11:11:55] WARNING: C:\buildkite-
agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0b3782d1791676daf-1\xgboost\xgboost-ci-windows\src\learner.cc:742:
Parameters: { "n_estimators" } are not used.

  warnings.warn(smsg, UserWarning)

[150]   train-rmse:2.20958      validation-rmse:10.81809
[200]   train-rmse:1.60785      validation-rmse:10.53766
[250]   train-rmse:1.28960      validation-rmse:10.37736
[300]   train-rmse:1.07537      validation-rmse:10.32707
[349]   train-rmse:0.92537      validation-rmse:10.30443
```

[115]:

```
#apply model to the test data
dtest_reg = xgb.DMatrix(X_test)
preds_test = model.predict(dtest_reg)

# Calculate rolling averages
window_size = 14  # 7-day rolling window
y_test_smoothed = y_test.rolling(window=window_size).mean()
preds_test_smoothed = pd.Series(preds_test, index=y_test.index).
 ↪rolling(window=window_size).mean()

# Calculate residuals on training data
residuals = y_train['attendences'] - model.predict(xgb.DMatrix(X_train))

# Calculate the standard deviation of these residualsa
error_std = np.std(residuals)

# Generate upper and lower confidence bounds
confidence_interval = 1.96 * error_std  # 95% confidence interval
upper_bound = preds_test + confidence_interval
lower_bound = preds_test - confidence_interval
#smooth upper and lower bounds
upper_bound_smoothed = pd.Series(upper_bound, index=y_test.index).
 ↪rolling(window=window_size).mean()
lower_bound_smoothed = pd.Series(lower_bound, index=y_test.index).
 ↪rolling(window=window_size).mean()
```

```
plt.figure(figsize=(20,10))
plt.plot(y_test_smoothed.index, preds_test_smoothed, label='Predictions␣
 ↪(Smoothed)', color='orange')
plt.plot(y_test_smoothed.index, y_test_smoothed, label='Actuals (Smoothed)',␣
 ↪color='blue')
plt.fill_between(y_test.index, lower_bound_smoothed, upper_bound_smoothed,␣
 ↪color='gray', alpha=0.3, label='95% Confidence Interval')
plt.legend()
plt.title('Smoothed Predicted ER Visits vs Actuals with 95% confidence␣
 ↪interval')
plt.show()
```



4 Attempt #4: Cross validating performance accross multiple time series splits

[129]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import TimeSeriesSplit
import xgboost as xgb
from sklearn.metrics import mean_squared_error
from xgboost import XGBRegressor
from sklearn.model_selection import GridSearchCV

# Load and prepare the data
df = pd.read_csv('Davis.csv', parse_dates=['date'])
```

```
df.set_index('date', inplace=True)
df = df.drop(['Unnamed: 0', 'hospital'], axis=1)

# Add lag and rolling features to the whole dataset
lags = [1, 2, 3, 7, 14, 28, 365]
for lag in lags:
    df[f'lag_{lag}'] = df['attendences'].shift(lag)

rolling_windows = [3]
for window in rolling_windows:
    df[f'rolling_avg_{window}'] = df['attendences'].rolling(window=window).
 ↪mean()

# Drop rows with NaN values that were created by shift and rolling
df.dropna(inplace=True)

# Time Series Cross-validation
tscv = TimeSeriesSplit(n_splits=5)

for train_index, test_index in tscv.split(df):
    train, test = df.iloc[train_index], df.iloc[test_index]
    X_train, y_train = train.drop('attendences', axis=1), train['attendences']
    X_test, y_test = test.drop('attendences', axis=1), test['attendences']

    # Define the model
    model = XGBRegressor()

    # Parameter grid
    param_grid = {
        'max_depth': [3, 4, 5],
        'learning_rate': [0.01, 0.1, 0.2],
        'n_estimators': [100, 200],
        'subsample': [0.7, 0.8, 0.9]
    }

    # Grid search
    grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=3,␣
 ↪scoring='neg_mean_squared_error', verbose=1)
    grid_search.fit(X_train, y_train)
    print("Best parameters:", grid_search.best_params_)

    # Use best parameters to train the model
    best_params = grid_search.best_params_
    final_model = XGBRegressor(**best_params)
    final_model.fit(X_train, y_train)

    # Predict and evaluate
```

```
    predictions = final_model.predict(X_test)
    rmse = np.sqrt(mean_squared_error(y_test, predictions))
    print(f"Fold RMSE: {rmse:.3f}")
```

```
Fitting 3 folds for each of 54 candidates, totalling 162 fits
Best parameters: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 200,
'subsample': 0.7}
Fold RMSE: 11.868
Fitting 3 folds for each of 54 candidates, totalling 162 fits
Best parameters: {'learning_rate': 0.2, 'max_depth': 3, 'n_estimators': 200,
'subsample': 0.7}
Fold RMSE: 6.593
Fitting 3 folds for each of 54 candidates, totalling 162 fits
Best parameters: {'learning_rate': 0.2, 'max_depth': 3, 'n_estimators': 200,
'subsample': 0.9}
Fold RMSE: 6.141
Fitting 3 folds for each of 54 candidates, totalling 162 fits
Best parameters: {'learning_rate': 0.2, 'max_depth': 3, 'n_estimators': 200,
'subsample': 0.7}
Fold RMSE: 10.375
Fitting 3 folds for each of 54 candidates, totalling 162 fits
Best parameters: {'learning_rate': 0.2, 'max_depth': 3, 'n_estimators': 200,
'subsample': 0.8}
Fold RMSE: 5.128
```

4.1 Training final model

[132]:
```
# Assuming best_params are defined from your previous GridSearch
df = pd.read_csv('Davis.csv', parse_dates=['date'])
df.set_index('date', inplace=True)
#drop the columns that are not needed, unnamed:0 and hospital
df = df.drop(['Unnamed: 0', 'hospital'], axis=1)

#Creating lag features
df['lag_1'] = df['attendences'].shift(1)
df['lag_2'] = df['attendences'].shift(2)
df['lag_3'] = df['attendences'].shift(3)
df['lag_7'] = df['attendences'].shift(7)
df['lag_14'] = df['attendences'].shift(14)
df['lag_28'] = df['attendences'].shift(28)
df['lag_365'] = df['attendences'].shift(365)

#Adding somewhat arbitrary rolling averages
df['roll_avg_3'] = df['attendences'].rolling(window=3).mean()
#df['roll_avg_7'] = df['attendences'].rolling(window=7).mean()
#df['roll_avg_14'] = df['attendences'].rolling(window=14).mean()
#df['roll_avg_30'] = df['attendences'].rolling(window=30).mean()
```

```python
#drop rows with any missing values in the lag or rolling avg columns
df = df.dropna()

#separate out the features and target variable
X, y = df.drop('attendences', axis=1), df[['attendences']]
#Split data into train/test split
X_train, X_test, y_train, y_test = X[:'2014'], X['2015':], y[:'2014'], y['2015':
↪]
# Create regression matrices
dtrain_reg = xgb.DMatrix(X_train, y_train, enable_categorical=True)
dtest_reg = xgb.DMatrix(X_test, y_test, enable_categorical=True)

best_params = {'learning_rate': 0.2, 'max_depth': 3, 'n_estimators': 200,␣
↪'subsample': 0.7}
final_model = XGBRegressor(**best_params)
final_model.fit(X_train, y_train)

predictions = final_model.predict(X_test)
rmse = np.sqrt(mean_squared_error(y_test, predictions))
print(f"Fold RMSE: {rmse:.3f}")
```

```
Fold RMSE: 10.965
```

[137]:

```python
#apply model to the test data
dtest_reg = xgb.DMatrix(X_test)
preds_test = final_model.predict(X_test)

# Calculate rolling averages
window_size = 14  # 7-day rolling window
y_test_smoothed = y_test.rolling(window=window_size).mean()
preds_test_smoothed = pd.Series(preds_test, index=y_test.index).
↪rolling(window=window_size).mean()

# Calculate residuals on training data
residuals = y_train['attendences'] - final_model.predict(X_train)

# Calculate the standard deviation of these residualsa
error_std = np.std(residuals)

# Generate upper and lower confidence bounds
confidence_interval = 1.96 * error_std  # 95% confidence interval
upper_bound = preds_test + confidence_interval
lower_bound = preds_test - confidence_interval
#smooth upper and lower bounds
upper_bound_smoothed = pd.Series(upper_bound, index=y_test.index).
↪rolling(window=window_size).mean()
```

```
lower_bound_smoothed = pd.Series(lower_bound, index=y_test.index).
 ↪rolling(window=window_size).mean()

plt.figure(figsize=(20,10))
plt.plot(y_test_smoothed.index, preds_test_smoothed, label='Predictions␣
 ↪(Smoothed)', color='orange')
plt.plot(y_test_smoothed.index, y_test_smoothed, label='Actuals (Smoothed)',␣
 ↪color='blue')
plt.fill_between(y_test.index, lower_bound_smoothed, upper_bound_smoothed,␣
 ↪color='gray', alpha=0.3, label='95% Confidence Interval')
plt.legend()
plt.title('Smoothed Predicted ER Visits vs Actuals with 95% confidence␣
 ↪interval')
plt.show()
```