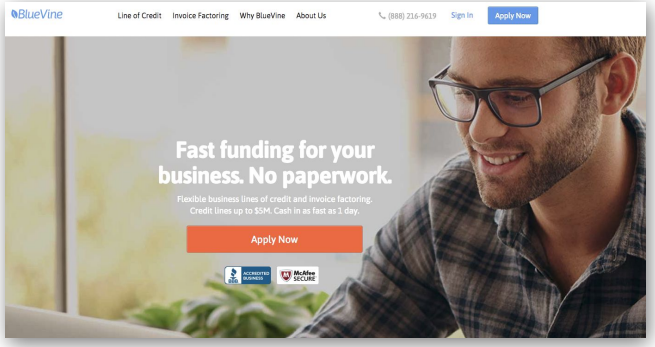
BlueVine

Anomaly Detection using Neural Networks

Dean Langsam

What We Do

- **Mission**
  - Offer convenient and flexible access to working capital for small and medium sized businesses
- **Products:**
  - Revolving line of credit
  - Invoice Factoring (Receivables backed financing)



- ✓ **24 hr. approvals**
- ✓ **100% online**
- ✓ **On demand**
- ✓ **Simple pricing**

Data Science in BlueVine

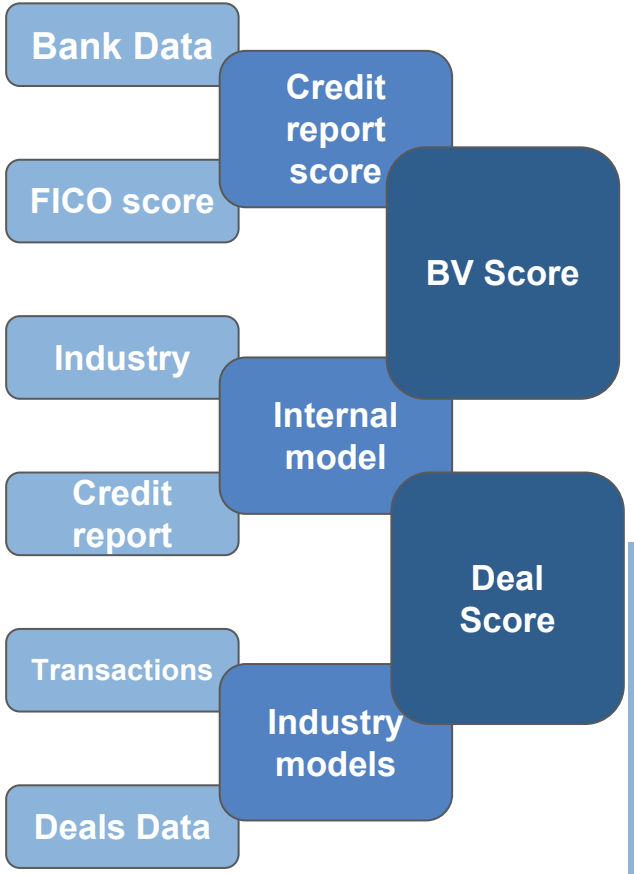
- Bridge the gap between **FAST** and **RISKY** in approving loans
- Build credit models
- Analyze fraud and fraudulent behaviour
- Analyze financial strength of clients
- Incorporating 3rd party data
- Data pipelines for hundreds of fundamental variables

Data Flow

	balance	income	feature_4	feature_5	feature_6	feature_7	feature_8	feature_9	feature_10	feature_11	...	feature_491	feature_492	feature_493	feature_494	feature_495	feature_496	feature_497	feature_498	feature_499	internal_model_score
user_id																					
10	1500.0	14000.0	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	medium
20	-1000.0	99000.0	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	medium
30	2000.0	25000.0	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	medium
40	10000.0	27000.0	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	medium
50	5000.0	13000.0	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	medium
60	7000.0	16000.0	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	medium
70	2000.0	35000.0	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	good
80	3000.0	85000.0	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	bad

8 rows × 499 columns

Data Flow



- Several primary data sources
- Dozens of models
- Hundreds of variables
- Thousands of total "things" to monitor

	variable	type	value	timestamp
user_id				
10	balance	numeric	1500	2018-04-01 01:00:00
20	balance	numeric	-1000	2018-04-01 02:00:00
30	income	numeric	25000	2018-04-01 01:00:00
40	income	numeric	27000	2018-04-01 02:00:00
50	income	numeric	13000	2018-04-01 03:00:00
60	income	numeric	16000	2018-04-01 04:00:00
70	internal_model_score	enum	good	2018-04-01 03:00:00
80	internal_model_score	enum	bad	2018-04-01 04:00:00
50	feature_4	date	...	...
10	feature_5	boolean	...	...
10	feature_6	enum	...	...
20	feature_7	text	...	...
50	feature_8	boolean	...	...
30	feature_9	date	...	...
10	feature_10	numeric	...	...
60	feature_11	text	...	...
70	feature_12	date	...	...
30	feature_13	boolean	...	...
60	feature_14	boolean	...	...
30	feature_15	date	...	...
80	feature_16	boolean	...	...
80	feature_17	text	...	...
30	feature_18	enum	...	...
60	feature_19	boolean	...	...
50	feature_20	enum	...	...
30	feature_21	date	...	...
70	feature_22	boolean	...	...
40	feature_23	numeric	...	...
10	feature_24	enum	...	...
70	feature_25	numeric	...	...
...	...	...	...	...
60	feature_466	boolean	...	...
80	feature_467	enum	...	...
60	feature_468	boolean	...	...

user_id	balance	income	feature_4	feature_5	feature_6	feature_7	feature_8	feature_9	feature_10	feature_11	...	feature_491	feature_492	feature_493	feature_494	feature_495	feature_496	feature_497	feature_498	feature_499	internal_model_score
10	1500.0	14000.0	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	medium
20	-1000.0	99000.0	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	medium
30	2000.0	25000.0	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	medium
40	10000.0	27000.0	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	medium
50	5000.0	13000.0	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	medium
60	7000.0	16000.0	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	medium
70	2000.0	35000.0	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	good
80	3000.0	85000.0	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	bad

8 rows × 499 columns

Anomaly Detection

Why we need it?

- Many data sources
- Big Team: Each model depends on results of other models
- Model scores are **mission critical** for the company
- Anomalies are proxies for corrupted data

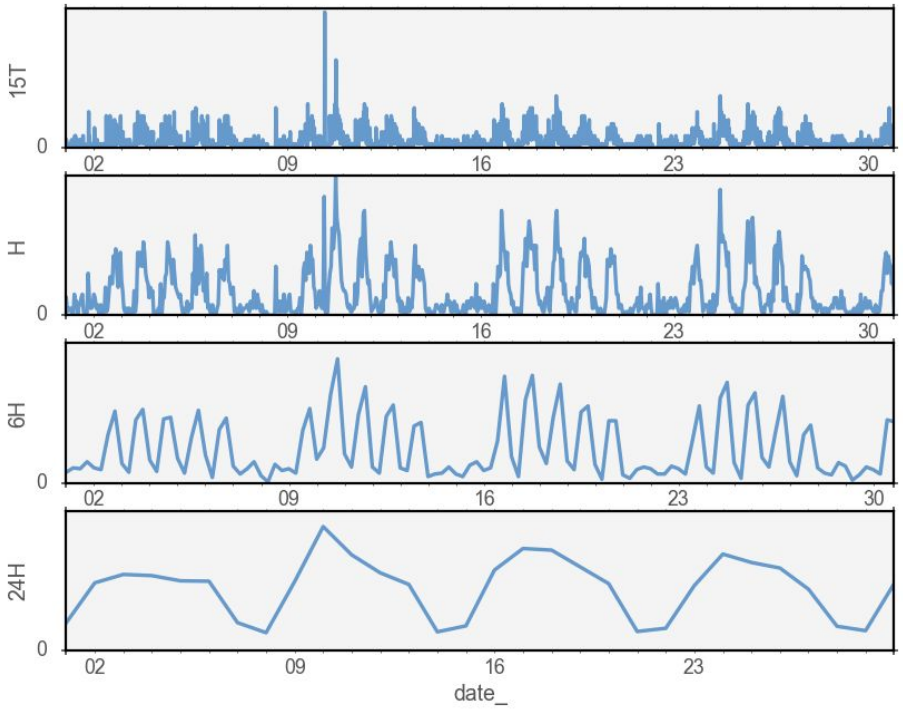
In this Talk

- Anomaly detection project
  - My choices
  - Build it end to end
- My first neural network
- Useful and modern Pandas
- Python is a friend, not a foe.

Initialization

- **Explore**: A small subset of variables
- **Understanding**: monitor counts data
- **Realization**:
  - Complex data pipeline
  - Several different underlying tasks
  - Independent tasks
- **Solution**: “Microservice” architecture

Same variable counts data on different intervals



Pandas and datetimes

- Penny drop moments (choir moments)

```
x = (1,2,3,
    4,
    5,6
)
```

- Why not

```
df2 = (df
      .groupby('col')
      .count()
      )
```

- Index with date (df.loc[])
- df.resample()
- df.assign (col = lambda...)

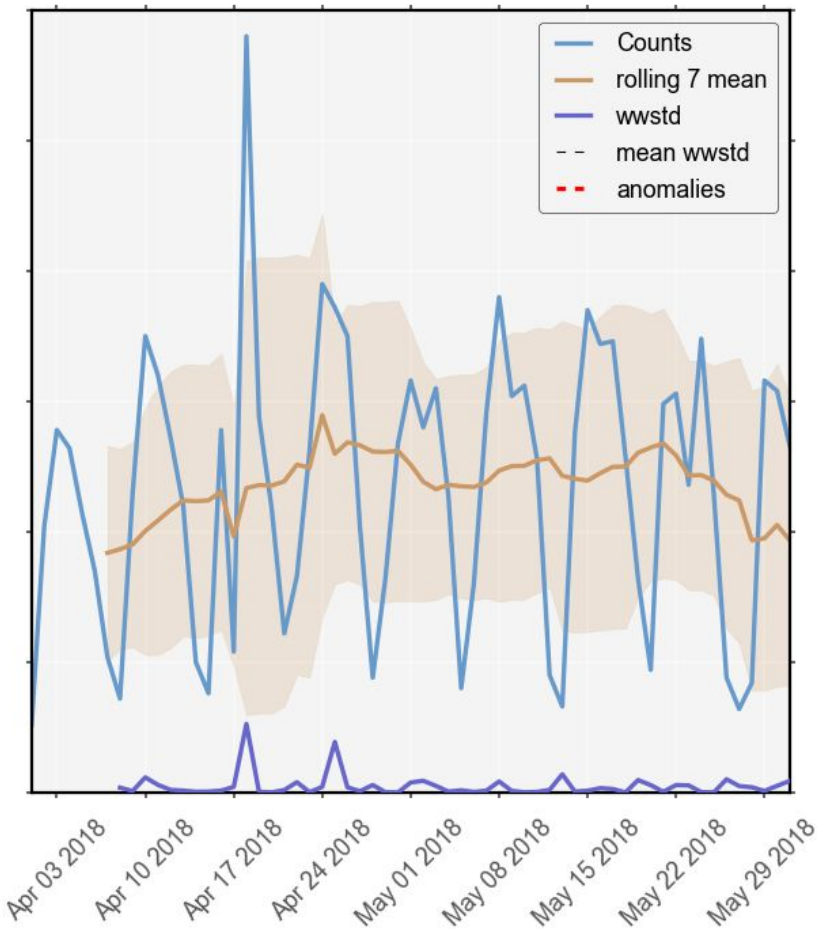
	val_updated	details	var_id
date_			
2018-05-24 21:40:19.779824+00:00	2018-05-24 21:40:19.779824+00:00	...	...
2018-04-18 04:57:53.556528+00:00	2018-04-18 04:57:53.556528+00:00	...	...
2018-04-20 14:43:27.927768+00:00	2018-04-20 14:43:27.927769+00:00	...	...
2018-05-14 23:18:17.710691+00:00	2018-05-14 23:18:17.710691+00:00	...	...
2018-05-31 02:25:23.537382+00:00	2018-05-31 02:25:23.537382+00:00	...	...

```
rs = (df
    .set_index('date_')
    .dropna()
    .loc['2017-1':,:]
    .resample('d')
    .apply('count')
    .rename(columns={'var_id':'counts'})
    .loc[:,['counts']]
#   .assign(weekend= lambda df: df.index.weekday >=5)
    .assign(diff_ = lambda df: df.counts.diff(1))
  )
```

	counts	weekend	diff_
date_			
2018-04-01 00:00:00+00:00	[illegible]	True	[illegible]
2018-04-02 00:00:00+00:00	[illegible]	False	[illegible]
2018-04-03 00:00:00+00:00	[illegible]	False	[illegible]
2018-04-04 00:00:00+00:00	[illegible]	False	[illegible]
2018-04-05 00:00:00+00:00	[illegible]	False	[illegible]
2018-04-06 00:00:00+00:00	[illegible]	False	[illegible]

Pandas date index

- .dt accessor
  - df.col.dt.day
  - .floor('1h') and .ceil('3d')
- df.col.rolling(window).mean()
- df.col.rolling(window).std()
- df.col.expanding...
- df.col.ewm...



Packages

- Luminol by LinkedIn
- Donut
  - Uses Tensorflow
- Skyline by Esty
  - No longer maintained

Update

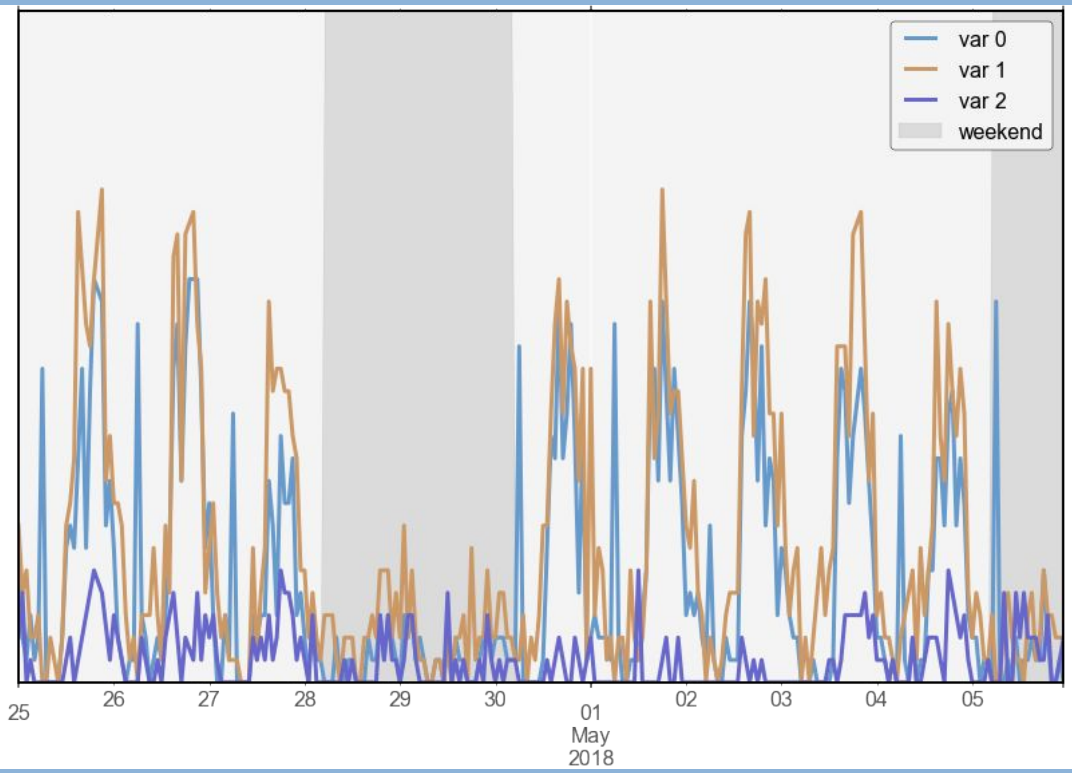
Fit (Training)

Predict

Alert

Update

- Acts as a “sensor”
- Aggregates raw data into fixed-time, fixed-scope observations
- Reads production tables on fixed time-intervals and writes aggregated summaries



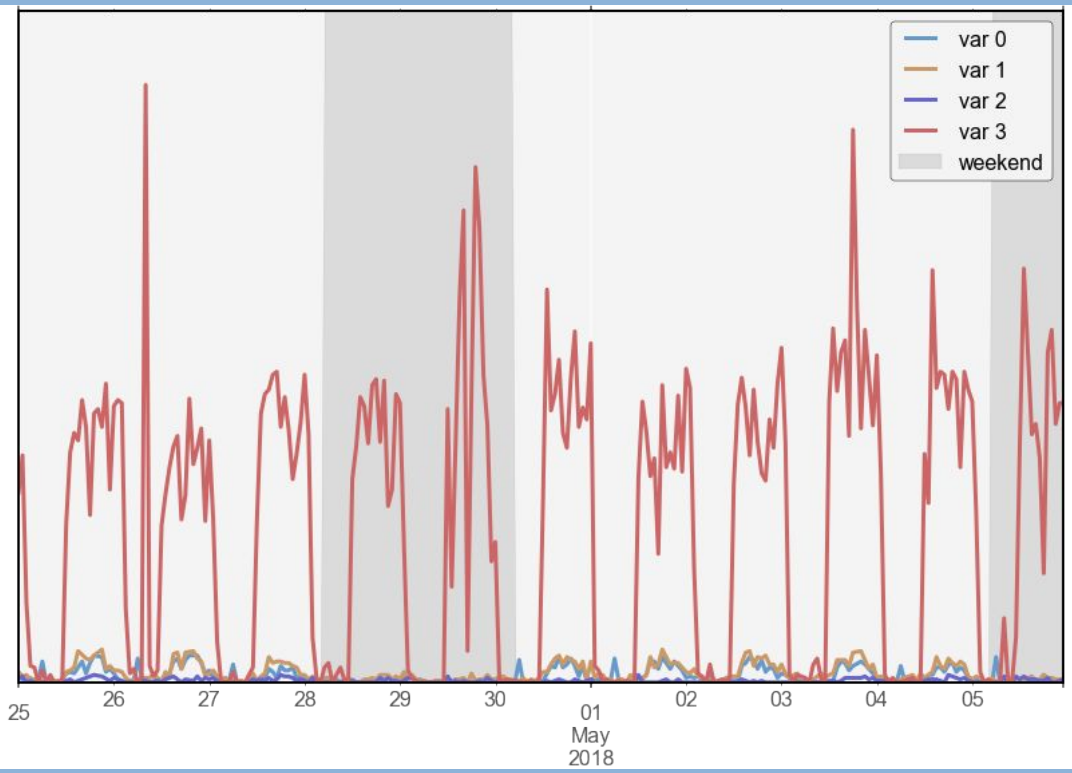
Fit (Training)

Predict

Alert

Update

- Some variables write more frequently than others
- Some variables show different patterns than others



Fit (Training)

Predict

Alert

Update

- df.pipe()
- pd.date_range()
- pd.reindex()

```
df = (df
    .pipe(count_db_writes, freq='1h')
    .pipe(pad_db_counts,min_date=min_date, max_date=max_date,ids=ids)
    .pipe(expand_db_counts)
    )
```

```
def pad_vector_counts (df):
    ret = df.copy()
    date_index = pd.date_range(start=min_date, end=max_date,
                    freq='H', tz='UTC', name='date_')
    ...
    ret = ret.reindex(index=date_index.union(ret.index), columns=var_ids)
    return ret
```

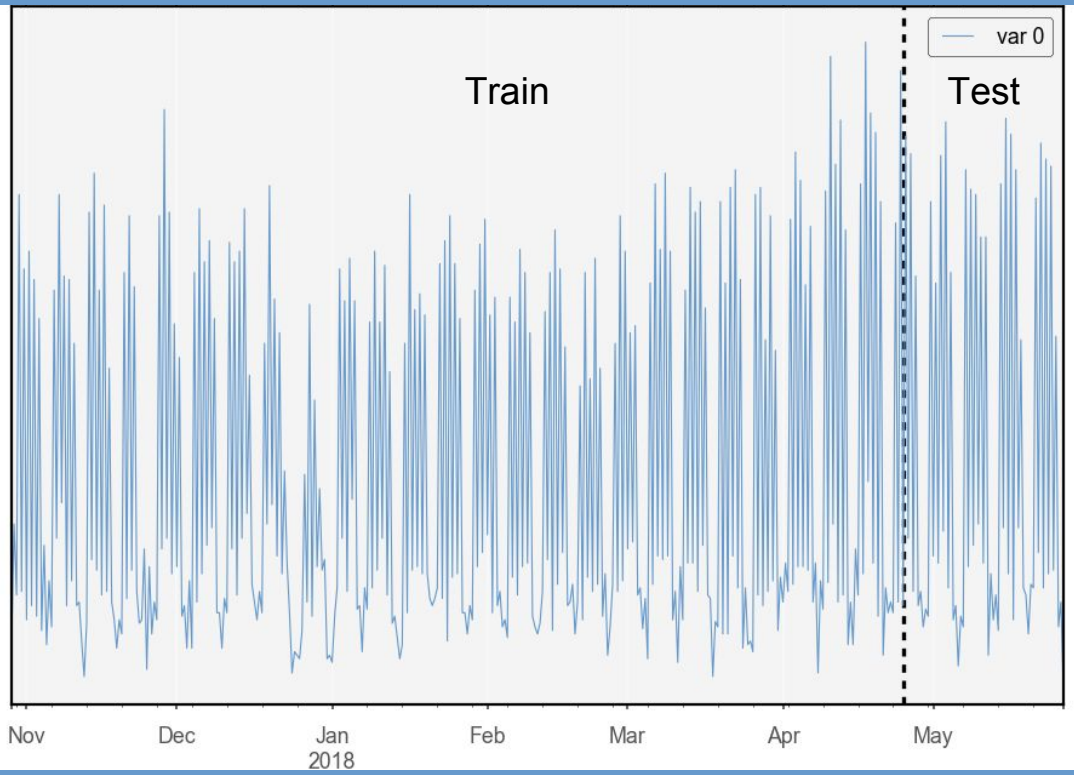
Fit (Training)

Predict

Alert

Fit (Training and Retraining)

- Given aggregated data, creates many time series
- Processes time-series into sliding windows
- Train/test split
- Data cleaning pipeline



Fit (Training and Retraining)

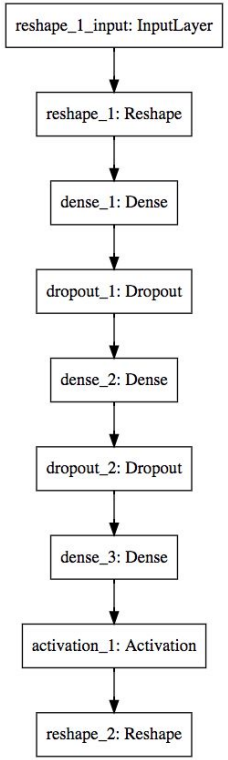
- Make u-turns: Use asserts
- One-stop-shop for cleaning
  - .pipe(clean)

```
def retrain_model(data):
  assert complete_counts_history(data), 'meaningful message {}'.format('')

  # ...
  train_start, train_end, test_start, test_end = get_train_test_period()
  train_data = create_sliding_windows(data,
                                      start=train_start, end=train_end, training=True)
  test_data = create_sliding_windows(data,
                                     start=test_start, end=test_end, training=False)
  scaler = MinMaxScaler().fit(train_data[['y']])
  X_train, y_train = process_input(train_data, scaler)
  X_test, y_test = process_input(test_data, scaler)
  # ...
  return model
```

Fit (Training and Retraining)

- Learns a fully-connected Neural network for each variable on counts data
- Optimizes MAE
- A good trade off between practical and efficient



Layer (type)	Output Shape	Param #
reshape_1 (Reshape)	(None, 14)	0
dense_1 (Dense)	(None, 32)	480
dropout_1 (Dropout)	(None, 32)	0
dense_2 (Dense)	(None, 16)	528
dropout_2 (Dropout)	(None, 16)	0
dense_3 (Dense)	(None, 1)	17
activation_1 (Activation)	(None, 1)	0
reshape_2 (Reshape)	(None, 1, 1)	0

Total params: 1,025
Trainable params: 1,025
Non-trainable params: 0

Fit (Training and Retraining)

- Create model inside a function
- Use functools.partial to iterate over options

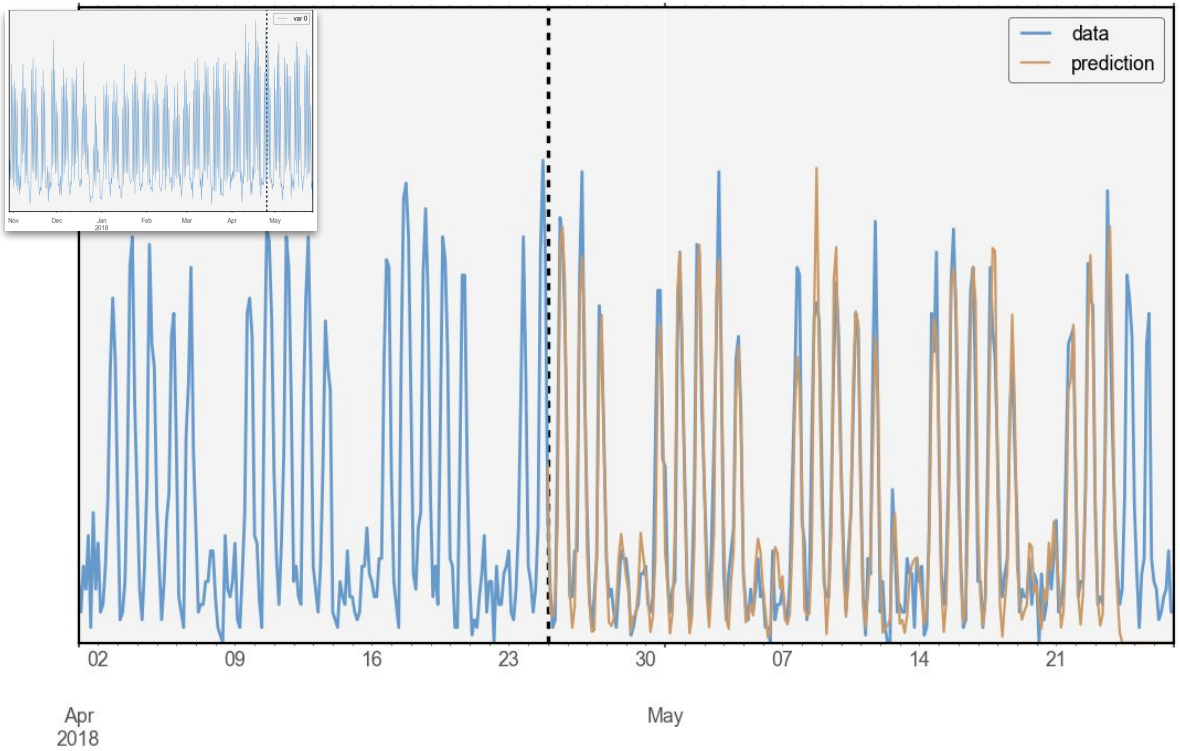
```
def make_dense_model(X_train, y_train, nb=16, loss='mse',
dropout=0.2, optimizer='adam'):
  in_shp = X_train.shape
  out_shp = y_train.shape[1]
  model = Sequential()
  model.add(Reshape((-1,), input_shape=(in_shp[1], in_shp[2])))
  model.add(Dense(2 * nb, activation='tanh'))
  model.add(Dropout(dropout))
  model.add(Dense(nb, activation='tanh'))
  model.add(Dropout(dropout))
  model.add(Dense(out_shp))
  model.add(Activation('relu'))
  model.add(Reshape((out_shp, 1)))
  model.compile(optimizer=optimizer, loss=loss)
  return model
```

```
from functools import partial

make_model = partial(make_dense_model, nb_nodes=16,
                     loss='mae')
###
models = [
  partial(make_dense_model, nb=16, loss='mae'),
  partial(make_lstm_model,    nb=16, loss='mse'),
  …,
  partial(make_lstm_model,    nb=4, loss='mse')
]
###
def make_and_fit(X,y,make_model):
  model = make_model(X, y)
  # Do more things
  fit_model(model, X, y)
  return model
```

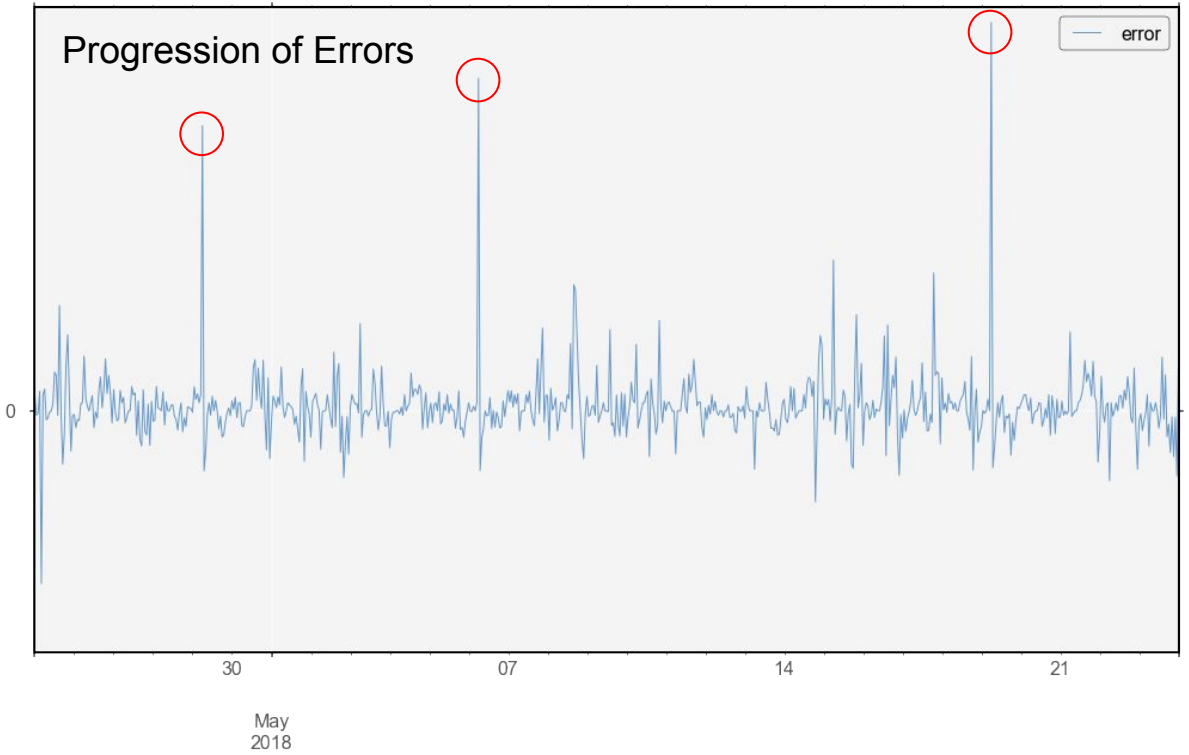
Fit (Training and Retraining)

- Uses test errors to compare model to previous models and chooses the best one
- Looks at errors of individual predictions



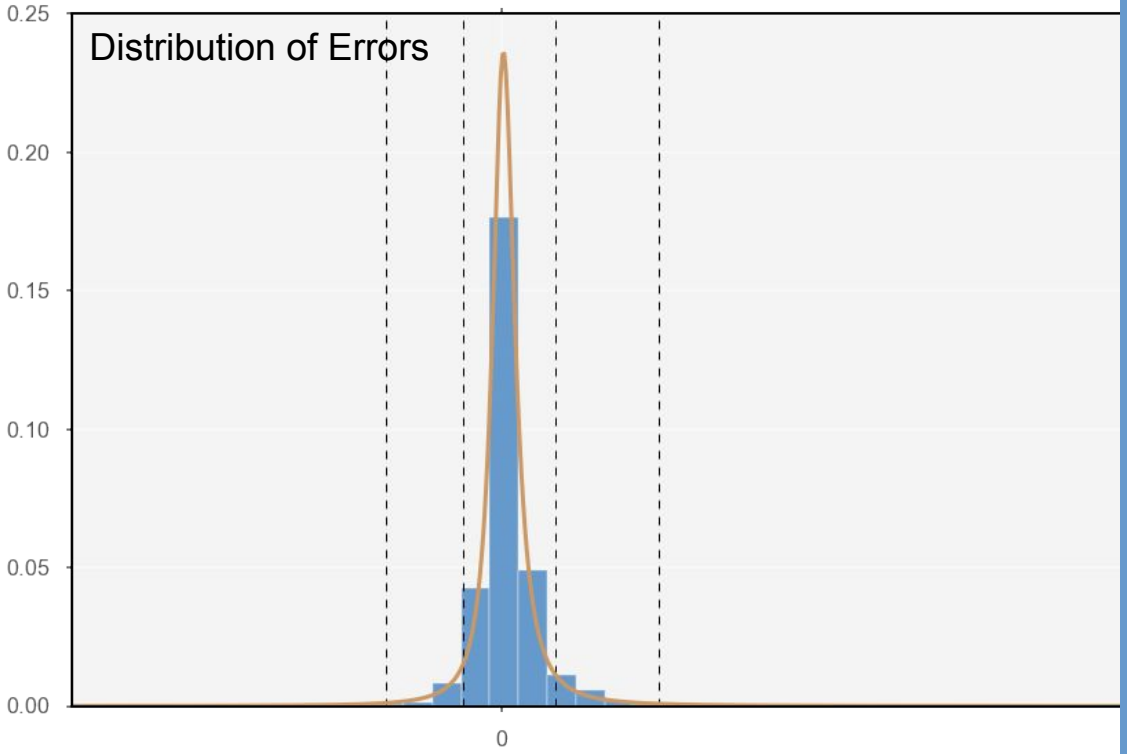
Fit (Training and Retraining)

- On the test period, creates a histogram of prediction error per observation
- Fits the histogram to a non-central T distribution
- Finds thresholds that constitute anomalies



Fit (Training and Retraining)

- Distribution parameters and thresholds are saved to metadata table
- Models are saved to S3
- Model info is saved to metadata table

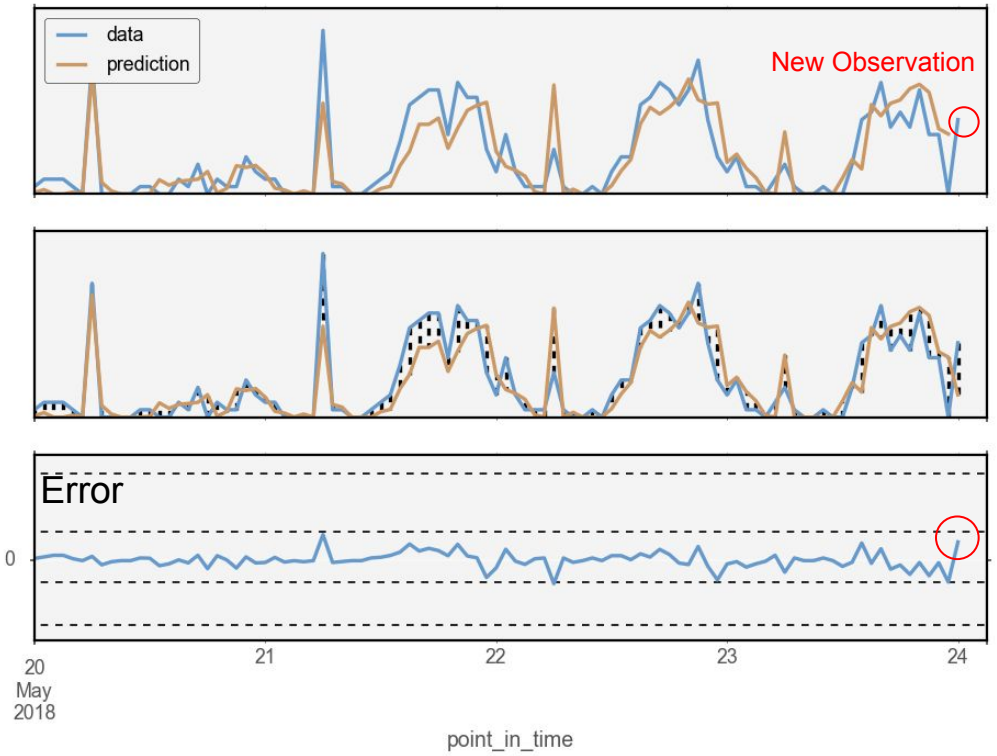


Update

Fit (Training)

Predict

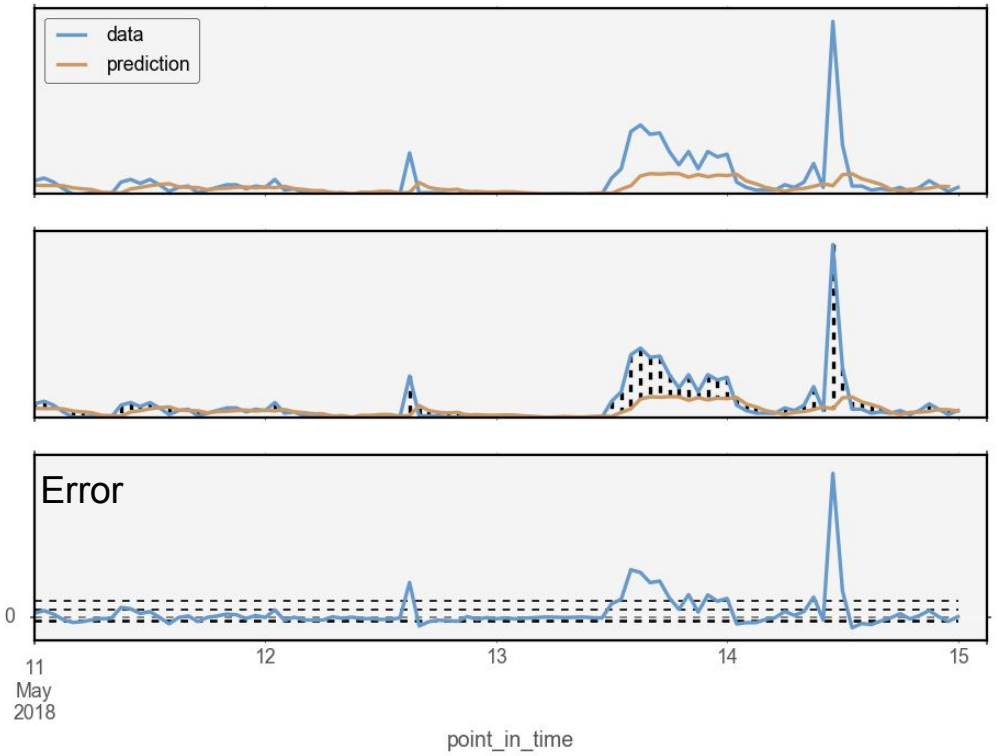
- Counts data is written
- Predicted data is compared against real data
- Error is compared against thresholds
- Error and anomaly info are saved to database



Alert

Predict

- Counts data is written
- Predicted data is compared against real data
- Error is compared against thresholds
- Error and anomaly info are saved to database



Predict

- Luckily we have the pipeline
- All model parameters exist
  - But not as python objects
- Make sure you can recreate everything from metadata
  - even the charts
- **Exception**: The model itself
- Make sure you can recreate the model itself from metadata
- getattr()

```
scaler = MinMaxScaler().fit([[db['scaler_min']], [db['scaler_max']]])
observation = create_sliding_windows(data, **db)
logger.info("Sliding windows created")
X, y = process_input(observation, scaler)
model = load_keras_model(**db)
```

```
def get_distribution_from_dict(db):
    dist_params = get_dist_params_from_db(db)
    dist = getattr(scipy.stats, db['dist_name'])
    return dist(*dist_params)
```

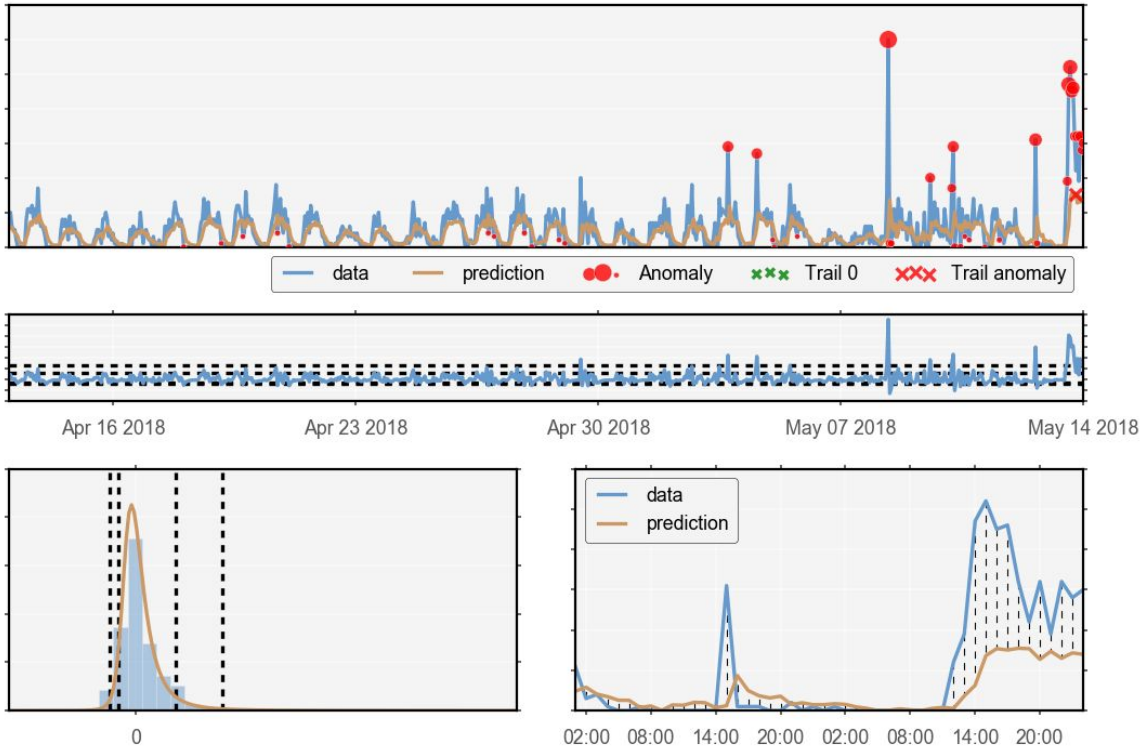
Update

Fit (Training)

Predict

Alert

- Take all anomalies from the database
- Create chart for each anomaly
- The chart captures all relevant information for current anomaly
  - Time series of actual vs. predicted
  - Time series of error progression
  - Histogram of errors with underlying distribution



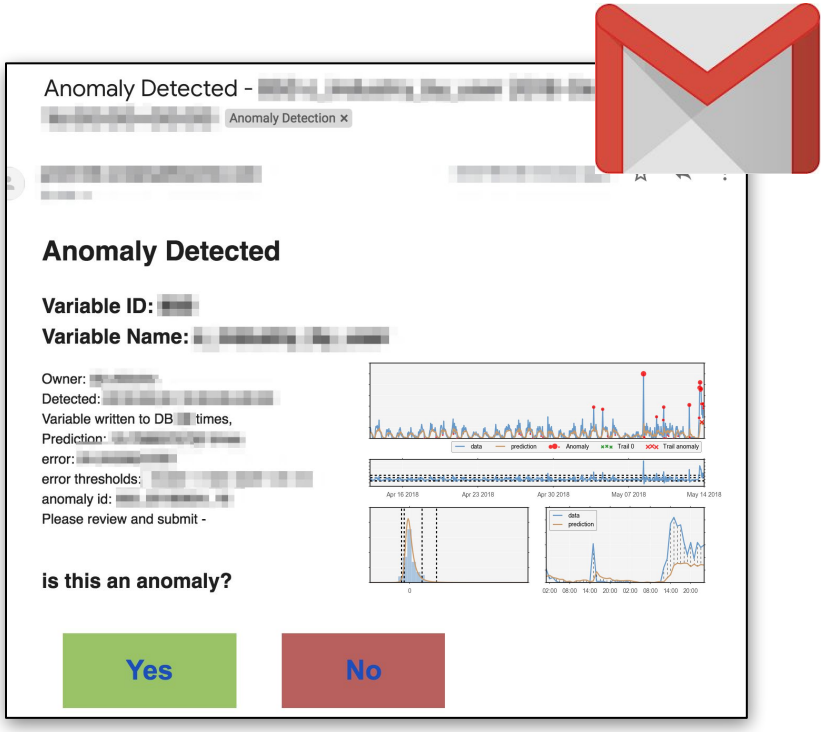
Update

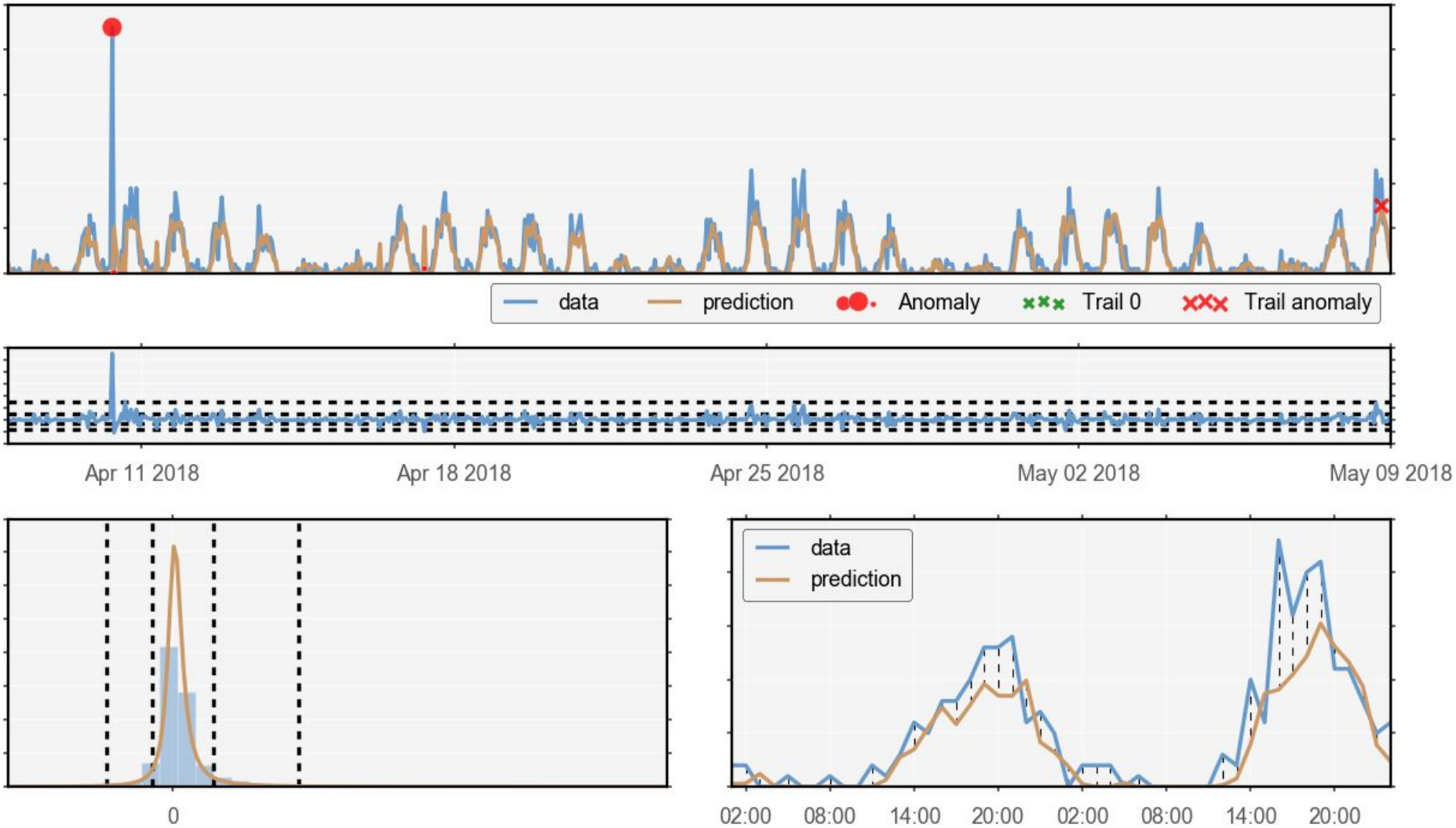
Fit (Training)

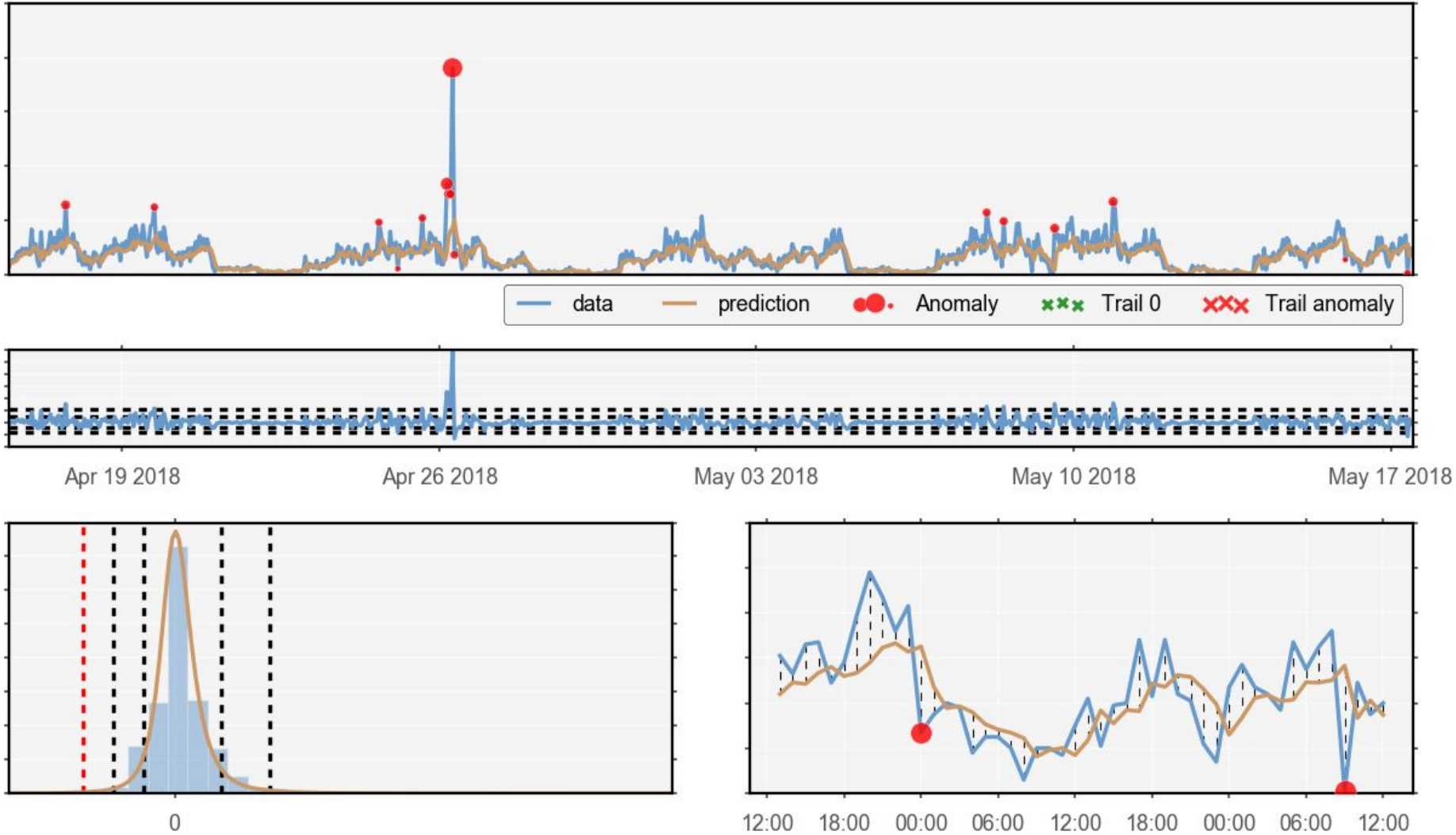
Predict

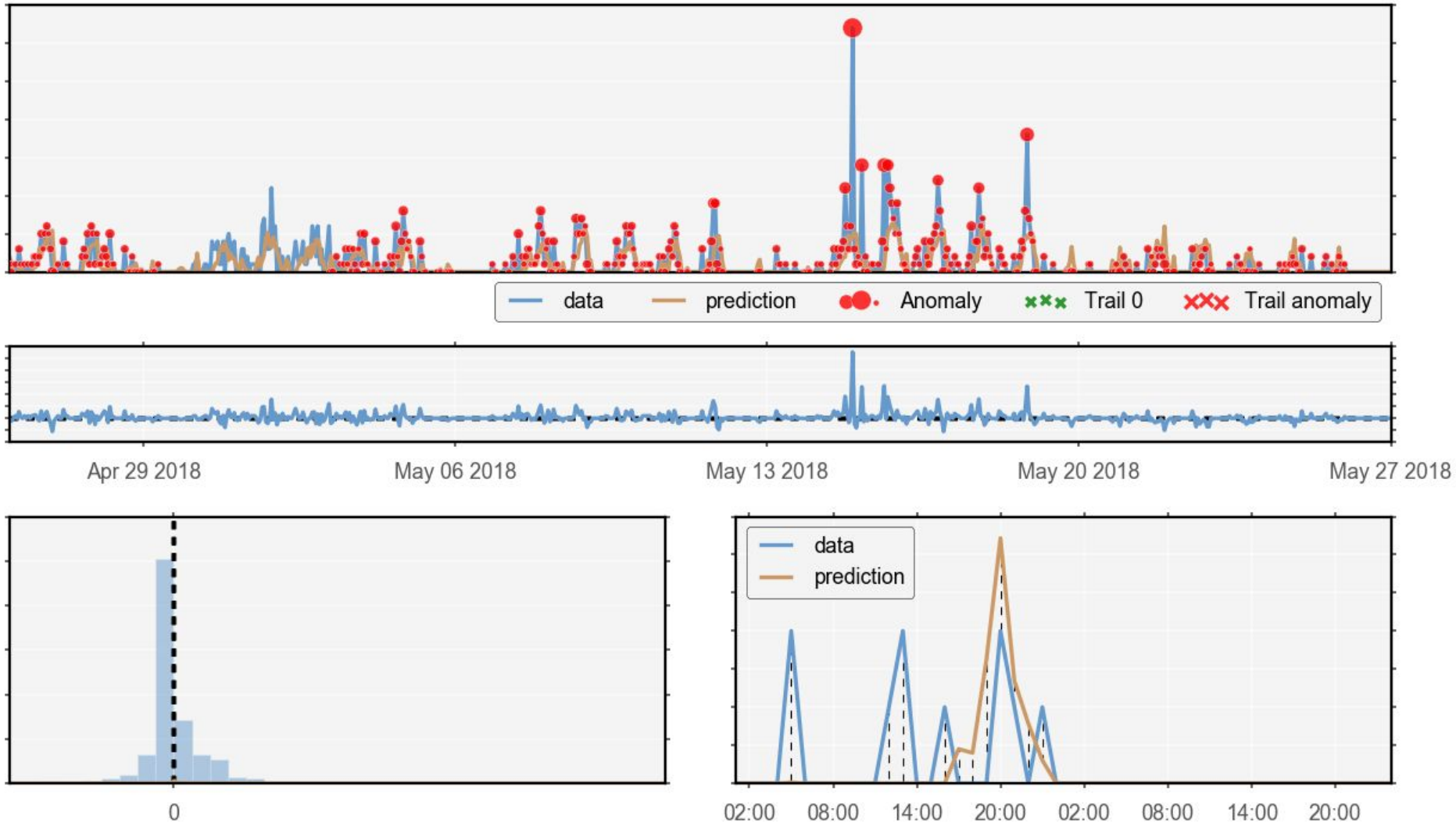
Alert

- Support several types of anomalies
- Feedback mechanism will allow supervised learning in the future




data
prediction
Anomaly
Trail 0
Trail anomaly
Apr 11 2018
Apr 18 2018
Apr 25 2018
May 02 2018
May 09 2018
0
data
prediction
02:00
08:00
14:00
20:00
02:00
08:00
14:00
20:00


data
prediction
Anomaly
Trail 0
Trail anomaly
Apr 19 2018
Apr 26 2018
May 03 2018
May 10 2018
May 17 2018
0
12:00
18:00
00:00
06:00
12:00
18:00
00:00
06:00
12:00


data
prediction
Anomaly
Trail 0
Trail anomaly
Apr 29 2018
May 06 2018
May 13 2018
May 20 2018
May 27 2018
0
data
prediction
02:00
08:00
14:00
20:00
02:00
08:00
14:00
20:00

Conclusions

- You don’t need to have a PHD in machine vision to deploy neural networks
- Pure Python has a lot more to offer than you use (But not dataframes)
- Diving into Pandas is a lot faster than inventing Pandas

BlueVine

Thank you

BlueVine