Some libraries/packages have been renamed or refactored within the repository, but the math is the same.

```
from apis.alpaca_markets import AlpacaMarkets

alpaca_markets = AlpacaMarkets(ALPACA_KEY, ALPACA_SECRET, SYMBOL)

historical_market_bars: dict[str, any] =
alpaca_markets.historical_market_bars(limit_bars=None,
weeks_data_window=144)[SYMBOL]
length_historical_market: int = len(historical_market_bars)

training_set_max_index = round((length_historical_market) *
TRAINING_SET_PORCENTAGE)

from database.duck_db import DuckDB

training_db = DuckDB(f'{SYMBOL.lower()}_training.set')
training_db.truncate()
training_set = [bar for bar in
historical_market_bars[:training_set_max_index]]

test_db = DuckDB(f'{SYMBOL.lower()}_test.set')
test_db.truncate()

config_db = DuckDB(f'{SYMBOL.lower()}_config.set')
config_db.truncate()

training_volume_set = [bar.volume for bar in training_set]
training_trade_count_set = [bar.trade_count for bar in training_set]
training_vwap_sep = [bar.vwap for bar in training_set]
```

```
from feature_encoders.z_score import ZScore

volume_zscore_obj: ZScore = ZScore(training_volume_set)
trade_count_zscore_obj: ZScore = ZScore(training_trade_count_set)
vwap_zscore_obj: ZScore = ZScore(training_vwap_sep)

zscore_config = {
    'volume_means': volume_zscore_obj.means,
    'volume_std': volume_zscore_obj.std,

    'trade_count_means': trade_count_zscore_obj.means,
    'trade_count_std': trade_count_zscore_obj.std,

    'vwap_means': vwap_zscore_obj.means,
    'vwap_std': vwap_zscore_obj.std
}

config_db.insert(zscore_config)

from datetime import datetime

from features_vectorizer import TimeSeriesConfig, TimeConfig,
NormalizationConfig, features_vectorizer

training_set_max_index_offset: int = training_set_max_index - 1
training_set = []

for index, bar in enumerate(historical_market_bars):
    raw_window: list[dict] = historical_market_bars[index -
WINDOW_PERIODS: index]
    raw_prices_window: list[float] = [bar_dict.close for bar_dict in
raw_window]

    if len(raw_prices_window) < WINDOW_PERIODS:
        continue

    price_window_range: float = bar.high - bar.low

    stop_loss: float = 0.5 * price_window_range
    take_profit: float = 1.5 * price_window_range

    bar_timestamp: datetime = bar.timestamp
    bar_formated_timestamp: str = bar_timestamp.strftime('%Y-%m-%d %H:
%M %Z')

    bar_volume = bar.volume
    bar_trade_count = bar.trade_count
    bar_vwap = bar.vwap

    time_series_config: TimeSeriesConfig =
TimeSeriesConfig(raw_vector=raw_prices_window, periods=WINDOW_PERIODS)
```

```
    time_config: TimeConfig =
TimeConfig(current_minute=bar_timestamp.minute + 1,
current_hour=bar_timestamp.hour + 1,
current_day_of_week=bar_timestamp.weekday() + 1,
current_month=bar_timestamp.month)
    normalization_config: NormalizationConfig =
NormalizationConfig(volume_zscore_obj=volume_zscore_obj,
volume=bar_volume, trade_count_zscore_obj=trade_count_zscore_obj,
trade_count=bar_trade_count, vwap_zscore_obj=vwap_zscore_obj,
vwap=bar_vwap)

    features_vector = features_vectorizer(time_series_config,
time_config, normalization_config)

    entry_bar_price = bar.close

    up_take_profit_price = entry_bar_price + take_profit
    up_stop_loss_price = entry_bar_price - stop_loss
    up_stop_loss_was_triggered = False

    down_take_profit_price = entry_bar_price - take_profit
    down_stop_loss_price = entry_bar_price + stop_loss
    down_stop_loss_was_triggered = False

    next_price_bars = historical_market_bars[index + 1: index + 1 +
WINDOW_PERIODS]
    bar_label = 0.5

    for next_bar in next_price_bars:

        if next_bar.high <= up_stop_loss_price:
            up_stop_loss_was_triggered = True

        if next_bar.high >= up_take_profit_price and
up_stop_loss_was_triggered == False:
            bar_label = 1
            break

        if next_bar.low >= down_stop_loss_price:
            down_stop_loss_was_triggered = True

        if next_bar.low <= down_take_profit_price and
down_stop_loss_was_triggered == False:
            bar_label = 0
            break

    bar_metadata_vectorized = {
        'timestamp': bar_formated_timestamp,
        'vector': features_vector,
        'label': bar_label
```

```
    }

    if index <= training_set_max_index_offset:
        training_set.append(bar_metadata_vectorized)

    if index >= training_set_max_index_offset:
        test_db.insert(bar_metadata_vectorized)

training_set_sanitized = []

for example in training_set:
    if example['label'] == 0.5:
        continue

    training_set_sanitized.append(example)

training_db.truncate_and_insert_list(training_set_sanitized)

from core.uncertainty_simple_perceptron import
UncertaintySimplePerceptron

uncertainty_simple_perceptron = UncertaintySimplePerceptron(NAME,
DESCRIPTION, AUTHOR)

uncertainty_simple_perceptron.train(training_db.database_file_name,
EPOCHS, PATIENCE, LEARNING_RATE)
```

```
Current Epoch: 1/170 | Error Rate: 0.5211385199240987 | Elapsed Time
(ms): 85.4759
Current Epoch: 2/170 | Error Rate: 0.5124857685009487 | Elapsed Time
(ms): 83.1233
Current Epoch: 3/170 | Error Rate: 0.5151043643263757 | Elapsed Time
(ms): 83.3402
Current Epoch: 4/170 | Error Rate: 0.5163567362428843 | Elapsed Time
(ms): 86.3979
Current Epoch: 5/170 | Error Rate: 0.5125237191650854 | Elapsed Time
(ms): 85.4073
Current Epoch: 6/170 | Error Rate: 0.5098671726755218 | Elapsed Time
(ms): 85.3607
Current Epoch: 7/170 | Error Rate: 0.5074003795066414 | Elapsed Time
(ms): 98.1593
Current Epoch: 8/170 | Error Rate: 0.5055028462998102 | Elapsed Time
(ms): 80.7929
Current Epoch: 9/170 | Error Rate: 0.5031878557874763 | Elapsed Time
(ms): 80.6139
Current Epoch: 10/170 | Error Rate: 0.5001138519924099 | Elapsed Time
(ms): 84.0378
Current Epoch: 11/170 | Error Rate: 0.49806451612903224 | Elapsed Time
(ms): 77.978
Current Epoch: 12/170 | Error Rate: 0.49555977229601517 | Elapsed Time
(ms): 82.4673
```

```
Current Epoch: 13/170 | Error Rate: 0.49404174573055026 | Elapsed Time (ms): 82.5503
Current Epoch: 14/170 | Error Rate: 0.49240986717267554 | Elapsed Time (ms): 83.5415
Current Epoch: 15/170 | Error Rate: 0.4901707779886148 | Elapsed Time (ms): 79.4928
Current Epoch: 16/170 | Error Rate: 0.4876280834914611 | Elapsed Time (ms): 86.051
Current Epoch: 17/170 | Error Rate: 0.4849335863377609 | Elapsed Time (ms): 79.1315
Current Epoch: 18/170 | Error Rate: 0.4826565464895636 | Elapsed Time (ms): 83.4238
Current Epoch: 19/170 | Error Rate: 0.4774193548387097 | Elapsed Time (ms): 83.4745
Current Epoch: 20/170 | Error Rate: 0.47506641366223906 | Elapsed Time (ms): 78.8611
Current Epoch: 21/170 | Error Rate: 0.4629981024667932 | Elapsed Time (ms): 80.0184
Current Epoch: 22/170 | Error Rate: 0.43070208728652754 | Elapsed Time (ms): 81.9144
Current Epoch: 23/170 | Error Rate: 0.31586337760910815 | Elapsed Time (ms): 75.6202
Current Epoch: 24/170 | Error Rate: 0.28956356736242883 | Elapsed Time (ms): 75.6803
Current Epoch: 25/170 | Error Rate: 0.28755218216318784 | Elapsed Time (ms): 75.654
Current Epoch: 26/170 | Error Rate: 0.2900569259962049 | Elapsed Time (ms): 73.0569
Current Epoch: 27/170 | Error Rate: 0.2893358633776091 | Elapsed Time (ms): 73.5526
Current Epoch: 28/170 | Error Rate: 0.2893738140417457 | Elapsed Time (ms): 73.0449
Current Epoch: 29/170 | Error Rate: 0.290853889943074 | Elapsed Time (ms): 69.88
Current Epoch: 30/170 | Error Rate: 0.2884629981024668 | Elapsed Time (ms): 67.4469
Current Epoch: 31/170 | Error Rate: 0.2882732447817837 | Elapsed Time (ms): 69.1565
Current Epoch: 32/170 | Error Rate: 0.2887286527514232 | Elapsed Time (ms): 69.7186
Current Epoch: 33/170 | Error Rate: 0.2894876660341556 | Elapsed Time (ms): 67.59
Current Epoch: 34/170 | Error Rate: 0.2858823529411765 | Elapsed Time (ms): 67.9566
Current Epoch: 35/170 | Error Rate: 0.2879696394686907 | Elapsed Time (ms): 75.0225
Current Epoch: 36/170 | Error Rate: 0.28857685009487666 | Elapsed Time (ms): 73.0495
Current Epoch: 37/170 | Error Rate: 0.28789373814041747 | Elapsed Time
```

```
(ms): 73.2993
Current Epoch: 38/170 | Error Rate: 0.2887286527514232 | Elapsed Time
(ms): 68.2486
Current Epoch: 39/170 | Error Rate: 0.28629981024667933 | Elapsed Time
(ms): 71.7435
Current Epoch: 40/170 | Error Rate: 0.2884250474383302 | Elapsed Time
(ms): 70.7052
Current Epoch: 41/170 | Error Rate: 0.2882732447817837 | Elapsed Time
(ms): 73.0283
Current Epoch: 42/170 | Error Rate: 0.28823529411764703 | Elapsed Time
(ms): 73.6208
Current Epoch: 43/170 | Error Rate: 0.2882732447817837 | Elapsed Time
(ms): 71.5291
Early Stopping Trigged at Epoch 43 — No Improvement in Error Rate for
20 epochs.
Model Restored from Epoch 34 (Error Rate: 0.2858823529411765).
Early Stopping — Models Saved as
`tesla_stocks_uncertainty_simple_perceptron_hft_2025-11-16_00-33-
45.json` in the Directory: c:\Users\csutt\OneDrive\Desktop\aardvark-
uncertainty_simple_perceptron\models.
```

```
from numpy import array

successful_pred: int = 0
failed_pred: int = 0
uncertainty_pred: int = 0

for example in test_db.line_by_line():
    inference_prediction =
uncertainty_simple_perceptron.inference(input_features=array(example['
vector']), epsilon=0.007)

    prediction, net_output = inference_prediction

    if prediction == example['label']:
        successful_pred += 1

    elif prediction == 0.5:
        uncertainty_pred += 1

    else:
        failed_pred += 1

print(successful_pred, failed_pred, uncertainty_pred)
```

```
2095 0 9827
```