In [1]:

```
import pandas
import numpy
import sklearn
```

In [2]:

```
from sklearn import model_selection
from sklearn import ensemble
from sklearn import metrics
from sklearn import pipeline
from sklearn import impute
from sklearn import preprocessing
from sklearn import feature_selection
from sklearn import model_selection
```

In [35]:

```
import strat
```

In [4]:

```
import constant
```

In [5]:

```
# stocks_file_path, sp500_file_path = strat.saved_data_file_paths_from_url(url=constant.d
ata_url)
# assert stocks_file_path == constant.stocks_file_path
# assert sp500_file_path == constant.sp500_file_path
```

In [6]:

```
stocks_data_frame = strat.data_frame_from_data_file_path(constant.stocks_file_path)
sp500_data_frame = strat.data_frame_from_data_file_path(constant.sp500_file_path)
```

In [7]:

```
constant.utmost_date
```

Out[7]:

```
'2020-01-01'
```

In [8]:

```
print(f"Stocks DataFrame:\n{stocks_data_frame.head()}\n")
print(f"S&P 500 DataFrame:\n{sp500_data_frame.head()}\n")
```

```
Stocks DataFrame:
            date      price ticker
71611 2013-02-08  45.080002      A
71612 2013-02-11  44.599998      A
71613 2013-02-12  44.619999      A
71614 2013-02-13  44.750000      A
71615 2013-02-14  44.580002      A

S&P 500 DataFrame:
           date        price
2579 2012-07-31  1379.319946
2578 2012-08-01  1375.140015
2577 2012-08-02  1365.000000
2576 2012-08-03  1390.989990
2575 2012-08-06  1394.229980
```

In [9]:

```
sp500_data_frame[constant.ColumnNames.date].unique()
```

Out[9]:

```
<DatetimeArray>
['2012-07-31 00:00:00', '2012-08-01 00:00:00', '2012-08-02 00:00:00',
 '2012-08-03 00:00:00', '2012-08-06 00:00:00', '2012-08-07 00:00:00',
 '2012-08-08 00:00:00', '2012-08-09 00:00:00', '2012-08-10 00:00:00',
 '2012-08-13 00:00:00',
 ...
 '2022-10-17 00:00:00', '2022-10-18 00:00:00', '2022-10-19 00:00:00',
 '2022-10-20 00:00:00', '2022-10-21 00:00:00', '2022-10-24 00:00:00',
 '2022-10-25 00:00:00', '2022-10-26 00:00:00', '2022-10-27 00:00:00',
 '2022-10-28 00:00:00']
Length: 2580, dtype: datetime64[ns]
```

In [ ]:

```

```

In [10]:

```
stocks_data_frame.dropna(inplace=True)
sp500_data_frame.dropna(inplace=True)
```

In [11]:

```
# Feature Engineering
def rolling_mean_series(prices, window=20):
    return prices.rolling(window=window).mean()
def rolling_standard_deviation_series(prices, window=20):
    return prices.rolling(window=window).std()
def bollinger_lower_band_series(rolling_means, rolling_standard_deviations):
    return rolling_means - (2 * rolling_standard_deviations)
def bollinger_upper_band_series(rolling_means, rolling_standard_deviations):
    return rolling_means + (2 * rolling_standard_deviations)

def relative_strength_index_series(prices, window=14):
    delta_prices = prices.diff()
    gain_series = (delta_prices.where(delta_prices > 0, 0)).rolling(window=window).mean(
)
    loss_series = (-delta_prices.where(delta_prices < 0, 0)).rolling(window=window).mean
()
    rs_series = gain_series / loss_series
    rsi_series = 100 - (100 / (1 + rs_series))
    return rsi_series

def moving_average_convergence_divergence_series(delta_prices, short_window=12, long_wind
ow=26):
    short_ema_series = delta_prices.ewm(span=short_window, min_periods=1).mean()
    long_ema_series = delta_prices.ewm(span=long_window, min_periods=1).mean()
    macd_series = short_ema_series - long_ema_series
    return macd_series
```

In [12]:

```
stocks_data_frame = stocks_data_frame.sort_values(by=[constant.ColumnNames.ticker, const
ant.ColumnNames.date])
```

In [13]:

```
def add_features_to_data_frame(the_data_frame):
    the_data_frame['rolling_mean_20'] = the_data_frame.groupby(
        by=constant.ColumnNames.ticker, group_keys=False
    ).apply(
        lambda data_frame: rolling_mean_series(data_frame[constant.ColumnNames.price])
    )
    the_data_frame['rolling_standard_deviations_20'] = the_data_frame.groupby(
        by=constant.ColumnNames.ticker,group_keys=False
    ).apply(
```

```
        lambda data_frame: rolling_standard_deviation_series(data_frame[constant.ColumnNa
mes.price])
    )
    the_data_frame['bollinger_lower_band'] = bollinger_lower_band_series(
        the_data_frame['rolling_mean_20'], the_data_frame['rolling_standard_deviations_2
0'],
    )
    the_data_frame['bollinger_upper_band'] = bollinger_upper_band_series(
        the_data_frame['rolling_mean_20'], the_data_frame['rolling_standard_deviations_2
0'],
    )
    the_data_frame['relative_strength_index'] = relative_strength_index_series(the_data_f
rame[constant.ColumnNames.price])
    the_data_frame['moving_average_convergence_divergence'] = moving_average_convergence_
divergence_series(
        the_data_frame[constant.ColumnNames.price]
    )

    the_data_frame['rolling_mean_50'] = the_data_frame.groupby(
        by=constant.ColumnNames.ticker, group_keys=False
    ).apply(
        lambda data_frame: rolling_mean_series(data_frame[constant.ColumnNames.price], 5
0)
    )

    the_data_frame['rolling_mean_100'] = the_data_frame.groupby(
        by=constant.ColumnNames.ticker, group_keys=False
    ).apply(
        lambda data_frame: rolling_mean_series(data_frame[constant.ColumnNames.price], 1
00)
    )

    the_data_frame['momentum'] = the_data_frame[constant.ColumnNames.price] / the_data_f
rame[constant.ColumnNames.price].shift(10) - 1
    the_data_frame['volatility'] = the_data_frame[constant.ColumnNames.price].rolling(wi
ndow=20).std()

    the_data_frame = the_data_frame.dropna()

    features_column_names = [
        'rolling_mean_20','rolling_standard_deviations_20',
        'bollinger_lower_band','bollinger_lower_band',
        'moving_average_convergence_divergence',
        'volatility', 'momentum',
        'rolling_mean_50','rolling_mean_100'
    ]

    return the_data_frame, features_column_names
```

In [14]:

```
stocks_data_frame, features_column_names = add_features_to_data_frame(stocks_data_frame)
```

```
/tmp/ipykernel_311251/2817862000.py:4: DeprecationWarning: DataFrameGroupBy.apply operate
d on the grouping columns. This behavior is deprecated, and in a future version of pandas
the grouping columns will be excluded from the operation. Either pass `include_groups=Fal
se` to exclude the groupings or explicitly select the grouping columns after groupby to s
ilence this warning.
  ).apply(
/tmp/ipykernel_311251/2817862000.py:9: DeprecationWarning: DataFrameGroupBy.apply operate
d on the grouping columns. This behavior is deprecated, and in a future version of pandas
the grouping columns will be excluded from the operation. Either pass `include_groups=Fal
se` to exclude the groupings or explicitly select the grouping columns after groupby to s
ilence this warning.
  ).apply(
/tmp/ipykernel_311251/2817862000.py:25: DeprecationWarning: DataFrameGroupBy.apply operat
ed on the grouping columns. This behavior is deprecated, and in a future version of panda
s the grouping columns will be excluded from the operation. Either pass `include_groups=F
alse` to exclude the groupings or explicitly select the grouping columns after groupby to
silence this warning.
  ).apply(
/tmp/ipykernel_311251/2817862000.py:31: DeprecationWarning: DataFrameGroupBy.apply operat
```

```
ed on the grouping columns. This behavior is deprecated, and in a future version of panda
s the grouping columns will be excluded from the operation. Either pass `include_groups=F
alse` to exclude the groupings or explicitly select the grouping columns after groupby to
silence this warning.
  ).apply(
```

In [15]:

```
stocks_data_frame = stocks_data_frame.dropna()
```

In [16]:

```
stocks_data_frame[features_column_names]
```

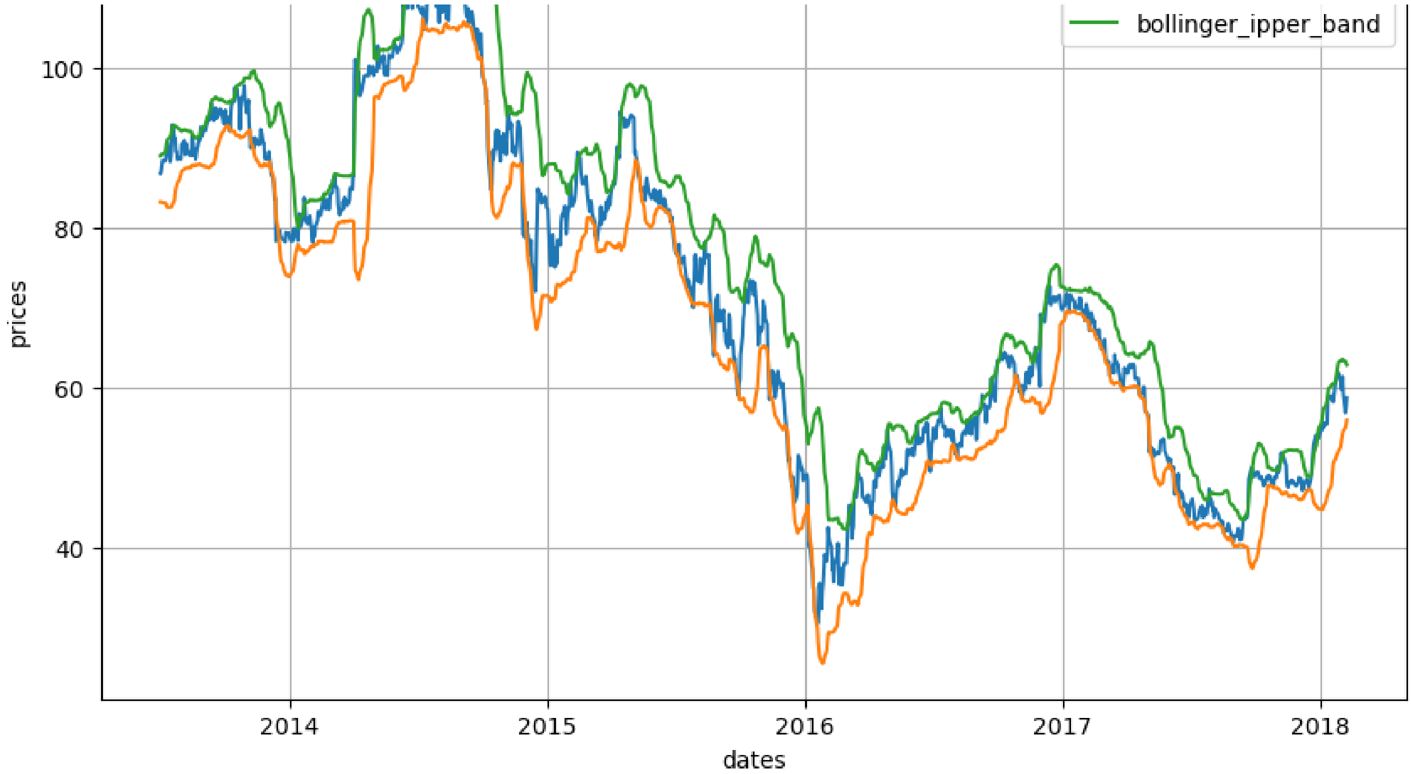
Out[16]:

| | rolling_mean_20 | rolling_standard_deviations_20 | bollinger_lower_band | bollinger_lower_band | moving_average_converge |
|---|---|---|---|---|---|
| **71710** | 43.6940 | 0.969810 | 41.754381 | 41.754381 | |
| **71711** | 43.6260 | 0.955683 | 41.714633 | 41.714633 | |
| **71712** | 43.5870 | 0.911108 | 41.764785 | 41.764785 | |
| **71713** | 43.5475 | 0.855188 | 41.837124 | 41.837124 | |
| **71714** | 43.5410 | 0.846173 | 41.848653 | 41.848653 | |
| **...** | ... | ... | ... | ... | |
| **619035** | 76.4605 | 2.097659 | 72.265182 | 72.265182 | |
| **619036** | 76.6730 | 1.882795 | 72.907411 | 72.907411 | |
| **619037** | 76.6965 | 1.841753 | 73.012994 | 73.012994 | |
| **619038** | 76.6480 | 1.920917 | 72.806167 | 72.806167 | |
| **619039** | 76.5855 | 1.992590 | 72.600320 | 72.600320 | |

569100 rows × 9 columns

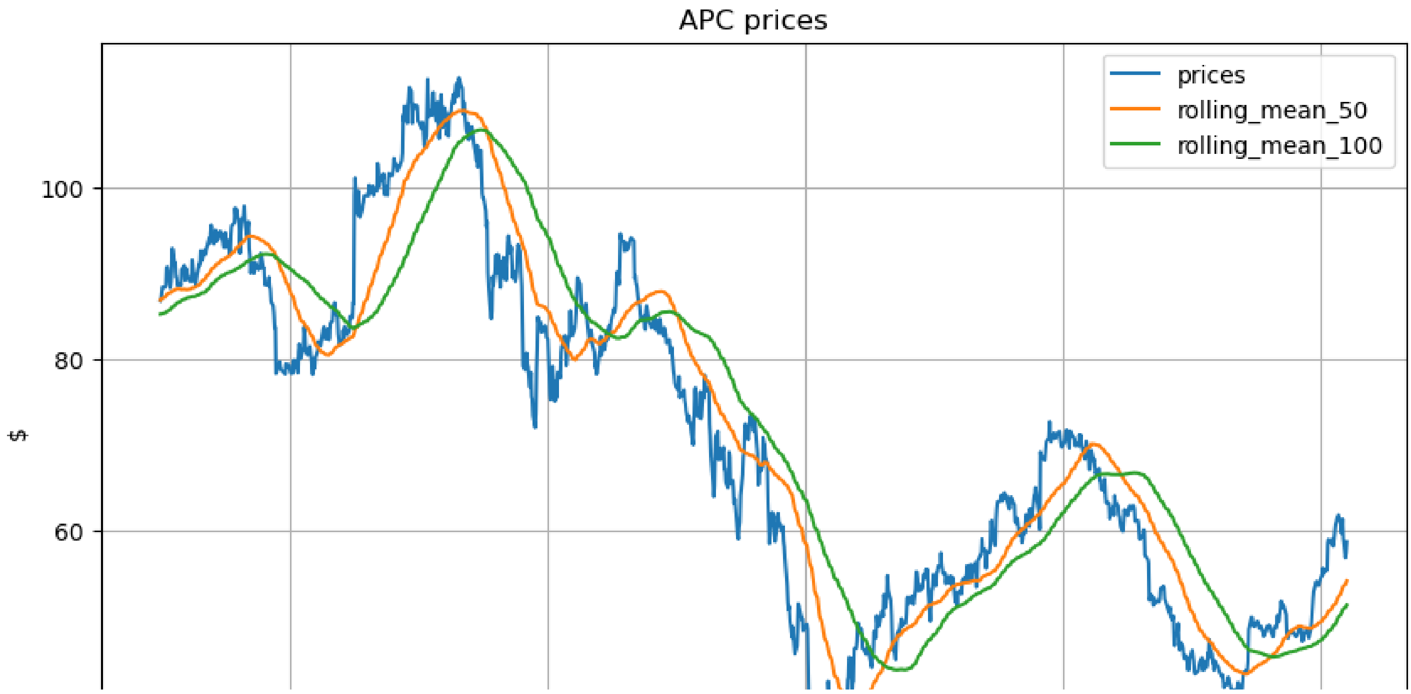
In [17]:

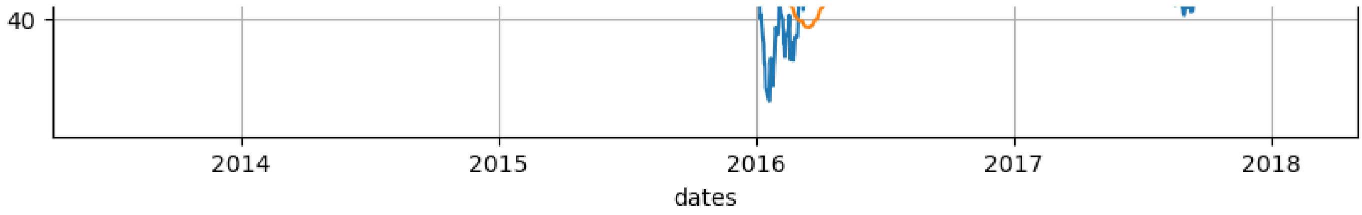
```
random_ticker = stocks_data_frame['ticker'].unique()[45]
random_ticker_data_frame = stocks_data_frame[
       stocks_data_frame['ticker'] == random_ticker
]
random_ticker_prices = random_ticker_data_frame['price']
random_ticker_dates = random_ticker_data_frame['date']
random_ticker_bollinger_lower_band =random_ticker_data_frame['bollinger_lower_band']
random_ticker_bollinger_upper_band =random_ticker_data_frame['bollinger_upper_band']

strat.plot_multiple_series(
    x=random_ticker_dates,
    y_series_list=[
        random_ticker_prices,
        random_ticker_bollinger_lower_band,
        random_ticker_bollinger_upper_band
    ],
    labels=[
        'prices',
        'bollinger_lower_band',
        'bollinger_ipper_band',
    ],
    title=f"{random_ticker} prices",
    xlabel='dates',
    ylabel='prices'
)
```




In [18]:

```
random_ticker_bollinger_lower_band =random_ticker_data_frame['bollinger_lower_band']
random_ticker_bollinger_upper_band =random_ticker_data_frame['bollinger_upper_band']
random_ticker_rolling_mean_50 = random_ticker_data_frame['rolling_mean_50']
random_ticker_rolling_mean_100 = random_ticker_data_frame['rolling_mean_100']
strat.plot_multiple_series(
    x=random_ticker_dates,
    y_series_list=[
        random_ticker_prices,
    random_ticker_rolling_mean_50,
    random_ticker_rolling_mean_100,
    ],
    labels=[
        'prices',
        'rolling_mean_50',
        'rolling_mean_100',
    ],
    title=f"{random_ticker} prices",
    xlabel='dates',
    ylabel='$'
)
```

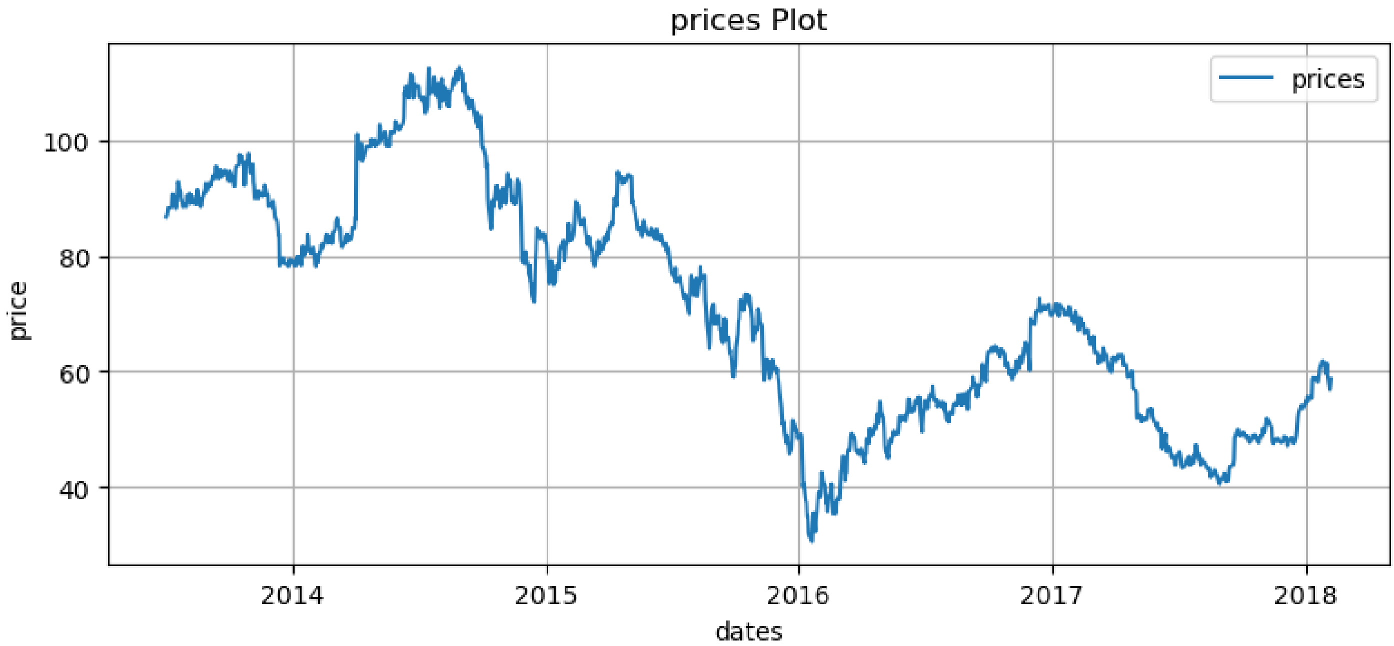




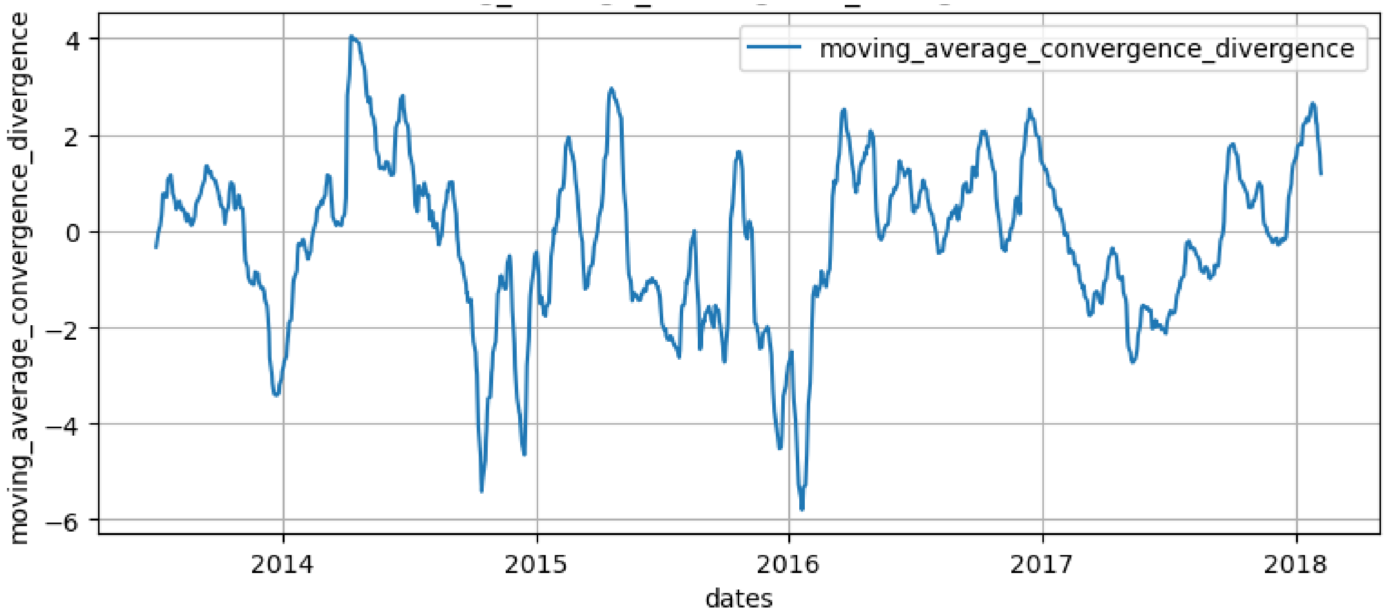

In [19]:

```
random_ticker_moving_average_convergence_divergence = (
    random_ticker_data_frame['moving_average_convergence_divergence']
)
random_ticker_momentum = random_ticker_data_frame['momentum']
random_ticker_volatility = random_ticker_data_frame['volatility']

strat.plot_multiple_series_separate_axes(
    x=random_ticker_dates,
    y_series_list=[
        random_ticker_prices,
        random_ticker_moving_average_convergence_divergence,
        random_ticker_momentum,
    random_ticker_volatility,
    ],
    names=[
        'prices',
        'moving_average_convergence_divergence',
    ],
    curve_labels=[
        'prices',
        'moving_average_convergence_divergence',
        'momentum',
        'volatility',

    ],
    title=f"{random_ticker} prices",
    xlabel='dates',
    y_labels=[
        'price',
        'moving_average_convergence_divergence',
    ]
)
```

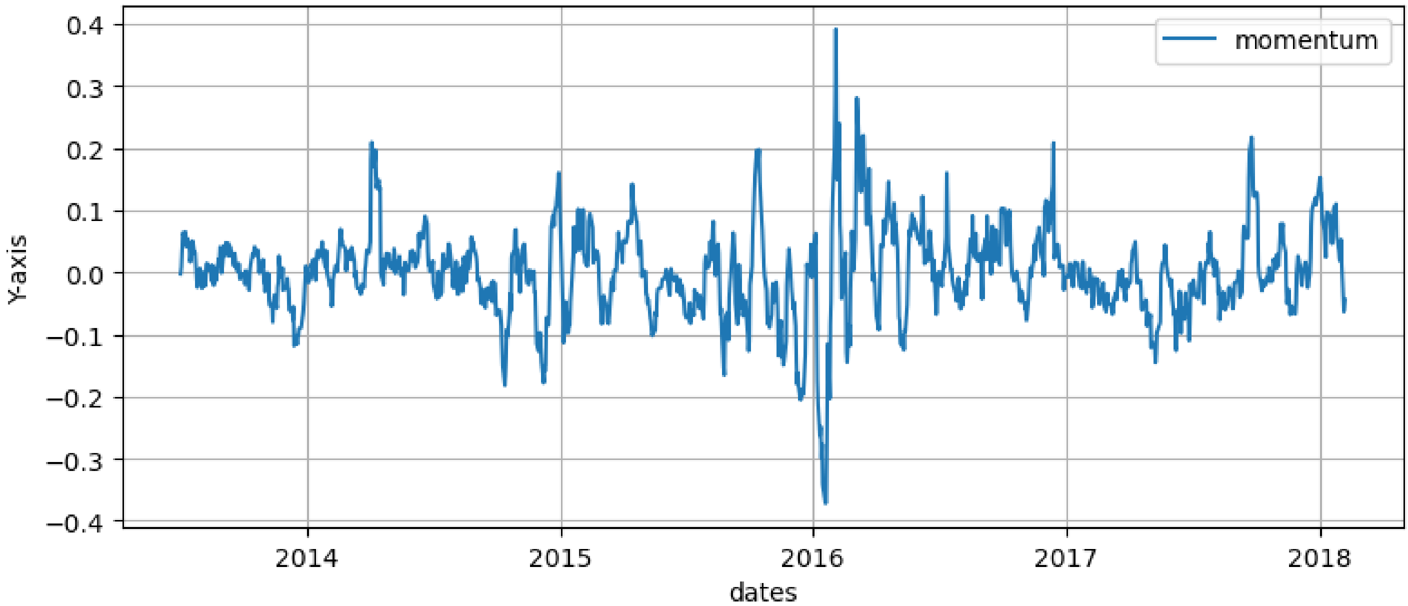
APC prices



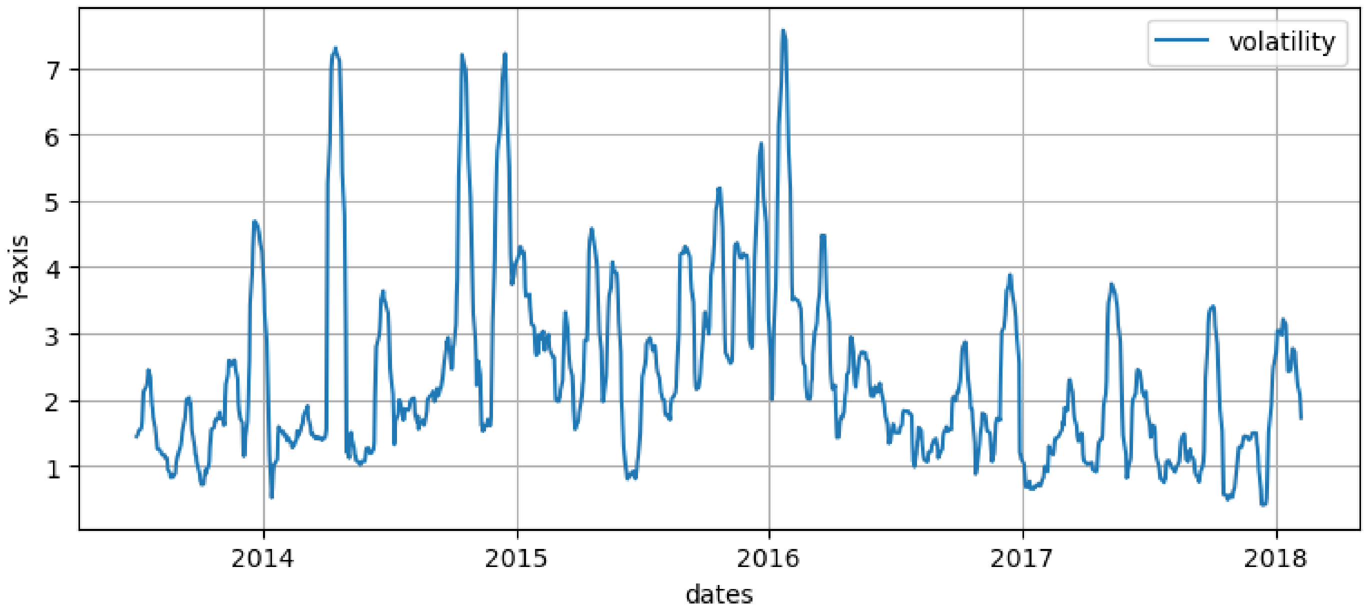

moving_average_convergence_divergence Plot




Series 3 Plot




Series 4 Plot




In [20]:

```
stocks_data_frame.head(3)
```

Out[20]:

| date | price | ticker | rolling_mean_20 | rolling_standard_deviations_20 | bollinger_lower_band | bollinger_upper_band | r |
|---|---|---|---|---|---|---|---|

| | date | price | ticker | rolling_mean_20 | rolling_standard_deviations_20 | bollinger_lower_band | bollinger_upper_band | r |
|---|---|---|---|---|---|---|---|---|
| 71710 | 2013-07-02 | 43.090000 | A | 43.694 | 0.969810 | 41.754381 | 45.633619 | |
| 71711 | 2013-07-03 | 43.169998 | A | 43.626 | 0.955683 | 41.714633 | 45.537367 | |
| 71712 | 2013-07-05 | 44.230000 | A | 43.587 | 0.911108 | 41.764785 | 45.409216 | |

In [21]:

```
# stocks_data_frame = stocks_data_frame.dropna()
```

In [22]:

```
stocks_data_frame['next_day_return'] = stocks_data_frame[constant.ColumnNames.price].shi
ft(-1) - stocks_data_frame[constant.ColumnNames.price]
stocks_data_frame[['price', 'next_day_return']].iloc[30:35]
```

Out[22]:

| | price | next_day_return |
|---|---|---|
| 71740 | 46.509998 | 0.450001 |
| 71741 | 46.959999 | -0.169998 |
| 71742 | 46.790001 | -0.350002 |
| 71743 | 46.439999 | 0.590000 |
| 71744 | 47.029999 | -0.139999 |

In [23]:

```
stocks_data_frame[['price']].iloc[30:35]
```

Out[23]:

| | price |
|---|---|
| 71740 | 46.509998 |
| 71741 | 46.959999 |
| 71742 | 46.790001 |
| 71743 | 46.439999 |
| 71744 | 47.029999 |

In [24]:

```
stocks_data_frame[constant.ColumnNames.price].shift(-1).iloc[30:35]
```

Out[24]:

```
71740    46.959999
71741    46.790001
71742    46.439999
71743    47.029999
71744    46.889999
Name: price, dtype: float32
```

In [25]:

```
stocks_data_frame['next_price_direction_signal'] = numpy.sign(stocks_data_frame['next_day
_return'])
target_column_name = 'next_price_direction_signal'
stocks_data_frame = stocks_data_frame[
        stocks_data_frame[target_column_name] != 0
]
```

In [26]:

```
# Remove NaN values caused by shifting
stocks_data_frame = stocks_data_frame.dropna()
```

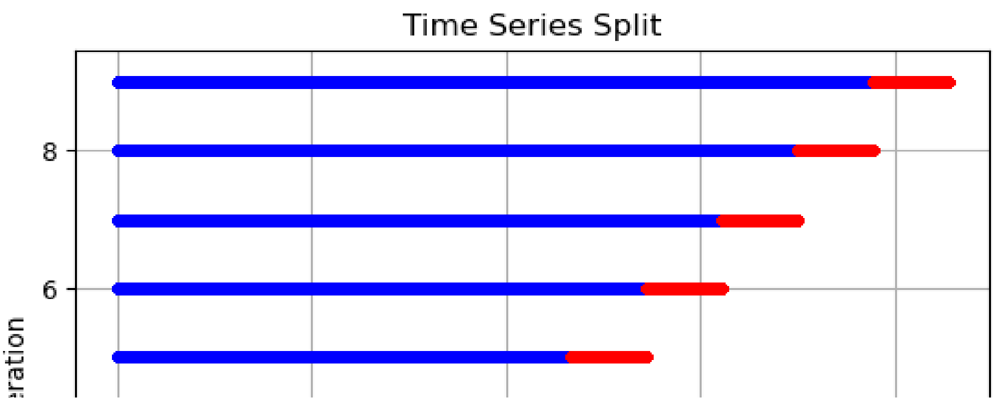
In [27]:

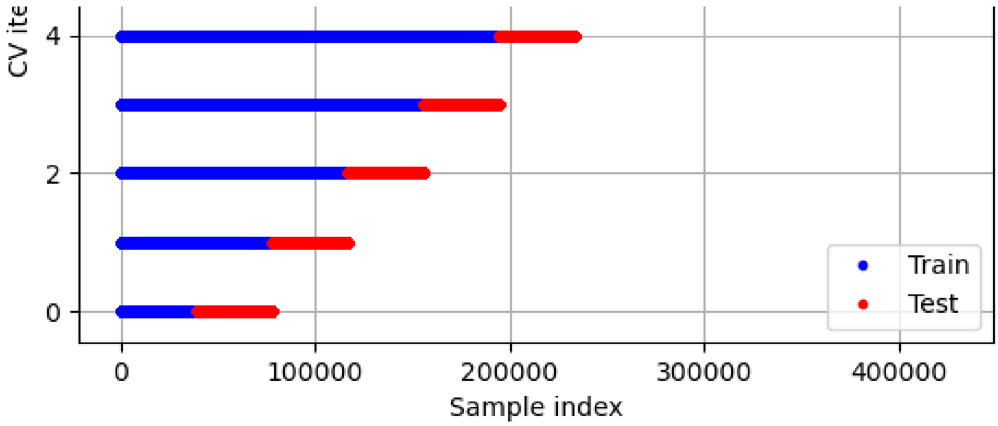
```
# Splitting the data into train and test sets based on date
nontest_data_frame = stocks_data_frame[stocks_data_frame[constant.ColumnNames.date] < con
stant.test_splitting_day_time_64]
test_data_frame = stocks_data_frame[stocks_data_frame[constant.ColumnNames.date] >= cons
tant.test_splitting_day_time_64]
print(f"Train dataset:\n{nontest_data_frame[[constant.ColumnNames.date]]}\n")
print(f"Test dataset:\n{test_data_frame[constant.ColumnNames.date]}\n")
```

```
Train dataset:
             date
71710  2013-07-02
71711  2013-07-03
71712  2013-07-05
71713  2013-07-08
71714  2013-07-09
...           ...
618758 2016-12-23
618759 2016-12-27
618760 2016-12-28
618761 2016-12-29
618762 2016-12-30

[427010 rows x 1 columns]

Test dataset:
72593    2017-01-03
72594    2017-01-04
72595    2017-01-05
72596    2017-01-06
72597    2017-01-09
            ...
619034   2018-01-31
619035   2018-02-01
619036   2018-02-02
619037   2018-02-05
619038   2018-02-06
Name: date, Length: 137390, dtype: datetime64[ns]
```

In [28]:

```
time_series_cross_validator = sklearn.model_selection.TimeSeriesSplit(n_splits=10)

strat.plot_time_series_split(
    time_series_cross_validator,
    nontest_data_frame[
        nontest_data_frame[target_column_name] != 0
    ][constant.ColumnNames.date],
    folder_path=constant.graph_folder_path ,
    file_name=constant.time_series_split_file_name,
)
```






In [29]:

```
nontest_data_frame[target_column_name]
```

Out[29]:

```
71710     1.0
71711     1.0
71712     1.0
71713     1.0
71714     1.0
         ...
618758   -1.0
618759   -1.0
618760    1.0
618761   -1.0
618762    1.0
Name: next_price_direction_signal, Length: 427010, dtype: float32
```

In [30]:

```
train_accuracies = []
validate_accuracies = []
train_aucs = []
validate_aucs = []
cross_validation_index = 0
best_validate_auc = 0
best_model = None

target_output_column_name = 'next_day_return'
pipeline = sklearn.pipeline.Pipeline([
    ('imputer', sklearn.impute.SimpleImputer(strategy='mean')),
    ('scaler', sklearn.preprocessing.StandardScaler()),
    ('boost', sklearn.ensemble.GradientBoostingClassifier()),
    # ('clf', sklearn.ensemble.RandomForestClassifier()),
    # ('clf', sklearn.ensemble.RandomForestClassifier(
    #     n_estimators=20,
    #     max_depth=30 ,
    #     min_samples_split=5,
    #     max_features='sqrt',
    #     class_weight='balanced',
    # )),
])

for train_index, validate_index in time_series_cross_validator.split(nontest_data_frame):
    print()
    input_train, input_validate = (
        nontest_data_frame[features_column_names].iloc[train_index],
        nontest_data_frame[features_column_names].iloc[validate_index],
    )
    target_output_train, target_output_validate = (
        nontest_data_frame[target_column_name].iloc[train_index],
        nontest_data_frame[target_column_name].iloc[validate_index],
    )

    pipeline.fit(input_train, target_output_train)
    train_predictions = pipeline.predict(input_train)
```

```
    validate_predictions = pipeline.predict(input_validate)

    print(f"{target_output_validate = }")
    print(f"{target_output_validate.shape = }")

    print(f"{train_predictions = }")
    print(f"{train_predictions.shape = }")

    print(f"{validate_predictions = }")
    print(f"{validate_predictions.shape = }")

    train_accuracy = sklearn.metrics.accuracy_score(target_output_train, train_predictio
ns)
    validate_accuracy = sklearn.metrics.accuracy_score(target_output_validate, validate_
predictions)

    print(f"{train_accuracy = }")
    print(f"{validate_accuracy = }")

    train_accuracies.append(train_accuracy)
    validate_accuracies.append(validate_accuracy)

    train_prediction_probabilities = pipeline.predict_proba(input_train)
    validate_prediction_probabilities = pipeline.predict_proba(input_validate)

    print(f"{train_prediction_probabilities = }")
    print(f"{train_prediction_probabilities.shape = }")

    print(f"{validate_prediction_probabilities = }")
    print(f"{validate_prediction_probabilities.shape = }")


    train_auc = sklearn.metrics.roc_auc_score(
        y_true=target_output_train,
        y_score=train_predictions,
        # multi_class='ovr',
    )
    validate_auc = sklearn.metrics.roc_auc_score(
        y_true=target_output_validate,
        y_score=validate_predictions,
        # multi_class='ovr',
    )

    print(f"{train_auc = }")
    print(f"{validate_auc = }")

    train_aucs.append(train_auc)
    validate_aucs.append(validate_auc)
    if best_model is None or best_validate_auc < best_validate_auc:
        best_model = pipeline

    print(f"Fold {cross_validation_index}: Train Accuracy: {train_accuracy}, Validation
Accuracy: {validate_accuracy}")
    print(f"Fold {cross_validation_index}: Train AUC: {train_auc}, Validation AUC: {vali
date_auc}")
    cross_validation_index += 1
```

```
target_output_validate = 54523    -1.0
54524     1.0
54525    -1.0
54526     1.0
54527     1.0
         ...
110557   -1.0
110558    1.0
110559   -1.0
110560   -1.0
110561   -1.0
Name: next_price_direction_signal, Length: 38819, dtype: float32
target_output_validate.shape = (38819,)
train_predictions = array([1., 1., 1., ..., 1., 1., 1.], dtype=float32)
train_predictions.shape = (38820,)
```

```
validate_predictions = array([ 1.,  1.,  1., ...,  1., -1.,  1.], dtype=float32)
validate_predictions.shape = (38819,)
train_accuracy = 0.5553838227717671
validate_accuracy = 0.5223473041551817
train_prediction_probabilities = array([[0.43724851, 0.56275149],
       [0.44896451, 0.55103549],
       [0.44896451, 0.55103549],
       ...,
       [0.44141807, 0.55858193],
       [0.44771134, 0.55228866],
       [0.42578248, 0.57421752]])
train_prediction_probabilities.shape = (38820, 2)
validate_prediction_probabilities = array([[0.44141807, 0.55858193],
       [0.44141807, 0.55858193],
       [0.44141807, 0.55858193],
       ...,
       [0.47968354, 0.52031646],
       [0.50018369, 0.49981631],
       [0.47886914, 0.52113086]])
validate_prediction_probabilities.shape = (38819, 2)
train_auc = 0.536076126761791
validate_auc = 0.5073626977375946
Fold 0: Train Accuracy: 0.5553838227717671, Validation Accuracy: 0.5223473041551817
Fold 0: Train AUC: 0.536076126761791, Validation AUC: 0.5073626977375946

target_output_validate = 110562   -1.0
110563   -1.0
110564   -1.0
110565   -1.0
110566   -1.0
          ...
170749   -1.0
170750    1.0
170751    1.0
170752   -1.0
170753    1.0
Name: next_price_direction_signal, Length: 38819, dtype: float32
target_output_validate.shape = (38819,)
train_predictions = array([ 1.,  1.,  1., ...,  1., -1.,  1.], dtype=float32)
train_predictions.shape = (77639,)
validate_predictions = array([ 1.,  1.,  1., ...,  1., -1.,  1.], dtype=float32)
validate_predictions.shape = (38819,)
train_accuracy = 0.5413774005332371
validate_accuracy = 0.52098199335377
train_prediction_probabilities = array([[0.44941805, 0.55058195],
       [0.4654908 , 0.5345092 ],
       [0.45635095, 0.54364905],
       ...,
       [0.49808533, 0.50191467],
       [0.51395835, 0.48604165],
       [0.47859454, 0.52140546]])
train_prediction_probabilities.shape = (77639, 2)
validate_prediction_probabilities = array([[0.47742332, 0.52257668],
       [0.45780337, 0.54219663],
       [0.45780337, 0.54219663],
       ...,
       [0.48839013, 0.51160987],
       [0.50051826, 0.49948174],
       [0.49706794, 0.50293206]])
validate_prediction_probabilities.shape = (38819, 2)
train_auc = 0.5227256938097126
validate_auc = 0.5052347725578004
Fold 1: Train Accuracy: 0.5413774005332371, Validation Accuracy: 0.52098199335377
Fold 1: Train AUC: 0.5227256938097126, Validation AUC: 0.5052347725578004

target_output_validate = 170754    1.0
170755   -1.0
170756    1.0
170757    1.0
170758   -1.0
          ...
223251   -1.0
```

```
223252   -1.0
223253   -1.0
223254    1.0
223255    1.0
Name: next_price_direction_signal, Length: 38819, dtype: float32
target_output_validate.shape = (38819,)
train_predictions = array([1., 1., 1., ..., 1., 1., 1.], dtype=float32)
train_predictions.shape = (116458,)
validate_predictions = array([1., 1., 1., ..., 1., 1., 1.], dtype=float32)
validate_predictions.shape = (38819,)
train_accuracy = 0.5351199574095382
validate_accuracy = 0.5213941626523094
train_prediction_probabilities = array([[0.45264831, 0.54735169],
       [0.48293401, 0.51706599],
       [0.46828242, 0.53171758],
       ...,
       [0.48989538, 0.51010462],
       [0.49934537, 0.50065463],
       [0.49139922, 0.50860078]])
train_prediction_probabilities.shape = (116458, 2)
validate_prediction_probabilities = array([[0.49139922, 0.50860078],
       [0.48545838, 0.51454162],
       [0.48545838, 0.51454162],
       ...,
       [0.48501766, 0.51498234],
       [0.48791512, 0.51208488],
       [0.48501766, 0.51498234]])
validate_prediction_probabilities.shape = (38819, 2)
train_auc = 0.5174740501983546
validate_auc = 0.5028802864045576
Fold 2: Train Accuracy: 0.5351199574095382, Validation Accuracy: 0.5213941626523094
Fold 2: Train AUC: 0.5174740501983546, Validation AUC: 0.5028802864045576

target_output_validate = 223256   -1.0
223257   -1.0
223258    1.0
223259   -1.0
223260   -1.0
           ...
281975    1.0
281976   -1.0
281977    1.0
281978   -1.0
281979   -1.0
Name: next_price_direction_signal, Length: 38819, dtype: float32
target_output_validate.shape = (38819,)
train_predictions = array([1., 1., 1., ..., 1., 1., 1.], dtype=float32)
train_predictions.shape = (155277,)
validate_predictions = array([1., 1., 1., ..., 1., 1., 1.], dtype=float32)
validate_predictions.shape = (38819,)
train_accuracy = 0.5333243171879931
validate_accuracy = 0.5213941626523094
train_prediction_probabilities = array([[0.46589431, 0.53410569],
       [0.46525899, 0.53474101],
       [0.47621941, 0.52378059],
       ...,
       [0.48122328, 0.51877672],
       [0.48122328, 0.51877672],
       [0.48122328, 0.51877672]])
train_prediction_probabilities.shape = (155277, 2)
validate_prediction_probabilities = array([[0.48122328, 0.51877672],
       [0.4719203 , 0.5280797 ],
       [0.47682136, 0.52317864],
       ...,
       [0.4481599 , 0.5518401 ],
       [0.45797604, 0.54202396],
       [0.47144178, 0.52855822]])
validate_prediction_probabilities.shape = (38819, 2)
train_auc = 0.5151124942094213
validate_auc = 0.5026951810417369
Fold 3: Train Accuracy: 0.5333243171879931, Validation Accuracy: 0.5213941626523094
Fold 3: Train AUC: 0.5151124942094213, Validation AUC: 0.5026951810417369
```

```
target_output_validate = 281981   -1.0
281982   -1.0
281983    1.0
281984    1.0
281985    1.0
         ...
356937   -1.0
356938   -1.0
356939   -1.0
356940    1.0
356941   -1.0
Name: next_price_direction_signal, Length: 38819, dtype: float32
target_output_validate.shape = (38819,)
train_predictions = array([1., 1., 1., ..., 1., 1., 1.], dtype=float32)
train_predictions.shape = (194096,)
validate_predictions = array([1., 1., 1., ..., 1., 1., 1.], dtype=float32)
validate_predictions.shape = (38819,)
train_accuracy = 0.530067595416701
validate_accuracy = 0.523841417862387
train_prediction_probabilities = array([[0.46820844, 0.53179156],
       [0.47053961, 0.52946039],
       [0.47441202, 0.52558798],
       ...,
       [0.45201716, 0.54798284],
       [0.46344953, 0.53655047],
       [0.47698774, 0.52301226]])
train_prediction_probabilities.shape = (194096, 2)
validate_prediction_probabilities = array([[0.45744149, 0.54255851],
       [0.46100096, 0.53899904],
       [0.46983437, 0.53016563],
       ...,
       [0.45117877, 0.54882123],
       [0.45345474, 0.54654526],
       [0.45605456, 0.54394544]])
validate_prediction_probabilities.shape = (38819, 2)
train_auc = 0.5113816403654342
validate_auc = 0.5064811652738608
Fold 4: Train Accuracy: 0.530067595416701, Validation Accuracy: 0.523841417862387
Fold 4: Train AUC: 0.5113816403654342, Validation AUC: 0.5064811652738608

target_output_validate = 356942    1.0
356943   -1.0
356944   -1.0
356945   -1.0
356946    1.0
         ...
391609    1.0
391610    1.0
391611    1.0
391612   -1.0
391613   -1.0
Name: next_price_direction_signal, Length: 38819, dtype: float32
target_output_validate.shape = (38819,)
train_predictions = array([1., 1., 1., ..., 1., 1., 1.], dtype=float32)
train_predictions.shape = (232915,)
validate_predictions = array([1., 1., 1., ..., 1., 1., 1.], dtype=float32)
validate_predictions.shape = (38819,)
train_accuracy = 0.5296052207886998
validate_accuracy = 0.5224503464798166
train_prediction_probabilities = array([[0.4593929 , 0.5406071 ],
       [0.4691946 , 0.5308054 ],
       [0.47781261, 0.52218739],
       ...,
       [0.44755187, 0.55244813],
       [0.45085257, 0.54914743],
       [0.45236417, 0.54763583]])
train_prediction_probabilities.shape = (232915, 2)
validate_prediction_probabilities = array([[0.45103664, 0.54896336],
       [0.44844462, 0.55155538],
       [0.44331855, 0.55668145],
       ...,
```

```
        [0.47797602, 0.52202398],
        [0.48533161, 0.51466839],
        [0.48760329, 0.51239671]])
validate_prediction_probabilities.shape = (38819, 2)
train_auc = 0.5116114372671305
validate_auc = 0.5014396252830978
Fold 5: Train Accuracy: 0.5296052207886998, Validation Accuracy: 0.5224503464798166
Fold 5: Train AUC: 0.5116114372671305, Validation AUC: 0.5014396252830978

target_output_validate = 391614   -1.0
391615    1.0
391616   -1.0
391617   -1.0
391618    1.0
         ...
451530    1.0
451531    1.0
451532   -1.0
451533    1.0
451534    1.0
Name: next_price_direction_signal, Length: 38819, dtype: float32
target_output_validate.shape = (38819,)
train_predictions = array([1., 1., 1., ..., 1., 1., 1.], dtype=float32)
train_predictions.shape = (271734,)
validate_predictions = array([1., 1., 1., ..., 1., 1., 1.], dtype=float32)
validate_predictions.shape = (38819,)
train_accuracy = 0.5296171991727204
validate_accuracy = 0.5195394008088823
train_prediction_probabilities = array([[0.46073899, 0.53926101],
        [0.46498295, 0.53501705],
        [0.47767234, 0.52232766],
        ...,
        [0.4809532 , 0.5190468 ],
        [0.48993145, 0.51006855],
        [0.48977505, 0.51022495]])
train_prediction_probabilities.shape = (271734, 2)
validate_prediction_probabilities = array([[0.48977505, 0.51022495],
        [0.48977505, 0.51022495],
        [0.4903832 , 0.5096168 ],
        ...,
        [0.46331632, 0.53668368],
        [0.45999472, 0.54000528],
        [0.46423772, 0.53576228]])
validate_prediction_probabilities.shape = (38819, 2)
train_auc = 0.510743733916181
validate_auc = 0.5021592035414781
Fold 6: Train Accuracy: 0.5296171991727204, Validation Accuracy: 0.5195394008088823
Fold 6: Train AUC: 0.510743733916181, Validation AUC: 0.5021592035414781

target_output_validate = 451535    1.0
451536    1.0
451537    1.0
451538    1.0
451539   -1.0
         ...
506311   -1.0
506312   -1.0
506313   -1.0
506314    1.0
506315   -1.0
Name: next_price_direction_signal, Length: 38819, dtype: float32
target_output_validate.shape = (38819,)
train_predictions = array([1., 1., 1., ..., 1., 1., 1.], dtype=float32)
train_predictions.shape = (310553,)
validate_predictions = array([1., 1., 1., ..., 1., 1., 1.], dtype=float32)
validate_predictions.shape = (38819,)
train_accuracy = 0.5284444201150851
validate_accuracy = 0.5229140369406734
train_prediction_probabilities = array([[0.46453599, 0.53546401],
        [0.46426983, 0.53573017],
        [0.47703458, 0.52296542],
        ...,
```

```
       [0.46545051, 0.53454949],
       [0.46200109, 0.53799891],
       [0.46358716, 0.53641284]])
train_prediction_probabilities.shape = (310553, 2)
validate_prediction_probabilities = array([[0.47791867, 0.52208133],
       [0.47791867, 0.52208133],
       [0.47516284, 0.52483716],
       ...,
       [0.46350653, 0.53649347],
       [0.4567977 , 0.5432023 ],
       [0.45984706, 0.54015294]])
validate_prediction_probabilities.shape = (38819, 2)
train_auc = 0.5093383042091786
validate_auc = 0.5050905709344585
Fold 7: Train Accuracy: 0.5284444201150851, Validation Accuracy: 0.5229140369406734
Fold 7: Train AUC: 0.5093383042091786, Validation AUC: 0.5050905709344585

target_output_validate = 506316   -1.0
506317    1.0
506318   -1.0
506319   -1.0
506320   -1.0
         ...
562700   -1.0
562701   -1.0
562702    1.0
562703    1.0
562704    1.0
Name: next_price_direction_signal, Length: 38819, dtype: float32
target_output_validate.shape = (38819,)
train_predictions = array([1., 1., 1., ..., 1., 1., 1.], dtype=float32)
train_predictions.shape = (349372,)
validate_predictions = array([1., 1., 1., ..., 1., 1., 1.], dtype=float32)
validate_predictions.shape = (38819,)
train_accuracy = 0.5282220670231158
validate_accuracy = 0.5247945593652593
train_prediction_probabilities = array([[0.46049971, 0.53950029],
       [0.46457487, 0.53542513],
       [0.47901448, 0.52098552],
       ...,
       [0.47028813, 0.52971187],
       [0.4598952 , 0.5401048 ],
       [0.46396961, 0.53603039]])
train_prediction_probabilities.shape = (349372, 2)
validate_prediction_probabilities = array([[0.49235167, 0.50764833],
       [0.49128116, 0.50871884],
       [0.49193019, 0.50806981],
       ...,
       [0.46049971, 0.53950029],
       [0.46049971, 0.53950029],
       [0.46049971, 0.53950029]])
validate_prediction_probabilities.shape = (38819, 2)
train_auc = 0.5089552593896394
validate_auc = 0.503170349436993
Fold 8: Train Accuracy: 0.5282220670231158, Validation Accuracy: 0.5247945593652593
Fold 8: Train AUC: 0.5089552593896394, Validation AUC: 0.503170349436993

target_output_validate = 562705   -1.0
562706    1.0
562707   -1.0
562708    1.0
562709    1.0
         ...
618758   -1.0
618759   -1.0
618760    1.0
618761   -1.0
618762    1.0
Name: next_price_direction_signal, Length: 38819, dtype: float32
target_output_validate.shape = (38819,)
train_predictions = array([1., 1., 1., ..., 1., 1., 1.], dtype=float32)
train_predictions.shape = (388191,)
```

```
validate_predictions = array([1., 1., 1., ..., 1., 1., 1.], dtype=float32)
validate_predictions.shape = (38819,)
train_accuracy = 0.5278097637503187
validate_accuracy = 0.5236868543754347
train_prediction_probabilities = array([[0.45928604, 0.54071396],
       [0.46618725, 0.53381275],
       [0.47854089, 0.52145911],
       ...,
       [0.45915152, 0.54084848],
       [0.45915152, 0.54084848],
       [0.45915152, 0.54084848]])
train_prediction_probabilities.shape = (388191, 2)
validate_prediction_probabilities = array([[0.47710682, 0.52289318],
       [0.45915152, 0.54084848],
       [0.46644072, 0.53355928],
       ...,
       [0.49041784, 0.50958216],
       [0.49041784, 0.50958216],
       [0.4821964 , 0.5178036 ]])
validate_prediction_probabilities.shape = (38819, 2)
train_auc = 0.5082314942526219
validate_auc = 0.5047064910588959
Fold 9: Train Accuracy: 0.5278097637503187, Validation Accuracy: 0.5236868543754347
Fold 9: Train AUC: 0.5082314942526219, Validation AUC: 0.5047064910588959
```

In [31]:

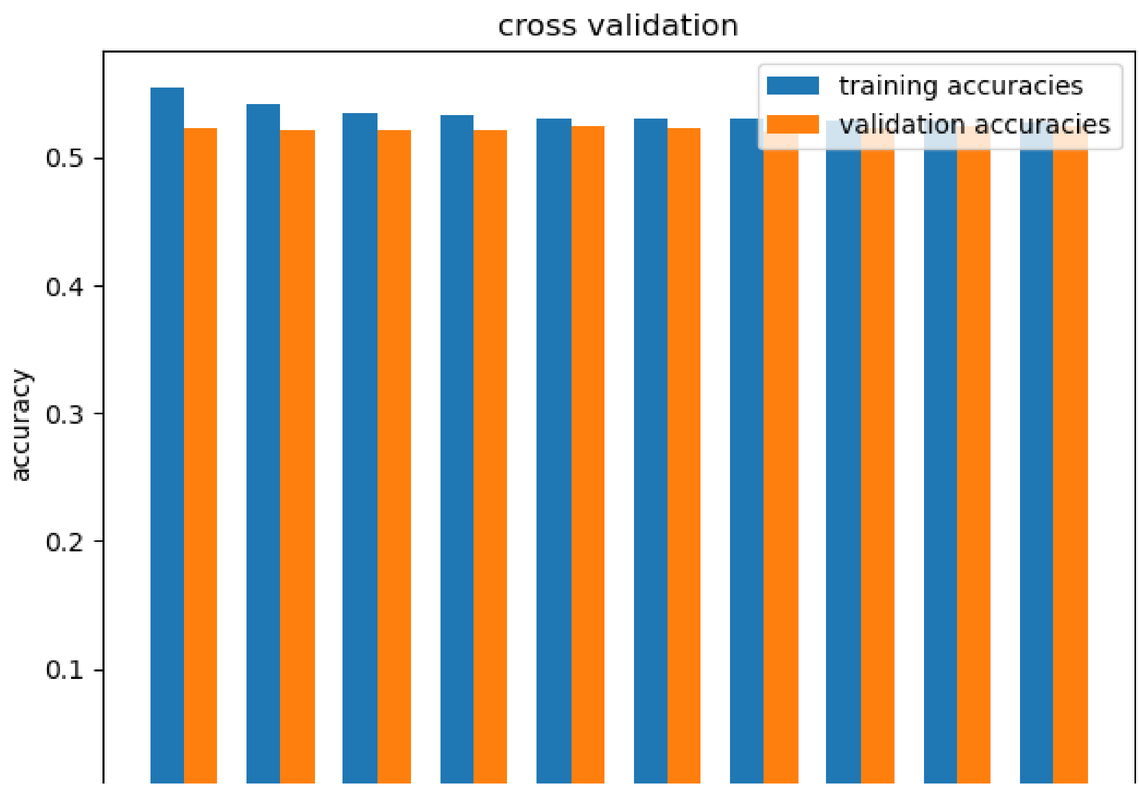
```
list(range(len(train_accuracies)))
```

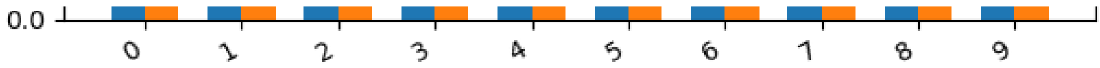
Out[31]:

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

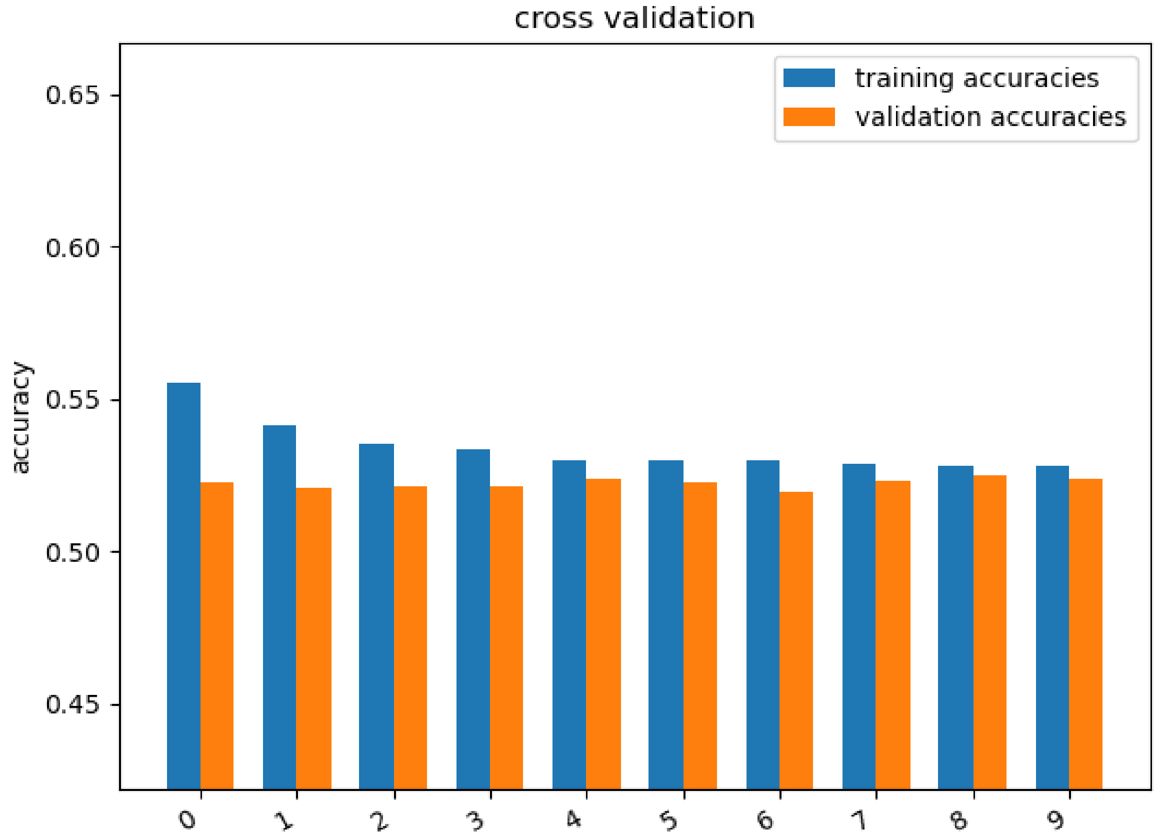
In [36]:

```
strat.create_grouped_bar_chart(
    group_labels=list(range(len(train_accuracies))),
    group1_data=train_accuracies,
    group2_data=validate_accuracies,
    group1_label='training accuracies',
    group2_label='validation accuracies',
    y_axis_label='accuracy',
    chart_title='cross validation',
    folder_path=constant.graph_folder_path,
    file_name='cross_validation_accuracies.png'
)
```





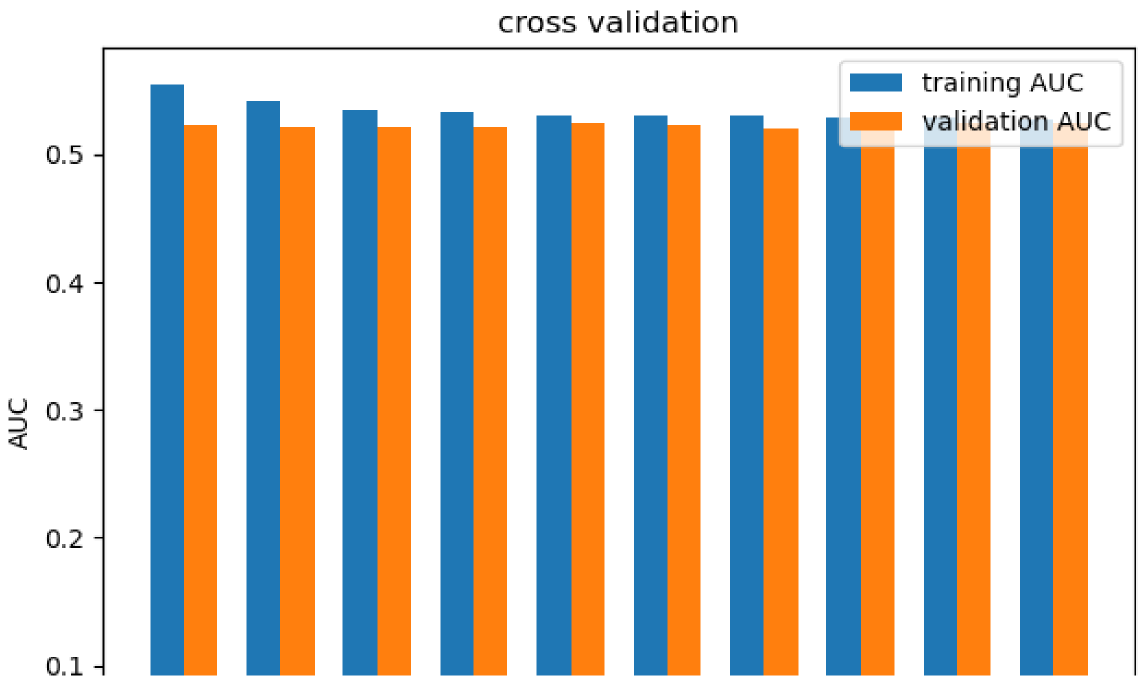

Out[36]:

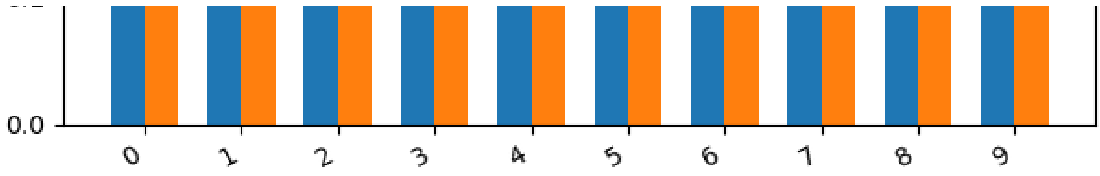



<Figure size 640x480 with 0 Axes>

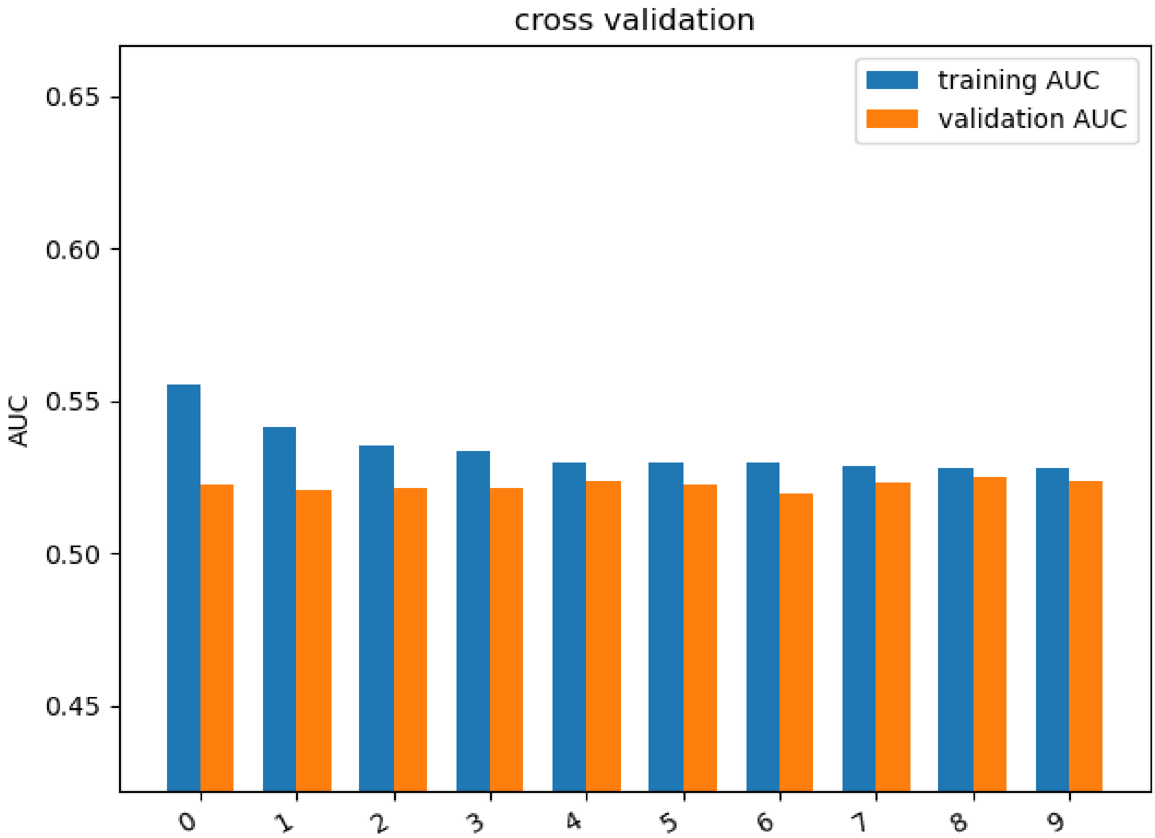
In [37]:

```
strat.create_grouped_bar_chart(
    group_labels=list(range(len(train_accuracies))),
    group1_data=train_accuracies,
    group2_data=validate_accuracies,
    group1_label='training AUC',
    group2_label='validation AUC',
    y_axis_label='AUC',
    chart_title='cross validation',
    folder_path=constant.graph_folder_path,
    file_name='cross_validation_aucs.png',
)
```






Out[37]:




<Figure size 640x480 with 0 Axes>

In [38]:

```
import pickle
import os

# Save the selected model
with open(os.path.join(constant.model_folder_path, 'gradient_boosting_classifier.pkl'), '
wb') as f:
    pickle.dump(best_model, f)
```

In [ ]:

```

```