Gradient Boosting for Cambridge UK Weather Forecasting

Gradient boosting models for time series analysis of Cambridge UK temperature measurements taken at the University computer lab weather station.

This notebook is being developed on Google Colab, using LightGBM and the Darts time series package. Initially I was most interested in short term temperature forecasts (less than 2 hours) but now mostly produce results up to 24 hours in the future for comparison with earlier baselines.

See my previous notebooks, web apps etc:

- Cambridge UK temperature forecast python notebooks
- Cambridge UK temperature forecast R models
- Bayesian optimisation of prophet temperature model
- Cambridge University Computer Laboratory weather station R shiny web app

The linked notebooks, web apps etc contain further details including:

- data description
- data cleaning and preparation
- data exploration

In particular, see the notebooks:

- cammet_baselines_2021 including persistent, simple exponential smoothing, Holt Winter's exponential smoothing and vector autoregression
- VAR_baseline updated VAR baseline used for model comparison
- keras_mlp_fcn_resnet_time_series, which uses a streamlined version of data preparation from Tensorflow time series forecasting tutorial
- lstm_time_series with stacked LSTMs, bidirectional LSTMs and ConvLSTM1D networks
- cnn_time_series with Conv1D, multi-head Conv1D, Conv2D and Inception-style models
- encoder_decoder which includes autoencoder with attention, encoder decoder with teacher forcing, transformer with teacher forcing and padding, encoder only with MultiHeadAttention
- feature_engineering solar-based and meteorology-based calculated features, rolling stats, tsfeatures, catch22, bivariate features and more
- tsfresh_feature_engineering automated feature engineering and selection for time series analysis of Cambridge UK weather measurements

Some of the above repositories, notebooks, web apps etc were built on both less data and less thoroughly cleaned data.

Table of Contents

TODO Add internal links before "final" commits

Some sections may get added/deleted during development. Don't want any broken links, so finish later.

- What Worked
- What Failed
- Rejected Ideas
- Future Work

Metadata

Gradient Boosted Model Building

Gradient boosted models train an ensemble of weak prediction models, which are usually decision trees. The ensemble model is iteratively built from weak learners and are added to produce a final strong learner. Successive weak learners focus more on the examples that previous weak learners got wrong. Historically, gradient boosted models have performed well in Kaggle competitions. The majority of M5 forecasting accuracy competition entries used gradient boosted trees, specifically lightGBM. See also, Forecasting with trees for more on the M5 competition and the dominance of boosted tree methods. However, tree-based models may be unable to learn a trend and forecast outside the bounds of the training data. This should not be a big problem with this cyclical data set.

Even simple lag-based feature engineering for multi-variate time series multi-step forecasting requires a considerable amount of effort, so a time series framework is beneficial. I had hoped to use the skforecast framework. Unfortunately, as of the current version (0.5.0) it does not support forecasts with multivariate data.

The sktime package supports boosting methods via their sklearn compatible make_reduction function. Initially it gave very poor results for the boosting methods I tried and the `make_reduction` function seemed to be somewhat of a second class option. In it's defence, it has built in support for tsfresh and catch22 feature extraction plus conformal interval functionality among many other interesting features.

The darts package is focused on time series forecasting but can also be used for filtering.

darts time series package:

- pros
  - supports multi-variate data
  - allows custom lag selection
  - future covariates which many of the other time series packages do not support
- cons
  - random forest (from sklearn) implementation is slow

The following are a few points I consider when building these gradient boosted models.

Forecast horizons:

- next 24 hours - 48 30 min steps ahead

Metrics:

- mse - mean squared error
  - mse used for loss function to avoid potential problems with infinite values from the square root function
  - rmse - root mean squared error is used for comparison with baselines
  - Huber loss may be worth exploring in the future if outliers remain an issue
- mae - median absolute error
- mape - mean absolute percentage error
  - Not used - mape fails when values, like temperature, become zero

Parameters to consider optimising:

- lags
- exogenous variables
  - time component representations
    - sinusoidal
    - periodic spline features
  - feature extraction
    - meterology-based features
    - solar-based features
    - rolling statistics - min, mean, max etc
    - catch22
    - tsfeatures
    - bivariate features - correlations, distances, kernels etc
- lightGBM hyperparameters

    - `n_estimators`
    - `learning_rate`
    - `feature_fraction`
    - `bagging_fraction`
    - `bagging_freq`
    - `min_child_samples`

Models considered:

- darts lightGBM
    - lightGBM
    - from Microsoft
    - annecdotally, seems quite fast
    - 1 model for each forecast step
    - feature selection
        - using Boruta-style shadow variables
    - lag selection
        1. using all available future covariate lags
        2. setting target lags and past covariate lags to be equal
        3. using every nth target/past lag to reduce compute time
            - where n is 2, 3, 4, 6

Code Setup

Install darts Library

Install darts because it is currently not available on google colab.

WARNING: You may need to restart the google colab runtime after this install.

```
#!pip install darts
# ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviour i
# tensorflow 2.8.2+zzzcolab20220719082949 requires tensorboard<2.9,>=2.8, but you have tensorboard 2.10.1 which is incompatibl

#!pip install "u8darts[torch]"
# ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviour i
# tensorflow 2.8.2+zzzcolab20220719082949 requires tensorboard<2.9,>=2.8, but you have tensorboard 2.10.1 which is incompatibl

#!pip install "u8darts[pmdarima]"

!pip install "u8darts[notorch]"
#!pip install "darts"
#!pip install "u8darts[all]"  # Slow - can take 15 mins (installs numerous nvidia packages)
```

```
  Collecting fugue-sql-antlr>=0.1.6 (from fugue>=0.8.1->statsforecast>=1.4->u8darts[notorch])
    Downloading fugue-sql-antlr-0.2.0.tar.gz (154 kB)
       ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 154.7/154.7 kB 14.6 MB/s eta 0:00:00
    Preparing metadata (setup.py) ... done
  Requirement already satisfied: sqlglot in /usr/local/lib/python3.10/dist-packages (from fugue>=0.8.1->statsforecast>=1.4-
  Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from fugue>=0.8.1->statsforecast>=1.4->
  Requirement already satisfied: llvmlite<0.42,>=0.41.0dev0 in /usr/local/lib/python3.10/dist-packages (from numba>=0.51->p
  Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from plotly->catboost>=1.0.6->
  Collecting antlr4-python3-runtime<4.12 (from fugue-sql-antlr>=0.1.6->fugue>=0.8.1->statsforecast>=1.4->u8darts[notorch])
    Downloading antlr4_python3_runtime-4.11.1-py3-none-any.whl (144 kB)
       ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 144.2/144.2 kB 16.3 MB/s eta 0:00:00
  Requirement already satisfied: pyarrow>=6.0.1 in /usr/local/lib/python3.10/dist-packages (from triad>=0.9.3->fugue>=0.8.1
  Requirement already satisfied: fsspec>=2022.5.0 in /usr/local/lib/python3.10/dist-packages (from triad>=0.9.3->fugue>=0.8
  Collecting fs (from triad>=0.9.3->fugue>=0.8.1->statsforecast>=1.4->u8darts[notorch])
    Downloading fs-2.4.16-py2.py3-none-any.whl (135 kB)
       ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 135.3/135.3 kB 10.3 MB/s eta 0:00:00
  Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->fugue>=0.8.1->sta
  Requirement already satisfied: appdirs~=1.4.3 in /usr/local/lib/python3.10/dist-packages (from fs->triad>=0.9.3->fugue>=(
  Building wheels for collected packages: pyod, fugue-sql-antlr
    Building wheel for pyod (setup.py) ... done
    Created wheel for pyod: filename=pyod-1.1.3-py3-none-any.whl size=190250 sha256=7b1ddbaa46106456983de3f32fa7877b6637f8c
    Stored in directory: /root/.cache/pip/wheels/05/f8/db/124d43bec122d6ec0ab3713fadfe25ebed8af52ec561682b4e
    Building wheel for fugue-sql-antlr (setup.py) ... done
    Created wheel for fugue-sql-antlr: filename=fugue_sql_antlr-0.2.0-py3-none-any.whl size=158196 sha256=24900c966c9d68d2b
    Stored in directory: /root/.cache/pip/wheels/5a/b5/4e/216953a1c711da55de29ed7ecf158b4a5bf32ef93d69ad66dd
  Successfully built pyod fugue-sql-antlr
  Installing collected packages: antlr4-python3-runtime, slicer, fs, coreforecast, utilsforecast, triad, shap, pyod, nfours
  Successfully installed adagio-0.2.4 antlr4-python3-runtime-4.11.1 catboost-1.2.3 coreforecast-0.0.8 fs-2.4.16 fugue-0.8.7
```

Load Libraries

Load most of the required packages.

```
import re
import sys
import math
import timeit
import datetime
import itertools
import subprocess
import pkg_resources

import numpy as np
import pandas as pd
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import seaborn as sns
import lightgbm as lgb
from tqdm import tqdm
from scipy import stats, special
from itertools import product
import statsmodels.api as sm
from statsmodels.tsa.stattools import acf, pacf
from statsmodels.nonparametric.smoothers_lowess import lowess
from sklearn.utils import check_X_y
from sklearn.metrics import r2_score
from sklearn.preprocessing import MinMaxScaler
from sklearn.feature_selection import f_regression, mutual_info_regression, r_regression


from darts import TimeSeries
from darts.models.forecasting.lgbm import LightGBMModel


# Reduces variance in results but won't eliminate it :-(
%env PYTHONHASHSEED=0
import random

# set seed to make all processes deterministic
seed = 0
random.seed(seed)
np.random.seed(seed)


%matplotlib inline

# Prevent lightGBM 'Converting column-vector to 1d array' warning
import warnings
warnings.filterwarnings(
    action   = 'ignore',
    category = UserWarning,
    module   = r'.*lightgbm'
)
```

'

```
from google.colab import drive
drive.mount('/content/drive')
```

```
WARNING:darts.models:Support for Torch based models not available. To enable them, install "darts", "u8darts[torch]" or '
env: PYTHONHASHSEED=0
Mounted at /content/drive
```

Environment Variables

Set some environment variables:

```
HORIZON = 48
Y_COL   = 'y_des' # 'y_des_fft' 'y_res' 'y'
FUT_COV = ['irradiance', 'za_rad']


# CORE_FEATS = [Y_COL, 'dew.point_des', 'humidity', 'pressure']
# FUT_FEATS = ['irradiance', 'za_rad', 'azimuth_cos']
# DAY  = 24 * 60 * 60
# YEAR = 365.2425 * DAY
# DAILY_OBS  = 48
# YEARLY_OBS = int(365.2425 * DAILY_OBS)  # annual observations
# DAY_SECS_STEP = int(DAY / DAILY_OBS)
# VALID_YEAR = 2021
# TEST_YEAR  = 2022
```

Custom Functions

Next, define some utility functions:

- _check_obs_preds_lens_eq
- rmse_
- mse_
- mae_
- summarise_backtest
- print_rmse_mae
- drop_cols_correlated_with_feat_cols
- drop_problem_cols
- summarise_historic_comparison
- plot_lagged_feat_imp_subplot
- plot_highlighted_lagged_feat_imp_subset
- get_pastcov_features
- get_pastcov_lags
- plot_lagged_feature_importances
- plot_feature_importances
- get_feature_importances
- expand_grid
- keep_key
- get_historic_comparison
- plot_one_step_abs_err_boxplot
- plot_one_step_residuals_dist
- plot_one_step_residuals
- plot_one_step_obs_preds_dists
- plot_one_step_obs_vs_preds
- plot_one_step_diagnostics
- plot_one_step_residuals_qq
- plot_one_step_residuals_acf
- _plot_xy_for_label
- plot_multistep_obs_vs_preds
- plot_multistep_obs_vs_mean_preds_by_step
- plot_multistep_obs_preds_dists
- plot_multistep_residuals
- plot_multistep_residuals_dist
- plot_multistep_residuals_vs_predicted
- se_

- metric_ci_vals
- plot_horizon_metrics
- plot_horizon_metrics_boxplots
- plot_multistep_diagnostics
- _filter_out_missing
- plot_multistep_forecast_examples
- get_rmse_mae_from_backtest
- plot_lgb_learning_curve
- plot_learning_curves
- get_main_plot_title
- build_two_lgbm_models
- drop_correlated_cols
- plot_observation_examples
- sanity_check_df_rows_cols_labels
- sanity_check_before_after_dfs
- compare_train_valid_test_sanity_dfs
- sanity_check_train_valid_test
- print_train_valid_test_shapes
- plot_feature_history_single_df
- plot_feature_history
- plot_feature_history_separately
- check_high_low_thresholds
- get_features_filename
- merge_data_and_aggs
- print_null_columns
- print_na_locations
- get_darts_series
- plot_short_term_acf
- plot_long_term_acf
- print_df_summary
- get_approx_overlap
- add_transform_column
- add_feature_column
- add_shadow_imp_column
- add_feature_window_transform_column
- add_feature_transform_column
- get_multi_model_feat_imps
- get_above_between_below_feats_with_model_lag
- plot_multi_model_feat_imps
- plot_x_y_importance_interaction
- plot_multi_model_importance_interactions
- groupby_multi_model_feat_imps
- plot_feat_imp_cumsum
- plot_multi_model_single_feature_imp
- load_features_file
- load_train_valid_test_features
- get_feature_selection_data
- get_feature_selection_scores
- get_above_between_below_features
- get_multi_step_feat_sel_scores
- groupby_multi_step_feat_sel_scores
- plot_multi_step_feat_sel_scores
- summarise_multi_step_feat_sel_scores
- summarise_multi_model_feat_imps

```
def _check_obs_preds_lens_eq(obs, preds):
    obs_preds_lens_eq = 1

    if len(obs) != len(preds):
        print("obs:  ", len(obs))
        print("preds:", len(preds))
        obs_preds_lens_eq = 0
```

```
    return obs_preds_lens_eq


def rmse_(obs, preds):
    if _check_obs_preds_lens_eq(obs, preds) == 0:
        stop()
    else:
        return np.sqrt(np.mean((obs - preds) ** 2))


def mse_(obs, preds):
    if _check_obs_preds_lens_eq(obs, preds) == 0:
        stop()
    else:
        return np.mean((obs - preds) ** 2)


def mae_(obs, preds):
    "mean absolute error - equivalent to the keras loss function"
    if _check_obs_preds_lens_eq(obs, preds) == 0:
        stop()
    else:
        return np.mean(np.abs(obs - preds))      # keras loss
        # return np.median(np.abs(obs - preds))  # earlier baselines


# TODO Remove me?
def summarise_backtest(backtest, df, horizon = HORIZON, digits = 6, y_col = Y_COL):

    if len(backtest[0]) == 1:
        print("\n# Backtest RMSE:", round(rmse_(val_ser[-len(backtest):].values(), backtest.values()), digits))
        print("# Backtest MAE: ",   round( mae_(val_ser[-len(backtest):].values(), backtest.values()), digits))

        print("\nbacktest[", y_col, "]:\n", sep='')
        backtest_stats = stats.describe(backtest[y_col].values())
        print("count\t", backtest_stats[0])
        print("mean\t",  round(backtest_stats[2][0], digits))
        print("std\t",   round(np.sqrt(backtest_stats[3][0]), digits))
        print("min\t",   round(np.min(backtest_stats[1]), digits))
        print("25%\t",   round(np.percentile(backtest[y_col].values(), 25), digits))
        print("50%\t",   round(np.median(backtest[y_col].values()), digits))
        print("75%\t",   round(np.percentile(backtest[y_col].values(), 75), digits))
        print("max\t",   round(np.max(backtest_stats[1]), digits))
    elif len(backtest[0]) == horizon:
        preds_df = pd.concat([backtest[i].pd_dataframe() for i in range(len(backtest))], axis=0)
        trues_df = df.loc[preds_df.index, [y_col]]
        hist_comp = pd.concat([trues_df, preds_df[y_col]], axis = 1)
        hist_comp.columns = [y_col, 'pred']
        list_int = [i for i in range(1, horizon + 1)]
        reps = len(hist_comp) // len(list_int)
        hist_comp['step'] = np.tile(list_int, reps)

        print("\nBacktest RMSE all:", round(rmse_(hist_comp[y_col], hist_comp['pred']), digits))
        print("Backtest MAE all: ",    round(mae_(hist_comp[y_col], hist_comp['pred']), digits))

        print("\n# Backtest RMSE 48th:", round(rmse_(hist_comp.loc[hist_comp['step'] == horizon, y_col], \
                                                   hist_comp.loc[hist_comp['step'] == horizon, 'pred']), digits))
        print("# Backtest MAE 48th: ",    round(mae_(hist_comp.loc[hist_comp['step'] == horizon, y_col], \
                                                   hist_comp.loc[hist_comp['step'] == horizon, 'pred']), digits))

        lasttest_stats = stats.describe(hist_comp['pred'])
        print("\nbacktest[", y_col, "]:\n", sep='')
        print("count\t", len(hist_comp['pred']))
        print("mean\t",  round(lasttest_stats[2], digits))
        print("std\t",   round(np.sqrt(lasttest_stats[3]), digits))
        print("min\t",   round(np.min(lasttest_stats[1]), digits))
        print("25%\t",   round(np.percentile(hist_comp['pred'], 25), digits))
        print("50%\t",   round(np.median(hist_comp['pred']), digits))
        print("75%\t",   round(np.percentile(hist_comp['pred'], 75), digits))
        print("max\t",   round(np.max(lasttest_stats[1]), digits))


def print_rmse_mae(obs, preds, postfix_str, prefix_str = '', digits = 6):
    print(prefix_str, "Backtest RMSE ", postfix_str, ": ",
          round(rmse_(obs, preds), digits),
          sep='')
    print(prefix_str, "Backtest MAE ",  postfix_str, ":  ",
          round( mae_(obs, preds), digits),
          sep='')
    print()
```

```
def drop_cols_correlated_with_feat_cols(df, feats_df, threshold=0.95):

  for feat_col in feats_df.columns:
    corrs = df.corrwith(feats_df[feat_col])
    drop_cols = corrs[(corrs > threshold) & (corrs != 1.0)]

    for i in range(len(drop_cols)):
      drop_col = drop_cols.index[i]
      if drop_col in df.columns:  # and drop_col not in feats_df.columns:
        del df[drop_col]

  return df


def drop_problem_cols(df, lag, drop_cor=True,
                      var_cutoff=0.05, cor_cutoff=0.95, na_cutoff=0.05,
                      verbose = False):

  if verbose:
    print('drop_problem_cols - start:', df.shape)


  # drop all NA columns
  df = df.dropna(axis = 1, how = 'all')

  if verbose:
    print('drop_problem_cols - after dropna:', df.shape)


  # drop single value columns
  df = df.loc[:, (df != df.iloc[lag]).any()]

  if verbose:
    print('drop_problem_cols - after drop single value cols:', df.shape)


  # drop low variance columns
  if 'ds' in df.columns:
    df = df.drop(['ds'], axis=1)
    df = df.loc[:, df.std() > var_cutoff]
    df['ds'] = df.index
  else:
    df = df.loc[:, df.std() > var_cutoff]

  if verbose:
    print('drop_problem_cols - after drop low var cols:', df.shape)


  # drop highly correlated columns
  if drop_cor:
    df = drop_correlated_cols(df, cor_cutoff)

  if verbose:
    print('drop_problem_cols - after drop correlated cols:', df.shape)


  # drop cols with high % of NA values
  pc_thresh = int(na_cutoff * df.shape[0])
  print('columns with null values:')
  display(df.isnull().sum())
  df = df.loc[:, df.isnull().sum() < pc_thresh]

  if verbose:
    print('drop_problem_cols - after drop high % of NAs:', df.shape)

  return df


def summarise_historic_comparison(hc, df, horizon = HORIZON,
                                  digits = 6,
                                  y_col = Y_COL,
                                  df_name = 'valid_df'):

    print('\n')
    print_rmse_mae(hc[y_col], hc['pred'], 'all')

    obs   = hc.loc[hc['step'] == horizon, y_col]
    preds = hc.loc[hc['step'] == horizon, 'pred']
    if horizon == 1:
      post_str = '1st'
    elif horizon == 2:
      post_str = '2nd'
```

```
    elif horizon == 3:
      post_str = '3rd'
    else:
      post_str = str(horizon) + 'th'
    print_rmse_mae(obs, preds, post_str, '# ')

    obs   = hc.loc[hc['missing'] == 0.0, y_col]
    preds = hc.loc[hc['missing'] == 0.0, 'pred']
    print_rmse_mae(obs, preds, 'miss==0')

    obs   = hc.loc[hc['missing'] == 1.0, y_col]
    preds = hc.loc[hc['missing'] == 1.0, 'pred']
    print_rmse_mae(obs, preds, 'miss==1')

    if y_col == 'y_des':
      preds = hc['pred'] - hc['y_yearly'] - hc['y_daily'] - hc['y_trend']
    elif y_col == 'y_des_fft':
      preds = hc['pred'] - hc['y_fft']
    elif y_col == 'y':
      preds = hc['pred']
    elif y_col == 'y_res':
      preds = hc['pred'] - hc['y_yearly'] - hc['y_daily']

    preds.dropna(inplace=True)
    lasttest_stats = stats.describe(preds)
    print("\nbacktest['", y_col, "']:", sep='')
    print("count\t", len(preds))
    print("mean\t",  round(lasttest_stats[2], digits))
    print("std\t",   round(np.sqrt(lasttest_stats[3]), digits))
    print("min\t",   round(np.min(lasttest_stats[1]), digits))
    print("25%\t",   round(np.percentile(preds, 25), digits))
    print("50%\t",   round(np.median(preds), digits))
    print("75%\t",   round(np.percentile(preds, 75), digits))
    print("max\t",   round(np.max(lasttest_stats[1]), digits))

    print("\n", df_name, "['", y_col, "']:\n", df[y_col].describe(), '\n', sep='')


def plot_lagged_feat_imp_subplot(fi_df, subset):
    bar_height = 0.25
    title = 'Feature importance'

    fi_max = fi_df['importance'].max()
    if fi_max > 100:
      xl_max = int(np.ceil(fi_max / 100.0)) * 100
    elif fi_max > 10:
      xl_max = int(np.ceil(fi_max / 10.0)) * 10
    else:
      xl_max = fi_max

    if subset is not None:
      fi_df = fi_df.loc[fi_df['feature'].str.contains(subset, regex=True), :]
      title += ' - ' + subset + ' features'
      plt.figure(figsize=(7, 3))
    else:
      title += ' - all features'
      plt.figure(figsize=(20, 10))

    plt.xlim(0, xl_max)
    plt.barh(width  = fi_df['importance'],
             y      = fi_df['feature'],
             height = bar_height)


    def plot_highlighted_lagged_feat_imp_subset(data, subset_str, hl_col, bar_height = 0.25):
        data_subset = data.loc[data['feature'].str.contains(subset_str, regex=True), :]

        plt.barh(width  = data_subset['importance'],
                 y      = data_subset['feature'],
                 height = bar_height,
                 color  = hl_col)


    plot_highlighted_lagged_feat_imp_subset(fi_df, 'shadow', 'red')

    yregex = '^' + Y_COL
    ytarg_str = yregex + '_target_'
    plot_highlighted_lagged_feat_imp_subset(fi_df, ytarg_str, 'green')

    ypcov_str = yregex + '_pastcov_'
    plot_highlighted_lagged_feat_imp_subset(fi_df, ypcov_str, 'blue')
```

```
    plt.title(title)
    plt.show()


def get_pastcov_features(fi_df):
    pcov_feats_long = fi_df.loc[fi_df['feature'].str.contains('_pastcov_'), 'feature'].to_list()

    r = re.compile('_pastcov_.*$')

    pcov_feats_dups = [r.sub('', pcov_feat_long) for pcov_feat_long in pcov_feats_long]
    pcov_feats = list(set(pcov_feats_dups))

    if Y_COL in pcov_feats:
      pcov_feats.remove(Y_COL)

    return pcov_feats


def get_pastcov_lags(fi_df):
    pcov_feats_long = fi_df.loc[fi_df['feature'].str.contains('_pastcov_'), 'feature'].to_list()

    r = re.compile('^.*_pastcov_')

    pcov_lags_dups = [r.sub('', pcov_feat_long) for pcov_feat_long in pcov_feats_long]
    pcov_lags = list(set(pcov_lags_dups))

    return pcov_lags


# TODO: Also, consider combining plot_feature_importances and
#       plot_lagged_feature_importances into a single function
def plot_lagged_feature_importances(model):
    '''Plot feature importance for models with multiple lags

    Should be easier to compare importance across features, lags, targets
    and past covariates

    No support for future covariates

    Use plot_feature_importances function for lags = 1, past_cov_lags = 1 models

    '''

    imp_thresh = 0
    if isinstance(model, CatBoostModel):
      imp_thresh = 0  # some catboost importance values below 1
    elif isinstance(model, LightGBMModel):
      imp_thresh = 1

    imp_df = pd.DataFrame({'feature':    model.lagged_feature_names,
                           'importance': model.model.feature_importances_})
    imp_df = imp_df.sort_values('importance')

    plot_lagged_feat_imp_subplot(imp_df, None)
    plot_lagged_feat_imp_subplot(imp_df, '^' + Y_COL)

    pcov_feats = get_pastcov_features(imp_df)
    for pcov_feat in pcov_feats:
        plot_lagged_feat_imp_subplot(imp_df, '^' + pcov_feat)

    pcov_lags = get_pastcov_lags(imp_df)
    for pcov_lag in pcov_lags:
        plot_lagged_feat_imp_subplot(imp_df, '_pastcov_' + pcov_lag + '$')


# TODO: Consider combining plot_feature_importances and
#       plot_lagged_feature_importances into a single function
def plot_feature_importances(model, \
                             y_col        = Y_COL, \
                             include_cols = None,  \
                             exclude_cols = None):
    '''Plot feature importances from lightGBM models

    WARNING: Only works with lags = 1 and lags_past_cov = 1
             Use plot_lagged_feature_importances for models with additional lags
    '''

    imp_thresh = 0
    if isinstance(model, CatBoostModel):
      imp_thresh = 0  # some catboost importance values below 1
    elif isinstance(model, LightGBMModel):
      imp_thresh = 1
```

```
    if include_cols is not None:
        col_names = include_cols
    else:
        col_names = model.lagged_feature_names

    cols_df = pd.DataFrame({'feature': col_names,
                            'importance': model.model.feature_importances_})
    cols_df = cols_df.sort_values('importance')
    cols_df = cols_df[cols_df.importance >= imp_thresh]

    if exclude_cols is not None:
      cols_df = cols_df[~cols_df['feature'].isin(exclude_cols)]

    plt.figure(figsize=(20, 10))
    plt.barh(width  = cols_df['importance'],
             y      = cols_df['feature'],
             height = 0.25);
    plt.barh(width  = cols_df.loc[cols_df['feature'].str.contains('shadow'),
                                  'importance'],
             y      = cols_df.loc[cols_df['feature'].str.contains('shadow'),
                                  'feature'],
             height = 0.25,
             color  = 'red')
    plt.title('Feature importance\nimportance threshold = ' + str(imp_thresh))
    plt.show()


def get_feature_importances(model,
                            y_col = Y_COL,
                            imp_thresh   = None,
                            include_cols = None,
                            exclude_cols = None,
                            verbose      = True):

    # imp_thresh = 0
    if imp_thresh is None and isinstance(model, CatBoostModel):
      imp_thresh = 0  # some catboost importance values below 1
    elif imp_thresh is None and isinstance(model, LightGBMModel):
      imp_thresh = 1

    if include_cols is not None:
        col_names = include_cols
    else:
        col_names = model.lagged_feature_names

    cols_df = pd.DataFrame({'feature': col_names,
                            'importance': model.model.feature_importances_})
    cols_df = cols_df.sort_values('importance')
    cols_df = cols_df[cols_df.importance >= imp_thresh]

    if exclude_cols is not None:
      cols_df = cols_df[~cols_df['feature'].isin(exclude_cols)]

    shadow_str = '_shadow_'
    if shadow_str in ''.join(cols_df['feature'].values):
        shad_thresh = cols_df.loc[cols_df['feature'].str.contains(shadow_str),
                                  'importance'].tail(1).values[0]
    else:
        shad_thresh = 0.0

    cols_df = cols_df[cols_df['importance'] > shad_thresh]

    if verbose:
      print(cols_df.to_string(index=False), sep='\n')

    inc_cols = []
    for feature in cols_df['feature'].values:
        inc_cols.append(re.sub('_(pastcov|target|futcov)_lag.*', '', feature))

    # remove duplicates from a list, while preserving order
    seen = set()
    inc_cols = [col for col in inc_cols if col not in seen and not seen.add(col)]

    if verbose:
      print('\ninc_cols:', inc_cols)

    return inc_cols


def expand_grid(dictionary):
   return pd.DataFrame([row for row in product(*dictionary.values())],
```

```
                        columns = dictionary.keys())


def keep_key(d, k):
  """ models = keep_key(models, 'datasets') """
  return {k: d[k]}


def get_historic_comparison(backtest, df, y_col = Y_COL, horizon = HORIZON):
    if horizon > 1:
      assert len(backtest[0]) > 1

    if y_col == 'y_des':
      cols = ['y_des', 'y_yearly', 'y_daily', 'y_trend']
    elif y_col == 'y_des_fft':
      cols = ['y_des_fft', 'y_fft']
    elif y_col == 'y_res':
      cols = ['y_res', 'y_yearly', 'y_daily']
    elif y_col == 'y':
      cols = ['y']

    cols.extend(['missing', 'isd_outlier'])

    preds_df = pd.concat([backtest[i].pd_dataframe() for i in range(len(backtest))], axis=0)
    trues_df = df.loc[preds_df.index, cols]

    hist_comp = pd.concat([trues_df, preds_df[y_col]], axis = 1)
    cols.append('pred')
    hist_comp.columns = cols

    # re-seasonalise
    if y_col == 'y_des':
      hist_comp['y_des'] += hist_comp['y_yearly'] + hist_comp['y_daily'] + hist_comp['y_trend']
      hist_comp['pred']  += hist_comp['y_yearly'] + hist_comp['y_daily'] + hist_comp['y_trend']
    elif y_col == 'y_des_fft':
      hist_comp['y_des_fft'] += hist_comp['y_fft']
      hist_comp['pred']      += hist_comp['y_fft']
    elif y_col == 'y_res':
      hist_comp['y_res'] += hist_comp['y_yearly'] + hist_comp['y_daily']
      hist_comp['pred']  += hist_comp['y_yearly'] + hist_comp['y_daily']


    hist_comp['res']    = hist_comp[y_col] - hist_comp['pred']
    hist_comp['res^2']  = hist_comp['res'] * hist_comp['res']
    hist_comp['res_sign']  = np.sign(hist_comp['res'])
    hist_comp['missing']   = hist_comp['missing']#.astype(int)

    list_int = [i for i in range(1, horizon+1)]
    reps = len(hist_comp) // len(list_int)
    hist_comp['step'] = np.tile(list_int, reps)
    hist_comp['id']   = np.repeat([i for i in range(reps)], horizon)
    hist_comp['date'] = hist_comp.index.values

    return hist_comp


def plot_one_step_abs_err_boxplot(one_step, title):
  one_step['abs_err'] = np.abs(one_step['res'])
  one_step[['abs_err']].boxplot(meanline  = False,
                                showmeans = True,
                                showcaps  = True,
                                showbox   = True,
                                )
  plt.title(title + '\nboxplot with mean and median')
  plt.suptitle('')
  plt.ylabel('absolute error')
  plt.show()


def plot_one_step_residuals_dist(one_step, title):
  plt.figure(figsize = (12, 16))
  plt.subplot(5, 1, 5)
  pd.Series(one_step['res']).plot(kind = 'density', label='residuals')
  plt.xlim(-10, 10)
  plt.title(title)
  plt.show()


def plot_one_step_residuals(one_step, title):
  x_miss = one_step.loc[one_step['missing'] == 1.0, 'obs'].index
  y_miss = one_step.loc[one_step['missing'] == 1.0, 'res']
```

```
  plt.figure(figsize = (12, 16))
  plt.subplot(5, 1, 4)
  plt.scatter(x = one_step.index, y = one_step['res'])
  plt.scatter(x_miss, y_miss, color='red', label='missing')
  plt.axhline(y = 0, color = 'grey')
  plt.xlabel('Index position')
  plt.ylabel('Residuals')
  plt.legend(loc='lower right')
  plt.title(title)
  plt.show()


def plot_one_step_obs_preds_dists(one_step, title):
  obs   = one_step['obs']
  preds = one_step['preds']
  r2score = r2_score(obs, preds)

  plt.figure(figsize = (12, 16))
  plt.subplot(5, 1, 3)
  pd.Series(obs).plot(kind = 'density', label='observations')
  pd.Series(preds).plot(kind = 'density', label='predictions')
  plt.xlim(-10, 40)
  plt.title(title)
  plt.legend()
  plt.annotate("$R^2$ = {:.3f}".format(r2score), (-7.5, 0.055))
  plt.show()


def plot_one_step_obs_vs_preds(one_step, title):

  obs   = one_step['obs']
  preds = one_step['preds']
  x_miss = one_step.loc[one_step['missing'] == 1.0, 'obs']
  y_miss = one_step.loc[one_step['missing'] == 1.0, 'preds']

  r2score = r2_score(obs, preds)

  plt.figure(figsize = (12, 16))
  plt.subplot(5, 1, 1)
  plt.scatter(x = obs, y = preds)
  plt.scatter(x_miss, y_miss, color='red', label='missing')
  plt.axline((0, 0), slope=1.0, color="grey")
  plt.xlabel('Observations')
  plt.ylabel('Predictions')
  plt.legend(loc='lower right')
  plt.annotate("$R^2$ = {:.3f}".format(r2score), (-9, 31))
  plt.title(title)
  plt.xlim((-10, 35))
  plt.ylim((-10, 35))
  plt.show()


def plot_one_step_diagnostics(model, data, val_series, val_pastcov_series, title, val_fut_cov=None):
  plot_feature_importances(model)

  # re-seasonalise observations
  if Y_COL == 'y_des':
    obs = data['y_des'] + data['y_yearly'] + data['y_daily'] + data['y_trend']
  elif Y_COL == 'y_des_fft':
    obs = data['y_des_fft'] + data['y_fft']
  else:
    obs = data[Y_COL]

  if val_fut_cov is None:
    res = model.residuals(series = val_series,
                          past_covariates = val_pastcov_series,
                          retrain = False).pd_series()
  else:
    res = model.residuals(series = val_series,
                          past_covariates = val_pastcov_series,
                          future_covariates = val_fut_cov,
                          retrain = False).pd_series()

  preds = obs + res
  preds = preds.dropna()
  obs  = obs[preds.index]
  res  = res[preds.index]
  miss = data.loc[preds.index, 'missing']

  print_rmse_mae(obs, preds, '1st', '# ')

  one_step = pd.concat([obs, preds, res, miss], axis=1)
```

```
  one_step.columns = ['obs', 'preds', 'res', 'missing']

  title = 'step = 1 ' + title
  plot_one_step_obs_vs_preds(one_step, title)
  # plot_obs_vs_mean_preds_by_step(hist, title)
  plot_one_step_obs_preds_dists(one_step, title)
  plot_one_step_residuals(one_step, title + ' residuals')
  plot_one_step_residuals_dist(one_step, title + ' residuals density')
  plot_one_step_residuals_acf(one_step, title + ' residuals acf')
  plot_one_step_residuals_qq(one_step, title + ' residuals qq-plot')
  plot_one_step_abs_err_boxplot(one_step, title)


def plot_one_step_residuals_qq(one_step, title_):
  fig, axs = plt.subplots(figsize=(6, 6))
  sm.qqplot(one_step['res'], line='q', ax=axs)
  axs.set_title(title_)
  plt.show()


def plot_one_step_residuals_acf(one_step, title_, max_lags = 300):
  plt.figure(figsize = (6, 6))

  acf = pd.DataFrame()
  acf_feat = 'res'

  acf[acf_feat] = [one_step[acf_feat].autocorr(l) for l in range(1, max_lags)]
  plt.plot(acf[acf_feat], label='residual')

  plt.axhline(0, linestyle='--', c='black')
  plt.ylabel('autocorrelation')
  plt.xlabel('time lags')
  plt.title(title_)
  plt.show()


def _plot_xy_for_label(data, label, x_feat, y_feat, color):
    x = data.loc[data[label] == 1.0, x_feat]
    y = data.loc[data[label] == 1.0, y_feat]

    if len(x) > 0:
        plt.scatter(x = x, y = y, color=color, alpha=0.5, label=label)


def plot_multistep_obs_vs_preds(hist, title, y_col=Y_COL):
    plt.figure(figsize = (12, 16))
    plt.subplot(5, 1, 1)
    plt.scatter(x = hist[y_col], y = hist['pred'])
    _plot_xy_for_label(hist, 'missing',      y_col, 'pred', 'red')
    plt.axline((0, 0), slope=1.0, color="grey")
    plt.xlabel('Observations')
    plt.ylabel('Predictions')
    plt.legend(loc='lower right')
    obs   = hist.loc[hist[[y_col, 'pred']].notnull().all(1), y_col]
    preds = hist.loc[hist[[y_col, 'pred']].notnull().all(1), 'pred']
    r2score = r2_score(obs, preds)
    plt.annotate("$R^2$ = {:.3f}".format(r2score), (-9, 31))
    plt.title(title)
    plt.xlim((-10, 35))
    plt.ylim((-10, 35))
    plt.show()


def plot_multistep_obs_vs_mean_preds_by_step(hist, title, y_col = Y_COL,
                                             step_ = HORIZON, ci = False):
    '''For specific step, plot mean prediction for each observation

    A 95 % confidence interval is plotted, but can be disabled
    '''

    mean_preds = hist.loc[hist['step'] == step_, [y_col, 'pred']].groupby(y_col).mean('pred')
    obs   = mean_preds.index.values
    preds = mean_preds['pred'].values

    plt.figure(figsize = (12, 16))
    ax = plt.subplot(5, 1, 2)
    plt.plot(obs, preds)

    if ci is True:
      ci = 1.96 * np.std(preds) / np.sqrt(len(obs))
      ax.fill_between(obs, (preds - ci), (preds + ci), color='b', alpha=.1)
```

```
    plt.axline((0, 0), slope=1.0, color="grey")
    r2score = r2_score(obs, preds)
    plt.annotate("$R^2$ = {:.3f} - step = {}".format(r2score, step_), (-9, 31))
    plt.title(title + ' step = ' + str(step_))
    plt.xlabel('Temperature')
    plt.ylabel('Mean prediction')
    plt.xlim((-10, 35))
    plt.ylim((-10, 35))
    plt.show()


def plot_multistep_obs_preds_dists(hist, title, y_col=Y_COL):
    obs   = hist.loc[hist[[y_col, 'pred']].notnull().all(1), y_col]
    preds = hist.loc[hist[[y_col, 'pred']].notnull().all(1), 'pred']
    r2score = r2_score(obs, preds)
    plt.figure(figsize = (12, 16))
    plt.subplot(5, 1, 3)
    pd.Series(obs).plot(kind = 'density', label='observations')
    pd.Series(preds).plot(kind = 'density', label='predictions')
    plt.xlim(-10, 40)
    plt.title(title)
    plt.legend()
    plt.annotate("$R^2$ = {:.3f}".format(r2score), (-7.5, 0.055))
    plt.show()


def plot_multistep_residuals(hist, title):
    plt.figure(figsize = (12, 16))
    plt.subplot(5, 1, 4)
    plt.scatter(x = range(len(hist)), y = hist['res'])
    hist['id.2'] = range(len(hist))
    _plot_xy_for_label(hist, 'missing',      'id.2', 'res', 'red')
    plt.axhline(y = 0, color = 'grey')
    plt.xlabel('Index position')
    plt.ylabel('Residuals')
    plt.legend(loc='lower right')
    plt.title(title)
    plt.show()


def plot_multistep_residuals_dist(hist, title):
    plt.figure(figsize = (12, 16))
    plt.subplot(5, 1, 5)
    pd.Series(hist['res']).plot(kind = 'density', label='residuals')
    plt.xlim(-10, 10)
    plt.title(title)
    plt.show()


# Unused?
# TODO Diagonal structure of these plots might need further consideration
#      Add lowess fit to check for problems
def plot_multistep_residuals_vs_predicted(hist, title):
    plt.subplot(5, 1, 5)
    plt.scatter(x = hist['pred'], y = hist['res'])
    _plot_xy_for_label(hist, 'missing',      'pred', 'res', 'red')
    plt.axhline(y = 0, color = 'grey')

    n = 24  # slow to run all points :-(
            # 12 takes approx 2 mins to run
            #  8 takes approx 4 mins to run
    xy = hist.iloc[::n, :]
    y_l = lowess(xy['res'], xy['pred'])
    plt.plot(y_l[:, 0], y_l[:, 1], 'green', label='lowess fit')

    plt.xlabel('Predictions')
    plt.ylabel('Residuals')
    plt.legend(loc='upper right')
    plt.title(title);


def se_(obs, preds, metric):
    '''Standard error of sum of squared residuals or sum of absolute residuals'''

    if _check_obs_preds_lens_eq(obs, preds) == 0:
        stop()

    if metric == 'rmse':
        se = np.sqrt(np.sum((obs - preds) ** 2) / len(obs))
    elif metric == 'mae':
        se = np.sqrt(np.sum(np.abs(obs - preds)) / len(obs))
    else:
```

```
        print('Unrecognised metric:', metric)
        print("metric should be 'rmse' or 'mae'")
        stop()

    return se


def metric_ci_vals(test_val, se, z_val = 1.95996):
    cil = z_val * se
    metric_cil = test_val - cil
    metric_ciu = test_val + cil

    return metric_cil, metric_ciu


# TODO: Remove unused confidence intervals
# NOTE: VAR baseline metrics cvar_rmse and cvar_mae hardcoded to 48 steps
def plot_horizon_metrics(hist, title, y_col=Y_COL, horizon = HORIZON, ci=False):
    steps = [i for i in range(1, horizon+1)]

    # calculate metrics
    # z_val_95 = 1.95996
    z_val_50 = 0.674
    rmse_h,   mae_h    = np.zeros(horizon), np.zeros(horizon)
    res_se_h, abs_se_h = np.zeros(horizon), np.zeros(horizon)
    rmse_ciu, rmse_cil = np.zeros(horizon), np.zeros(horizon)
    mae_ciu,  mae_cil  = np.zeros(horizon), np.zeros(horizon)

    for i in range(1, horizon+1):
      obs   = hist.loc[hist['step'] == i, y_col]
      preds = hist.loc[hist['step'] == i, 'pred']
      rmse_h[i-1] = rmse_(obs, preds)
      mae_h[i-1]  =  mae_(obs, preds)
      res_se_h[i-1] = se_(obs, preds, 'rmse')
      abs_se_h[i-1] = se_(obs, preds, 'mae')
      # mae_h[i]  = np.median(np.abs(obs - preds))  # for comparison with baselines
      rmse_cil[i-1], rmse_ciu[i-1] = metric_ci_vals(rmse_h[i-1], res_se_h[i-1], z_val_50)
      mae_cil[i-1],  mae_ciu[i-1]  = metric_ci_vals(mae_h[i-1],  abs_se_h[i-1], z_val_50)

    # plot metrics for horizons
    fig, axs = plt.subplots(1, 2, figsize = (14, 7))
    fig.suptitle(title + ' forecast horizon errors')
    axs = axs.ravel()


    mean_val_lab = title + ' mean value'
    axs[0].plot(steps, rmse_h, color='green', label=title)

    if ci is True:
      axs[0].fill_between(steps, rmse_cil, rmse_ciu, color='green', alpha=0.25)

    # i - initial, u - updated, c - corrected
    #ivar_rmse = np.array([0.39, 0.52, 0.64, 0.75, 0.86, 0.96, 1.06, 1.15, 1.23,
    #                      1.31, 1.38, 1.45, 1.51, 1.57, 1.63, 1.68, 1.73, 1.77,
    #                      1.81, 1.85, 1.89, 1.92, 1.96, 1.99, 2.02, 2.05, 2.08,
    #                      2.1 , 2.13, 2.15, 2.18, 2.2 , 2.22, 2.24, 2.26, 2.28,
    #                      2.3 , 2.31, 2.33, 2.35, 2.36, 2.38, 2.39, 2.4 , 2.42,
    #                      2.43, 2.44, 2.45])
    # NOTE: uvar_rmse tested on test_df
    #uvar_rmse = np.array([0.36, 0.49, 0.6, 0.7, 0.8, 0.89, 0.98, 1.06, 1.14,
    #                       1.21, 1.28, 1.35, 1.41, 1.47, 1.52, 1.57, 1.62, 1.66,
    #                       1.7, 1.74, 1.78, 1.81, 1.84, 1.87, 1.9, 1.93, 1.96,
    #                       1.99, 2.01, 2.03, 2.06, 2.08, 2.1, 2.12, 2.14, 2.16,
    #                       2.18,  2.19, 2.21, 2.23, 2.24, 2.26, 2.27, 2.29, 2.3,
    #                       2.31, 2.33, 2.34])
    cvar_rmse = np.array([0.49318888, 0.70222546, 0.88570688, 1.05495349,
    1.21081157, 1.34945832, 1.46844034, 1.57779714, 1.67754323, 1.7665827,
    1.84567039, 1.91561743, 1.97899766, 2.03616174, 2.08661944, 2.13396441,
    2.17809725, 2.21946156, 2.25780078, 2.29370568, 2.3272055,  2.35760153,
    2.38520845, 2.41076185, 2.43404716, 2.45466806, 2.47361784, 2.49117761,
    2.50625606, 2.52023589, 2.53319205, 2.54566125, 2.55764924, 2.56870554,
    2.57976955, 2.59102429, 2.6018822, 2.61242356, 2.62280045, 2.63353767,
    2.64410312, 2.65458709, 2.66532837, 2.67609086, 2.68675178, 2.69745108,
    2.71002892, 2.72445726])
    #axs[0].plot(steps, ivar_rmse, color='black', label='Initial VAR')
    axs[0].plot(steps, cvar_rmse, color='blue', label='Updated VAR')
    axs[0].hlines(np.mean(rmse_h), xmin=1, xmax=horizon,
                  color='green', linestyles='dotted', label=mean_val_lab)
    axs[0].hlines(np.mean(cvar_rmse), xmin=1, xmax=horizon,
                  color='blue', linestyles='dotted', label='Updated VAR mean value')
    axs[0].set_xlabel("horizon - half hour steps")
    axs[0].set_ylabel("rmse")
```

```
    axs[1].plot(steps, mae_h, color='green', label=title)

    if ci is True:
      axs[1].fill_between(steps, mae_cil, mae_ciu, color='green', alpha=0.25)

    # NOTE: ivar_mae tested on test_df
    #ivar_mae = np.array([0.39, 0.49, 0.57, 0.66, 0.74, 0.83, 0.91, 0.98, 1.05,
    #                     1.12, 1.18, 1.24, 1.29, 1.34, 1.39, 1.43, 1.47, 1.5 ,
    #                     1.53, 1.56, 1.59, 1.62, 1.64, 1.66, 1.68, 1.7 , 1.72,
    #                     1.73, 1.75, 1.76, 1.77, 1.78, 1.8 , 1.81, 1.82, 1.83,
    #                     1.83, 1.84, 1.85, 1.85, 1.86, 1.86, 1.87, 1.87, 1.88,
    #                     1.88, 1.89, 1.89])
    #uvar_mae = np.array([0.36, 0.45, 0.53, 0.61, 0.69, 0.76, 0.83, 0.9, 0.97,
    #                      1.03, 1.09, 1.14, 1.19, 1.24, 1.28, 1.32, 1.36, 1.4,
    #                      1.43, 1.46, 1.49, 1.52, 1.54, 1.56, 1.58, 1.6, 1.62,
    #                      1.63, 1.65, 1.66, 1.68, 1.69, 1.7, 1.71, 1.72, 1.73,
    #                      1.74, 1.74, 1.75, 1.75, 1.76, 1.76, 1.77, 1.77, 1.78,
    #                      1.78, 1.78, 1.78])
    cvar_mae = np.array([0.34694645, 0.50765333, 0.65132003, 0.78584432,
    0.9077075,  1.01705088, 1.11113622, 1.19759807, 1.27696634, 1.34941444,
    1.4134705,  1.47180058, 1.52304802, 1.56961154, 1.60903759, 1.64763418,
    1.68391297, 1.71690735, 1.74787094, 1.77721642, 1.80442554, 1.82951782,
    1.85358226, 1.87488643, 1.89346337, 1.91069565, 1.92613218, 1.94071845,
    1.95245349, 1.96323923, 1.9736734,  1.98370815, 1.99367508, 2.00204077,
    2.00992601, 2.01796976, 2.02747736, 2.03477489, 2.04173317, 2.04985428,
    2.05843847, 2.06731348, 2.07606609, 2.08533656, 2.09560914, 2.10668272,
    2.1183637,  2.13164371])
    #axs[1].plot(steps, ivar_mae, color='black', label='Initial VAR')
    axs[1].plot(steps, cvar_mae, color='blue', label='Updated VAR')
    axs[1].hlines(np.mean(mae_h), xmin=1, xmax=horizon,
                  color='green', linestyles='dotted', label=mean_val_lab)
    axs[1].hlines(np.mean(cvar_mae), xmin=1, xmax=horizon,
                  color='blue', linestyles='dotted', label='Updated VAR mean value')
    axs[1].set_xlabel("horizon - half hour steps")
    axs[1].set_ylabel("mae")

    plt.legend(bbox_to_anchor=(1.04, 0.5), loc="center left", borderaxespad=0)
    plt.show()


def plot_horizon_metrics_boxplots(hist, title):

  hist['abs_err'] = np.abs(hist['res'])
  hist[['abs_err', 'step']].boxplot(by='step',
                                    meanline=False,
                                    showmeans=True,
                                    showcaps=True,
                                    showbox=True,
                                    showfliers=False,
                                    )
  plt.title(title + '\nboxplots with mean and median')
  plt.suptitle('')
  plt.xlabel("horizon - half hour steps")
  plt.ylabel("absolute error")
  x_step = 10.0
  x_max  = np.ceil(np.max(hist.step) / x_step) * int(x_step)
  plt.xticks(np.arange(0, x_max, int(x_step)))
  plt.show()


def plot_multistep_diagnostics(hist, title, y_col=Y_COL):
  title = 'Multi-step ' + title
  plot_multistep_obs_vs_preds(hist, title, y_col)
  plot_multistep_obs_vs_mean_preds_by_step(hist, title, y_col)
  plot_multistep_obs_preds_dists(hist, title, y_col)
  plot_multistep_residuals(hist, title + ' residuals')
  plot_multistep_residuals_dist(hist, title + ' residuals density')
  plot_horizon_metrics(hist, title, y_col)
  plot_horizon_metrics_boxplots(hist, title)
  plot_multistep_forecast_examples(hist, title + ' forecast examples')


# TODO Refactor this
#      miss, preds, obs, res, err, dates etc "family" of variables
#      is a warning sign
#      try-catch around lagged_miss is clear indication of upsteam issues
#      Consider using a better data structure
#      Originally, I filtered out missing == 0 examples
#      This is more trouble than it's worth
#      Going forward, plot 'missing' observations in red
```

```
#      See also: plot_forecast_examples immediately below
def _filter_out_missing(pos_neg_rmse_all, miss, lags, subplots):
    '''Check if obs (lags and horizon) missing == 1.0
    and
    Avoid contiguous indices'''

    pos_neg_rmse = pd.Series(subplots)
    subplot_count = j = 0

    while subplot_count < subplots:
      restart = False
      idx = pos_neg_rmse_all.index[j]

      # Avoid indices in the first few observations
      # Would be incomplete
      if idx < lags:
        j += 1
        continue

      # Avoid contiguous indices - don't want 877, 878, 879
      if subplot_count > 0:
        for i in range(subplot_count):
          if abs(idx - pos_neg_rmse[i]) < lags:
            restart = True
            break

      if restart is False:
        try:
            lagged_miss = (miss.loc[idx - lags, :] == 1.0).any()
        except KeyError:
            lagged_miss = True

        horizon_miss = (miss.loc[idx, :] == 1.0).any()
        missing = lagged_miss or horizon_miss

        # NOTE Disabled filtering out examples with 'missing' values
        # if missing is False:
        pos_neg_rmse[subplot_count] = idx
        subplot_count += 1

      j += 1

    return pos_neg_rmse


# TODO Refactor this
#      miss, preds, obs, res, err, dates etc "family" of variables
#      is a warning sign
#      Consider using a better data structure
#      Originally, I filtered out missing == 0 examples
#      This is more trouble than it's worth
#      Going forward, plot 'missing' observations in red
#      See also: _filter_out_missing immediately above
def plot_multistep_forecast_examples(hist, title, subplots = 3, horizon = HORIZON, lags = 48):
    """Plot example forecasts with observations and lagged temperatures.
       Ensure examples are non-contiguous.

       First row shows near zero rmse forecasts.
       Second row shows most positive rmse forecasts.
       Third row shows most negative rmse forecasts.
    """

    assert subplots in [3, 4, 5]

    col = 'step'
    id_col = 'id'
    miss  = hist.pivot_table(index=id_col, columns=col, values='missing')
    preds = hist.pivot_table(index=id_col, columns=col, values='pred')
    obs   = hist.pivot_table(index=id_col, columns=col, values='y_des')
    res   = hist.pivot_table(index=id_col, columns=col, values='res')
    err   = hist.pivot_table(index=id_col, columns=col, values='res^2')
    dates = hist.pivot_table(index=id_col, columns=col, values='date')

    miss.dropna(inplace=True)
    # print("miss:", miss.shape)
    preds.dropna(inplace=True)
    # print("preds:", preds.shape)
    # obs.dropna(inplace=True)
    # print("obs:", obs.shape)
    res.dropna(inplace=True)
    # print("res:", res.shape)
    err.dropna(inplace=True)
```

```
# print("err:", err.shape)
dates.dropna(inplace=True)
# print("dates:", dates.shape)
dates = dates.iloc[err.index, :]
# print("dates indexed:", dates.shape)

# res_sign = np.sign(-res.mean(axis = 1))
# err_row_means = err.mean(axis = 1)
# rmse_rows = res_sign * np.sqrt(err_row_means)
err_row_means = np.sum(err, axis = 1) / horizon
# res_sum = np.sum(res, axis = 1)
# print("res_sum:",  len(res_sum))
# print(res_sum[0:5])
res_sign  = np.sign(np.sum(res, axis = 1))
rmse_rows = res_sign * np.sqrt(err_row_means)
# print("rmse_rows:", len(rmse_rows))
# print("res_sign:",  len(res_sign))
# print(res_sign[0:5])

# choose forecasts - check for missing == 0
# neg_rmse_all = np.argsort(rmse_rows)
##pos_rmse_all = np.flip(np.argsort(rmse_rows))
# pos_rmse_all = np.argsort(-rmse_rows)
neg_rmse_all = rmse_rows.sort_values()
# print(rmse_rows.loc[neg_rmse_all.index])
pos_rmse_all = neg_rmse_all[::-1]
# print(rmse_rows.loc[pos_rmse_all.index])
nz_rmse_all  = rmse_rows.abs().sort_values()
# print(rmse_rows.loc[nz_rmse_all.index])
# nz_rmse_all  = np.argsort(np.abs(rmse_rows))  # nz near zero
# print("\nneg_rmse_all:", len(neg_rmse_all))
# print(rmse_rows[neg_rmse_all[0:5]])
# print("pos_rmse_all:", len(pos_rmse_all))
# print(rmse_rows[pos_rmse_all[0:5]])
# print("nz_rmse_all: ", len(nz_rmse_all))
# print(rmse_rows[nz_rmse_all[0:5]])

nz_rmse  = _filter_out_missing(nz_rmse_all,  miss, lags, subplots)
pos_rmse = _filter_out_missing(pos_rmse_all, miss, lags, subplots)
neg_rmse = _filter_out_missing(neg_rmse_all, miss, lags, subplots)

plot_idx = np.concatenate((nz_rmse, pos_rmse, neg_rmse))
# print("\nplot_idx:", len(plot_idx))
# print("\nplot_idx:", plot_idx)

# plot forecasts
fig, axs = plt.subplots(3, subplots, sharey = True, figsize = (15, 10))
fig.tight_layout()
fig.subplots_adjust(hspace = 0.3, top = 0.87)
axs = axs.ravel()

myFmt = mdates.DateFormatter('%H:%M')

for i in range(3 * subplots):
  # print("plot_idx[i] - lags:",plot_idx[i],  plot_idx[i] - lags)
  axs[i].plot(dates.iloc[plot_idx[i] - lags, :],
              obs.loc[plot_idx[i] - lags, :],
              'blue',
              label='lagged observations')

  axs[i].plot(dates.iloc[plot_idx[i], :],
              obs.loc[plot_idx[i], :],
              'green',
              label='observations')

  axs[i].plot(dates.iloc[plot_idx[i], :],
              preds.loc[plot_idx[i], :],
              'orange',
              label='forecast')

  axs[i].xaxis.set_major_formatter(myFmt)
  obs_dates = dates.iloc[plot_idx[i] - lags, :]
  sub_title = "{0}, {1:d}, {2:.3f}".format(obs_dates.iloc[0],
                                           plot_idx[i],
                                           rmse_rows.loc[plot_idx[i]])
  axs[i].title.set_text(sub_title)

fig.suptitle(title + "\ninit date, period idx, signed rmse")
fig.text(0.5, 0.04, 'hour', ha='center')
fig.text(0.04, 0.5, 'Temperature - $^\circ$C', va='center', rotation='vertical')
plt.legend(bbox_to_anchor=(1.04, 0.5), loc="center left", borderaxespad=0)
plt.show();
```

```
# WARN This function probably has too many arguments - consider refactoring
def get_rmse_mae_from_backtest(model, param_df, i, series, past_cov, data, prefix, horizon=HORIZON, digits=6):
    backtest = model.historical_forecasts(series = series,
                                          past_covariates = past_cov,
                                          start   = 0.01,
                                          retrain = False,
                                          verbose = True,
                                          forecast_horizon = horizon,
                                          last_points_only = False)
    hc = get_historic_comparison(backtest, data)
    obs   = hc.loc[hc['step'] == horizon, Y_COL]
    preds = hc.loc[hc['step'] == horizon, 'pred']
    param_df.at[i, prefix + '_rmse'] = round(rmse_(obs, preds), digits)
    param_df.at[i, prefix + '_mae']  = round(mae_(obs,  preds), digits)

    return param_df


def plot_lgb_learning_curve(models, title = None, metric = 'l2', margin = None):
    '''Plot learning curve for lightGBM models using the lightGBM plot_metric function

    evals_results_ for validation data missing in action
    So, build 2 models - first with training data for validation
                       - second with validation data for validation
                       - pass both models in as a list
                       - order of models is important

    Training and validation curves are plotted when model.fit is called with
    both training and validation data:
    model.fit(series,
              past_covariates = past_cov,
              val_series = val_ser,
              val_past_covariates = val_past_cov)

    Primarily tested with catboost
    '''

    assert len(models) == 2

    final_rmse = []

    for model in models:
      assert hasattr(model, 'model')
      assert hasattr(model.model, 'evals_result_')

      final_rmse.append(model.model.evals_result_['valid_0'][metric][-1])

    if margin is None:
      lgb.plot_metric(models[0].model.evals_result_)
    else:
      assert margin > 0.0
      y_lim_min = min(final_rmse) - margin
      y_lim_max = max(final_rmse) + margin

      if y_lim_min < 0.0:
        y_lim_min = 0.0

      y_lim = (y_lim_min, y_lim_max)

      lgb.plot_metric(models[0].model.evals_result_, ylim = y_lim)

    plt.plot(models[1].model.evals_result_['valid_0']['l2'])
    plt.gca().get_lines()[0].set_color('blue')

    labels_ = ['train']
    if len(plt.gca().get_lines()) == 1 and plt.gca().get_label() == 'valid_0':
      labels_ = ['valid']

    if len(plt.gca().get_lines()) > 1:
      plt.gca().get_lines()[1].set_color('orange')
      labels_.append('valid')

    if title is not None:
      plt.title(title)

    plt.legend(labels = labels_)
    plt.show()


def plot_learning_curves(models, title, margin=0.05):
```

```
  print("\n")

  if type(models) is list and len(models) == 2:
    plot_lgb_learning_curve(models, title)
    plot_lgb_learning_curve(models, title, margin = margin)
  else:
    print('Unsupported number of models: ', len(models))
    print('models should have length 1 or 2!')

  print("\n")

  return None


def get_main_plot_title(pre_str, lag_params, mod_params):
  lag_params_str = ', '.join([f"{' '.join(map(str, v))}" for v in lag_params.items()])
  mod_params_str = ', '.join([f"{' '.join(map(str, v))}" for v in mod_params.items()])
  plot_title = pre_str + lag_params_str + '\n' + mod_params_str

  return plot_title


def build_two_lgbm_models(mod_params, data_params, train, valid):
  '''lgbm only for now ...'''

  series, past_cov, fut_cov = get_darts_series(train.loc['2016-01-12':,], data_params)
  val_ser, val_past_cov, val_fut_cov = get_darts_series(valid, data_params)

  model_tr1 = LightGBMModel(**mod_params)
  model1    = LightGBMModel(**mod_params)

  if data_params['fut_cov_cols'] is not None:
    model_tr1.fit(series,
                  past_covariates = past_cov,
                  future_covariates = fut_cov,
                  val_series = series,
                  val_past_covariates = past_cov,
                  val_future_covariates = fut_cov,
                  callbacks = [lgb.log_evaluation(0)]
                 )
  else:
    model_tr1.fit(series,
                  past_covariates = past_cov,
                  val_series = series,
                  val_past_covariates = past_cov,
                  callbacks = [lgb.log_evaluation(0)]
                 )

  if data_params['fut_cov_cols'] is not None:
    model1.fit(series,
               past_covariates = past_cov,
               future_covariates = fut_cov,
               val_series = val_ser,
               val_past_covariates = val_past_cov,
               val_future_covariates = val_fut_cov,
               callbacks = [lgb.log_evaluation(0)]
              )
  else:
    model1.fit(series,
               past_covariates = past_cov,
               val_series = val_ser,
               val_past_covariates = val_past_cov,
               callbacks = [lgb.log_evaluation(0)]
              )

  return model_tr1, model1


def drop_correlated_cols(dataset, threshold=0.95):
  '''Adapted from https://stackoverflow.com/a/44674459/100129'''

  col_corr = set()  # Set of all the names of deleted columns
  corr_matrix = dataset.corr(numeric_only=True).abs()

  for i in range(len(corr_matrix.columns)):
    for j in range(i):
      if (corr_matrix.iloc[i, j] >= threshold) and (corr_matrix.columns[j] not in col_corr):
        colname = corr_matrix.columns[i]
        col_corr.add(colname)
        if colname in dataset.columns:
          del dataset[colname]
```

```
  return dataset


def plot_observation_examples(df, cols, num_plots = 9):
    """Plot 9 sets of observations in 3 * 3 matrix"""

    num_plots_sqrt = int(np.sqrt(num_plots))
    assert num_plots_sqrt ** 2 == num_plots

    days = df.ds.dt.date.sample(n = num_plots).sort_values()
    p_data = [df[df.ds.dt.date.eq(days[i])] for i in range(num_plots)]

    fig, axs = plt.subplots(num_plots_sqrt, num_plots_sqrt, figsize = (15, 10))
    axs = axs.ravel()  # apl for the win :-)

    for i in range(num_plots):
        for col in cols:
            axs[i].plot(p_data[i]['ds'], p_data[i][col])
            axs[i].xaxis.set_tick_params(rotation = 20, labelsize = 10)

    fig.suptitle('Observation examples')
    fig.legend(cols, loc = 'lower center',  ncol = len(cols))

    return None


# TODO Change to operate on single dataframe - More useful function :-)
#      Change as far as possible - merge(), common_cols etc
#      Then write a wrapper to operate on before and after dataframes
#      combine results and calculate differences
def sanity_check_df_rows_cols_labels(before, after,
                                     row_var_cutoff=0.005, col_var_cutoff=0.05,
                                     col_corr_cutoff=0.,
                                     fast=True, verbose=False):
  '''Sanity check dataframes before and after modifications

  WARN: default row_var_cutoff, col_var_cutoff, col_corr_cutoff are fairly arbitrary
        there is some redundancy between these tests

  '''

  print_v = print if verbose else lambda *a, **k: None

  df = pd.DataFrame(columns = ['before', 'after', 'diff'])
  df_labels = []

  label = 'rows'
  # start_time = timeit.default_timer()
  i = 0
  df.loc[len(df), df.columns] = before.shape[i], after.shape[i], 0
  df_labels.append(label)
  # print('\t', label, round(timeit.default_timer() - start_time, 2))

  label = 'cols'
  # start_time = timeit.default_timer()
  i = 1
  df.loc[len(df), df.columns] = before.shape[i], after.shape[i], 0
  df_labels.append(label)
  # print('\t', label, round(timeit.default_timer() - start_time, 2))

  label = 'missing_rows'
  # start_time = timeit.default_timer()
  i = 0
  before_after = pd.merge(before, after, left_index=True, right_index=True, how='outer', indicator=True)
  missing_rows = before_after.loc[before_after['_merge'] == 'left_only', :]
  df.loc[len(df), df.columns] = 0, missing_rows.shape[i], 0
  if missing_rows.shape[i] > 0:
    print_v('\n', label, ':')
    print_v(missing_rows)
  df_labels.append(label)
  # print('\t', label, round(timeit.default_timer() - start_time, 2))

  label = 'missing_cols'
  # start_time = timeit.default_timer()
  i = 1
  common_cols = before.columns.intersection(after.columns)
  missing_cols = before.shape[i] - len(common_cols)
  df.loc[len(df), df.columns] = 0, missing_cols, 0
  if missing_cols > 0:
    print_v('\n', label, ':')
    print_v(set(before.columns) - set(common_cols))
  df_labels.append(label)
```

```
# print('\t', label, round(timeit.default_timer() - start_time, 2))

label = 'total_nas'
# start_time = timeit.default_timer()
df.loc[len(df), df.columns] = before.isna().sum().sum(), \
                              after.isna().sum().sum(), 0
df_labels.append(label)
# print('\t', label, round(timeit.default_timer() - start_time, 2))

label = 'rows_with_nas'
# start_time = timeit.default_timer()
before_rows_nas = before.isnull().any(axis=1).sum()
after_rows_nas  = after.isnull().any(axis=1).sum()
df.loc[len(df), df.columns] = before_rows_nas, after_rows_nas, 0
if before_rows_nas != after_rows_nas:
  print_v('\n', label, ':')
  print_v(before[before.isnull().any(axis=1)])
  print_v(after[after.isnull().any(axis=1)])
df_labels.append(label)
# print('\t', label, round(timeit.default_timer() - start_time, 2))

label = 'cols_with_nas'
# start_time = timeit.default_timer()
before_cols_nas = before.isnull().any().sum()
after_cols_nas  = after.isnull().any().sum()
df.loc[len(df), df.columns] = before_cols_nas, after_cols_nas, 0
if before_cols_nas != after_cols_nas:
  print_v('\n', label, ':')
  print_v(before.isnull().any().index.values)
  print_v(after.isnull().any().index.values)
df_labels.append(label)
# print('\t', label, round(timeit.default_timer() - start_time, 2))

label = 'single_value_rows'
if not fast:
  # start_time = timeit.default_timer()
  before_single_value_rows = np.sum(before.nunique(axis=1) <= 1)
  after_single_value_rows  = np.sum(after.nunique(axis=1) <= 1)
  df.loc[len(df), df.columns] = before_single_value_rows, \
                                after_single_value_rows, 0
  if before_single_value_rows != after_single_value_rows:
    print_v('\n', label, ':')
    print_v(before[before.nunique(axis=1) <= 1])
    print_v(after[after.nunique(axis=1) <= 1])
  df_labels.append(label)
  # print('\t', label, round(timeit.default_timer() - start_time, 2))

label = 'single_value_cols'
# start_time = timeit.default_timer()
before_single_value_cols = np.sum(before.nunique() <= 1)
after_single_value_cols  = np.sum(after.nunique() <= 1)
df.loc[len(df), df.columns] = before_single_value_cols, \
                              after_single_value_cols, 0
if before_single_value_cols != after_single_value_cols:
  print_v('\n', label, ':')
  print_v(before.columns[before.nunique() <= 1].values)
  print_v(after.columns[after.nunique() <= 1].values)
df_labels.append(label)
# print('\t', label, round(timeit.default_timer() - start_time, 2))

# warnings.resetwarnings()

with warnings.catch_warnings():
  warnings.simplefilter('ignore')

  label = 'low_var_rows'
  # start_time = timeit.default_timer()
  before_low_var_rows = (before.select_dtypes(include=[np.number]).std(axis=1) <= row_var_cutoff).sum()
  after_low_var_rows  = (after.select_dtypes(include=[np.number]).std(axis=1) <= row_var_cutoff).sum()
  df.loc[len(df), df.columns] = before_low_var_rows, after_low_var_rows, 0
  if before_low_var_rows != after_low_var_rows:
    print_v('\n', label, ':')
    print_v(before.select_dtypes(include=[np.number]).std(axis=1) <= row_var_cutoff)
    print_v(after.select_dtypes(include=[np.number]).std(axis=1)  <= row_var_cutoff)
  df_labels.append(label)
  # print('\t', label, round(timeit.default_timer() - start_time, 2))

  label = 'low_var_cols'
  # start_time = timeit.default_timer()
  before_low_var_cols = (before.select_dtypes(include=[np.number]).std() <= col_var_cutoff).sum()
  after_low_var_cols  = (after.select_dtypes(include=[np.number]).std() <= col_var_cutoff).sum()
  df.loc[len(df), df.columns] = before_low_var_cols, after_low_var_cols, 0
```

```
    if before_low_var_cols != after_low_var_cols:
      print_v('\n', label, ':')
      s = before.select_dtypes(include=[np.number]).std() <= col_var_cutoff
      t = after.select_dtypes(include=[np.number]).std()  <= col_var_cutoff
      print_v(s[s].index.values)
      print_v(t[t].index.values)
    df_labels.append(label)
    # print('\t', label, round(timeit.default_timer() - start_time, 2))

  label = 'duplicate_rows'
  # start_time = timeit.default_timer()
  before_dup_rows = before.shape[0] - before.drop_duplicates().shape[0]
  after_dup_rows  = after.shape[0]  - after.drop_duplicates().shape[0]
  df.loc[len(df), df.columns] = before_dup_rows, after_dup_rows, 0
  if before_dup_rows != after_dup_rows:
    print_v('\n', label, ':')
    print_v(before[before.duplicated(keep=False)])
    print_v(after[after.duplicated(keep=False)])
  df_labels.append(label)
  # print('\t', label, round(timeit.default_timer() - start_time, 2))

  label = 'highly_correlated_cols'
  # .copy() so we don't modify the original dataframe
  if not fast:
    # start_time = timeit.default_timer()
    before_high_corr_cols = before.shape[1] - drop_correlated_cols(before.copy(), col_corr_cutoff).shape[1]
    after_high_corr_cols  = after.shape[1]  - drop_correlated_cols(after.copy(), col_corr_cutoff).shape[1]
    df.loc[len(df), df.columns] = before_high_corr_cols, after_high_corr_cols, 0
    if before_high_corr_cols != after_high_corr_cols:
      print_v('\n', label, ':')
      print_v(set(before.columns) - set(drop_correlated_cols(before.copy(), col_corr_cutoff).columns))
      print_v(set(after.columns) - set(drop_correlated_cols(after.copy(), col_corr_cutoff).columns))
    df_labels.append(label)
    # print('\t', label, round(timeit.default_timer() - start_time, 2))

  label = 'duplicate_index_labels'
  # start_time = timeit.default_timer()
  before_idx_labels = before.index.duplicated().sum()
  after_idx_labels  = after.index.duplicated().sum()
  df.loc[len(df), df.columns] = before_idx_labels, after_idx_labels, 0
  if before_idx_labels != after_idx_labels:
    print_v('\n', label, ':')
    print_v(before.index.duplicated())
    print_v(after.index.duplicated())
  df_labels.append(label)
  # print('\t', label, round(timeit.default_timer() - start_time, 2))

  label = 'duplicate_col_labels'
  # start_time = timeit.default_timer()
  before_dup_col_labels = before.columns.duplicated().sum()
  after_dup_col_labels  = after.columns.duplicated().sum()
  df.loc[len(df), df.columns] = before_dup_col_labels, after_dup_col_labels, 0
  if before_dup_col_labels != after_dup_col_labels:
    print_v('\n', label, ':')
    print_v(before.columns.duplicated())
    print_v(after.columns.duplicated())
  df_labels.append(label)
  # print('\t', label, round(timeit.default_timer() - start_time, 2))

  # TODO Find renamed columns from before dataframe in after dataframe?


  df['diff'] = df['after'] - df['before']
  df.index = df_labels

  return df


def sanity_check_before_after_dfs(before_, after_, ds_name, fast=True, verbose=False):

  print('\n', ds_name, sep='')

  # Reasons I HATE pandas number Inf a neverending series:
  # PerformanceWarning: DataFrame is highly fragmented.  This is usually the
  # result of calling `frame.insert` many times, which has poor performance.
  # Consider joining all columns at once using pd.concat(axis=1) instead. To
  # get a de-fragmented frame, use `newframe = frame.copy()`
  before = before_.copy()
  after  = after_.copy()

  # start_time = timeit.default_timer()
  sanity_df = sanity_check_df_rows_cols_labels(before, after, fast=fast, verbose=verbose)
```

```
  # print('\t sanity_check_df_rows_cols_labels', round(timeit.default_timer() - start_time, 2))


  # start_time = timeit.default_timer()
  print('before.index.equals(after.index):', before.index.equals(after.index))

  # check index freq is set and are equal
  print('before.index.freq == after.index.freq:', before.index.freq == after.index.freq)
  if verbose:
    print('before.index.freq:', before.index.freq)
    print('after.index.freq:',  after.index.freq)


  # check if common column dtypes have changed
  common_cols = before.columns.intersection(after.columns)
  print('before[common_cols].dtypes == after[common_cols].dtypes:',
        (before[common_cols].dtypes == after[common_cols].dtypes).all())
  if verbose:
    print('before[common_cols].dtypes:', before[common_cols].dtypes)
    print('after[common_cols].dtypes:',  after[common_cols].dtypes)

  # check if describe() summaries are equal
  print('before[common_cols].describe() == after[common_cols].describe():',
        (before[common_cols].describe() == after[common_cols].describe()).all().all())
  if verbose:
    print(before[common_cols].describe() == after[common_cols].describe())

  # check after subsetted by before equals before
  print('\nbefore[common_cols].equals(after[common_cols]):',
  before[common_cols].dropna().drop_duplicates().equals(after[common_cols].dropna().drop_duplicates())
  )
  if verbose:
    print('before.isin(after):',
    before[common_cols].dropna().drop_duplicates().isin(after[common_cols].dropna().drop_duplicates()).all().all()
    )
    print(before.dropna().drop_duplicates().isin(after.dropna().drop_duplicates()).all())
    print(before.dropna().drop_duplicates().isin(after.dropna().drop_duplicates()))


  # Reasons I HATE pandas number Inf a neverending series:
  # PerformanceWarning: DataFrame is highly fragmented.  This is usually the
  # result of calling `frame.insert` many times, which has poor performance.
  # Consider joining all columns at once using pd.concat(axis=1) instead. To
  # get a de-fragmented frame, use `newframe = frame.copy()`
  # calculate duplicate row counts then find mean duplicate count
  # for each column and finally find mean of means aka redundancy
  # warnings.resetwarnings()
  with warnings.catch_warnings():
    warnings.simplefilter('ignore')
    before_red = before.dropna().groupby(before.select_dtypes(include=np.number).columns.tolist(), as_index=False).size().mea
    after_red  = after.dropna().groupby(after.select_dtypes(include=np.number).columns.tolist(), as_index=False).size().mean
    print('redundancy before > after:', before_red > after_red)
    print('mean before feature redundancy:', round(before_red, 3))
    print('mean after feature redundancy: ', round(after_red,  3))

  # Check all data is numeric, finite (but allow NAs) and reasonably shaped
  # If any problems then this will error out
  # Only checking 'after' dataframe
  # https://scikit-learn.org/stable/modules/generated/sklearn.utils.check_X_y.html
  if Y_COL in after.columns:
    _, _ = check_X_y(after.drop(columns=[Y_COL, 'ds']),
                     after[Y_COL],
                     y_numeric = True,
                     force_all_finite = 'allow-nan')

  print()
  # print('\t end sanity_check_before_after_dfs', round(timeit.default_timer() - start_time, 2))

  display(sanity_df)

  return sanity_df


def compare_train_valid_test_sanity_dfs(train_sanity, valid_sanity, test_sanity, ex_labels=None):
  '''...'''

  if ex_labels is None:
    ex_labels = ['rows']

  train_sanity = train_sanity.loc[~train_sanity.index.isin(ex_labels)]
  valid_sanity = valid_sanity.loc[~valid_sanity.index.isin(ex_labels)]
  test_sanity  =  test_sanity.loc[~test_sanity.index.isin(ex_labels)]
```

```
  if not train_sanity.equals(valid_sanity):
    print('WARN: train_sanity != valid_sanity')
    display(pd.concat([train_sanity, valid_sanity]).drop_duplicates(keep=False))

  if not train_sanity.equals(test_sanity):
    print('WARN: train_sanity != test_sanity')
    display(pd.concat([train_sanity, test_sanity]).drop_duplicates(keep=False))

  if not test_sanity.equals(valid_sanity):
    print('WARN: test_sanity != valid_sanity')
    display(pd.concat([test_sanity, valid_sanity]).drop_duplicates(keep=False))

  return None


# TODO Remove some of the code duplication
def sanity_check_train_valid_test(train_df, valid_df, test_df,
                                  over_cols = ['y_des', 'dew.point_des', 'humidity', 'pressure'],
                                  dp = 2):

  # Check number of columns is equal
  if (train_df.shape[1] != valid_df.shape[1]) or \
     (train_df.shape[1] != test_df.shape[1])  or \
     (valid_df.shape[1] != test_df.shape[1]):
    print('ERROR: Inconsistent number of columns!')
    print('train_df.shape[1]:', train_df.shape[1])
    print('valid_df.shape[1]:', valid_df.shape[1])
    print('test_df.shape[1]:',  test_df.shape[1])


  # Check column names are equal
  if not (train_df.columns == valid_df.columns).all():
    print('ERROR: Inconsistent train_df, valid_df column names!')
    print('train_df.columns:', train_df.columns)
    print('valid_df.columns:', valid_df.columns)

  if not (train_df.columns == test_df.columns).all():
    print('ERROR: Inconsistent train_df, test_df column names!')
    print('train_df.columns:', train_df.columns)
    print('test_df.columns:',  test_df.columns)

  if not (valid_df.columns == test_df.columns).all():
    print('ERROR: Inconsistent valid_df, test_df column names!')
    print('valid_df.columns:', valid_df.columns)
    print('test_df.columns:',  test_df.columns)


  # Check column dtypes are equal
  if not (train_df.dtypes == valid_df.dtypes).all():
    print('ERROR: Inconsistent train_df, valid_df dtypes!')
    print('train_df.dtypes:', train_df.dtypes)
    print('valid_df.dtypes:', valid_df.dtypes)

  if not (train_df.dtypes == test_df.dtypes).all():
    print('ERROR: Inconsistent train_df, test_df dtypes!')
    print('train_df.dtypes:', train_df.dtypes)
    print('test_df.dtypes:',  test_df.dtypes)

  if not (valid_df.dtypes == test_df.dtypes).all():
    print('ERROR: Inconsistent valid_df, test_df dtypes!')
    print('valid_df.dtypes:', valid_df.dtypes)
    print('test_df.dtypes:',  test_df.dtypes)


  # Check index freqs are equal
  if train_df.index.freq != valid_df.index.freq:
    print('ERROR: Inconsistent train_df, valid_df index frequencies!')
    print('train_df.index.freq:', train_df.index.freq)
    print('valid_df.index.freq:', valid_df.index.freq)

  if train_df.index.freq != test_df.index.freq:
    print('ERROR: Inconsistent train_df, test_df index frequencies!')
    print('train_df.index.freq:', train_df.index.freq)
    print('test_df.index.freq:',   test_df.index.freq)

  if valid_df.index.freq != test_df.index.freq:
    print('ERROR: Inconsistent valid_df, test_df index frequencies!')
    print('valid_df.index.freq:', valid_df.index.freq)
    print('test_df.index.freq:',   test_df.index.freq)
```

```
# Verify dataframes are different!
if train_df.equals(valid_df):
  print('ERROR: train_df == valid_df!')

if train_df.equals(test_df):
  print('ERROR: train_df == test_df!')

if valid_df.equals(test_df):
  print('ERROR: valid_df == test_df!')


# Check no overlap between train_df.index and valid_df.index
# train_df.index strictly before valid_df.index and test_df.index
if max(train_df.index) >= min(valid_df.index):
  print('ERROR: Overlap between train_df, valid_df indices!')
  print('max(train_df.index):', max(train_df.index))
  print('min(valid_df.index):', max(valid_df.index))

# Check no overlap between train_df.index and test_df.index
# train_df.index strictly before valid_df.index and test_df.index
if max(train_df.index) >= min(test_df.index):
  print('ERROR: Overlap between train_df, test_df indices!')
  print('max(train_df.index):', max(train_df.index))
  print('min(test_df.index):',  max(test_df.index))


# Check no overlap between valid_df.index and test_df.index
# valid_df.index can be before or after test_df.index
if (max(valid_df.index) >= min(test_df.index)) and \
   (max(valid_df.index) <= max(test_df.index)):
  print('ERROR: Overlap between valid_df, test_df indices!')
  print('valid_df.index:', max(valid_df.index), '-', max(valid_df.index))
  print('test_df.index:',  max(test_df.index),  '-', max(test_df.index))

if (min(valid_df.index) >= min(test_df.index)) and \
   (min(valid_df.index) <= max(test_df.index)):
  print('ERROR: Overlap between valid_df, test_df indices!')
  print('valid_df.index:', max(valid_df.index), '-', max(valid_df.index))
  print('test_df.index:',  max(test_df.index),  '-', max(test_df.index))


# TODO: Consider enforcing a gap of 1 day to 1 week between
#       train_df.index and {valid_df,test_df}.index to avoid data leakage?


# Check train_df has more observations than valid_df and test_df
if valid_df.shape[0] > train_df.shape[0]:
  print('ERROR: valid_df more observations than train_df!')
  print('train_df observations:', train_df.shape[0])
  print('valid_df observations:', valid_df.shape[0])

if test_df.shape[0] > train_df.shape[0]:
  print('ERROR: test_df more observations than train_df!')
  print('train_df observations:', train_df.shape[0])
  print('test_df observations:',  test_df.shape[0])


# Check valid_df and test_df have equal number of observations
# valid_df and test_df may be different sizes but
# large size difference may indicate an issue
# TODO: Use calendar.isleap() to check if leap year
if valid_df.shape[0] != test_df.shape[0]:
  print('WARN: Inconsistent number of valid_df, test_df rows.  Leap year?')


# Check valid_df and test_df are each 1 year long
YEAR_OBS_MIN = 48 * 365
YEAR_OBS_MAX = 48 * 366
if (valid_df.shape[0] < YEAR_OBS_MIN) or \
   (valid_df.shape[0] > YEAR_OBS_MAX):
  print('ERROR: valid_df should be 1 year long [',
        YEAR_OBS_MIN, ',', YEAR_OBS_MAX, ']!')
  print('valid_df observations:', valid_df.shape[0])

if (test_df.shape[0] < YEAR_OBS_MIN) or \
   (test_df.shape[0] > YEAR_OBS_MAX):
  print('ERROR: test_df should be 1 year long [',
        YEAR_OBS_MIN, ',', YEAR_OBS_MAX, ']!')
  print('test_df observations:', test_df.shape[0])

# Check approx number of overlapping rows between train_df and valid_df
dups_pc_lim = 15.0
```

```
  n_dups, dups_pc = get_approx_overlap(train_df, valid_df, over_cols, decs=dp)
  if dups_pc > dups_pc_lim:
    print('WARN: high overlap between train_df and valid_df rows!')
    print(f"Number of shared rows: {n_dups}")
    print(f'Approximate overlap: {dups_pc} %\n')
    # print(f'Decimal places: {dp}')
    # print('Overlap features:', over_cols)

  # Check approx number of overlapping rows between train_df and test_df
  n_dups, dups_pc = get_approx_overlap(train_df, test_df, over_cols, decs=dp)
  if dups_pc > dups_pc_lim:
    print('WARN: high overlap between train_df and test_df rows!')
    print(f"Number of shared rows: {n_dups}")
    print(f'Approximate overlap: {dups_pc} %\n')
    # print(f'Decimal places: {dp}')
    # print('Overlap features:', over_cols)

  return None


def print_train_valid_test_shapes(df, train_df, valid_df, test_df):
  print("df shape: ",            df.shape)
  print("train shape:   ", train_df.shape)
  print("valid shape:   ", valid_df.shape)
  print("test shape:    ",  test_df.shape)

  return None


def plot_feature_history_single_df(data, var, missing=False):
    plt.figure(figsize = (12, 6))
    plt.scatter(data.index, data[var],
                label='train', color='black', s=3)
    if missing:
      label = 'missing'
      x_lab = data.loc[data[label] == 1.0, 'ds']
      y_lab = data.loc[data[label] == 1.0, var]
      plt.scatter(x_lab, y_lab, color='red', label=label, s=3)

    plt.title(var)
    plt.show()


def plot_feature_history(train, valid, test, var, missing=False):
    label = 'missing'

    plt.figure(figsize = (12, 6))
    plt.scatter(train.index, train[var],
                label='train', color='black', s=3)
    if missing:
      x_lab = train.loc[train[label] == 1.0, 'ds']
      y_lab = train.loc[train[label] == 1.0, var]
      plt.scatter(x_lab, y_lab, color='red', label=label, s=3)

    plt.scatter(valid.index, valid[var],
                label='valid', color='blue', s=3)
    if missing:
      x_lab = valid.loc[valid[label] == 1.0, 'ds']
      y_lab = valid.loc[valid[label] == 1.0, var]
      plt.scatter(x_lab, y_lab, color='red', label=label, s=3)

    plt.scatter(test.index,  test[var],
                label='test', color='purple', s=3)
    if missing:
      x_lab = test.loc[test[label] == 1.0, 'ds']
      y_lab = test.loc[test[label] == 1.0, var]
      plt.scatter(x_lab, y_lab, color='red', label=label, s=3)

    plt.title(var)
    plt.show()


def plot_feature_history_separately(train, valid, test, var):
    fig, axs = plt.subplots(1, 3, figsize = (14, 7))

    axs[0].plot(train.index, train[var])
    axs[0].set_title('train')

    axs[1].plot(valid.index, valid[var])
    axs[1].set_title('valid')
    axs[1].set_xticks(axs[1].get_xticks(), axs[1].get_xticklabels(), rotation=45, ha='right')
```

```
    axs[2].plot(test.index,  test[var])
    axs[2].set_title('test')
    axs[2].set_xticks(axs[2].get_xticks(), axs[2].get_xticklabels(), rotation=45, ha='right')

    fig.suptitle(var)
    plt.show()


def check_high_low_thresholds(df, ds=None):
  '''Check main features from dataframe are within reasonable thresholds'''

  all_ok = True
  feats = ['y', 'dew.point', 'humidity', 'pressure',
           'wind.speed.mean', 'wind.speed.max']
  highs = [ 45,  25, 100, 1060, 35, 70]
  lows  = [-20, -20,   5,  950,  0,  0]

  thresh = pd.DataFrame({'feat': feats,
                         'high': highs,
                         'low':  lows,})
  thresh.index = feats

  for feat in feats:
    feat_high = thresh.loc[feat, 'high']
    feat_low  = thresh.loc[feat, 'low']

    if not df[feat].between(feat_low, feat_high).all():
      all_ok = False
      print('%15s [%3d, %3d] - % 7.3f, % 7.3f' %
            (feat, feat_low, feat_high,
            round(min(df[feat]), 3), round(max(df[feat]), 3)))

  # check if dew.point ever greater than temperature
  if df.loc[df['dew.point'] > df['y'], ['y', 'dew.point']].shape[0] != 0:
    all_ok = False
    print('dew.point > y:')
    display(df.loc[df['dew.point'] > df['y'], ['y', 'dew.point']])

  if all_ok is False:
    print(' ... from', ds)

  return None


def get_features_filename(feat_name, data_name, date_str, file_ext='.csv.xz'):
    return feat_name + data_name + date_str + file_ext


def merge_data_and_aggs(data, aggs):
  data = pd.concat((data, aggs), axis=1)

  data['ds'] = data.index
  data = data[~data.index.duplicated(keep = 'first')]
  data = data.asfreq(freq = '30min')

  return data


def print_null_columns(df, df_name):
  print('\n', df_name, 'null columns:')
  display(df[df.columns[df.isnull().any()]].isnull().sum())


def print_na_locations(df):
  '''Print index row and column labels for NA in dataframe'''

  for index, row in df[df.isna().any(axis=1)].items():
    for col_name, _ in row.items():
      if pd.isnull(df.loc[index, col_name]):
        print(index, col_name)

  return None


def get_darts_series(data, data_params):
  series   = TimeSeries.from_dataframe(data, value_cols=data_params['y_col'])
  past_cov = TimeSeries.from_dataframe(data, value_cols=data_params['past_cov_cols'])

  if data_params['fut_cov_cols'] is not None:
    fut_cov = TimeSeries.from_dataframe(data, value_cols=data_params['fut_cov_cols'])
  else:
    fut_cov = None
```

```
  return series, past_cov, fut_cov


def plot_short_term_acf(data, acf_feats, acf_cols, title_,
                        mean_feat = False, max_lags = 300):
  plt.figure(figsize = (12, 6))

  acf = pd.DataFrame()

  for acf_feat, acf_col in zip(acf_feats, acf_cols):
    acf[acf_feat] = [data[acf_feat].autocorr(l) for l in range(1, max_lags)]
    plt.plot(acf[acf_feat], label=acf_feat, c=acf_col)

  if mean_feat:
    acf['mean_acf'] = acf.mean(axis=1)
    plt.plot(acf['mean_acf'], label='mean_acf', c='black')

  plt.axhline(0, linestyle='--', c='black')
  plt.axhline(0.875, linestyle=':', c='lightgrey')
  plt.ylabel('autocorrelation')
  plt.xlabel('time lags')
  plt.title(title_)
  plt.legend()
  plt.show()


def plot_long_term_acf(data, var, num_years=3):
  # WARN: Slow function :-(
  #       Results are more useful when displaying more years of data
  pd.plotting.autocorrelation_plot(data[var].head(17532 * num_years))
  plt.title(var)
  plt.show()


def print_df_summary(df):
  print("Shape:")
  display(df.shape)

  total_nas = df.isna().sum().sum()
  rows_nas  = df.isnull().any(axis=1).sum()
  cols_nas  = df.isnull().any().sum()
  print('\nTotal NAs:', total_nas)
  print('Rows with NAs:', rows_nas)
  print('Cols with NAs:', cols_nas)

  print("\nInfo:")
  display(df.info())

  print("\nSummary stats:")
  display(df.describe())

  print("\nRaw data:")
  display(df)
  print("\n")


def get_approx_overlap(X1, X2, over_cols, decs=2, verbose=False):
  '''Calculate approximate overlap between 2 dataframes of different sizes.

  If exact values are used then overlap is probably too low,
  so use np.round() to reduce precision.
  Use MinMaxScaler so single decimals parameter is applicable to all columns.
  Assumes X1 is train and X2 is valid/test.
  Duplicates dropped from X1 & X2 before calculating overlap.
  Percent overlap can be greater than 100 if decs is too low.

  Based on https://stackoverflow.com/a/71002234/100129
  '''

  assert X1.shape[0] >= X2.shape[0]

  X1 = X1[over_cols].drop_duplicates()
  X2 = X2[over_cols].drop_duplicates()

  Xcomb = pd.concat((X1, X2), axis=0, ignore_index=True)

  # scale
  scaler = MinMaxScaler()
  Xscl = scaler.fit_transform(Xcomb)

  # round
```

```
  df_scl = pd.DataFrame(np.round(Xscl, decimals=decs), columns=over_cols)

  # count overlaps
  n_uniq = df_scl.drop_duplicates().shape[0]
  n_dup = X1.shape[0] + X2.shape[0] - n_uniq
  dup_pc = round(n_dup * 100 / X2.shape[0], 2)

  if verbose:
    print(f"Number of shared rows: {n_dup}")
    print(f'Approximate overlap: {dup_pc} %\n')

  if dup_pc > 100.0:
    print('Approx. overlap over 100 %!')
    print('Increase decs argument')
    print(f"decs = {decs}\n")

  return n_dup, dup_pc


def add_transform_column(mod_vi_df, us_feats):
  mod_vi_df['transform'] = 'None'

  ne_mask = mod_vi_df['feature_transform'] != mod_vi_df['feature']
  ne_fs = mod_vi_df.loc[ne_mask, 'feature'].to_list()
  ne_fts = mod_vi_df.loc[ne_mask, 'feature_transform'].to_list()


  ne_ts = [ne_ft.replace(ne_fs[i] + '_', '') for i, ne_ft in enumerate(ne_fts)]

  mod_vi_df.loc[ne_mask, 'transform'] = ne_ts

  for us_feat in us_feats:
    if us_feat.endswith('_des'):
      # 'y_des', 'dew.point_des'
      mod_vi_df.loc[mod_vi_df['feature_transform'] == us_feat, 'transform'] = 'des'
    else:
      # 't_pot', 'vp_def', 'air_density', 'ground_hf', 'za_rad', ...
      mod_vi_df.loc[mod_vi_df['feature'] == us_feat, 'transform'] = 'None'

  wind_mask = mod_vi_df['feature_transform_lag'].str.contains('_window_')
  mod_vi_df.loc[wind_mask, 'transform'] = mod_vi_df.loc[wind_mask, 'feature_transform_lag'].str.extract(r'_window_\d+_(.*)_[1

  return mod_vi_df


def add_feature_column(mod_vi_df, us_feats):
  mod_vi_df['feature'] = ''

  # extract features which DO NOT contain underscores
  # eg. irradiance, pressure ...
  useq0_mask = mod_vi_df['feature_transform'].str.count('_') == 0
  mod_vi_df.loc[useq0_mask, 'feature'] = mod_vi_df.loc[useq0_mask, 'feature_transform']
  # print('useq0:')
  # display(mod_vi_df.loc[useq0_mask, 'feature'])


  # extract features which DO contain underscores
  # cannot distinguish between y_des and pressure_grad
  useq1_mask = mod_vi_df['feature_transform'].str.count('_') == 1
  # pandas, for clowns by clowns - expand=False your arse (at least an hour wasted)
  mod_vi_df.loc[useq1_mask, 'feature'] = mod_vi_df.loc[useq1_mask, 'feature_transform'].str.extract(r'^(.*)_', expand=False)
  # print('useq1:')
  # display(mod_vi_df.loc[useq1_mask, 'feature'])

  # cannot distinguish between y_des_hist_mode and pressure_intervals_intervals_mean
  usgt1_mask = mod_vi_df['feature_transform'].str.count('_') > 1
  # pandas, for clowns by clowns - expand=False your arse (at least an hour wasted)
  mod_vi_df.loc[usgt1_mask, 'feature'] = mod_vi_df.loc[usgt1_mask, 'feature_transform'].str.extract(r'^([a-zA-Z0-9\.]+_[a-zA-
  # print('usgt1:')
  # display(mod_vi_df.loc[usgt1_mask, 'feature'])

  # try and fix cannot distinguish problems mentioned above
  for us_feat in us_feats:
    mod_vi_df.loc[mod_vi_df['feature_transform'] == us_feat, 'feature'] = us_feat

  wind_mask = mod_vi_df['feature_transform_lag'].str.contains('_window_')
  mod_vi_df.loc[wind_mask, 'feature'] = mod_vi_df.loc[wind_mask, 'feature_transform_lag'].str.extract(r'^(.*?)_window_\d+', e

  return mod_vi_df


def add_shadow_imp_column(mod_vi_df):
```

```
  shad_imp = mod_vi_df.loc[mod_vi_df['feature'] == 'y_des_shadow', ['feature', 'model', 'lag', 'imp']]
  shad_imp = shad_imp.rename(columns={'imp': 'shadow_imp'})

  mod_vi_si = pd.merge(mod_vi_df, shad_imp, how='outer', left_on=['model', 'lag'], right_on=['model', 'lag'])
  mod_vi_si.drop('feature_y', axis=1, inplace=True)
  mod_vi_si.rename(columns={'feature_x': 'feature'}, inplace=True)
  mod_vi_si['shadow_imp'] = mod_vi_si['shadow_imp'].fillna(0)
  mod_vi_si['shadow_imp'] = mod_vi_si['shadow_imp'].astype('int')

  return mod_vi_si


def add_feature_window_transform_column(mod_vi_df):
  mod_vi_df['feature_window_transform'] = mod_vi_df['feature_transform_lag']
  with warnings.catch_warnings():
    warnings.simplefilter(action='ignore', category=FutureWarning)
    mod_vi_df['feature_window_transform'] = mod_vi_df['feature_window_transform'].str.replace('_lag.*', '', regex=True)
    mod_vi_df['feature_window_transform'] = mod_vi_df['feature_window_transform'].str.replace('_(target|pastcov|futcov)', ''
    mod_vi_df['feature_window_transform'] = mod_vi_df['feature_window_transform'].str.replace('_shift_\d+', '', regex=True)

  return mod_vi_df


def add_feature_transform_column(mod_vi_df):
  mod_vi_df['feature_transform'] = mod_vi_df['feature_window_transform']
  with warnings.catch_warnings():
    warnings.simplefilter(action='ignore', category=FutureWarning)
    mod_vi_df['feature_transform'] = mod_vi_df['feature_transform'].str.replace('_window_\d+', '', regex=True)

  return mod_vi_df


def get_multi_model_feat_imps(mmodel, horizon=HORIZON, verbose=False):
  mod_vi_list = []

  for i in range(horizon):
    mod_vi = pd.DataFrame({
        'model': i,
        'feature_transform_lag': mmodel.lagged_feature_names,
        'imp': mmodel.get_multioutput_estimator(i, 0).feature_importances_})
    mod_vi_list.append(mod_vi)

  mod_vi_df = pd.concat(mod_vi_list)
  mod_vi_df.reset_index(drop=True, inplace=True)

  # should have been fixed upstream - my bad
  with warnings.catch_warnings():
    warnings.simplefilter(action='ignore', category=FutureWarning)
    mod_vi_df['feature_transform_lag'] = mod_vi_df['feature_transform_lag'].str.replace(r'^_(.*?)_\1', r'_\1', regex=True)

  mod_vi_df = add_feature_window_transform_column(mod_vi_df)
  mod_vi_df = add_feature_transform_column(mod_vi_df)

  # us_feats - feature names which contain underscores
  us_feats = ['y_des', 'dew.point_des', 't_pot', 'vp_def', 'air_density',
              'ground_hf', 'za_rad', 'azimuth_cos', 'azimuth_sin',
              'rain_prev_6_hours', 'rain_prev_12_hours', 'rain_prev_24_hours',
              'rain_prev_24_hours_binary', 'rain_prev_48_hours',
              'rain_prev_48_hours_binary', 'mixing_ratio', 'specific_humidity',
              'vapour_pressure',]
  mod_vi_df = add_feature_column(mod_vi_df, us_feats)
  mod_vi_df = add_transform_column(mod_vi_df, us_feats)

  mod_vi_df['lag']  = mod_vi_df['feature_transform_lag'].str.extract('_lag(-?\d+)$').fillna(0).astype(int)
  mod_vi_df['type'] = mod_vi_df['feature_transform_lag'].str.extract('_(target|pastcov|futcov)_').fillna('')
  mod_vi_df['shift']  = mod_vi_df['feature_transform_lag'].str.extract('_shift_(\d+)_').fillna(0).astype(int)
  mod_vi_df['window'] = mod_vi_df['feature_transform_lag'].str.extract('_window_(\d+)_').fillna(0).astype(int)

  # very slow (2 or 3 mins) :-(
  mod_vi_df = get_above_between_below_feats_with_model_lag(mod_vi_df, ['dew.point_des', 'pressure', 'humidity'], 'cf')
  # mod_vi_df = get_above_between_below_feats_with_model_lag(mod_vi_df, ['irradiance', 'za_rad'], 'sf')
  # mod_vi_df = get_above_between_below_feats_with_model_lag(mod_vi_df, ['y_des_shadow'], 'shadow')

  mod_vi_si = add_shadow_imp_column(mod_vi_df)

  mod_vi_si['shadow_geq'] = 0
  mod_vi_si.loc[mod_vi_si['shadow_imp'] >= mod_vi_si['imp'], 'shadow_geq'] = 1

  if verbose:
    display(mod_vi_si)

  return mod_vi_si
```

```
# WARN Very slow - 2 or 3 mins :-(
# TODO Speed it up!
def get_above_between_below_feats_with_model_lag(fs, feats, feat_str, imp='imp'):
  # cf - core features
  # sf - solar features

  fs['above_' + feat_str] = 0
  fs['below_' + feat_str] = 0
  if len(feats) > 1:
      fs['between_' + feat_str] = 0

  # groupby model lag feature
  fs_gb = fs[['model', 'lag', 'feature', imp]].groupby(['model', 'lag', 'feature'])

  for name, _ in fs_gb:
    model, lag, feature = name
    min_score = np.min(fs.loc[(fs['model'] == model) & (fs['lag'] == lag) & (fs['feature'].isin(feats)), imp])
    max_score = np.max(fs.loc[(fs['model'] == model) & (fs['lag'] == lag) & (fs['feature'].isin(feats)), imp])

    fs.loc[(fs['model'] == model) & (fs['lag'] == lag) & (fs[imp] > max_score), 'above_' + feat_str] = 1
    fs.loc[(fs['model'] == model) & (fs['lag'] == lag) & (fs[imp] < min_score), 'below_' + feat_str] = 1

    if len(feats) > 1:
      fs.loc[(fs['model'] == model) & (fs['lag'] == lag) & (fs[imp] >= min_score) & (fs[imp] <= max_score), 'between_' + feat

  return fs


# TODO Fix code duplication with plot_multi_model_single_feature_imp
def plot_multi_model_feat_imps(model_vi, title, imp_col='imp', ft_mean_lim=10):
  if title is None:
    title = ''

  n = 10
  display(model_vi.sort_values(imp_col).tail(n))
  display(model_vi.sort_values(imp_col).head(n))

  model_vi.boxplot(imp_col)
  plt.title(title + ' - Variable importance')
  plt.show()

  model_vi.boxplot(imp_col, by='type')
  plt.ylabel(str(imp_col))
  plt.title(title + ' - Feature types')
  plt.suptitle('')
  plt.show()

  bxpm = model_vi.boxplot(imp_col, by='model')
  plt.ylabel(str(imp_col))
  plt.title(title + ' - Models')
  plt.suptitle('')
  n = 2
  plt.setp(bxpm.get_xticklabels()[::n], visible=False)
  plt.show()

  if np.min(model_vi['lag']) < 0:
    bxpl = model_vi.loc[model_vi['lag'] < 0,].boxplot(imp_col, by='lag')
    plt.ylabel(str(imp_col))
    plt.title(title + ' - Past Lags')
    plt.suptitle('')
    plt.show()

  if np.max(model_vi['lag']) > 0:
    bxpl = model_vi.loc[model_vi['lag'] >= 0,].boxplot(imp_col, by='lag')
    plt.ylabel(str(imp_col))
    plt.title(title + ' - Future Lags')
    plt.suptitle('')
    n = 2
    plt.setp(bxpl.get_xticklabels()[::n], visible=False)
    plt.show()

  if model_vi['window'].nunique() >= 2:
    model_vi.boxplot(imp_col, by='window')
    plt.ylabel(str(imp_col))
    plt.title(title + ' - Windows')
    plt.suptitle('')
    plt.show()

  if model_vi['shift'].nunique() >= 2:
    model_vi.boxplot(imp_col, by='shift')
```

```
    plt.ylabel(str(imp_col))
    plt.title(title + ' - Shifts')
    plt.suptitle('')
    plt.show()

  model_vi.boxplot(imp_col, by='feature_transform')
  plt.ylabel(str(imp_col))
  plt.xticks(rotation=45, ha='right')
  plt.title(title + ' - Feature Transform')
  plt.suptitle('')
  plt.show()

  model_vi.loc[model_vi['above_cf'] == 1,].boxplot(imp_col, by='feature')
  plt.xticks(rotation=45, ha='right')
  plt.ylabel(str(imp_col))
  plt.title(title + ' - Features (above core features)')
  plt.suptitle('')
  plt.show()

  model_vi.loc[model_vi['between_cf'] == 1,].boxplot(imp_col, by='feature')
  plt.xticks(rotation=45, ha='right')
  plt.ylabel(str(imp_col))
  plt.title(title + ' - Features (between core features)')
  plt.suptitle('')
  plt.show()

  model_vi.loc[model_vi['above_cf'] == 1,].boxplot(imp_col, by='transform')
  plt.xticks(rotation=45, ha='right')
  plt.ylabel(str(imp_col))
  plt.title(title + ' - Transforms (above core features)')
  plt.suptitle('')
  plt.show()

  model_vi.loc[model_vi['between_cf'] == 1,].boxplot(imp_col, by='transform')
  plt.xticks(rotation=45, ha='right')
  plt.ylabel(str(imp_col))
  plt.title(title + ' - Transforms (between core features)')
  plt.suptitle('')
  plt.show()

  model_vi.boxplot(imp_col, by='feature')
  plt.ylabel(str(imp_col))
  plt.xticks(rotation=45, ha='right')
  plt.title(title + ' - Base Features')
  plt.suptitle('')
  plt.show()

  model_vi.boxplot(imp_col, by='transform')
  plt.ylabel(str(imp_col))
  plt.xticks(rotation=45, ha='right')
  plt.title(title + ' - Transform')
  plt.suptitle('')
  plt.show()


def plot_x_y_importance_interaction(model_vi, xvar, yvar, imp_var, title):
  x = sorted(model_vi[xvar].unique())
  y = sorted(model_vi[yvar].unique())
  X, Y = np.meshgrid(x, y)

  model_vi_grp_min  = model_vi[[yvar, xvar, imp_var]].groupby([yvar, xvar]).min(imp_var)
  model_vi_grp_max  = model_vi[[yvar, xvar, imp_var]].groupby([yvar, xvar]).max(imp_var)
  model_vi_grp_mean = model_vi[[yvar, xvar, imp_var]].groupby([yvar, xvar]).mean(imp_var)

  zs_min  = np.array(model_vi_grp_min[imp_var])
  zs_max  = np.array(model_vi_grp_max[imp_var])
  zs_mean = np.array(model_vi_grp_mean[imp_var])
  Zmin  = zs_min.reshape(X.shape)
  Zmax  = zs_max.reshape(X.shape)
  Zmean = zs_mean.reshape(X.shape)

  fig = plt.figure(figsize=(14, 6))
  ax1 = fig.add_subplot(131, projection='3d')
  ax1.plot_surface(X, Y, Zmin, cmap='viridis')
  ax1.set_xlabel(xvar.title())
  ax1.set_ylabel(yvar.title())
  ax1.set_title(title + ' - min')

  ax2 = fig.add_subplot(132, projection='3d')
  ax2.plot_surface(X, Y, Zmean, cmap='viridis')
  ax2.set_xlabel(xvar.title())
  ax2.set_ylabel(yvar.title())
```

```
  ax2.set_title(title + ' - mean')

  ax3 = fig.add_subplot(133, projection='3d')
  ax3.plot_surface(X, Y, Zmax, cmap='viridis')
  ax3.set_xlabel(xvar.title())
  ax3.set_ylabel(yvar.title())
  ax3.set_title(title + ' - max')
  ax3.set_zlabel(str(imp_var).title())
  plt.show()


def plot_multi_model_importance_interactions(model_vi, title=None):

  plot_x_y_importance_interaction(model_vi, 'model', 'lag', 'imp', title)

  if model_vi['window'].nunique() >= 2:
    plot_x_y_importance_interaction(model_vi, 'model', 'window', 'imp', title)
    plot_x_y_importance_interaction(model_vi.loc[model_vi['lag'] < 0,],
                                    'lag',
                                    'window',
                                    'imp',
                                    title)

  if model_vi['shift'].nunique() >= 2:
    plot_x_y_importance_interaction(model_vi, 'model', 'shift', 'imp', title)
    plot_x_y_importance_interaction(model_vi.loc[model_vi['lag'] < 0,],
                                    'lag',
                                    'shift',
                                    'imp',
                                    title)


def groupby_multi_model_feat_imps(mod_imps, group, verbose=False):
  if verbose:
    print('y_des_shadow:')
    display(mod_imps.loc[mod_imps['feature'] == 'y_des_shadow', 'imp'].describe())

  # mi_gb - model importance group by
  mi_gb_desc = mod_imps[[group, 'imp']].groupby(group).describe()
  mi_gb_desc = mi_gb_desc.droplevel(axis=1, level=0)
  mi_gb_desc = mi_gb_desc.astype({'count': 'int', 'min': 'int', '25%': 'int',
                                  '50%': 'int', '75%': 'int', 'max': 'int'})
  if verbose:
    display(mi_gb_desc)


  mi_gb_sum = mod_imps[[group, 'imp']].groupby(group).sum()
  mi_gb_sum = mi_gb_sum.rename(columns={'imp': 'sum'})
  if verbose:
    print('\nsum:')
    display(mi_gb_sum)

  mi_gb_desc_sum = pd.merge(mi_gb_desc, mi_gb_sum, left_index=True, right_index=True)


  mi_gb_0imp = mod_imps.loc[mod_imps['imp'] == 0].groupby(group).size().to_frame()
  mi_gb_0imp = mi_gb_0imp.rename(columns={0: 'num_0'})
  if verbose:
    print('\nnum 0 imp:')
    display(mi_gb_0imp.sort_values('num_0'))


  mi_gb_desc_sum_0imp = pd.merge(mi_gb_desc_sum, mi_gb_0imp, left_index=True, right_index=True)
  mi_gb_desc_sum_0imp['pc_0'] = mi_gb_desc_sum_0imp['num_0'] * 100 / mi_gb_desc_sum_0imp['count']
  # display(mi_gb_desc_sum_0imp)


  mi_gb_shad_geq = mod_imps[[group, 'shadow_geq']].groupby(group).sum()
  mi_gb_shad_geq = mi_gb_shad_geq.rename(columns={'shadow_geq': 'num_shad_geq'})
  if verbose:
    print('\nnum_shad_geq:')
    display(mi_gb_shad_geq)

  mi_gb_all = pd.merge(mi_gb_desc_sum_0imp, mi_gb_shad_geq, left_index=True, right_index=True)
  mi_gb_all['pc_shad_geq'] = mi_gb_all['num_shad_geq'] * 100 / mi_gb_all['count']
  # display(mi_gb_all)

  return mi_gb_all


def plot_feat_imp_cumsum(mod_vi, title):
  mod_impcs = mod_vi[['model', 'imp']].groupby('model').apply(lambda grp: grp.imp.sort_values(ascending=False).cumsum().reset
```

```
  mod_impcs = mod_impcs.melt(ignore_index=False).reset_index().rename(columns={'imp': 'order', 'value': 'imp_cumsum'})

  xvar = 'order'
  yvar = 'model'
  fig = plt.figure(figsize=(14, 6))
  ax1 = fig.add_subplot(121, projection='3d')
  ax1.plot_trisurf(mod_impcs[xvar], mod_impcs[yvar], mod_impcs['imp_cumsum'],
                   cmap='viridis', linewidth=0)

  ax1.set_xlabel(xvar.title())
  ax1.set_ylabel(yvar.title())
  ax1.set_title(title)

  ax2 = fig.add_subplot(122, projection='3d')
  ax2.plot_trisurf(mod_impcs[yvar], mod_impcs[xvar], mod_impcs['imp_cumsum'],
                   cmap='viridis', linewidth=0)

  ax2.set_xlabel(yvar.title())
  ax2.set_ylabel(xvar.title())
  ax2.set_zlabel('imp_cumsum')
  ax2.set_title(title)
  plt.show()


# TODO Fix code duplication with plot_multi_model_feat_imps
def plot_multi_model_single_feature_imp(model_vi, col_name, col_value, title, imp_col='imp'):
  if title is None:
    title = col_value
  else:
    title = title + ' ' + col_value

  model_vi = model_vi.loc[model_vi[col_name] == col_value,]

  model_vi.boxplot(imp_col)
  plt.title(title + ' - Variable importance')
  plt.show()

  if model_vi['type'].nunique() >= 2:
    model_vi.boxplot(imp_col, by='type')
    plt.ylabel(str(imp_col))
    plt.title(title + ' - Feature types')
    plt.suptitle('')
    plt.show()

  bxpm = model_vi.boxplot(imp_col, by='model')
  plt.ylabel(str(imp_col))
  plt.title(title + ' - Models')
  plt.suptitle('')
  n = 2
  plt.setp(bxpm.get_xticklabels()[::n], visible=False)
  plt.show()

  # palette = ['red', 'green']
  # box_cols = ['y_des_window_24_hist_mode', 'y_des_shadow']
  # model3_vi_bp = model3_vi.loc[model3_vi['feature'].isin(box_cols)]
  # sns.boxplot(data=model3_vi_bp, x='lag', y='imp', hue='feature',
  #             gap=.2, palette=palette, fill=False, linewidth=.75)
  # plt.show()

  if np.min(model_vi['lag']) < 0:
    bxpl = model_vi.loc[model_vi['lag'] < 0,].boxplot(imp_col, by='lag')
    plt.ylabel(str(imp_col))
    plt.title(title + ' - Past Lags')
    plt.suptitle('')
    plt.show()

  if np.max(model_vi['lag']) > 0:
    bxpl = model_vi.loc[model_vi['lag'] >= 0,].boxplot(imp_col, by='lag')
    plt.ylabel(str(imp_col))
    plt.title(title + ' - Future Lags')
    plt.suptitle('')
    n = 2
    plt.setp(bxpl.get_xticklabels()[::n], visible=False)
    plt.show()

  if model_vi['window'].nunique() >= 2:
    model_vi.boxplot(imp_col, by='window')
    plt.ylabel(str(imp_col))
    plt.title(title + ' - Windows')
    plt.suptitle('')
    plt.show()
```

```
  if model_vi['shift'].nunique() >= 2:
    model_vi.boxplot(imp_col, by='shift')
    plt.ylabel(str(imp_col))
    plt.title(title + ' - Shifts')
    plt.suptitle('')
    plt.show()

  if model_vi['transform'].nunique() >= 2:
    model_vi.boxplot(imp_col, by='transform')
    plt.ylabel(str(imp_col))
    plt.title(title + ' - Transforms')
    plt.suptitle('')
    plt.show()


  xvar = 'model'
  yvar = 'lag'

  fig = plt.figure(figsize=(14, 7))
  ax1 = fig.add_subplot(121, projection='3d')
  ax1.plot_trisurf(model_vi[xvar], model_vi[yvar], model_vi[imp_col],
                   cmap='viridis', linewidth=0)
  ax1.set_xlabel(xvar.title())
  ax1.set_ylabel(yvar.title())
  ax1.set_title(title)

  ax2 = fig.add_subplot(122, projection='3d')
  ax2.plot_trisurf(model_vi[yvar], model_vi[xvar], model_vi[imp_col],
                   cmap='viridis', linewidth=0)
  ax2.set_xlabel(yvar.title())
  ax2.set_ylabel(xvar.title())
  # ax2.zaxis.set_rotate_label(False)  # disable automatic rotation
  ax2.set_zlabel(str(imp_col))  #, rotation=-90)
  ax2.set_title(title)
  plt.show()
  print('\n\n')


def load_features_file(feature_set,
                       data_set,
                       location = 'gdrive',
                       date_str = '.2022.09.20',
                       filex    = '.parquet'):

  if location == 'github':
    base_url = 'https://github.com/makeyourownmaker/CambridgeTemperatureNotebooks/blob/main/data/features/'
    filex += '?raw=true'
  elif location == 'gdrive':
    base_url = '/content/drive/MyDrive/data/CambridgeTemperatureNotebooks/features/'
  else:
    print("Unsupported 'location' in load_features_file function:")
    print('  location =', location)

  file_str = feature_set + '_' + data_set + date_str + filex
  data_url = base_url + file_str

  df = pd.read_parquet(data_url)

  df.set_index('ds', drop=False, inplace=True)
  df = df[~df.index.duplicated(keep='first')]
  df = df.asfreq(freq='30min')

  return df


def load_train_valid_test_features(feature_set, location='gdrive'):
  train = load_features_file(feature_set, 'train', location)
  valid = load_features_file(feature_set, 'valid', location)
  test  = load_features_file(feature_set, 'test',  location)

  sanity_check_train_valid_test(train, valid, test)

  check_high_low_thresholds(train, 'train '+feature_set)
  check_high_low_thresholds(valid, 'valid '+feature_set)
  check_high_low_thresholds(test,   'test '+feature_set)

  return train, valid, test


def get_feature_selection_data(df, sel_cols, y_col, fs_lags, pred_step):
  feat_cols = df.columns.to_list()
```

```
  excludes = ['_ucm_', '_yhat', '_diff_', '_yearly', '_daily', '_trend',
              'humidity_des', 'pressure_des', 'ds', 'spike', 'tsclean',
              'day.', 'year', 'rain', 'wind.', '.log', 'dy_dh', 'dy_dp',
              'missing', 'dT_dH', 'dT_dP', 'dT_dTdp', 'known_inaccuracy',
              'isd_', 'long_run', 'cooksd_out', 'tau'
              ]
  feat_cols = [feat_col for feat_col in feat_cols if all([x not in feat_col for x in excludes])]

  feat_cols.extend(sel_cols)
  feat_cols = list(set(feat_cols))

  feat_cols.remove(y_col)
  feat_cols.remove('y')
  feat_cols.remove('dew.point')

  all_cols = [*feat_cols, y_col]
  df_nona = df[all_cols].dropna()

  # Add lagged features
  if fs_lags is not None:
    # PerformanceWarning: DataFrame is highly fragmented ...
    # df_nona_lagged = df_nona.assign(**{
    #   f'{col}_lag_{lag}': df_nona[col].shift(lag)
    #   for lag in fs_lags
    #   for col in feat_cols})
    df_nona_lagged_only = pd.DataFrame({
       f'{col}_lag_{lag}': df_nona[col].shift(lag)
       for lag in fs_lags
       for col in feat_cols})
    df_nona_lagged_plus = pd.concat([df_nona, df_nona_lagged_only], axis=1)
    df_nona_lagged_plus = df_nona_lagged_plus.dropna()
    X_df = df_nona_lagged_plus.drop(y_col, axis=1)
    y_df = df_nona_lagged_plus[[y_col]]
  else:
    X_df = df_nona[feat_cols]
    y_df = df_nona[[y_col]]

  y_df = y_df[y_col].shift(-pred_step).dropna()
  X_df = X_df.head(y_df.shape[0])

  return X_df, y_df


def get_feature_selection_scores(df, sel_cols, lags=None, pred_step=0,
                                 y_col=Y_COL, sort_col='f_test', mi=False):
  '''WARNING: These tests assume a linear model.  This may not be optimal.

     Don't draw any hasty conclusions from these scores.
  '''

  X_df, y_df = get_feature_selection_data(df, sel_cols, y_col, lags, pred_step)
  feat_cols = X_df.columns

  f_tests, _ = f_regression(X_df, y_df)
  f_tests /= np.sum(f_tests)

  r_tests = r_regression(X_df, y_df)
  r_tests /= np.sum(r_tests)

  fs_df = pd.DataFrame({
                        'r_test': r_tests.round(6),
                        'f_test': f_tests.round(6),
                       })
  fs_df.index = feat_cols

  # Slow :-(
  if mi:
    mi_feats = mutual_info_regression(X_df, y_df)
    mi_feats /= np.sum(mi_feats)
    fs_df['mi'] = mi_feats

  if sort_col == 'f_test':
    fs_df = fs_df.sort_values('f_test', ascending=False)
  elif sort_col == 'r_test':
    fs_df = fs_df.sort_values('r_test', ascending=False)
  elif mi is True and sort_col == 'mi':
    fs_df = fs_df.sort_values('mi', ascending=False)

  return fs_df


def get_above_between_below_features(fs, feats, feat_str, imp='f_test'):
```

```
  # cf - core features
  # sf - solar features

  fs['above_' + feat_str] = 0
  fs['below_' + feat_str] = 0

  min_score = np.min(fs.loc[feats, imp])
  max_score = np.max(fs.loc[feats, imp])

  fs.loc[fs[imp] > max_score, 'above_' + feat_str] = 1
  fs.loc[fs[imp] < min_score, 'below_' + feat_str] = 1

  if len(feats) > 1:
    fs['between_' + feat_str] = 0
    fs.loc[(fs[imp] >= min_score) & (fs[imp] <= max_score), 'between_' + feat_str] = 1

  return fs


def get_multi_step_feat_sel_scores(train, sel_cols, lags=None, horizon=48):
  fs_steps_list = []

  for i in range(horizon):
    fs_df = get_feature_selection_scores(train, sel_cols, lags, pred_step=i)
    fs_df['rank'] = [i for i in range(fs_df.shape[0])]
    fs_df['step'] = i

    if lags is None:
      fs_df['feature_transform'] = fs_df.index
    else:
      fs_df['feature_transform_lag'] = fs_df.index

    fs_df.drop('r_test', axis=1, inplace=True)
    fs_df = get_above_between_below_features(fs_df, ['dew.point_des', 'pressure', 'humidity'], 'cf')
    fs_df = get_above_between_below_features(fs_df, ['irradiance', 'za_rad'], 'sf')
    fs_df = get_above_between_below_features(fs_df, ['y_des_shadow'], 'shadow')
    fs_steps_list.append(fs_df)

  fs_steps = pd.concat(fs_steps_list)

  # add window, shift, transform, feature ...

  if lags is not None:
    fs_steps['feature_transform'] = fs_steps['feature_transform_lag']

    with warnings.catch_warnings():
      warnings.simplefilter(action='ignore', category=FutureWarning)
      fs_steps['feature_transform'] = fs_steps['feature_transform'].str.replace('_lag.*', '', regex=True)

    fs_steps['lag'] = fs_steps['feature_transform_lag'].str.extract('_lag(-?\d+)$').fillna(0).astype(int)
    fs_feats = ['feature_transform_lag', 'feature_transform', 'feature',
                'transform', 'window', 'shift', 'lag', 'step', 'f_test',
                'rank', 'above_cf', 'between_cf', 'below_cf', 'above_sf',
                'between_sf', 'below_sf', 'above_shadow', 'below_shadow']
  else:
    fs_steps['lag'] = 0
    fs_feats = ['feature_transform', 'feature', 'transform', 'window', 'shift',
                'lag', 'step', 'f_test', 'rank', 'above_cf', 'between_cf',
                'below_cf', 'above_sf', 'between_sf', 'below_sf',
                'above_shadow', 'below_shadow']

  fs_steps['shift']  = fs_steps['feature_transform'].str.extract('_shift_(\d+)_').fillna(0).astype(int)
  fs_steps['window'] = fs_steps['feature_transform'].str.extract('_window_(\d+)_').fillna(0).astype(int)

  fs_steps['transform'] = fs_steps['feature_transform'].str.extract('_window_\d+_([a-zA-Z0-9\._]+)$').fillna('None')

  fs_steps.loc[fs_steps['transform'].str.match('intervals_intervals_mean'), 'transform'] = 'intervals_mean'

  fs_steps.loc[fs_steps['feature_transform'].str.endswith('_shadow'), 'transform'] = 'shadow'
  fs_steps.loc[fs_steps['feature_transform'].str.endswith('_grad'), 'transform'] = 'grad'
  fs_steps.loc[fs_steps['feature_transform'].str.endswith('_des'), 'transform'] = 'des'

  fs_steps['feature'] = fs_steps['feature_transform'].str.extract('([a-zA-Z0-9\.]+)_').fillna('None')
  fs_steps.loc[fs_steps['transform'] == 'None', 'feature'] = fs_steps.loc[fs_steps['transform'] == 'None', 'feature_transforı

  fs_steps = fs_steps[fs_feats]

  return fs_steps


def groupby_multi_step_feat_sel_scores(mod_imps, group, verbose=False):
  if verbose:
```

```
    print('y_des_shadow:')
    display(mod_imps.loc[mod_imps['feature_transform'] == 'y_des_shadow', 'f_test'].describe())

  # mi_gb - model importance group by
  mi_gb_desc = mod_imps[[group, 'f_test']].groupby(group).describe()
  mi_gb_desc = mi_gb_desc.droplevel(axis=1, level=0)
  mi_gb_desc = mi_gb_desc.astype({'count': 'int'})#, 'min': 'int', '25%': 'int',
                                  #'50%': 'int', '75%': 'int', 'max': 'int'})
  if verbose:
    display(mi_gb_desc)


  mi_gb_sum = mod_imps[[group, 'f_test']].groupby(group).sum()
  mi_gb_sum = mi_gb_sum.rename(columns={'f_test': 'sum'})
  if verbose:
    print('\nsum:')
    display(mi_gb_sum)

  mi_gb_desc_sum = pd.merge(mi_gb_desc, mi_gb_sum, left_index=True, right_index=True)


  mi_gb_0imp = mod_imps.loc[mod_imps['f_test'] == 0.0].groupby(group).size().to_frame()
  mi_gb_0imp = mi_gb_0imp.rename(columns={0: 'num_0'})
  if verbose:
    print('\nnum 0 f_test:')
    display(mi_gb_0imp.sort_values('num_0'))


  mi_gb_desc_sum_0imp = pd.merge(mi_gb_desc_sum, mi_gb_0imp, how='outer', left_index=True, right_index=True)
  mi_gb_desc_sum_0imp['num_0'] = mi_gb_desc_sum_0imp['num_0'].fillna(0)
  mi_gb_desc_sum_0imp['pc_0'] = mi_gb_desc_sum_0imp['num_0'] * 100 / mi_gb_desc_sum_0imp['count']
  # display(mi_gb_desc_sum_0imp)

  return mi_gb_desc_sum_0imp


def plot_multi_step_feat_sel_scores(model_vi, title, imp_col='f_test'):
  if title is None:
    title = ''

  model_vi.boxplot(imp_col)
  plt.ylabel(str(imp_col))
  plt.title(title + ' - Variable importance')
  plt.suptitle('')
  plt.show()

  bxpm = model_vi.boxplot(imp_col, by='step')
  plt.ylabel(str(imp_col))
  plt.title(title + ' - forecast step')
  plt.suptitle('')
  n = 2
  plt.setp(bxpm.get_xticklabels()[::n], visible=False)
  plt.show()

  if model_vi['window'].nunique() >= 2:
    model_vi.boxplot(imp_col, by='window')
    plt.ylabel(str(imp_col))
    plt.title(title + ' - Windows')
    plt.suptitle('')
    plt.show()

  if model_vi['shift'].nunique() >= 2:
    model_vi.boxplot(imp_col, by='shift')
    plt.ylabel(str(imp_col))
    plt.title(title + ' - Shifts')
    plt.suptitle('')
    plt.show()

  model_vi.loc[model_vi['above_cf'] == 1,].boxplot(imp_col, by='feature_transform')
  plt.xticks(rotation=45, ha='right')
  plt.ylabel(str(imp_col))
  plt.title(title + ' - Features (above core features)')
  plt.suptitle('')
  plt.show()

  model_vi.loc[model_vi['between_cf'] == 1,].boxplot(imp_col, by='feature_transform')
  plt.xticks(rotation=45, ha='right')
  plt.ylabel(str(imp_col))
  plt.title(title + ' - Features (between core features)')
  plt.suptitle('')
  plt.show()
```

```
  model_vi.boxplot(imp_col, by='feature')
  plt.xticks(rotation=45, ha='right')
  plt.ylabel(str(imp_col))
  plt.title(title + ' - Base Feature')
  plt.suptitle('')
  plt.show()

  model_vi.boxplot(imp_col, by='transform')
  plt.xticks(rotation=45, ha='right')
  plt.ylabel(str(imp_col))
  plt.title(title + ' - Transform')
  plt.suptitle('')
  plt.show()

  if model_vi['lag'].nunique() >= 2:
    model_vi.boxplot(imp_col, by='lag')
    plt.ylabel(str(imp_col))
    plt.title(title + ' - Lag')
    plt.suptitle('')
    plt.show()

  if model_vi['step'].nunique() >= 2 and model_vi['lag'].nunique() >= 2:
    plot_x_y_importance_interaction(model_vi, 'step', 'lag', 'f_test', title)

  if model_vi['window'].nunique() >= 2 and model_vi['lag'].nunique() >= 2:
    plot_x_y_importance_interaction(model_vi, 'window', 'lag', 'f_test', title)

  if model_vi['step'].nunique() >= 2 and model_vi['window'].nunique() >= 2:
    plot_x_y_importance_interaction(model_vi, 'step', 'window', 'f_test', title)


def summarise_multi_step_feat_sel_scores(train, title, sel_cols, fs_lags=None, ft_mean_lim=0.01, verbose=False):
  fs = get_multi_step_feat_sel_scores(train, sel_cols, lags=fs_lags)

  if verbose:
    display(fs)

  if fs_lags is not None:
    print('feature_transform_lag - ft_mean_lim = ' + str(ft_mean_lim) + ':')
    fs_ftl = groupby_multi_step_feat_sel_scores(fs, 'feature_transform_lag')
    display(fs_ftl.loc[fs_ftl['mean'] >= ft_mean_lim,].sort_values('mean', ascending=False))
    print('lag:')
    fs_lag = groupby_multi_step_feat_sel_scores(fs, 'lag')
    display(fs_lag.sort_values('mean', ascending=False))
  else:
    print('feature_transform - ft_mean_lim = ' + str(ft_mean_lim) + ':')
    fs_ft = groupby_multi_step_feat_sel_scores(fs, 'feature_transform')
    display(fs_ft.loc[fs_ft['mean'] >= ft_mean_lim,].sort_values('mean', ascending=False))

  if verbose:
    fs_feature = groupby_multi_step_feat_sel_scores(fs, 'feature')
    print('feature:')
    display(fs_feature)

    fs_transform = groupby_multi_step_feat_sel_scores(fs, 'transform')
    print('transform:')
    display(fs_transform)

    fs_window = groupby_multi_step_feat_sel_scores(fs, 'window')
    print('window:')
    display(fs_window)

  plot_multi_step_feat_sel_scores(fs, title)


def summarise_multi_model_feat_imps(model_vi, title, verbose=False):
  if verbose:
    display(model_vi)

    for gb_col in ['model', 'lag', 'type', 'shift', 'window', 'feature', 'transform']:
      if model_vi[gb_col].nunique() >= 2:
        vi_by_gb = groupby_multi_model_feat_imps(model_vi, gb_col)
        display(vi_by_gb); print()

  plot_multi_model_feat_imps(model_vi, title)
  plot_multi_model_importance_interactions(model_vi, title)

  if verbose:
    plot_feat_imp_cumsum(model_vi, title)
```

Data Setup

Load Pre-calculated Features

All the feature files combined are too large for the git LFS free-usage tier. Features are stored in parquet files on either github or google drive. The google drive files are not publicly available. Train, valid and test files are provided for each feature set.

There are quite a few missing values around the beginning of 2016. Earlier work calculating first differences for the core features showed there were substantial differences before early 2016 and afterwards. So, training data starts after early 2016. Summary of feature generation runtime and number of features in each feature set.

Feature set	Run time	Features
meteorology-based	0 mins	9
solar-based	2 mins	4
default	0 mins	109
roll_stats	1 min	111
cross_corr	1 min	29
catch22	12 mins	305
tsfeatures	1 hour 30 mins	246
intervals_etc	50 mins	40
bivariate	6 hours 15 mins	103
pairwise	18 mins	88
---	---	---
Total	9 hours 10 mins	1,031

Linear interpolation was used to fill sequences of 12 (6 hours) or less consecutive NAs. Some features were removed due to longer sequences of NAs. 12 is a somewhat arbitrary cutoff. There are probably a few redundant features between feature sets.

So far, most of the meteorology-based features have not proven useful. Saturation vapor pressure, `svp`, is a possible exception. This is a bit surprising. The solar-based calculated features (`irradiance`, `za_rad`) used as future covariates in the darts lightGBM notebook proved beneficial. The meteorology-based and solar-based features are included in the default feature set which also includes seasonal decompositions, wind related features and some other experiments. The meteorology-based and solar-based features are not available separately.

The `intervals_mean`, and to a lesser extent `pacf`, features from the tsfeatures package (intervals_etc feature set) were very useful in the darts lightGBM notebook. The pacf-based features are not that surprising. Further investigation may be justified for the intervals-based features. It seems to be an indication of positive demand.

Unfortunately, when combined these feature sets exceed the git LFS free usage tier. The default and cross_corr datasets are available on github repo because they are below the 50 MB limit.

See feature_engineering.ipynb notebook for further details.

Load default features:

```
train_df, valid_df, test_df = load_train_valid_test_features('default', location='github')

print('train_df:')
print_df_summary(train_df)
plot_cols = ['y', 'humidity', 'dew.point', 'wind.speed.mean.x',
             'wind.speed.mean.y']  # 'pressure',
plot_observation_examples(train_df, plot_cols)
```

```
WARN: high overlap between train_df and valid_df rows!
Number of shared rows: 5231
Approximate overlap: 29.86 %

WARN: high overlap between train_df and test_df rows!
Number of shared rows: 5707
Approximate overlap: 32.57 %

train_df:
Shape:
(87168, 109)

Total NAs: 0
Rows with NAs: 0
Cols with NAs: 0

Info:
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 87168 entries, 2016-01-12 00:00:00 to 2020-12-31 23:30:00
Freq: 30T
Columns: 109 entries, ds to y_des_shadow
dtypes: datetime64[us](1), float64(89), int64(19)
memory usage: 73.2 MB
None

Summary stats:
```

	ds	y	humidity	dew.point	pressure	pressure.log	y_window_48_min_max_diff	wind.speed.mean.s
count	87168	87168.000000	87168.000000	87168.000000	87168.000000	87168.000000	87168.000000	87168.00(
mean	2018-07-07 23:45:00	9.652911	77.762962	5.355061	1014.848902	6.922424	8.432414	1.709
min	2016-01-12 00:00:00	-6.800000	7.000000	-10.000000	966.000000	6.873164	0.800000	0.00(
25%	2017-04-09 23:52:30	4.800000	67.000000	1.500000	1008.000000	6.915723	5.700000	0.836
50%	2018-07-07 23:45:00	8.800000	82.000000	5.300000	1016.000000	6.923629	8.000000	1.732
75%	2019-10-04 23:37:30	14.100000	92.000000	9.100000	1023.000000	6.930495	10.800000	2.509
max	2020-12-31 23:30:00	36.100000	100.000000	20.900000	1051.000000	6.957497	21.100000	5.403
std	NaN	6.503256	18.134399	5.108536	12.102744	0.011968	3.659642	1.07

8 rows × 109 columns

```
Raw data:
```

	ds	y	humidity	dew.point	pressure	pressure.log	y_window_48_min_max_diff	wind.speed.mean.sqrt	wind.spe
ds									
2016-01-12 00:00:00	2016-01-12 00:00:00	1.6	96.0	1.0	986.0	6.893656	4.422576	2.000000	
2016-01-12 00:30:00	2016-01-12 00:30:00	2.0	94.0	1.1	986.0	6.893656	4.422576	2.428992	
2016-01-12 01:00:00	2016-01-12 01:00:00	2.8	87.0	0.9	987.0	6.894670	4.422576	2.549510	
2016-01-12 01:30:00	2016-01-12 01:30:00	2.0	93.0	1.0	987.0	6.894670	4.422576	2.408319	
2016-01-12 02:00:00	2016-01-12 02:00:00	2.4	89.0	0.8	987.0	6.894670	4.422576	2.626785	
...	...	...	...	...	...	...	...	...	
2020-12-31 21:30:00	2020-12-31 21:30:00	-2.8	96.0	-3.3	1006.0	6.913737	3.200000	0.000000	
2020-12-31 22:00:00	2020-12-31 22:00:00	-3.2	100.0	-3.2	1007.0	6.914731	3.200000	0.000000	

2020-12-31 22:30:00	2020-12-31 22:30:00	-3.6	100.0	-3.6	1007.0	6.914731	3.200000	0.000000
2020-12-31 23:00:00	2020-12-31 23:00:00	-4.4	97.0	-4.8	1007.0	6.914731	3.200000	0.000000
2020-12-31 23:30:00	2020-12-31 23:30:00	-4.8	99.0	-4.9	1007.0	6.914731	3.200000	0.632456

87168 rows × 109 columns

The `pressure` feature is not included in the 24-hour example plots. It takes values from approximately 950 to 1,050 mbar. Plotting it with the other features is not helpful.

Feature engineering experiments

Add following features to default feature set.

Calculations (transforms):

- gradient
- standard deviation of gradient

- histogram mode
- peak to peak range

Variables:

- `y_des`
- `dew.point_des`
- `humidity`
- `pressure`

Windows:

- 12, 24, 36, 48, 60, 72, 84, 96

```
from numba import jit


def add_fast_p2p(data, var, window):
  new_name = var + '_window_' + str(window) + '_min_max_diff'
  data[new_name] = data[var].shift(1).rolling(window, min_periods=1).agg(['min', 'max']).diff(axis=1)['max']
  return data


@jit(nopython=True)
def histogram_mode(x, nbins = 10):
    """
    Measures the mode of the data vector using histograms with a given number of bins.
    Reference: https://cran.r-project.org/web/packages/tsfeatures/vignettes/tsfeatures.html

    Args:
        x: The univariate time series array in the form of 1d numpy array.
        nbins: int; Number of bins to get the histograms. Default value is 10.

    Returns:
        Mode of the data vector using histograms.
    """

    cnt, val = np.histogram(x, bins=nbins)
    return val[cnt.argmax()]


def _add_grad_hist_mode_p2p_feats(data):
  aggs_l = []  # Prevents highly fragmented dataframes

  for col in ['y_des', 'dew.point_des', 'humidity', 'pressure']:
    agg = pd.DataFrame(index=data.index)  # Prevents highly fragmented dataframes
    agg[col+'_grad'] = np.gradient(data[col])

    for window in [12, 24, 36, 48, 60, 72, 84, 96]:
      col_str = col + '_window_' + str(window) + '_'
      agg[col_str+'grad_std']  = agg[col+'_grad'].shift(1).rolling(window, min_periods=1).std()
      agg[col_str+'hist_mode'] = data[col].shift(1).rolling(window, min_periods=1).apply(histogram_mode, raw=True)
      data = add_fast_p2p(data, col, window)

    aggs_l.append(agg)

  aggs = pd.concat(aggs_l, axis=1)
  aggs = aggs.fillna(method='bfill')  # Small number of NAs in first row
  data = data.fillna(method='bfill')  # Small number of NAs in first row

  data = pd.concat((data, aggs), axis=1)

  # data.set_index('ds', drop = False, inplace = True)
  data['ds'] = data.index
  data = data[~data.index.duplicated(keep = 'first')]
  data = data.asfreq(freq = '30min')

  return data


def add_grad_hist_mode_p2p_feats(train, valid, test, title_str):
  train_orig = train.copy(deep=True)
  valid_orig = valid.copy(deep=True)
  test_orig  = test.copy(deep=True)

  train = _add_grad_hist_mode_p2p_feats(train)
  valid = _add_grad_hist_mode_p2p_feats(valid)
  test  = _add_grad_hist_mode_p2p_feats(test)

  # Running sanity checks takes longer than calculating features!
  sanity_check_train_valid_test(train, valid, test)
```

```
  train_sanity = sanity_check_before_after_dfs(train_orig, train, 'train_'+title_str)
  valid_sanity = sanity_check_before_after_dfs(valid_orig, valid, 'valid_'+title_str)
  test_sanity  = sanity_check_before_after_dfs(test_orig,  test,  'test_'+title_str)

  compare_train_valid_test_sanity_dfs(train_sanity, valid_sanity, test_sanity)

  check_high_low_thresholds(train)
  check_high_low_thresholds(valid)
  check_high_low_thresholds(test)

  return train, valid, test


train_df, valid_df, test_df = add_grad_hist_mode_p2p_feats(train_df, valid_df, test_df, 'df')
```

```
WARN: high overlap between train_df and valid_df rows!
Number of shared rows: 5231
Approximate overlap: 29.86 %

WARN: high overlap between train_df and test_df rows!
Number of shared rows: 5707
Approximate overlap: 32.57 %


train_df
before.index.equals(after.index): True
before.index.freq == after.index.freq: True
before[common_cols].dtypes == after[common_cols].dtypes: False
before[common_cols].describe() == after[common_cols].describe(): False

before[common_cols].equals(after[common_cols]): False
redundancy before > after: False
mean before feature redundancy: 46.212
mean after feature redundancy:  68.632
```

	before	after	diff
rows	87168	87168	0
cols	109	209	100
missing_rows	0	0	0
missing_cols	0	0	0
total_nas	0	0	0
rows_with_nas	0	0	0
cols_with_nas	0	0	0
single_value_cols	4	4	0
low_var_rows	0	0	0
low_var_cols	17	17	0
duplicate_rows	0	0	0
duplicate_index_labels	0	0	0
duplicate_col_labels	0	0	0

```
valid_df
before.index.equals(after.index): True
before.index.freq == after.index.freq: True
before[common_cols].dtypes == after[common_cols].dtypes: False
before[common_cols].describe() == after[common_cols].describe(): False

before[common_cols].equals(after[common_cols]): False
redundancy before > after: False
mean before feature redundancy: 45.973
mean after feature redundancy:  68.404
```

	before	after	diff
rows	17520	17520	0
cols	109	209	100
missing_rows	0	0	0
missing_cols	0	0	0
total_nas	0	0	0
rows_with_nas	0	0	0
cols_with_nas	0	0	0
single_value_cols	4	4	0
low_var_rows	0	0	0
low_var_cols	18	18	0
duplicate_rows	0	0	0
duplicate_index_labels	0	0	0
duplicate_col_labels	0	0	0

```
test_df
before.index.equals(after.index): True
before.index.freq == after.index.freq: True
before[common_cols].dtypes == after[common_cols].dtypes: False
before[common_cols].describe() == after[common_cols].describe(): False

before[common_cols].equals(after[common_cols]): False
```

```
before[common_cols].equals(after[common_cols]): False
redundancy before > after: False
mean before feature redundancy: 45.925
mean after feature redundancy:  68.441
```

	before	after	diff
rows	17520	17520	0
cols	109	209	100
missing_rows	0	0	0
missing_cols	0	0	0
total_nas	0	0	0
rows_with_nas	0	0	0
cols_with_nas	0	0	0
single_value_cols	5	5	0
low_var_rows	0	0	0
low_var_cols	17	17	0
duplicate_rows	0	0	0
duplicate_index_labels	0	0	0
duplicate_col_labels	0	0	0

```
WARN: train_sanity != valid_sanity
```

	before	after	diff
low_var_cols	17	17	0
low_var_cols	18	18	0

```
WARN: train_sanity != test_sanity
```

	before	after	diff
single_value_cols	4	4	0
single_value_cols	5	5	0

```
WARN: test_sanity != valid_sanity
```

	before	after	diff
single_value_cols	5	5	0
low_var_cols	17	17	0
single_value_cols	4	4	0
low_var_cols	18	18	0

Added 100 additional features to each of the default feature sets (`train_df`, `valid_df`, `test_df`).

Load additional feature sets

Currently only loading the intervals_etc data sets.

- intervals_etc
  - intervals_mean features
  - pacf features

The remaining data sets were loaded at the feature comparisons stage. Links to relevant commits are provided later.

```
# train_pair, valid_pair, test_pair = load_train_valid_test_features('pairwise')
# print('train_pair:')
# print_df_summary(train_pair)

# train_ccorr, valid_ccorr, test_ccorr = load_train_valid_test_features('cross_corr', location='github')
# print('train_ccorr:')
# print_df_summary(train_ccorr)

# train_stats, valid_stats, test_stats = load_train_valid_test_features('roll_stats')
# print('train_stats:')
# print_df_summary(train_stats)

# train_biv, valid_biv, test_biv = load_train_valid_test_features('bivariate')
# print('train_biv:')
# print_df_summary(train_biv)
```

```
# train_c22, valid_c22, test_c22 = load_train_valid_test_features('catch22')
# print('train_c22:')
# print_df_summary(train_c22)

# train_tsf, valid_tsf, test_tsf = load_train_valid_test_features('tsfeatures')
# print('train_tsf:')
# print_df_summary(train_tsf)

train_ints, valid_ints, test_ints = load_train_valid_test_features('intervals_etc')
print('train_ints:')
print_df_summary(train_ints)
```

```
WARN: high overlap between train_df and valid_df rows!
Number of shared rows: 5231
Approximate overlap: 29.86 %

WARN: high overlap between train_df and test_df rows!
Number of shared rows: 5707
Approximate overlap: 32.57 %

train_ints:
Shape:
(87168, 149)

Total NAs: 0
Rows with NAs: 0
Cols with NAs: 0

Info:
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 87168 entries, 2016-01-12 00:00:00 to 2020-12-31 23:30:00
Freq: 30T
Columns: 149 entries, ds to y_des_window_96_pacf_features_diff2x_pacf5
dtypes: datetime64[us](1), float64(129), int64(19)
memory usage: 99.8 MB
None

Summary stats:
```

	ds	y	humidity	dew.point	pressure	pressure.log	y_window_48_min_max_diff	wind.speed.mean.s
count	87168	87168.000000	87168.000000	87168.000000	87168.000000	87168.000000	87168.000000	87168.00(
mean	2018-07-07 23:45:00	9.652911	77.762962	5.355061	1014.848902	6.922424	8.432414	1.709
min	2016-01-12 00:00:00	-6.800000	7.000000	-10.000000	966.000000	6.873164	0.800000	0.00(
25%	2017-04-09 23:52:30	4.800000	67.000000	1.500000	1008.000000	6.915723	5.700000	0.836
50%	2018-07-07 23:45:00	8.800000	82.000000	5.300000	1016.000000	6.923629	8.000000	1.732
75%	2019-10-04 23:37:30	14.100000	92.000000	9.100000	1023.000000	6.930495	10.800000	2.509
max	2020-12-31 23:30:00	36.100000	100.000000	20.900000	1051.000000	6.957497	21.100000	5.403
std	NaN	6.503256	18.134399	5.108536	12.102744	0.011968	3.659642	1.07

8 rows × 149 columns

```
Raw data:
```

	ds	y	humidity	dew.point	pressure	pressure.log	y_window_48_min_max_diff	wind.speed.mean.sqrt	wind.spe
ds									
2016-01-12 00:00:00	2016-01-12 00:00:00	1.6	96.0	1.0	986.0	6.893656	4.422576	2.000000	
2016-01-12 00:30:00	2016-01-12 00:30:00	2.0	94.0	1.1	986.0	6.893656	4.422576	2.428992	
2016-01-12 01:00:00	2016-01-12 01:00:00	2.8	87.0	0.9	987.0	6.894670	4.422576	2.549510	
2016-01-12 01:30:00	2016-01-12 01:30:00	2.0	93.0	1.0	987.0	6.894670	4.422576	2.408319	
2016-01-12 02:00:00	2016-01-12 02:00:00	2.4	89.0	0.8	987.0	6.894670	4.422576	2.626785	
...	...	...	...	...	...	...	...	...	
2020-12-31 21:30:00	2020-12-31 21:30:00	-2.8	96.0	-3.3	1006.0	6.913737	3.200000	0.000000	
2020-12-31 22:00:00	2020-12-31 22:00:00	-3.2	100.0	-3.2	1007.0	6.914731	3.200000	0.000000	

2020-12-31 22:30:00	2020-12-31 22:30:00	-3.6	100.0	-3.6	1007.0	6.914731	3.200000	0.000000
2020-12-31 23:00:00	2020-12-31 23:00:00	-4.4	97.0	-4.8	1007.0	6.914731	3.200000	0.000000
2020-12-31 23:30:00	2020-12-31 23:30:00	-4.8	99.0	-4.9	1007.0	6.914731	3.200000	0.632456

87168 rows × 149 columns

There are 149 features in each of the intervals_etc feature sets.

Add gradient and histogram mode features to the intervals_etc feature set:

```
train_ints, valid_ints, test_ints = add_grad_hist_mode_p2p_feats(train_ints, valid_ints, test_ints, 'ints')
```

```
WARN: high overlap between train_df and valid_df rows!
Number of shared rows: 5231
Approximate overlap: 29.86 %

WARN: high overlap between train_df and test_df rows!
Number of shared rows: 5707
Approximate overlap: 32.57 %


train_ints
before.index.equals(after.index): True
before.index.freq == after.index.freq: True
before[common_cols].dtypes == after[common_cols].dtypes: False
before[common_cols].describe() == after[common_cols].describe(): False

before[common_cols].equals(after[common_cols]): False
redundancy before > after: False
mean before feature redundancy: 30.314
mean after feature redundancy:  55.517
```

	before	after	diff
rows	87168	87168	0
cols	149	249	100
missing_rows	0	0	0
missing_cols	0	0	0
total_nas	0	0	0
rows_with_nas	0	0	0
cols_with_nas	0	0	0
single_value_cols	4	4	0
low_var_rows	0	0	0
low_var_cols	17	17	0
duplicate_rows	0	0	0
duplicate_index_labels	0	0	0
duplicate_col_labels	0	0	0

```
valid_ints
before.index.equals(after.index): True
before.index.freq == after.index.freq: True
before[common_cols].dtypes == after[common_cols].dtypes: False
before[common_cols].describe() == after[common_cols].describe(): False

before[common_cols].equals(after[common_cols]): False
redundancy before > after: False
mean before feature redundancy: 29.302
mean after feature redundancy:  54.825
```

	before	after	diff
rows	17520	17520	0
cols	149	249	100
missing_rows	0	0	0
missing_cols	0	0	0
total_nas	0	0	0
rows_with_nas	0	0	0
cols_with_nas	0	0	0
single_value_cols	4	4	0
low_var_rows	0	0	0
low_var_cols	18	18	0
duplicate_rows	0	0	0
duplicate_index_labels	0	0	0
duplicate_col_labels	0	0	0

```
test_ints
before.index.equals(after.index): True
before.index.freq == after.index.freq: True
before[common_cols].dtypes == after[common_cols].dtypes: False
before[common_cols].describe() == after[common_cols].describe(): False

before[common_cols].equals(after[common_cols]): False
```

```
before[common_cols].equals(after[common_cols]): False
redundancy before > after: False
mean before feature redundancy: 31.458
mean after feature redundancy:  56.167
```

	before	after	diff
rows	17520	17520	0
cols	149	249	100
missing_rows	0	0	0
missing_cols	0	0	0
total_nas	0	0	0
rows_with_nas	0	0	0
cols_with_nas	0	0	0
single_value_cols	5	5	0
low_var_rows	0	0	0
low_var_cols	17	17	0
duplicate_rows	0	0	0
duplicate_index_labels	0	0	0
duplicate_col_labels	0	0	0

```
WARN: train_sanity != valid_sanity
```

	before	after	diff
low_var_cols	17	17	0
low_var_cols	18	18	0

```
WARN: train_sanity != test_sanity
```

	before	after	diff
single_value_cols	4	4	0
single_value_cols	5	5	0

```
WARN: test_sanity != valid_sanity
```

	before	after	diff
single_value_cols	5	5	0
low_var_cols	17	17	0
single_value_cols	4	4	0
low_var_cols	18	18	0

Added 100 additional features to each of the intervals_etc feature sets (`train_ints`, `valid_ints`, `test_ints`).

Initial feature selection score experiments

There are over 1,000 features across 8 feature sets. It is not possible to run all these features with darts/lightGBM and use lagged features. The run time is too long. Some quick feature selection score calculations are necessary for feature prioritisation.

I use the following feature selection approach from sklearn: [univariate linear regression tests returning F-statistic and p-values for testing the effect of a single regressor, sequentially for many regressors.]

There are two F-statistic summary functions:

- `get_feature_selection_scores`
  - returns single ordered pandas dataframe of F-test values for each individual variable, window, transform, lag **combination**
- `summarise_multi_step_feat_sel_scores`
  - returns tables and plots **grouped** by each individual variable, window, transform and lag

Meteorology-based feature selection summary with `get_feature_selection_scores` (lag = 0):

```
sel_cols = ['pressure', 'humidity', 'dew.point_des', 'irradiance', 'za_rad']
fs_df = get_feature_selection_scores(train_df, sel_cols)
display(fs_df.head(40))
```

	r_test	f_test
y_des_window_12_hist_mode	0.048141	0.177442
y_des_window_24_hist_mode	0.038822	0.060965
y_des_window_60_hist_mode	0.035924	0.046358
y_des_window_48_hist_mode	0.035015	0.042622
y_des_window_36_hist_mode	0.033700	0.037773
y_des_window_72_hist_mode	0.033496	0.037073
dew.point_des	0.033002	0.035435
dew.point_des_window_12_hist_mode	0.032721	0.034536
dew.point_des_window_24_hist_mode	0.032107	0.032650
t_pot	0.031947	0.032176
svp	0.031774	0.031670
ground_hf	0.031704	0.031467
y_des_window_84_hist_mode	0.031054	0.029645
dew.point_des_window_36_hist_mode	0.030947	0.029353
y_des_window_96_hist_mode	0.030794	0.028944
air_density	-0.029656	0.026056
dew.point_des_window_48_hist_mode	0.029083	0.024706
dew.point_des_window_60_hist_mode	0.027123	0.020538
vp_def	-0.026479	0.019305
dew.point_des_window_72_hist_mode	0.025738	0.017966
dew.point_des_window_84_hist_mode	0.024618	0.016082
dew.point_des_window_96_hist_mode	0.023337	0.014120
H2OC	0.022569	0.013034
mixing_ratio	0.022543	0.013000
specific_humidity	0.022214	0.012554
vapour_pressure	0.022198	0.012533
ah	0.021683	0.011860
humidity	-0.017501	0.007293
y_des_window_36_min_max_diff	0.014424	0.004793
dew.point_des_window_12_grad_std	0.013590	0.004222
y_des_window_48_min_max_diff	0.013586	0.004219
y_des_window_60_min_max_diff	0.013280	0.004020
y_des_window_72_min_max_diff	0.013184	0.003959
y_des_window_24_min_max_diff	0.013012	0.003851
y_des_window_84_min_max_diff	0.012502	0.003540
dew.point_des_window_12_min_max_diff	0.012415	0.003488
humidity_window_60_hist_mode	-0.011725	0.003094
y_des_window_96_min_max_diff	0.011340	0.002886
humidity_window_12_hist_mode	-0.011314	0.002872
dew.point_des_window_60_grad_std	0.011295	0.002862

Features with F-statistic values above the core features (`dew.point_des`, `humidity`, `pressure`) are worth prioritising. Features with F-statistic values between the core features may be worth exploring further. Features with F-statistic values below the core features should probably be ignored.

The histogram mode features are worth prioritising at lag=0. Disappointingly, the gradient-based features did not appear in the ordered lag=0 table. They later on proved very useful at low lag values. The F-statistic feature selection process is much faster to run than a full lightGBM/darts model but there will be differences with the lightGBM variable importance values.

Full F-test feature selection scores for intervals_etc feature set using the `summarise_multi_step_feat_sel_scores` function.

Selected lags:

- 1, 2, 4, 6, 12, 18, 24, 30, 36, 42, 48

Tables:

- feature_window_transform_lag
- feature_transform_lag
- lags
- features
- transforms
- windows
- shifts if present
- types - target, past covariate, future covariate

Plots:

- Boxplots
    - all F-test values
    - forecast steps - usually 1 to 48
    - windows
    - feature_transform
        - above core features
        - between core features
    - base features
    - transforms
- Surface plots
    - forecast steps by window values
        - minimum
        - mean
        - maximum

When summarising F-statistic values in tables the following values are calculated:

- count, mean, standard deviation, minimum,
- 25th percentile, 50th percentile, 75th percentile, maximum
- above, between and below core features
- above, between and below solar features
- above and below Boruta shadow feature
- number zero F-test score
- percent zero F-test score

```
sel_cols = ['pressure', 'humidity', 'dew.point_des', 'irradiance', 'za_rad']
sel_lags = [1, 2, 4, 6, 12, 18, 24, 30, 36, 42, 48]
vb = True

# summarise_multi_step_feat_sel_scores(train_df, 'default', sel_cols, sel_lags, verbose=vb)  # 2m 27s
# summarise_multi_step_feat_sel_scores(train_pair, 'pairwise', sel_cols, sel_lags, verbose=vb)  # 3m 20s
# summarise_multi_step_feat_sel_scores(train_ccorr, 'cross-correlation', sel_cols, sel_lags, verbose=vb)  # 1m 35s
# summarise_multi_step_feat_sel_scores(train_stats, 'rolling stats', sel_cols, sel_lags, verbose=vb)  # 4m 34s
summarise_multi_step_feat_sel_scores(train_ints, 'intervals etc', sel_cols, sel_lags, verbose=vb)  # 2m 56s

# summarise_multi_step_feat_sel_scores(train_biv, 'bivariate', sel_cols, sel_lags, verbose=vb)  # 3m 44s
# summarise_multi_step_feat_sel_scores(train_tsf, 'tsfeatures', sel_cols, sel_lags, verbose=vb)  # 7m 45s
# summarise_multi_step_feat_sel_scores(train_c22, 'catch22', sel_cols, sel_lags, verbose=vb)  # 12m 26s
```

y_des_window_24_intervals_intervals_mean_lag_12	y_des_window_24_intervals_intervals_mean_lag_12	y_des_window_24_intervals_intervals_mean
y_des_window_12_intervals_intervals_mean_lag_2	y_des_window_12_intervals_intervals_mean_lag_2	y_des_window_12_intervals_intervals_mean
...	...	...
pressure_window_48_grad_std_lag_24	pressure_window_48_grad_std_lag_24	pressure_window_48_grad_std
humidity_window_96_grad_std_lag_2	humidity_window_96_grad_std_lag_2	humidity_window_96_grad_std
humidity_window_72_grad_std_lag_48	humidity_window_72_grad_std_lag_48	humidity_window_72_grad_std
y_des_window_12_pacf_features_diff1x_pacf5_lag_2	y_des_window_12_pacf_features_diff1x_pacf5_lag_2	y_des_window_12_pacf_features_diff1x_pacf5
pressure_window_60_grad_std_lag_2	pressure_window_60_grad_std_lag_2	pressure_window_60_grad_std

93888 rows × 18 columns

feature_transform_lag - ft_mean_lim = 0.01:

feature_transform_lag	**count**	**mean**	**std**	**min**	**25%**	**50%**	**75%**	**max**	**sum**	**num_0**	**pc_0**

lag:

lag	**count**	**mean**	**std**	**min**	**25%**	**50%**	**75%**	**max**	**sum**	**num_0**	**pc_0**
0	93888	0.000511	0.001584	0.0	0.000008	0.000049	0.000176	0.014267	47.999504	7545	8.036171

feature:

feature	**count**	**mean**	**std**	**min**	**25%**	**50%**	**75%**	**max**	**sum**	**num_0**	**pc_0**
H2OC	576	0.000416	1.834488e-04	0.000131	0.000240	0.000412	0.000604	0.000715	0.239431	0.0	0.000000
ah	576	0.000412	1.811348e-04	0.000124	0.000239	0.000397	0.000599	0.000732	0.237029	0.0	0.000000
air_density	576	0.000345	2.378251e-04	0.000036	0.000192	0.000304	0.000452	0.001351	0.198486	0.0	0.000000
azimuth	576	0.000002	1.964467e-06	0.000000	0.000000	0.000001	0.000003	0.000010	0.001032	169.0	29.340278
azimuth_cos	576	0.000002	1.964467e-06	0.000000	0.000000	0.000001	0.000003	0.000010	0.001032	169.0	29.340278
azimuth_rad	576	0.000002	1.964467e-06	0.000000	0.000000	0.000001	0.000003	0.000010	0.001032	169.0	29.340278
azimuth_sin	576	0.000002	1.964467e-06	0.000000	0.000000	0.000001	0.000003	0.000010	0.001032	169.0	29.340278
declination	576	0.000002	6.994477e-07	0.000001	0.000001	0.000002	0.000002	0.000003	0.001132	0.0	0.000000
dew.point	14976	0.000281	3.632630e-04	0.000000	0.000051	0.000096	0.000425	0.001898	4.204909	162.0	1.081731
ground_hf	576	0.000586	3.099868e-04	0.000139	0.000361	0.000507	0.000755	0.001634	0.337259	0.0	0.000000
humidity	14976	0.000042	6.084205e-05	0.000000	0.000002	0.000017	0.000050	0.000399	0.623687	2089.0	13.948985
irradiance	576	0.000043	3.508737e-05	0.000000	0.000012	0.000040	0.000066	0.000154	0.025008	29.0	5.034722
mixing_ratio	576	0.000414	1.828678e-04	0.000131	0.000239	0.000411	0.000602	0.000712	0.238619	0.0	0.000000
pressure	14976	0.000042	4.483450e-05	0.000000	0.000002	0.000029	0.000067	0.000184	0.627195	1865.0	12.453259
specific_humidity	576	0.000402	1.731679e-04	0.000134	0.000237	0.000402	0.000578	0.000682	0.231524	0.0	0.000000
svp	576	0.000559	3.442451e-04	0.000092	0.000310	0.000469	0.000756	0.001644	0.322198	0.0	0.000000
t_pot	576	0.000533	3.043742e-04	0.000104	0.000321	0.000460	0.000684	0.001670	0.306941	0.0	0.000000
vapour_pressure	576	0.000414	1.726023e-04	0.000145	0.000249	0.000414	0.000592	0.000695	0.238377	0.0	0.000000
vp_def	576	0.000299	2.777509e-04	0.000008	0.000059	0.000206	0.000463	0.001067	0.172100	0.0	0.000000
y	38592	0.001036	2.353455e-03	0.000000	0.000014	0.000062	0.000633	0.014267	39.989501	2368.0	6.135987
za	576	0.000002	1.807442e-06	0.000000	0.000000	0.000001	0.000003	0.000007	0.000990	178.0	30.902778
za_rad	576	0.000002	1.807442e-06	0.000000	0.000000	0.000001	0.000003	0.000007	0.000990	178.0	30.902778

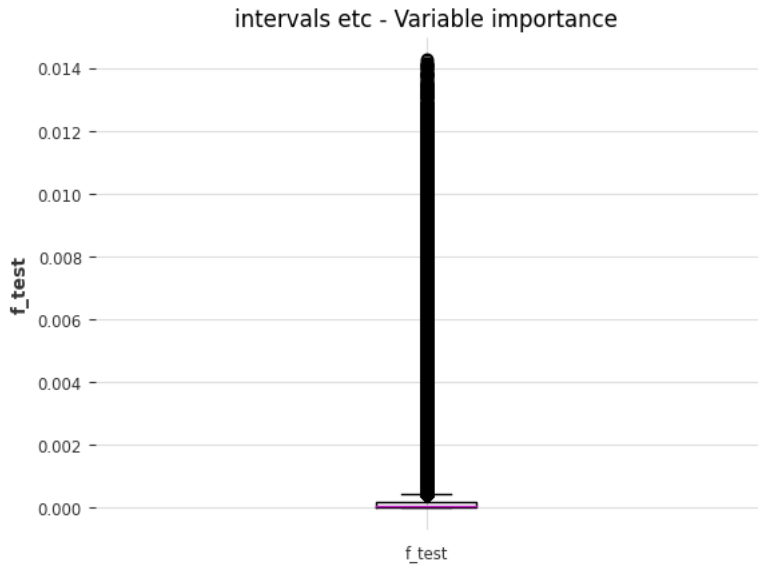
transform:

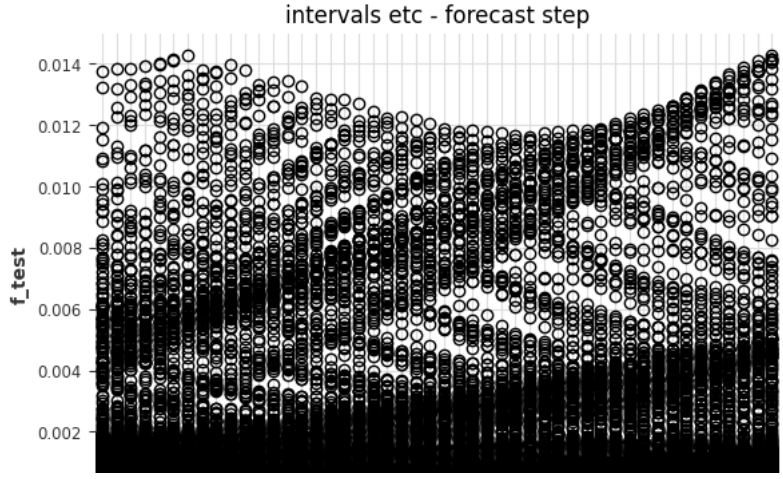
transform	**count**	**mean**	**std**	**min**	**25%**	**50%**	**75%**	**max**	**sum**	**num_0**	**pc_0**
None	11520	2.293275e-04	2.802384e-04	0.000000	0.000002	0.000113	0.000406	0.001670	2.641853	1169.0	10.147569
des	576	1.078727e-03	4.934823e-04	0.000351	0.000612	0.001034	0.001583	0.001898	0.621347	0.0	0.000000
grad	2304	2.842578e-05	4.436303e-05	0.000000	0.000003	0.000012	0.000032	0.000243	0.065493	180.0	7.812500
grad_std	18432	2.803852e-05	4.560730e-05	0.000000	0.000001	0.000004	0.000037	0.000315	0.516806	3720.0	20.182292
hist_mode	18432	5.269541e-04	6.344655e-04	0.000000	0.000091	0.000257	0.000771	0.009203	9.712818	160.0	0.868056

intervals_mean	5760	5.647873e-03	3.257418e-03	0.000707	0.002919	0.004887	0.007942	0.014267	32.531748	0.0	0.000000
min_max_diff	19008	6.204893e-05	5.754269e-05	0.000000	0.000021	0.000041	0.000082	0.000325	1.179426	156.0	0.820707
pacf_features_diff1x_pacf5	5760	3.499375e-05	4.297891e-05	0.000000	0.000002	0.000017	0.000057	0.000293	0.201564	803.0	13.940972
pacf_features_diff2x_pacf5	5760	2.484045e-05	2.485306e-05	0.000000	0.000004	0.000017	0.000040	0.000115	0.143081	461.0	8.003472
pacf_features_x_pacf5	5760	6.690399e-05	6.868898e-05	0.000000	0.000014	0.000042	0.000103	0.000383	0.385367	321.0	5.572917
shadow	576	1.736111e-09	4.166667e-08	0.000000	0.000000	0.000000	0.000000	0.000001	0.000001	575.0	99.826389

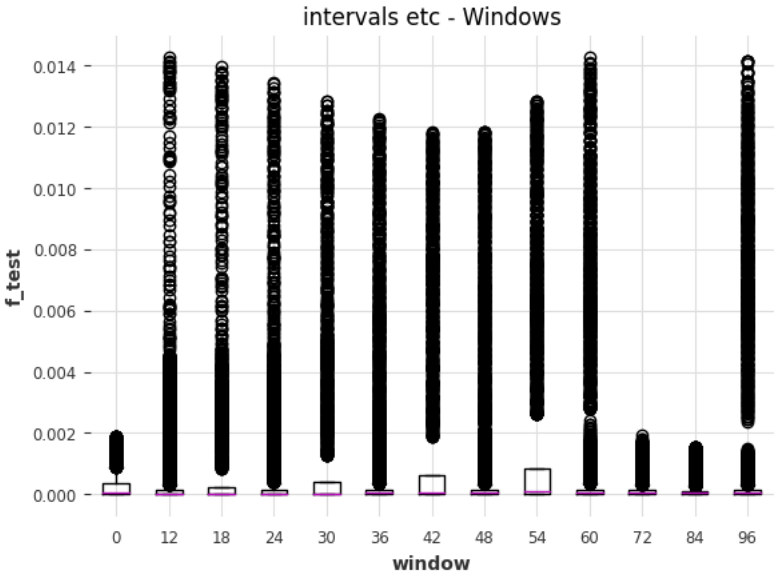
window:

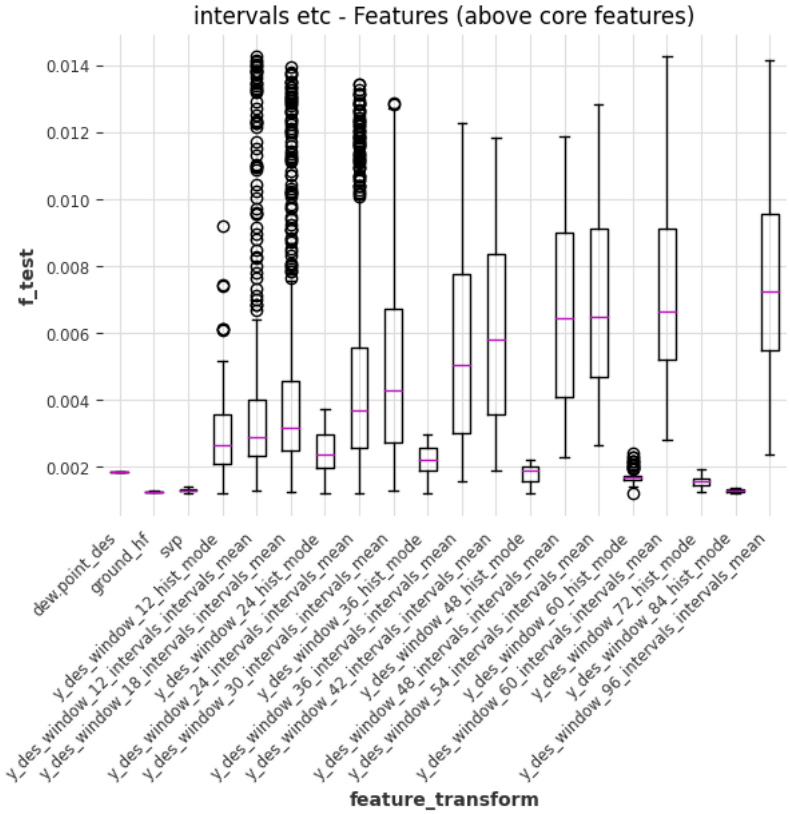
window	**count**	**mean**	**std**	**min**	**25%**	**50%**	**75%**	**max**	**sum**	**num_0**	**pc_0**
0	14976	0.000222	0.000326	0.0	0.000002	0.000043	0.000363	0.001898	3.328694	1924	12.847222
12	9216	0.000408	0.001186	0.0	0.000002	0.000012	0.000126	0.014256	3.761298	1330	14.431424
18	2304	0.000963	0.002274	0.0	0.000001	0.000007	0.000245	0.013950	2.217793	457	19.835069
24	9216	0.000462	0.001373	0.0	0.000005	0.000029	0.000158	0.013441	4.253719	728	7.899306
30	2304	0.001269	0.002649	0.0	0.000006	0.000021	0.000387	0.012845	2.922851	212	9.201389
36	9216	0.000525	0.001548	0.0	0.000017	0.000049	0.000160	0.012257	4.834452	508	5.512153
42	2304	0.001562	0.002983	0.0	0.000023	0.000059	0.000612	0.011815	3.599299	19	0.824653
48	9792	0.000557	0.001689	0.0	0.000026	0.000076	0.000153	0.011850	5.451563	373	3.809232
54	2304	0.001791	0.003305	0.0	0.000032	0.000087	0.000856	0.012838	4.125755	13	0.564236
60	9216	0.000609	0.001873	0.0	0.000024	0.000060	0.000163	0.014267	5.612213	664	7.204861
72	6912	0.000177	0.000322	0.0	0.000017	0.000053	0.000139	0.001923	1.221950	570	8.246528
84	6912	0.000162	0.000290	0.0	0.000016	0.000055	0.000124	0.001550	1.120295	341	4.933449
96	9216	0.000602	0.001945	0.0	0.000020	0.000065	0.000140	0.014151	5.549622	406	4.405382

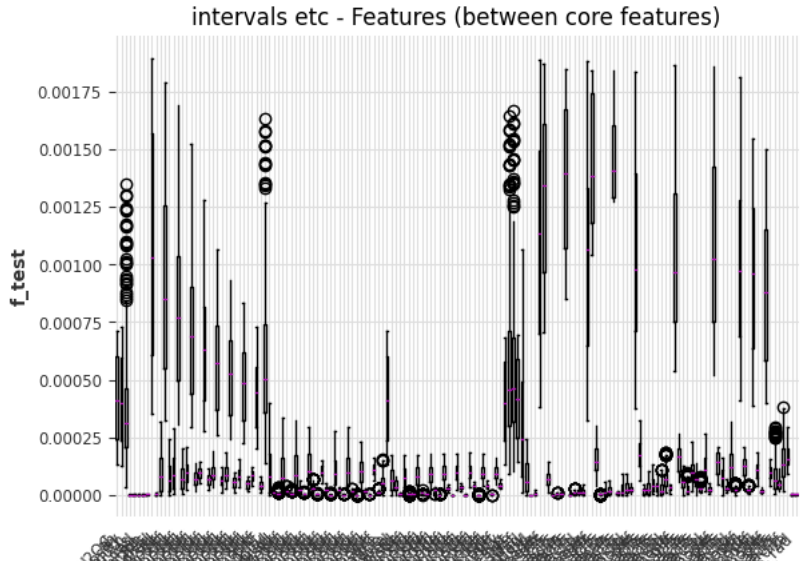


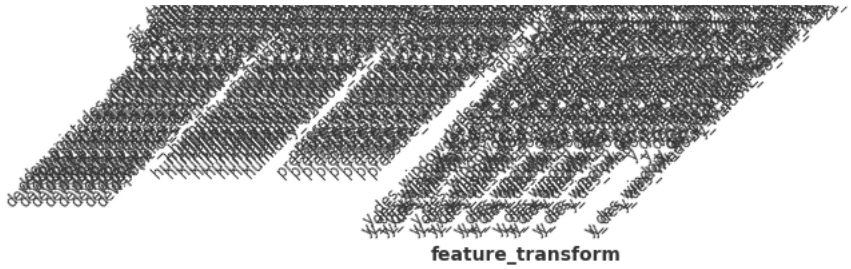


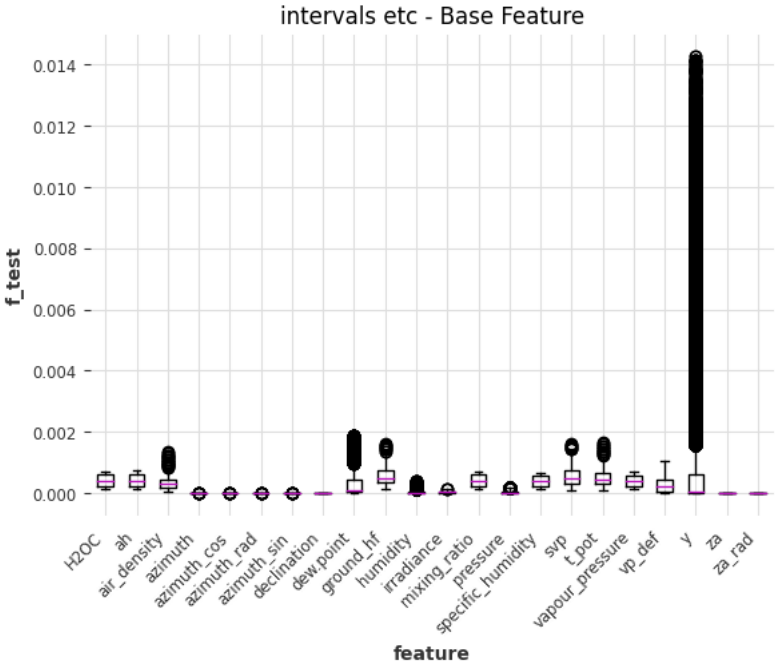

0.000
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47
step

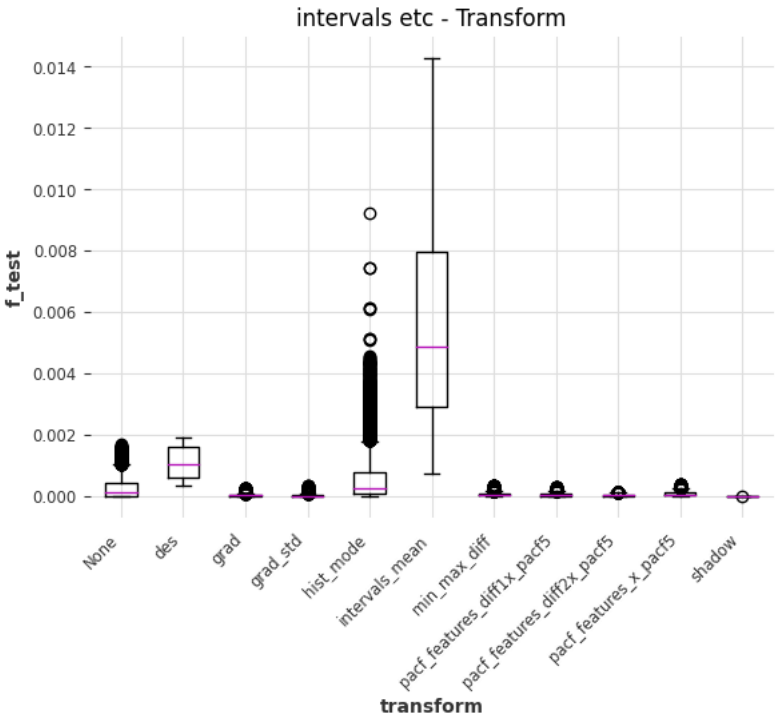

intervals etc - Windows
f_test
0.014
0.012
0.010
0.008
0.006
0.004
0.002
0.000
0 12 18 24 30 36 42 48 54 60 72 84 96
window

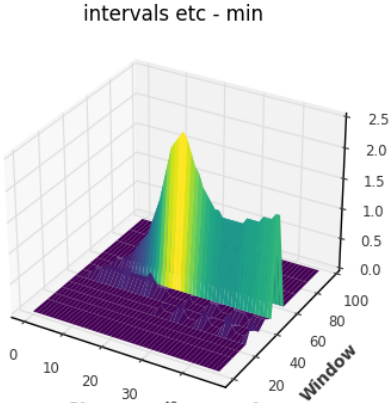

intervals etc - Features (above core features)
f_test
0.014
0.012
0.010
0.008
0.006
0.004
0.002
dew.point_des
ground_hf
svp
y_des_window_12_hist_mode
y_des_window_12_intervals_mean
y_des_window_18_intervals_mean
y_des_window_24_hist_mode
y_des_window_24_intervals_mean
y_des_window_30_intervals_mean
y_des_window_36_hist_mode
y_des_window_36_intervals_mean
y_des_window_42_intervals_mean
y_des_window_48_hist_mode
y_des_window_48_intervals_mean
y_des_window_54_intervals_mean
y_des_window_60_hist_mode
y_des_window_60_intervals_mean
y_des_window_72_hist_mode
y_des_window_72_intervals_mean
y_des_window_84_hist_mode
y_des_window_96_intervals_mean
feature_transform

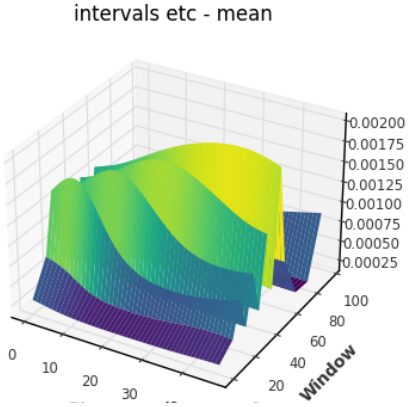

intervals etc - Features (between core features)
f_test
0.00175
0.00150
0.00125
0.00100
0.00075
0.00050
0.00025
0.00000

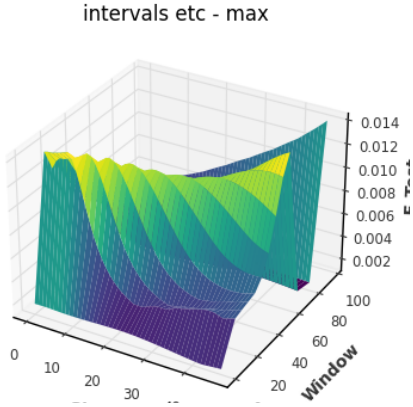

feature_transform

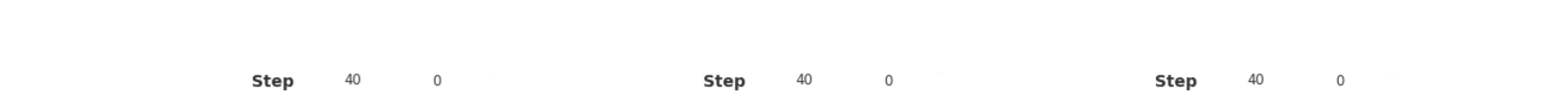

intervals etc - Base Feature
f_test
0.014
0.012
0.010
0.008
0.006
0.004
0.002
0.000
H2OC
ah
air_density
azimuth
azimuth_cos
azimuth_rad
azimuth_sin
declination
dew_point
ground_hf
humidity
irradiance
mixing_ratio
pressure
specific_humidity
svp
t_pot
vapour_pressure
vp_def
y
za
za_rad
feature


intervals etc - Transform
f_test
0.014
0.012
0.010
0.008
0.006
0.004
0.002
0.000
None
des
grad
grad_std
hist_mode
intervals_mean
min_max_diff
pacf_features_diff1x_pacf5
pacf_features_diff2x_pacf5
pacf_features_x_pacf5
shadow
transform


intervals etc - min
2.5
2.0
1.5
1.0
0.5
0.0
0
10
20
30
20
40
60
80
100
Window


intervals etc - mean
0.00200
0.00175
0.00150
0.00125
0.00100
0.00075
0.00050
0.00025
0
10
20
30
20
40
60
80
100
Window


intervals etc - max
0.014
0.012
0.010
0.008
0.006
0.004
0.002
0
10
20
30
20
40
60
80
100
Window



Last relevant commit before removal for brevity:

- default
- pairwise
- cross-correlation
- rolling statistics
- intervals etc - also shown above
- bivariate
- tsfeatures
- catch22

Little to nothing useful in the pairwise or cross-correlation feature sets. The min, max and mean features for `y_des` and `dew.point_des` from the rolling statistics feature set may prove somewhat useful. There were no compelling features in the bivariate set. The tsfeatures set was probably the best set; particularly the `intervals_mean` and `histogram_mode` features, but others may be worth experimenting with. There are two outlier-based features from the catch22 set which may be worth considering: DN_OutlierInclude_n_001_mdrmd (negative outlier timing) and DN_OutlierInclude_p_001_mdrmd (positive outlier timing).

The intervals etc feature set contains an expanded set of `intervals_mean` and `pacf`-based features from the tsfeatures package. Additional window lengths were used compared to the tsfeatures dataset.

I was disappointed that the seasonal strength feature did not perform better. Perhaps I should have calculated this separately for the yearly and daily components.

The `grad` and `pacf` features did not perform that well despite proving useful in the lightGBM models. Based on lightGBM variable importance, the gradient features were most useful for the early lags - 1, 2, 3. Again from lightGBM variable importance, the pacf features were most useful for later lags. It's not surprising that there are differences with lightGBM. The F-test feature selection scores take less than 4 minutes to run for the bivariate feature set. I killed the bivariate lightGBM models after running for 6 hours. The lightGBM models must be built on subsets of the larger feature sets. These feature selection scores can prioritise feature subsets for the lightGBM models.

LightGBM Models

I build darts lightGBM multi-step forecast models with past and future covariates. A few things are worth noting:

- I use `multi_models: True`
  - separate model will be trained for each future step to predict
  - one model per forecast step
  - 48 lightGBM models
  - same features and lags used for each model
  - does not use recurrent prediction
    - does not feed in past predictions to help with future predictions
- avoid using single lightGBM model with `horizon = 48` and `output_chunk_length = 1`
  - automatically uses 'unavailable' features from the future for later forecast steps
  - which leads to artificially low error metrics

It is unrealistic to concurrently optimise all of the time series gradient boosting components on google colab. So, an iterative process is necessary.

There are 3 major components to optimise:

- Lag selection
  - using core features
  - target series lags
  - past covariate (feature) lags
  - future covariate (irradiance etc) lags
    - which can include both past and future lags
- Feature selection
  - using selected lags
  - the original features
  - feature engineering work
- Hyperparameter optimisation
  - using selected lags and features
  - optuna package

I don't show it in this notebook, but I did a little manual hyperparameter optimisation before lag selection. I found it useful to set the lightGBM parameter:

```
 'feature_fraction': 0.5
```

I use this with lag selection and feature selection.

See also:

- Darts quickstart guide including target series, past covariates and future covariates.
- LightGBM model support in darts package.

Lag Selection

lightGBM provides variable importance measures, which I use for feature selection.

Boruta-style shadow variables can be used for feature selection. Shadow variables find relevant features by comparing the original variables importance with the importance from randomly permuted copies of variables aka shadows. All variables with importance values below the shadow feature importances can be safely rejected. Anecdotally, features with importance values above but close to the shadow variable importances should also be considered for rejection. This might suggest that lightGBM feature importance is slightly inflated. This has been informally noted elsewhere. The `y_des_shadow` variable was also used with the F-statistic feature selection experiments.

I set the future lags to, 1 to 48, which corresponds to the forecast steps. I also set the past covariate lags equal to the future covariate lags. Models are built using only the core features: `y_des`, `dew.point_des`, `humidity` and `pressure` plus the solar-based future covariates `irradiance` and `za_rad`.

The process for comparing lags on core features:

```
 for i in [6, 4, 3, 2]:
   init_lags1 = [-1, -2, -4]
   mid_lags1  = [i for i in range(-6, -52, -j) if i > -50]
   target_lags1 = pastcov_lags1 = init_lags1 + mid_lags1
   # build model
   # validate model
   # summarise model
   # summarise variable importance
```

```
# train = train_df_tsf
# valid = valid_df_tsf
# train = train_df_biv
# valid = valid_df_biv
# train = train_df_pair
# valid = valid_df_pair
# train = train_df_c22
# valid = valid_df_c22
# train = train_df_roll
# valid = valid_df_roll
# train = train_df_ccor
# valid = valid_df_ccor
train = train_df
valid = valid_df

# y_des MUST BE included in ex_cols at this stage
ex_cols1 = ['y', 'y_seasonal', 'y_res', 'y_orig', 'ds', # 'y_des',
            'dew.point',
            #'humidity', 'pressure',
            'wind.speed.mean',
            'humidity_des', 'pressure_des', 'humidity_des_diff_1', 'pressure_des_diff_1',
            'humidity_seasonal', 'dew.point_seasonal', 'pressure_seasonal',
            'humidity_res', 'dew.point_res', 'pressure_res',
            'wind.speed.mean_res', 'wind.speed.mean_seasonal',
            'wind.speed.mean_yearly', 'wind.speed.mean_daily',
            'humidity_daily', 'pressure_daily', 'dew.point_daily',
            'humidity_yearly', 'pressure_yearly', 'dew.point_yearly',
            'dew.point_yhat',  'pressure_yhat', 'humidity_yhat', 'y_yhat',
            'wind.speed.mean_daily.x', 'wind.speed.mean_daily.y',
            'wind.speed.mean_yearly.x', 'wind.speed.mean_yearly.y',
            'wind.speed.mean_yhat.x', 'wind.speed.mean_yhat.y',
            'wind.speed.mean_des.x', 'wind.speed.mean_des.y',
            'wind.speed.mean.x', 'wind.speed.mean.y',
            'sunshine',
            'wind.speed.max', 'wind.speed.max.sqrt',
            'wind.speed.mean.sqrt.x', 'wind.speed.mean.sqrt.y',
            'wind.speed.max.sqrt.x', 'wind.speed.max.sqrt.y',
            'wind.speed.mean_des',
```

```
            'declination', 'za', #'za_rad', 'irradiance',
            'y_daily', 'y_yearly',
            'wind.speed.mean.sqrt',
            'wind.bearing.mean',
            'wind.speed.mean_yhat', 'pressure.log',
            'wind.speed.mean.y_yearly', 'wind.speed.mean.x_yearly',
            'wind.speed.mean.y_daily', 'wind.speed.mean.x_daily',
            'dew.point_trend', 'y_trend', 'humidity_trend', 'pressure_trend',
            'wind.speed.mean_trend', 'declination', 'declination_diff_1',
            'day.sin.1', 'day.cos.1', 'year.sin.1', 'year.cos.1',
            'humidity_diff_1', 'ground_hf', 'irradiance_diff_1', 'y_des_diff_1',
            'pressure_diff_1', 'dew.point_des_diff_1',
            #'svp', 'rainfall', 'ah',
            'nao', 'mei', 'azimuth_diff_1', 'za_diff_1', 'za_rad_diff_1',
            'azimuth_cos_diff_1', 'azimuth_sin_diff_1',
            #'mixing_ratio', 'specific_humidity', 'vapor_pressure',
            'azimuth_sin',
            'azimuth_rad_diff_1', 'y_fft_x', 'y_fft_y', 'y_fft', 'dew.point_fft'
            ]
inc_cols1 = ['y_des', 'dew.point_des', 'pressure', 'humidity',
             't_pot',
             'svp', #'ground_hf',
             'air_density',
             'vp_def',
            #'y_yearly',
            #'y_yhat',
             'wind.speed.max.sqrt.x', 'wind.speed.max.sqrt.y',
             'H2OC',
             'mixing_ratio',
             'specific_humidity',
             'vapour_pressure',
             'ah',
             'irradiance',
             'za_rad',
             'azimuth_cos',
             'y_des_shadow',
#'y_grad',
#'y_window_48_grad_std',
#'y_window_48_hist_mode',
 'y_des_grad',
#'y_des_window_24_grad_std',
 'y_des_window_24_hist_mode',
#'y_des_window_24_min_max_diff',
#'dew.point_grad',
#'dew.point_window_48_grad_std',
#'dew.point_window_48_hist_mode',
#'dew.point_window_48_min_max_diff',
 'dew.point_des_grad',
#'dew.point_des_window_24_grad_std',
 'dew.point_des_window_24_hist_mode',
#'dew.point_des_window_24_min_max_diff',
 'humidity_grad',
#'humidity_window_24_grad_std',
 'humidity_window_24_hist_mode',
#'humidity_window_24_min_max_diff',
 'pressure_grad',
#'pressure_window_24_grad_std',
 'pressure_window_24_hist_mode',
#'pressure_window_24_min_max_diff'
             ]

inc_cols1 = ['y_des', 'dew.point_des', 'humidity', 'pressure',
             'wind.speed.max.sqrt.x', 'wind.speed.max.sqrt.y',
             'y_des_shadow',
             'y_des_window_24_hist_mode', 'dew.point_des_window_24_hist_mode',
             #'humidity_window_24_hist_mode', 'pressure_window_24_hist_mode',
             #'y_des_grad', 'y_des_window_24_hist_mode', 'dew.point_des_grad',
             #'dew.point_des_window_24_hist_mode'
             ]

data_params1 = {'y_col': Y_COL,
                'past_cov_cols': inc_cols1,
               #'past_cov_cols': train.columns.difference(ex_cols1),
               #'fut_cov_cols': None,
               'fut_cov_cols': FUT_COV,
               }

series, past_cov, fut_cov = get_darts_series(train, data_params1)
val_ser, val_past_cov, val_fut_cov = get_darts_series(valid, data_params1)

init_lags1 = [-1, -2, -4]
# late_lags1 = [-52, -96, -120]
```

```
mid_lags1  = [i for i in range(-6, -52, -4) if i > -50]
target_lags1 = pastcov_lags1 = init_lags1 + mid_lags1 # + late_lags1
# target_lags1  = [-1, -2, -4, -6, -12, -18, -24, -30, -36, -42, -48, -60, -72, -84, -96, -120]
# pastcov_lags1 = [-1, -2, -4, -6, -12, -18, -24, -30, -36, -42, -48, -60, -72, -84, -96, -120]
futcov_lags1  = (1, 48)


if data_params1['fut_cov_cols'] is not None:
  lag_params1 = {'lags': target_lags1,
                 'lags_past_covariates': pastcov_lags1,
                 'lags_future_covariates': futcov_lags1,
                }
else:
  lag_params1 = {'lags': target_lags1,
                 'lags_past_covariates': pastcov_lags1,
                }

mod_params1 = {'feature_fraction': 0.5,
               'output_chunk_length': 48,
               'multi_models': True,
              #'linear_trees': True,
              #'bagging_fraction': 0.25,
              #'bagging_freq': 2,
              #'num_leaves':  16
              }
npr_params1 = {'n_estimators': 200,
               'verbose': -1,
               #'early_stopping_round': 10,
              }  # Non-PRinting params
all_params1 = {**lag_params1,
               **mod_params1,
               **npr_params1,}

title1 = get_main_plot_title('lgbm - ', lag_params1, mod_params1)
model1 = LightGBMModel(**all_params1)


warnings.filterwarnings(
    action   = 'ignore',
    category = UserWarning,
    module   = r'.*sklearn'
)
warnings.filterwarnings(
    action   = 'ignore',
    category = FutureWarning,
    module   = r'.*sklearn'
)


if data_params1['fut_cov_cols'] is not None:
  model1.fit(series,
             past_covariates = past_cov,
             future_covariates = fut_cov,
             val_series = val_ser,
             val_past_covariates = val_past_cov,
             val_future_covariates = val_fut_cov,
            )
else:
  model1.fit(series,
             past_covariates = past_cov,
             val_series = val_ser,
             val_past_covariates = val_past_cov,
            )

if data_params1['fut_cov_cols'] is not None:
  backtest1 = model1.historical_forecasts(series = val_ser,
                                          past_covariates = val_past_cov,
                                          future_covariates = val_fut_cov,
                                          start   = 0.01,
                                          retrain = False,
                                          verbose = True,
                                          forecast_horizon = HORIZON,
                                          last_points_only = False
                                         )
else:
  backtest1 = model1.historical_forecasts(series = val_ser,
                                          past_covariates = val_past_cov,
                                          start   = 0.01,
                                          retrain = False,
                                          verbose = True,
                                          forecast_horizon = HORIZON,
                                          last_points_only = False
```

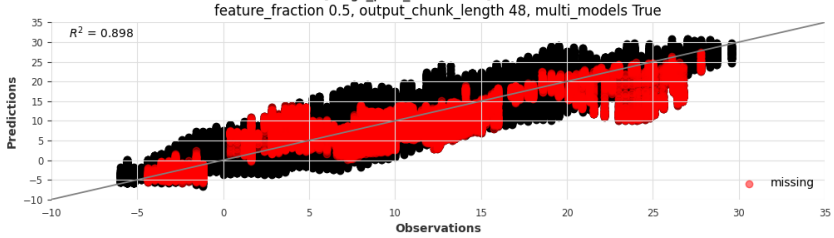
```
                                        )

hist_comp1 = get_historic_comparison(backtest1, valid)
summarise_historic_comparison(hist_comp1, valid)
plot_multistep_diagnostics(hist_comp1, title1)


# _window_24_ superior to 48 and 96


# inc_cols1 = ['y_des', 'dew.point_des', 'pressure', 'humidity',
#              't_pot', 'svp', 'air_density', 'vp_def',
#              'wind.speed.max.sqrt.x', 'wind.speed.max.sqrt.y',
#              'irradiance', 'za_rad', 'y_des_shadow', 'y_des_grad',
#              'y_des_window_24_hist_mode', 'dew.point_des_grad',
#              'dew.point_des_window_24_hist_mode', 'humidity_grad',
#              'humidity_window_24_hist_mode', 'pressure_grad',
#              'pressure_window_24_hist_mode']
# FUT_COV  = ['irradiance', 'za_rad']
# target_lags1  = [-1, -2, -4, -6, -12, -18, -24, -30, -36, -42, -48, -60, -72, -84, -96, -120]
# pastcov_lags1 = [-1, -2, -4, -6, -12, -18, -24, -30, -36, -42, -48, -60, -72, -84, -96, -120]
# futcov_lags1  = (1, 48)
# 'feature_fraction': 0.5, 'multi_models': True, 'n_estimators': 200,
# Backtest RMSE all: 2.057164
# Backtest MAE all:  1.507607
# Backtest RMSE 48th: 2.600431
# Backtest MAE 48th:  2.009724

# inc_cols1 = ['y_des', 'dew.point_des', 'pressure', 'humidity']
# FUT_COV  = ['irradiance', 'za_rad']
# init_lags1 = [-1, -2, -4]
# mid_lags1  = [i for i in range(-6, -52, -6) if i > -50]
# target_lags1 = pastcov_lags1 = init_lags1 + mid_lags1
# futcov_lags1  = (1, 48)
# 'feature_fraction': 0.5, 'multi_models': True, 'n_estimators': 200,
# Backtest RMSE all: 2.16474
# Backtest MAE all:  1.604902
# Backtest RMSE 48th: 2.721805
# Backtest MAE 48th:  2.112284

# inc_cols1 = ['y_des', 'dew.point_des', 'pressure', 'humidity']
# FUT_COV  = ['irradiance', 'za_rad']
# init_lags1 = [-1, -2, -4]
# mid_lags1  = [i for i in range(-6, -52, -4) if i > -50]
# target_lags1 = pastcov_lags1 = init_lags1 + mid_lags1
# futcov_lags1  = (1, 48)
# 'feature_fraction': 0.5, 'multi_models': True, 'n_estimators': 200,
# Backtest RMSE all: 2.163194
# Backtest MAE all:  1.603735
# Backtest RMSE 48th: 2.705667
# Backtest MAE 48th:  2.096854

# inc_cols1 = ['y_des', 'dew.point_des', 'pressure', 'humidity']
# FUT_COV  = ['irradiance', 'za_rad']
# init_lags1 = [-1, -2, -4]
# mid_lags1  = [i for i in range(-6, -52, -3) if i > -50]
# target_lags1 = pastcov_lags1 = init_lags1 + mid_lags1
# futcov_lags1  = (1, 48)
# 'feature_fraction': 0.5, 'multi_models': True, 'n_estimators': 200,
# Backtest RMSE all: 2.164828
# Backtest MAE all:  1.605242
# Backtest RMSE 48th: 2.724813
# Backtest MAE 48th:  2.110654

# inc_cols1 = ['y_des', 'dew.point_des', 'pressure', 'humidity']
# FUT_COV  = ['irradiance', 'za_rad']
# init_lags1 = [-1, -2, -4]
# mid_lags1  = [i for i in range(-6, -52, -2) if i > -50]
# target_lags1 = pastcov_lags1 = init_lags1 + mid_lags1
# futcov_lags1  = (1, 48)
# 'feature_fraction': 0.5, 'multi_models': True, 'n_estimators': 200,
# Backtest RMSE all: 2.16699
# Backtest MAE all:  1.606742
# Backtest RMSE 48th: 2.713683
# Backtest MAE 48th:  2.102387

# inc_cols1 = ['y_des', 'dew.point_des', 'pressure', 'humidity',
#              'wind.speed.max.sqrt.x', 'wind.speed.max.sqrt.y',
#              'irradiance', 'za_rad', 'y_des_grad',
#              'y_des_window_24_hist_mode', 'dew.point_des_grad',
#              'dew.point_des_window_24_hist_mode']
# FUT_COV  = ['irradiance', 'za_rad']
```

```
# init_lags1 = [-1, -2, -4]
# mid_lags1  = [i for i in range(-6, -52, -4) if i > -50]
# target_lags1 = pastcov_lags1 = init_lags1 + mid_lags1
# futcov_lags1  = (1, 48)
# 'feature_fraction': 0.5, 'multi_models': True, 'n_estimators': 200,
# Backtest RMSE all: 2.035917
# Backtest MAE all:  1.491213
# Backtest RMSE 48th: 2.591591
# Backtest MAE 48th:  2.00561

# inc_cols1 = ['y_des', 'dew.point_des', 'pressure', 'humidity',
#              'wind.speed.max.sqrt.x', 'wind.speed.max.sqrt.y',
#              'svp', 'y_des_grad',
#              'y_des_window_24_hist_mode', 'dew.point_des_grad',
#              'dew.point_des_window_24_hist_mode']
# FUT_COV  = ['irradiance', 'za_rad']
# init_lags1 = [-1, -2, -4]
# mid_lags1  = [i for i in range(-6, -52, -4) if i > -50]
# target_lags1 = pastcov_lags1 = init_lags1 + mid_lags1
# futcov_lags1  = (1, 48)
# 'feature_fraction': 0.5, 'multi_models': True, 'n_estimators': 200,
# Backtest RMSE all: 2.039794
# Backtest MAE all:  1.491369
# Backtest RMSE 48th: 2.558673
# Backtest MAE 48th:  1.98458

# inc_cols1 = ['y_des', 'dew.point_des', 'humidity', 'pressure',
#              'wind.speed.max.sqrt.x', 'wind.speed.max.sqrt.y',
#              't_pot', 'svp', 'air_density', 'vp_def', 'H2OC',
#              'mixing_ratio', 'specific_humidity', 'vapour_pressure',
#              'ah', 'y_des_shadow',]
# FUT_COV  = ['irradiance', 'za_rad']
# init_lags1 = [-1, -2, -4]
# mid_lags1  = [i for i in range(-6, -52, -4) if i > -50]
# target_lags1 = pastcov_lags1 = init_lags1 + mid_lags1
# futcov_lags1  = (1, 48)
# 'feature_fraction': 0.5, 'multi_models': True, 'n_estimators': 200,
# Backtest RMSE all: 2.060649
# Backtest MAE all:  1.520969
# Backtest RMSE 48th: 2.609496
# Backtest MAE 48th:  2.007966

# inc_cols1 = ['y_des', 'dew.point_des', 'humidity', 'pressure',
#              'wind.speed.max.sqrt.x', 'wind.speed.max.sqrt.y',
#              'irradiance', 'za_rad', 'y_des_shadow',]
# FUT_COV  = ['irradiance', 'za_rad']
# init_lags1 = [-1, -2, -4]
# mid_lags1  = [i for i in range(-6, -52, -4) if i > -50]
# target_lags1 = pastcov_lags1 = init_lags1 + mid_lags1
# futcov_lags1  = (1, 48)
# 'feature_fraction': 0.5, 'multi_models': True, 'n_estimators': 200,
# Backtest RMSE all: 2.052126
# Backtest MAE all:  1.513914
# Backtest RMSE 48th: 2.591405
# Backtest MAE 48th:  2.000576

# inc_cols1 = ['y_des', 'dew.point_des', 'humidity', 'pressure',
#              'wind.speed.max.sqrt.x', 'wind.speed.max.sqrt.y',
#              'y_des_grad', 'dew.point_des_grad', 'humidity_grad',
#              'pressure_grad', 'y_des_shadow',]
# FUT_COV  = ['irradiance', 'za_rad']
# init_lags1 = [-1, -2, -4]
# mid_lags1  = [i for i in range(-6, -52, -4) if i > -50]
# target_lags1 = pastcov_lags1 = init_lags1 + mid_lags1
# futcov_lags1  = (1, 48)
# 'feature_fraction': 0.5, 'multi_models': True, 'n_estimators': 200,
# Backtest RMSE all: 2.005001
# Backtest MAE all:  1.466328
# Backtest RMSE 48th: 2.544891
# Backtest MAE 48th:  1.966069
# So far, best model on default feature set!

# inc_cols1 = ['y_des', 'dew.point_des', 'humidity', 'pressure',
#              'wind.speed.max.sqrt.x', 'wind.speed.max.sqrt.y',
#              'y_des_window_24_hist_mode', 'dew.point_des_window_24_hist_mode',
#              'humidity_window_24_hist_mode', 'pressure_window_24_hist_mode',
#              'y_des_shadow',]
# FUT_COV  = ['irradiance', 'za_rad']
# init_lags1 = [-1, -2, -4]
# mid_lags1  = [i for i in range(-6, -52, -4) if i > -50]
# target_lags1 = pastcov_lags1 = init_lags1 + mid_lags1
# futcov_lags1  = (1, 48)
```

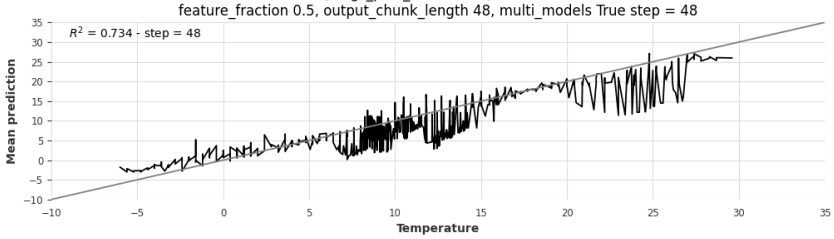
```
# 'feature_fraction': 0.5, 'multi_models': True, 'n_estimators': 200,
# Backtest RMSE all: 2.075639
# Backtest MAE all:  1.535223
# Backtest RMSE 48th: 2.583967
# Backtest MAE 48th:  2.003548

# inc_cols1 = ['y_des', 'dew.point_des', 'humidity', 'pressure',
#              'wind.speed.max.sqrt.x', 'wind.speed.max.sqrt.y',
#              'y_des_window_48_hist_mode', 'dew.point_des_window_48_hist_mode',
#              'humidity_window_48_hist_mode', 'pressure_window_48_hist_mode',
#              'y_des_shadow',]
# FUT_COV  = ['irradiance', 'za_rad']
# init_lags1 = [-1, -2, -4]
# mid_lags1  = [i for i in range(-6, -52, -4) if i > -50]
# target_lags1 = pastcov_lags1 = init_lags1 + mid_lags1
# futcov_lags1  = (1, 48)
# 'feature_fraction': 0.5, 'multi_models': True, 'n_estimators': 200,
# Backtest RMSE all: 2.096232
# Backtest MAE all:  1.550654
# Backtest RMSE 48th: 2.632937
# Backtest MAE 48th:  2.033491

# inc_cols1 = ['y_des', 'dew.point_des', 'humidity', 'pressure',
#              'wind.speed.max.sqrt.x', 'wind.speed.max.sqrt.y',
#              'y_des_window_24_hist_mode', 'dew.point_des_window_24_hist_mode',
#              'y_des_shadow',]
# FUT_COV  = ['irradiance', 'za_rad']
# init_lags1 = [-1, -2, -4]
# mid_lags1  = [i for i in range(-6, -52, -4) if i > -50]
# target_lags1 = pastcov_lags1 = init_lags1 + mid_lags1
# futcov_lags1  = (1, 48)
# 'feature_fraction': 0.5, 'multi_models': True, 'n_estimators': 200,
# Backtest RMSE all: 2.065958
# Backtest MAE all:  1.527929
# Backtest RMSE 48th: 2.618883
# Backtest MAE 48th:  2.027644
```

```
Backtest RMSE all: 2.065958
Backtest MAE all:  1.527929

# Backtest RMSE 48th: 2.618883
# Backtest MAE 48th:  2.027644

Backtest RMSE miss==0: 1.99705
Backtest MAE miss==0:  1.491759

Backtest RMSE miss==1: 3.072288
Backtest MAE miss==1:  2.195961


backtest['y_des']:
count    830304
mean     -0.850793
std      3.019367
min      -11.609231
25%      -2.934387
50%      -1.128022
75%      1.068177
max      12.354061

valid_df['y_des']:
count    17520.000000
mean        -0.855508
std          3.651553
min         -9.967023
25%         -3.448224
50%         -1.195697
75%          1.403691
max         13.030489
Name: y_des, dtype: float64
```

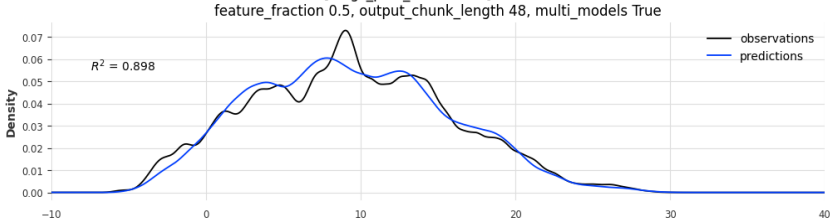
Multi-step lgbm - lags [-1, -2, -4, -6, -10, -14, -18, -22, -26, -30, -34, -38, -42, -46], lags_past_covariates [-1, -2, -4, -6, -10, -14, -18, -22, -26, -30, -34, -38, -42, -46], lags_future_covariates (1, 48) feature_fraction 0.5, output_chunk_length 48, multi_models True



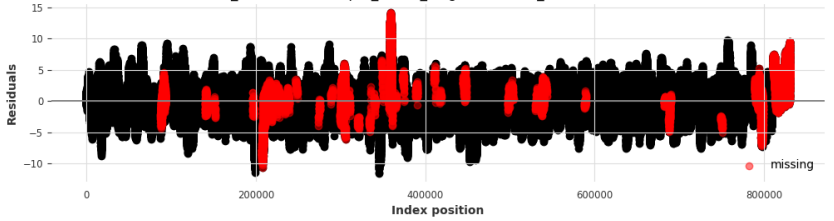
Multi-step lgbm - lags [-1, -2, -4, -6, -10, -14, -18, -22, -26, -30, -34, -38, -42, -46], lags_past_covariates [-1, -2, -4, -6, -10, -14, -18, -22, -26, -30, -34, -38, -42, -46], lags_future_covariates (1, 48) feature_fraction 0.5, output_chunk_length 48, multi_models True step = 48



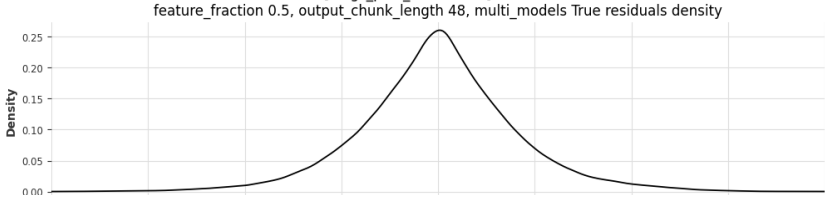
Multi-step lgbm - lags [-1, -2, -4, -6, -10, -14, -18, -22, -26, -30, -34, -38, -42, -46], lags_past_covariates [-1, -2, -4, -6, -10, -14, -18, -22, -26, -30, -34, -38, -42, -46], lags_future_covariates (1, 48) feature_fraction 0.5, output_chunk_length 48, multi_models True

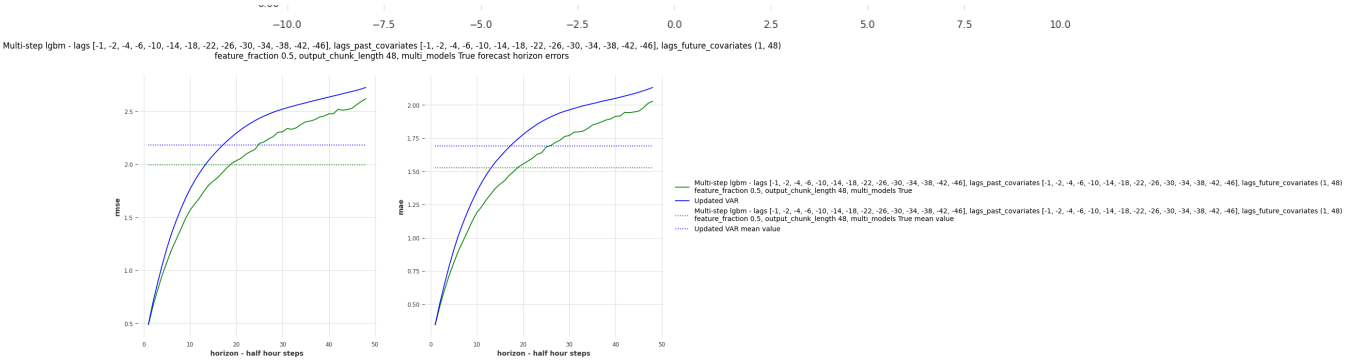


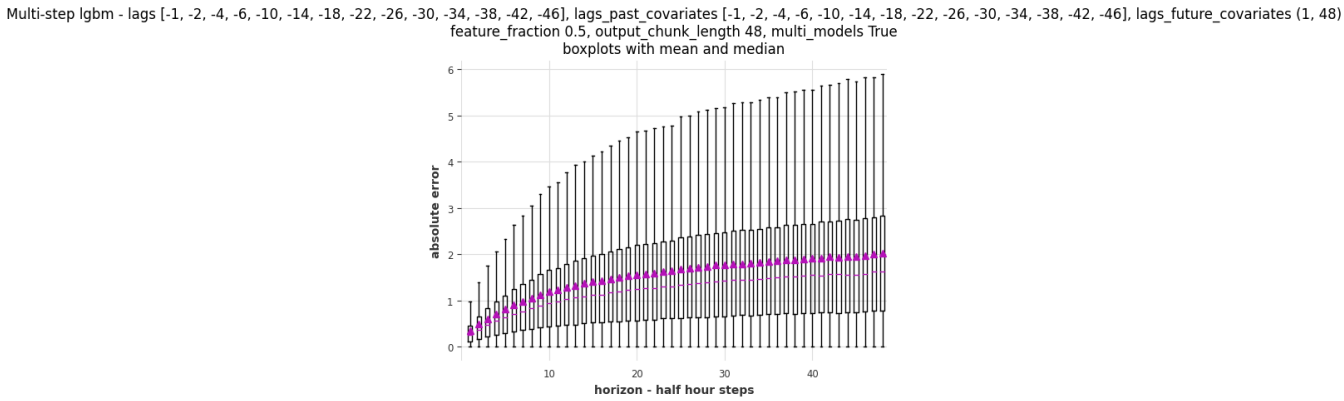
Multi-step lgbm - lags [-1, -2, -4, -6, -10, -14, -18, -22, -26, -30, -34, -38, -42, -46], lags_past_covariates [-1, -2, -4, -6, -10, -14, -18, -22, -26, -30, -34, -38, -42, -46], lags_future_covariates (1, 48) feature_fraction 0.5, output_chunk_length 48, multi_models True residuals

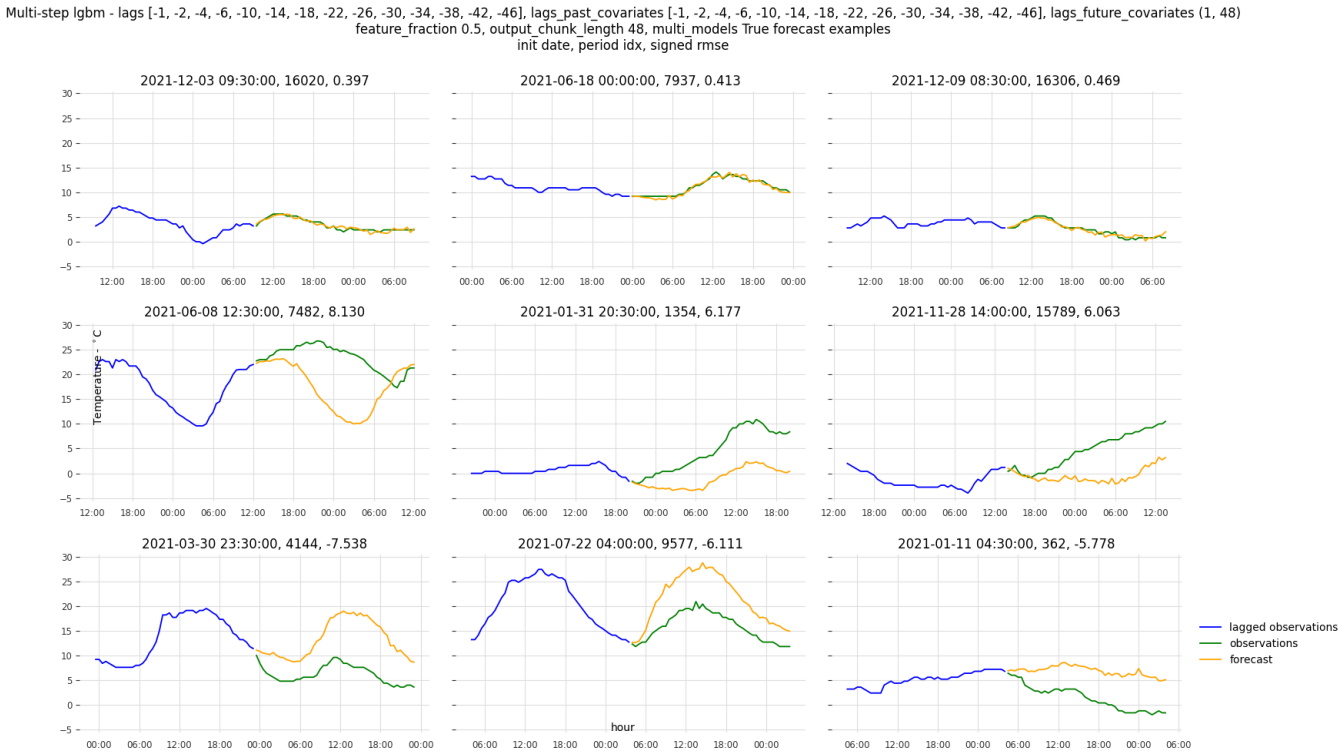


Multi-step lgbm - lags [-1, -2, -4, -6, -10, -14, -18, -22, -26, -30, -34, -38, -42, -46], lags_past_covariates [-1, -2, -4, -6, -10, -14, -18, -22, -26, -30, -34, -38, -42, -46], lags_future_covariates (1, 48) feature_fraction 0.5, output_chunk_length 48, multi_models True residuals density




Multi-step lgbm - lags [-1, -2, -4, -6, -10, -14, -18, -22, -26, -30, -34, -38, -42, -46], lags_past_covariates [-1, -2, -4, -6, -10, -14, -18, -22, -26, -30, -34, -38, -42, -46], lags_future_covariates (1, 48)
feature_fraction 0.5, output_chunk_length 48, multi_models True forecast horizon errors
rmse
horizon - half hour steps
mse
horizon - half hour steps
Multi-step lgbm - lags [-1, -2, -4, -6, -10, -14, -18, -22, -26, -30, -34, -38, -42, -46], lags_past_covariates [-1, -2, -4, -6, -10, -14, -18, -22, -26, -30, -34, -38, -42, -46], lags_future_covariates (1, 48)
feature_fraction 0.5, output_chunk_length 48, multi_models True
Updated VAR
Multi-step lgbm - lags [-1, -2, -4, -6, -10, -14, -18, -22, -26, -30, -34, -38, -42, -46], lags_past_covariates [-1, -2, -4, -6, -10, -14, -18, -22, -26, -30, -34, -38, -42, -46], lags_future_covariates (1, 48)
feature_fraction 0.5, output_chunk_length 48, multi_models True mean value
Updated VAR mean value


Multi-step lgbm - lags [-1, -2, -4, -6, -10, -14, -18, -22, -26, -30, -34, -38, -42, -46], lags_past_covariates [-1, -2, -4, -6, -10, -14, -18, -22, -26, -30, -34, -38, -42, -46], lags_future_covariates (1, 48)
feature_fraction 0.5, output_chunk_length 48, multi_models True
boxplots with mean and median
absolute error
horizon - half hour steps


Multi-step lgbm - lags [-1, -2, -4, -6, -10, -14, -18, -22, -26, -30, -34, -38, -42, -46], lags_past_covariates [-1, -2, -4, -6, -10, -14, -18, -22, -26, -30, -34, -38, -42, -46], lags_future_covariates (1, 48)
feature_fraction 0.5, output_chunk_length 48, multi_models True forecast examples
init date, period idx, signed rmse
2021-12-03 09:30:00, 16020, 0.397
2021-06-18 00:00:00, 7937, 0.413
2021-12-09 08:30:00, 16306, 0.469
2021-06-08 12:30:00, 7482, 8.130
2021-01-31 20:30:00, 1354, 6.177
2021-11-28 14:00:00, 15789, 6.063
2021-03-30 23:30:00, 4144, -7.538
2021-07-22 04:00:00, 9577, -6.111
2021-01-11 04:30:00, 362, -5.778
Temperature - °C
hour
lagged observations
observations
forecast

The penultimate plot comparing the lightGBM model, in green, with vector autoregression (VAR), in blue, indicates this is an acceptible starting model. The rmse and mae values also indicate this is a reasonable first model. The 'missing' values in red on the predictions versus observations plot are notable. These values were initially missing, so had to be imputed. They would have been better named 'imputed'. In many cases multiple features were simultaneously missing. Multiple imputation simply does not work. Some of the imputed values are unreasonable.

The `plot_multistep_diagnostics` function produces 7 or 8 model diagnostic plots:

- `plot_multistep_obs_vs_preds`
  - observations versus predictions for **all** 48 forecast steps
  - 'missing' values in red
- `plot_multistep_obs_vs_mean_preds_by_step`
  - observations versus predictions for **only** step 48 forecasts
- `plot_multistep_obs_preds_dists`
  - kernel density estimates for both observations and predictions
- `plot_multistep_residuals`
  - residuals for all 48 forecast steps
  - 'missing' values in red
- `plot_multistep_residuals_dist`
  - kernel density estimate for the residuals
- `plot_horizon_metrics`
  - mean rmse and mae values for each forecast step
  - updated VAR model in blue
  - multi-step lightGBM model in green
- `plot_horizon_metrics_boxplots`
  - boxplots of absolute error for each forecast step
- `plot_multistep_forecast_examples`
  - some examples of good and bad forecasts
  - top 3 plots are lowest error forecasts
  - middle 3 plots are largest positive error forecasts
  - bottom 3 plots are largest negative error forecasts

Summarise lightGBM variable importances

The lightGBM variable importance summary function `summarise_multi_model_feat_imps` is similar to the F-statistic summary function, `summarise_multi_step_feat_sel_scores`.

The `summarise_multi_model_feat_imps` function summarises variable importance by:

- Tables
  - model
  - lag
  - type - target, past covariate, future covariate
  - window
  - feature
  - transform
  - shift if present
- Boxplots
  - variable importance
  - type - target, past covariate, future covariate
  - model

        - past lags
        - future lags
        - window
        - shift if present
        - feature transform
        - features
            - above core features
            - between core features
        - transforms
            - above core features
            - between core features
- Surface plots
    - variable imporance by model and lag
        - minimum, mean and maximum
    - variable imporance by model and window
        - minimum, mean and maximum
    - variable imporance by lag and window
        - minimum, mean and maximum
    - cumulative sum of ordered variable importance for each model
        - ordered by largest importance first

When summarising variable importance in tables the following values are calculated:

- count, mean, standard deviation, minimum,
- 25th percentile, 50th percentile, 75th percentile, maximum
- above, between and below core features
- number zero importance
- percent zero importance
- number greater than or equal to shadow feature importance
- number greater than or equal to shadow feature importance

TODO Add recursive feature elimination (RFE) candidates to `get_multi_model_feat_imps` and/or `summarise_multi_model_feat_imps`. Ideally, would like to find lag, window, feature, transform, shift etc RFE candidates across all lightGBM models using mean variable importance and/or shadow variables.

```
title1 = 'model1'  # over-ride long titles
mod1_vi = get_multi_model_feat_imps(model1)
summarise_multi_model_feat_imps(mod1_vi, title1, verbose=True)
```

	model	feature_transform_lag	imp	feature_window_transform	feature_transform	feature	transform	lag	type	shift
0	0	y_des_target_lag-46	34	y_des	y_des	y_des	des	-46	target	0
1	0	y_des_target_lag-42	23	y_des	y_des	y_des	des	-42	target	0
2	0	y_des_target_lag-38	46	y_des	y_des	y_des	des	-38	target	0
3	0	y_des_target_lag-34	25	y_des	y_des	y_des	des	-34	target	0
4	0	y_des_target_lag-30	18	y_des	y_des	y_des	des	-30	target	0
...	...	...	...	...	...	...	...	...	...	...
11419	47	za_rad_futcov_lag45	10	za_rad	za_rad	za_rad	None	45	futcov	0
11420	47	irradiance_futcov_lag46	5	irradiance	irradiance	irradiance	None	46	futcov	0
11421	47	za_rad_futcov_lag46	7	za_rad	za_rad	za_rad	None	46	futcov	0
11422	47	irradiance_futcov_lag47	2	irradiance	irradiance	irradiance	None	47	futcov	0
11423	47	za_rad_futcov_lag47	8	za_rad	za_rad	za_rad	None	47	futcov	0

11424 rows × 16 columns

	count	mean	std	min	25%	50%	75%	max	sum	num_0	pc_0	num_shad_geq	pc_shad_geq
model													
0	238	25.210084	34.783811	0	12	18	26	359	6000	4	1.680672	4	1.680672
1	238	25.210084	29.316654	0	11	18	30	314	6000	4	1.680672	4	1.680672
2	238	25.210084	25.439056	0	10	21	31	251	6000	1	0.420168	1	0.420168
3	238	25.210084	23.979025	0	10	21	34	221	6000	1	0.420168	1	0.420168
4	238	25.210084	23.381219	0	9	20	35	190	6000	5	2.100840	5	2.100840
5	238	25.210084	23.034325	0	9	21	37	176	6000	11	4.621849	11	4.621849
6	238	25.210084	23.187322	0	8	19	36	165	6000	5	2.100840	5	2.100840
7	238	25.210084	23.224232	0	8	19	38	146	6000	9	3.781513	9	3.781513
8	238	25.210084	23.597495	0	8	18	37	142	6000	11	4.621849	11	4.621849
9	238	25.210084	24.151028	0	7	17	36	136	6000	10	4.201681	10	4.201681
10	238	25.210084	24.003472	0	7	16	38	129	6000	12	5.042017	12	5.042017
11	238	25.210084	24.687712	0	6	16	37	136	6000	11	4.621849	11	4.621849
12	238	25.210084	25.585918	0	6	16	37	129	6000	17	7.142857	17	7.142857
13	238	25.210084	25.735222	0	6	15	38	124	6000	11	4.621849	11	4.621849
14	238	25.210084	26.233574	0	6	14	39	130	6000	16	6.722689	16	6.722689
15	238	25.210084	26.122520	0	5	14	39	131	6000	17	7.142857	17	7.142857
16	238	25.210084	26.104908	0	5	14	40	132	6000	16	6.722689	16	6.722689
17	238	25.210084	26.896229	0	5	14	41	130	6000	18	7.563025	18	7.563025
18	238	25.210084	27.676819	0	4	13	41	130	6000	19	7.983193	19	7.983193
19	238	25.210084	27.023001	0	5	14	40	140	6000	18	7.563025	18	7.563025
20	238	25.210084	27.786976	0	5	12	40	138	6000	19	7.983193	19	7.983193
21	238	25.210084	27.190801	0	5	13	40	147	6000	15	6.302521	15	6.302521
22	238	25.210084	26.942782	0	5	13	41	141	6000	18	7.563025	18	7.563025
23	238	25.210084	27.820059	0	5	12	43	129	6000	16	6.722689	16	6.722689
24	238	25.210084	27.484825	0	5	12	40	119	6000	18	7.563025	18	7.563025
25	238	25.210084	28.101046	0	5	12	41	126	6000	17	7.142857	17	7.142857
26	238	25.210084	27.536818	0	5	13	37	125	6000	16	6.722689	16	6.722689
27	238	25.210084	27.558874	0	5	12	39	127	6000	19	7.983193	19	7.983193
28	238	25.210084	28.269312	0	5	11	39	128	6000	15	6.302521	15	6.302521
29	238	25.210084	28.789758	0	4	12	40	150	6000	16	6.722689	16	6.722689
30	238	25.210084	28.177069	0	5	12	40	140	6000	16	6.722689	16	6.722689
31	238	25.210084	28.786387	0	5	11	41	144	6000	18	7.563025	18	7.563025
32	238	25.210084	28.776858	0	5	12	40	150	6000	14	5.882353	14	5.882353
33	238	25.210084	28.950379	0	5	13	40	145	6000	16	6.722689	16	6.722689
34	238	25.210084	28.400949	0	5	11	42	139	6000	17	7.142857	17	7.142857

35	238	25.210084	28.664324	0	5	12	40	158	6000	17	7.142857	17	7.142857
36	238	25.210084	29.139523	0	5	12	42	158	6000	16	6.722689	16	6.722689
37	238	25.210084	29.618923	0	5	11	39	166	6000	16	6.722689	16	6.722689
38	238	25.210084	29.548180	0	5	11	42	175	6000	15	6.302521	15	6.302521
39	238	25.210084	28.971505	0	5	12	40	154	6000	15	6.302521	15	6.302521
40	238	25.210084	30.248862	0	4	11	38	164	6000	13	5.462185	13	5.462185
41	238	25.210084	30.455023	0	4	10	39	175	6000	16	6.722689	16	6.722689
42	238	25.210084	30.583323	0	4	12	39	181	6000	21	8.823529	21	8.823529
43	238	25.210084	30.631848	0	3	11	41	175	6000	21	8.823529	21	8.823529
44	238	25.210084	30.556408	0	4	11	41	184	6000	20	8.403361	20	8.403361
45	238	25.210084	31.689787	0	4	11	41	194	6000	20	8.403361	20	8.403361
46	238	25.210084	31.537500	0	4	10	41	196	6000	23	9.663866	23	9.663866
47	238	25.210084	31.333614	0	3	10	39	197	6000	17	7.142857	17	7.142857

	count	mean	std	min	25%	50%	75%	max	sum	num_0	pc_0	num_shad_geq	pc_shad_geq
lag													
-46	480	45.350000	41.675934	0	11	27	71	154	21768	38	7.916667	38	7.916667
-42	480	37.389583	32.635654	0	10	28	61	122	17947	32	6.666667	32	6.666667
-38	480	32.854167	25.858608	0	9	30	52	103	15770	39	8.125000	39	8.125000
-34	480	29.785417	23.312567	0	10	26	45	96	14297	42	8.750000	42	8.750000
-30	480	30.800000	23.104410	0	11	29	48	86	14784	40	8.333333	40	8.333333
-26	480	29.481250	21.766450	0	11	27	44	93	14151	40	8.333333	40	8.333333
-22	480	28.768750	21.095040	0	11	25	43	81	13809	42	8.750000	42	8.750000
-18	480	29.835417	21.847832	0	11	28	46	93	14321	41	8.541667	41	8.541667
-14	480	31.114583	23.642461	0	11	27	48	106	14935	34	7.083333	34	7.083333
-10	480	30.495833	20.968213	0	13	29	45	95	14638	32	6.666667	32	6.666667
-6	480	34.679167	22.533510	0	16	32	52	118	16646	32	6.666667	32	6.666667
-4	480	34.802083	24.996626	0	16	31	49	192	16705	41	8.541667	41	8.541667
-2	480	45.095833	30.234421	0	23	43	61	159	21646	43	8.958333	43	8.958333
-1	576	62.315972	50.428089	0	14	63	93	359	35894	45	7.812500	45	7.812500
0	96	7.427083	7.692436	0	3	5	9	50	713	4	4.166667	4	4.166667
1	96	9.406250	14.602507	0	2	5	10	85	903	12	12.500000	12	12.500000
2	96	10.468750	13.234686	0	3	6	13	76	1005	7	7.291667	7	7.291667
3	96	10.531250	12.779056	0	2	5	11	55	1011	6	6.250000	6	6.250000
4	96	12.437500	13.834216	0	4	7	14	58	1194	4	4.166667	4	4.166667
5	96	10.208333	10.719305	0	2	6	13	46	980	3	3.125000	3	3.125000
6	96	11.458333	11.193905	0	3	7	17	50	1100	3	3.125000	3	3.125000
7	96	11.072917	9.611665	0	4	8	18	40	1063	2	2.083333	2	2.083333
8	96	10.927083	10.971009	0	3	6	15	54	1049	7	7.291667	7	7.291667
9	96	12.958333	11.754432	0	4	8	21	47	1244	3	3.125000	3	3.125000
10	96	12.229167	10.670648	0	4	8	17	47	1174	1	1.041667	1	1.041667
11	96	12.052083	11.230944	0	3	8	17	40	1157	4	4.166667	4	4.166667
12	96	11.697917	9.897495	0	4	8	16	40	1123	1	1.041667	1	1.041667
13	96	11.604167	8.822792	0	5	9	16	38	1114	2	2.083333	2	2.083333
14	96	10.729167	9.095377	0	5	7	15	40	1030	3	3.125000	3	3.125000
15	96	10.302083	7.690212	0	5	9	14	39	989	1	1.041667	1	1.041667
16	96	9.125000	6.611394	0	4	8	12	30	876	2	2.083333	2	2.083333
18	96	8.875000	7.365996	0	3	8	11	34	852	3	3.125000	3	3.125000
19	96	8.510417	6.271061	0	4	7	11	27	817	1	1.041667	1	1.041667
20	96	8.187500	7.196947	0	3	7	10	34	786	7	7.291667	7	7.291667
21	96	7.489583	6.683829	0	3	6	10	37	719	2	2.083333	2	2.083333

22	96	6.552083	6.167606	0	2	5	8	28	629	4	4.166667	4	4.166667
23	96	6.718750	7.013826	0	2	5	8	33	645	5	5.208333	5	5.208333
24	96	6.302083	5.477456	0	2	5	8	25	605	7	7.291667	7	7.291667
25	96	6.864583	6.596043	0	2	5	8	28	659	3	3.125000	3	3.125000
26	96	6.697917	6.338055	0	3	5	8	30	643	3	3.125000	3	3.125000
27	96	7.406250	6.835785	0	3	5	10	33	711	4	4.166667	4	4.166667
28	96	7.437500	6.045115	0	3	5	10	25	714	2	2.083333	2	2.083333
29	96	7.656250	6.905113	0	3	5	11	38	735	3	3.125000	3	3.125000
30	96	6.885417	5.237255	0	3	5	9	21	661	1	1.041667	1	1.041667
31	96	8.322917	6.717604	0	3	6	12	26	799	1	1.041667	1	1.041667
33	96	9.354167	7.672204	0	4	6	16	30	898	2	2.083333	2	2.083333
34	96	9.531250	7.976219	0	3	6	14	33	915	1	1.041667	1	1.041667
35	96	8.729167	7.553987	0	4	6	13	34	838	1	1.041667	1	1.041667
37	96	9.718750	8.229220	0	4	6	15	30	933	3	3.125000	3	3.125000
38	96	7.822917	5.632602	0	3	6	11	25	751	2	2.083333	2	2.083333
39	96	6.604167	4.470910	0	3	6	10	17	634	5	5.208333	5	5.208333
40	96	8.208333	5.442361	0	3	7	11	24	788	3	3.125000	3	3.125000
41	96	9.281250	7.429250	0	4	7	13	32	891	2	2.083333	2	2.083333
42	96	7.302083	5.494724	0	3	6	9	26	701	1	1.041667	1	1.041667
44	96	7.979167	6.331175	0	3	6	10	28	766	3	3.125000	3	3.125000
45	96	6.625000	5.569182	0	3	5	9	32	636	4	4.166667	4	4.166667
46	96	6.114583	6.196552	0	2	4	9	36	587	3	3.125000	3	3.125000
47	96	4.229167	4.731705	0	1	3	6	21	406	14	14.583333	14	14.583333

type	**count**	**mean**	**std**	**min**	**25%**	**50%**	**75%**	**max**	**sum**	**num_0**	**pc_0**	**num_shad_geq**	**pc_shad_geq**
futcov	4704	8.773810	8.442087	0	3	6	12	85	41272	161	3.422619	161	3.422619
pastcov	6048	36.958333	30.265124	0	11	34	55	247	223524	535	8.845899	535	8.845899

window	**count**	**mean**	**std**	**min**	**25%**	**50%**	**75%**	**max**	**sum**	**num_0**	**pc_0**	**num_shad_geq**	**pc_shad_geq**
0	10080	20.451587	24.938186	0	5	11	28	359	206152	696	6.904762	696	6.904762

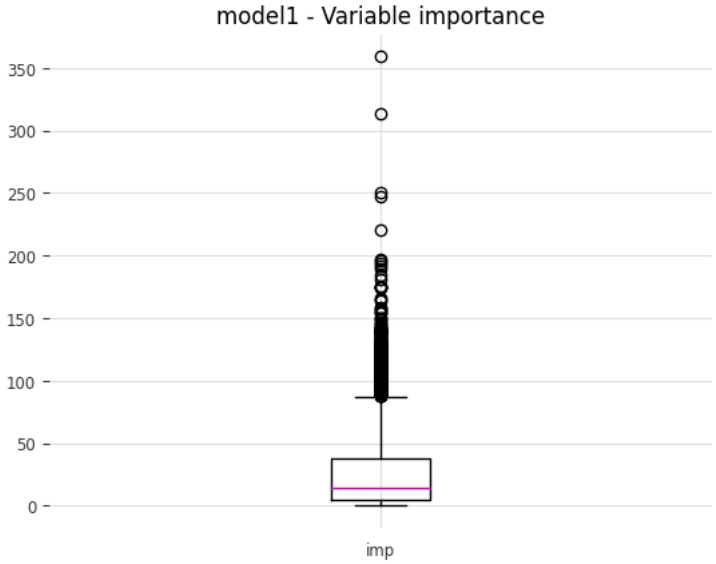
feature	**count**	**mean**	**std**	**min**	**25%**	**50%**	**75%**	**max**	**sum**	**num_0**	**pc_0**	**num_shad_geq**	**pc_shad_geq**
dew.point_des	1344	39.980655	30.283391	0	10	41	64	128	53734	2	0.148810	2	0.148810
irradiance	2352	5.373724	5.351868	0	2	4	7	38	12639	149	6.335034	149	6.335034
y_des	2688	28.121652	30.790087	0	5	18	44	359	75591	533	19.828869	533	19.828869
za_rad	2352	12.173895	9.528615	0	6	9	16	85	28633	12	0.510204	12	0.510204

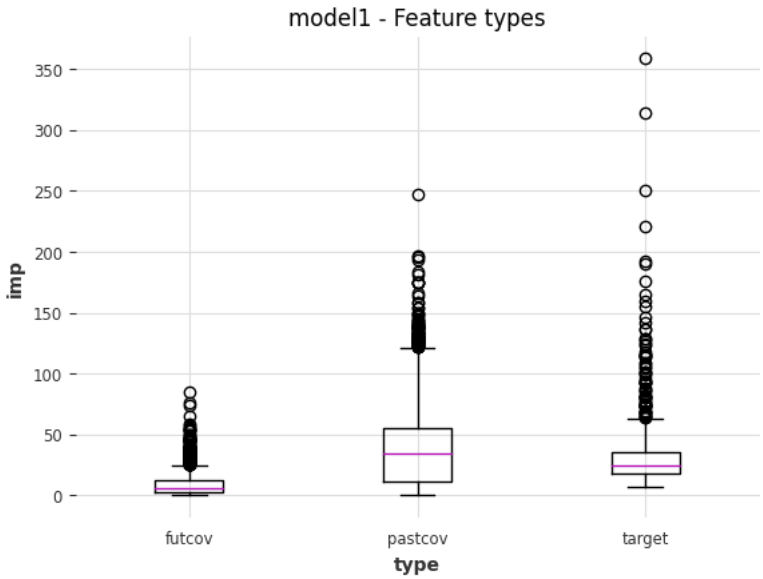
transform	**count**	**mean**	**std**	**min**	**25%**	**50%**	**75%**	**max**	**sum**	**num_0**	**pc_0**	**num_shad_geq**	**pc_shad_geq**
None	7392	21.465774	24.898466	0	5	11	32	197	158675	161	2.178030	161	2.178030
des	2016	23.170139	26.365278	0	9	15	26	359	46711	2	0.099206	2	0.099206
shadow	672	1.139881	4.321445	0	0	0	0	43	766	533	79.315476	533	79.315476

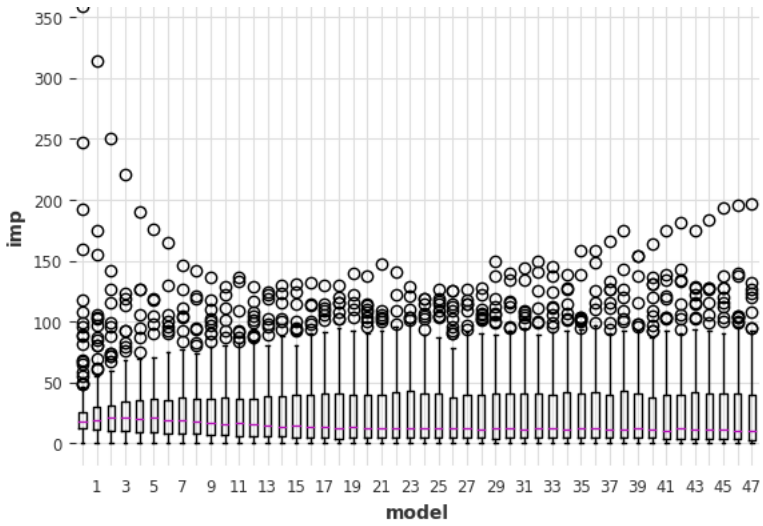
	model	**feature_transform_lag**	**imp**	**feature_window_transform**	**feature_transform**	**feature**	**transform**	**lag**	**type**	**shif**
965	4	y_des_target_lag-1	190	y_des	y_des	y_des	des	-1	target	
11	0	y_des_target_lag-4	192	y_des	y_des	y_des	des	-4	target	
10844	45	pressure_pastcov_lag-1	194	pressure	pressure	pressure	None	-1	pastcov	
11082	46	pressure_pastcov_lag-1	196	pressure	pressure	pressure	None	-1	pastcov	
11320	47	pressure_pastcov_lag-1	197	pressure	pressure	pressure	None	-1	pastcov	

727	3	y_des_target_lag-1	221	y_des	y_des	y_des	des	-1	target	
131	0	y_des_pastcov_lag-1	247	y_des	y_des	y_des	des	-1	pastcov	
489	2	y_des_target_lag-1	251	y_des	y_des	y_des	des	-1	target	
251	1	y_des_target_lag-1	314	y_des	y_des	y_des	des	-1	target	
13	0	y_des_target_lag-1	359	y_des	y_des	y_des	des	-1	target	

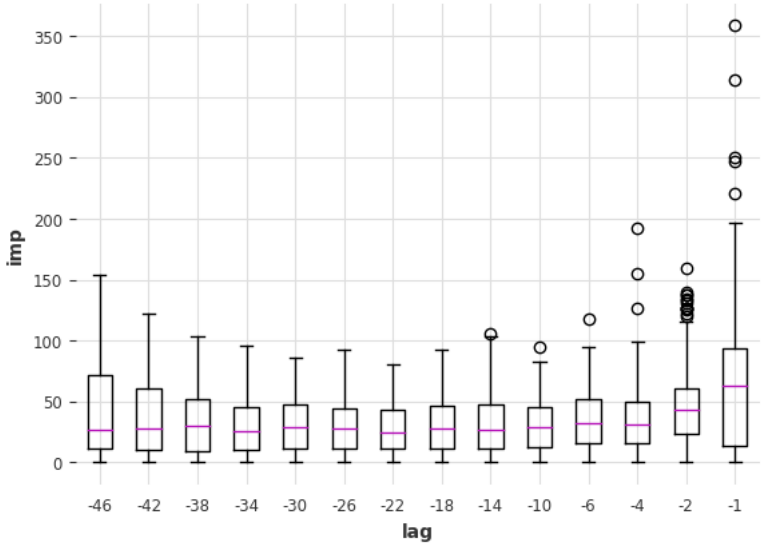
	model	feature_transform_lag	imp	feature_window_transform	feature_transform	feature	transform	lag	type	shi
2023	8	y_des_shadow_pastcov_lag-4	0	y_des_shadow	y_des_shadow	y_des	shadow	-4	pastcov	
5575	23	y_des_shadow_pastcov_lag-10	0	y_des_shadow	y_des_shadow	y_des	shadow	-10	pastcov	
10378	43	irradiance_futcov_lag1	0	irradiance	irradiance	irradiance	None	1	futcov	
5566	23	y_des_shadow_pastcov_lag-14	0	y_des_shadow	y_des_shadow	y_des	shadow	-14	pastcov	
10374	43	irradiance_futcov_lag-1	0	irradiance	irradiance	irradiance	None	-1	futcov	
10371	43	y_des_shadow_pastcov_lag-1	0	y_des_shadow	y_des_shadow	y_des	shadow	-1	pastcov	
10362	43	y_des_shadow_pastcov_lag-2	0	y_des_shadow	y_des_shadow	y_des	shadow	-2	pastcov	
10353	43	y_des_shadow_pastcov_lag-4	0	y_des_shadow	y_des_shadow	y_des	shadow	-4	pastcov	
10344	43	y_des_shadow_pastcov_lag-6	0	y_des_shadow	y_des_shadow	y_des	shadow	-6	pastcov	
10335	43	y_des_shadow_pastcov_lag-10	0	y_des_shadow	y_des_shadow	y_des	shadow	-10	pastcov	



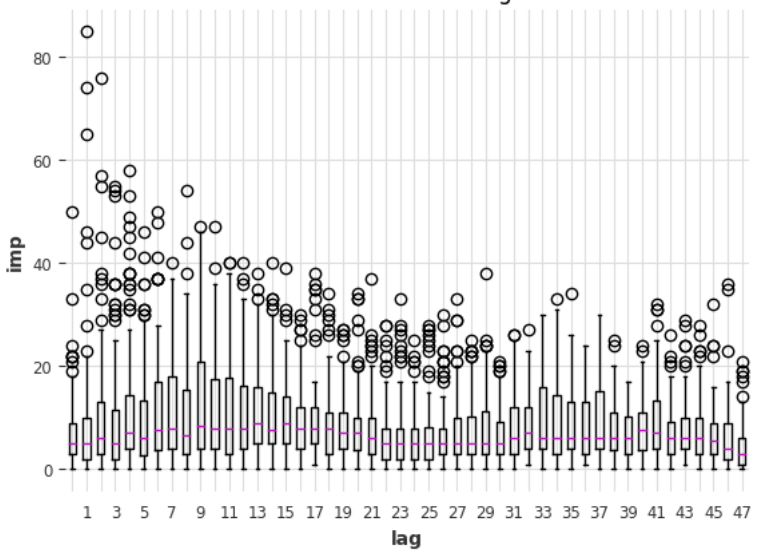




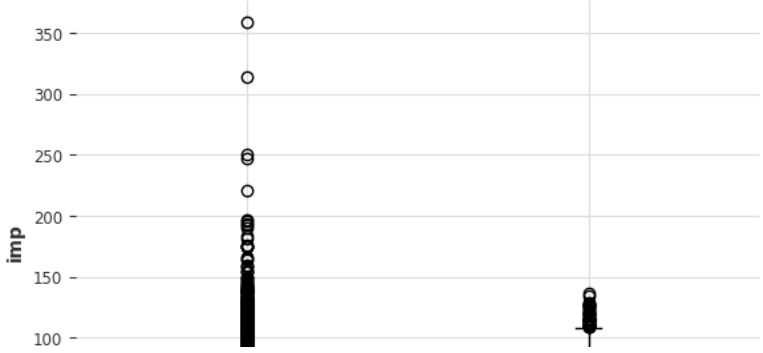
model1 - Past Lags

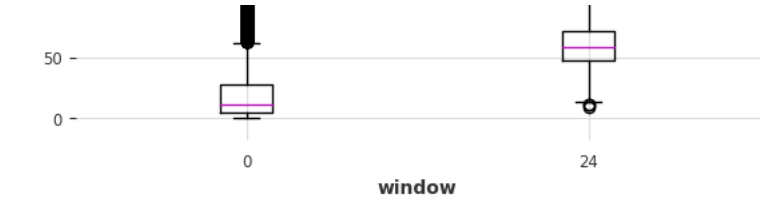


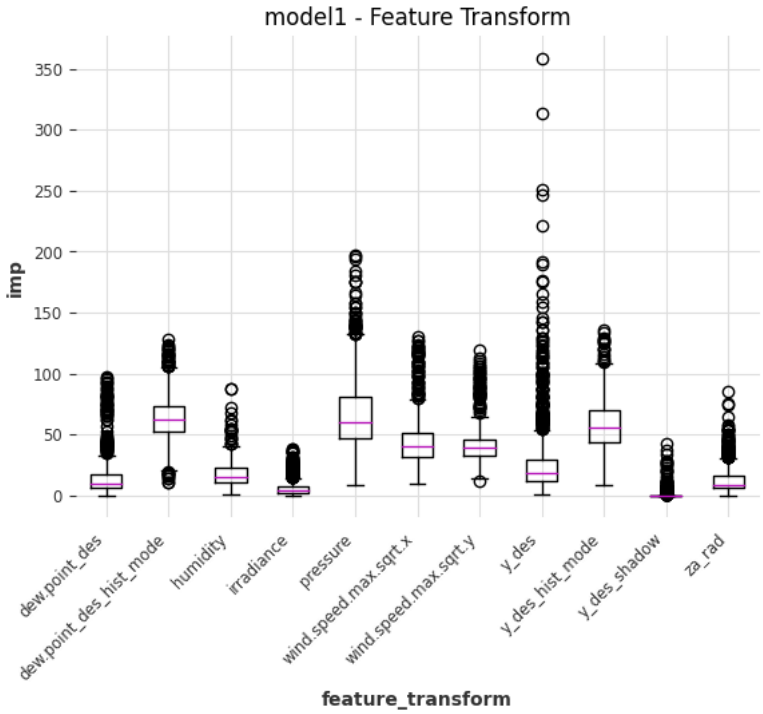
model1 - Future Lags

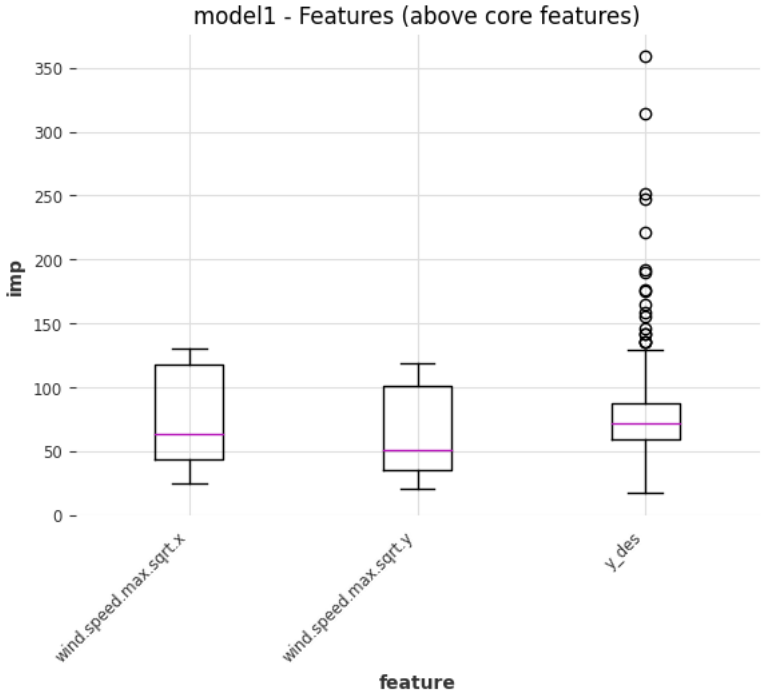


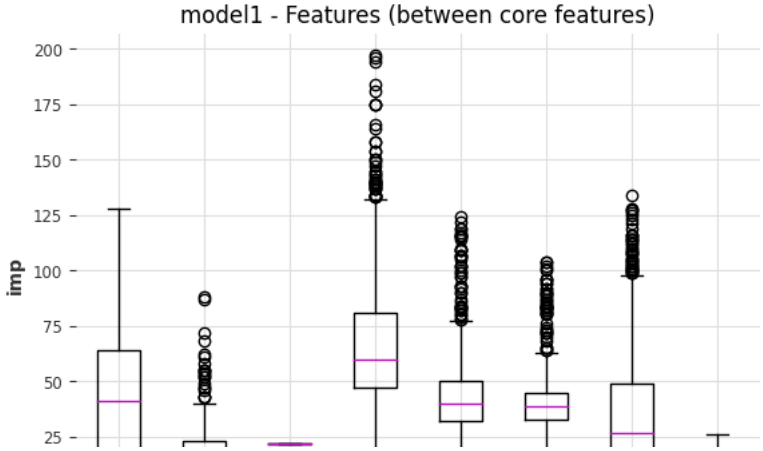
model1 - Windows

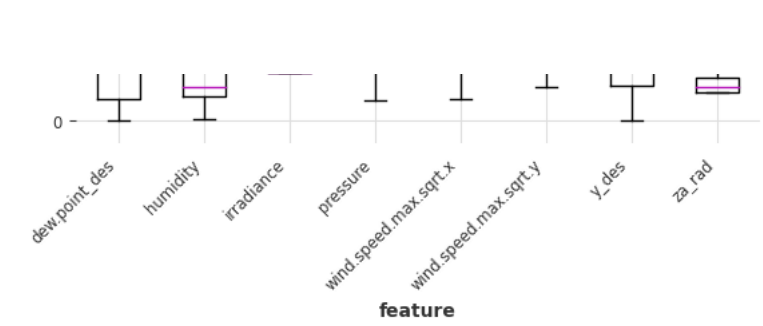


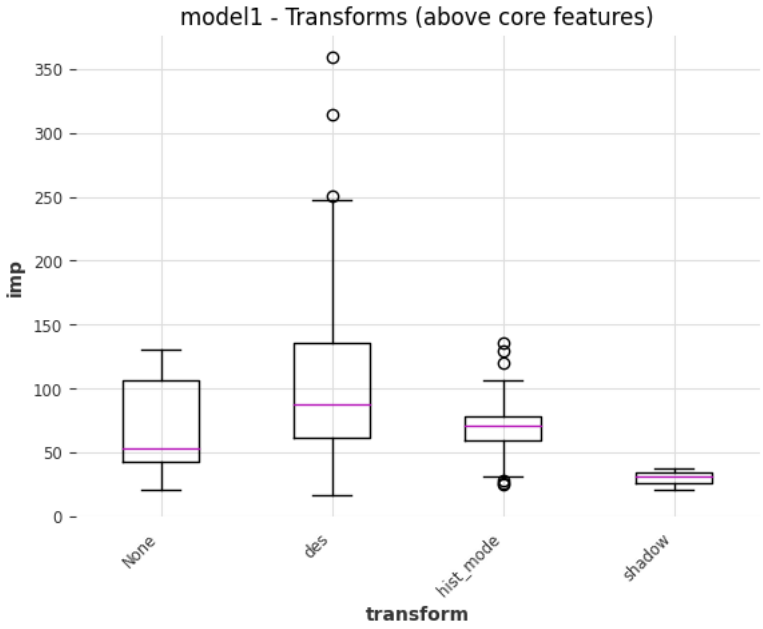

50
0
0
24
window

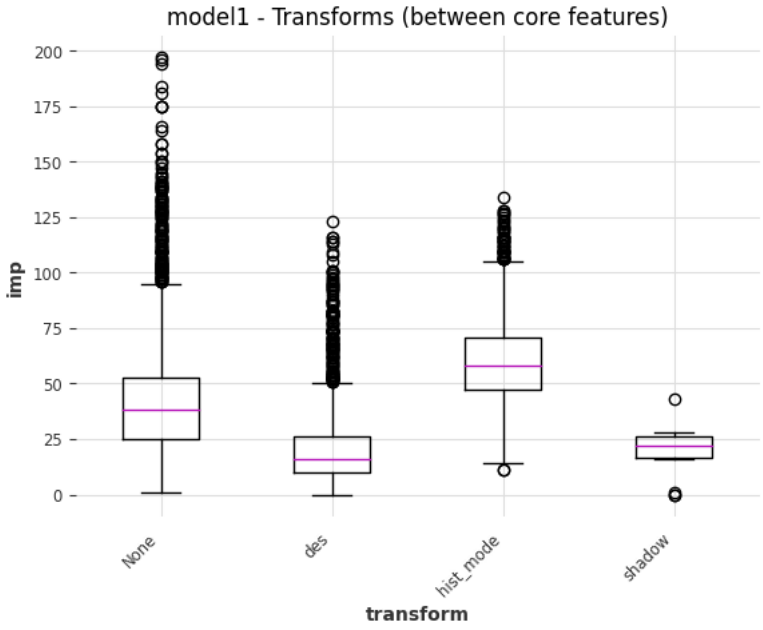

model1 - Feature Transform
350
300
250
200
150
100
50
0
imp
dew.point_des
dew.point_des_hist_mode
humidity
irradiance
pressure
wind.speed.max.sqrt.x
wind.speed.max.sqrt.y
y_des
y_des_hist_mode
y_des_shadow
za_rad
feature_transform

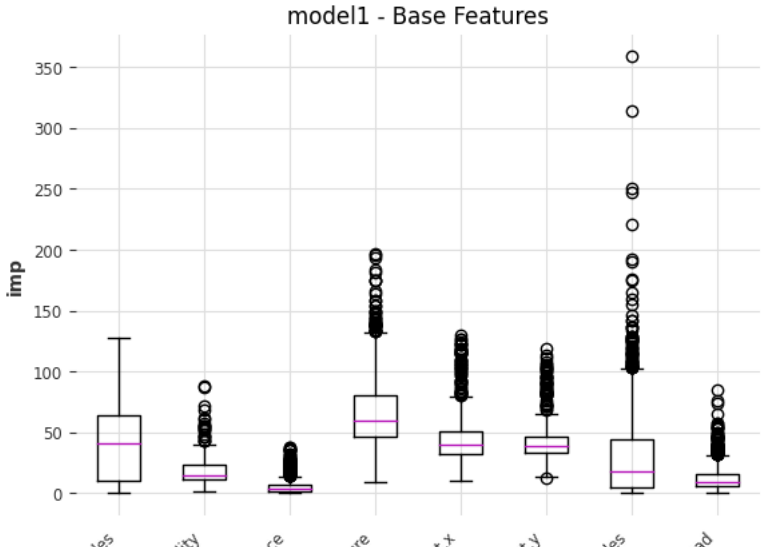

model1 - Features (above core features)
350
300
250
200
150
100
50
0
imp
wind.speed.max.sqrt.x
wind.speed.max.sqrt.y
y_des
feature

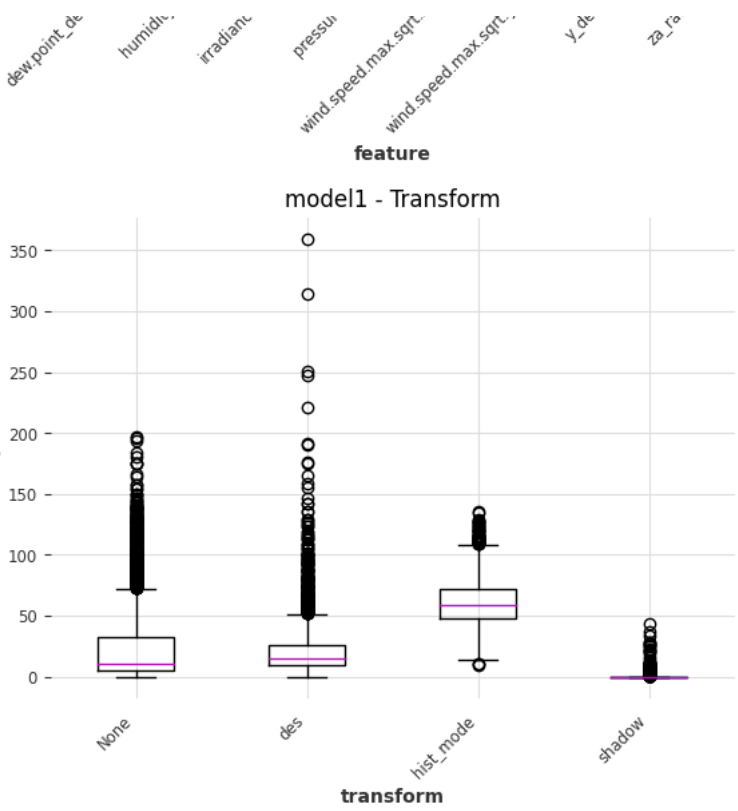

model1 - Features (between core features)
200
175
150
125
100
75
50
25
imp


0
dew.point_des
humidity
irradiance
pressure
wind.speed.max.sqrt.x
wind.speed.max.sqrt.y
y_des
za_rad
feature

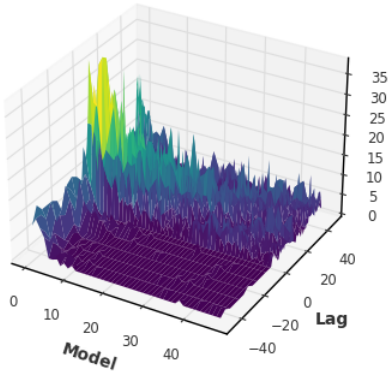

model1 - Transforms (above core features)
imp
0
50
100
150
200
250
300
350
None
des
hist_mode
shadow
transform


model1 - Transforms (between core features)
imp
0
25
50
75
100
125
150
175
200
None
des
hist_mode
shadow
transform

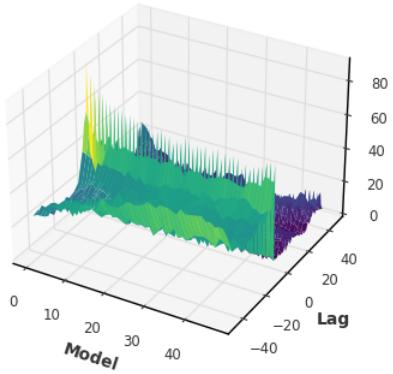

model1 - Base Features
imp
0
50
100
150
200
250
300
350



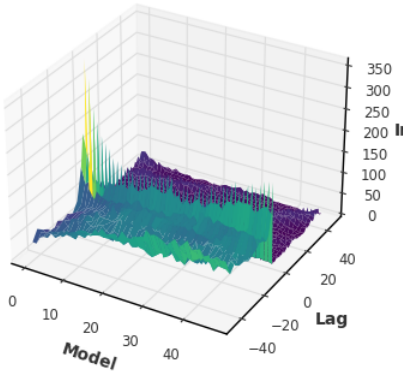
model1 - min



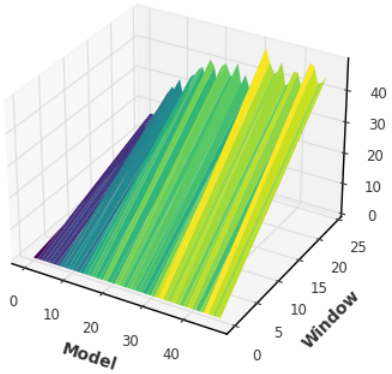
model1 - mean



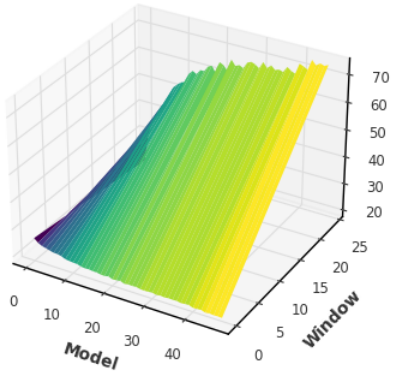
model1 - max



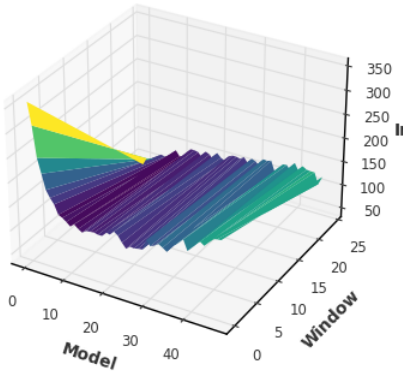
model1 - min

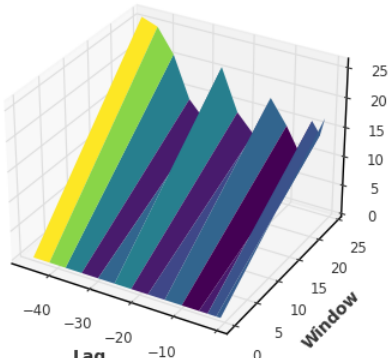


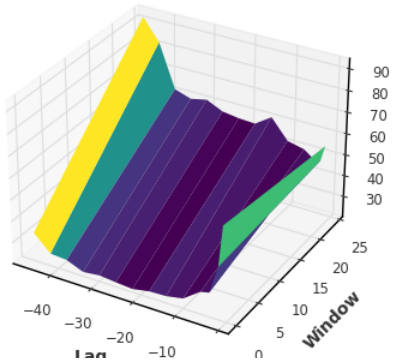
model1 - mean

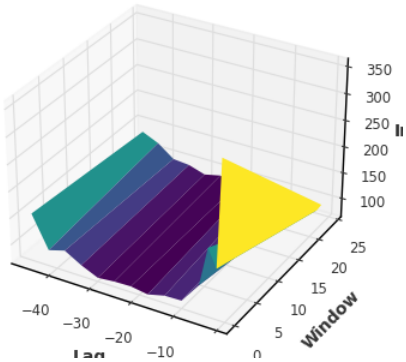


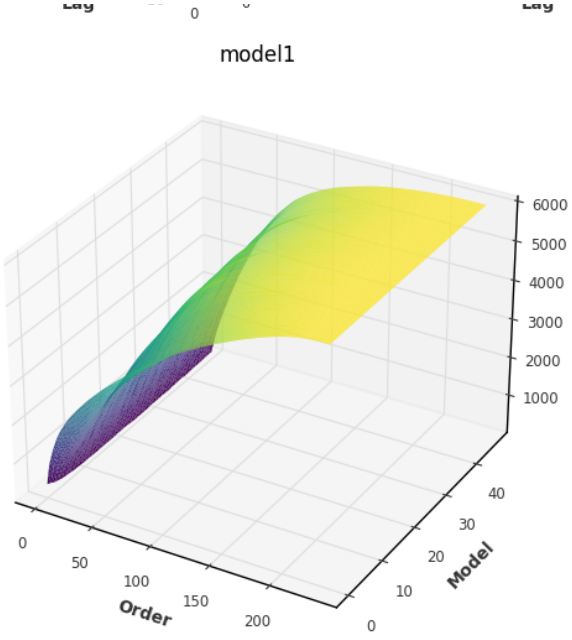
model1 - max

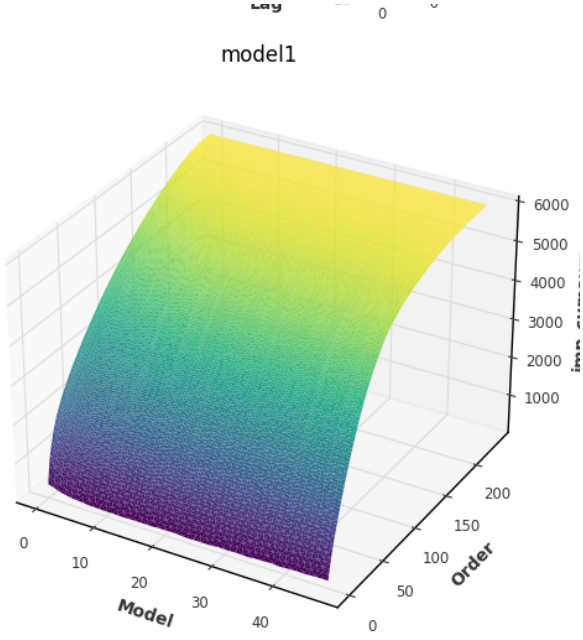












Last commit before removal for brevity:

- every 6th lag
- every 4th lag
- every 3rd lag
- every 2nd lag

default feature set experiments (every 4th lag):

- meterology only
  - indicates `svp`, `t_pot` and `vp_def` (in that order) are beneficial
- solar only
- gradient only
  - particularly useful at earlier lags: -1, -2
- histogram mode
  - window 24
    - potentially useful but needs further experimentation
  - window 24 y_des and dew.point_des only
  - window 48

The commit history contains many other lag selection/feature selection experiments.

Results for the different lag increments:

lag increment	rmse - all	mae - all	rmse - 48th	mae - 48th
6	2.16474	1.604902	2.721805	2.112284
4	2.163194	1.603735	2.705667	2.096854
3	2.164828	1.605242	2.724813	2.110654
2	2.16699	1.606742	2.713683	2.102387

Using every 4th lag improves the rmse and mae at the 48th forecast step. Going forward, I use every 4th lag for the target and past covariates (along with the initial lags: -1, -2, -4). Given more time, I would experiment with different target and past covariate lags.

The lightGBM variable importance indicates the significance of the `svp` (saturation vapor pressure) meteorology feature, which is in agreement with the F-statistic feature scores. The variable importance for the histogram_mode features is relatively low compared to the relevant F-statistic feature scores. This suggests limited usefulness for the F-statistic score experiments.

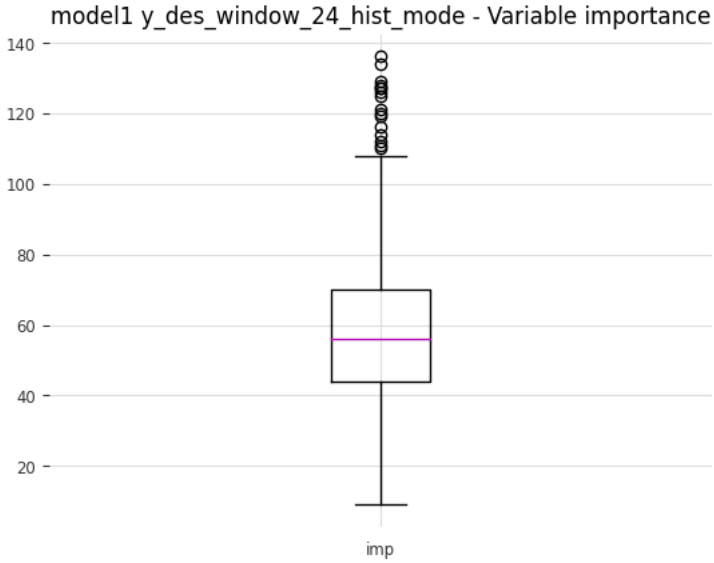
Plot lightGBM importance for individual features, transforms etc

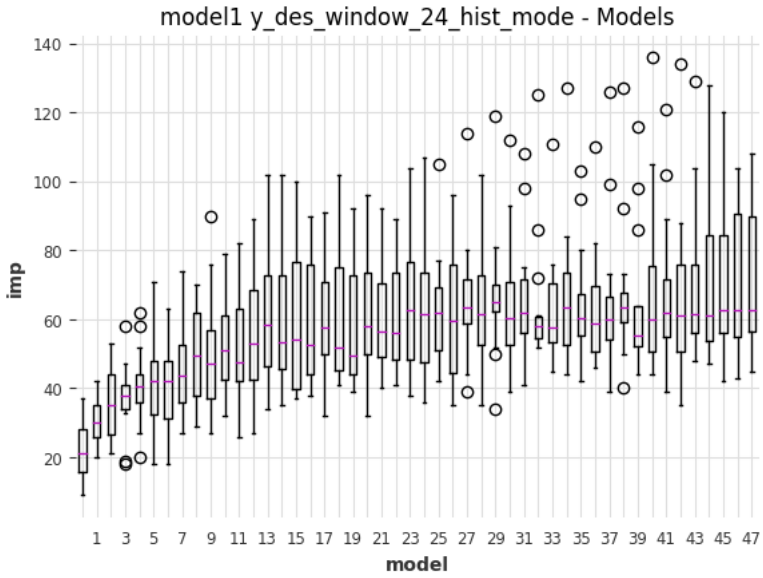
The `summarise_multi_model_feat_imps` function used above gives **aggregated** tables and plots across **multiple** features, transforms, windows etc. The `plot_multi_model_single_feature_imp` function produces plots for **individual** features, transforms, windows etc.

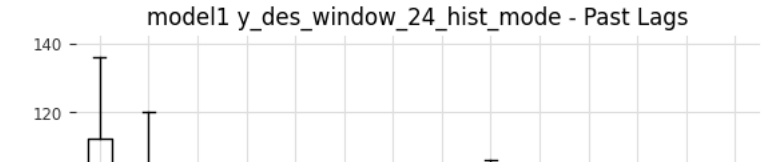
Here I show examples of plotting lightGBM variable importance for **individual** features etc:

- feature_window_transform - `y_des_window_24_hist_mode`
- feature_transform - `y_des_hist_mode`
- feature - `y_des`
- transform - `hist_mode`

```
# 'y_des_window_24_hist_mode'
plot_multi_model_single_feature_imp(mod1_vi, 'feature_window_transform', 'y_des_window_24_hist_mode', title1)
plot_multi_model_single_feature_imp(mod1_vi, 'feature_transform', 'y_des_hist_mode', title1)
plot_multi_model_single_feature_imp(mod1_vi, 'feature', 'y_des', title1)
plot_multi_model_single_feature_imp(mod1_vi, 'transform', 'hist_mode', title1)
```







The `plot_multi_model_single_feature_imp` plots indicate that the `y_des_window_24_hist_mode` feature is less helpful for earlier forecast step models (forecast steps 10 and earlier) and most useful at lower lags (-40 and lower).

Feature Selection

There are more than 1,000 features across 8 feature sets. It is not possible to build all 48 forecast darts-lightGBM models with 1,000 features and also use lagged features. I stopped the model building process after 6 hours with the 305 catch22 features (and earlier selected lags). To make some progress prioritised features from each feature set are used to build all 48 lightGBM models with the earlier selected lags. The prioritised features are based on the F-statistic scores and some preliminary lightGBM models.

Highlights from each feature set include:

- cross-correlation
  - all features
- pairwise
  - y_des_dew.point_des
  - euclidean
  - linear
  - rbf
  - sigmoid
- rolling statistics

    - window 48
    - y_des
    - mean and std
    - min and max
    - skew and kurt
- bivariate
    - y_des and window 48
    - window 48
- catch22
    - CO_trev_1_num
    - DN_OutlierInclude
    - MD_hrv_classic_pnn40
    - SB_BinaryStats
- tsfeatures
    - intervals_mean
    - acf_features_x_acf10
    - pacf_features_x_pacf5
    - acf_features_x_acf1

Results for the best features from the tsfeatures set are shown below. Links are provided to specific commits for the other features.

```
# train = train_df
# valid = valid_df
# train = train_pair
# valid = valid_pair
# train = train_ccorr
# valid = valid_ccorr
# train = train_stats
# valid = valid_stats
# train = train_biv
# valid = valid_biv
# train = train_c22
# valid = valid_c22
train = train_tsf
valid = valid_tsf


# y_des MUST BE included in ex_cols at this stage
ex_cols2 = ['y', 'y_seasonal', 'y_res', 'y_orig', 'ds', # 'y_des',
            'dew.point',
            #'humidity', 'pressure',
            'wind.speed.mean',
            'humidity_des', 'pressure_des', 'humidity_des_diff_1', 'pressure_des_diff_1',
            'humidity_seasonal', 'dew.point_seasonal', 'pressure_seasonal',
            'humidity_res', 'dew.point_res', 'pressure_res',
            'wind.speed.mean_res', 'wind.speed.mean_seasonal',
            'wind.speed.mean_yearly', 'wind.speed.mean_daily',
            'humidity_daily', 'pressure_daily', 'dew.point_daily',
            'humidity_yearly', 'pressure_yearly', 'dew.point_yearly',
            'dew.point_yhat',  'pressure_yhat', 'humidity_yhat', 'y_yhat',
            'wind.speed.mean_daily.x', 'wind.speed.mean_daily.y',
            'wind.speed.mean_yearly.x', 'wind.speed.mean_yearly.y',
            'wind.speed.mean_yhat.x', 'wind.speed.mean_yhat.y',
            'wind.speed.mean_des.x', 'wind.speed.mean_des.y',
            'wind.speed.mean.x', 'wind.speed.mean.y',
            'sunshine',
            'wind.speed.max', 'wind.speed.max.sqrt',
            'wind.speed.mean.sqrt.x', 'wind.speed.mean.sqrt.y',
            'wind.speed.max.sqrt.x', 'wind.speed.max.sqrt.y',
            'wind.speed.mean_des',
            'declination', 'za', #'za_rad', 'irradiance',
            'y_daily', 'y_yearly',
            'wind.speed.mean.sqrt',
            'wind.bearing.mean',
            'wind.speed.mean_yhat', 'pressure.log',
            'wind.speed.mean.y_yearly', 'wind.speed.mean.x_yearly',
            'wind.speed.mean.y_daily', 'wind.speed.mean.x_daily',
            'dew.point_trend', 'y_trend', 'humidity_trend', 'pressure_trend',
            'wind.speed.mean_trend', 'declination', 'declination_diff_1',
            'day.sin.1', 'day.cos.1', 'year.sin.1', 'year.cos.1',
            'humidity_diff_1', 'ground_hf', 'irradiance_diff_1', 'y_des_diff_1',
            'pressure_diff_1', 'dew.point_des_diff_1',
            #'svp', 'rainfall', 'ah',
```

```
            'nao', 'mei', 'azimuth_diff_1', 'za_diff_1', 'za_rad_diff_1',
            'azimuth_cos_diff_1', 'azimuth_sin_diff_1',
            #'mixing_ratio', 'specific_humidity', 'vapor_pressure',
            'azimuth_sin',
            'azimuth_rad_diff_1', 'y_fft_x', 'y_fft_y', 'y_fft', 'dew.point_fft'
            ]
inc_cols2 = ['y_des', 'dew.point_des', 'pressure', 'humidity',
              't_pot',
              'svp', #'ground_hf',
              'air_density',
              'vp_def',
             #'y_yearly',
             #'y_yhat',
              'wind.speed.max.sqrt.x', 'wind.speed.max.sqrt.y',
             #'H2OC',
             #'mixing_ratio',
             #'specific_humidity',
             #'vapour_pressure',
             #'ah',
              'irradiance',
              'za_rad',
             #'azimuth_cos',
              'y_des_shadow',]

inc_cols2 = ['y_des', 'dew.point_des', 'humidity', 'pressure',
              'wind.speed.max.sqrt.x', 'wind.speed.max.sqrt.y', 'svp',
              'y_des_window_48_intervals_intervals_mean',
              'y_des_window_48_acf_features_x_acf10'
             ] #+ \
             #[i for i in train_tsf.columns.to_list() if 'y_des_window_48_intervals_intervals_mean' in i or 'y_des_window_48_a


# y_des and window 48 - 26
# [i for i in train_tsf.columns.to_list() if 'y_des_window_48_' in i]
# _window_48_intervals_intervals_mean - 2
# [i for i in train_tsf.columns.to_list() if '_window_48_intervals_intervals_mean' in i]
# _intervals_intervals_mean - 8
# [i for i in train_tsf.columns.to_list() if '_intervals_intervals_mean' in i]
# acf_features_x_acf10 - 3
# [i for i in train_tsf.columns.to_list() if '_window_48_acf_features_x_acf10' in i]
# pacf_features_x_pacf5 - 3
# [i for i in train_tsf.columns.to_list() if '_window_48_pacf_features_x_pacf5' in i]
# acf_features_x_acf1 - 6
# [i for i in train_tsf.columns.to_list() if '_window_48_acf_features_x_acf1' in i]
# _window_48_intervals_intervals_mean or _window_48_acf_features_x_acf10 - 5
# [i for i in train_tsf.columns.to_list() if '_window_48_intervals_intervals_mean' in i or '_window_48_acf_features_x_acf10'


data_params2 = {'y_col': Y_COL,
                'past_cov_cols': inc_cols2,
               #'past_cov_cols': train.columns.difference(ex_cols2),
               #'fut_cov_cols': None,
                'fut_cov_cols': FUT_COV,
               }

series, past_cov, fut_cov = get_darts_series(train, data_params2)
val_ser, val_past_cov, val_fut_cov = get_darts_series(valid, data_params2)

init_lags2 = [-1, -2, -4]
# late_lags2 = [-43, -44, -45]
mid_lags2  = [i for i in range(-6, -52, -4) if i > -50]
target_lags2 = pastcov_lags2 = init_lags2 + mid_lags2  # + late_lags2
# target_lags2  = [-1, -2, -4, -6, -12, -18, -24, -30, -36, -42, -48, -60, -72, -84, -96, -120]
# pastcov_lags2 = [-1, -2, -4, -6, -12, -18, -24, -30, -36, -42, -48, -60, -72, -84, -96, -120]
futcov_lags2  = (1, 48)


if data_params2['fut_cov_cols'] is not None:
  lag_params2 = {'lags': target_lags2,
                 'lags_past_covariates': pastcov_lags2,
                 'lags_future_covariates': futcov_lags2,
                }
else:
  lag_params2 = {'lags': target_lags2,
                 'lags_past_covariates': pastcov_lags2,
                }

mod_params2 = {'feature_fraction': 0.5,
               'output_chunk_length': 48,
               'multi_models': True,
               'linear_trees': True,
              #'bagging_fraction': 0.25,
```

```
              #'bagging_freq': 2,
              #'num_leaves':  16
              }
npr_params2 = {#'n_estimators': 200,
               'n_estimators': 50,
               'verbose': -1,
               #'early_stopping_round': 10,
              }  # Non-PRinting params
all_params2 = {**lag_params2,
               **mod_params2,
               **npr_params2,}

title2 = get_main_plot_title('lgbm - ', lag_params2, mod_params2)
model2 = LightGBMModel(**all_params2)


warnings.filterwarnings(
    action   = 'ignore',
    category = UserWarning,
    module   = r'.*sklearn'
)
warnings.filterwarnings(
    action   = 'ignore',
    category = FutureWarning,
    module   = r'.*sklearn'
)


if data_params2['fut_cov_cols'] is not None:
  model2.fit(series,
             past_covariates = past_cov,
             future_covariates = fut_cov,
             val_series = val_ser,
             val_past_covariates = val_past_cov,
             val_future_covariates = val_fut_cov,
            )
else:
  model2.fit(series,
             past_covariates = past_cov,
             val_series = val_ser,
             val_past_covariates = val_past_cov,
            )

if data_params2['fut_cov_cols'] is not None:
  backtest2 = model2.historical_forecasts(series = val_ser,
                                          past_covariates = val_past_cov,
                                          future_covariates = val_fut_cov,
                                          start   = 0.01,
                                          retrain = False,
                                          verbose = True,
                                          forecast_horizon = HORIZON,
                                          last_points_only = False
                                         )
else:
  backtest2 = model2.historical_forecasts(series = val_ser,
                                          past_covariates = val_past_cov,
                                          start   = 0.01,
                                          retrain = False,
                                          verbose = True,
                                          forecast_horizon = HORIZON,
                                          last_points_only = False
                                         )

hist_comp2 = get_historic_comparison(backtest2, valid)
summarise_historic_comparison(hist_comp2, valid)
plot_multistep_diagnostics(hist_comp2, title2)


# inc_cols1 = ['y_des', 'dew.point_des', 'pressure', 'humidity',
#              'wind.speed.max.sqrt.x', 'wind.speed.max.sqrt.y', 'svp',] + \
#          [i for i in train_pair.columns.values if 'y_des_dew.point_des' in i]
# FUT_COV  = ['irradiance', 'za_rad']
# init_lags1 = [-1, -2, -4]
# mid_lags1  = [i for i in range(-6, -52, -4) if i > -50]
# target_lags1 = pastcov_lags1 = init_lags1 + mid_lags1
# futcov_lags1  = (1, 48)
# 'feature_fraction': 0.5, 'multi_models': True, 'n_estimators': 200,
# Backtest RMSE all: 2.054629
# Backtest MAE all:  1.516239
# Backtest RMSE 48th: 2.607135
# Backtest MAE 48th:  2.010718
```

```
# inc_cols1 = ['y_des', 'dew.point_des', 'pressure', 'humidity',
#              'wind.speed.max.sqrt.x', 'wind.speed.max.sqrt.y', 'svp',] + \
#             [i for i in train_pair.columns.values if '_euclidean' in i]
# FUT_COV  = ['irradiance', 'za_rad']
# init_lags1 = [-1, -2, -4]
# mid_lags1  = [i for i in range(-6, -52, -4) if i > -50]
# target_lags1 = pastcov_lags1 = init_lags1 + mid_lags1
# futcov_lags1  = (1, 48)
# 'feature_fraction': 0.5, 'multi_models': True, 'n_estimators': 200,
# Backtest RMSE all: 2.066055
# Backtest MAE all:  1.522467
# Backtest RMSE 48th: 2.612072
# Backtest MAE 48th:  2.017451

# inc_cols1 = ['y_des', 'dew.point_des', 'pressure', 'humidity',
#              'wind.speed.max.sqrt.x', 'wind.speed.max.sqrt.y', 'svp',] + \
#             [i for i in train_pair.columns.values if '_linear' in i]
# FUT_COV  = ['irradiance', 'za_rad']
# init_lags1 = [-1, -2, -4]
# mid_lags1  = [i for i in range(-6, -52, -4) if i > -50]
# target_lags1 = pastcov_lags1 = init_lags1 + mid_lags1
# futcov_lags1  = (1, 48)
# 'feature_fraction': 0.5, 'multi_models': True, 'n_estimators': 200,
# Backtest RMSE all: 2.06173
# Backtest MAE all:  1.518413
# Backtest RMSE 48th: 2.621235
# Backtest MAE 48th:  2.014441

# inc_cols1 = ['y_des', 'dew.point_des', 'pressure', 'humidity',
#              'wind.speed.max.sqrt.x', 'wind.speed.max.sqrt.y', 'svp',] + \
#             [i for i in train_pair.columns.values if '_rbf' in i]
# FUT_COV  = ['irradiance', 'za_rad']
# init_lags1 = [-1, -2, -4]
# mid_lags1  = [i for i in range(-6, -52, -4) if i > -50]
# target_lags1 = pastcov_lags1 = init_lags1 + mid_lags1
# futcov_lags1  = (1, 48)
# 'feature_fraction': 0.5, 'multi_models': True, 'n_estimators': 200,
# Backtest RMSE all: 2.064636
# Backtest MAE all:  1.522389
# Backtest RMSE 48th: 2.602822
# Backtest MAE 48th:  2.013418

# inc_cols1 = ['y_des', 'dew.point_des', 'pressure', 'humidity',
#              'wind.speed.max.sqrt.x', 'wind.speed.max.sqrt.y', 'svp',] + \
#             [i for i in train_pair.columns.values if '_sigmoid' in i]
# FUT_COV  = ['irradiance', 'za_rad']
# init_lags1 = [-1, -2, -4]
# mid_lags1  = [i for i in range(-6, -52, -4) if i > -50]
# target_lags1 = pastcov_lags1 = init_lags1 + mid_lags1
# futcov_lags1  = (1, 48)
# 'feature_fraction': 0.5, 'multi_models': True, 'n_estimators': 200,
# Backtest RMSE all: 2.061121
# Backtest MAE all:  1.518418
# Backtest RMSE 48th: 2.608599
# Backtest MAE 48th:  2.005442

# inc_cols1 = ['y_des', 'dew.point_des', 'pressure', 'humidity',
#              'wind.speed.max.sqrt.x', 'wind.speed.max.sqrt.y', 'svp',] + \
#             [i for i in train_ccorr.columns.values if '_window_' in i]
# FUT_COV  = ['irradiance', 'za_rad']
# init_lags1 = [-1, -2, -4]
# mid_lags1  = [i for i in range(-6, -52, -4) if i > -50]
# target_lags1 = pastcov_lags1 = init_lags1 + mid_lags1
# futcov_lags1  = (1, 48)
# 'feature_fraction': 0.5, 'multi_models': True, 'n_estimators': 200,
# Backtest RMSE all: 2.149262
# Backtest MAE all:  1.592483
# Backtest RMSE 48th: 2.68731
# Backtest MAE 48th:  2.075344

# rolling stats data with all y_des_window_48 features
# inc_cols1 = ['y_des', 'dew.point_des', 'pressure', 'humidity',
#              'wind.speed.max.sqrt.x', 'wind.speed.max.sqrt.y', 'svp',] + \
# [i for i in train_stats.columns.to_list() if i.startswith('y_des_window_48_')]
# FUT_COV  = ['irradiance', 'za_rad']
# init_lags1 = [-1, -2, -4]
# mid_lags1  = [i for i in range(-6, -52, -4) if i > -50]
# target_lags1 = pastcov_lags1 = init_lags1 + mid_lags1
# futcov_lags1  = (1, 48)
# 'feature_fraction': 0.5, 'multi_models': True, 'n_estimators': 200,
# Backtest RMSE all: 2.093057
# Backtest MAE all:  1.543771
```

```
# Backtest RMSE 48th: 2.659828
# Backtest MAE 48th:  2.063213

# rolling stats data with mean and std _window_48 features
# inc_cols1 = ['y_des', 'dew.point_des', 'pressure', 'humidity',
#              'wind.speed.max.sqrt.x', 'wind.speed.max.sqrt.y', 'svp',] + \
# [i for i in train_stats.columns.to_list() if (i.endswith('_window_48_mean') or
#                                               i.endswith('_window_48_std')) and
#                                               not i.startswith('y_window') and
#                                               'dew.point_window_' not in i]
# FUT_COV  = ['irradiance', 'za_rad']
# init_lags1 = [-1, -2, -4]
# mid_lags1  = [i for i in range(-6, -52, -4) if i > -50]
# target_lags1 = pastcov_lags1 = init_lags1 + mid_lags1
# futcov_lags1  = (1, 48)
# 'feature_fraction': 0.5, 'multi_models': True, 'n_estimators': 200,
# Backtest RMSE all: 2.06731
# Backtest MAE all:  1.529796
# Backtest RMSE 48th: 2.632536
# Backtest MAE 48th:  2.030491

# rolling stats data with mean and std features
# inc_cols1 = ['y_des', 'dew.point_des', 'pressure', 'humidity',
#              'wind.speed.max.sqrt.x', 'wind.speed.max.sqrt.y', 'svp',] + \
# [i for i in train_stats.columns.to_list() if (i.endswith('_mean') or
#                                               i.endswith('_std')) and
#                                               not i.startswith('y_window') and
#                                               'dew.point_window_' not in i]
# FUT_COV  = ['irradiance', 'za_rad']
# init_lags1 = [-1, -2, -4]
# mid_lags1  = [i for i in range(-6, -52, -4) if i > -50]
# target_lags1 = pastcov_lags1 = init_lags1 + mid_lags1
# futcov_lags1  = (1, 48)
# 'feature_fraction': 0.5, 'multi_models': True, 'n_estimators': 200,
# Backtest RMSE all: 2.097057
# Backtest MAE all:  1.549243
# Backtest RMSE 48th: 2.638807
# Backtest MAE 48th:  2.028391

# rolling stats data with min and max _window_48_ features
# inc_cols1 = ['y_des', 'dew.point_des', 'pressure', 'humidity',
#              'wind.speed.max.sqrt.x', 'wind.speed.max.sqrt.y', 'svp',] + \
# [i for i in train_stats.columns.to_list() if (i.endswith('_window_48_min') or
#                                               i.endswith('_window_48_max')) and
#                                              not i.startswith('y_window') and
#                                              'dew.point_window_' not in i]
# FUT_COV  = ['irradiance', 'za_rad']
# init_lags1 = [-1, -2, -4]
# mid_lags1  = [i for i in range(-6, -52, -4) if i > -50]
# target_lags1 = pastcov_lags1 = init_lags1 + mid_lags1
# futcov_lags1  = (1, 48)
# 'feature_fraction': 0.5, 'multi_models': True, 'n_estimators': 200,
# Backtest RMSE all: 2.109358
# Backtest MAE all:  1.560926
# Backtest RMSE 48th: 2.664458
# Backtest MAE 48th:  2.067135

# rolling stats data with skew and kurt _window_48_ features
# inc_cols1 = ['y_des', 'dew.point_des', 'pressure', 'humidity',
#              'wind.speed.max.sqrt.x', 'wind.speed.max.sqrt.y', 'svp',] + \
# [i for i in train_stats.columns.to_list() if (i.endswith('_window_48_skew') or
#                                               i.endswith('_window_48_kurt')) and
#                                              not i.startswith('y_window') and
#                                              'dew.point_window_' not in i]
# FUT_COV  = ['irradiance', 'za_rad']
# init_lags1 = [-1, -2, -4]
# mid_lags1  = [i for i in range(-6, -52, -4) if i > -50]
# target_lags1 = pastcov_lags1 = init_lags1 + mid_lags1
# futcov_lags1  = (1, 48)
# 'feature_fraction': 0.5, 'multi_models': True, 'n_estimators': 200,
# Backtest RMSE all: 2.073491
# Backtest MAE all:  1.529933
# Backtest RMSE 48th: 2.603304
# Backtest MAE 48th:  2.008714

# rolling stats data all _window_48_ features
# inc_cols1 = ['y_des', 'dew.point_des', 'pressure', 'humidity',
#              'wind.speed.max.sqrt.x', 'wind.speed.max.sqrt.y', 'svp',] + \
# [i for i in train_stats.columns.to_list() if '_window_48_' in i and
#                                              not i.startswith('y_window') and
#                                              'dew.point_window_' not in i]
# FUT_COV  = ['irradiance', 'za_rad']
```

```
# init_lags1 = [-1, -2, -4]
# mid_lags1  = [i for i in range(-6, -52, -4) if i > -50]
# target_lags1 = pastcov_lags1 = init_lags1 + mid_lags1
# futcov_lags1  = (1, 48)
# 'feature_fraction': 0.5, 'multi_models': True, 'n_estimators': 200,
# Backtest RMSE all: 2.100728
# Backtest MAE all:  1.560154
# Backtest RMSE 48th: 2.644715
# Backtest MAE 48th:  2.060707

# bivariate data all y_des and _window_48_ features
# inc_cols1 = ['y_des', 'dew.point_des', 'pressure', 'humidity',
#              'wind.speed.max.sqrt.x', 'wind.speed.max.sqrt.y', 'svp',] + \
# [i for i in train_biv.columns.to_list() if 'y_des_' in i and '_window_48_' in i]
# FUT_COV  = ['irradiance', 'za_rad']
# init_lags1 = [-1, -2, -4]
# mid_lags1  = [i for i in range(-6, -52, -4) if i > -50]
# target_lags1 = pastcov_lags1 = init_lags1 + mid_lags1
# futcov_lags1  = (1, 48)
# 'feature_fraction': 0.5, 'multi_models': True, 'n_estimators': 200,
# Backtest RMSE all: 2.082737
# Backtest MAE all:  1.545648
# Backtest RMSE 48th: 2.65684
# Backtest MAE 48th:  2.055289

# bivariate data all _window_48_ features
# inc_cols1 = ['y_des', 'dew.point_des', 'pressure', 'humidity',
#              'wind.speed.max.sqrt.x', 'wind.speed.max.sqrt.y', 'svp',] + \
# [i for i in train_biv.columns.to_list() if '_window_48_' in i]
# FUT_COV  = ['irradiance', 'za_rad']
# init_lags1 = [-1, -2, -4]
# mid_lags1  = [i for i in range(-6, -52, -4) if i > -50]
# target_lags1 = pastcov_lags1 = init_lags1 + mid_lags1
# futcov_lags1  = (1, 48)
# 'feature_fraction': 0.5, 'multi_models': True, 'n_estimators': 200,
# Backtest RMSE all: 2.100234
# Backtest MAE all:  1.557241
# Backtest RMSE 48th: 2.686324
# Backtest MAE 48th:  2.075239

# catch22 data all y_des_window_48_ features
# inc_cols1 = ['y_des', 'dew.point_des', 'pressure', 'humidity',
#              'wind.speed.max.sqrt.x', 'wind.speed.max.sqrt.y', 'svp',] + \
# [i for i in train_c22.columns.to_list() if 'y_des_' in i and '_window_48_' in i]
# FUT_COV  = ['irradiance', 'za_rad']
# init_lags1 = [-1, -2, -4]
# mid_lags1  = [i for i in range(-6, -52, -4) if i > -50]
# target_lags1 = pastcov_lags1 = init_lags1 + mid_lags1
# futcov_lags1  = (1, 48)
# 'feature_fraction': 0.5, 'multi_models': True, 'n_estimators': 200,
# Backtest RMSE all: 2.075468
# Backtest MAE all:  1.538511
# Backtest RMSE 48th: 2.607097
# Backtest MAE 48th:  2.020683

# catch22 data all _window_48_CO_trev_1_num features
# inc_cols1 = ['y_des', 'dew.point_des', 'pressure', 'humidity',
#              'wind.speed.max.sqrt.x', 'wind.speed.max.sqrt.y', 'svp',] + \
# [i for i in train_c22.columns.to_list() if '_window_48_CO_trev_1_num' in i]
# FUT_COV  = ['irradiance', 'za_rad']
# init_lags1 = [-1, -2, -4]
# mid_lags1  = [i for i in range(-6, -52, -4) if i > -50]
# target_lags1 = pastcov_lags1 = init_lags1 + mid_lags1
# futcov_lags1  = (1, 48)
# 'feature_fraction': 0.5, 'multi_models': True, 'n_estimators': 200,
# Backtest RMSE all: 2.090498
# Backtest MAE all:  1.540294
# Backtest RMSE 48th: 2.645187
# Backtest MAE 48th:  2.035425

# catch22 data all _window_24_CO_trev_1_num features
# inc_cols1 = ['y_des', 'dew.point_des', 'pressure', 'humidity',
#              'wind.speed.max.sqrt.x', 'wind.speed.max.sqrt.y', 'svp',] + \
# [i for i in train_c22.columns.to_list() if '_window_24_CO_trev_1_num' in i]
# FUT_COV  = ['irradiance', 'za_rad']
# init_lags1 = [-1, -2, -4]
# mid_lags1  = [i for i in range(-6, -52, -4) if i > -50]
# target_lags1 = pastcov_lags1 = init_lags1 + mid_lags1
# futcov_lags1  = (1, 48)
# 'feature_fraction': 0.5, 'multi_models': True, 'n_estimators': 200,
# Backtest RMSE all: 2.067709
# Backtest MAE all:  1.524606
```

```
# Backtest RMSE 48th: 2.625651
# Backtest MAE 48th:  2.02091

# catch22 data all _window_96_CO_trev_1_num features
# inc_cols1 = ['y_des', 'dew.point_des', 'pressure', 'humidity',
#              'wind.speed.max.sqrt.x', 'wind.speed.max.sqrt.y', 'svp',] + \
# [i for i in train_c22.columns.to_list() if '_window_96_CO_trev_1_num' in i]
# FUT_COV  = ['irradiance', 'za_rad']
# init_lags1 = [-1, -2, -4]
# mid_lags1  = [i for i in range(-6, -52, -4) if i > -50]
# target_lags1 = pastcov_lags1 = init_lags1 + mid_lags1
# futcov_lags1  = (1, 48)
# 'feature_fraction': 0.5, 'multi_models': True, 'n_estimators': 200,
# Backtest RMSE all: 2.097159
# Backtest MAE all:  1.552527
# Backtest RMSE 48th: 2.647037
# Backtest MAE 48th:  2.057304

# catch22 data all y_des_window_ and _CO_trev_1_num features
# inc_cols1 = ['y_des', 'dew.point_des', 'pressure', 'humidity',
#              'wind.speed.max.sqrt.x', 'wind.speed.max.sqrt.y', 'svp',] + \
# [i for i in train_c22.columns.to_list() if 'y_des_window_' in i and '_CO_trev_1_num' in i]
# FUT_COV  = ['irradiance', 'za_rad']
# init_lags1 = [-1, -2, -4]
# mid_lags1  = [i for i in range(-6, -52, -4) if i > -50]
# target_lags1 = pastcov_lags1 = init_lags1 + mid_lags1
# futcov_lags1  = (1, 48)
# 'feature_fraction': 0.5, 'multi_models': True, 'n_estimators': 200,
# Backtest RMSE all: 2.07258
# Backtest MAE all:  1.530422
# Backtest RMSE 48th: 2.637695
# Backtest MAE 48th:  2.03489

# catch22 data and y_des_window_96_CO_trev_1_num feature
# inc_cols1 = ['y_des', 'dew.point_des', 'pressure', 'humidity',
#              'wind.speed.max.sqrt.x', 'wind.speed.max.sqrt.y', 'svp',
#              'y_des_window_96_CO_trev_1_num']
# FUT_COV  = ['irradiance', 'za_rad']
# init_lags1 = [-1, -2, -4]
# mid_lags1  = [i for i in range(-6, -52, -4) if i > -50]
# target_lags1 = pastcov_lags1 = init_lags1 + mid_lags1
# futcov_lags1  = (1, 48)
# 'feature_fraction': 0.5, 'multi_models': True, 'n_estimators': 200,
# Backtest RMSE all: 2.075651
# Backtest MAE all:  1.531054
# Backtest RMSE 48th: 2.626728
# Backtest MAE 48th:  2.037548

# catch22 data and _window_48_DN_OutlierInclude_ features
# inc_cols1 = ['y_des', 'dew.point_des', 'pressure', 'humidity',
#              'wind.speed.max.sqrt.x', 'wind.speed.max.sqrt.y', 'svp'] + \
# [i for i in train_c22.columns.to_list() if '_window_48_DN_OutlierInclude_' in i]
# FUT_COV  = ['irradiance', 'za_rad']
# init_lags1 = [-1, -2, -4]
# mid_lags1  = [i for i in range(-6, -52, -4) if i > -50]
# target_lags1 = pastcov_lags1 = init_lags1 + mid_lags1
# futcov_lags1  = (1, 48)
# 'feature_fraction': 0.5, 'multi_models': True, 'n_estimators': 200,
# Backtest RMSE all: 2.067599
# Backtest MAE all:  1.526987
# Backtest RMSE 48th: 2.570544
# Backtest MAE 48th:  1.979192

# catch22 data and _window_96_DN_OutlierInclude_ features
# inc_cols1 = ['y_des', 'dew.point_des', 'pressure', 'humidity',
#              'wind.speed.max.sqrt.x', 'wind.speed.max.sqrt.y', 'svp'] + \
# [i for i in train_c22.columns.to_list() if '_window_96_DN_OutlierInclude_' in i]
# FUT_COV  = ['irradiance', 'za_rad']
# init_lags1 = [-1, -2, -4]
# mid_lags1  = [i for i in range(-6, -52, -4) if i > -50]
# target_lags1 = pastcov_lags1 = init_lags1 + mid_lags1
# futcov_lags1  = (1, 48)
# 'feature_fraction': 0.5, 'multi_models': True, 'n_estimators': 200,
# Backtest RMSE all: 2.087411
# Backtest MAE all:  1.536907
# Backtest RMSE 48th: 2.625025
# Backtest MAE 48th:  2.015757

# catch22 data and _window_24_DN_OutlierInclude_ features
# inc_cols1 = ['y_des', 'dew.point_des', 'pressure', 'humidity',
#              'wind.speed.max.sqrt.x', 'wind.speed.max.sqrt.y', 'svp'] + \
# [i for i in train_c22.columns.to_list() if '_window_24_DN_OutlierInclude_' in i]
```

```
# FUT_COV  = ['irradiance', 'za_rad']
# init_lags1 = [-1, -2, -4]
# mid_lags1  = [i for i in range(-6, -52, -4) if i > -50]
# target_lags1 = pastcov_lags1 = init_lags1 + mid_lags1
# futcov_lags1  = (1, 48)
# 'feature_fraction': 0.5, 'multi_models': True, 'n_estimators': 200,
# Backtest RMSE all: 2.056417
# Backtest MAE all:  1.516452
# Backtest RMSE 48th: 2.607113
# Backtest MAE 48th:  2.003751

# catch22 data and y_des_window_24_DN_OutlierInclude_ features
# inc_cols1 = ['y_des', 'dew.point_des', 'pressure', 'humidity',
#              'wind.speed.max.sqrt.x', 'wind.speed.max.sqrt.y', 'svp'] + \
# [i for i in train_c22.columns.to_list() if 'y_des_window_24_DN_OutlierInclude_' in i]
# FUT_COV  = ['irradiance', 'za_rad']
# init_lags1 = [-1, -2, -4]
# mid_lags1  = [i for i in range(-6, -52, -4) if i > -50]
# target_lags1 = pastcov_lags1 = init_lags1 + mid_lags1
# futcov_lags1  = (1, 48)
# 'feature_fraction': 0.5, 'multi_models': True, 'n_estimators': 200,
# Backtest RMSE all: 2.055091
# Backtest MAE all:  1.515725
# Backtest RMSE 48th: 2.602113
# Backtest MAE 48th:  2.003303

# catch22 data and _window_48_MD_hrv_classic_pnn40 features
# inc_cols1 = ['y_des', 'dew.point_des', 'pressure', 'humidity',
#              'wind.speed.max.sqrt.x', 'wind.speed.max.sqrt.y', 'svp'] + \
# [i for i in train_c22.columns.to_list() if '_window_48_MD_hrv_classic_pnn40' in i]
# FUT_COV  = ['irradiance', 'za_rad']
# init_lags1 = [-1, -2, -4]
# mid_lags1  = [i for i in range(-6, -52, -4) if i > -50]
# target_lags1 = pastcov_lags1 = init_lags1 + mid_lags1
# futcov_lags1  = (1, 48)
# 'feature_fraction': 0.5, 'multi_models': True, 'n_estimators': 200,
# Backtest RMSE all: 2.081552
# Backtest MAE all:  1.533475
# Backtest RMSE 48th: 2.642724
# Backtest MAE 48th:  2.039954

# catch22 data and _window_48_SB_BinaryStats_ features
# inc_cols1 = ['y_des', 'dew.point_des', 'pressure', 'humidity',
#              'wind.speed.max.sqrt.x', 'wind.speed.max.sqrt.y', 'svp'] + \
# [i for i in train_c22.columns.to_list() if '_window_48_SB_BinaryStats_' in i]
# FUT_COV  = ['irradiance', 'za_rad']
# init_lags1 = [-1, -2, -4]
# mid_lags1  = [i for i in range(-6, -52, -4) if i > -50]
# target_lags1 = pastcov_lags1 = init_lags1 + mid_lags1
# futcov_lags1  = (1, 48)
# 'feature_fraction': 0.5, 'multi_models': True, 'n_estimators': 200,
# Backtest RMSE all: 2.078989
# Backtest MAE all:  1.5376
# Backtest RMSE 48th: 2.613613
# Backtest MAE 48th:  2.028031

# tsfeatures data and y_des_window_48_ features
# inc_cols1 = ['y_des', 'dew.point_des', 'pressure', 'humidity',
#              'wind.speed.max.sqrt.x', 'wind.speed.max.sqrt.y', 'svp'] + \
# [i for i in train_tsf.columns.to_list() if 'y_des_window_48_' in i]
# FUT_COV  = ['irradiance', 'za_rad']
# init_lags1 = [-1, -2, -4]
# mid_lags1  = [i for i in range(-6, -52, -4) if i > -50]
# target_lags1 = pastcov_lags1 = init_lags1 + mid_lags1
# futcov_lags1  = (1, 48)
# 'feature_fraction': 0.5, 'multi_models': True, 'n_estimators': 200,
# Backtest RMSE all: 1.046695
# Backtest MAE all:  0.741364
# Backtest RMSE 48th: 1.561273
# Backtest MAE 48th:  1.176769
# Radically improved forecasts!
# Mainly due to intervals_mean, but also acf_features_x_acf10,
# pacf_features_x_pacf5 and acf_features_x_acf1 (in that order)
# Notable increase in mse and mae after step 42

# tsfeatures data and _window_48_intervals_intervals_mean features
# inc_cols1 = ['y_des', 'dew.point_des', 'pressure', 'humidity',
#              'wind.speed.max.sqrt.x', 'wind.speed.max.sqrt.y', 'svp'] + \
# [i for i in train_tsf.columns.to_list() if '_window_48_intervals_intervals_mean' in i]
# FUT_COV  = ['irradiance', 'za_rad']
# init_lags1 = [-1, -2, -4]
# mid_lags1  = [i for i in range(-6, -52, -4) if i > -50]
```

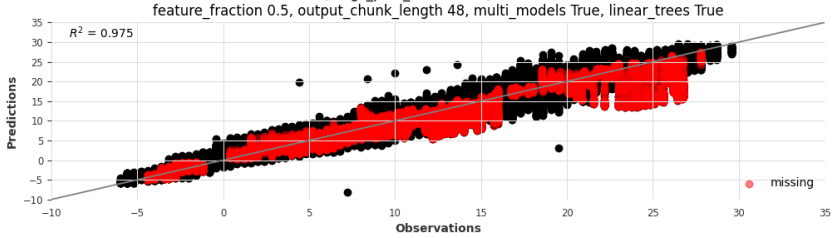
```
# target_lags1 = pastcov_lags1 = init_lags1 + mid_lags1
# futcov_lags1  = (1, 48)
# 'feature_fraction': 0.5, 'multi_models': True, 'n_estimators': 200,
# Backtest RMSE all: 1.184336
# Backtest MAE all:  0.8198
# Backtest RMSE 48th: 1.672435
# Backtest MAE 48th:  1.237791

# tsfeatures data and _intervals_intervals_mean features
# inc_cols1 = ['y_des', 'dew.point_des', 'pressure', 'humidity',
#              'wind.speed.max.sqrt.x', 'wind.speed.max.sqrt.y', 'svp'] + \
# [i for i in train_tsf.columns.to_list() if '_intervals_intervals_mean' in i]
# FUT_COV  = ['irradiance', 'za_rad']
# init_lags1 = [-1, -2, -4]
# mid_lags1  = [i for i in range(-6, -52, -4) if i > -50]
# target_lags1 = pastcov_lags1 = init_lags1 + mid_lags1
# futcov_lags1  = (1, 48)
# 'feature_fraction': 0.5, 'multi_models': True, 'n_estimators': 200,
# Backtest RMSE all: 1.101371
# Backtest MAE all:  0.720795
# Backtest RMSE 48th: 1.666715
# Backtest MAE 48th:  1.232753

# tsfeatures data and _window_48_acf_features_x_acf10 features
# inc_cols1 = ['y_des', 'dew.point_des', 'pressure', 'humidity',
#              'wind.speed.max.sqrt.x', 'wind.speed.max.sqrt.y', 'svp'] + \
# [i for i in train_tsf.columns.to_list() if '_window_48_acf_features_x_acf10' in i]
# FUT_COV  = ['irradiance', 'za_rad']
# init_lags1 = [-1, -2, -4]
# mid_lags1  = [i for i in range(-6, -52, -4) if i > -50]
# target_lags1 = pastcov_lags1 = init_lags1 + mid_lags1
# futcov_lags1  = (1, 48)
# 'feature_fraction': 0.5, 'multi_models': True, 'n_estimators': 200,
# Backtest RMSE all: 1.58474
# Backtest MAE all:  1.139813
# Backtest RMSE 48th: 2.246654
# Backtest MAE 48th:  1.721046
# Remarkable improvement

# tsfeatures data and _acf_features_x_acf10 features
# inc_cols1 = ['y_des', 'dew.point_des', 'pressure', 'humidity',
#              'wind.speed.max.sqrt.x', 'wind.speed.max.sqrt.y', 'svp'] + \
# [i for i in train_tsf.columns.to_list() if '_acf_features_x_acf10' in i]
# FUT_COV  = ['irradiance', 'za_rad']
# init_lags1 = [-1, -2, -4]
# mid_lags1  = [i for i in range(-6, -52, -4) if i > -50]
# target_lags1 = pastcov_lags1 = init_lags1 + mid_lags1
# futcov_lags1  = (1, 48)
# 'feature_fraction': 0.5, 'multi_models': True, 'n_estimators': 200,
# Backtest RMSE all: 1.559509
# Backtest MAE all:  1.108709
# Backtest RMSE 48th: 2.261477
# Backtest MAE 48th:  1.732537
# Longer windows are better

# tsfeatures data and _window_48_pacf_features_x_pacf5 features
# inc_cols1 = ['y_des', 'dew.point_des', 'pressure', 'humidity',
#              'wind.speed.max.sqrt.x', 'wind.speed.max.sqrt.y', 'svp'] + \
# [i for i in train_tsf.columns.to_list() if '_window_48_pacf_features_x_pacf5' in i]
# FUT_COV  = ['irradiance', 'za_rad']
# init_lags1 = [-1, -2, -4]
# mid_lags1  = [i for i in range(-6, -52, -4) if i > -50]
# target_lags1 = pastcov_lags1 = init_lags1 + mid_lags1
# futcov_lags1  = (1, 48)
# 'feature_fraction': 0.5, 'multi_models': True, 'n_estimators': 200,
# Backtest RMSE all: 1.725309
# Backtest MAE all:  1.251047
# Backtest RMSE 48th: 2.352581
# Backtest MAE 48th:  1.798736

# tsfeatures data and _window_48_acf_features_x_acf1 features
# inc_cols1 = ['y_des', 'dew.point_des', 'pressure', 'humidity',
#              'wind.speed.max.sqrt.x', 'wind.speed.max.sqrt.y', 'svp'] + \
# [i for i in train_tsf.columns.to_list() if '_window_48_acf_features_x_acf1' in i]
# FUT_COV  = ['irradiance', 'za_rad']
# init_lags1 = [-1, -2, -4]
# mid_lags1  = [i for i in range(-6, -52, -4) if i > -50]
# target_lags1 = pastcov_lags1 = init_lags1 + mid_lags1
# futcov_lags1  = (1, 48)
# 'feature_fraction': 0.5, 'multi_models': True, 'n_estimators': 200,
# Backtest RMSE all: 1.560277
# Backtest MAE all:  1.118079
```

```
# Backtest RMSE 48th: 2.252831
# Backtest MAE 48th:  1.706912
# _window_48_acf_features_x_acf10 and _window_48_acf_features_x_acf1
# together not so beneficial

# tsfeatures data and _window_48_intervals_intervals_mean or _window_48_acf_features_x_acf10 features
# inc_cols1 = ['y_des', 'dew.point_des', 'pressure', 'humidity',
#              'wind.speed.max.sqrt.x', 'wind.speed.max.sqrt.y', 'svp'] + \
# [i for i in train_tsf.columns.to_list() if '_window_48_intervals_intervals_mean' in i or '_window_48_acf_features_x_acf10'
# FUT_COV  = ['irradiance', 'za_rad']
# init_lags1 = [-1, -2, -4]
# mid_lags1  = [i for i in range(-6, -52, -4) if i > -50]
# target_lags1 = pastcov_lags1 = init_lags1 + mid_lags1
# futcov_lags1  = (1, 48)
# 'feature_fraction': 0.5, 'multi_models': True, 'n_estimators': 200,
# Backtest RMSE all: 1.049989
# Backtest MAE all:  0.739793
# Backtest RMSE 48th: 1.569379
# Backtest MAE 48th:  1.170854
# Very similar to tsfeatures data and y_des_window_48_ features
# but includes 5 extra features instead of 26 :-)

# tsfeatures data and y_des_window_48_intervals_intervals_mean or y_des_window_48_acf_features_x_acf10 features
# inc_cols1 = ['y_des', 'dew.point_des', 'pressure', 'humidity',
#              'wind.speed.max.sqrt.x', 'wind.speed.max.sqrt.y', 'svp',
#              'y_des_window_48_intervals_intervals_mean',
#              'y_des_window_48_acf_features_x_acf10']
# FUT_COV  = ['irradiance', 'za_rad']
# init_lags1 = [-1, -2, -4]
# mid_lags1  = [i for i in range(-6, -52, -4) if i > -50]
# target_lags1 = pastcov_lags1 = init_lags1 + mid_lags1
# futcov_lags1  = (1, 48)
# 'feature_fraction': 0.5, 'multi_models': True, 'n_estimators': 200,
# Backtest RMSE all: 1.039208
# Backtest MAE all:  0.734717
# Backtest RMSE 48th: 1.553754
# Backtest MAE 48th:  1.164894
# Very similar to tsfeatures data and y_des_window_48_ features
# but includes 2 extra features instead of 26 :-)

# tsfeatures data and _window_48_intervals_intervals_mean or
#                     _window_48_acf_features_x_acf10 features or
#                     _window_48_pacf_features_x_pacf5
# inc_cols1 = ['y_des', 'dew.point_des', 'pressure', 'humidity',
#              'wind.speed.max.sqrt.x', 'wind.speed.max.sqrt.y', 'svp'] + \
# [i for i in train_tsf.columns.to_list() if '_window_48_intervals_intervals_mean' in i
#                                         or '_window_48_acf_features_x_acf10' in i
#                                         or '_window_48_pacf_features_x_pacf5']
# FUT_COV  = ['irradiance', 'za_rad']
# init_lags1 = [-1, -2, -4]
# mid_lags1  = [i for i in range(-6, -52, -4) if i > -50]
# target_lags1 = pastcov_lags1 = init_lags1 + mid_lags1
# futcov_lags1  = (1, 48)
# 'feature_fraction': 0.5, 'multi_models': True, 'n_estimators': 200,
# Backtest RMSE all: 1.032584
# Backtest MAE all:  0.726959
# Backtest RMSE 48th: 1.567786
# Backtest MAE 48th:  1.171314

# tsfeatures data and y_des_window_48_intervals_intervals_mean or
#                     y_des_window_48_acf_features_x_acf10 features or
#                     y_des_window_48_pacf_features_x_pacf5
# inc_cols1 = ['y_des', 'dew.point_des', 'pressure', 'humidity',
#              'wind.speed.max.sqrt.x', 'wind.speed.max.sqrt.y', 'svp'] + \
# [i for i in train_tsf.columns.to_list() if 'y_des_window_48_intervals_intervals_mean' in i
#                                         or 'y_des_window_48_acf_features_x_acf10' in i
#                                         or 'y_des_window_48_pacf_features_x_pacf5']
# FUT_COV  = ['irradiance', 'za_rad']
# init_lags1 = [-1, -2, -4]
# mid_lags1  = [i for i in range(-6, -52, -4) if i > -50]
# target_lags1 = pastcov_lags1 = init_lags1 + mid_lags1
# futcov_lags1  = (1, 48)
# 'feature_fraction': 0.5, 'multi_models': True, 'n_estimators': 200,
# Backtest RMSE all: 1.019547
# Backtest MAE all:  0.720891
# Backtest RMSE 48th: 1.530872
# Backtest MAE 48th:  1.149493
# Probably not going to get much better than this from this feature set
# Small improvement by adding y_des_window_48_pacf_features_x_pacf5 to
# y_des_window_48_intervals_intervals_mean and
# y_des_window_48_acf_features_x_acf10 features
# Adding additional lags and/or window_96 features to reduce sudden mse/mae
```

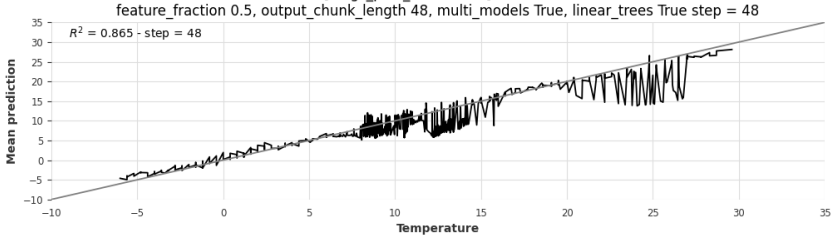
```
# increase around step 42 may be a better option than adding
# y_des_window_48_acf_features_x_acf10

# tsfeatures data and y_des_window_48_intervals_intervals_mean or y_des_window_48_acf_features_x_acf10 features
# inc_cols1 = ['y_des', 'dew.point_des', 'pressure', 'humidity',
#              'wind.speed.max.sqrt.x', 'wind.speed.max.sqrt.y', 'svp',
#              'y_des_window_48_intervals_intervals_mean',
#              'y_des_window_48_acf_features_x_acf10']
# FUT_COV  = ['irradiance', 'za_rad']
# init_lags1 = [-1, -2, -4]
# mid_lags1  = [i for i in range(-6, -52, -4) if i > -50]
# target_lags1 = pastcov_lags1 = init_lags1 + mid_lags1
# futcov_lags1  = (1, 48)
# 'feature_fraction': 0.5, 'multi_models': True,
# 'linear_trees': True, 'n_estimators': 50,
# Backtest RMSE all: 1.031075
# Backtest MAE all:  0.721472
# Backtest RMSE 48th: 1.487469
# Backtest MAE 48th:  1.09403
# Small but worthwhile improvement by switching to 'linear_trees': True
# More or less halves run time :-)
```

```
Backtest RMSE all: 1.031075
Backtest MAE all:  0.721472

# Backtest RMSE 48th: 1.487469
# Backtest MAE 48th:  1.09403

Backtest RMSE miss==0: 0.951777
Backtest MAE miss==0:  0.695555

Backtest RMSE miss==1: 1.991747
Backtest MAE miss==1:  1.20013


backtest['y_des']:
count    830304
mean     -0.85081
std      3.405905
min      -18.984048
25%      -3.302107
50%      -1.16953
75%      1.485439
max      16.111213

valid_df['y_des']:
count    17520.000000
mean        -0.855508
std          3.651553
min         -9.967023
25%         -3.448224
50%         -1.195697
75%          1.403691
max         13.030489
Name: y_des, dtype: float64
```

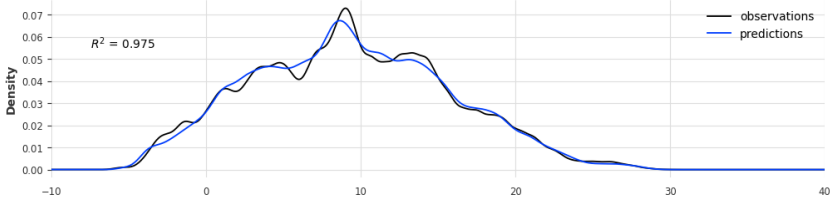
Multi-step lgbm - lags [-1, -2, -4, -6, -10, -14, -18, -22, -26, -30, -34, -38, -42, -46], lags_past_covariates [-1, -2, -4, -6, -10, -14, -18, -22, -26, -30, -34, -38, -42, -46], lags_future_covariates (1, 48) feature_fraction 0.5, output_chunk_length 48, multi_models True, linear_trees True



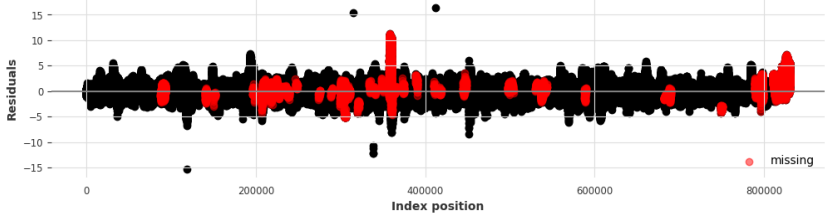
Multi-step lgbm - lags [-1, -2, -4, -6, -10, -14, -18, -22, -26, -30, -34, -38, -42, -46], lags_past_covariates [-1, -2, -4, -6, -10, -14, -18, -22, -26, -30, -34, -38, -42, -46], lags_future_covariates (1, 48) feature_fraction 0.5, output_chunk_length 48, multi_models True, linear_trees True step = 48



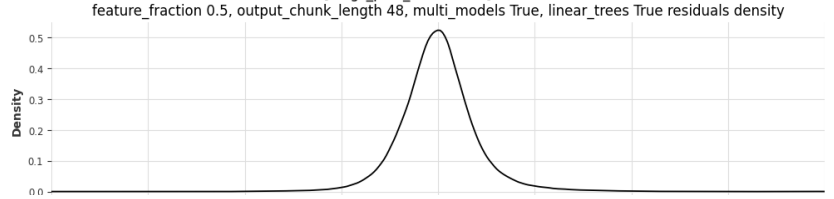
Multi-step lgbm - lags [-1, -2, -4, -6, -10, -14, -18, -22, -26, -30, -34, -38, -42, -46], lags_past_covariates [-1, -2, -4, -6, -10, -14, -18, -22, -26, -30, -34, -38, -42, -46], lags_future_covariates (1, 48) feature_fraction 0.5, output_chunk_length 48, multi_models True, linear_trees True

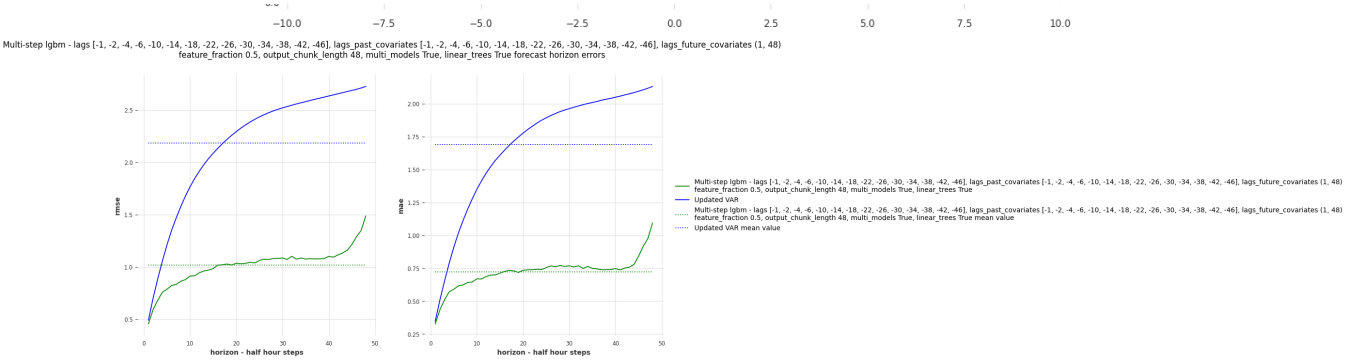


Multi-step lgbm - lags [-1, -2, -4, -6, -10, -14, -18, -22, -26, -30, -34, -38, -42, -46], lags_past_covariates [-1, -2, -4, -6, -10, -14, -18, -22, -26, -30, -34, -38, -42, -46], lags_future_covariates (1, 48) feature_fraction 0.5, output_chunk_length 48, multi_models True, linear_trees True residuals

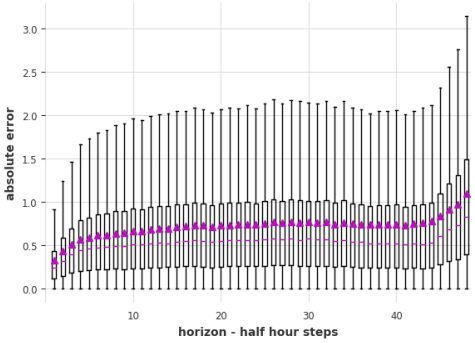


Multi-step lgbm - lags [-1, -2, -4, -6, -10, -14, -18, -22, -26, -30, -34, -38, -42, -46], lags_past_covariates [-1, -2, -4, -6, -10, -14, -18, -22, -26, -30, -34, -38, -42, -46], lags_future_covariates (1, 48) feature_fraction 0.5, output_chunk_length 48, multi_models True, linear_trees True residuals density

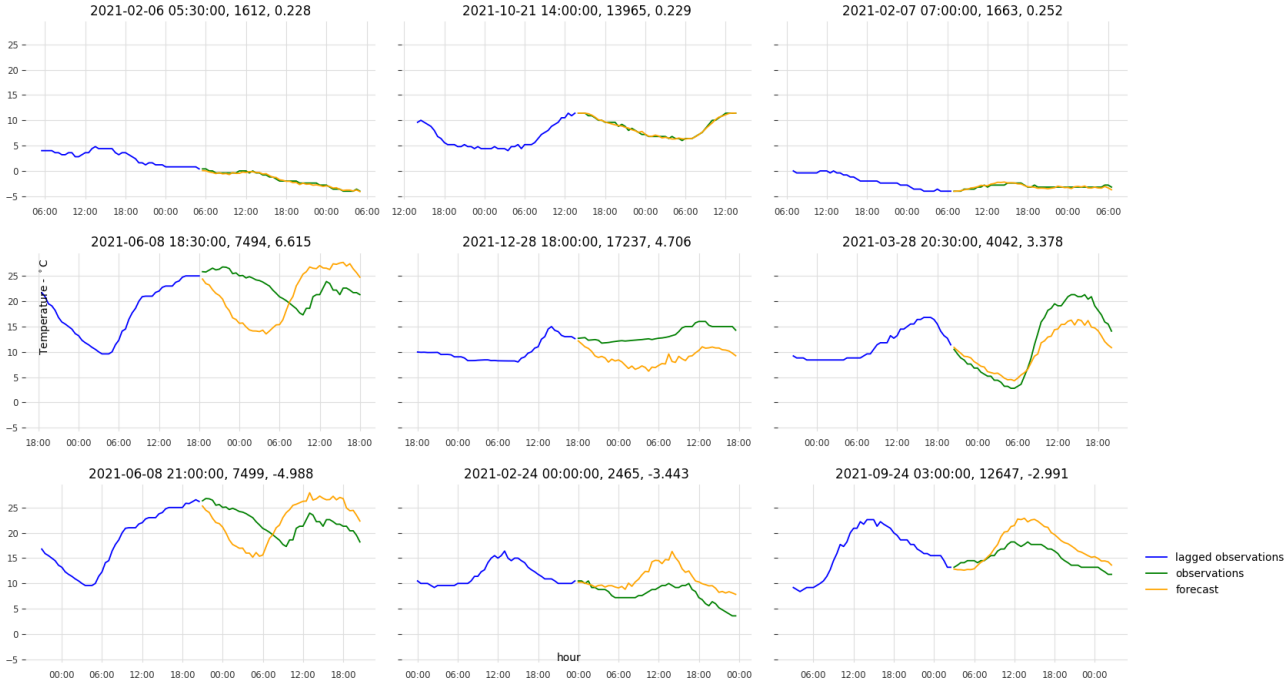




Multi-step lgbm - lags [-1, -2, -4, -6, -10, -14, -18, -22, -26, -30, -34, -38, -42, -46], lags_past_covariates [-1, -2, -4, -6, -10, -14, -18, -22, -26, -30, -34, -38, -42, -46], lags_future_covariates (1, 48)
feature_fraction 0.5, output_chunk_length 48, multi_models True, linear_trees True
boxplots with mean and median



Multi-step lgbm - lags [-1, -2, -4, -6, -10, -14, -18, -22, -26, -30, -34, -38, -42, -46], lags_past_covariates [-1, -2, -4, -6, -10, -14, -18, -22, -26, -30, -34, -38, -42, -46], lags_future_covariates (1, 48)
feature_fraction 0.5, output_chunk_length 48, multi_models True, linear_trees True forecast examples
init date, period idx, signed rmse



```
title2 = 'model2'  # over-ride long titles
mod2_vi = get_multi_model_feat_imps(model2)
summarise_multi_model_feat_imps(mod2_vi, title2, verbose=True)
```

	model	feature_transform_lag	imp	feature_window_transform	feature_transform	feature	transform	lag	type	shift
0	0	y_des_target_lag-46	1	y_des	y_des	y_des	des	-46	target	0
1	0	y_des_target_lag-42	8	y_des	y_des	y_des	des	-42	target	0
2	0	y_des_target_lag-38	7	y_des	y_des	y_des	des	-38	target	0
3	0	y_des_target_lag-34	2	y_des	y_des	y_des	des	-34	target	0
4	0	y_des_target_lag-30	5	y_des	y_des	y_des	des	-30	target	0
...	...	...	...	...	...	...	...	...	...	...
11419	47	za_rad_futcov_lag45	7	za_rad	za_rad	za_rad	None	45	futcov	0
11420	47	irradiance_futcov_lag46	1	irradiance	irradiance	irradiance	None	46	futcov	0
11421	47	za_rad_futcov_lag46	8	za_rad	za_rad	za_rad	None	46	futcov	0
11422	47	irradiance_futcov_lag47	0	irradiance	irradiance	irradiance	None	47	futcov	0
11423	47	za_rad_futcov_lag47	3	za_rad	za_rad	za_rad	None	47	futcov	0

11424 rows × 16 columns

	count	mean	std	min	25%	50%	75%	max	sum	num_0	pc_0	num_shad_geq	pc_shad_geq
model													
0	238	6.302521	20.407682	0	0	2	5	248	1500	75	31.512605	75	31.512605
1	238	6.302521	14.765997	0	0	2	6	174	1500	64	26.890756	64	26.890756
2	238	6.302521	12.133612	0	0	2	7	121	1500	62	26.050420	62	26.050420
3	238	6.302521	10.606546	0	1	2	7	99	1500	58	24.369748	58	24.369748
4	238	6.302521	10.071520	0	1	2	7	81	1500	53	22.268908	53	22.268908
5	238	6.302521	9.487771	0	1	2	8	71	1500	48	20.168067	48	20.168067
6	238	6.302521	9.514860	0	1	3	7	55	1500	46	19.327731	46	19.327731
7	238	6.302521	9.764358	0	1	2	7	66	1500	47	19.747899	47	19.747899
8	238	6.302521	9.693231	0	1	3	7	58	1500	47	19.747899	47	19.747899
9	238	6.302521	9.722786	0	1	3	7	67	1500	43	18.067227	43	18.067227
10	238	6.302521	9.728426	0	1	3	7	69	1500	47	19.747899	47	19.747899
11	238	6.302521	9.432912	0	1	3	7	62	1500	39	16.386555	39	16.386555
12	238	6.302521	9.717143	0	1	3	7	67	1500	41	17.226891	41	17.226891
13	238	6.302521	9.656597	0	1	3	7	61	1500	44	18.487395	44	18.487395
14	238	6.302521	10.109155	0	1	3	6	63	1500	35	14.705882	35	14.705882
15	238	6.302521	10.086591	0	1	2	7	61	1500	49	20.588235	49	20.588235
16	238	6.302521	10.001734	0	1	3	6	64	1500	35	14.705882	35	14.705882
17	238	6.302521	10.121252	0	1	2	6	75	1500	37	15.546218	37	15.546218
18	238	6.302521	10.421541	0	1	3	6	76	1500	49	20.588235	49	20.588235
19	238	6.302521	10.500195	0	1	2	6	75	1500	33	13.865546	33	13.865546
20	238	6.302521	10.447421	0	1	3	6	72	1500	42	17.647059	42	17.647059
21	238	6.302521	10.859327	0	1	2	6	84	1500	40	16.806723	40	16.806723
22	238	6.302521	10.779771	0	1	3	5	83	1500	38	15.966387	38	15.966387
23	238	6.302521	11.126852	0	1	2	6	93	1500	43	18.067227	43	18.067227
24	238	6.302521	11.121162	0	1	2	5	84	1500	40	16.806723	40	16.806723
25	238	6.302521	11.279760	0	1	3	6	85	1500	42	17.647059	42	17.647059
26	238	6.302521	11.479980	0	1	2	6	95	1500	45	18.907563	45	18.907563
27	238	6.302521	11.442798	0	1	2	6	93	1500	44	18.487395	44	18.487395
28	238	6.302521	11.148824	0	1	3	5	78	1500	42	17.647059	42	17.647059
29	238	6.302521	11.531322	0	1	3	6	72	1500	41	17.226891	41	17.226891
30	238	6.302521	11.582800	0	1	2	5	77	1500	36	15.126050	36	15.126050
31	238	6.302521	11.617719	0	1	2	6	75	1500	49	20.588235	49	20.588235
32	238	6.302521	11.634776	0	1	2	5	83	1500	40	16.806723	40	16.806723
33	238	6.302521	11.847523	0	1	2	5	84	1500	46	19.327731	46	19.327731
34	238	6.302521	12.520400	0	1	2	5	85	1500	48	20.168067	48	20.168067

35	238	6.302521	12.700081	0	1	2	5	90	1500	44	18.487395	44	18.487395
36	238	6.302521	13.310035	0	1	2	6	101	1500	52	21.848739	52	21.848739
37	238	6.302521	13.424931	0	1	2	5	118	1500	40	16.806723	40	16.806723
38	238	6.302521	14.006361	0	1	2	5	123	1500	38	15.966387	38	15.966387
39	238	6.302521	13.828091	0	1	2	5	129	1500	49	20.588235	49	20.588235
40	238	6.302521	13.623357	0	1	3	5	137	1500	41	17.226891	41	17.226891
41	238	6.302521	13.898702	0	1	3	6	138	1500	39	16.386555	39	16.386555
42	238	6.302521	14.101833	0	1	3	5	144	1500	43	18.067227	43	18.067227
43	238	6.302521	14.523983	0	1	3	6	157	1500	38	15.966387	38	15.966387
44	238	6.302521	13.973184	0	1	3	6	149	1500	43	18.067227	43	18.067227
45	238	6.302521	14.349204	0	1	3	6	159	1500	39	16.386555	39	16.386555
46	238	6.302521	13.872873	0	1	3	6	151	1500	37	15.546218	37	15.546218
47	238	6.302521	13.179420	0	1	3	6	152	1500	33	13.865546	33	13.865546

	count	mean	std	min	25%	50%	75%	max	sum	num_0	pc_0	num_shad_geq	pc_shad_geq
lag													
-46	480	3.412500	4.087336	0	1	2	5	30	1638	95	19.791667	95	19.791667
-42	480	2.181250	2.833306	0	0	1	3	25	1047	125	26.041667	125	26.041667
-38	480	3.708333	4.956842	0	1	2	4	29	1780	85	17.708333	85	17.708333
-34	480	4.595833	8.261336	0	1	2	4	57	2206	108	22.500000	108	22.500000
-30	480	7.108333	12.214879	0	1	3	7	66	3412	74	15.416667	74	15.416667
...	...	...	...	...	...	...	...	...	...	...	...	...	...
43	96	2.479167	2.375219	0	1	2	4	12	238	17	17.708333	17	17.708333
44	96	3.343750	3.882018	0	1	2	5	19	321	22	22.916667	22	22.916667
45	96	2.187500	2.442227	0	0	1	3	11	210	27	28.125000	27	28.125000
46	96	1.406250	1.780468	0	0	1	2	8	135	40	41.666667	40	41.666667
47	96	1.281250	1.303462	0	0	1	2	5	123	36	37.500000	36	37.500000

62 rows × 13 columns

	count	mean	std	min	25%	50%	75%	max	sum	num_0	pc_0	num_shad_geq	pc_shad_geq
type													
futcov	4704	3.012117	3.952038	0	0	2	4	51	14169	1191	25.318878	1191	25.318878
pastcov	6048	8.088955	14.347227	0	1	3	8	159	48922	879	14.533730	879	14.533730
target	672	13.257440	17.456288	0	2	8	19	248	8909	54	8.035714	54	8.035714

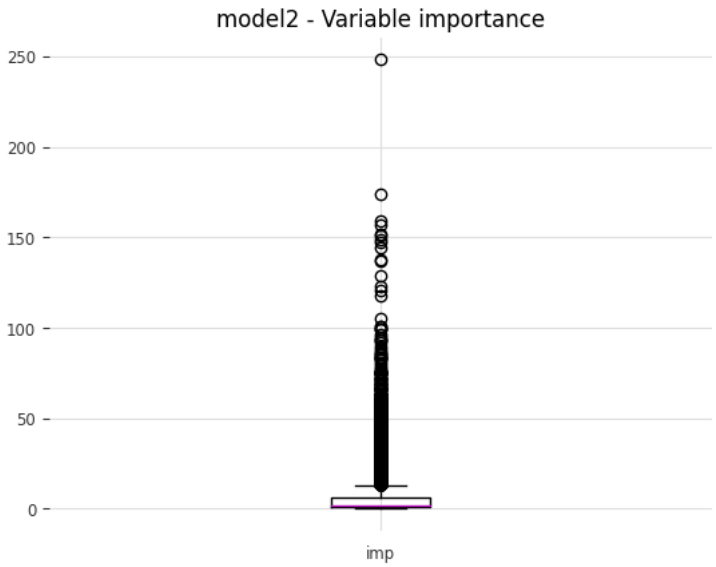
	count	mean	std	min	25%	50%	75%	max	sum	num_0	pc_0	num_shad_geq	pc_shad_geq
window													
0	10080	3.999802	6.883551	0	1	2	5	248	40318	2098	20.813492	2098	20.813492
48	1344	23.572917	22.723939	0	7	17	33	159	31682	26	1.934524	26	1.934524

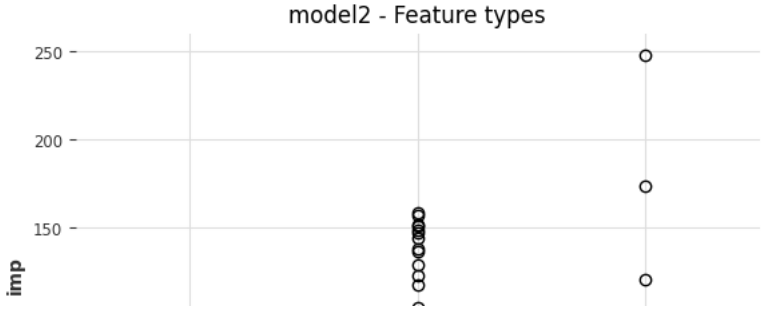
	count	mean	std	min	25%	50%	75%	max	sum	num_0	pc_0	num_shad_geq	pc_shad_geq
feature													
dew.point_des	672	2.065476	2.658709	0	1	1	3	18	1388	165	24.553571	165	24.553571
humidity	672	2.767857	3.195980	0	1	2	4	20	1860	147	21.875000	147	21.875000
irradiance	2352	1.538690	1.866729	0	0	1	2	15	3619	909	38.647959	909	38.647959
pressure	672	5.471726	5.414759	0	2	4	7	35	3677	60	8.928571	60	8.928571
svp	672	3.181548	3.291558	0	1	2	4	21	2138	96	14.285714	96	14.285714
wind.speed.max.sqrt.x	672	3.139881	3.544238	0	1	2	4	23	2110	119	17.708333	119	17.708333
wind.speed.max.sqrt.y	672	1.873512	2.079726	0	0	1	3	14	1259	177	26.339286	177	26.339286
y_des	2688	16.889509	20.255901	0	3	10	23	248	45399	169	6.287202	169	6.287202
za_rad	2352	4.485544	4.838942	0	1	3	6	51	10550	282	11.989796	282	11.989796

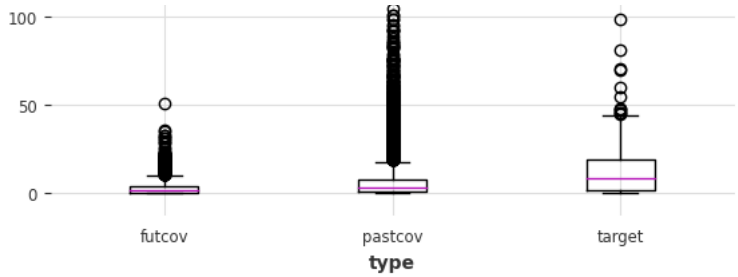
transform	count	mean	std	min	25%	50%	75%	max	sum	num_0	pc_0	num_shad_geq	pc_shad_geq
None	8064	3.126612	3.913421	0	1	2	4	51	25213	1790	22.197421	1790	22.197421
acf_features_x_acf10	672	20.382440	19.068264	0	5	15	28	95	13697	19	2.827381	19	2.827381
des	2016	7.492560	12.667731	0	1	3	9	248	15105	308	15.277778	308	15.277778
intervals_intervals_mean	672	26.763393	25.486017	0	8	20	37	159	17985	7	1.041667	7	1.041667

	model	feature_transform_lag	imp	feature_window_transform	feature_transform	f
9896	41	y_des_window_48_intervals_intervals_mean_pastc...	138	y_des_window_48_intervals_intervals_mean	y_des_intervals_intervals_mean	
10134	42	y_des_window_48_intervals_intervals_mean_pastc...	144	y_des_window_48_intervals_intervals_mean	y_des_intervals_intervals_mean	
131	0	y_des_pastcov_lag-1	147	y_des	y_des	
10610	44	y_des_window_48_intervals_intervals_mean_pastc...	149	y_des_window_48_intervals_intervals_mean	y_des_intervals_intervals_mean	
11086	46	y_des_window_48_intervals_intervals_mean_pastc...	151	y_des_window_48_intervals_intervals_mean	y_des_intervals_intervals_mean	
11324	47	y_des_window_48_intervals_intervals_mean_pastc...	152	y_des_window_48_intervals_intervals_mean	y_des_intervals_intervals_mean	
10372	43	y_des_window_48_intervals_intervals_mean_pastc...	157	y_des_window_48_intervals_intervals_mean	y_des_intervals_intervals_mean	
10848	45	y_des_window_48_intervals_intervals_mean_pastc...	159	y_des_window_48_intervals_intervals_mean	y_des_intervals_intervals_mean	
251	1	y_des_target_lag-1	174	y_des	y_des	
13	0	y_des_target_lag-1	248	y_des	y_des	

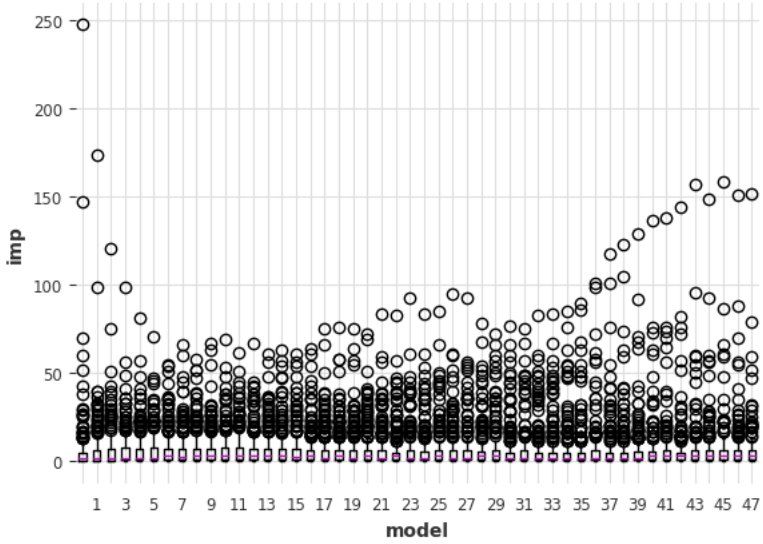
	model	feature_transform_lag	imp	feature_window_transform	feature_transform	feature	transform	l
3157	13	wind.speed.max.sqrt.x_pastcov_lag-26	0	wind.speed.max.sqrt.x	wind.speed.max.sqrt.x	wind.speed.max.sqrt.x	None	-
7784	32	irradiance_futcov_lag13	0	irradiance	irradiance	irradiance	None	
1898	7	irradiance_futcov_lag45	0	irradiance	irradiance	irradiance	None	
9910	41	irradiance_futcov_lag5	0	irradiance	irradiance	irradiance	None	
1900	7	irradiance_futcov_lag46	0	irradiance	irradiance	irradiance	None	
1902	7	irradiance_futcov_lag47	0	irradiance	irradiance	irradiance	None	
1904	8	y_des_target_lag-46	0	y_des	y_des	y_des	des	-
9906	41	irradiance_futcov_lag3	0	irradiance	irradiance	irradiance	None	
4491	18	za_rad_futcov_lag32	0	za_rad	za_rad	za_rad	None	
4490	18	irradiance_futcov_lag32	0	irradiance	irradiance	irradiance	None	



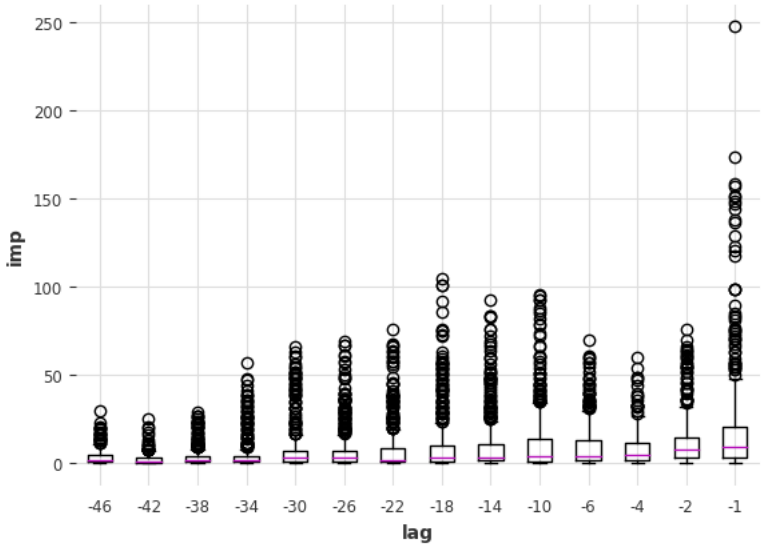




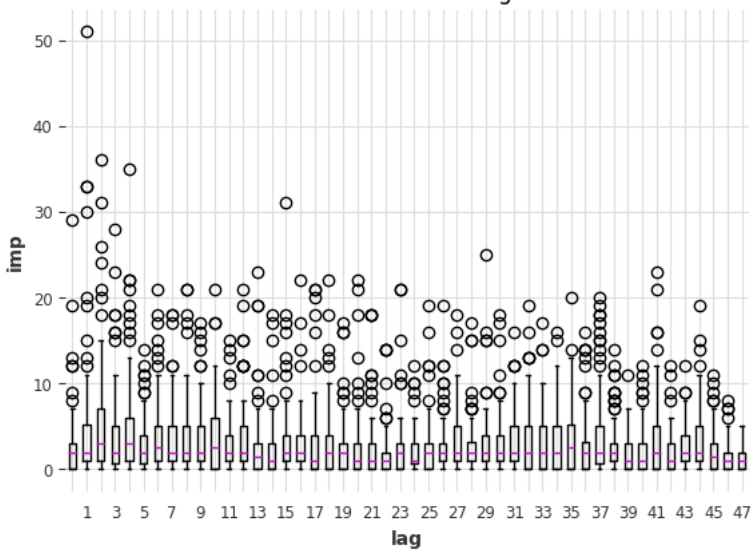
model2 - Models



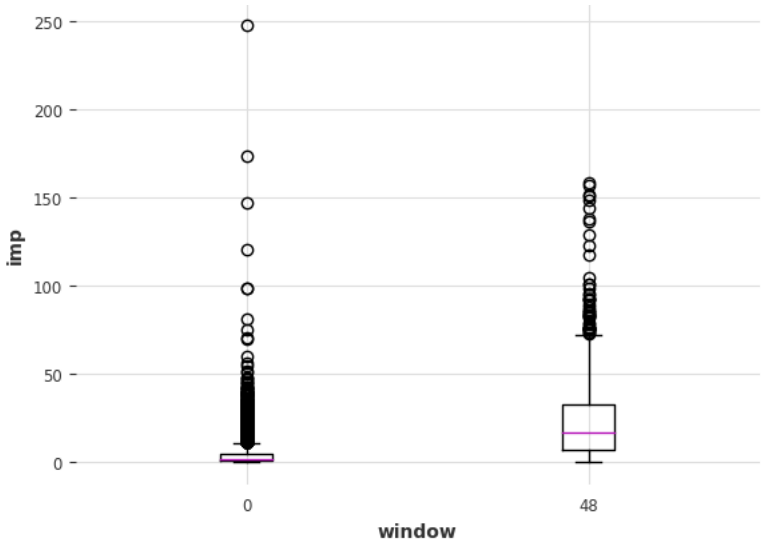
model2 - Past Lags



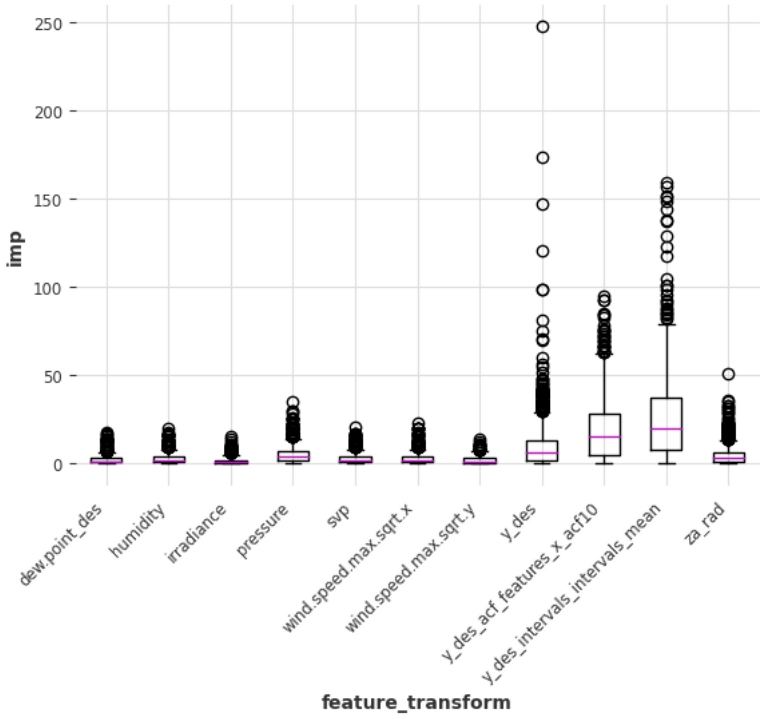
model2 - Future Lags



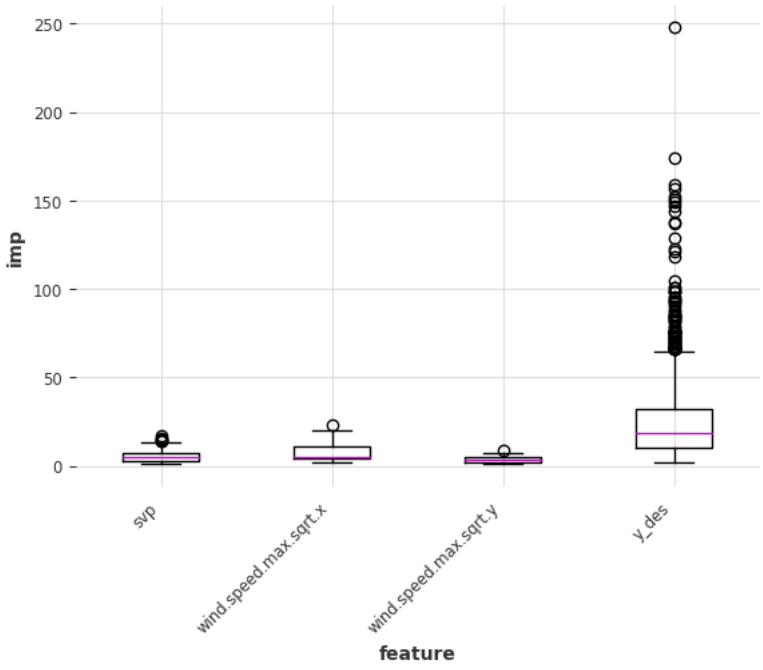
model2 - Windows



model2 - Feature Transform



model2 - Features (above core features)



model2 - Features (between core features)