Predict Bike Sharing Demand with AutoGluon Template

Project: Predict Bike Sharing Demand with AutoGluon

This notebook is a template with each step that you need to complete for the project.

Please fill in your code where there are explicit `?` markers in the notebook. You are welcome to add more cells and code as you see fit.

Once you have completed all the code implementations, please export your notebook as a HTML file so the reviews can view your code. Make sure you have all outputs correctly outputted.

`File–> Export Notebook As... –> Export Notebook as HTML`

There is a writeup to complete as well after all code implememtation is done. Please answer all questions and attach the necessary tables and charts. You can complete the writeup in either markdown or PDF.

Completing the code template and writeup template will cover all of the rubric points for this project.

The rubric contains "Stand Out Suggestions" for enhancing the project beyond the minimum requirements. The stand out suggestions are optional. If you decide to pursue the "stand out suggestions", you can include the code in this notebook and also discuss the results in the writeup file.

Step 1: Create an account with Kaggle

Create Kaggle Account and download API key

Below is example of steps to get the API username and key. Each student will have their own username and key.

1. Open account settings. kaggle1.png kaggle2.png
2. Scroll down to API and click Create New API Token. kaggle3.png kaggle4.png
3. Open up `kaggle.json` and use the username and key. kaggle5.png

Step 2: Download the Kaggle dataset using the kaggle python library

Open up Sagemaker Studio and use starter template

1. Notebook should be using a `ml.t3.medium` instance (2 vCPU + 4 GiB)
2. Notebook should be using kernal: `Python 3 (MXNet 1.8 Python 3.7 CPU Optimized)`

Install packages

```
!pip install -U pip
!pip install -U setuptools wheel
!pip install -U "mxnet<2.0.0" bokeh==2.0.1
!pip install autogluon #--no-cache-dir
# Without --no-cache-dir, smaller aws instances may have trouble installing
```

Show hidden output

Setup Kaggle API Key

```
# create the .kaggle directory and an empty kaggle.json file
!mkdir -p /root/.kaggle
!touch /root/.kaggle/kaggle.json
!chmod 600 /root/.kaggle/kaggle.json
```

```
# Fill in your user name and key from creating the kaggle account and API token file
import json
kaggle_username = "sabaka2"
kaggle_key = "85d8d604804339013d8ca565fee9ec96"

# Save API token the kaggle.json file
with open("/root/.kaggle/kaggle.json", "w") as f:
    f.write(json.dumps({"username": kaggle_username, "key": kaggle_key}))
```

Download and explore dataset

Go to the [bike sharing demand competition](#) and agree to the terms

kaggle6.png

```
# Download the dataset, it will be in a .zip file so you'll need to unzip it as well.
!kaggle competitions download -c bike-sharing-demand
# If you already downloaded it you can use the -o command to overwrite the file
!unzip -o bike-sharing-demand.zip
```

```
Downloading bike-sharing-demand.zip to /content
100% 189k/189k [00:00<00:00, 712kB/s]
100% 189k/189k [00:00<00:00, 711kB/s]
Archive:  bike-sharing-demand.zip
  inflating: sampleSubmission.csv
  inflating: test.csv
  inflating: train.csv
```

```
import pandas as pd
from autogluon.tabular import TabularPredictor
```

```
# Create the train dataset in pandas by reading the csv
# Set the parsing of the datetime column so you can use some of the `dt` features in pandas later
train =  pd.read_csv('train.csv', parse_dates=['datetime'])
train.head()
```

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count
0	2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0	3	13	16
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0	8	32	40
2	2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0.0	5	27	32
3	2011-01-01 03:00:00	1	0	0	1	9.84	14.395	75	0.0	3	10	13
4	2011-01-01 04:00:00	1	0	0	1	9.84	14.395	75	0.0	0	1	1

```
# Simple output of the train dataset to view some of the min/max/varition of the dataset features.
print(train.describe())
```

```
                           datetime        season       holiday  \
count                         10886  10886.000000  10886.000000
mean   2011-12-27 05:56:22.399411968      2.506614      0.028569
min              2011-01-01 00:00:00      1.000000      0.000000
25%              2011-07-02 07:15:00      2.000000      0.000000
50%              2012-01-01 20:30:00      3.000000      0.000000
75%              2012-07-01 12:45:00      4.000000      0.000000
max              2012-12-19 23:00:00      4.000000      1.000000
std                              NaN      1.116174      0.166599

         workingday       weather         temp         atemp      humidity  \
count  10886.000000  10886.000000  10886.00000  10886.000000  10886.000000
mean       0.680875      1.418427     20.23086     23.655084     61.886460
min        0.000000      1.000000      0.82000      0.760000      0.000000
25%        0.000000      1.000000     13.94000     16.665000     47.000000
50%        1.000000      1.000000     20.50000     24.240000     62.000000
```

```
75%        1.000000      2.000000     26.24000     31.060000     77.000000
max        1.000000      4.000000     41.00000     45.455000    100.000000
std        0.466159      0.633839      7.79159      8.474601     19.245033

          windspeed        casual    registered         count
count  10886.000000  10886.000000  10886.000000  10886.000000
mean      12.799395     36.021955    155.552177    191.574132
min        0.000000      0.000000      0.000000      1.000000
25%        7.001500      4.000000     36.000000     42.000000
50%       12.998000     17.000000    118.000000    145.000000
75%       16.997900     49.000000    222.000000    284.000000
max       56.996900    367.000000    886.000000    977.000000
std        8.164537     49.960477    151.039033    181.144454
```

```
# Create the test pandas dataframe in pandas by reading the csv, remember to parse the datetime!
test = pd.read_csv('test.csv', parse_dates=['datetime'])
test.head()
```

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed
0	2011-01-20 00:00:00	1	0	1	1	10.66	11.365	56	26.0027
1	2011-01-20 01:00:00	1	0	1	1	10.66	13.635	56	0.0000
2	2011-01-20 02:00:00	1	0	1	1	10.66	13.635	56	0.0000
3	2011-01-20 03:00:00	1	0	1	1	10.66	12.880	56	11.0014
4	2011-01-20 04:00:00	1	0	1	1	10.66	12.880	56	11.0014

```
# Same thing as train and test dataset
submission = pd.read_csv('sampleSubmission.csv', parse_dates=['datetime'])
submission.head()
```

	datetime	count
0	2011-01-20 00:00:00	0
1	2011-01-20 01:00:00	0
2	2011-01-20 02:00:00	0
3	2011-01-20 03:00:00	0
4	2011-01-20 04:00:00	0

Step 3: Train a model using AutoGluon's Tabular Prediction

Requirements:

- We are predicting `count`, so it is the label we are setting.
- Ignore `casual` and `registered` columns as they are also not present in the test dataset.
- Use the `root_mean_squared_error` as the metric to use for evaluation.
- Set a time limit of 10 minutes (600 seconds).
- Use the preset `best_quality` to focus on creating the best model.

```
train = train.drop(columns=['casual', 'registered'])
```

```
predictor = TabularPredictor(label="count", eval_metric='root_mean_squared_error').fit(train_data=train, time_limit=600, preset
```

Show hidden output

Review AutoGluon's training run with ranking of models that did the best.

```
performance = predictor.evaluate(train)
print(performance)
```

```
/usr/local/lib/python3.10/dist-packages/dask/dataframe/__init__.py:42: FutureWarning:
Dask dataframe query planning is disabled because dask-expr is not installed.
```

```
You can install it with `pip install dask[dataframe]` or `conda install dask`.
This will raise in a future version.

  warnings.warn(msg, FutureWarning)
{'root_mean_squared_error': -85.78142480758537, 'mean_squared_error': -7358.452842019422, 'mean_absolute_error': -55.3297
```

```
predictor.fit_summary()
```

```
    'max_base_models_per_type': 5,
    'save_bag_folds': True,
    'use_child_oof': True},
   'CatBoost_BAG_L1': {'use_orig_features': True,
    'max_base_models': 25,
    'max_base_models_per_type': 5,
    'save_bag_folds': True},
   'WeightedEnsemble_L2': {'use_orig_features': False,
    'max_base_models': 25,
    'max_base_models_per_type': 5,
    'save_bag_folds': True},
   'LightGBMXT_BAG_L2': {'use_orig_features': True,
    'max_base_models': 25,
    'max_base_models_per_type': 5,
    'save_bag_folds': True},
   'LightGBM_BAG_L2': {'use_orig_features': True,
    'max_base_models': 25,
    'max_base_models_per_type': 5,
    'save_bag_folds': True},
   'WeightedEnsemble_L3': {'use_orig_features': False,
    'max_base_models': 25,
    'max_base_models_per_type': 5,
    'save_bag_folds': True}},
  'leaderboard':                      model   score_val              eval_metric  pred_time_val  \
  0     WeightedEnsemble_L3  -55.123669  root_mean_squared_error      46.554677
  1         LightGBM_BAG_L2  -55.159472  root_mean_squared_error      33.113358
  2       LightGBMXT_BAG_L2  -60.897195  root_mean_squared_error      46.106592
  3   KNeighborsDist_BAG_L1  -84.125061  root_mean_squared_error       0.101436
  4     WeightedEnsemble_L2  -84.125061  root_mean_squared_error       0.102985
  5   KNeighborsUnif_BAG_L1 -101.546199  root_mean_squared_error       0.112823
  6  RandomForestMSE_BAG_L1 -116.548359  root_mean_squared_error       0.838605
  7         LightGBM_BAG_L1 -131.054162  root_mean_squared_error       2.338241
  8       LightGBMXT_BAG_L1 -131.460909  root_mean_squared_error      29.046163
  9         CatBoost_BAG_L1 -132.353281  root_mean_squared_error       0.230176

       fit_time  pred_time_val_marginal  fit_time_marginal  stack_level  \
  0  374.166624                0.002170           0.058821            3
  1  286.815213                0.445915          39.235108            2
  2  334.872695               13.439148          87.292590            2
  3    0.105579                0.101436           0.105579            1
  4    0.148080                0.001550           0.042501            2
  5    0.107367                0.112823           0.107367            1
  6   21.579986                0.838605          21.579986            1
  7   48.483054                2.338241          48.483054            1
  8  112.638537               29.046163         112.638537            1
  9   64.665583                0.230176          64.665583            1

     can_infer  fit_order
  0       True         10
  1       True          9
  2       True          8
  3       True          2
  4       True          7
  5       True          1
  6       True          5
  7       True          4
  8       True          3
  9       True          6  }
```

Start coding or generate with AI.

Create predictions from test dataset

```
predictions = predictor.predict(test)
predictions.head()
```

	count
0	36.705181
1	43.758060
2	47.846897
3	52.561142
4	53.246292

dtype: float32

NOTE: Kaggle will reject the submission if we don't set everything to be > 0.

```
# Describe the `predictions` series to see if there are any negative values
predictions_description = predictions.describe()
print(predictions_description)
```

```
count    6493.000000
mean       99.240425
std        89.459610
min        -3.421120
25%        16.177315
50%        63.820770
75%       171.560120
max       367.068390
Name: count, dtype: float64
```

```
num_negative_values = (predictions < 0).sum()
print(f"Number of negative values: {num_negative_values}")
```

```
Number of negative values: 2
```

```
predictions[predictions < 0] = 0
```

Set predictions to submission dataframe, save, and submit

```
submission["count"] = predictions
print(submission["count"].head())
submission.to_csv("submission.csv", index=False)
```

```
0    36.705181
1    43.758060
2    47.846897
3    52.561142
4    53.246292
Name: count, dtype: float32
```

```
!kaggle competitions submit -c bike-sharing-demand -f submission.csv -m "original data submission1"
```

```
100% 188k/188k [00:00<00:00, 309kB/s]
Successfully submitted to Bike Sharing Demand
```

View submission via the command line or in the web browser under the competition's page - `My Submissions`

```
!kaggle competitions submissions -c bike-sharing-demand | tail -n +1 | head -n 10
```

```
fileName                    date                description                          status    publicScore  privateScore
--------------------------  ------------------  -----------------------------------  --------  -----------  ------------
submission.csv              2024-09-19 14:29:24 original data submission1            complete  1.84061      1.84061
submission_new_hpo.csv      2024-09-18 23:05:44 new features with hyperparameters-2  complete  0.48480      0.48480
submission_new_hpo.csv      2024-09-18 22:46:12 new features with hyperparameters    complete  0.59355      0.59355
submission_new_hpo.csv      2024-09-18 22:06:45 new features with hyperparameters    complete  0.49050      0.49050
submission_new_hpo.csv      2024-09-18 21:36:14 new features with hyperparameters    complete  0.49050      0.49050
submission_new_features.csv 2024-09-18 15:41:15 new features                         complete  0.67692      0.67692
submission_new_features.csv 2024-09-18 14:57:27 new features                         complete  1.67733      1.67733
submission_new_features.csv 2024-09-18 02:19:01 new features                         complete  1.67743      1.67743
```

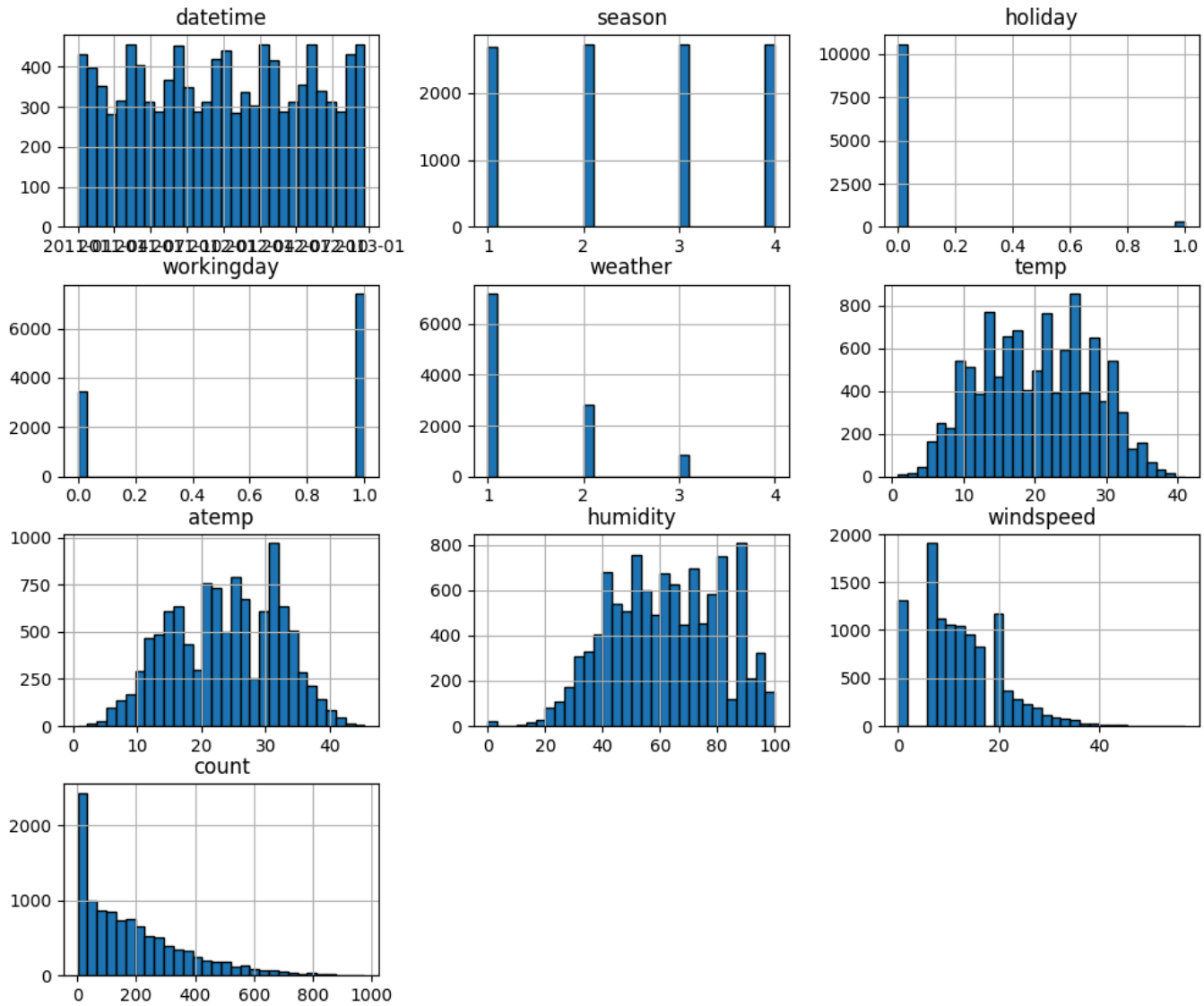
Initial score of ?

Step 4: Exploratory Data Analysis and Creating an additional feature

- Any additional feature will do, but a great suggestion would be to separate out the datetime into hour, day, or month parts.

```
# Create a histogram of all features to show the distribution of each one relative to the data. This is part of the exploritory
train.hist(figsize=(12, 10), bins=30, edgecolor='black')
```

```
array([[<Axes: title={'center': 'datetime'}>,
        <Axes: title={'center': 'season'}>,
        <Axes: title={'center': 'holiday'}>],
       [<Axes: title={'center': 'workingday'}>,
        <Axes: title={'center': 'weather'}>,
        <Axes: title={'center': 'temp'}>],
       [<Axes: title={'center': 'atemp'}>,
        <Axes: title={'center': 'humidity'}>,
        <Axes: title={'center': 'windspeed'}>],
       [<Axes: title={'center': 'count'}>, <Axes: >, <Axes: >]],
      dtype=object)
```




```
# create a new feature
train['hour'] = train['datetime'].dt.hour
test['hour'] = test['datetime'].dt.hour
```

Make category types for these so models know they are not just numbers

- AutoGluon originally sees these as ints, but in reality they are int representations of a category.
- Setting the dtype to category will classify these as categories in AutoGluon.

```
def get_season(date):
    month = date.month
    if month in [12, 1, 2]:
        return 'Winter'
    elif month in [3, 4, 5]:
        return 'Spring'
    elif month in [6, 7, 8]:
        return 'Summer'
    elif month in [9, 10, 11]:
        return 'Fall'
    else:
        return 'Unknown'

def get_weather(atemp):
    if atemp < 10:
        return 'Cold'
    elif 10 <= atemp < 20:
        return 'Cool'
    elif 20 <= atemp < 30:
        return 'Warm'
    elif atemp >= 30:
        return 'Hot'
    else:
        return 'Unknown'  # In case of invalid input
```

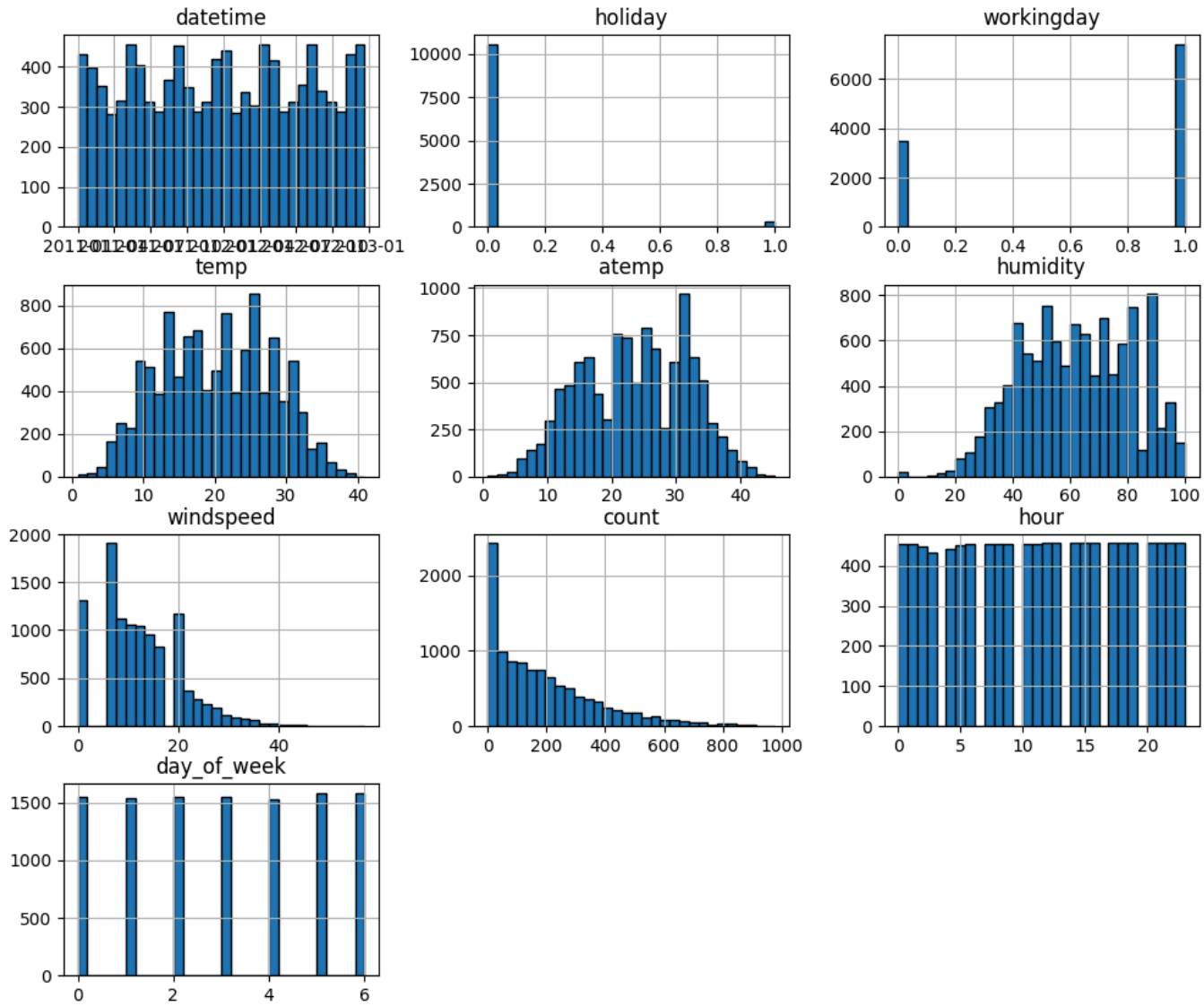
```
train['weather'] = train['atemp'].apply(get_weather)
train['weather'] = train['weather'].astype('category')
test['weather'] = test['atemp'].apply(get_weather)
test['weather'] = test['weather'].astype('category')
train['season'] = train['datetime'].apply(get_season)
train["season"] = train["season"].astype('category')
test["season"] = test['datetime'].apply(get_season)
test["season"] = test["season"].astype('category')
train['day_of_week'] = train['datetime'].dt.dayofweek
test['day_of_week'] = test['datetime'].dt.dayofweek
```

```
# View are new feature
train.head()
```

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	count	hour	day_of_week
0	2011-01-01 00:00:00	Winter	0	0	Cool	9.84	14.395	81	0.0	16	0	5
1	2011-01-01 01:00:00	Winter	0	0	Cool	9.02	13.635	80	0.0	40	1	5
2	2011-01-01 02:00:00	Winter	0	0	Cool	9.02	13.635	80	0.0	32	2	5
3	2011-01-01 03:00:00	Winter	0	0	Cool	9.84	14.395	75	0.0	13	3	5
4	2011-01-01 04:00:00	Winter	0	0	Cool	9.84	14.395	75	0.0	1	4	5

```
# View histogram of all features again now with the hour feature
train.hist(figsize=(12, 10), bins=30, edgecolor='black')
```

```
array([[<Axes: title={'center': 'datetime'}>,
        <Axes: title={'center': 'holiday'}>,
        <Axes: title={'center': 'workingday'}>],
       [<Axes: title={'center': 'temp'}>,
        <Axes: title={'center': 'atemp'}>,
        <Axes: title={'center': 'humidity'}>],
       [<Axes: title={'center': 'windspeed'}>,
        <Axes: title={'center': 'count'}>,
        <Axes: title={'center': 'hour'}>],
       [<Axes: title={'center': 'day_of_week'}>, <Axes: >, <Axes: >]],
      dtype=object)
```



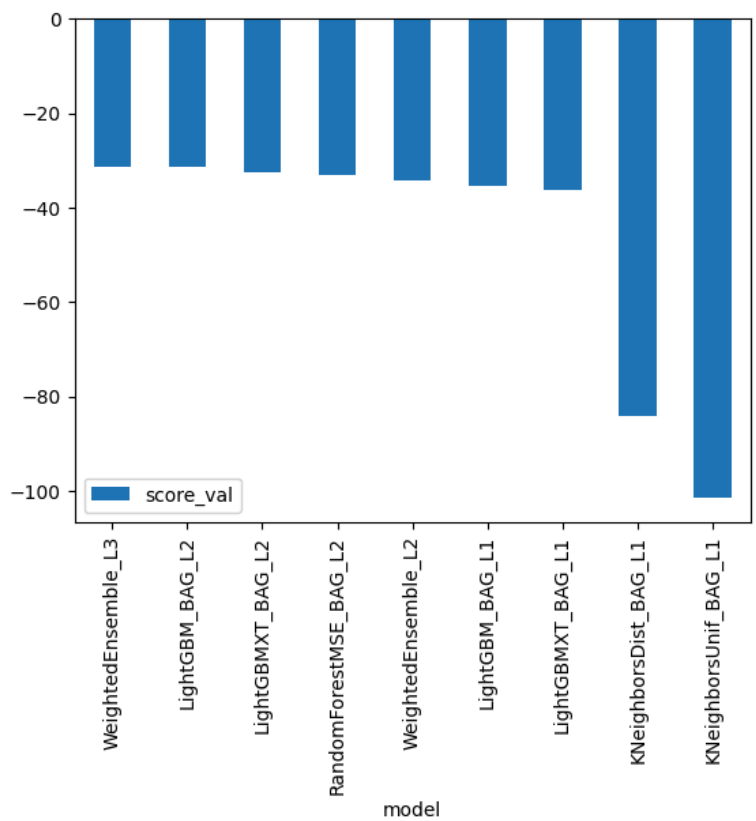
Step 5: Rerun the model with the same settings as before, just with more features

```
predictor_new_features2 = TabularPredictor(label="count", eval_metric='root_mean_squared_error').fit(train_data=train, time_lir
```

Show hidden output

```
performance_new_features2 = predictor_new_features2.evaluate(train)
print(performance_new_features2)

predictor_new_features2.fit_summary()
predictor_new_features2.leaderboard(silent=True).plot(kind="bar", x="model", y="score_val")
```

```
{'root_mean_squared_error': -19.476397051538957, 'mean_squared_error': -379.3300421091954, 'mean_absolute_error': -13.186
*** Summary of fit() ***
Estimated performance of each model:
                    model   score_val              eval_metric  pred_time_val    fit_time  pred_time_val_marginal  fit_ti
0     WeightedEnsemble_L3  -31.366154  root_mean_squared_error      67.959713  373.306901                0.004591
1         LightGBM_BAG_L2  -31.462139  root_mean_squared_error      64.999748  288.377863                0.811282
2       LightGBMXT_BAG_L2  -32.561460  root_mean_squared_error      66.244080  294.026102                2.055614
3  RandomForestMSE_BAG_L2  -33.166791  root_mean_squared_error      65.088226  291.144146                0.899760
4     WeightedEnsemble_L2  -34.239538  root_mean_squared_error      64.112820  250.108942                0.001055
5         LightGBM_BAG_L1  -35.383666  root_mean_squared_error       7.904369   81.320753                7.904369
6       LightGBMXT_BAG_L1  -36.142399  root_mean_squared_error      56.119587  168.710230               56.119587
7   KNeighborsDist_BAG_L1  -84.125061  root_mean_squared_error       0.087809    0.056213                0.087809
8   KNeighborsUnif_BAG_L1 -101.546199  root_mean_squared_error       0.076701    0.055146                0.076701
Number of models trained: 9
Types of models trained:
{'WeightedEnsembleModel', 'StackerEnsembleModel_LGB', 'StackerEnsembleModel_KNN', 'StackerEnsembleModel_RF'}
Bagging used: True  (with 8 folds)
Multi-layer stack-ensembling used: True  (with 3 levels)
Feature Metadata (Processed):
(raw dtype, special dtypes):
('category', [])             : 2 | ['season', 'weather']
('float', [])                : 3 | ['temp', 'atemp', 'windspeed']
('int', [])                  : 3 | ['humidity', 'hour', 'day_of_week']
('int', ['bool'])            : 2 | ['holiday', 'workingday']
('int', ['datetime_as_int']) : 4 | ['datetime', 'datetime.year', 'datetime.month', 'datetime.day']
*** End of fit() summary ***
/usr/local/lib/python3.10/dist-packages/autogluon/core/utils/plots.py:169: UserWarning: AutoGluon summary plots cannot be
  warnings.warn('AutoGluon summary plots cannot be created because bokeh is not installed. To see plots, please do: "pip
<Axes: xlabel='model'>
```




```
# Remember to set all negative values to zero
predictions_new2 = predictor_new_features2.predict(test)
predictions_new2.head()
predictions_description_new2 = predictions_new2.describe()
print(predictions_description_new2)

num_negative_values_new2 = (predictions_new2 < 0).sum()
print(f"Number of negative values: {num_negative_values_new2}")
```

```
count    6493.000000
mean      147.626312
std       130.852676
min         2.053551
```

```
25%        48.401764
50%       115.637779
75%       202.952057
max       797.545349
Name: count, dtype: float64
Number of negative values: 0
```

```
predictions_new2[predictions_new2 < 0] = 0
```

```
# Same submitting predictions
submission["count"] = predictions_new2
print(submission["count"].head())
submission.to_csv("submission_new_features2.csv", index=False)
```

```
0    19.830393
1    15.760131
2    14.801182
3    11.409325
4     9.552790
Name: count, dtype: float32
```

```
!kaggle competitions submit -c bike-sharing-demand -f submission_new_features2.csv -m "new features submission2"
```

```
100% 188k/188k [00:01<00:00, 169kB/s]
Successfully submitted to Bike Sharing Demand
```

```
!kaggle competitions submissions -c bike-sharing-demand | tail -n +1 | head -n 12
```

```
fileName                      date                description                           status    publicScore  privateSco
----------------------------  ------------------  ----------------------------------    --------  -----------  ----------
submission_new_features2.csv  2024-09-19 19:56:20  new features submission2              complete  0.75401      0.75401
submission_new_hpo5.csv       2024-09-19 18:01:48  new features with hpo5                complete  0.61440      0.61440
submission_new_hpo4.csv       2024-09-19 16:35:40  new features with hpo4                complete  0.51728      0.51728
submission_new_hpo3.csv       2024-09-19 16:06:03  new features with hpo3                complete  0.56763      0.56763
submission_new_hpo2.csv       2024-09-19 15:54:03  new features with hpo2                complete  0.56763      0.56763
submission_new_hpo1.csv       2024-09-19 15:21:01  new features with hpo1                complete  0.58200      0.58200
submission_new_features.csv   2024-09-19 14:57:42  new features submission1              complete  0.68507      0.68507
submission.csv                2024-09-19 14:29:24  original data submission1             complete  1.84061      1.84061
submission_new_hpo.csv        2024-09-18 23:05:44  new features with hyperparameters-2   complete  0.48480      0.48480
submission_new_hpo.csv        2024-09-18 22:46:12  new features with hyperparameters     complete  0.59355      0.59355
```

New Score of ?

Step 6: Hyper parameter optimization

- There are many options for hyper parameter optimization.
- Options are to change the AutoGluon higher level parameters or the individual model hyperparameters.
- The hyperparameters of the models themselves that are in AutoGluon. Those need the `hyperparameter` and `hyperparameter_tune_kwargs` arguments.

```
# hyperparameter_tune_kwargs = {
#     'num_trials': 20,              # Try 20 different configurations
#     'scheduler': 'local',          # Run on local machine
#     'searcher': 'random',          # Random search for hyperparameters
#     'max_t': 600                   # Max time for each trial is 600 seconds
# }
```

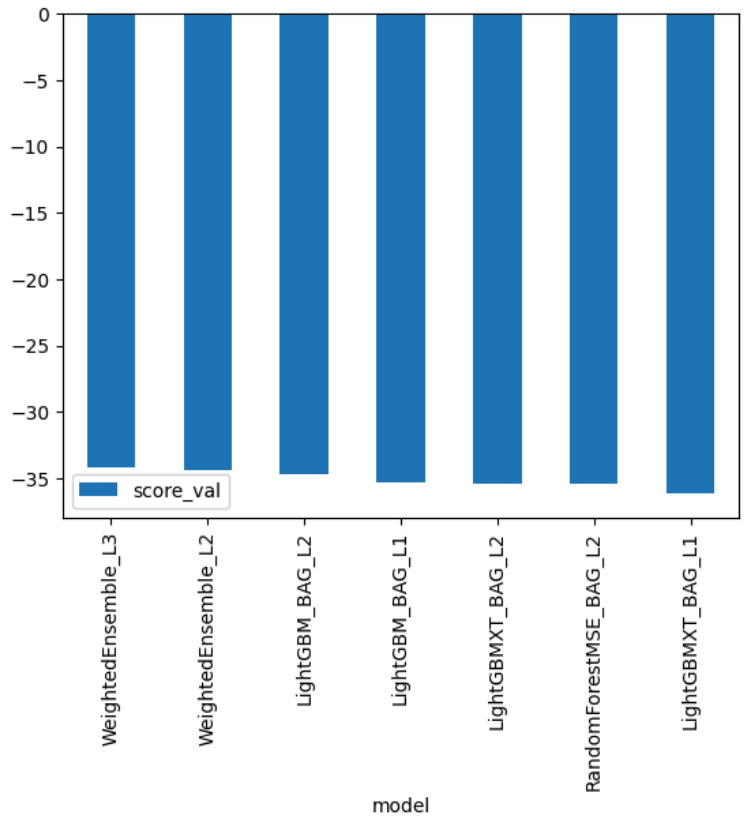
```
# Exclude poor-performing models and define hyperparameters for tuning
hyperparameters = {
    'GBM': [
        {
        'learning_rate': 0.05,   # Smaller learning rate for better generalization
        'num_leaves': 31,        # Maximum number of leaves per tree
        'feature_fraction': 0.8, # Random subset of features for each iteration
        'bagging_fraction': 0.8, # Random subset of data for each iteration
        'bagging_freq': 5,       # Bagging performed every 5 iterations
        'max_depth': 10,         # Depth of the trees (you can set to -1 for no limit)
        'num_boost_round': 1000, # Number of boosting rounds
```

```
        'early_stopping_rounds': 50  # Early stopping if performance doesn't improve
        }
    ],
    'RF': [
        {
            'n_estimators': 100,           # Number of trees in the forest
            'max_depth': 10,               # Maximum depth of each tree
            'min_samples_split': 5,        # Minimum number of samples required to split an internal node
            'min_samples_leaf': 2,         # Minimum number of samples required to be at a leaf node
            'max_features': 'sqrt',        # Number of features to consider when looking for the best split
            'bootstrap': True              # Whether bootstrap samples are used when building trees
        }
    ],
    'CAT': [
        {
            'learning_rate': 0.05,       # Smaller learning rate for more gradual training
            'depth': 6,                  # Depth of the trees
            'l2_leaf_reg': 3.0,          # Regularization to avoid overfitting
            'iterations': 1000,          # Number of boosting iterations
            'one_hot_max_size': 10,      # Maximum size of categorical features for one-hot encoding
            'eval_metric': 'RMSE',       # Root Mean Squared Error (for regression tasks)
            'od_type': 'Iter',           # Early stopping based on the number of iterations
            'od_wait': 100               # Wait for improvement in 100 iterations before stopping
        }
    ],
}


excluded_model_types = ['KNN']
time_limit = 600

# 1. only exclude models            hpo1
# 2. exclude models +  hyperparams  hpo2
# 3. increase time to 1000 sec      hpo3

predictor_new_hpo1 = TabularPredictor(
    label='count',
    eval_metric='root_mean_squared_error'
).fit(
    train_data=train,
    time_limit=time_limit,
    presets='best_quality',
    excluded_model_types=excluded_model_types,  # Exclude KNN
)
```

Show hidden output

```
performance_new_hpo1 = predictor_new_hpo1.evaluate(train)
print(performance_new_hpo1)
predictor_new_hpo1.fit_summary()
predictor_new_hpo1.leaderboard(silent=True).plot(kind="bar", x="model", y="score_val")
```

```
{'root_mean_squared_error': -14.477321139849748, 'mean_squared_error': -209.59282738634042, 'mean_absolute_error': -9.298
*** Summary of fit() ***
Estimated performance of each model:
                    model  score_val              eval_metric  pred_time_val    fit_time  pred_time_val_marginal  fit_time
0     WeightedEnsemble_L3 -34.182172  root_mean_squared_error      77.180456  330.845855                0.001240
1     WeightedEnsemble_L2 -34.355720  root_mean_squared_error      76.247625  259.242725                0.000870
2         LightGBM_BAG_L2 -34.708988  root_mean_squared_error      76.407416  294.626449                0.160661
3         LightGBM_BAG_L1 -35.300237  root_mean_squared_error      11.354382   81.920167               11.354382
4       LightGBMXT_BAG_L2 -35.410542  root_mean_squared_error      76.691324  293.069369                0.444569
5  RandomForestMSE_BAG_L2 -35.416246  root_mean_squared_error      77.018555  295.365890                0.771800
6       LightGBMXT_BAG_L1 -36.142399  root_mean_squared_error      64.892374  177.300834               64.892374
Number of models trained: 7
Types of models trained:
{'WeightedEnsembleModel', 'StackerEnsembleModel_LGB', 'StackerEnsembleModel_RF'}
Bagging used: True  (with 8 folds)
Multi-layer stack-ensembling used: True  (with 3 levels)
Feature Metadata (Processed):
(raw dtype, special dtypes):
('category', [])             : 2 | ['season', 'weather']
('float', [])                : 3 | ['temp', 'atemp', 'windspeed']
('int', [])                  : 3 | ['humidity', 'hour', 'day_of_week']
('int', ['bool'])            : 2 | ['holiday', 'workingday']
('int', ['datetime_as_int']) : 4 | ['datetime', 'datetime.year', 'datetime.month', 'datetime.day']
*** End of fit() summary ***
/usr/local/lib/python3.10/dist-packages/autogluon/core/utils/plots.py:169: UserWarning: AutoGluon summary plots cannot be
  warnings.warn('AutoGluon summary plots cannot be created because bokeh is not installed. To see plots, please do: "pip
<Axes: xlabel='model'>
```




```
# Remember to set all negative values to zero
predictions_new_hpo1 = predictor_new_hpo1.predict(test)
predictions_new_hpo1.head()
predictions_description_new_hpo1 = predictions_new_hpo1.describe()
print(predictions_description_new_hpo1)
num_negative_values = (predictions_new_hpo1 < 0).sum()
print(f"Number of negative values: {num_negative_values}")
predictions_new_hpo1[predictions_new_hpo1 < 0] = 0
```

```
count    6493.000000
mean      190.406830
std       173.555099
min       -25.502981
25%        47.715908
50%       148.126572
```

```
75%       282.608582
max       886.808838
Name: count, dtype: float64
Number of negative values: 103
```

```
# Same submitting predictions
submission["count"] = predictions_new_hpo1
print(submission["count"].head())
submission.to_csv("submission_new_hpo1.csv", index=False)
```

```
0    17.823381
1     5.715229
2     4.324823
3     4.441064
4     4.176526
Name: count, dtype: float32
```

```
!kaggle competitions submit -c bike-sharing-demand -f submission_new_hpo1.csv -m "new features with hpo1"
```

```
100% 188k/188k [00:00<00:00, 306kB/s]
Successfully submitted to Bike Sharing Demand
```

```
!kaggle competitions submissions -c bike-sharing-demand | tail -n +1 | head -n 10
```

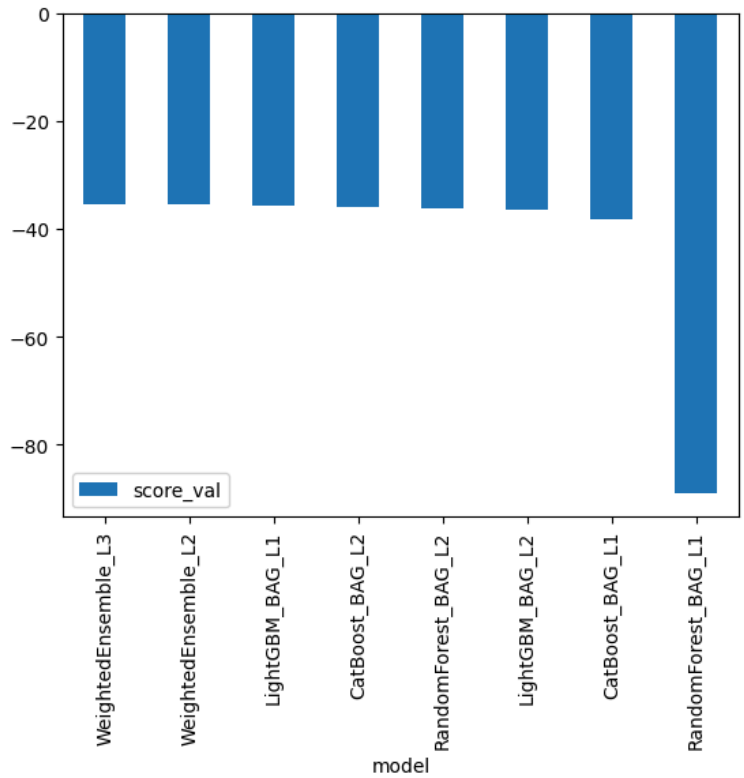
fileName	date	description	status	publicScore	privateScor
submission_new_hpo1.csv	2024-09-19 15:21:01	new features with hpo1	complete	0.58200	0.58200
submission_new_features.csv	2024-09-19 14:57:42	new features submission1	complete	0.68507	0.68507
submission.csv	2024-09-19 14:29:24	original data submission1	complete	1.84061	1.84061
submission_new_hpo.csv	2024-09-18 23:05:44	new features with hyperparameters-2	complete	0.48480	0.48480
submission_new_hpo.csv	2024-09-18 22:46:12	new features with hyperparameters	complete	0.59355	0.59355
submission_new_hpo.csv	2024-09-18 22:06:45	new features with hyperparameters	complete	0.49050	0.49050
submission_new_hpo.csv	2024-09-18 21:36:14	new features with hyperparameters	complete	0.49050	0.49050
submission_new_features.csv	2024-09-18 15:41:15	new features	complete	0.67692	0.67692

```
# 2. exclude +  hyperparams hpo2
predictor_new_hpo2 = TabularPredictor(
    label='count',
    eval_metric='root_mean_squared_error'
).fit(
    train_data=train,
    time_limit=time_limit,  # Adjust time limit as needed
    presets='best_quality',
    hyperparameters=hyperparameters,
    excluded_model_types=excluded_model_types,
)
```

Show hidden output

```
performance_new_hpo2 = predictor_new_hpo2.evaluate(train)
print(performance_new_hpo2)
predictor_new_hpo2.fit_summary()
predictor_new_hpo2.leaderboard(silent=True).plot(kind="bar", x="model", y="score_val")
```

```
{'root_mean_squared_error': -22.379799921419355, 'mean_squared_error': -500.85544452276184, 'mean_absolute_error': -14.42
*** Summary of fit() ***
Estimated performance of each model:
                 model  score_val              eval_metric  pred_time_val    fit_time  pred_time_val_marginal  fit_time_ma
0  WeightedEnsemble_L3 -35.481964  root_mean_squared_error       3.591426  128.185781                0.000952            0
1  WeightedEnsemble_L2 -35.579192  root_mean_squared_error       2.966970   91.557700                0.001032            0
2      LightGBM_BAG_L1 -35.731527  root_mean_squared_error       2.838163   42.858309                2.838163           42
3      CatBoost_BAG_L2 -35.942125  root_mean_squared_error       3.312475  125.781837                0.115207           33
4  RandomForest_BAG_L2 -36.296722  root_mean_squared_error       3.475267   94.938793                0.278000            2
5      LightGBM_BAG_L2 -36.596124  root_mean_squared_error       3.350437  119.260593                0.153169           26
6      CatBoost_BAG_L1 -38.140505  root_mean_squared_error       0.127776   48.668673                0.127776           48
7  RandomForest_BAG_L1 -88.924396  root_mean_squared_error       0.231329    1.039885                0.231329            1
Number of models trained: 8
Types of models trained:
{'WeightedEnsembleModel', 'StackerEnsembleModel_CatBoost', 'StackerEnsembleModel_LGB', 'StackerEnsembleModel_RF'}
Bagging used: True  (with 8 folds)
Multi-layer stack-ensembling used: True  (with 3 levels)
Feature Metadata (Processed):
(raw dtype, special dtypes):
('category', [])             : 2 | ['season', 'weather']
('float', [])                : 3 | ['temp', 'atemp', 'windspeed']
('int', [])                  : 3 | ['humidity', 'hour', 'day_of_week']
('int', ['bool'])            : 2 | ['holiday', 'workingday']
('int', ['datetime_as_int']) : 4 | ['datetime', 'datetime.year', 'datetime.month', 'datetime.day']
*** End of fit() summary ***
/usr/local/lib/python3.10/dist-packages/autogluon/core/utils/plots.py:169: UserWarning: AutoGluon summary plots cannot be
  warnings.warn('AutoGluon summary plots cannot be created because bokeh is not installed. To see plots, please do: "pip
<Axes: xlabel='model'>
```




```
predictions_new_hpo2 = predictor_new_hpo2.predict(test)
predictions_new_hpo2.head()
predictions_description_new_hpo2 = predictions_new_hpo2.describe()
print(predictions_description_new_hpo2)
num_negative_values = (predictions_new_hpo2 < 0).sum()
print(f"Number of negative values: {num_negative_values}")
predictions_new_hpo2[predictions_new_hpo2 < 0] = 0
```

```
count    6493.000000
mean      190.372559
std       173.766769
min       -23.048771
25%        47.012825
50%       147.574249
75%       285.164734
```

```
max       890.309570
Name: count, dtype: float64
Number of negative values: 96
```

```
submission["count"] = predictions_new_hpo2
print(submission["count"].head())
submission.to_csv("submission_new_hpo2.csv", index=False)
```

```
0    13.463066
1     2.278706
2     0.458701
3     2.398850
4     2.369717
Name: count, dtype: float32
```

```
!kaggle competitions submit -c bike-sharing-demand -f submission_new_hpo2.csv -m "new features with hpo2"
```

```
100% 188k/188k [00:00<00:00, 296kB/s]
Successfully submitted to Bike Sharing Demand
```

```
!kaggle competitions submissions -c bike-sharing-demand | tail -n +1 | head -n 10
```

fileName	date	description	status	publicScore	privateScor
submission_new_hpo2.csv	2024-09-19 15:54:03	new features with hpo2	complete	0.56763	0.56763
submission_new_hpo1.csv	2024-09-19 15:21:01	new features with hpo1	complete	0.58200	0.58200
submission_new_features.csv	2024-09-19 14:57:42	new features submission1	complete	0.68507	0.68507
submission.csv	2024-09-19 14:29:24	original data submission1	complete	1.84061	1.84061
submission_new_hpo.csv	2024-09-18 23:05:44	new features with hyperparameters-2	complete	0.48480	0.48480
submission_new_hpo.csv	2024-09-18 22:46:12	new features with hyperparameters	complete	0.59355	0.59355
submission_new_hpo.csv	2024-09-18 22:06:45	new features with hyperparameters	complete	0.49050	0.49050
submission_new_hpo.csv	2024-09-18 21:36:14	new features with hyperparameters	complete	0.49050	0.49050

```
time_limit = 1000
##############################
# 3. only time limit hpo3
predictor_new_hpo3 = TabularPredictor(
    label='count',
    eval_metric='root_mean_squared_error'
).fit(
    train_data=train,
    time_limit=time_limit,  # Adjust time limit as needed
    presets='best_quality',
    hyperparameters=hyperparameters,
    excluded_model_types=excluded_model_types,
)
```

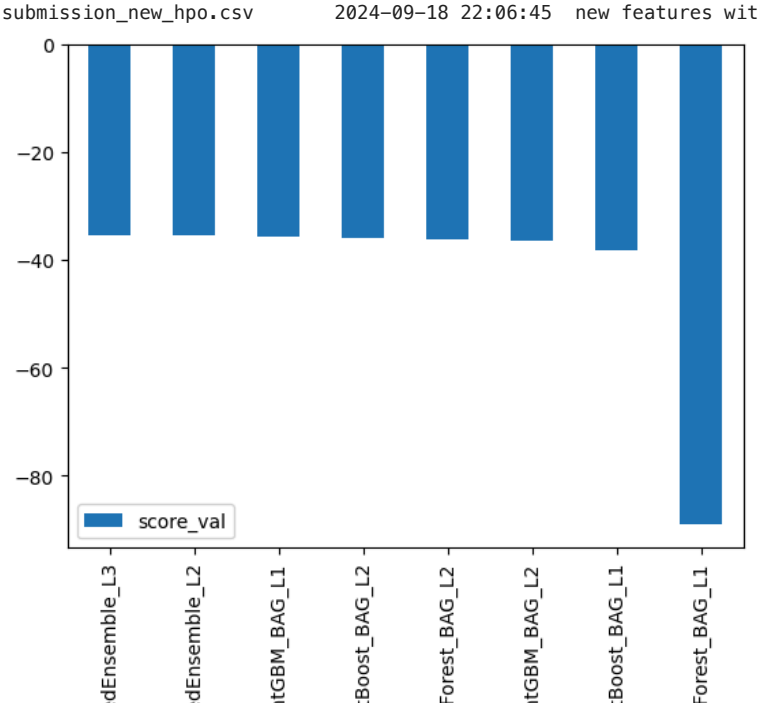
Show hidden output

```
performance_new_hpo3 = predictor_new_hpo3.evaluate(train)
print(performance_new_hpo3)
predictor_new_hpo3.fit_summary()
predictor_new_hpo3.leaderboard(silent=True).plot(kind="bar", x="model", y="score_val")
```

```
predictions_new_hpo3 = predictor_new_hpo3.predict(test)
predictions_new_hpo3.head()
predictions_description_new_hpo3 = predictions_new_hpo3.describe()
print(predictions_description_new_hpo3)
num_negative_values = (predictions_new_hpo3 < 0).sum()
print(f"Number of negative values: {num_negative_values}")
predictions_new_hpo3[predictions_new_hpo3 < 0] = 0
```

```
submission["count"] = predictions_new_hpo3
print(submission["count"].head())
submission.to_csv("submission_new_hpo3.csv", index=False)

!kaggle competitions submit -c bike-sharing-demand -f submission_new_hpo3.csv -m "new features with hpo3"
!kaggle competitions submissions -c bike-sharing-demand | tail -n +1 | head -n 10
```

```
{'root_mean_squared_error': -22.379799938813722, 'mean_squared_error': -500.8554453013267, 'mean_absolute_error': -14.4216
*** Summary of fit() ***
Estimated performance of each model:
                 model  score_val              eval_metric  pred_time_val    fit_time  pred_time_val_marginal  fit_time_ma
0  WeightedEnsemble_L3 -35.481964  root_mean_squared_error       5.948557  152.741427                0.001060           0.
1  WeightedEnsemble_L2 -35.579192  root_mean_squared_error       5.243127  108.105030                0.001039           0.
2      LightGBM_BAG_L1 -35.731527  root_mean_squared_error       5.068578   50.760226                5.068578          50.
3      CatBoost_BAG_L2 -35.942125  root_mean_squared_error       5.652593  150.817706                0.105721          41.
4  RandomForest_BAG_L2 -36.296722  root_mean_squared_error       5.841776  111.087182                0.294904           1.
5      LightGBM_BAG_L2 -36.596124  root_mean_squared_error       5.733250  145.045569                0.186377          35.
6      CatBoost_BAG_L1 -38.140505  root_mean_squared_error       0.173509   57.314312                0.173509          57.
7  RandomForest_BAG_L1 -88.924396  root_mean_squared_error       0.304785    1.127600                0.304785           1.
Number of models trained: 8
Types of models trained:
{'WeightedEnsembleModel', 'StackerEnsembleModel_CatBoost', 'StackerEnsembleModel_LGB', 'StackerEnsembleModel_RF'}
Bagging used: True  (with 8 folds)
Multi-layer stack-ensembling used: True  (with 3 levels)
Feature Metadata (Processed):
(raw dtype, special dtypes):
('category', [])             : 2 | ['season', 'weather']
('float', [])                : 3 | ['temp', 'atemp', 'windspeed']
('int', [])                  : 3 | ['humidity', 'hour', 'day_of_week']
('int', ['bool'])            : 2 | ['holiday', 'workingday']
('int', ['datetime_as_int']) : 4 | ['datetime', 'datetime.year', 'datetime.month', 'datetime.day']
*** End of fit() summary ***
/usr/local/lib/python3.10/dist-packages/autogluon/core/utils/plots.py:169: UserWarning: AutoGluon summary plots cannot be
  warnings.warn('AutoGluon summary plots cannot be created because bokeh is not installed. To see plots, please do: "pip
count    6493.000000
mean      190.372559
std       173.766769
min       -23.048771
25%        47.012825
50%       147.574249
75%       285.164703
max       890.309570
Name: count, dtype: float64
Number of negative values: 96
0    13.463066
1     2.278706
2     0.458701
3     2.398850
4     2.369717
Name: count, dtype: float32
100% 188k/188k [00:00<00:00, 289kB/s]
Successfully submitted to Bike Sharing DemandfileName                     date                 description
--------------------------  -------------------  ----------------------------------  --------  -----------  -----------
submission_new_hpo3.csv     2024-09-19 16:06:03  new features with hpo3              complete  0.56763      0.56763
submission_new_hpo2.csv     2024-09-19 15:54:03  new features with hpo2              complete  0.56763      0.56763
submission_new_hpo1.csv     2024-09-19 15:21:01  new features with hpo1              complete  0.58200      0.58200
submission_new_features.csv 2024-09-19 14:57:42  new features submission1            complete  0.68507      0.68507
submission.csv              2024-09-19 14:29:24  original data submission1           complete  1.84061      1.84061
submission_new_hpo.csv      2024-09-18 23:05:44  new features with hyperparameters-2 complete  0.48480      0.48480
submission_new_hpo.csv      2024-09-18 22:46:12  new features with hyperparameters   complete  0.59355      0.59355
submission_new_hpo.csv      2024-09-18 22:06:45  new features with hyperparameters   complete  0.49050      0.49050
```



New Score of ?

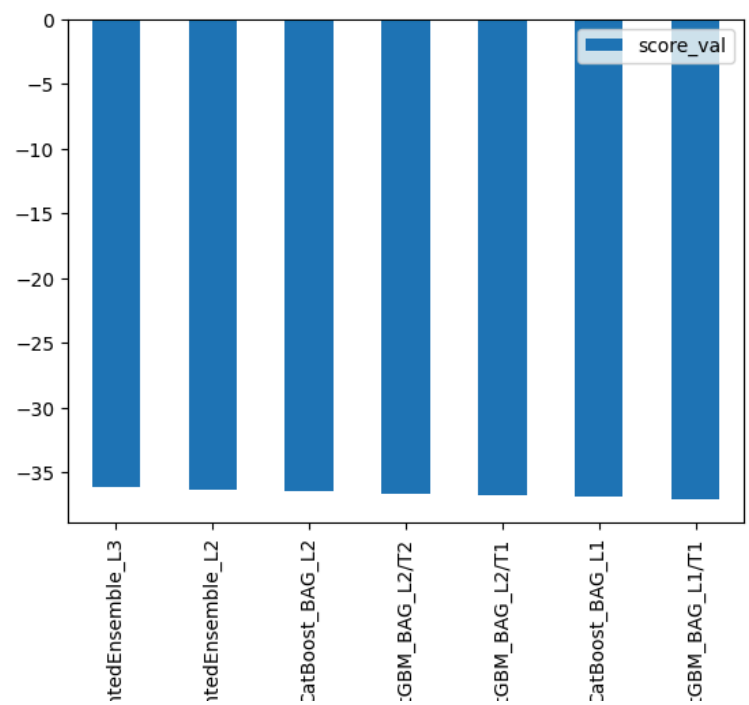
```
time_limit = 600
hyperparameters = {
    'GBM': [
        {
        'learning_rate': 0.01,  # Lower learning rate for more gradual updates
        'num_boost_round': 1500,  # More boosting rounds
        'num_leaves': 40,  # Increase leaves for more complex trees
        'feature_fraction': 0.8,  # Try lowering feature fraction for better generalization
        'bagging_fraction': 0.7,  # Decrease to reduce overfitting
        'bagging_freq': 5,  # Bagging every 5 iterations
        'max_depth': 15,  # Allow deeper trees
        'early_stopping_rounds': 100  # Early stop if no improvement
        }
    ],
    'CAT': [
        {
            'iterations': 1200,  # Increase iterations
            'depth': 8,  # Adjust depth
            'learning_rate': 0.03,  # Slightly lower learning rate
            'l2_leaf_reg': 4.0,  # Increase regularization to avoid overfitting
            'one_hot_max_size': 10,  # Categorical features with more categories use one-hot encoding
            'eval_metric': 'RMSE',  # Ensure you're optimizing for RMSE
            'od_type': 'Iter',  # Use iterative early stopping
            'od_wait': 50  # Early stopping after 50 iterations without improvement
            }
        ],
}

hyperparameter_tune_kwargs = {
    'num_trials': 20,              # Try 20 different configurations
    'scheduler': 'local',          # Run on local machine
    'searcher': 'random',          # Random search for hyperparameters
    'max_t': 600                   # Max time for each trial is 600 seconds
}
excluded_model_types = ['RandomForest', 'KNN']

##############################
# 4. hyperparams,hyperparameter_tune_kwargs, modelexclude  hpo4
predictor_new_hpo4 = TabularPredictor(
    label='count',
    eval_metric='root_mean_squared_error'
).fit(
    train_data=train,
    time_limit=time_limit,  # Adjust time limit as needed
    presets='best_quality',
    hyperparameters=hyperparameters,
    excluded_model_types=excluded_model_types,
    hyperparameter_tune_kwargs=hyperparameter_tune_kwargs
)
```

Show hidden output

```
performance_new_hpo4 = predictor_new_hpo4.evaluate(train)
print(performance_new_hpo4)
predictor_new_hpo4.fit_summary()
predictor_new_hpo4.leaderboard(silent=True).plot(kind="bar", x="model", y="score_val")


predictions_new_hpo4 = predictor_new_hpo4.predict(test)
predictions_new_hpo4.head()
predictions_description_new_hpo4 = predictions_new_hpo4.describe()
```

```
print(predictions_description_new_hpo4)
num_negative_values = (predictions_new_hpo4 < 0).sum()
print(f"Number of negative values: {num_negative_values}")
predictions_new_hpo4[predictions_new_hpo4 < 0] = 0


submission["count"] = predictions_new_hpo4
print(submission["count"].head())
submission.to_csv("submission_new_hpo4.csv", index=False)

!kaggle competitions submit -c bike-sharing-demand -f submission_new_hpo4.csv -m "new features with hpo4"
!kaggle competitions submissions -c bike-sharing-demand | tail -n +1 | head -n 10
```

```
{'root_mean_squared_error': -26.451148962962055, 'mean_squared_error': -699.6632814608085, 'mean_absolute_error': -16.825
*** Summary of fit() ***
Estimated performance of each model:
                 model  score_val              eval_metric  pred_time_val    fit_time  pred_time_val_marginal  fit_time_ma
0  WeightedEnsemble_L3 -36.159395  root_mean_squared_error       9.004457  249.571208                0.002559            0
1  WeightedEnsemble_L2 -36.290011  root_mean_squared_error       7.397759  150.402372                0.001327            0
2      CatBoost_BAG_L2 -36.439848  root_mean_squared_error       7.509053  187.953966                0.112620           37
3   LightGBM_BAG_L2/T2 -36.600079  root_mean_squared_error       8.889277  211.945760                1.492845           61
4   LightGBM_BAG_L2/T1 -36.738209  root_mean_squared_error       8.280383  214.872155                0.883950           64
5      CatBoost_BAG_L1 -36.828813  root_mean_squared_error       0.183485   71.999720                0.183485           71
6   LightGBM_BAG_L1/T1 -37.056601  root_mean_squared_error       7.212947   78.380610                7.212947           78
Number of models trained: 7
Types of models trained:
{'WeightedEnsembleModel', 'StackerEnsembleModel_LGB', 'StackerEnsembleModel_CatBoost'}
Bagging used: True  (with 8 folds)
Multi-layer stack-ensembling used: True  (with 3 levels)
Feature Metadata (Processed):
(raw dtype, special dtypes):
('category', [])             : 2 | ['season', 'weather']
('float', [])                : 3 | ['temp', 'atemp', 'windspeed']
('int', [])                  : 3 | ['humidity', 'hour', 'day_of_week']
('int', ['bool'])            : 2 | ['holiday', 'workingday']
('int', ['datetime_as_int']) : 4 | ['datetime', 'datetime.year', 'datetime.month', 'datetime.day']
*** End of fit() summary ***
/usr/local/lib/python3.10/dist-packages/autogluon/core/utils/plots.py:169: UserWarning: AutoGluon summary plots cannot be
  warnings.warn('AutoGluon summary plots cannot be created because bokeh is not installed. To see plots, please do: "pip
count    6493.000000
mean      190.309570
std       172.785767
min       -10.918406
25%        47.419312
50%       148.868439
75%       284.332977
max       876.426147
Name: count, dtype: float64
Number of negative values: 32
0    13.938789
1     4.093366
2     1.929388
3     2.782651
4     2.931469
Name: count, dtype: float32
100% 188k/188k [00:01<00:00, 177kB/s]
Successfully submitted to Bike Sharing DemandfileName                     date                 description
--------------------------  -------------------  ----------------------------------  --------  -----------  ------------
submission_new_hpo4.csv     2024-09-19 16:35:40  new features with hpo4              complete  0.51728      0.51728
submission_new_hpo3.csv     2024-09-19 16:06:03  new features with hpo3              complete  0.56763      0.56763
submission_new_hpo2.csv     2024-09-19 15:54:03  new features with hpo2              complete  0.56763      0.56763
submission_new_hpo1.csv     2024-09-19 15:21:01  new features with hpo1              complete  0.58200      0.58200
submission_new_features.csv 2024-09-19 14:57:42  new features submission1            complete  0.68507      0.68507
submission.csv              2024-09-19 14:29:24  original data submission1           complete  1.84061      1.84061
submission_new_hpo.csv      2024-09-18 23:05:44  new features with hyperparameters-2 complete  0.48480      0.48480
submission_new_hpo.csv      2024-09-18 22:46:12  new features with hyperparameters   complete  0.59355      0.59355
```




```
hyperparameters = {
    'GBM': [
        {'extra_trees': True, 'ag_args': {'name_suffix': 'XT'}},
        {'learning_rate': 0.05, 'num_leaves': 31, 'feature_fraction': 0.8}
    ],
    'CAT': [
        {'depth': 6, 'l2_leaf_reg': 3.0},
        {'depth': 8, 'grow_policy': 'Depthwise'}
    ],
    'XGB': [
        {'max_depth': 6, 'learning_rate': 0.1},
        {'max_depth': 10, 'learning_rate': 0.05}
    ],
    'NN_TORCH': [
        {'num_layers': 3, 'hidden_size': 128, 'dropout_prob': 0.1, 'learning_rate': 0.001}
    ]
}

predictor_new_hpo5 = TabularPredictor(
    label='count',
    eval_metric='root_mean_squared_error'
).fit(
    train_data=train,
    time_limit=time_limit,  # Adjust time limit as needed
    presets='best_quality',
    hyperparameters=hyperparameters,
    excluded_model_types=excluded_model_types,
)
```

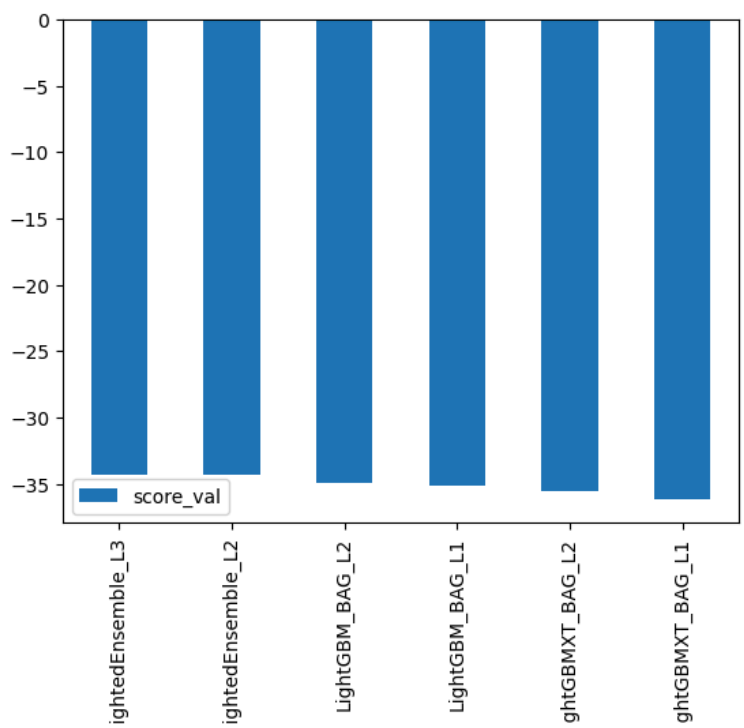
Show hidden output

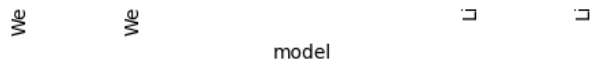
```
performance_new_hpo5 = predictor_new_hpo5.evaluate(train)
print(performance_new_hpo5)
predictor_new_hpo5.fit_summary()
predictor_new_hpo5.leaderboard(silent=True).plot(kind="bar", x="model", y="score_val")
```

```
predictions_new_hpo5 = predictor_new_hpo5.predict(test)
predictions_new_hpo5.head()
predictions_description_new_hpo5 = predictions_new_hpo5.describe()
print(predictions_description_new_hpo5)
num_negative_values = (predictions_new_hpo5 < 0).sum()
print(f"Number of negative values: {num_negative_values}")
predictions_new_hpo5[predictions_new_hpo5 < 0] = 0
```

```
submission["count"] = predictions_new_hpo5
print(submission["count"].head())
submission.to_csv("submission_new_hpo5.csv", index=False)

!kaggle competitions submit -c bike-sharing-demand -f submission_new_hpo5.csv -m "new features with hpo5"
!kaggle competitions submissions -c bike-sharing-demand | tail -n +1 | head -n 10
```

```
{'root_mean_squared_error': -14.601223582489723, 'mean_squared_error': -213.195730105854, 'mean_absolute_error': -9.45508
*** Summary of fit() ***
Estimated performance of each model:
                 model  score_val              eval_metric  pred_time_val    fit_time  pred_time_val_marginal  fit_time_ma
0  WeightedEnsemble_L3 -34.264396  root_mean_squared_error      96.341338  316.801059                0.001557            0
1  WeightedEnsemble_L2 -34.350652  root_mean_squared_error      96.072733  276.197695                0.001138            0
2      LightGBM_BAG_L2 -34.922211  root_mean_squared_error      96.339780  316.762206                0.268186           40
3      LightGBM_BAG_L1 -35.104548  root_mean_squared_error      13.164066   86.320622               13.164066           86
4    LightGBMXT_BAG_L2 -35.529800  root_mean_squared_error      96.670201  317.410384                0.598607           41
5    LightGBMXT_BAG_L1 -36.142399  root_mean_squared_error      82.907528  189.850192               82.907528          189
Number of models trained: 6
Types of models trained:
{'WeightedEnsembleModel', 'StackerEnsembleModel_LGB'}
Bagging used: True  (with 8 folds)
Multi-layer stack-ensembling used: True  (with 3 levels)
Feature Metadata (Processed):
(raw dtype, special dtypes):
('category', [])             : 2 | ['season', 'weather']
('float', [])                : 3 | ['temp', 'atemp', 'windspeed']
('int', [])                  : 3 | ['humidity', 'hour', 'day_of_week']
('int', ['bool'])            : 2 | ['holiday', 'workingday']
('int', ['datetime_as_int']) : 4 | ['datetime', 'datetime.year', 'datetime.month', 'datetime.day']
*** End of fit() summary ***
/usr/local/lib/python3.10/dist-packages/autogluon/core/utils/plots.py:169: UserWarning: AutoGluon summary plots cannot be
  warnings.warn('AutoGluon summary plots cannot be created because bokeh is not installed. To see plots, please do: "pip
count    6493.000000
mean      190.193268
std       173.355576
min       -34.301201
25%        48.061779
50%       147.612396
75%       283.574432
max       880.251160
Name: count, dtype: float64
Number of negative values: 142
0    19.087494
1     5.052511
2     3.375005
3     3.322581
4     3.186225
Name: count, dtype: float32
100% 187k/187k [00:00<00:00, 285kB/s]
Successfully submitted to Bike Sharing DemandfileName                    date                 description
---------------------------  -------------------  ----------------------------------  --------  -----------  -----------
submission_new_hpo5.csv      2024-09-19 18:01:48  new features with hpo5              complete  0.61440      0.61440
submission_new_hpo4.csv      2024-09-19 16:35:40  new features with hpo4              complete  0.51728      0.51728
submission_new_hpo3.csv      2024-09-19 16:06:03  new features with hpo3              complete  0.56763      0.56763
submission_new_hpo2.csv      2024-09-19 15:54:03  new features with hpo2              complete  0.56763      0.56763
submission_new_hpo1.csv      2024-09-19 15:21:01  new features with hpo1              complete  0.58200      0.58200
submission_new_features.csv  2024-09-19 14:57:42  new features submission1            complete  0.68507      0.68507
submission.csv               2024-09-19 14:29:24  original data submission1           complete  1.84061      1.84061
submission_new_hpo.csv       2024-09-18 23:05:44  new features with hyperparameters-2 complete  0.48480      0.48480
```






```
hyperparameter_tune_kwargs = {
    'num_trials': 20,              # Try 20 different configurations
    'scheduler': 'local',          # Run on local machine
    'searcher': 'random',          # Random search for hyperparameters
    'max_t': 600                   # Max time for each trial is 600 seconds
}


# Exclude poor-performing models and define hyperparameters for tuning
hyperparameters = {
    'GBM': [
        {'num_leaves': 31, 'learning_rate': 0.05, 'num_boost_round': 100},  # LightGBM_BAG_L1
        {'num_leaves': 64, 'learning_rate': 0.03, 'num_boost_round': 200, 'extra_trees': True},  # LightGBMXT_BAG_L2
    ],
    'RF': [
        {'n_estimators': 200, 'max_depth': 20},  # RandomForestMSE_BAG_L2
    ],
    'CAT': [
        {'depth': 6, 'learning_rate': 0.1},  # CatBoost_BAG_L1 - optional exclusion
    ],
}

excluded_model_types = ['KNN', 'CAT']

predictor_new_hpo6 = TabularPredictor(
    label='count',
    eval_metric='root_mean_squared_error'
).fit(
    train_data=train,
    time_limit=2100,  # Adjust time limit as needed
    presets='best_quality',
    num_bag_folds=5,  # Enables bagging
    num_stack_levels=2,  # Adds stacking
    hyperparameters=hyperparameters,  # Apply the tuned hyperparameters
    excluded_model_types=excluded_model_types,  # Exclude KNN and optionally CatBoost
    hyperparameter_tune_kwargs=hyperparameter_tune_kwargs  # Apply hyperparameter tuning
)
```

Show hidden output

```
performance_new_hpo6 = predictor_new_hpo6.evaluate(train)
print(performance_new_hpo6)
predictor_new_hpo6.fit_summary()
predictor_new_hpo6.leaderboard(silent=True).plot(kind="bar", x="model", y="score_val")
```

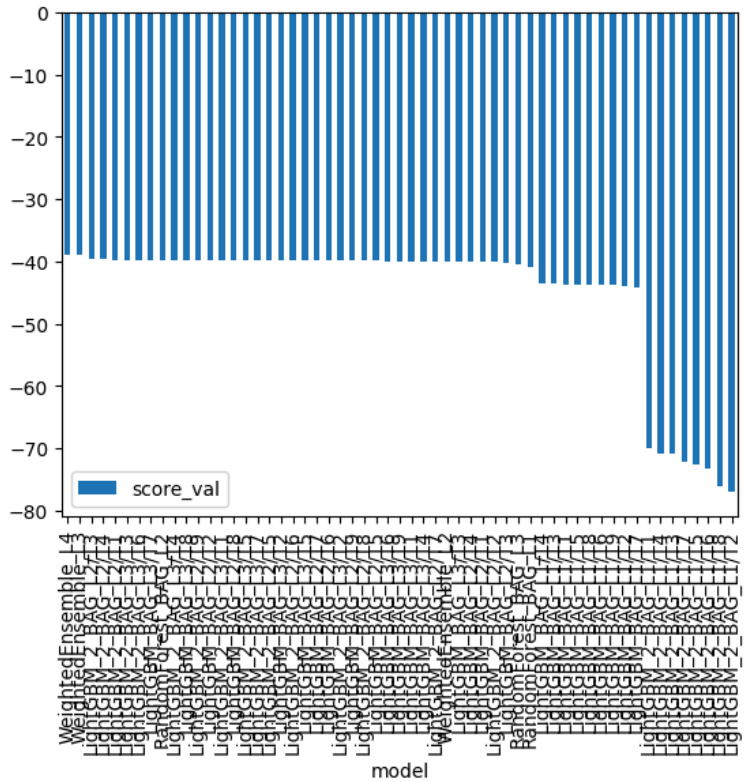
```
predictions_new_hpo6 = predictor_new_hpo6.predict(test)
predictions_new_hpo6.head()
predictions_description_new_hpo6 = predictions_new_hpo6.describe()
print(predictions_description_new_hpo6)
num_negative_values = (predictions_new_hpo6 < 0).sum()
print(f"Number of negative values: {num_negative_values}")
predictions_new_hpo6[predictions_new_hpo6 < 0] = 0
```

```
submission["count"] = predictions_new_hpo6
print(submission["count"].head())
submission.to_csv("submission_new_hpo6.csv", index=False)

!kaggle competitions submit -c bike-sharing-demand -f submission_new_hpo6.csv -m "new features with hpo6"
!kaggle competitions submissions -c bike-sharing-demand | tail -n +1 | head -n 10
```

```
{'root_mean_squared_error': -24.041199476088977, 'mean_squared_error': -577.9792722491009, 'mean_absolute_error': -14.2473
*** Summary of fit() ***
Estimated performance of each model:
                   model  score_val              eval_metric  pred_time_val     fit_time  pred_time_val_marginal  fit_time
0    WeightedEnsemble_L4 -38.866315  root_mean_squared_error      17.834112   964.424213                0.009089
1    WeightedEnsemble_L3 -38.923886  root_mean_squared_error      11.480837   558.288430                0.001498
2   LightGBM_2_BAG_L2/T3 -39.516641  root_mean_squared_error      10.252568   441.607559                0.401053
3   LightGBM_2_BAG_L2/T4 -39.630428  root_mean_squared_error      10.323291   438.347003                0.471775
4   LightGBM_2_BAG_L2/T1 -39.676151  root_mean_squared_error      10.667203   440.031950                0.815688
5   LightGBM_2_BAG_L3/T3 -39.708451  root_mean_squared_error      17.825023   964.279827                0.307158
6   LightGBM_2_BAG_L3/T6 -39.724349  root_mean_squared_error      17.842237   960.266480                0.324372
7     LightGBM_BAG_L3/T7 -39.735620  root_mean_squared_error      17.686483   958.697613                0.168618
8    RandomForest_BAG_L2 -39.764065  root_mean_squared_error      10.448413   485.183140                0.596898
9   LightGBM_2_BAG_L3/T4 -39.764349  root_mean_squared_error      17.879830   971.902098                0.361965
10    LightGBM_BAG_L3/T8 -39.766125  root_mean_squared_error      17.681036   959.358323                0.163171
11  LightGBM_2_BAG_L2/T9 -39.770464  root_mean_squared_error      10.214834   438.426243                0.363319
12    LightGBM_BAG_L2/T2 -39.785295  root_mean_squared_error      10.009613   437.599732                0.158098
13  LightGBM_2_BAG_L3/T1 -39.794484  root_mean_squared_error      18.072521   963.804584                0.554656
14    LightGBM_BAG_L2/T8 -39.797646  root_mean_squared_error      10.064741   437.730843                0.213226
15  LightGBM_2_BAG_L3/T5 -39.805070  root_mean_squared_error      17.991835   962.907531                0.473969
16  LightGBM_2_BAG_L3/T7 -39.805601  root_mean_squared_error      18.330670   963.195241                0.812804
17  LightGBM_2_BAG_L2/T5 -39.812638  root_mean_squared_error      10.342887   441.384401                0.491372
18    LightGBM_BAG_L3/T2 -39.827026  root_mean_squared_error      17.656074   961.723058                0.138209
19  LightGBM_2_BAG_L2/T6 -39.834598  root_mean_squared_error      10.390557   439.971141                0.539042
20    LightGBM_BAG_L3/T5 -39.837287  root_mean_squared_error      17.635073   958.468132                0.117208
21    LightGBM_BAG_L2/T7 -39.843918  root_mean_squared_error      10.960736   441.619077                1.109221
22    LightGBM_BAG_L2/T6 -39.857811  root_mean_squared_error       9.959526   442.221708                0.108011
23  LightGBM_2_BAG_L3/T2 -39.858246  root_mean_squared_error      18.258092   961.193622                0.740227
24    LightGBM_BAG_L2/T9 -39.864472  root_mean_squared_error       9.957956   440.941211                0.106440
25  LightGBM_2_BAG_L2/T8 -39.881890  root_mean_squared_error      10.261394   440.748075                0.409878
26    LightGBM_BAG_L2/T5 -39.885057  root_mean_squared_error      10.205992   440.722847                0.354477
27    LightGBM_BAG_L3/T6 -39.901952  root_mean_squared_error      17.710853   960.274309                0.192988
28    LightGBM_BAG_L3/T9 -39.911281  root_mean_squared_error      17.676189   960.622160                0.158324
29    LightGBM_BAG_L3/T1 -39.930304  root_mean_squared_error      17.609822   957.714297                0.091957
30    LightGBM_BAG_L3/T4 -39.938965  root_mean_squared_error      17.635835   960.492187                0.117970
31  LightGBM_2_BAG_L2/T7 -39.988148  root_mean_squared_error      10.434087   439.568457                0.582572
32   WeightedEnsemble_L2 -40.019612  root_mean_squared_error       1.078497    83.984021                0.001270
33    LightGBM_BAG_L3/T3 -40.031350  root_mean_squared_error      17.610585   958.784619                0.092720
34    LightGBM_BAG_L2/T4 -40.033262  root_mean_squared_error       9.994263   437.970570                0.142748
35    LightGBM_BAG_L2/T1 -40.039757  root_mean_squared_error      10.035031   438.624011                0.183516
36  LightGBM_2_BAG_L2/T2 -40.105059  root_mean_squared_error      10.337329   440.696135                0.485814
37    LightGBM_BAG_L2/T3 -40.128011  root_mean_squared_error       9.984719   440.110577                0.133204
38   RandomForest_BAG_L3 -40.341486  root_mean_squared_error      18.295734  1010.174319                0.777869
39   RandomForest_BAG_L1 -40.865432  root_mean_squared_error       0.468518    17.718006                0.468518
40    LightGBM_BAG_L1/T4 -43.436702  root_mean_squared_error       0.284545    22.610137                0.284545
41    LightGBM_BAG_L1/T3 -43.577649  root_mean_squared_error       0.149347    21.337055                0.149347
42    LightGBM_BAG_L1/T1 -43.655737  root_mean_squared_error       0.203706    22.241017                0.203706
43    LightGBM_BAG_L1/T5 -43.669529  root_mean_squared_error       0.225091    23.490895                0.225091
44    LightGBM_BAG_L1/T8 -43.758578  root_mean_squared_error       0.120457    21.373649                0.120457
45    LightGBM_BAG_L1/T6 -43.781264  root_mean_squared_error       0.194602    21.636574                0.194602
46    LightGBM_BAG_L1/T9 -43.781264  root_mean_squared_error       0.265316    21.062228                0.265316
47    LightGBM_BAG_L1/T2 -43.997586  root_mean_squared_error       0.199910    23.540038                0.199910
48    LightGBM_BAG_L1/T7 -44.169440  root_mean_squared_error       0.164887    23.952114                0.164887
49  LightGBM_2_BAG_L1/T1 -69.949954  root_mean_squared_error       1.560199    26.158452                1.560199
50  LightGBM_2_BAG_L1/T4 -70.768479  root_mean_squared_error       0.522519    23.956675                0.522519
51  LightGBM_2_BAG_L1/T3 -70.810907  root_mean_squared_error       1.509157    26.643432                1.509157
52  LightGBM_2_BAG_L1/T7 -72.223184  root_mean_squared_error       1.289268    25.671272                1.289268
53  LightGBM_2_BAG_L1/T5 -72.616958  root_mean_squared_error       0.622241    25.384210                0.622241
54  LightGBM_2_BAG_L1/T6 -73.316794  root_mean_squared_error       0.684447    23.470022                0.684447
55  LightGBM_2_BAG_L1/T8 -76.101025  root_mean_squared_error       0.675498    22.437423                0.675498
56  LightGBM_2_BAG_L1/T2 -76.939353  root_mean_squared_error       0.711805    22.149708                0.711805
Number of models trained: 57
Types of models trained:
{'WeightedEnsembleModel', 'StackerEnsembleModel_LGB', 'StackerEnsembleModel_RF'}
Bagging used: True  (with 5 folds)
Multi-layer stack-ensembling used: True  (with 4 levels)
Feature Metadata (Processed):
(raw dtype, special dtypes):
('category', [])             : 2 | ['season', 'weather']
('float', [])                : 3 | ['temp', 'atemp', 'windspeed']
('int', [])                  : 3 | ['humidity', 'hour', 'day_of_week']
('int', ['bool'])            : 2 | ['holiday', 'workingday']
('int', ['datetime_as_int']) : 4 | ['datetime', 'datetime.year', 'datetime.month', 'datetime.day']
*** End of fit() summary ***
/usr/local/lib/python3.10/dist-packages/autogluon/core/utils/plots.py:169: UserWarning: AutoGluon summary plots cannot be
  warnings.warn('AutoGluon summary plots cannot be created because bokeh is not installed. To see plots, please do: "pip i
count    6493.000000
mean      193.139328
std       174.147583
min         2.781564
25%        49.317539
50%       151.653000
75%       288.005157
```

```
max       872.455566
Name: count, dtype: float64
Number of negative values: 0
0    12.094127
1     5.861425
2     3.857027
3     3.288101
4     3.428241
Name: count, dtype: float32
100% 188k/188k [00:00<00:00, 316kB/s]
Successfully submitted to Bike Sharing Demand
```

fileName	date	description			
submission_new_hpo6.csv	2024-09-19 22:10:50	new features with hpo6	complete	0.48414	0.48414
submission_new_hpo6.csv	2024-09-19 20:48:25	new features with hpo6	complete	0.48480	0.48480
submission_new_features2.csv	2024-09-19 19:56:20	new features submission2	complete	0.75401	0.75401
submission_new_hpo5.csv	2024-09-19 18:01:48	new features with hpo5	complete	0.61440	0.61440
submission_new_hpo4.csv	2024-09-19 16:35:40	new features with hpo4	complete	0.51728	0.51728
submission_new_hpo3.csv	2024-09-19 16:06:03	new features with hpo3	complete	0.56763	0.56763
submission_new_hpo2.csv	2024-09-19 15:54:03	new features with hpo2	complete	0.56763	0.56763
submission_new_hpo1.csv	2024-09-19 15:21:01	new features with hpo1	complete	0.58200	0.58200



```
!kaggle competitions submissions -c bike-sharing-demand | tail -n +1 | head -n 11
```

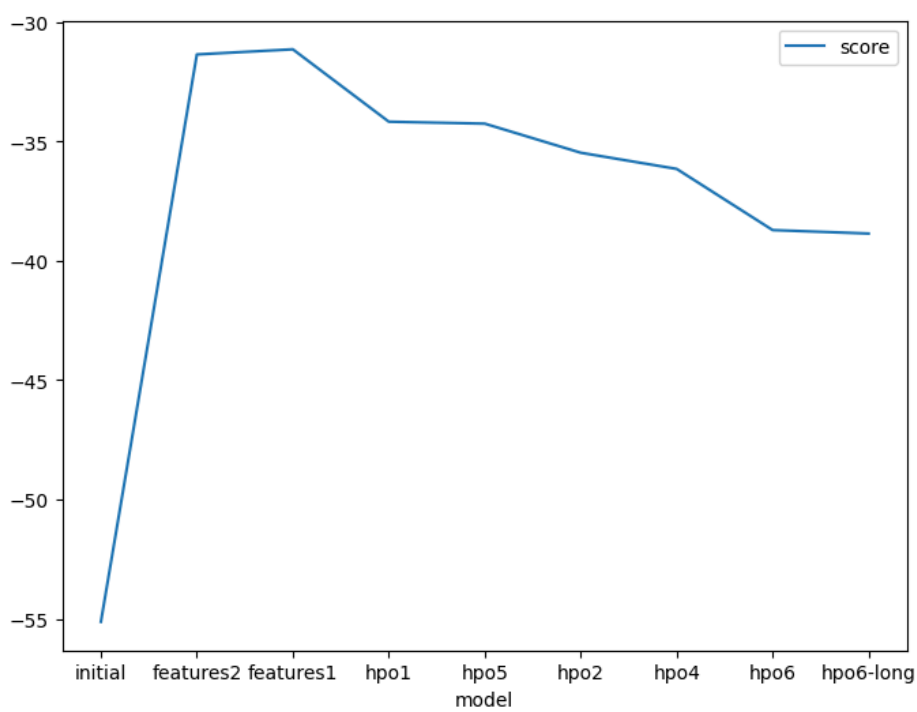
```
fileName                      date                 description                          status    publicScore  privateSco
----------------------------  -------------------  -----------------------------------  --------  -----------  ----------
submission_new_hpo6.csv       2024-09-19 20:48:25  new features with hpo6               complete  0.48480      0.48480
submission_new_features2.csv  2024-09-19 19:56:20  new features submission2             complete  0.75401      0.75401
submission_new_hpo5.csv       2024-09-19 18:01:48  new features with hpo5               complete  0.61440      0.61440
submission_new_hpo4.csv       2024-09-19 16:35:40  new features with hpo4               complete  0.51728      0.51728
submission_new_hpo3.csv       2024-09-19 16:06:03  new features with hpo3               complete  0.56763      0.56763
submission_new_hpo2.csv       2024-09-19 15:54:03  new features with hpo2               complete  0.56763      0.56763
submission_new_hpo1.csv       2024-09-19 15:21:01  new features with hpo1               complete  0.58200      0.58200
submission_new_features.csv   2024-09-19 14:57:42  new features submission1             complete  0.68507      0.68507
submission.csv                2024-09-19 14:29:24  original data submission1            complete  1.84061      1.84061
```

Step 7: Write a Report

Refer to the markdown file for the full report

Creating plots and table for report

```
# Taking the top model score from each training run and creating a line plot to show improvement
# You can create these in the notebook and save them to PNG or use some other tool (e.g. google sheets, excel)
fig = pd.DataFrame(
    {
        "model": ["initial","features2", "features1", "hpo1", "hpo5", "hpo2","hpo4", "hpo6", "hpo6-long"],
        "score": [-55.123669, -31.366154, -31.151678, -34.182172, -34.264396, -35.481964, -36.159395, -38.721307, -38.866315
    }
).plot(x="model", y="score", figsize=(8, 6)).get_figure()
fig.savefig('model_train_score.png')
```




```
# Take the 3 kaggle scores and creating a line plot to show improvement
fig = pd.DataFrame(
    {
        "test_eval": ["initial", "features2", "features1", "hpo1", "hpo5", "hpo2", "hpo4", "hpo6", "hpo6-long"],
        "score": [1.84061,  0.75401 , 0.68507 , 0.58200 , 0.6144, 0.56763 , 0.51728, 0.4848,  0.48414]
    }
```