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
```

```
  Attempting uninstall: hyperopt
    Found existing installation: hyperopt 0.1.2
    Uninstalling hyperopt-0.1.2:
      Successfully uninstalled hyperopt-0.1.2
  Attempting uninstall: dask
    Found existing installation: dask 2022.2.1
    Uninstalling dask-2022.2.1:
      Successfully uninstalled dask-2022.2.1
  Attempting uninstall: torchvision
    Found existing installation: torchvision 0.14.0+cu116
    Uninstalling torchvision-0.14.0+cu116:
      Successfully uninstalled torchvision-0.14.0+cu116
  Attempting uninstall: torchtext
    Found existing installation: torchtext 0.14.0
    Uninstalling torchtext-0.14.0:
      Successfully uninstalled torchtext-0.14.0
  Attempting uninstall: statsmodels
    Found existing installation: statsmodels 0.12.2
    Uninstalling statsmodels-0.12.2:
      Successfully uninstalled statsmodels-0.12.2
  Attempting uninstall: scikit-image
    Found existing installation: scikit-image 0.18.3
    Uninstalling scikit-image-0.18.3:
      Successfully uninstalled scikit-image-0.18.3
  Attempting uninstall: lightgbm
    Found existing installation: lightgbm 2.2.3
    Uninstalling lightgbm-2.2.3:
      Successfully uninstalled lightgbm-2.2.3
  Attempting uninstall: distributed
    Found existing installation: distributed 2022.2.1
    Uninstalling distributed-2022.2.1:
      Successfully uninstalled distributed-2022.2.1
  Attempting uninstall: albumentations
    Found existing installation: albumentations 1.2.1
    Uninstalling albumentations-1.2.1:
      Successfully uninstalled albumentations-1.2.1
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviour is the sou
torchaudio 0.13.0+cu116 requires torch==1.13.0, but you have torch 1.12.1 which is incompatible.
panel 0.12.1 requires bokeh<2.4.0,>=2.3.0, but you have bokeh 2.0.1 which is incompatible.
grpcio-status 1.48.2 requires grpcio>=1.48.2, but you have grpcio 1.43.0 which is incompatible.
google-cloud-bigquery 3.3.6 requires grpcio<2.0dev,>=1.47.0, but you have grpcio 1.43.0 which is incompatible.
```

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 = "rethinaduraisj"
kaggle_key = "f533c759f6b84b373c7bc03930fa4eb3"

# 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, 424kB/s]
100% 189k/189k [00:00<00:00, 423kB/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")
```

```
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

```
train = train.drop('casual', axis=1)
```

```
train = train.drop('registered', axis=1)
```

```
train["datetime"]=pd.to_datetime(train["datetime"])
```

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

	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	co
count	10886.000000	10886.000000	10886.000000	10886.000000	10886.00000	10886.000000	10886.000000	10886.000000	10886.000
mean	2.506614	0.028569	0.680875	1.418427	20.23086	23.655084	61.886460	12.799395	191.574
std	1.116174	0.166599	0.466159	0.633839	7.79159	8.474601	19.245033	8.164537	181.144
min	1.000000	0.000000	0.000000	1.000000	0.82000	0.760000	0.000000	0.000000	1.000
25%	2.000000	0.000000	0.000000	1.000000	13.94000	16.665000	47.000000	7.001500	42.000
50%	3.000000	0.000000	1.000000	1.000000	20.50000	24.240000	62.000000	12.998000	145.000
75%	4.000000	0.000000	1.000000	2.000000	26.24000	31.060000	77.000000	16.997900	284.000
max	4.000000	1.000000	1.000000	4.000000	41.00000	45.455000	100.000000	56.996900	977.000

```
train.head()
```

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

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

```
predictor = TabularPredictor(label="count", problem_type="regression", eval_metric="root_mean_squared_error").fit(
    train_data=train, time_limit=600, presets="best_quality"
)
```

```
No path specified. Models will be saved in: "AutogluonModels/ag-20221228_035727/"
Presets specified: ['best_quality']
Stack configuration (auto_stack=True): num_stack_levels=1, num_bag_folds=8, num_bag_sets=20
Beginning AutoGluon training ... Time limit = 600s
AutoGluon will save models to "AutogluonModels/ag-20221228_035727/"
AutoGluon Version:  0.6.1
Python Version:     3.8.16
Operating System:   Linux
Platform Machine:   x86_64
Platform Version:   #1 SMP Fri Aug 26 08:44:51 UTC 2022
Train Data Rows:    10886
Train Data Columns: 9
Label Column: count
Preprocessing data ...
Using Feature Generators to preprocess the data ...
Fitting AutoMLPipelineFeatureGenerator...
        Available Memory:                    12210.18 MB
        Train Data (Original)  Memory Usage: 0.78 MB (0.0% of available memory)
        Inferring data type of each feature based on column values. Set feature_metadata_in to manually specify special dty
        Stage 1 Generators:
                Fitting AsTypeFeatureGenerator...
                        Note: Converting 2 features to boolean dtype as they only contain 2 unique values.
        Stage 2 Generators:
                Fitting FillNaFeatureGenerator...
        Stage 3 Generators:
                Fitting IdentityFeatureGenerator...
                Fitting DatetimeFeatureGenerator...
/usr/local/lib/python3.8/dist-packages/autogluon/features/generators/datetime.py:59: FutureWarning: casting datetime64[ns,
  good_rows = series[~series.isin(bad_rows)].astype(np.int64)
        Stage 4 Generators:
                Fitting DropUniqueFeatureGenerator...
        Types of features in original data (raw dtype, special dtypes):
                ('datetime', []) : 1 | ['datetime']
                ('float', [])    : 3 | ['temp', 'atemp', 'windspeed']
                ('int', [])      : 5 | ['season', 'holiday', 'workingday', 'weather', 'humidity']
        Types of features in processed data (raw dtype, special dtypes):
                ('float', [])                : 3 | ['temp', 'atemp', 'windspeed']
                ('int', [])                  : 3 | ['season', 'weather', 'humidity']
                ('int', ['bool'])            : 2 | ['holiday', 'workingday']
                ('int', ['datetime_as_int']) : 5 | ['datetime', 'datetime.year', 'datetime.month', 'datetime.day', 'datetim
        0.5s = Fit runtime
        9 features in original data used to generate 13 features in processed data.
        Train Data (Processed) Memory Usage: 0.98 MB (0.0% of available memory)
Data preprocessing and feature engineering runtime = 0.52s ...
AutoGluon will gauge predictive performance using evaluation metric: 'root_mean_squared_error'
        This metric's sign has been flipped to adhere to being higher_is_better. The metric score can be multiplied by -1 t
        To change this, specify the eval_metric parameter of Predictor()
AutoGluon will fit 2 stack levels (L1 to L2)
```

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

```
predictor.fit_summary()
```

```
4                4.364693         71.353436           2       True
5                0.047710          0.045261           1       True
6                0.001361          0.871554           2       True
7                0.045886          0.043096           1       True
8                0.642890         12.323581           1       True
9                0.637248          6.103706           1       True
10               0.142637        199.241410           1       True
11               1.224466         33.795891           1       True
12               9.212330         71.632729           1       True
13               0.565181         59.961397           1       True

    fit_order
0          14
1          12
2          11
3          13
4          10
5           2
6           9
7           1
8           5
9           7
10          6
11          4
12          3
```

Create predictions from test dataset

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

```
0    23.358217
1    41.972752
2    45.887352
3    49.803349
4    51.955547
Name: count, 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.describe()
```

```
count    6493.000000
mean      100.752563
std        89.666618
min         3.069366
25%        20.069902
50%        64.100052
75%       167.219009
max       366.163910
Name: count, dtype: float64
```

```
# How many negative values do we have?
sum(n < 0 for n in predictions.values.flatten())
```

```
0
```

```
# Set them to zero
predictions[predictions < 0] = 0
```

Set predictions to submission dataframe, save, and submit

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

```
!kaggle competitions submit -c bike-sharing-demand -f submission.csv -m "first raw submission"
```

```
100% 188k/188k [00:00<00:00, 206kB/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 6
```

```
fileName        date                 description           status    publicScore  privateScore
--------------  -------------------  --------------------  --------  -----------  ------------
submission.csv  2022-12-22 07:02:43  first raw submission  complete  1.80014      1.80014
```

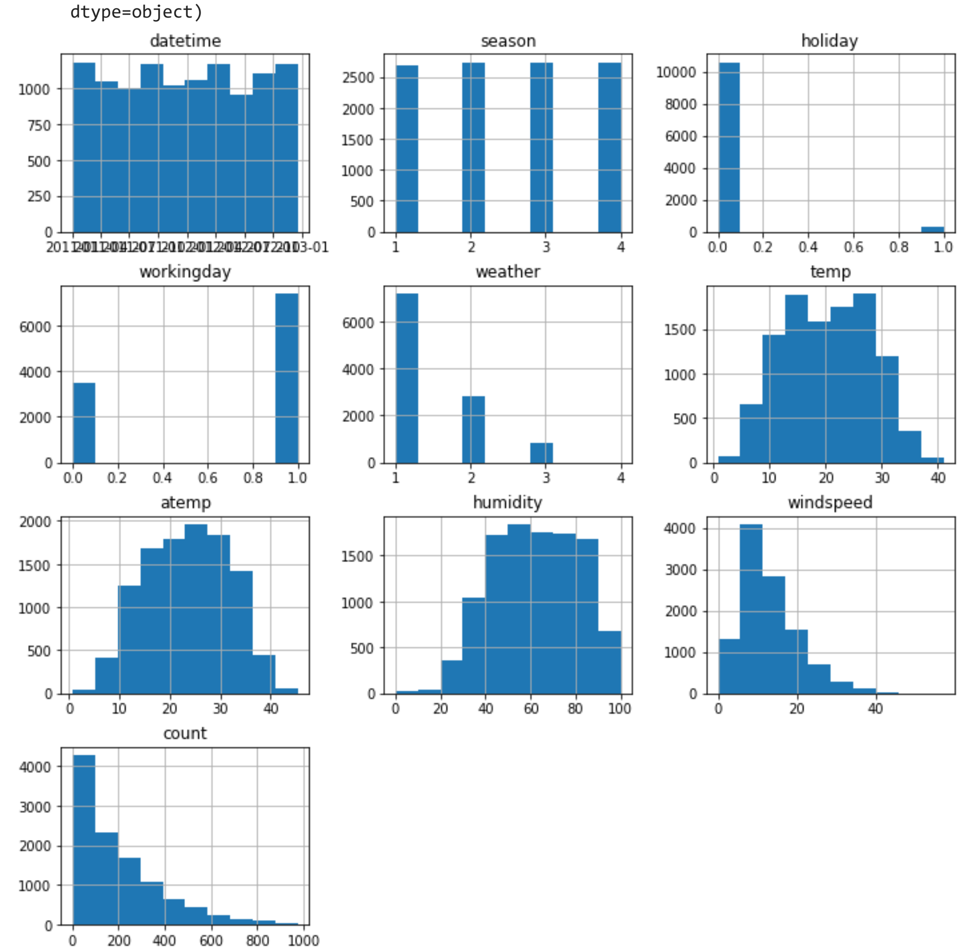
Initial score of `?`

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 data analysis
train.hist(figsize=(12,12))
```

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f3132f49610>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f3132f17820>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f3132ec6c40>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f3132ef40d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f3132ead460>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f3132e59790>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f3132e59880>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f3132e09cd0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f3132def4c0>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f3132d998b0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f3132d47c10>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f3132d741f0>]],
      dtype=object)
```




```
# create a new feature
train["day"] = train["datetime"].dt.day
test["day"] = test["datetime"].dt.day
```

```
train["month"] = train["datetime"].dt.month
test["month"] = test["datetime"].dt.month

train["year"] = train["datetime"].dt.year
test["year"] = test["datetime"].dt.year

train["hour"] = train["datetime"].dt.hour
test["hour"] = test["datetime"].dt.hour
```

```
train.tail()
```

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	count	day	month	year	hour
10881	2012-12-19 19:00:00	4	0	1	1	15.58	19.695	50	26.0027	336	19	12	2012	19
10882	2012-12-19 20:00:00	4	0	1	1	14.76	17.425	57	15.0013	241	19	12	2012	20
10883	2012-12-19 21:00:00	4	0	1	1	13.94	15.910	61	15.0013	168	19	12	2012	21
10884	2012-12-19 22:00:00	4	0	1	1	13.94	17.425	61	6.0032	129	19	12	2012	22
10885	2012-12-19 23:00:00	4	0	1	1	13.12	16.665	66	8.9981	88	19	12	2012	23

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.

```
train["season"] = train["season"].astype('category')
train["weather"] = train["weather"].astype('category')
test["season"] = test["season"].astype('category')
test["weather"] = test["weather"].astype('category')
```

```
train.info()
```

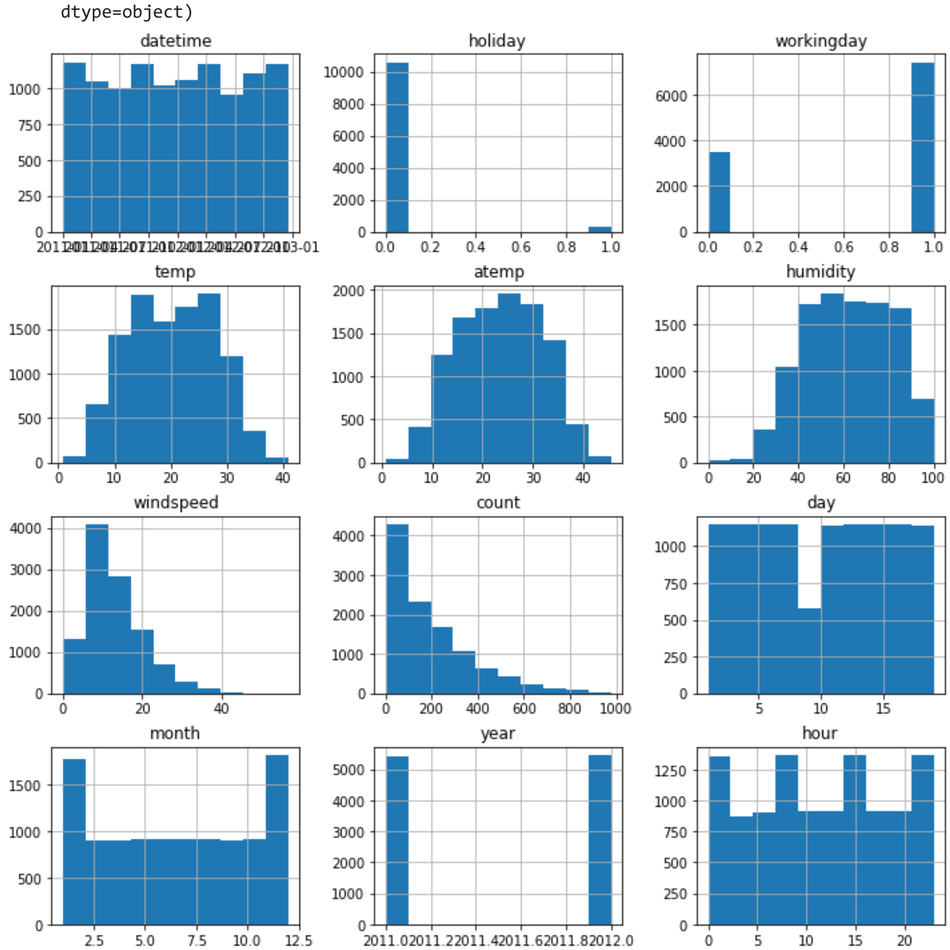
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10886 entries, 0 to 10885
Data columns (total 14 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   datetime    10886 non-null  datetime64[ns]
 1   season      10886 non-null  category
 2   holiday     10886 non-null  int64
 3   workingday  10886 non-null  int64
 4   weather     10886 non-null  category
 5   temp        10886 non-null  float64
 6   atemp       10886 non-null  float64
 7   humidity    10886 non-null  int64
 8   windspeed   10886 non-null  float64
 9   count       10886 non-null  int64
 10  day         10886 non-null  int64
 11  month       10886 non-null  int64
 12  year        10886 non-null  int64
 13  hour        10886 non-null  int64
dtypes: category(2), datetime64[ns](1), float64(3), int64(8)
memory usage: 1.0 MB
```

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

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	count	day	month	year	hour
10881	2012-12-19 19:00:00	4	0	1	1	15.58	19.695	50	26.0027	336	19	12	2012	19
10882	2012-12-19 20:00:00	4	0	1	1	14.76	17.425	57	15.0013	241	19	12	2012	20
10883	2012-12-19 21:00:00	4	0	1	1	13.94	15.910	61	15.0013	168	19	12	2012	21
10884	2012-12-19 22:00:00	4	0	1	1	13.94	17.425	61	6.0032	129	19	12	2012	22
10885	2012-12-19 23:00:00	4	0	1	1	13.12	16.665	66	8.9981	88	19	12	2012	23

```
# View histogram of all features again now with the hour feature
train.hist(figsize=(12,12))
```

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f3132a9f130>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f31325d4820>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f313257ac40>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f31325aa0d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f313255bcd0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f3132512370>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f3132512460>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f31324bcbe0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f313249da00>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f3132455160>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f31323fe880>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f3132429fa0>]],
      dtype=object)
```

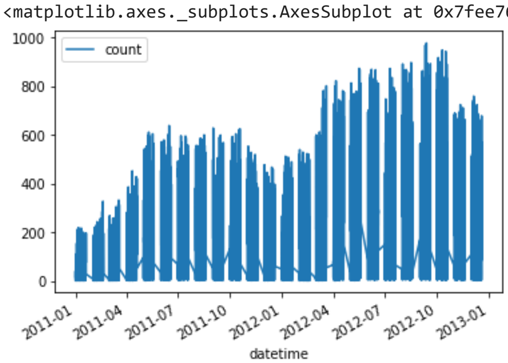



```
#correlation matrix
train.corr()
```

	holiday	workingday	temp	atemp	humidity	windspeed	count	day	month	year	hou
holiday	1.000000	-0.250491	0.000295	-0.005215	0.001929	0.008409	-0.005393	-0.015877	0.001731	0.012021	-0.00035
workingday	-0.250491	1.000000	0.029966	0.024660	-0.010880	0.013373	0.011594	0.009829	-0.003394	-0.002482	0.00278
temp	0.000295	0.029966	1.000000	0.984948	-0.064949	-0.017852	0.394454	0.015551	0.257589	0.061226	0.14543
atemp	-0.005215	0.024660	0.984948	1.000000	-0.043536	-0.057473	0.389784	0.011866	0.264173	0.058540	0.14034
humidity	0.001929	-0.010880	-0.064949	-0.043536	1.000000	-0.318607	-0.317371	-0.011335	0.204537	-0.078606	-0.27801
windspeed	0.008409	0.013373	-0.017852	-0.057473	-0.318607	1.000000	0.101369	0.036157	-0.150192	-0.015221	0.14663
count	-0.005393	0.011594	0.394454	0.389784	-0.317371	0.101369	1.000000	0.019826	0.166862	0.260403	0.40060
day	-0.015877	0.009829	0.015551	0.011866	-0.011335	0.036157	0.019826	1.000000	0.001974	0.001800	0.00113
month	0.001731	-0.003394	0.257589	0.264173	0.204537	-0.150192	0.166862	0.001974	1.000000	-0.004932	-0.00681
year	0.012021	-0.002482	0.061226	0.058540	-0.078606	-0.015221	0.260403	0.001800	-0.004932	1.000000	-0.00423
hour	-0.000354	0.002780	0.145430	0.140343	-0.278011	0.146631	0.400601	0.001132	-0.006818	-0.004234	1.00000

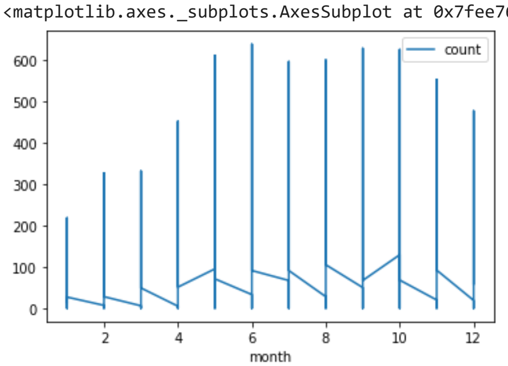
```
#bike demand evolution
train.plot("datetime","count")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fee76618ee0>
```




```
#bike demand evolution monthly 2011Y
train[train.year==2011].plot(x="month",y="count",)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fee7607a040>
```




```
#bike demand evolution monthly 2012Y
train[train.year==2012].plot(x="month",y="count")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fee7610e100>
```

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

```python
predictor_new_features = TabularPredictor(label="count", problem_type="regression", eval_metric="root_mean_squared_error").fit(
    train_data=train, time_limit=600, presets="best_quality"
)
```

```
	104.93s	 = Training   runtime
	11.42s	 = Validation runtime
Fitting model: LightGBM_BAG_L1 ... Training model for up to 288.01s of the 488.0s of remaining time.
	Fitting 8 child models (S1F1 - S1F8) | Fitting with ParallelLocalFoldFittingStrategy
	-33.9173	 = Validation score   (-root_mean_squared_error)
	49.22s	 = Training   runtime
	2.77s	 = Validation runtime
Fitting model: RandomForestMSE_BAG_L1 ... Training model for up to 232.82s of the 432.81s of remaining time.
	-38.3808	 = Validation score   (-root_mean_squared_error)
	15.3s	 = Training   runtime
	0.66s	 = Validation runtime
Fitting model: CatBoost_BAG_L1 ... Training model for up to 215.73s of the 415.71s of remaining time.
	Fitting 8 child models (S1F1 - S1F8) | Fitting with ParallelLocalFoldFittingStrategy
	-34.1429	 = Validation score   (-root_mean_squared_error)
	189.11s	 = Training   runtime
	0.21s	 = Validation runtime
Fitting model: ExtraTreesMSE_BAG_L1 ... Training model for up to 21.68s of the 221.67s of remaining time.
	-38.4827	 = Validation score   (-root_mean_squared_error)
	9.52s	 = Training   runtime
	0.64s	 = Validation runtime
Fitting model: NeuralNetFastAI_BAG_L1 ... Training model for up to 10.71s of the 210.7s of remaining time.
	Fitting 8 child models (S1F1 - S1F8) | Fitting with ParallelLocalFoldFittingStrategy
	-124.3961	 = Validation score   (-root_mean_squared_error)
	33.57s	 = Training   runtime
	0.61s	 = Validation runtime
Completed 1/20 k-fold bagging repeats ...
Fitting model: WeightedEnsemble_L2 ... Training model for up to 360.0s of the 172.05s of remaining time.
	-32.1387	 = Validation score   (-root_mean_squared_error)
	0.58s	 = Training   runtime
	0.0s	 = Validation runtime
Fitting 9 L2 models ...
Fitting model: LightGBMXT_BAG_L2 ... Training model for up to 171.43s of the 171.41s of remaining time.
	Fitting 8 child models (S1F1 - S1F8) | Fitting with ParallelLocalFoldFittingStrategy
	-31.3638	 = Validation score   (-root_mean_squared_error)
	36.0s	 = Training   runtime
	1.02s	 = Validation runtime
Fitting model: LightGBM_BAG_L2 ... Training model for up to 129.51s of the 129.49s of remaining time.
	Fitting 8 child models (S1F1 - S1F8) | Fitting with ParallelLocalFoldFittingStrategy
	-30.6728	 = Validation score   (-root_mean_squared_error)
	33.26s	 = Training   runtime
	0.36s	 = Validation runtime
Fitting model: RandomForestMSE_BAG_L2 ... Training model for up to 91.22s of the 91.2s of remaining time.
	-31.6192	 = Validation score   (-root_mean_squared_error)
	34.9s	 = Training   runtime
	0.75s	 = Validation runtime
Fitting model: CatBoost_BAG_L2 ... Training model for up to 54.34s of the 54.33s of remaining time.
	Fitting 8 child models (S1F1 - S1F8) | Fitting with ParallelLocalFoldFittingStrategy
	-30.8709	 = Validation score   (-root_mean_squared_error)
	59.62s	 = Training   runtime
	0.19s	 = Validation runtime
Completed 1/20 k-fold bagging repeats ...
Fitting model: WeightedEnsemble_L3 ... Training model for up to 360.0s of the -10.59s of remaining time.
	-30.4062	 = Validation score   (-root_mean_squared_error)
	0.36s	 = Training   runtime
	0.0s	 = Validation runtime
AutoGluon training complete, total runtime = 611.01s ... Best model: "WeightedEnsemble_L3"
TabularPredictor saved. To load, use: predictor = TabularPredictor.load("AutogluonModels/ag-20221222_145915/")
```

```python
predictor_new_features.fit_summary()
```

```
2           CatBoost_BAG_L2  -30.870852      16.601074  461.366352
3         LightGBMXT_BAG_L2  -31.363830      17.438860  437.742588
4    RandomForestMSE_BAG_L2  -31.619210      17.162649  436.643322
5       WeightedEnsemble_L2  -32.138741      15.120193  359.190915
6           LightGBM_BAG_L1  -33.917338       2.774429   49.219321
7           CatBoost_BAG_L1  -34.142905       0.214352  189.114883
8         LightGBMXT_BAG_L1  -34.387021      11.417783  104.927209
9    RandomForestMSE_BAG_L1  -38.380819       0.661006   15.299405
10     ExtraTreesMSE_BAG_L1  -38.482739       0.637595    9.521254
11    KNeighborsDist_BAG_L1  -84.125061       0.051627    0.046045
12    KNeighborsUnif_BAG_L1 -101.546199       0.049995    0.046259
13  NeuralNetFastAI_BAG_L1 -124.396121       0.607431   33.570796

    pred_time_val_marginal  fit_time_marginal  stack_level  can_infer  \
0                 0.001340           0.363033            3       True
1                 0.356639          33.257779            2       True
2                 0.186855          59.621181            2       True
3                 1.024641          35.997416            2       True
4                 0.748431          34.898150            2       True
5                 0.000996           0.584053            2       True
6                 2.774429          49.219321            1       True
7                 0.214352         189.114883            1       True
8                11.417783         104.927209            1       True
9                 0.661006          15.299405            1       True
10                0.637595           9.521254            1       True
11                0.051627           0.046045            1       True
12                0.049995           0.046259            1       True
13                0.607431          33.570796            1       True

    fit_order
0          14
1          11
2          13
3          10
4          12
5           9
6           4
7           6
8           3
9           5
10          7
11          2
12          1
```

```
# Remember to set all negative values to zero
predictions_new_features = predictor_new_features.predict(test)
predictions_new_features.head()
```

```
0    16.401878
1    11.047426
2    10.272551
3     9.231811
4     8.183758
Name: count, dtype: float32
```

```
sum(n < 0 for n in predictions_new_features.values.flatten())
```

```
0
```

```
submission_new_features = pd.read_csv("sampleSubmission.csv")
submission_new_features.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

```
# Same submitting predictions
submission_new_features["count"] = predictions_new_features
submission_new_features.to_csv("submission_new_features.csv", index=False)
```

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

```
100% 188k/188k [00:03<00:00, 60.2kB/s]
Successfully submitted to Bike Sharing Demand
```

```
!kaggle competitions submissions -c bike-sharing-demand | tail -n +1 | head -n 6
```

```
fileName                     date                 description           status    publicScore  privateScore
---------------------------  -------------------  --------------------  --------  -----------  ------------
submission_new_features.csv  2022-12-22 15:15:42  new features          complete  0.64732      0.64732
submission.csv               2022-12-22 07:02:43  first raw submission  complete  1.80014      1.80014
```

New Score of `0.64732`

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.

```
predictor_new_hpo = TabularPredictor(label="count", problem_type="regression", eval_metric="root_mean_squared_error").fit(
    train_data=train,num_gpus=1,time_limit=600, num_bag_folds=2, num_bag_sets=1, num_stack_levels=3, presets="best_quality"
)
```

```
No path specified. Models will be saved in: "AutogluonModels/ag-20221229_085223/"
Presets specified: ['best_quality']
Stack configuration (auto_stack=True): num_stack_levels=3, num_bag_folds=2, num_bag_sets=1
Beginning AutoGluon training ... Time limit = 600s
AutoGluon will save models to "AutogluonModels/ag-20221229_085223/"
AutoGluon Version:  0.6.1
Python Version:     3.8.16
Operating System:   Linux
Platform Machine:   x86_64
Platform Version:   #1 SMP Fri Aug 26 08:44:51 UTC 2022
Train Data Rows:    10886
Train Data Columns: 13
Label Column: count
Preprocessing data ...
Using Feature Generators to preprocess the data ...
Fitting AutoMLPipelineFeatureGenerator...
        Available Memory:                    11654.42 MB
        Train Data (Original)  Memory Usage: 0.98 MB (0.0% of available memory)
        Inferring data type of each feature based on column values. Set feature_metadata_in to manually specify special dtypes of th
        Stage 1 Generators:
                Fitting AsTypeFeatureGenerator...
                        Note: Converting 3 features to boolean dtype as they only contain 2 unique values.
        Stage 2 Generators:
                Fitting FillNaFeatureGenerator...
        Stage 3 Generators:
                Fitting IdentityFeatureGenerator...
                Fitting CategoryFeatureGenerator...
                        Fitting CategoryMemoryMinimizeFeatureGenerator...
                Fitting DatetimeFeatureGenerator...
/usr/local/lib/python3.8/dist-packages/autogluon/features/generators/datetime.py:59: FutureWarning: casting datetime64[ns, UTC] valu
  good_rows = series[~series.isin(bad_rows)].astype(np.int64)
        Stage 4 Generators:
                Fitting DropUniqueFeatureGenerator...
        Types of features in original data (raw dtype, special dtypes):
                ('category', []) : 2 | ['season', 'weather']
                ('datetime', []) : 1 | ['datetime']
                ('float', [])    : 3 | ['temp', 'atemp', 'windspeed']
                ('int', [])      : 7 | ['holiday', 'workingday', 'humidity', 'day', 'month', ...]
        Types of features in processed data (raw dtype, special dtypes):
                ('category', [])             : 2 | ['season', 'weather']
                ('float', [])                : 3 | ['temp', 'atemp', 'windspeed']
                ('int', [])                  : 4 | ['humidity', 'day', 'month', 'hour']
                ('int', ['bool'])            : 3 | ['holiday', 'workingday', 'year']
                ('int', ['datetime_as_int']) : 5 | ['datetime', 'datetime.year', 'datetime.month', 'datetime.day', 'datetime.dayofwe
        0.3s = Fit runtime
        13 features in original data used to generate 17 features in processed data.
        Train Data (Processed) Memory Usage: 1.1 MB (0.0% of available memory)
Data preprocessing and feature engineering runtime = 0.38s ...
AutoGluon will gauge predictive performance using evaluation metric: 'root_mean_squared_error'
        This metric's sign has been flipped to adhere to being higher_is_better. The metric score can be multiplied by -1 to get the
        To change this, specify the eval_metric parameter of Predictor()
AutoGluon will fit 4 stack levels (L1 to L4) ...
```

```
Fitting 11 L1 models ...
Fitting model: KNeighborsUnif_BAG_L1 ... Training model for up to 199.82s of the 599.6s of remaining time.
	-101.5462	 = Validation score   (-root mean squared error)
```

```
predictor_new_hpo.fit_summary()
```

```
1              0.001015          0.695664            4       True
2              0.288916          3.038737            2       True
3              0.034459         21.748844            3       True
4              0.631673          8.818562            3       True
5              0.000981          0.311804            5       True
6              0.045899         13.427199            4       True
7              0.433297         30.904074            3       True
8              1.866063          7.733432            2       True
9              0.654379         28.923467            3       True
10             0.067509          2.501557            3       True
11             0.074822         77.438849            2       True
12             0.632092          9.416392            4       True
13             0.097694          4.383719            3       True
14             0.645207          8.725624            2       True
15             0.652325         30.429245            4       True
16             0.150445          2.441851            3       True
17             0.074505          2.215205            4       True
18             0.658314         26.941700            2       True
19             0.154596          2.332428            4       True
20             0.000949          0.498457            2       True
21             1.981085          4.695249            1       True
22             0.601768         13.562908            1       True
23             4.715242         12.244923            1       True
24             0.097511        188.636190            1       True
25             0.037318          0.039276            1       True
26             0.046529          0.065238            1       True
27             0.231332         14.072533            2       True

    fit_order
0          14
1          22
2           9
3          18
4          19
5          28
6          26
7          20
8           8
9          17
10         16
11         11
12         27
13         21
14         12
15         25
16         15
17         24
18         10
19         23
20          7
21          4
22          5
23          3
24          6
25          2
26          1
27         13  }
```

```
predictions_hpo = predictor_new_hpo.predict(test)
predictions_hpo.head()
```

```
0    17.045513
1    11.393410
2     9.331726
3     7.929997
4     7.117903
Name: count, dtype: float32
```

```
submission_hpo = pd.read_csv("sampleSubmission.csv")
submission_hpo.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

```
# Remember to set all negative values to zero
sum(n < 0 for n in predictions_hpo.values.flatten())
```

```
0
```

```
predictions_hpo[predictions_hpo < 0] = 0
```

```
sum(n < 0 for n in predictions_hpo.values.flatten())
```

```
0
```

```
# Same submitting predictions
submission_hpo["count"] = predictions_hpo
submission_hpo.to_csv("submission_new_hpo.csv", index=False)
```

```
:ions submit -c bike-sharing-demand -f submission_new_hpo.csv -m "new features with hyperparameters(1,1,3)"
```

```
100% 188k/188k [00:02<00:00, 76.9kB/s]
Successfully submitted to Bike Sharing Demand
```

```
!kaggle competitions submissions -c bike-sharing-demand
```

```
fileName                     date                 description                               status    publicScore  privateScore
---------------------------  -------------------  ----------------------------------------  --------  -----------  ------------
submission_new_hpo.csv       2022-12-29 09:05:14  new features with hyperparameters(1,1,3)  complete  0.64038      0.64038
submission_new_hpo.csv       2022-12-29 08:50:02  new features with hyperparameters(3,1,3)  complete  0.62255      0.62255
submission_new_hpo.csv       2022-12-28 06:40:30  new features with hyperparameters(9,5,3)  complete  0.74931      0.74931
submission_new_hpo.csv       2022-12-28 06:39:11  new features with hyperparameters(5,1,1)  complete  0.74931      0.74931
submission_new_hpo.csv       2022-12-23 16:03:13  new features with hyperparameters(5,1,1)  complete  0.68468      0.68468
submission_new_hpo.csv       2022-12-23 15:37:43  new features with hyperparameters(7,1,3)  complete  0.68365      0.68365
submission_new_hpo.csv       2022-12-23 15:23:54  new features with hyperparameters(5,1,3)  complete  0.67858      0.67858
submission_new_hpo.csv       2022-12-23 14:01:12  new features with hyperparameters         complete  0.79443      0.79443
submission_new_features.csv  2022-12-22 15:15:42  new features                              complete  0.64732      0.64732
submission.csv               2022-12-22 07:02:43  first raw submission                      complete  1.80014      1.80014
```

New Score of `0.62255`

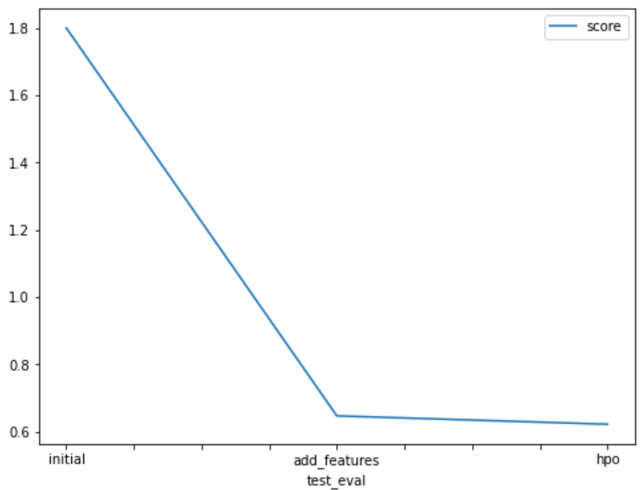
Step 7: Write a Report

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", "add_features", "hpo"],
        "score": [1.80014, 0.64732, 0.62255]
    }
).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", "add_features", "hpo"],
        "score": [1.80014, 0.64732, 0.62255]
    }
).plot(x="test_eval", y="score", figsize=(8, 6)).get_figure()
fig.savefig('model_test_score.png')
```



Hyperparameter table

```
# The 3 hyperparameters we tuned with the kaggle score as the result
pd.DataFrame({
    "model": ["initial", "add_features", "hpo"],
    "model_used" : "WeightedEnsemble_L3",
    "num_bag_folds": [5, 1, 3],
    "num_bag_sets": [1, 1, 1],
    "num_stack_levels": [3, 3, 3],
    "score": [0.67858, 0.64037, 0.62255]
})
```

	model	model_used	num_bag_folds	num_bag_sets	num_stack_levels	score
0	initial	WeightedEnsemble_L3	5	1	3	0.67858
1	add_features	WeightedEnsemble_L3	1	1	3	0.64037
2	hpo	WeightedEnsemble_L3	3	1	3	0.62255