

```
In [1]: import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from scipy.stats import norm
import datetime
import re
import warnings
from datetime import datetime
from sklearn.linear_model import ElasticNetCV, LassoCV, RidgeCV
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.svm import SVR
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import RobustScaler
from sklearn.model_selection import KFold, cross_val_score
from sklearn.metrics import mean_squared_error, mean_absolute_error
from mlxtend.regressor import StackingCVRegressor
from xgboost import XGBRegressor
from lightgbm import LGBMRegressor
import scipy.stats as stats
import sklearn.linear_model as linear_model
from sklearn.manifold import TSNE
from sklearn.cluster import KMeans
import xgboost as xgb
from sklearn.ensemble import AdaBoostRegressor, BaggingRegressor, ExtraTreesRegressor
from xgboost import plot_importance, plot_tree
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score, KFold, GridSearchCV, TimeSeriesSplit
from sklearn.metrics import mean_absolute_error

plt.rcParams['axes.labelsize'] = 14
plt.rcParams['xtick.labelsize'] = 12
plt.rcParams['ytick.labelsize'] = 12
plt.rcParams['text.color'] = 'k'

warnings.filterwarnings("ignore")
plt.style.use('fivethirtyeight')
```

```
In [2]: #reducing the memory usage before using the dataframes for plotting and modelling
def reduce_mem_usage(df, verbose=True):
    numerics = ['int16', 'int32', 'int64', 'float16', 'float32', 'float64']
    start_mem = df.memory_usage().sum() / 1024**2
    for col in df.columns:
        col_type = df[col].dtypes
        if col_type in numerics:
            c_min = df[col].min()
            c_max = df[col].max()
            if str(col_type)[:3] == 'int':
                if c_min > np.iinfo(np.int8).min and c_max < np.iinfo(np.int8).max:
                    df[col] = df[col].astype(np.int8)
                elif c_min > np.iinfo(np.int16).min and c_max < np.iinfo(np.int16).max:
                    df[col] = df[col].astype(np.int16)
                elif c_min > np.iinfo(np.int32).min and c_max < np.iinfo(np.int32).max:
                    df[col] = df[col].astype(np.int32)
                elif c_min > np.iinfo(np.int64).min and c_max < np.iinfo(np.int64).max:
                    df[col] = df[col].astype(np.int64)
                else:
                    if c_min > np.finfo(np.float16).min and c_max < np.finfo(np.float16).max:
                        df[col] = df[col].astype(np.float16)
                    elif c_min > np.finfo(np.float32).min and c_max < np.finfo(np.float32).max:
                        df[col] = df[col].astype(np.float32)
                    else:
                        df[col] = df[col].astype(np.float64)
            end_mem = df.memory_usage().sum() / 1024**2
            if verbose:
                print('Mem. usage decreased to {:.2f} Mb ({:.1f}% reduction)'.format(end_mem, 100 * (start_mem - end_mem) / start_mem))
    return df
```

```
In [6]: df_weather = pd.read_csv('homeF2016.csv')
df_meter = pd.read_csv('HomeF-meter3_2016.csv', parse_dates=[0], index_col=[0])
```

```
In [7]: df_weather = reduce_mem_usage(df_weather)
df_meter = reduce_mem_usage(df_meter)
```

Mem. usage decreased to 0.35 Mb (62.5% reduction)
 Mem. usage decreased to 12.50 Mb (67.5% reduction)

```
In [8]: df_weather.shape
```

Out[8]: (8760, 14)

In [10]: df_weather.head()

Out[10]:

| | temperature | icon | humidity | visibility | summary | apparentTemperature | pressure | windSpeed |
|---|-------------|---------------------|----------|------------|---------------|---------------------|----------|-----------|
| 0 | 36.53125 | clear-night | 0.609863 | 10.0 | Clear | 29.765625 | 1017.0 | 9.10937 |
| 1 | 36.31250 | partly-cloudy-night | 0.600098 | 10.0 | Mostly Cloudy | 29.593750 | 1016.0 | 8.92968 |
| 2 | 35.78125 | clear-night | 0.609863 | 10.0 | Clear | 29.484375 | 1016.0 | 7.89843 |
| 3 | 35.62500 | clear-night | 0.629883 | 10.0 | Clear | 31.046875 | 1016.0 | 5.35156 |
| 4 | 35.96875 | clear-night | 0.620117 | 10.0 | Clear | 30.406250 | 1016.0 | 6.80078 |

◀ ▶

In [9]: df_meter.shape

Out[9]: (503925, 9)

In [11]: df_meter = df_meter.resample('H').mean()

In [12]: df_meter.shape

Out[12]: (8399, 9)

In [13]: df_meter.tail()

Out[13]:

| Date & Time | Usage [kW] | Generation [kW] | Net_Meter [kW] | Volt [kW] | Garage_E [kW] | Garage_W [kW] | Phase_A [kW] | Phase_B [kW] |
|---------------------|------------|-----------------|----------------|-----------|---------------|---------------|--------------|--------------|
| 2016-12-15 18:00:00 | 5.214844 | 0.005379 | 5.222656 | 3.679688 | 0.003836 | 0.002710 | 2.882812 | 2.333984 |
| 2016-12-15 19:00:00 | 2.765625 | 0.007347 | 2.773438 | 1.070312 | 0.001105 | 0.000967 | 1.432617 | 1.332031 |
| 2016-12-15 20:00:00 | 1.877930 | 0.007507 | 1.885742 | 0.002180 | 0.000003 | 0.000354 | 1.171875 | 0.706055 |
| 2016-12-15 21:00:00 | 1.246094 | 0.008385 | 1.254883 | 0.002192 | 0.000003 | 0.000355 | 0.642090 | 0.604004 |
| 2016-12-15 22:00:00 | 0.941406 | 0.008804 | 0.950195 | 0.002197 | 0.000003 | 0.000354 | 0.465820 | 0.475342 |

◀ ▶

Data Preprocessing and EDA

Weather data exists from Jan 1, 2016 to Dec 31, 2016 on an hourly basis. Meter data exists from Jan 1, 2014 to Dec 15, 2016 on per minute basis. Icon and Summary columns from weather data are useless and hence discarded. Net_Meter [kW] column contains almost similar data to that of Usage[kW] features and hence discarded. Also, weather data is converted to EST and both dataframes are indexed on date and time column. To combine both the data into a single dataframe, the meter dataframe is resampled on hourly basis and then both of them are merged to get a combined dataframe.

```
In [14]: df_weather = df_weather.drop(['icon', 'summary'], axis=1)
df_weather['Date & Time'] = pd.to_datetime(df_weather['time'], unit='s')
df_weather['Date & Time'] = df_weather['Date & Time'] - pd.Timedelta(hours=5)
df_weather.index = df_weather['Date & Time']
df_weather = df_weather.drop(['Date & Time', 'time'], axis=1)
```

```
In [15]: df_weather.head()
```

Out[15]:

| | temperature | humidity | visibility | apparentTemperature | pressure | windSpeed | cloudCover |
|---------------------|-------------|----------|------------|---------------------|----------|-----------|------------|
| Date & Time | | | | | | | |
| 2016-01-01 00:00:00 | 36.53125 | 0.609863 | 10.0 | 29.765625 | 1017.0 | 9.109375 | Nan |
| 2016-01-01 01:00:00 | 36.31250 | 0.600098 | 10.0 | 29.593750 | 1016.0 | 8.929688 | 0.7 |
| 2016-01-01 02:00:00 | 35.78125 | 0.609863 | 10.0 | 29.484375 | 1016.0 | 7.898438 | Nan |
| 2016-01-01 03:00:00 | 35.62500 | 0.629883 | 10.0 | 31.046875 | 1016.0 | 5.351562 | Nan |
| 2016-01-01 04:00:00 | 35.96875 | 0.620117 | 10.0 | 30.406250 | 1016.0 | 6.800781 | Nan |

```
In [16]: df_meter_hourly = df_meter
df_weather_hourly = df_weather
df_meter_daily = df_meter_hourly.resample('D').mean()
df_weather_daily = df_weather_hourly.resample('D').mean()
```

```
In [17]: df_meter_hourly.isna().sum()
```

```
Out[17]: Usage          1  
Generation      1  
Net_Meter       1  
Volt            1  
Garage_E        1  
Garage_W        1  
Phase_A         1  
Phase_B         1  
Solar           1  
dtype: int64
```

```
In [19]: df_meter_hourly = df_meter_hourly.dropna()
```

```
In [20]: df_weather_hourly.isna().sum()
```

```
Out[20]: temperature      0  
humidity          0  
visibility         0  
apparentTemperature 0  
pressure          0  
windSpeed          0  
cloudCover        960  
windBearing        0  
precipIntensity    0  
dewPoint           0  
precipProbability   0  
dtype: int64
```

```
In [21]: df_weather_hourly = df_weather_hourly.fillna(0)
```

```
In [22]: df_weather_daily.isna().sum()
```

```
Out[22]: temperature      0  
humidity          0  
visibility         0  
apparentTemperature 0  
pressure          0  
windSpeed          0  
cloudCover        0  
windBearing        0  
precipIntensity    0  
dewPoint           0  
precipProbability   0  
dtype: int64
```

```
In [23]: df_meter_daily.isna().sum()
```

```
Out[23]: Usage [kW]      0
Generation [kW]      0
Net_Meter [kW]       0
Volt [kW]            0
Garage_E [kW]        0
Garage_W [kW]        0
Phase_A [kW]          0
Phase_B [kW]          0
Solar [kW]           0
dtype: int64
```

```
In [24]: comb_df_hourly = pd.merge(df_meter_hourly, df_weather_hourly, left_index=True,
right_index =True, how='left')
comb_df_daily = pd.merge(df_meter_daily, df_weather_daily, left_index=True, ri
ght_index =True, how='left')
```

```
In [25]: comb_df_hourly.shape
```

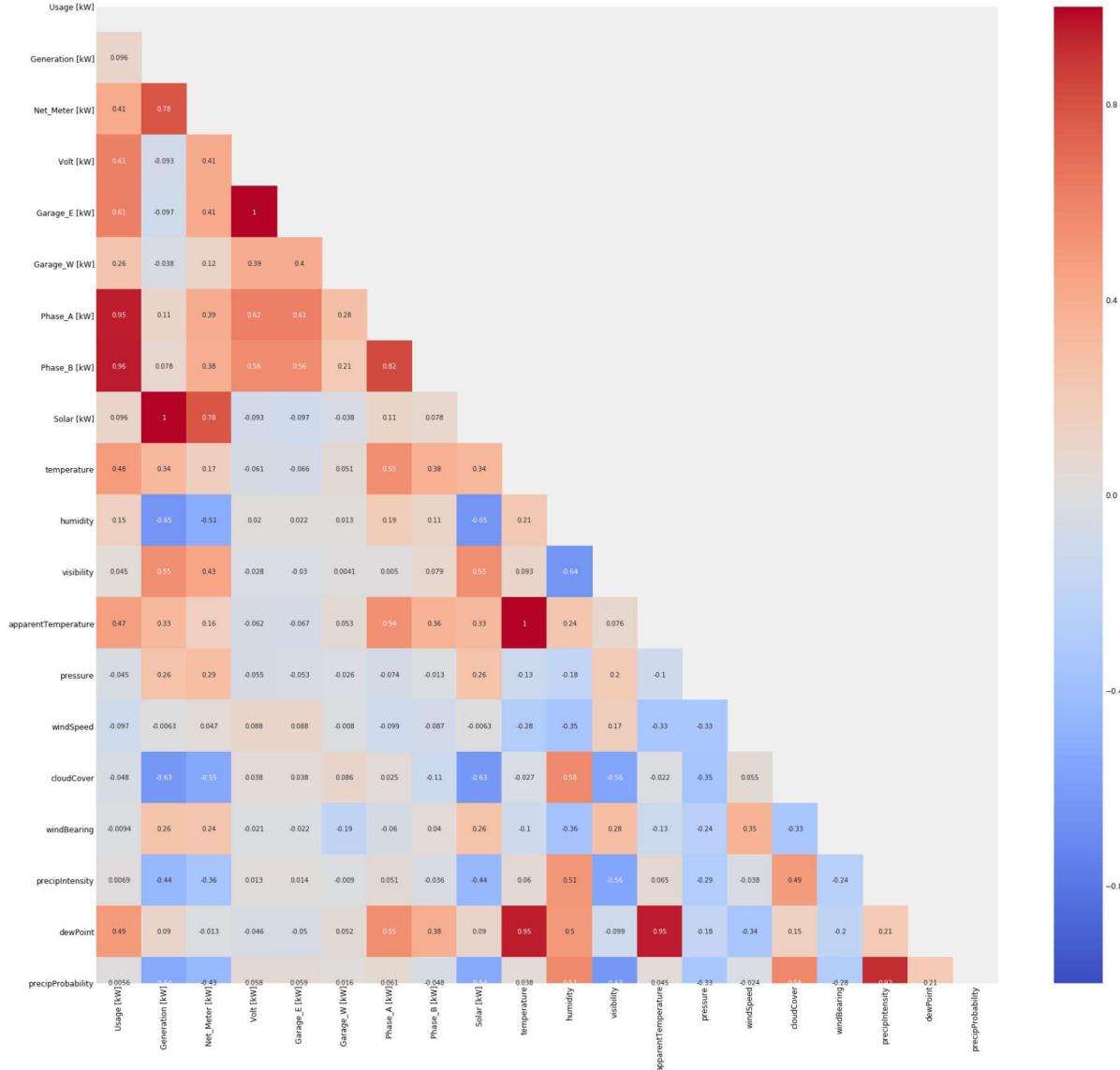
```
Out[25]: (8398, 20)
```

```
In [26]: comb_df_daily.shape
```

```
Out[26]: (350, 20)
```

```
In [27]: corr = comb_df_daily.corr()
mask = np.zeros_like(corr, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True
plt.figure(figsize=(28,28))
sns.heatmap(corr, xticklabels=corr.columns, yticklabels=corr.columns, cmap='coolwarm', annot=True, mask = mask, vmin=-1)
```

Out[27]: <matplotlib.axes._subplots.AxesSubplot at 0x7f1d0eba2090>



```
In [28]: #dropping apparentTemperature as it almost conveys the same info as temperature
#e. precipProbability has very high correlation
#with precipIntensity and hence dropped. Generation feature has correlation of
#1 with Solar and probably conveys the same
#info of solar electricity generated and hence discarded. Phase_a & Phase_B ar
e nothing but breakdown of total consumption
#and hence discarded.
comb_df_daily = comb_df_daily.drop(['apparentTemperature', 'precipProbability',
                                     'Generation [kW]', 'Phase_A [kW]', 'Phase_B [kW]'], axis=1)
```

```
In [29]: #taking log for pressure values
comb_df_daily['pressure'] = comb_df_daily['pressure'].apply(np.log)

def remove_braces(df):
    regex = re.compile(r"\[\|\]|<", re.IGNORECASE)
    df.columns = [regex.sub("_", col) if any(x in str(col) for x in set(['[', ']', '<'])) else col for col in df.columns]
    return df

#remving braces from feature names as XGBoost doesn't allow braces
comb_df_daily = remove_braces(comb_df_daily)
```

```
In [30]: comb_df_daily.shape
```

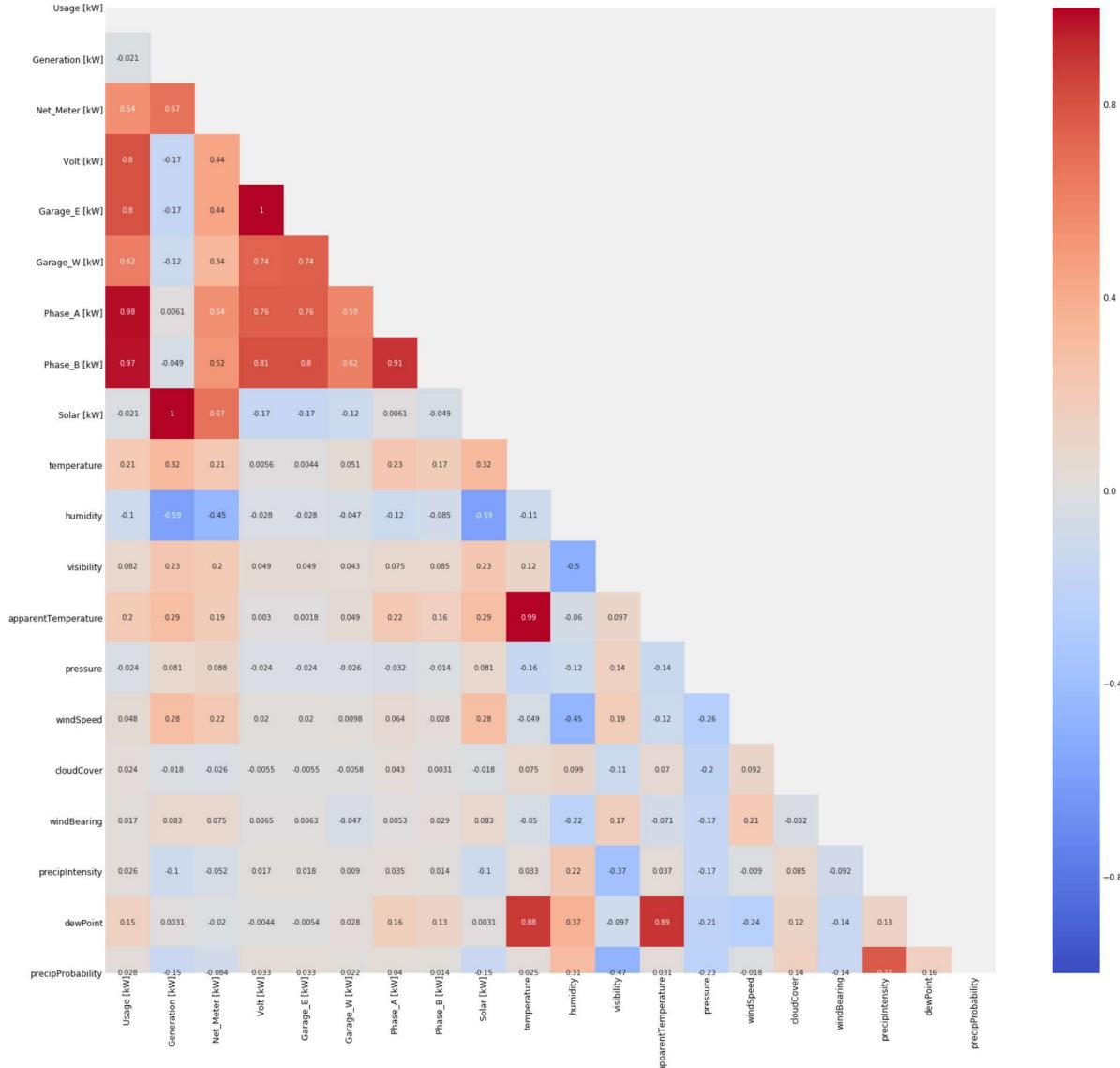
```
Out[30]: (350, 15)
```

```
In [31]: comb_df_daily.isna().sum()
```

```
Out[31]: Usage_kW_          0
Net_Meter_kw_           0
Volt_kw_                0
Garage_E_kw_             0
Garage_W_kw_             0
Solar_kw_                0
temperature              0
humidity                 0
visibility                0
pressure                  0
windSpeed                 0
cloudCover                0
windBearing                0
precipIntensity            0
dewPoint                  0
dtype: int64
```

```
In [32]: corr = comb_df_hourly.corr()
mask = np.zeros_like(corr, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True
plt.figure(figsize=(25,25))
sns.heatmap(corr, xticklabels=corr.columns, yticklabels=corr.columns, cmap='coolwarm', annot=True, mask = mask, vmin=-1)
```

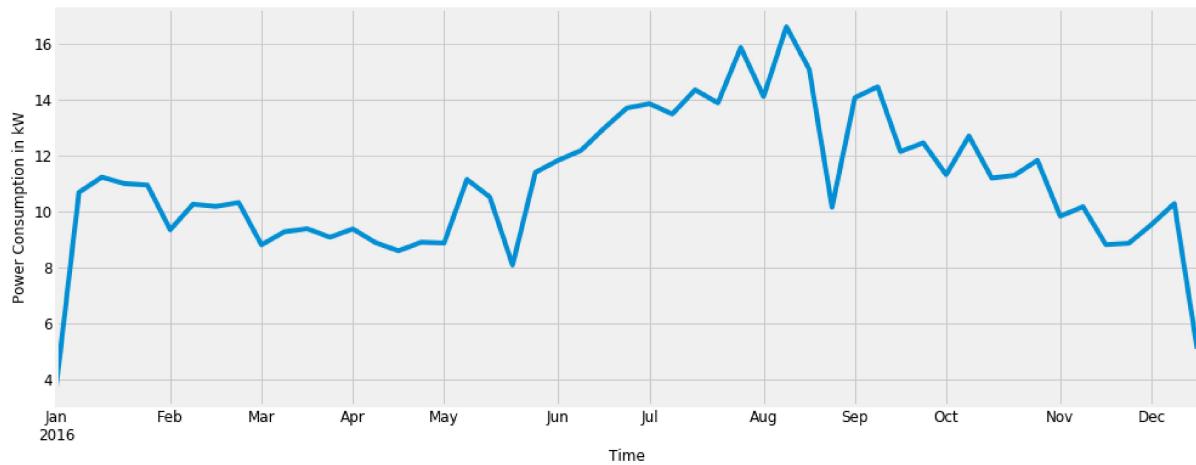
Out[32]: <matplotlib.axes._subplots.AxesSubplot at 0x7f1d11f17750>



```
In [33]: #dropping apparentTemperature as it almost conveys the same info as temperature. precipProbability has very high correlation
#with precipIntensity and hence dropped. Generation feature has correlation of 1 with Solar and probably conveys the same
#info of solar electricity generated and hence discarded. Phase_a & Phase_B are nothing but breakdown of total consumption
#and hence discarded.
comb_df_hourly = comb_df_hourly.drop(['apparentTemperature', 'precipProbability',
                                         'Generation [kW]', 'Phase_A [kW]', 'Phase_B [kW]'], axis=1)
comb_df_hourly['pressure'] = comb_df_hourly['pressure'].apply(np.log)
comb_df_hourly = remove_braces(comb_df_hourly)
comb_df_hourly.shape
```

Out[33]: (8398, 15)

```
In [35]: weekly_summary = pd.DataFrame()
weekly_summary['use [kW]'] = comb_df_daily['Usage_kw'].resample('W').sum()
ax = weekly_summary['use [kW]'].plot(figsize=(15, 6))
ax.set_xlabel('Time')
ax.set_ylabel('Power Consumption in kW')
plt.show()
```



The electricity consumption for House F is as shown above. It is highly irregular trended and we see a sharp rise in the usage during January. Consumption reduces between May-June and again rises from June till August. Thereafter, it goes on decreasing.

Feature Engineering

Hourly Data

We add lagged features for weather related data in the dataframe so that our models can learn more from the data. All weather related features for last 7 samples are added as new features for each sample in the hourly dataframe.

```
In [36]: # Build Lagged weather predictors
lagged_df = comb_df_hourly.copy()

# Next hour's load values.
lagged_df['load_hour_after'] = lagged_df['Usage _kW_'].shift(-1)

for day in range(8):
    lagged_df['temperature_d' + str(day)] = lagged_df.temperature.shift(day)
    lagged_df['windSpeed_d' + str(day)] = lagged_df.windSpeed.shift(day)
    lagged_df['humidity_d' + str(day)] = lagged_df.humidity.shift(day)
    lagged_df['pressure_d' + str(day)] = lagged_df.pressure.shift(day)
    lagged_df['cloudCover_d' + str(day)] = lagged_df.cloudCover.shift(day)
    lagged_df['windBearing_d' + str(day)] = lagged_df.windBearing.shift(day)
    lagged_df['precipIntensity_d' + str(day)] = lagged_df.precipIntensity.shift(day)
    lagged_df['dewPoint_d' + str(day)] = lagged_df.dewPoint.shift(day)
    lagged_df['load_d' + str(day)] = lagged_df['Usage _kW_'].shift(day)

lagged_df = lagged_df.dropna()

lagged_df = lagged_df.drop(columns=['temperature', 'windSpeed', 'humidity', 'pressure', 'cloudCover',
                                     'windBearing', 'precipIntensity', 'dewPoint', 'Usage _kW_'])
```

```
In [37]: X = lagged_df.drop(columns=['load_hour_after'])
y = lagged_df['load_hour_after']
```

```
In [38]: X.shape
```

```
Out[38]: (8390, 78)
```

```
In [39]: y.shape
```

```
Out[39]: (8390,)
```

Data Modelling

We split the data into a time series split with number of splits as 10. This is used to evaluate cross-validation mean absolute errors for models. We also split the data in the ratio of 80:20 for training and testing respectively. Out of this 80% data, all data points are also used for cross-validation. We train 10 models using this test data and also calculate their cross-validation MAE. We also create our own Blender model that weighs the prediction of each of these models to give a single prediction.

```
In [40]: #This baseline model simply takes 80% of data as train data and uses it to predict on remaining training data using data
#of previous 2 samples for each sample in the train set
def baseline_model(series):
    X = series.values
    X = X.astype('float32')
    train_size = int(len(X) * 0.80)
    train, test = X[0:train_size], X[train_size:]
    # walk-forward validation
    history = [x for x in train]
    predictions = []
    for i in range(len(test)):
        # predict
        yhat = history[-2]
        predictions.append(yhat)
        # observation
        obs = test[i]
        history.append(obs)

    # report performance
    mae = mean_absolute_error(test, predictions)
    print('MAE of baseline model: %.3f' % mae)
    return mae
```

```
In [41]: tscv = TimeSeriesSplit(n_splits=10)

def cv_mae(model, X, y):
    mae = -cross_val_score(model, X, y, scoring="neg_mean_absolute_error", cv=tscv)
    return (mae)

def mae(y, y_pred):
    return mean_absolute_error(y, y_pred)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=False)
```

```
In [45]: print('Building model...')

alphas_alt = [14.5, 14.6, 14.7, 14.8, 14.9, 15, 15.1, 15.2, 15.3, 15.4, 15.5]
alphas2 = [5e-05, 0.0001, 0.0002, 0.0003, 0.0004, 0.0005, 0.0006, 0.0007, 0.0008]
e_alphas = [0.0001, 0.0002, 0.0003, 0.0004, 0.0005, 0.0006, 0.0007]
e_l1ratio = [0.8, 0.85, 0.9, 0.95, 0.99, 1]

ridge = make_pipeline(RobustScaler(), RidgeCV(alphas=alphas_alt, cv=tscv))

lasso = make_pipeline(RobustScaler(), LassoCV(max_iter=1e7, alphas=alphas2, random_state=42, cv=tscv))

elasticnet = make_pipeline(RobustScaler(), ElasticNetCV(max_iter=1e7, alphas=e_alphas, cv=tscv, l1_ratio=e_l1ratio))

svr = make_pipeline(RobustScaler(), SVR(C= 20, epsilon= 0.008, gamma=0.0004,))

gbr = GradientBoostingRegressor(n_estimators=3000, learning_rate=0.03, max_depth=4, max_features='sqrt', min_samples_leaf=20, min_samples_split=10, loss='huber', random_state =42)

lightgbm = LGBMRegressor(objective='regression',
                         num_leaves=4,
                         learning_rate=0.005,
                         n_estimators=5000,
                         max_bin=200,
                         bagging_fraction=0.75,
                         bagging_freq=5,
                         bagging_seed=7,
                         feature_fraction=0.2,
                         feature_fraction_seed=7,
                         verbose=-1,
                         )

xgboost = XGBRegressor(learning_rate=0.01,n_estimators=4060,
                      max_depth=4, min_child_weight=0,
                      gamma=0, subsample=0.7,
                      colsample_bytree=0.7,
                      objective='reg:linear', nthread=-1,
                      scale_pos_weight=1, seed=27,
                      reg_alpha=0.00005)

adaboost = AdaBoostRegressor()

extratrees = ExtraTreesRegressor()

bagging = BaggingRegressor()

stack_gen = StackingCVRegressor(regressors=(ridge, lasso, elasticnet, gbr, xgboost, lightgbm, adaboost, extratrees, bagging),
                                meta_regressor=xgboost,
                                use_features_in_secondary=True)

#Store models, scores and prediction values
models = {'Ridge': ridge,
```

```

        #'Lasso': Lasso,
        #'ElasticNet': elasticnet,
        #'SVR': svr,
        'GBR': gbr,
        'LightGBM': lightgbm,
        'XGBoost': xgboost,
        'AdaBoost': adaboost,
        #'Extratrees': extratrees,
        #'Bagging': bagging
    }
scores = {}

#Evaluating model score
print('Evaluating model cross-validation scores...')
for name, model in models.items():
    score = cv_mae(model, X_train, y_train)
    print((name + ": {:.4f} ({:.4f})\n").format(score.mean(), score.std()))
    scores[name] = (score.mean(), score.std())

print('Done evaluating')

print('Fitting models...')

#print('Fitting StackingRegressor...')
#stack_gen_model = stack_gen.fit(np.array(X_train), np.array(y_train))

#print('Fitting ElasticNet...')
#elastic_model_full_data = elasticnet.fit(X_train,y_train)

#print('Fitting Lasso...')
#lasso_model_full_data = Lasso.fit(X_train,y_train)

print('Fitting Ridge...')
ridge_model_full_data = ridge.fit(X_train,y_train)

#print('Fitting SVR...')
#svr_model_full_data = svr.fit(X_train,y_train)

print('Fitting GradientBoosting...')
gbr_model_full_data = gbr.fit(X_train,y_train)

print('Fitting XGBoost...')
xgb_model_full_data = xgboost.fit(X_train,y_train)

print('Fitting LightGBM...')
lgb_model_full_data = lightgbm.fit(X_train,y_train)

print('Fitting AdaBoost...')
adaboost_model_full_data = adaboost.fit(X_train,y_train)

#print('Fitting extratrees...')
#extratrees_model_full_data = extratrees.fit(X_train,y_train)

#print('Fitting Bagging...')
#bagging_model_full_data = bagging.fit(X_train,y_train)

print('Done fitting all models')

```

Building model...

Evaluating model cross-validation scores...

Ridge: 0.7190 (0.1196)

GBR: 0.6110 (0.1089)

LightGBM: 0.6133 (0.1042)

[04:02:10] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linea
r is now deprecated in favor of reg:squarederror.

[04:02:13] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linea
r is now deprecated in favor of reg:squarederror.

[04:02:18] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linea
r is now deprecated in favor of reg:squarederror.

[04:02:25] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linea
r is now deprecated in favor of reg:squarederror.

[04:02:35] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linea
r is now deprecated in favor of reg:squarederror.

[04:02:46] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linea
r is now deprecated in favor of reg:squarederror.

[04:02:58] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linea
r is now deprecated in favor of reg:squarederror.

[04:03:12] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linea
r is now deprecated in favor of reg:squarederror.

[04:03:29] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linea
r is now deprecated in favor of reg:squarederror.

[04:03:47] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linea
r is now deprecated in favor of reg:squarederror.

XGBoost: 0.6294 (0.1109)

AdaBoost: 1.0275 (0.1909)

Done evaluating

Fitting models...

Fitting Ridge...

Fitting GradientBoosting...

Fitting XGBoost...

[04:04:53] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linea
r is now deprecated in favor of reg:squarederror.

Fitting LightGBM...

Fitting AdaBoost...

Done fitting all models

```
In [57]: #Blending model predictions
print('Blending model predictions...')
#def blend_models_predict(X):
#    return ((0.045 * elastic_model_full_data.predict(X))+(0.5 * lasso_model_full_data.predict(X))
#            +(0.15 * ridge_model_full_data.predict(X)) + (0.1 * svr_model_full_data.predict(X))
#            +(0.15 * gbr_model_full_data.predict(X))+(0.01 * xgb_model_full_data.predict(X))
#            +(0.0005 * extratrees_model_full_data.predict(X)))
#            +(0.005 * bagging_model_full_data.predict(X))
#            +(0.001 * lgb_model_full_data.predict(X))+ (0.01 * stack_gen_model.predict(np.array(X))))
#
#def blend_models_predict(X):
#    return ((0.15 * ridge_model_full_data.predict(X))+(0.2 * gbr_model_full_data.predict(X))
#            +(0.2 * xgb_model_full_data.predict(X))+(0.5 * lgb_model_full_data.predict(X)))
#
mae_score = mae(y_train, blend_models_predict(X_train))
print('MAE score on train data:' + str(mae_score))
scores['Blender'] = (mae_score.mean(), mae_score.std())
```

Blending model predictions...
MAE score on train data:0.4450453365631982

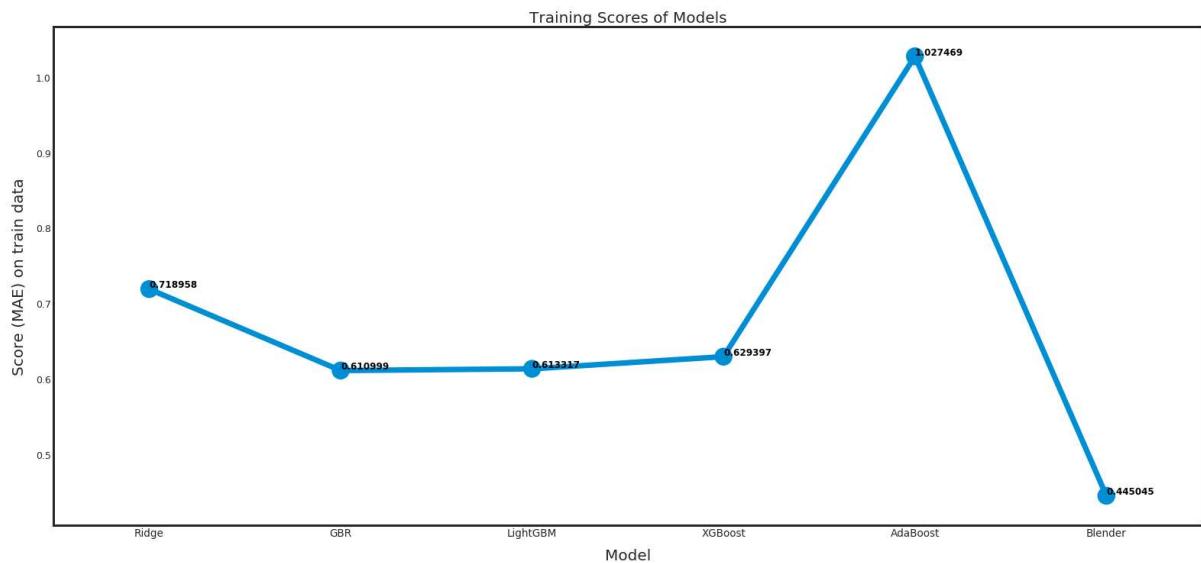
```
In [58]: sns.set_style("white")
fig = plt.figure(figsize=(24, 12))

ax = sns.pointplot(x=list(scores.keys()), y=[score for score, _ in scores.values()], markers=['o'], linestyles=['-'])
for i, score in enumerate(scores.values()):
    ax.text(i, score[0] + 0.002, '{:.6f}'.format(score[0]), horizontalalignment='left', size='large', color='black', weight='semibold')

plt.ylabel('Score (MAE) on train data', size=20, labelpad=12.5)
plt.xlabel('Model', size=20, labelpad=12.5)
plt.tick_params(axis='x', labelsize=13.5)
plt.tick_params(axis='y', labelsize=12.5)

plt.title('Training Scores of Models', size=20)

plt.show()
```



As shown above, AdaBoost model has the highest traning MAE of 1.027 while the Blender model has the least MAE of 0.445. All the other models give an MAE between these values with a slight difference in each of them.

```
In [60]: #Store models, scores and prediction values
test_models = {'Ridge': ridge_model_full_data,
               #'Lasso': lasso_model_full_data,
               #'ElasticNet': elastic_model_full_data,
               #'SVR': svr_model_full_data,
               'GBR': gbr_model_full_data,
               'LightGBM': lgb_model_full_data,
               'XGBoost': xgb_model_full_data,
               'AdaBoost': adaboost_model_full_data,
               #'Extratrees': extratrees_model_full_data,
               #'Bagging': bagging_model_full_data
               }
test_scores = {}

#Evaluating model score
print('Evaluating model test scores...')
for name, model in test_models.items():
    score = mae( y_test, model.predict(X_test))
    print((name + ": {:.4f} \n").format(score))
    test_scores[name] = score
```

Evaluating model test scores...

Ridge: 0.6322

GBR: 0.4766

LightGBM: 0.4885

XGBoost: 0.5288

AdaBoost: 1.1902

```
In [61]: blender_score = mae(y_test, blend_models_predict(X_test))
test_scores['Blender'] = blender_score
print('MAE score of blender on test data:' + str(blender_score))

baseline_score = baseline_model(y)
test_scores['Baseline'] = baseline_score

sns.set_style("white")
fig = plt.figure(figsize=(24, 12))

ax = sns.pointplot(x=list(test_scores.keys()), y=[score for score in test_scores.values()], markers=['o'], linestyles=['-'])
for i, score in enumerate(test_scores.values()):
    ax.text(i, score + 0.002, '{:.6f}'.format(score), horizontalalignment='left', size='large', color='black', weight='semibold')

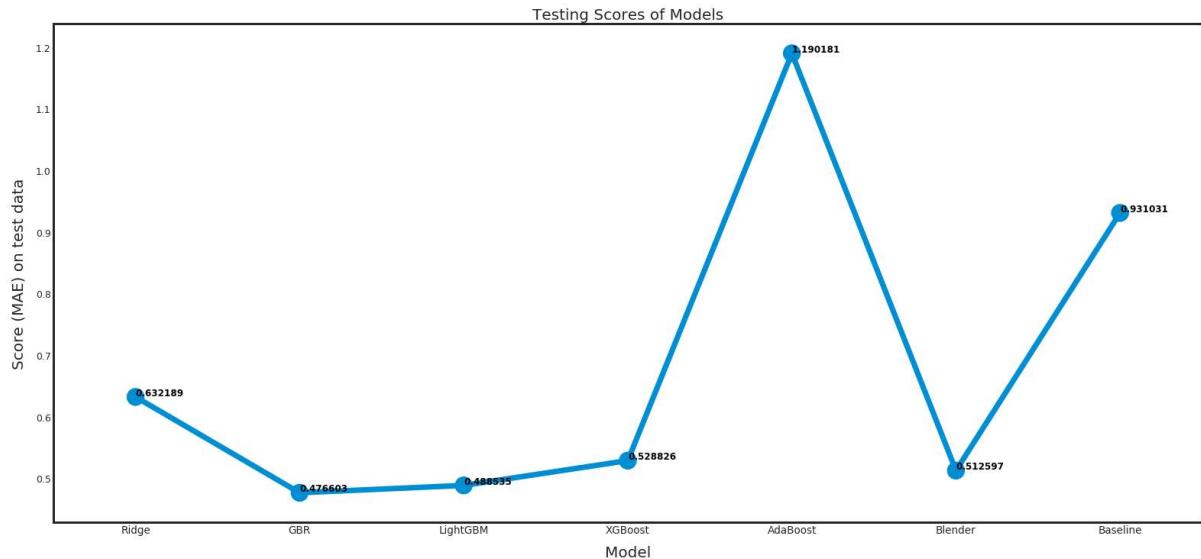
plt.ylabel('Score (MAE) on test data', size=20, labelpad=12.5)
plt.xlabel('Model', size=20, labelpad=12.5)
plt.tick_params(axis='x', labelsize=13.5)
plt.tick_params(axis='y', labelsize=12.5)

plt.title('Testing Scores of Models', size=20)

plt.show()
```

MAE score of blender on test data:0.5125965919433477

MAE of baseline model: 0.931



As shown above, similar to the training data, Adaboost model has the highest testing MAE of 1.19 while the GBR model has the least MAE of 0.476. All the other models give an MAE between these values with a slight difference in each of them. Our baseline mode has the MAE of 0.931 which is higher than most of the individual models except AdaBoost. With the help of GBR model, we get and MAE better than all the models as well as the baseline model.

```
In [62]: def plot_prediction_multistep(actual, prediction, start_date, title, prediction_label):
    date_rng = pd.date_range(start=start_date, periods=24, freq='H')
    plt.figure(figsize=(30,5))
    plt.title(title)
    plt.plot(actual.index, actual, label='Actual')
    plt.plot(actual.index, prediction, label=prediction_label)
    plt.ylabel('Power(kW)')
    plt.xlabel('Datetime')
    plt.legend()
    plt.show()
```

```
In [63]: def get_features(date, comb_df):
    features = comb_df_hourly.loc[date]
    #print(features)

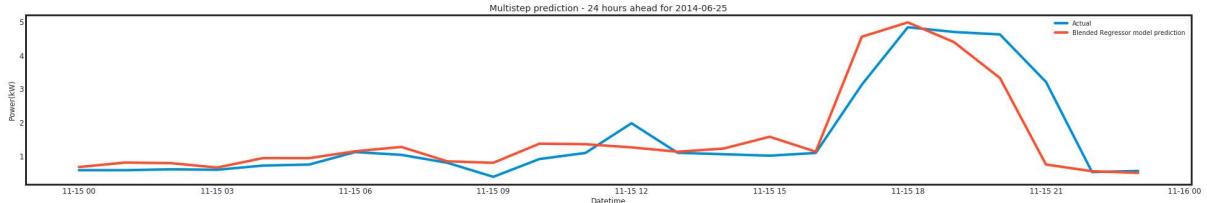
    for day in range(8):
        features['temperature_d' + str(day)] = comb_df_hourly.temperature.shift(day)
        features['windSpeed_d' + str(day)] = comb_df_hourly.windSpeed.shift(day)
        features['humidity_d' + str(day)] = comb_df_hourly.humidity.shift(day)
        features['pressure_d' + str(day)] = comb_df_hourly.pressure.shift(day)
        features['cloudCover_d' + str(day)] = comb_df_hourly.cloudCover.shift(day)
        features['windBearing_d' + str(day)] = comb_df_hourly.windBearing.shift(day)
        features['precipIntensity_d' + str(day)] = comb_df_hourly.precipIntensity.shift(day)
        features['dewPoint_d' + str(day)] = comb_df_hourly.dewPoint.shift(day)
        features['load_d' + str(day)] = comb_df_hourly['Usage_kw_'].shift(day)

    features = features.dropna()
    #print(features)

    features = features.drop(columns=['temperature', 'windSpeed', 'humidity',
                                      'pressure', 'cloudCover',
                                      'windBearing', 'precipIntensity', 'dewPoint',
                                      'Usage_kw_'])

    return features
```

```
In [69]: date = '2016-11-15'
prediction = gbr_model_full_data.predict(get_features(date, comb_df_hourly))
plot_prediction_multistep(actual=comb_df_hourly['Usage_kw'].loc[date], prediction=prediction, start_date=date, title='Multistep prediction - 24 hours ahead for 2014-06-25',
                           prediction_label='Blended Regressor model prediction')
```



The above graph is a sample prediction of Blender model for hourly electricity consumption on 2016-11-15. The model fits the actual consuption quite well.

Daily Data

Unlike for the hourly data, we don't add any lagged features for the daily data. Rest of the data split and traning procedure is the same as above.

```
In [70]: X = comb_df_daily.drop(columns = ['Usage_kw'])
y = comb_df_daily['Usage_kw']
```

```
In [71]: X.shape
```

```
Out[71]: (350, 14)
```

```
In [72]: y.shape
```

```
Out[72]: (350,)
```

```
In [73]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=False)
```

```
In [74]: tscv = TimeSeriesSplit(n_splits=10)
print('Building model...')

alphas_alt = [14.5, 14.6, 14.7, 14.8, 14.9, 15, 15.1, 15.2, 15.3, 15.4, 15.5]
alphas2 = [5e-05, 0.0001, 0.0002, 0.0003, 0.0004, 0.0005, 0.0006, 0.0007, 0.0008]
e_alphas = [0.0001, 0.0002, 0.0003, 0.0004, 0.0005, 0.0006, 0.0007]
e_l1ratio = [0.8, 0.85, 0.9, 0.95, 0.99, 1]

ridge = make_pipeline(RobustScaler(), RidgeCV(alphas=alphas_alt, cv=tscv))

lasso = make_pipeline(RobustScaler(), LassoCV(max_iter=1e7, alphas=alphas2, random_state=42, cv=tscv))

elasticnet = make_pipeline(RobustScaler(), ElasticNetCV(max_iter=1e7, alphas=e_alphas, cv=tscv, l1_ratio=e_l1ratio))

svr = make_pipeline(RobustScaler(), SVR(C= 20, epsilon= 0.008, gamma=0.0004,))

gbr = GradientBoostingRegressor(n_estimators=3000, learning_rate=0.03, max_depth=4, max_features='sqrt', min_samples_leaf=20, min_samples_split=10, loss='huber', random_state =42)

lightgbm = LGBMRegressor(objective='regression',
                         num_leaves=4,
                         learning_rate=0.005,
                         n_estimators=5000,
                         max_bin=200,
                         bagging_fraction=0.75,
                         bagging_freq=5,
                         bagging_seed=7,
                         feature_fraction=0.2,
                         feature_fraction_seed=7,
                         verbose=-1,
                         )

xgboost = XGBRegressor(learning_rate=0.01,n_estimators=4060,
                      max_depth=4, min_child_weight=0,
                      gamma=0, subsample=0.7,
                      colsample_bytree=0.7,
                      objective='reg:linear', nthread=-1,
                      scale_pos_weight=1, seed=27,
                      reg_alpha=0.00005)

adaboost = AdaBoostRegressor()

extratrees = ExtraTreesRegressor()

bagging = BaggingRegressor()

stack_gen = StackingCVRegressor(regressors=(ridge, lasso, elasticnet, gbr, xgboost, lightgbm, adaboost, extratrees, bagging),
                                 meta_regressor=xgboost,
                                 use_features_in_secondary=True)

#Store models, scores and prediction values
```

```
models = {'Ridge': ridge,
          'Lasso': lasso,
          'ElasticNet': elasticnet,
          'SVR': svr,
          'GBR': gbr,
          'LightGBM': lightgbm,
          'XGBoost': xgboost,
          'AdaBoost': adaboost,
          'Extratrees': extratrees,
          'Bagging': bagging}
scores = {}

#Evaluating model score
print('Evaluating model cross-validation scores...')
for name, model in models.items():
    score = cv_mae(model, X_train, y_train)
    print((name + ": {:.4f} ({:.4f})\n").format(score.mean(), score.std()))
    scores[name] = (score.mean(), score.std())

print('Done evaluating')

print('Fitting models...')

print('Fitting StackingRegressor...')
stack_gen_model = stack_gen.fit(np.array(X_train), np.array(y_train))

print('Fitting ElasticNet...')
elastic_model_full_data = elasticnet.fit(X_train,y_train)

print('Fitting Lasso...')
lasso_model_full_data = lasso.fit(X_train,y_train)

print('Fitting Ridge...')
ridge_model_full_data = ridge.fit(X_train,y_train)

print('Fitting SVR...')
svr_model_full_data = svr.fit(X_train,y_train)

print('Fitting GradientBoosting...')
gbr_model_full_data = gbr.fit(X_train,y_train)

print('Fitting XGBoost...')
xgb_model_full_data = xgboost.fit(X_train,y_train)

print('Fitting LightGBM...')
lgb_model_full_data = lightgbm.fit(X_train,y_train)

print('Fitting AdaBoost...')
adaboost_model_full_data = adaboost.fit(X_train,y_train)

print('Fitting extratrees...')
extratrees_model_full_data = extratrees.fit(X_train,y_train)

print('Fitting Bagging...')
bagging_model_full_data = bagging.fit(X_train,y_train)

print('Done fitting all models')
```

Building model...
Evaluating model cross-validation scores...
Ridge: 0.2591 (0.1421)

Lasso: 0.2450 (0.1379)

ElasticNet: 0.2434 (0.1379)

SVR: 0.2920 (0.1805)

GBR: 0.2271 (0.0858)

LightGBM: 0.2240 (0.0826)

[04:18:03] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linea
r is now deprecated in favor of reg:squarederror.
[04:18:03] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linea
r is now deprecated in favor of reg:squarederror.
[04:18:04] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linea
r is now deprecated in favor of reg:squarederror.
[04:18:04] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linea
r is now deprecated in favor of reg:squarederror.
[04:18:05] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linea
r is now deprecated in favor of reg:squarederror.
[04:18:05] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linea
r is now deprecated in favor of reg:squarederror.
[04:18:06] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linea
r is now deprecated in favor of reg:squarederror.
[04:18:07] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linea
r is now deprecated in favor of reg:squarederror.
[04:18:08] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linea
r is now deprecated in favor of reg:squarederror.
[04:18:08] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linea
r is now deprecated in favor of reg:squarederror.
XGBoost: 0.2002 (0.1070)

AdaBoost: 0.2139 (0.0971)

Extratrees: 0.1949 (0.0865)

Bagging: 0.2087 (0.0769)

Done evaluating
Fitting models...
Fitting StackingRegressor...
[04:18:41] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linea
r is now deprecated in favor of reg:squarederror.
[04:18:42] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linea
r is now deprecated in favor of reg:squarederror.
[04:18:43] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linea
r is now deprecated in favor of reg:squarederror.
[04:18:44] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linea
r is now deprecated in favor of reg:squarederror.
[04:18:45] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linea
r is now deprecated in favor of reg:squarederror.
[04:18:49] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linea
r is now deprecated in favor of reg:squarederror.

```
[04:18:56] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.  
Fitting ElasticNet...  
Fitting Lasso...  
Fitting Ridge...  
Fitting SVR...  
Fitting GradientBoosting...  
Fitting XGBoost...  
[04:19:05] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.  
Fitting LightGBM...  
Fitting AdaBoost...  
Fitting extratrees...  
Fitting Bagging...  
Done fitting all models
```

```
In [81]: #Blending model predictions  
print('Blending model predictions...')  
def blend_models_predict(X):  
    return ((0.15 * adaboost_model_full_data.predict(X))  
           +(0.05 * ridge_model_full_data.predict(X)) + (0.1 * svr_model_full_data.predict(X))  
           +(0.15 * gbr_model_full_data.predict(X))+(0.1 * xgb_model_full_data.predict(X))  
           +(0.01 * extratrees_model_full_data.predict(X))  
           +(0.05 * bagging_model_full_data.predict(X))  
           +(0.01 * lgb_model_full_data.predict(X))+ (0.4 * stack_gen_model.predict(np.array(X))))  
  
mae_score = mae(y_train, blend_models_predict(X_train))  
print('MAE score on train data:' + str(mae_score))  
scores['Blender'] = (mae_score.mean(), mae_score.std())
```

```
Blending model predictions...  
MAE score on train data:0.0491760619334717
```

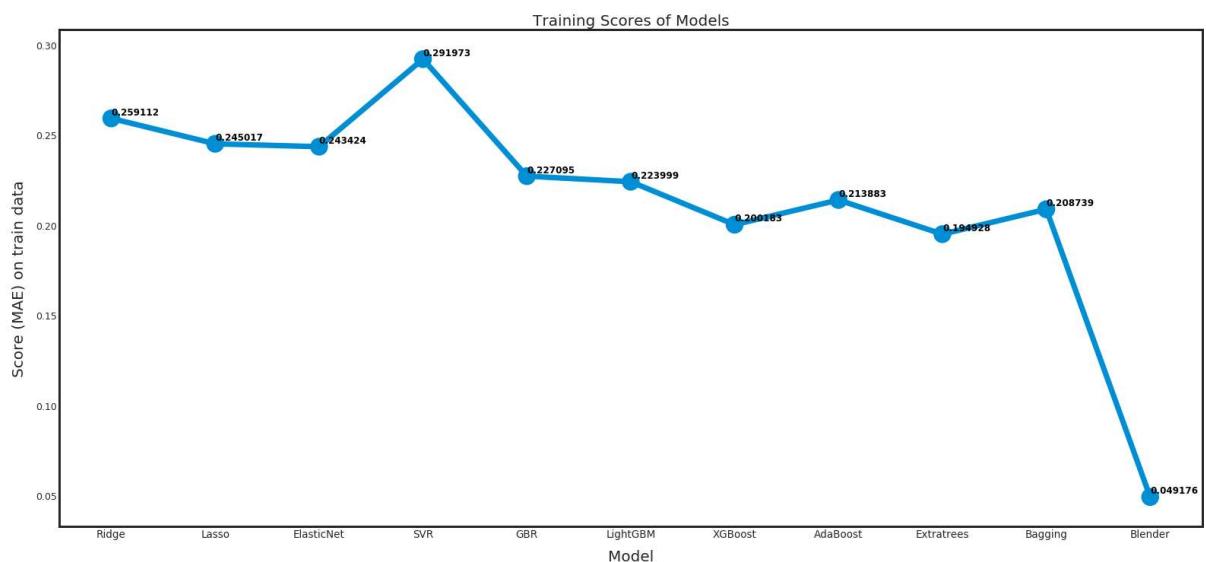
```
In [82]: sns.set_style("white")
fig = plt.figure(figsize=(24, 12))

ax = sns.pointplot(x=list(scores.keys()), y=[score for score, _ in scores.values()], markers=['o'], linestyles=['-'])
for i, score in enumerate(scores.values()):
    ax.text(i, score[0] + 0.002, '{:.6f}'.format(score[0]), horizontalalignment='left', size='large', color='black', weight='semibold')

plt.ylabel('Score (MAE) on train data', size=20, labelpad=12.5)
plt.xlabel('Model', size=20, labelpad=12.5)
plt.tick_params(axis='x', labelsize=13.5)
plt.tick_params(axis='y', labelsize=12.5)

plt.title('Training Scores of Models', size=20)

plt.show()
```



As shown above, SVR model has the highest traning MAE of 0.291 while the Blender model has the least MAE of 0.049. All the other models give an MAE which is lower than SVR but Blender model beats them all.

```
In [83]: #Store models, scores and prediction values
test_models = {'Ridge': ridge_model_full_data,
               'Lasso': lasso_model_full_data,
               'ElasticNet': elastic_model_full_data,
               'SVR': svr_model_full_data,
               'GBR': gbr_model_full_data,
               'LightGBM': lgb_model_full_data,
               'XGBoost': xgb_model_full_data,
               'AdaBoost': adaboost_model_full_data,
               'Extratrees': extratrees_model_full_data,
               'Bagging': bagging_model_full_data}
test_scores = {}

#Evaluating model score
print('Evaluating model test scores...')
for name, model in test_models.items():
    score = mae(y_test, model.predict(X_test))
    print((name + ": {:.4f} \n").format(score))
    test_scores[name] = score
```

Evaluating model test scores...

Ridge: 0.1788

Lasso: 0.1545

ElasticNet: 0.1547

SVR: 0.1842

GBR: 0.1805

LightGBM: 0.1689

XGBoost: 0.1673

AdaBoost: 0.1815

Extratrees: 0.1716

Bagging: 0.1792

```
In [84]: blender_score = mae(y_test, blend_models_predict(X_test))
test_scores['Blender'] = blender_score
print('MAE score of blender on test data:' + str(blender_score))

baseline_score = baseline_model(y)
test_scores['Baseline'] = baseline_score
```

MAE score of blender on test data:0.15616461660716166

MAE of baseline model: 0.378

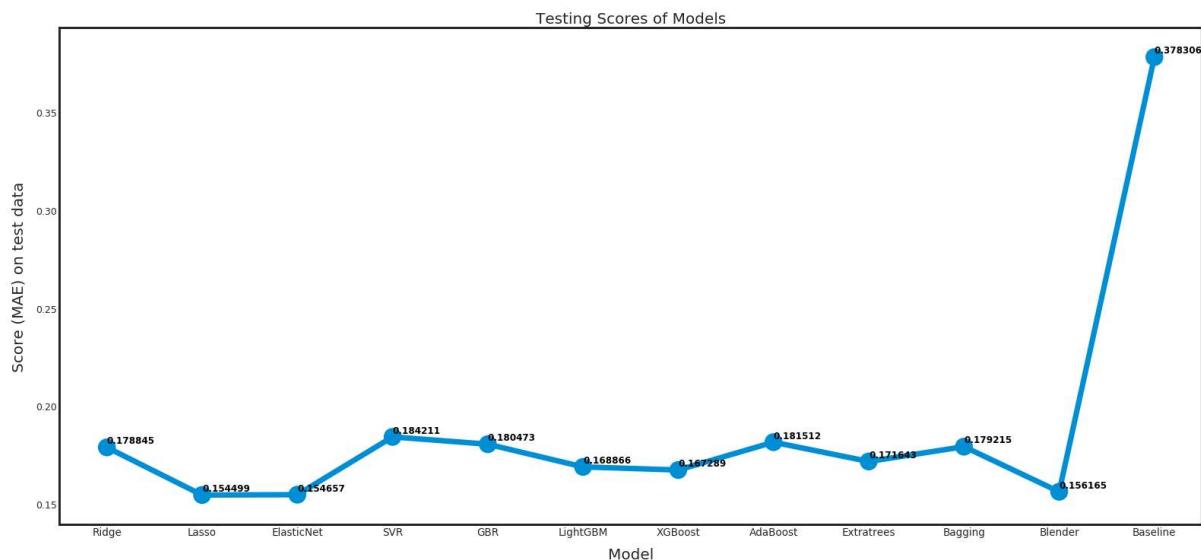
```
In [85]: sns.set_style("white")
fig = plt.figure(figsize=(24, 12))

ax = sns.pointplot(x=list(test_scores.keys()), y=[score for score in test_scores.values()], markers=['o'], linestyles=['-'])
for i, score in enumerate(test_scores.values()):
    ax.text(i, score + 0.002, '{:.6f}'.format(score), horizontalalignment='left', size='large', color='black', weight='semibold')

plt.ylabel('Score (MAE) on test data', size=20, labelpad=12.5)
plt.xlabel('Model', size=20, labelpad=12.5)
plt.tick_params(axis='x', labelsize=13.5)
plt.tick_params(axis='y', labelsize=12.5)

plt.title('Testing Scores of Models', size=20)

plt.show()
```



As shown above, similar to the training data, Baseline model has the highest testing MAE of 0.378 while the Lasso model has least MAE of 0.154 which is slightly better than Blender model which has the MAE of 0.156. All the other models give an MAE between these values with a slight difference in each of them. Since we got the least testing MAE with blender model, we predict daily consumption With the help of blender model.

```
In [86]: def subplot_prediction(actual, prediction,prediction_label):
    fig, axes = plt.subplots(nrows=2, ncols=1, figsize=(20, 15))

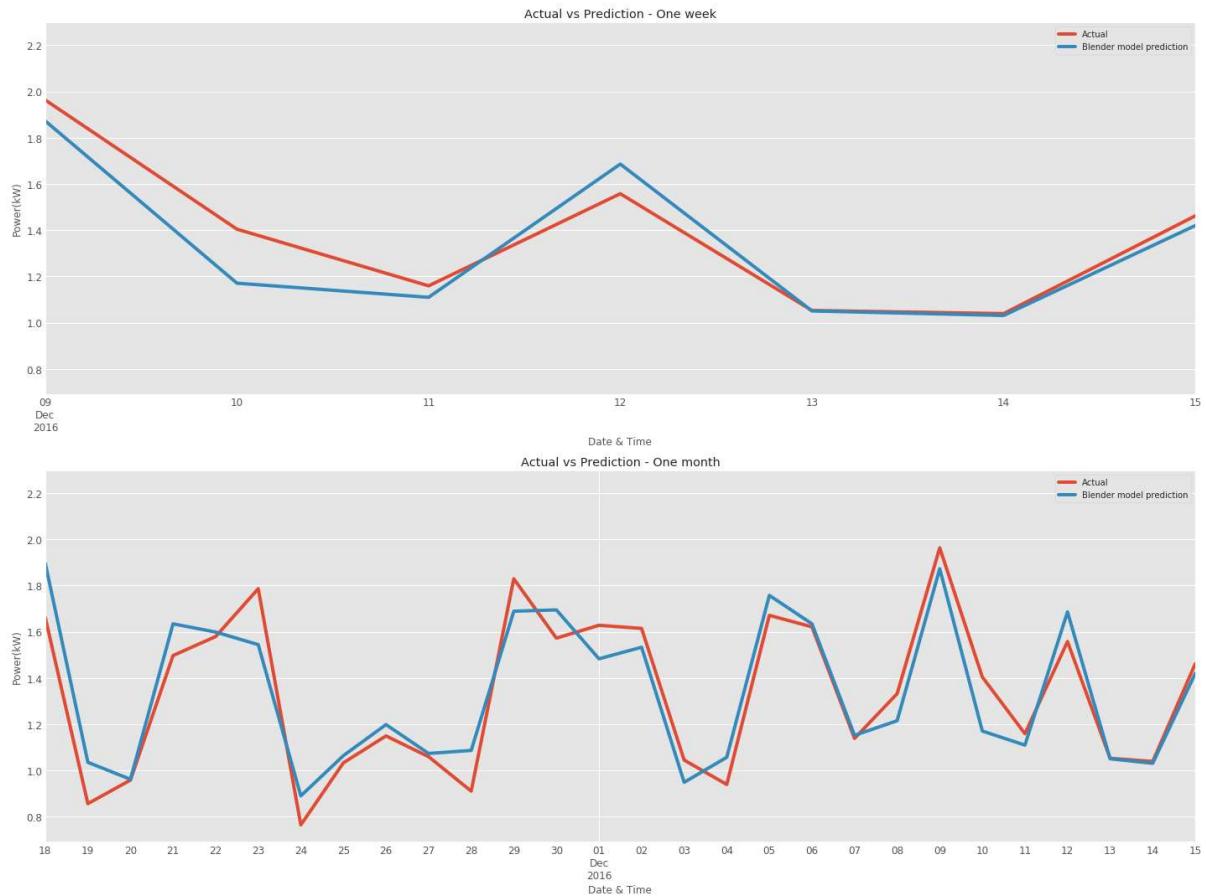
    con_df = pd.concat([actual.rename('Actual'),pd.DataFrame(prediction, index=actual.index, columns=[prediction_label])], axis=1)

    axes[0].set_title('Actual vs Prediction - One week')
    axes[0].set_ylabel('Power(kW)')
    axes[0].set_xlabel('Datetime')
    con_df.plot(ax=axes[0])
    axes[0].set_xlim(left=actual.index[-7] , right=actual.index[-1])

    axes[1].set_title('Actual vs Prediction - One month')
    axes[1].set_ylabel('Power(kW)')
    axes[1].set_xlabel('Datetime')
    con_df.plot(ax=axes[1])
    axes[1].set_xlim(left=actual.index[-7*4] , right=actual.index[-1])

    plt.tight_layout()
    plt.show()
```

```
In [87]: plt.style.use('ggplot')
subplot_prediction(y_test, blend_models_predict(X_test),prediction_label='Blender model prediction')
```



Above are the daily prediction plots for an entire last week of December for our blender model. The model seems to fit the actual consumption decently. Same is the case with the predictions of the model over last half of November and first half of December.