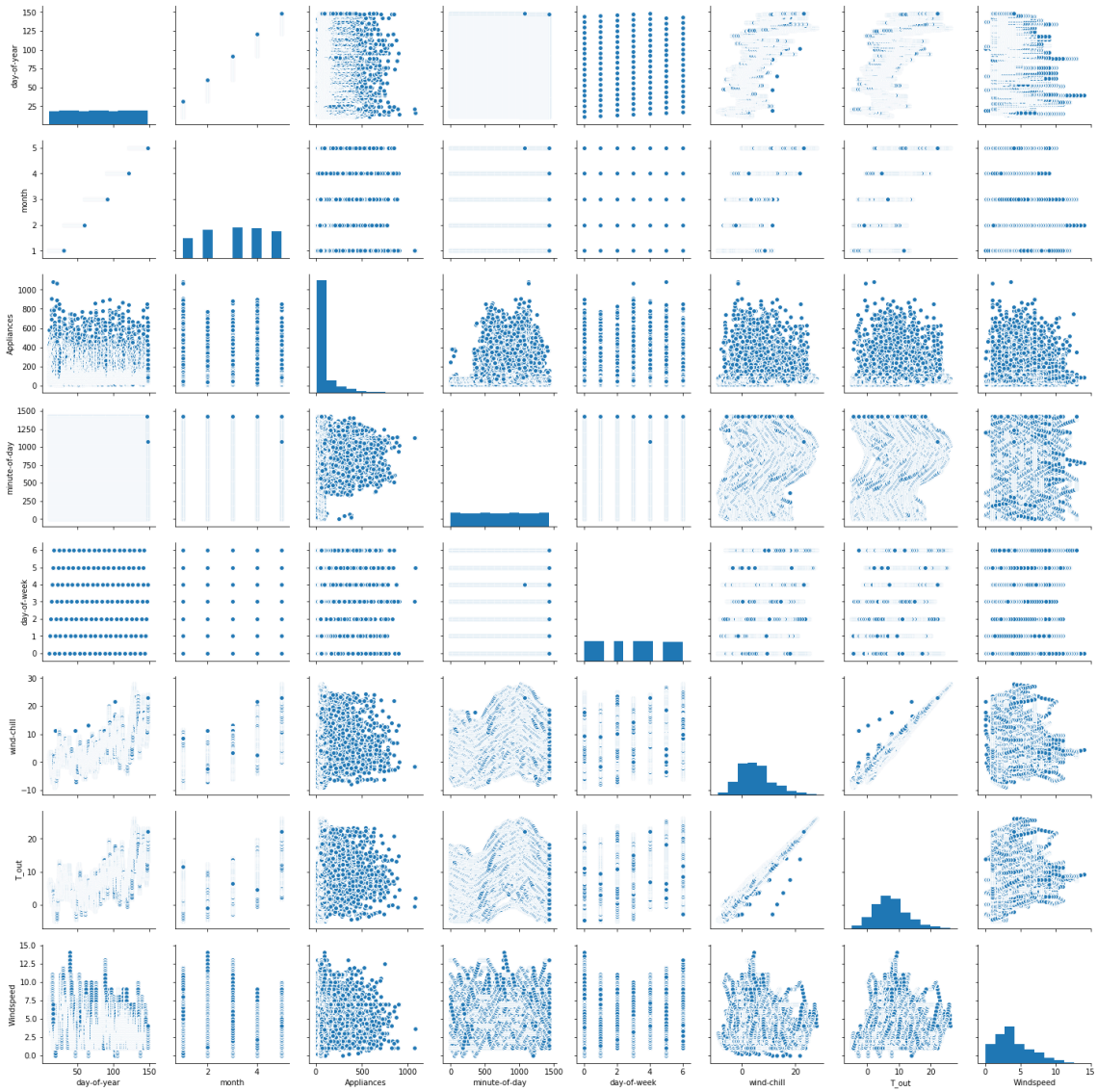
Part3

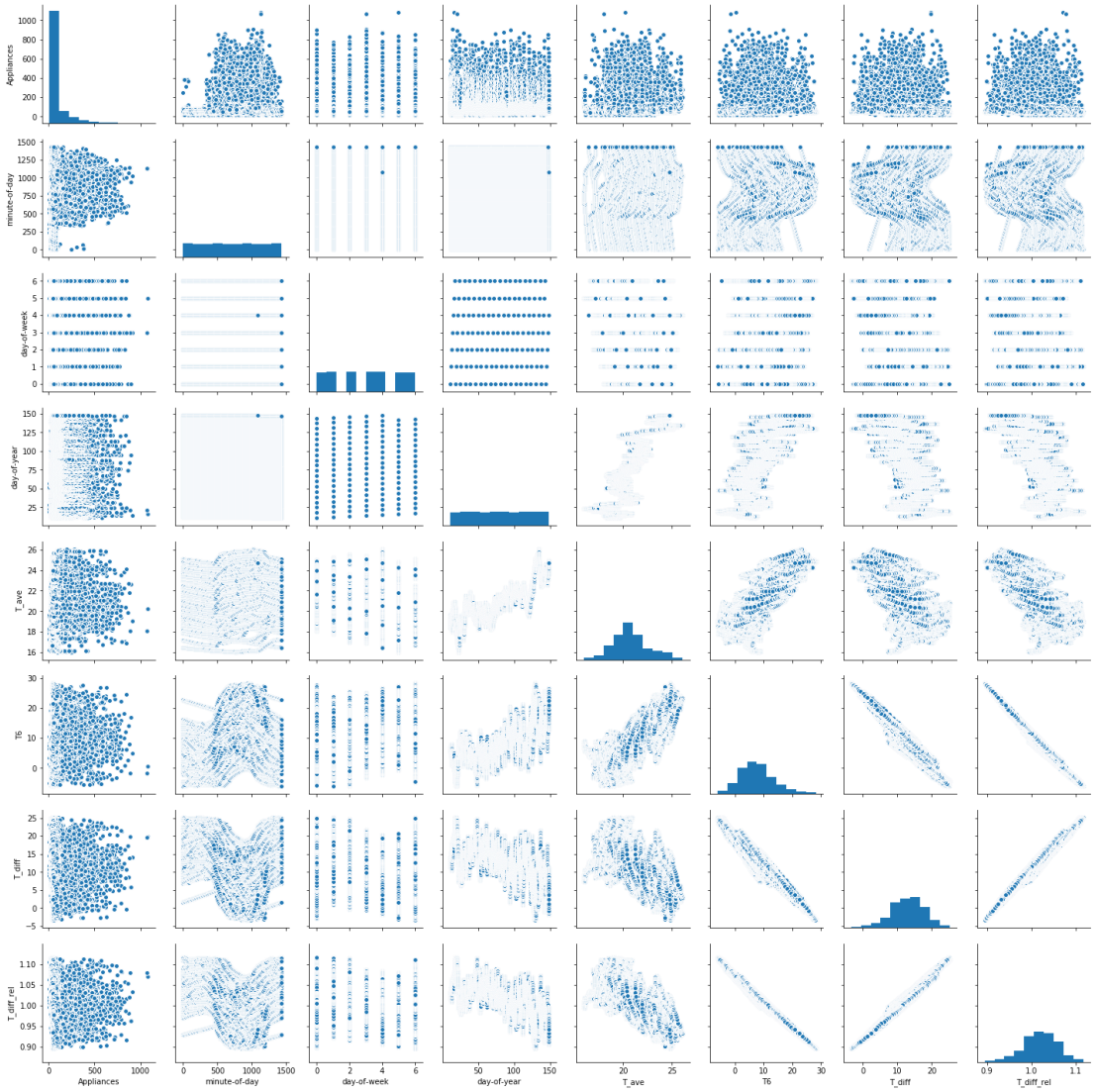
In this dataset, there are 29 columns. Apart from appliance-target features, we still have 28 features. When describing the dataset, we found column data has format of data. In order to get more detailed information, we decided to generate new features below: day-of-year, month, minute, day-of-week, hour and day. In this assignment, we divided column date into those parts. Furthermore, we calculate minute-of-day. We found there are lots of data of temperature so we introduced the concept of wind-chill. Wind-chill is the lowering of body temperature due to the passing-flow of lower-temperature air. By using this, we can combine several columns of data into one column without any loss. Here is calculation of wind-chill:

$$T_{\mathrm{wc}} = 13.12 + 0.6215T_{\mathrm{a}} - 11.37v^{+0.16} + 0.3965T_{\mathrm{a}}v^{+0.16}$$

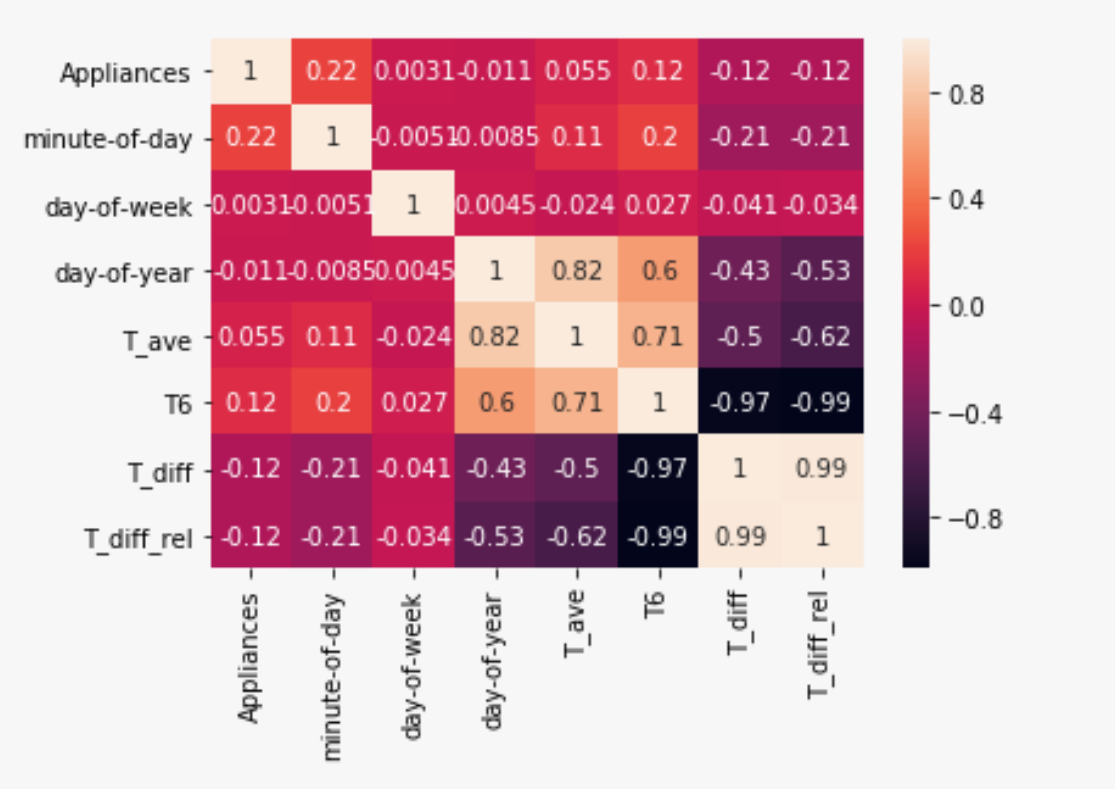
Here is the pair plot of features:



Also, we calculated average temperature of temperature columns except temperature outside because we already had wind-chill.
Here is pair plot of average temperature and other features:



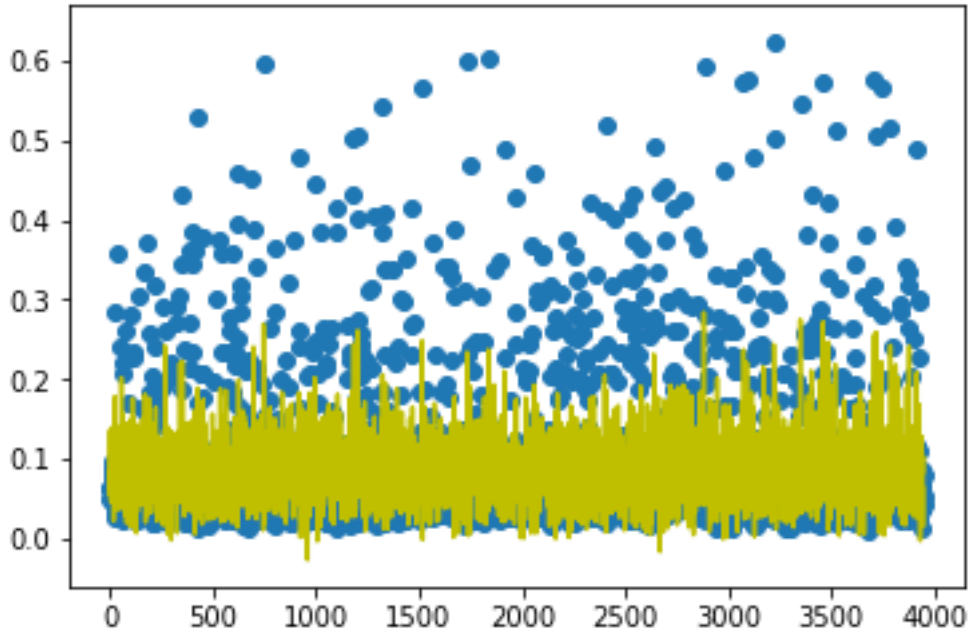
Here is heat map of those features:



Part4

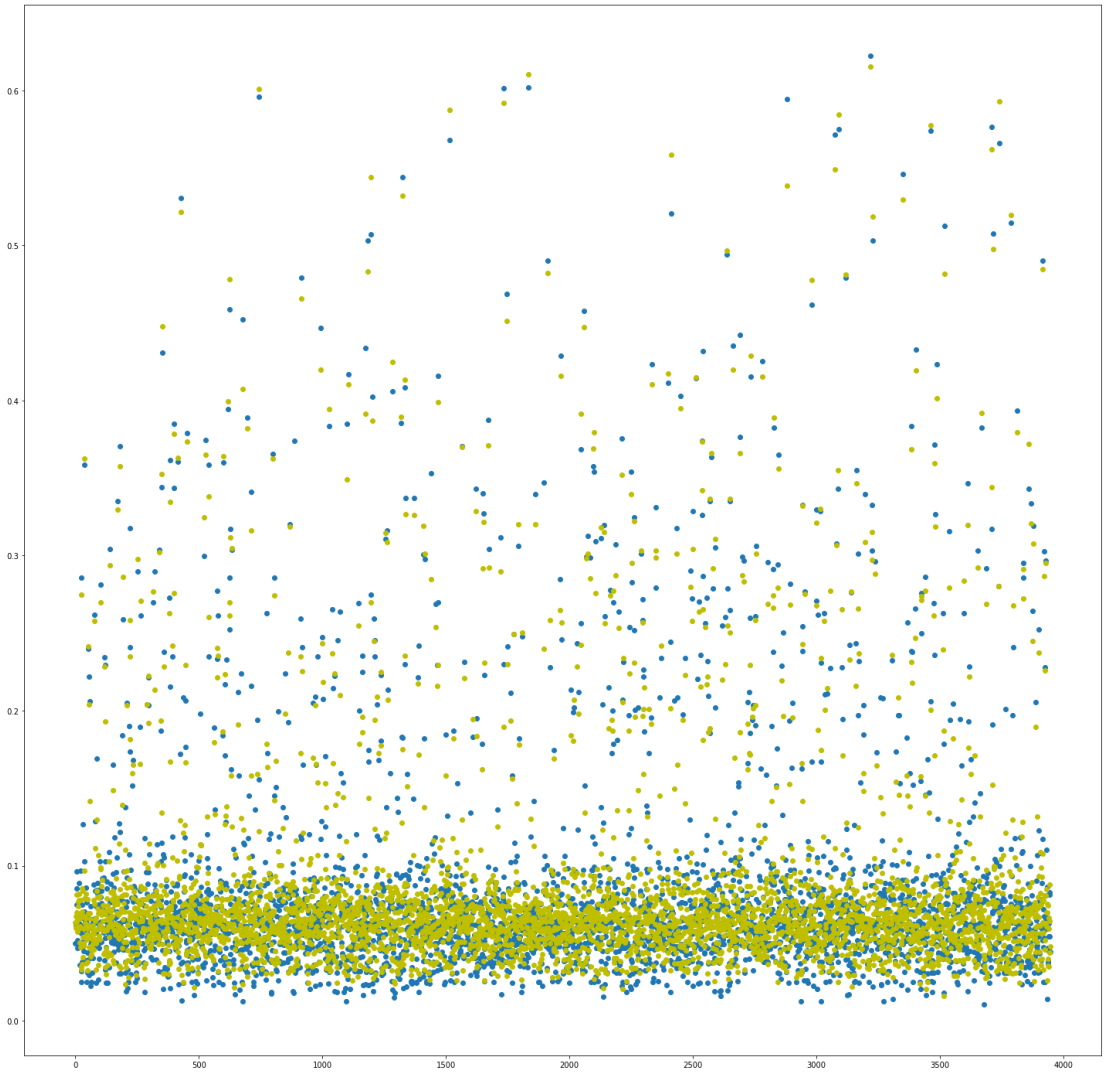
We combined the best performing algorithms into an ensemble, like algorithms with the best r-squared: Polynomial, Random Forest and decision trees The R2 (or R Squared) metric provides an indication of the goodness of fit of a set of predictions to the actual values. In statistical literature this measure is called the coefficient of determination. This is a value between 0 and 1 for no-fit and perfect fit respectively
For Linear regression model, here are output results and scatter:

```
Score: 0.22034318218030624
RMS: 0.06849887508397509
MAPE: 70.94891985643318
R2: 0.2203431821803062 4
MAE: 0.04658829270423372
```



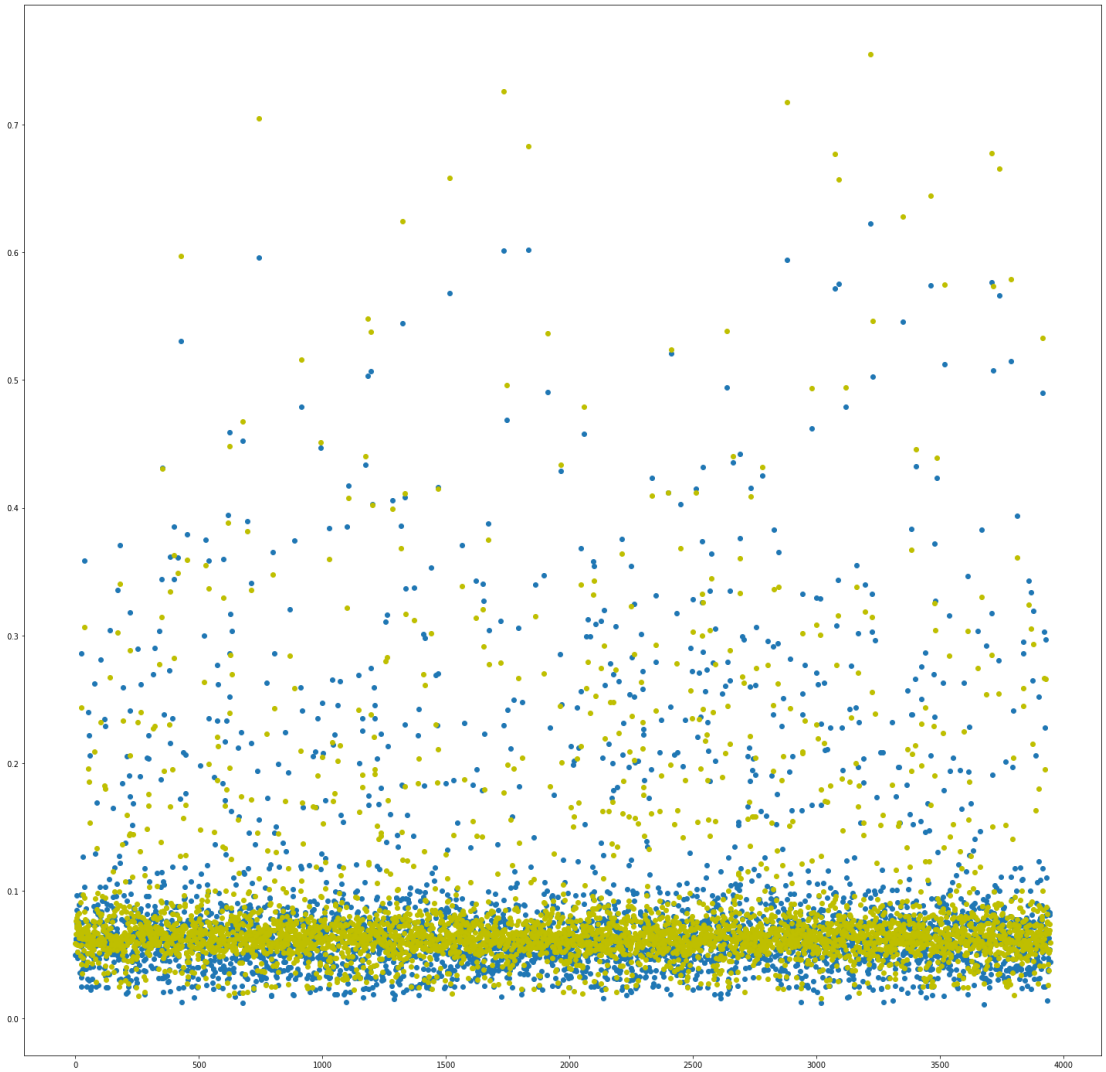
For random forest model, here are output results and scatter:

```
Score: 0.9465579341687849
RMS: 0.017933841112053463
MAPE: 19.144037481591873
R2: 0.9465579341687849
MAE: 0.011911473851883006
```



For neural network model, here are output results and scatter:

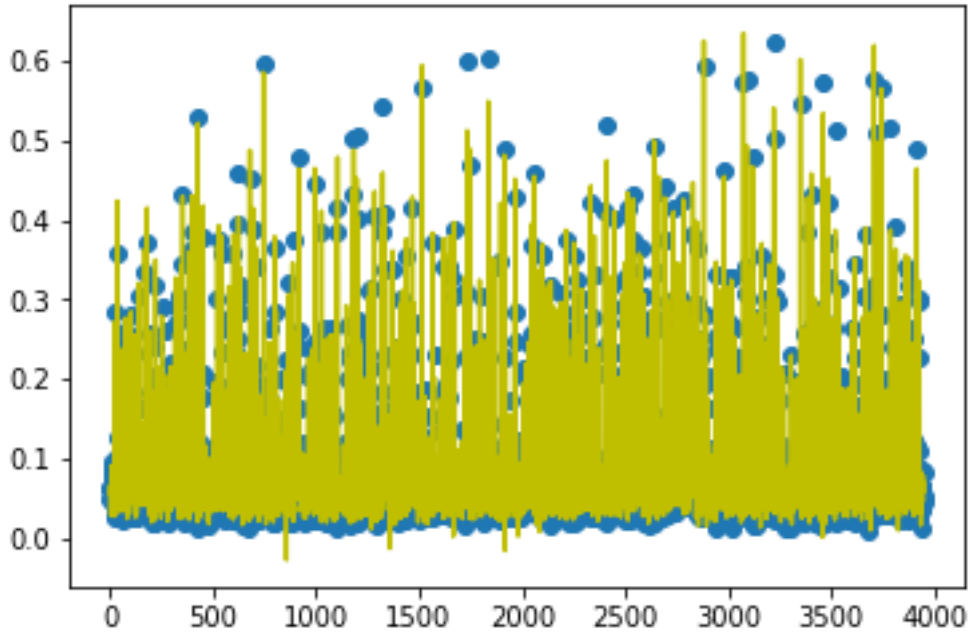
```
Score for train: 0.9428340492936431
Score: 0.9349424633561112
RMS: 0.01978702517861623
MAPE: 21.48418948506335
R2: 0.9349424633561112
MAE: 0.014037877799083297
```



We also try out polynomial regression and decision tree model.

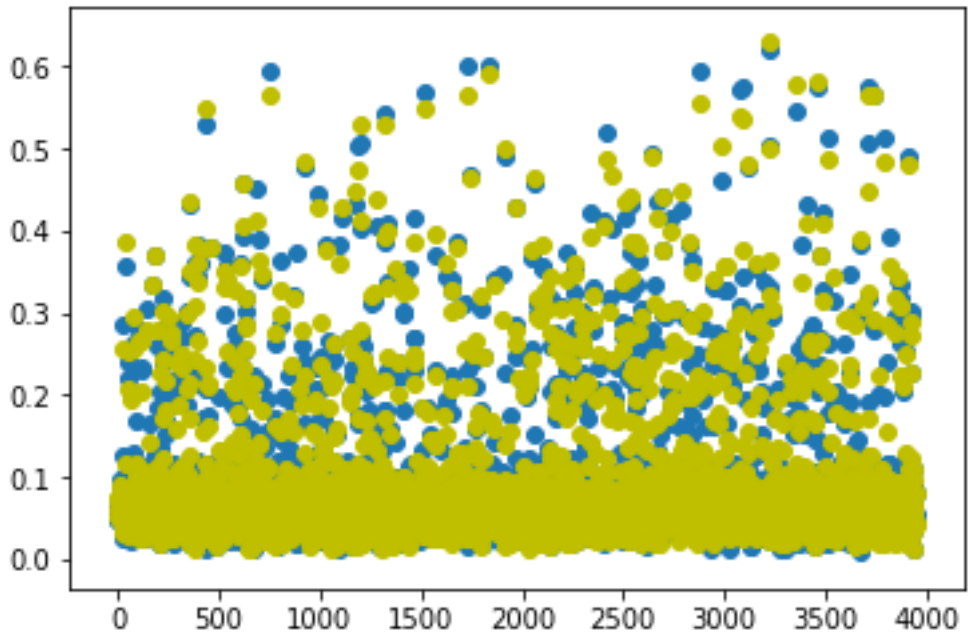
For neural polynomial regression, here are output results and scatter:

```
Score: 0.9666509046886581
RMS: 0.014166864085253233
MAPE: 14.422534352836344
R2: 0.9666509046886581
MAE: 0.009295765002677314
```



For neural decision tree, here are output results and scatter:

```
Score: 0.9109805351421665
RMS: 0.02314590618398445
MAPE: 23.149076212528506
R2: 0.9109805351421665
MAE: 0.014945712703360117
```



By comparing those output results and scatters, we found that the best model for this dataset is random forest model.

Part5

By using RandomForestRegressor, we can calculate importance values for each features. The results follows:

```
[0.00426554 0.00172193 0.00150963 0.00211112 0.00200439 0.00202243
 0.00215221 0.00206047 0.00219201 0.00313392 0.00222052 0.0014881
 0.00230649 0.00185351 0.00181535 0.0052133  0.00212874 0.00138475
 0.00179518 0.00163525 0.42639531 0.00210903 0.00283737 0.0022469
 0.00193172 0.00107188 0.00120345 0.00443207 0.00452527 0.00524082
 0.00306729 0.15885747 0.00566405 0.3249765  0.00208397 0.00111743
 0.0019951  0.0020421  0.00113283 0.00205459]
```

And recursive feature elimination can also provide Boolean value and rank order for each features:

```
[False False False  True False False False False  True  True False False
 False False False  True False False False False  True False False False
 False False False  True False  True False  True False  True False False
 False  True False False]
[ 6 20 24  1 17  9 25 10  1  1 14 26  3 22 19  1  7 29 21 27  1 16  8 13
 12 23 31  1  5  1  4  1  2  1 15 28 18  1 30 11]
```

Tpot:

The fit function initializes the algorithm to find the highest-scoring pipeline based on average k-fold cross-validation. Then, the pipeline is trained on the entire set of provided samples, and the TPOT instance can be used as a fitted model. Result:

```
Out[73]: 0.9360076787087618
```

Featuretools:

Featuretools excels at transforming temporal and relational datasets into feature matrices for machine learning. By using featuretools, we can get relation between each feature.

```
Out[64]: Entityset: energy
           Entities:
             ft_t [Rows: 19735, Columns: 10]
           Relationships:
             No relationships

In  [65]: es.entity_from_dataframe(entity_id = 'ft_rh',
                                   dataframe = ft_rh,
                                   index = 'id')

Out[65]: Entityset: energy
           Entities:
             ft_t [Rows: 19735, Columns: 10]
             ft_rh [Rows: 19735, Columns: 10]
           Relationships:
             No relationships

In  [66]: es.entity_from_dataframe(entity_id = 'ft_data',
                                   dataframe = ft_data,
                                   index = 'id')

Out[66]: Entityset: energy
           Entities:
             ft_t [Rows: 19735, Columns: 10]
             ft_rh [Rows: 19735, Columns: 10]
             ft_data [Rows: 19735, Columns: 43]
           Relationships:
             No relationships
```

Deployment of machine learning models requires repeating feature engineering steps

on new data. In some cases, these steps need to be performed in near real-time. Featuretools has capabilities to ease the deployment of feature engineering.
We built some features definitions using DFS. Because we have categorical features, we also encode them with one hot encoding based on the values in the training data. We have the exact same features as before, but calculated on using our test data.
Here is the result:

```
In [70]: feature_matrix
```

Out[70]:

	Appliances	lights	T1	RH_1	T2	RH_2	T3	RH_3	T4	RH_4	T5
id											
0	60	30	19.890000	47.596667	19.200000	44.790000	19.790000	44.730000	19.000000	45.566667	17.166
1	60	30	19.890000	46.693333	19.200000	44.722500	19.790000	44.790000	19.000000	45.992500	17.166
2	50	30	19.890000	46.300000	19.200000	44.626667	19.790000	44.933333	18.926667	45.890000	17.166
3	50	40	19.890000	46.066667	19.200000	44.590000	19.790000	45.000000	18.890000	45.723333	17.166
4	60	40	19.890000	46.333333	19.200000	44.530000	19.790000	45.000000	18.890000	45.530000	17.200
5	50	40	19.890000	46.026667	19.200000	44.500000	19.790000	44.933333	18.890000	45.730000	17.133
6	60	50	19.890000	45.766667	19.200000	44.500000	19.790000	44.900000	18.890000	45.790000	17.100
7	60	50	19.856667	45.560000	19.200000	44.500000	19.730000	44.900000	18.890000	45.863333	17.100
8	60	40	19.790000	45.597500	19.200000	44.433333	19.730000	44.790000	18.890000	45.790000	17.166
9	70	40	19.856667	46.090000	19.230000	44.400000	19.790000	44.863333	18.890000	46.096667	17.100
10	230	70	19.926667	45.863333	19.356667	44.400000	19.790000	44.900000	18.890000	46.430000	17.100
11	580	60	20.066667	46.396667	19.426667	44.400000	19.790000	44.826667	19.000000	46.430000	17.100
12	430	50	20.133333	48.000000	19.566667	44.400000	19.890000	44.900000	19.000000	46.363333	17.100
13	250	40	20.260000	52.726667	19.730000	45.100000	19.890000	45.493333	19.000000	47.223333	17.100
14	100	10	20.426667	55.893333	19.856667	45.833333	20.033333	47.526667	19.000000	48.696667	17.100
15	100	10	20.566667	53.893333	20.033333	46.756667	20.100000	48.466667	19.000000	48.490000	17.150
16	90	10	20.730000	52.660000	20.166667	47.223333	20.200000	48.530000	18.926667	48.156667	17.166
17	70	30	20.856667	53.660000	20.200000	47.056667	20.200000	48.447500	18.890000	47.963333	17.200
18	80	30	20.890000	51.193333	20.200000	46.330000	20.200000	48.193333	18.963333	48.630000	17.200

Tsfresh:
Tsfresh is used to extract characteristics from time series. When we want to calculate different characteristics like the maximal or minimal temperature, tsfresh can calculate those features automatically. The extracted features can be used to describe time series based on the extracted characteristics. Further, they can be used to build models that perform regression tasks on the time series. Results as follows:

```
In [55]: extracted_features
```

Out[55]:

variable	T2__abs_energy	T2__absolute_sum_of_changes	T2__agg_autocorrelation__f_agg_"mean"	T2__agg_autocorrela
id				
0	368.640000	0.0	0.0	0.0
1	368.640000	0.0	0.0	0.0
2	368.640000	0.0	0.0	0.0
3	368.640000	0.0	0.0	0.0
4	368.640000	0.0	0.0	0.0
5	368.640000	0.0	0.0	0.0
6	368.640000	0.0	0.0	0.0
7	368.640000	0.0	0.0	0.0
8	368.640000	0.0	0.0	0.0
9	369.792900	0.0	0.0	0.0
10	374.680544	0.0	0.0	0.0
11	377.395378	0.0	0.0	0.0
12	382.854444	0.0	0.0	0.0
13	389.272900	0.0	0.0	0.0
14	394.287211	0.0	0.0	0.0
15	401.334444	0.0	0.0	0.0
16	406.694444	0.0	0.0	0.0

To limit the number of irrelevant features, tsfresh deploys the fresh algorithm.

```
In [58]: features_filtered
```

Out[58]:

variable
id
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16

Part6

To begin with, we have RandomForestRegressor as the base:

```
In [130]: #Base
rf_reg = RandomForestRegressor()
rf_reg.fit(train_X,train_y)
```

```
Out[130]: RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
           oob_score=False, random_state=None, verbose=0, warm_start=False)
```

```
In [86]: pred_y = rf_reg.predict(test_X)
print("Score for train: "+str(rf_reg.score(train_X,train_y)))
print("Score: "+str(rf_reg.score(test_X,test_y)))
print("RMS: "+str(sqrt(metrics.mean_squared_error(test_y,pred_y))))
print("MAPE: "+str(mean_absolute_percentage_error(test_y,pred_y)))
print("R2: "+str(metrics.r2_score(test_y,pred_y)))
print("MAE: "+str(metrics.mean_absolute_error(test_y,pred_y)))
```

```
Score for train: 0.9915702596048136
Score: 0.9516610706751687
RMS: 0.01705611879980532
MAPE: 18.993608768091093
R2: 0.9516610706751686
MAE: 0.011468002315745776
```

Cross validation techniques:
We need to make sure our model get most of the patterns from the data correct and low on bias and variance. A solution to this problem is a procedure called cross-validation. A test set should still be held out for final evaluation, but the validation set is no longer needed when doing CV. In the basic approach, called k-fold CV, the training set is split into k smaller sets. The performance measure reported by k-fold cross-validation is then the average of the values computed in the loop.

```
In [111]: rf_reg = RandomForestRegressor()
          cross_validate(rf_reg, test_X, test_y, cv=5)

C:\Users\wenqi\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:122: FutureWarning: You are accessing a training score ('train_score'), which will not be available by default any more in 0.21. If you need training scores, please set return_train_score=True
  warnings.warn(*warn_args, **warn_kwargs)

Out[111]: {'fit_time': array([0.22418976, 0.21246481, 0.2023015 , 0.22096634, 0.20265746]),
           'score_time': array([0.00199556, 0.00298429, 0.        , 0.00199556, 0.00199103]),
           'test_score': array([0.82039218, 0.8530837 , 0.82409016, 0.90117162, 0.87655102]),
           'train_score': array([0.97104676, 0.97893739, 0.97339082, 0.97007904, 0.96745516])}

In [105]: cross_val_score(rf_reg, test_X, test_y, cv=5).mean()

Out[105]: 0.8523972301285943
```

Regularization:

Regularization basically adds the penalty as model complexity increases.

Regularization parameter penalizes all the parameters except intercept so that model generalizes the data and won't over fit.

L1 Regularization:

```
In [114]: mlp_reg2 = MLPRegressor(hidden_layer_sizes=(400,90),learning_rate='adaptive',solver='adam',random_state=3,learning_rate_init=0.001,max_iter=40
          cross_validate(mlp_reg2, test_X, test_y, cv=5)

C:\Users\wenqi\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:122: FutureWarning: You are accessing a training score ('train_score'), which will not be available by default any more in 0.21. If you need training scores, please set return_train_score=True
  warnings.warn(*warn_args, **warn_kwargs)

Out[114]: {'fit_time': array([0.40792131, 0.34281397, 0.30360746, 0.30029273, 0.28960633]),
           'score_time': array([0.00698161, 0.01562166, 0.00398993, 0.        , 0.00494814]),
           'test_score': array([0.13841787, 0.12107456, 0.1382989 , 0.11692627, 0.11034806]),
           'train_score': array([0.1410797 , 0.15357985, 0.13560692, 0.12900436, 0.1003923 ])}
```

L2 Regularization:

```
In [115]: #add L2 regularization
          mlp_reg2 = MLPRegressor(hidden_layer_sizes=(400,90),learning_rate='adaptive',solver='adam',random_state=3,learning_rate_init=0.001,max_iter=40
          cross_validate(mlp_reg2, test_X, test_y, cv=5)

C:\Users\wenqi\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:122: FutureWarning: You are accessing a training score ('train_score'), which will not be available by default any more in 0.21. If you need training scores, please set return_train_score=True
  warnings.warn(*warn_args, **warn_kwargs)

Out[115]: {'fit_time': array([1.12886024, 1.08572245, 0.88478136, 0.87482834, 0.84900045]),
           'score_time': array([0.01104522, 0.        , 0.        , 0.01558614, 0.        ]),
           'test_score': array([0.67176732, 0.65772642, 0.63790205, 0.56436262, 0.61956687]),
           'train_score': array([0.70441718, 0.67110998, 0.64894166, 0.583765  , 0.55805821])}
```

Elastic net:

```
Model Name: Elastic NetElasticNet(alpha=1.0, copy_X=True, fit_intercept=True, l1_ratio=0.5,
      max_iter=1000, normalize=False, positive=False, precompute=False,
      random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Score: -0.0002587115528174255 3
RMS: 0.07758680535785652
MAPE: 65.30910405425148
R2: -0.0002587115528174255 3
MAE: 0.045881227827121875
```

Grid Search:

Grid search exhaustively generates candidates from a grid of parameter values specified with the paramgrid parameter:

```
In [178]: param_grid = {'max_depth': [None, 5, 10, 15],
                        'n_estimators': np.arange(3, 20),
                        'max_features': ('auto','sqrt','log2')
          }
```

```
In [181]: grid.fit(test_X, test_y)

Out[181]: GridSearchCV(cv=7, error_score='raise',
          estimator=RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=5,
              max_features='auto', max_leaf_nodes=None,
              min_impurity_decrease=0.0, min_impurity_split=None,
              min_samples_leaf=1, min_samples_split=2,
              min_weight_fraction_leaf=0.0, n_estimators=3, n_jobs=1,
              oob_score=False, random_state=None, verbose=0, warm_start=False),
          fit_params={}, iid=True, n_jobs=1,
          param_grid={'max_depth': [None, 5, 10, 15], 'n_estimators': array([ 3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 1
9]), 'max_features': ('auto', 'sqrt', 'log2')},
          pre_dispatch='2*n_jobs', refit=True, scoring=None, verbose=0)
```

The result of best parameters as follow:

```
In [182]: grid.best_params_

Out[182]: {'max_depth': None, 'max_features': 'auto', 'n_estimators': 19}
```

```
In [186]: rf_reg = grid.best_estimator_

          pred_y = rf_reg.predict(test_X)
          print("Score for train: "+str(cross_val_score(rf_reg, test_X, test_y, cv=5).mean()))
          print("Score: "+str(rf_reg.score(test_X,test_y)))
          print("RMS: "+str(sqrt(metrics.mean_squared_error(test_y,pred_y))))
          print("MAPE: "+str(mean_absolute_percentage_error(test_y,pred_y)))
          print("R2: "+str(metrics.r2_score(test_y,pred_y)))
          print("MAE: "+str(metrics.mean_absolute_error(test_y,pred_y)))

Score for train: 0.8712407690757924
Score: 0.9790114885374709
RMS: 0.011238866650231605
MAPE: 9.771957047653059
R2: 0.9790114885374709
MAE: 0.006606608955862635
```

Part7

In this part, we have three steps.

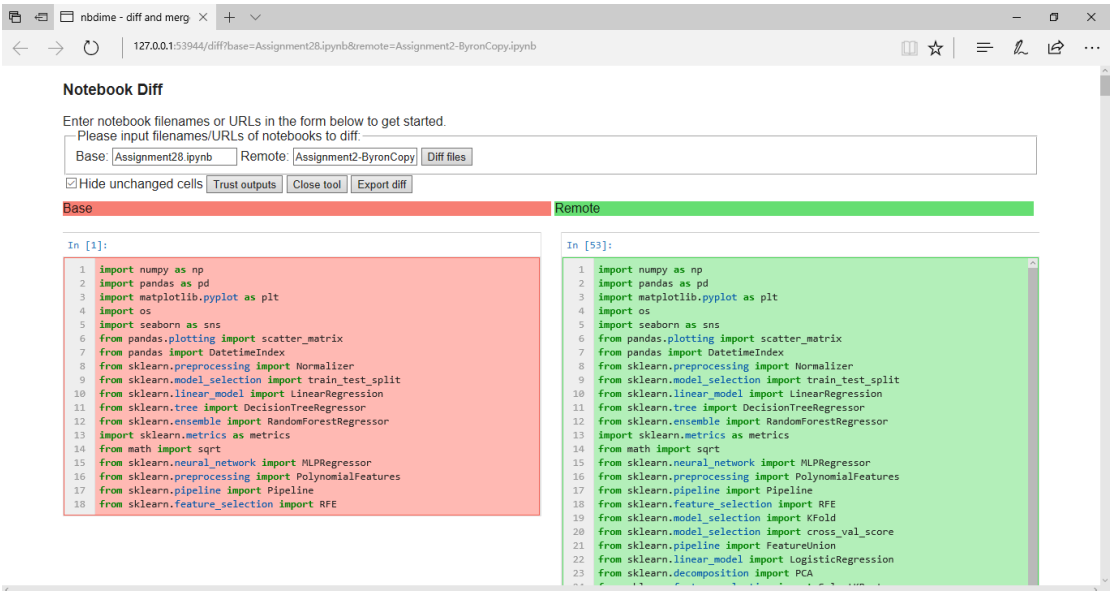
The first step is to create feature union. The second step is to create pipeline. As we mentioned before, random forest is the best model for this case. The last step is to evaluate pipeline. Result showed as follow: 0.9704315913681134

We know from the results in the previous section that Random Forest algorithm achieves good results on the dataset. But can it do better. The default value for the number of neighbors in Random Forest is 10. Here we can use a grid search to try a set of different numbers of neighbors and see if we can improve the score. The below example tries odd k values from 1 to 50, an arbitrary range covering a known good value of 10. Each k value (n neighbors) is evaluated using 10-fold cross-validation on a standardized copy of the training dataset.

```
Best: 0.953782 using {'n_estimators': 33}
0.862313 (0.010438) with: {'n_estimators': 1}
0.918763 (0.010577) with: {'n_estimators': 3}
0.932028 (0.011125) with: {'n_estimators': 5}
0.940776 (0.006071) with: {'n_estimators': 7}
0.945462 (0.006527) with: {'n_estimators': 9}
0.945700 (0.006557) with: {'n_estimators': 11}
0.946220 (0.006664) with: {'n_estimators': 13}
0.948166 (0.005843) with: {'n_estimators': 15}
0.950178 (0.005528) with: {'n_estimators': 17}
0.949680 (0.006174) with: {'n_estimators': 19}
0.950261 (0.006229) with: {'n_estimators': 21}
0.950791 (0.004759) with: {'n_estimators': 23}
0.951298 (0.005814) with: {'n_estimators': 25}
0.951647 (0.004516) with: {'n_estimators': 27}
0.951665 (0.006353) with: {'n_estimators': 29}
0.951239 (0.006418) with: {'n_estimators': 31}
0.953782 (0.006061) with: {'n_estimators': 33}
0.952695 (0.006095) with: {'n_estimators': 35}
0.952583 (0.005514) with: {'n_estimators': 37}
0.953048 (0.006273) with: {'n_estimators': 39}
0.952479 (0.005206) with: {'n_estimators': 40}
0.952086 (0.005107) with: {'n_estimators': 43}
0.952796 (0.006472) with: {'n_estimators': 45}
0.952583 (0.005800) with: {'n_estimators': 47}
0.953108 (0.005865) with: {'n_estimators': 49}
0.953546 (0.005683) with: {'n_estimators': 51}
```

Part8

Nbdime provides tools for diffing and merging Jupyter notebooks. Nbdime provides “content-aware” diffing and merging of Jupyter notebooks. It understands the structure of notebook documents. Therefore, it can make intelligent decisions when diffing and merging notebooks.
We can use command nbdiff-web Assignment28.ipynb Assignment2-ByronCopy.ipynb
Result like follow:


nbdime - diff and merg
127.0.0.1:53944/diff?base=Assignment28.ipynb&remote=Assignment2-ByronCopy.ipynb
Notebook Diff
Enter notebook filenames or URLs in the form below to get started.
Please input filenames/URLs of notebooks to diff:
Base: Assignment28.ipynb
Remote: Assignment2-ByronCopy
Diff files
Hide unchanged cells
Trust outputs
Close tool
Export diff
Base
Remote
In [1]:
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os
import seaborn as sns
from pandas.plotting import scatter_matrix
from pandas import DatetimeIndex
from sklearn.preprocessing import Normalizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
import sklearn.metrics as metrics
from math import sqrt
from sklearn.neural_network import MLPRegressor
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import Pipeline
from sklearn.feature_selection import RFE
In [53]:
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os
import seaborn as sns
from pandas.plotting import scatter_matrix
from pandas import DatetimeIndex
from sklearn.preprocessing import Normalizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
import sklearn.metrics as metrics
from math import sqrt
from sklearn.neural_network import MLPRegressor
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import Pipeline
from sklearn.feature_selection import RFE
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.pipeline import FeatureUnion
from sklearn.linear_model import LogisticRegression
from sklearn.decomposition import PCA