ADS Mid term Project

Energy Data Set

Presented By

Dhairya

Aditya

Ranga

Guided By

Prof. Shrikant

Introduction

- Energy prediction is very important feature in conservation process as the supply and demand aspect can be managed well
- Our presentation focuses on how to predict the next day energy consumption and also highlights various models that have implemented to do this analysis.

Part 1: Review of the Documents

- We have conducted and reviewed the documents have to made our analysis of what are the important features each document has to offer.
- Reviewing also requires us to identify the short coming of the document and in this case inaccuracies in the data have been seen.

DATA FRAME

In [58]:

```
df
```

Out[58]:

	date	Energy_consumed	Appliances	lights	Kitchen_Temp	Kitchen_Hum	LivingRoom_Temp	LivingRoom_Hum	LaundryRoom_Temp	LaundryRoom_Hu
0	2016-01-11 17:00:00	90	60	30	19.890000	47.596667	19.200000	44.790000	19.790000	44.7300
1	2016-01-11 17:10:00	90	60	30	19.890000	46.693333	19.200000	44.722500	19.790000	44.7900
2	2016-01-11 17:20:00	80	50	30	19.890000	46.300000	19.200000	44.626667	19.790000	44.9333
3	2016-01-11 17:30:00	90	50	40	19.890000	46.066667	19.200000	44.590000	19.790000	45.0000
4	2016-01-11 17:40:00	100	60	40	19.890000	46.333333	19.200000	44.530000	19.790000	45.0000
5	2016-01-11 17:50:00	90	50	40	19.890000	46.026667	19.200000	44.500000	19.790000	44.9333
6	2016-01-11 18:00:00	110	60	50	19.890000	45.766667	19.200000	44.500000	19.790000	44.9000
7	2016-01-11 18:10:00	110	60	50	19.856667	45.560000	19.200000	44.500000	19.730000	44.9000
8	2016-01-11 18:20:00	100	60	40	19.790000	45.597500	19.200000	44.433333	19.730000	44.7900
9	2016-01-11 18:30:00	110	70	40	19.856667	46.090000	19.230000	44.400000	19.790000	44.8633
10	2016-01-11 18:40:00	300	230	70	19.926667	45.863333	19.356667	44.400000	19.790000	44.9000
11	2016-01-11 18:50:00	640	580	60	20.066667	46.396667	19.426667	44.400000	19.790000	44.8266

Description of Appliances and Lights column

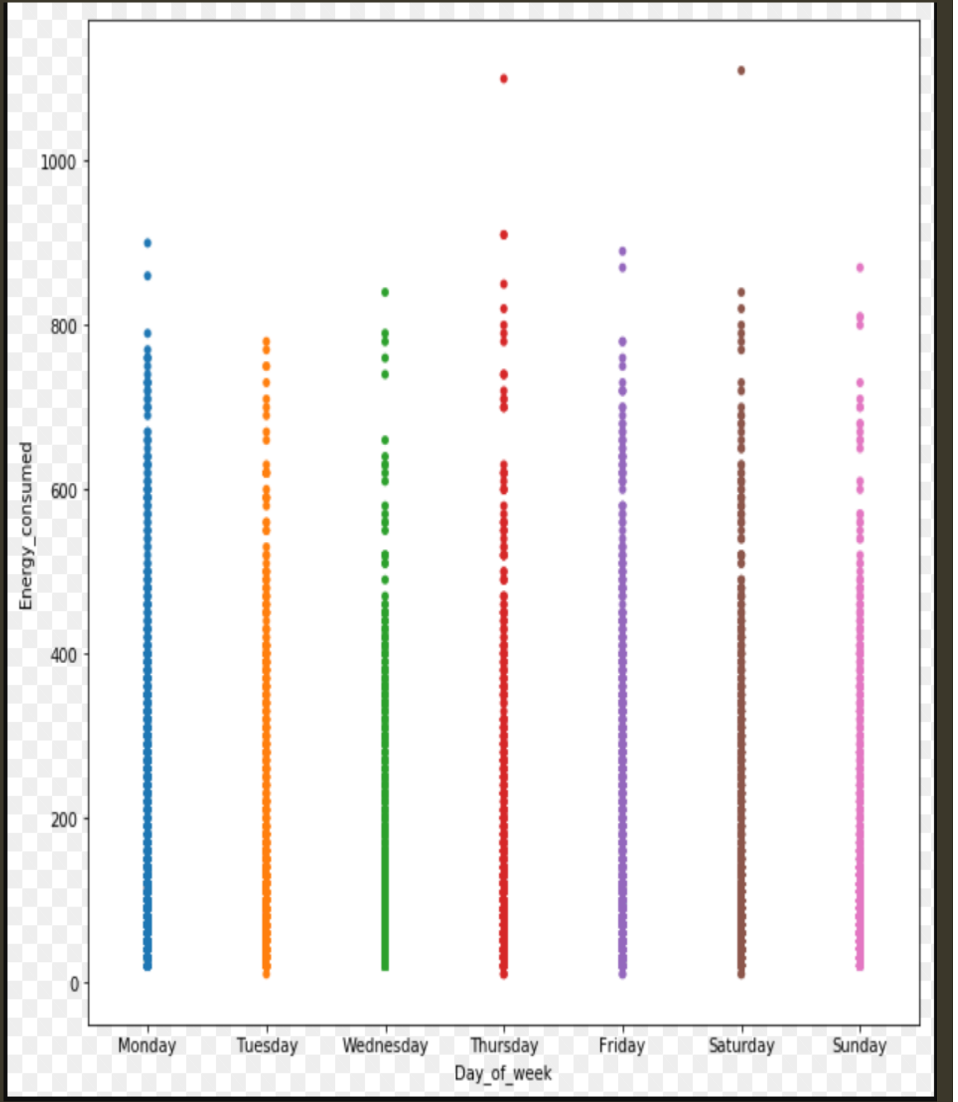
Descriptive analysis for appliances	
N	19735
Min	10
Max	1080
Mean	97.69496
First Quartile	50
Median	60
Third Quartile	100
Standard Deviation	102.5223
Variance	10510.82
Standard Error	0.729793

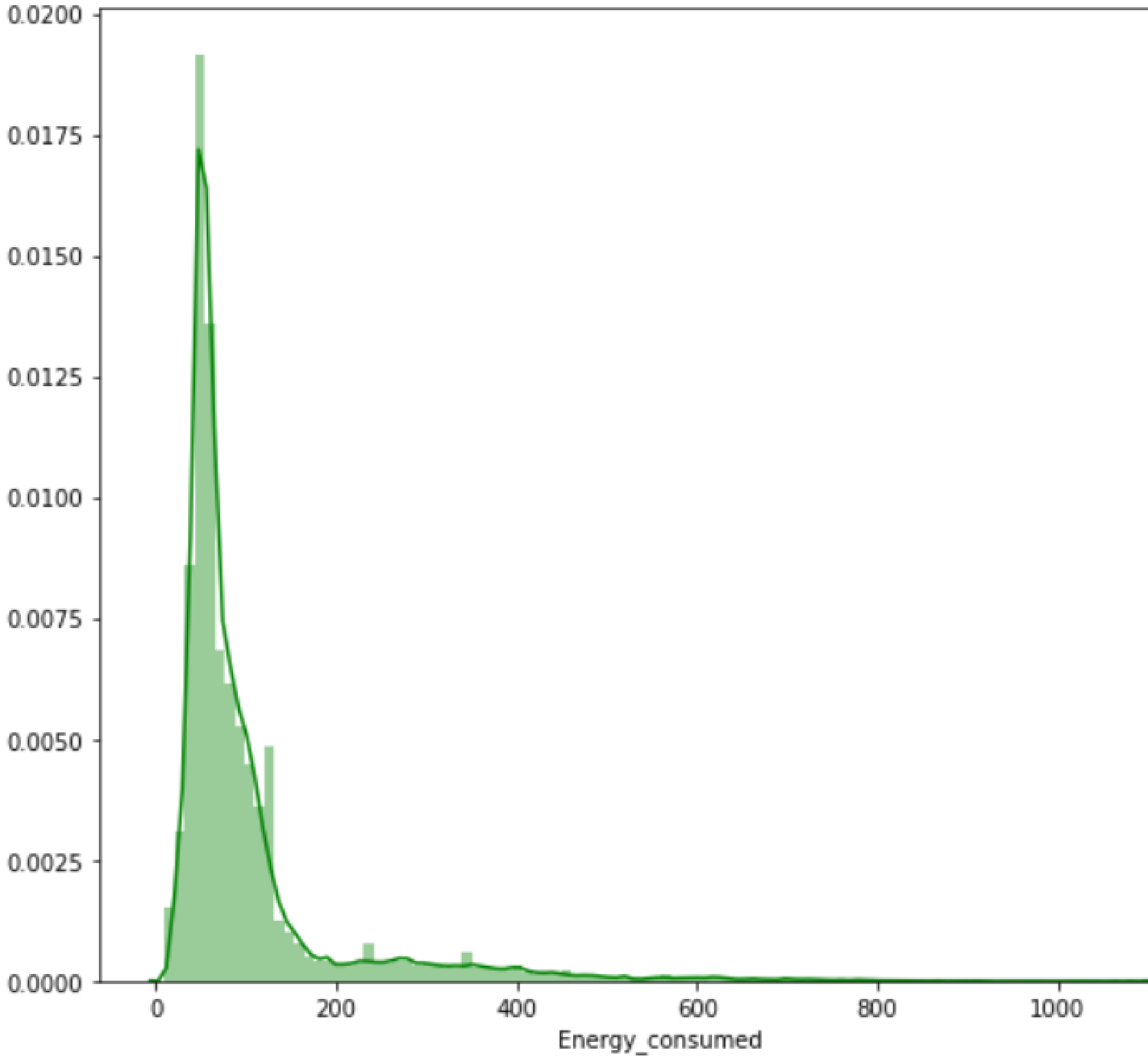
Descriptive analysis for Lights	
N	19735
Min	0
Max	70
Mean	3.801875
First Quartile	0
Median	0
Third Quartile	0
Standard Deviation	7.935787
Variance	62.97671
Standard Error	0.05649

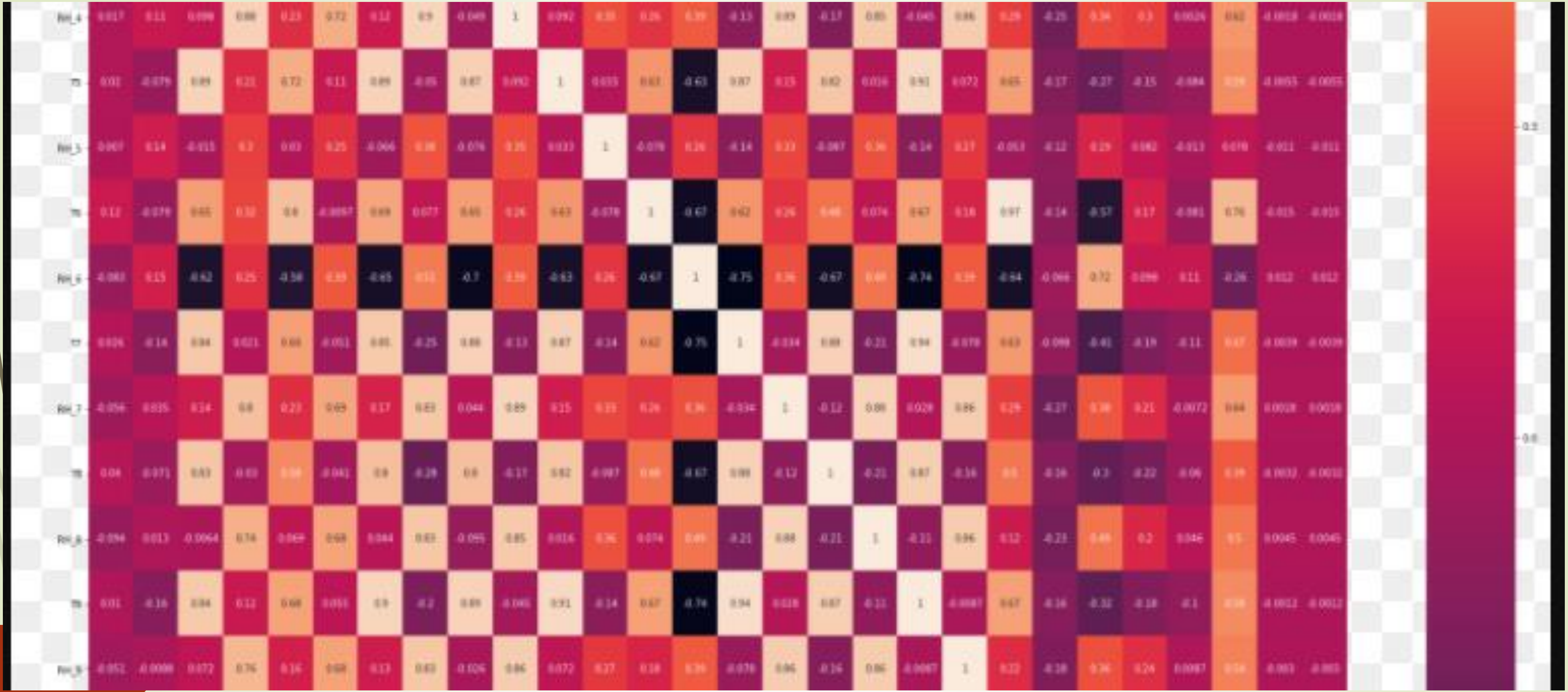
Part2:Exploratory Data Analysis

- A detailed Exploratory analysis of the data set has been done to identify various considerations to identify:
- Correlation between Variables
- To check if the variables are categorical or continuous variables
- Relationship between sample numbers and variables
- Check which variables are independent and which are dependent
- Do a time series analysis to identify trends

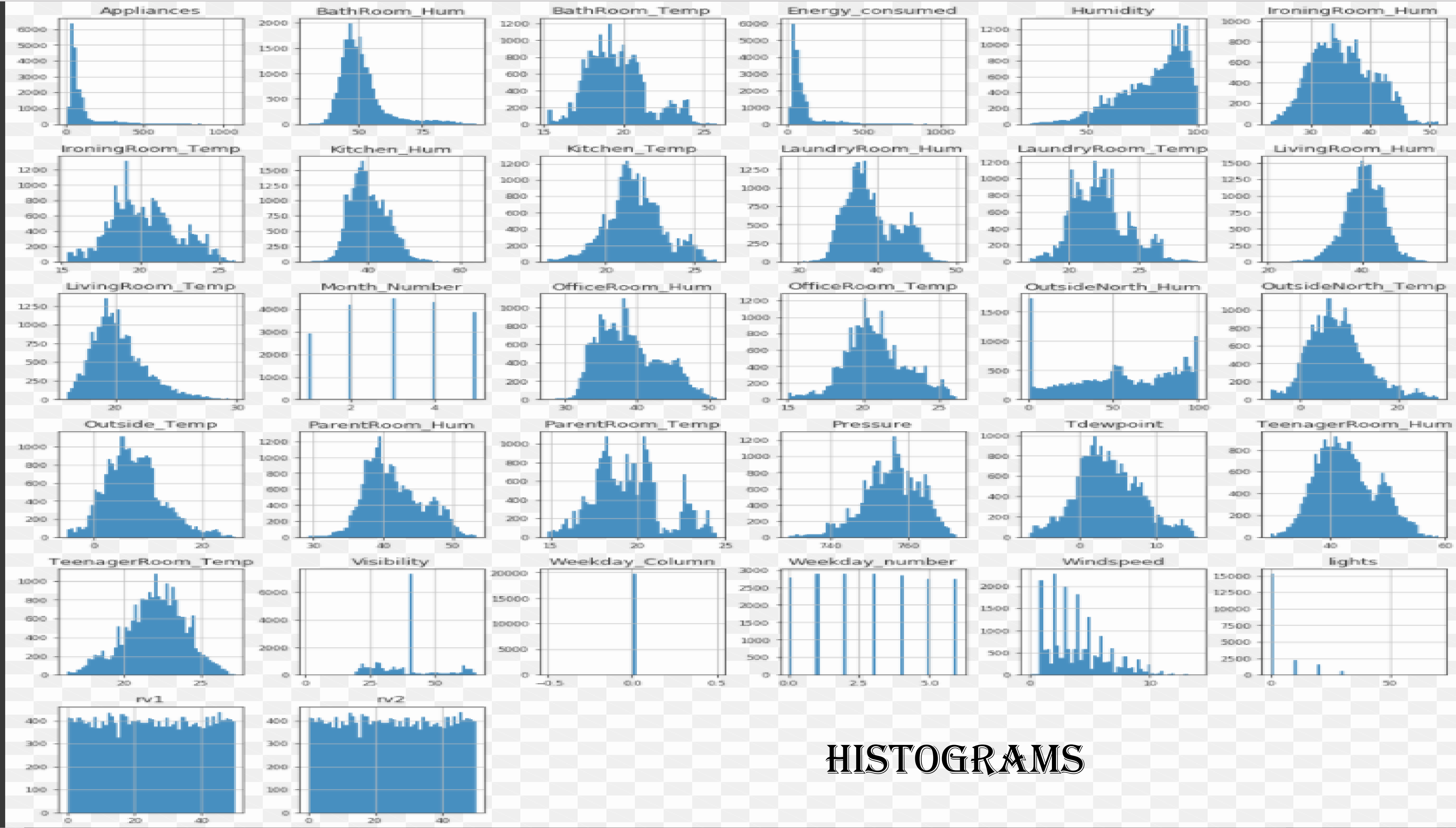
EDA Graphs


1000
800
600
400
200
0
Energy_consumed
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Sunday
Day_of_week


0.0200
0.0175
0.0150
0.0125
0.0100
0.0075
0.0050
0.0025
0.0000
0
200
400
600
800
1000
Energy_consumed



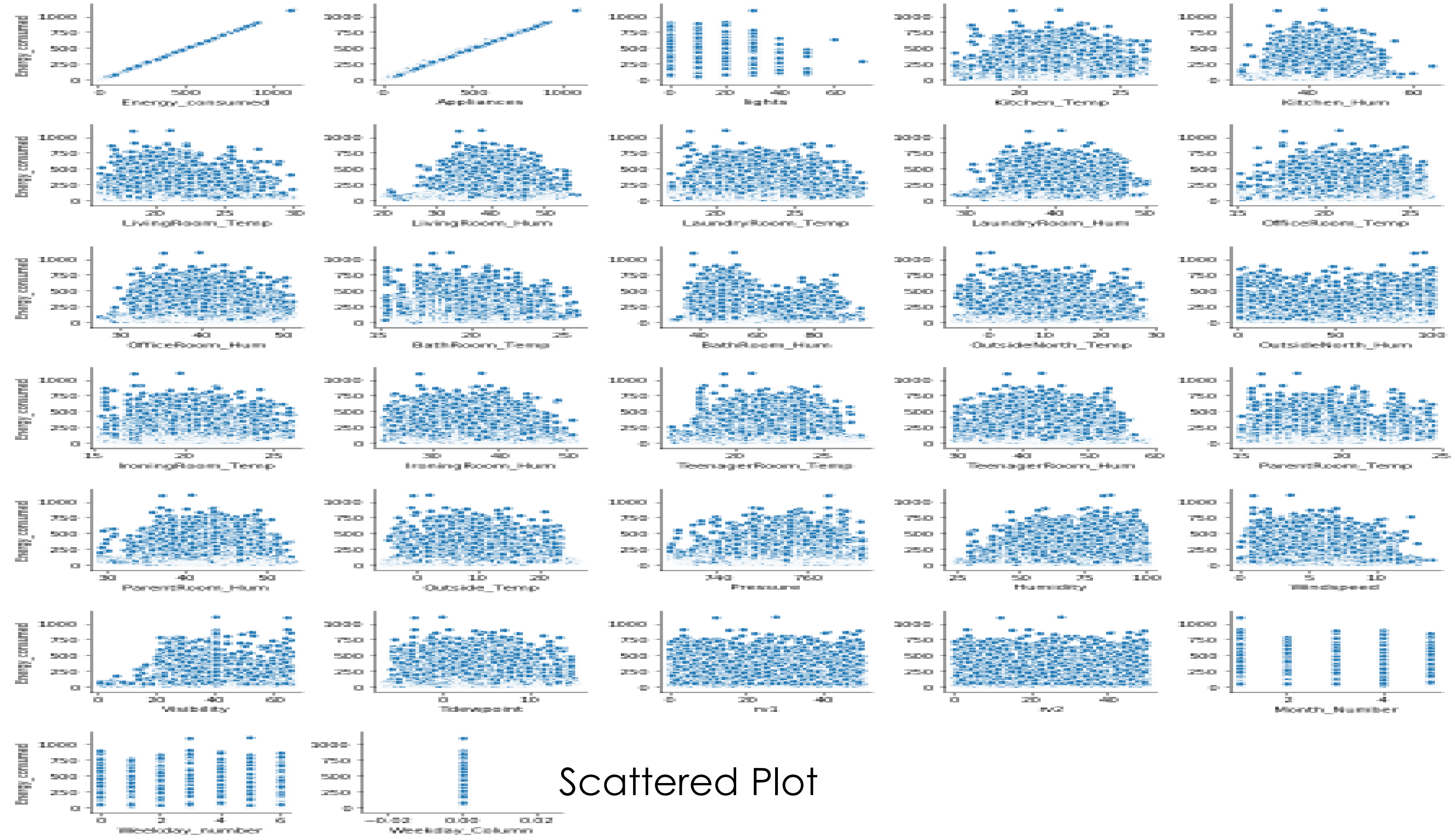
HEAT MAP


Appliances
BathRoom_Hum
BathRoom_Temp
Energy_consumed
Humidity
IroningRoom_Hum
IroningRoom_Temp
Kitchen_Hum
Kitchen_Temp
LaundryRoom_Hum
LaundryRoom_Temp
LivingRoom_Hum
LivingRoom_Temp
Month_Number
OfficeRoom_Hum
OfficeRoom_Temp
OutsideNorth_Hum
OutsideNorth_Temp
Outside_Temp
ParentRoom_Hum
ParentRoom_Temp
Pressure
Tdewpoint
TeenagerRoom_Hum
TeenagerRoom_Temp
Visibility
Weekday_Column
Weekday_number
Windspeed
lights
rv1
rv2

HISTOGRAMS

In [65]:

```
# Scatterplot w.r.t Energy_consumed
for i in range(0, len(df_num.columns), 5):
    sns.pairplot(data=df_num,
                 x_vars=df_num.columns[i:i+5],
```



Scattered Plot

Part3: Feature Engineering

- Addition of new column in the data frame name Energy consumed which is the energy consumed the complete house.(Appliances + Led Lights)
- Eliminating unnecessary variables

Part4: Predictive Algo’s

We have implemented:

- Multi-Linear Regression
- Random Forest
- MLP Regressor

We have divided the training and testing aspect of the data set in 80:20 distribution respectively.

ANN MODEL

	Using all Features		Using Selected Features	
	Training	**Testing**	**Training**	**Testing**
MAE	52.4026	54.1978	59.319	61.837
RMSE	97.5166	102.8480	98.312	104.40
R Squared	0.1093	0.10076	0.088	0.0938
MAPE	115.2379	115.2430	128.82	127.62
Accuracy	0.1093	0.10076	0.88	.0938

C

B

A

Important Features using Boruta

Kitchen_Hum	OutsideNorth_Hum
LivingRoom_Temp	IroningRoom_Hum
LivingRoom_Hum	TeenagerRoom_Hum
LaundryRoom_Hum	TeenagetRoom_Temp
OfficeRoom_Hum	ParentRoom_Temp
BathRoom_Hum	ParentRoom_Hum
OutsideNorth_Temp	Pressure
Humidity	Windspeed
Tdewpoint	

Feature Selection

	coef	std err	t	P>|t|	[0.025	0.975]
const	176.5734	90.297	1.955	0.051	-0.417	353.564
x1	14.4408	0.607	23.779	0.000	13.251	15.631
x2	-11.8819	0.988	-12.024	0.000	-13.819	-9.945
x3	-11.9967	0.627	-19.142	0.000	-13.225	-10.768
x4	7.2320	0.692	10.452	0.000	5.876	8.588
x5	2.4861	0.573	4.340	0.000	1.363	3.609
x6	0.2314	0.087	2.654	0.008	0.061	0.402
x7	5.4992	0.619	8.878	0.000	4.285	6.713
x8	0.3044	0.065	4.681	0.000	0.177	0.432
x9	-2.7674	0.403	-6.862	0.000	-3.558	-1.977
x10	9.1791	0.756	12.136	0.000	7.697	10.662
x11	-5.2951	0.356	-14.874	0.000	-5.993	-4.597
x12	-2.1600	0.399	-5.414	0.000	-2.942	-1.378
x13	-9.4982	1.527	-6.220	0.000	-12.491	-6.505
x14	-0.0339	0.105	-0.324	0.746	-0.239	0.171
x15	-1.2144	0.311	-3.900	0.000	-1.825	-0.604
x16	6.1321	1.489	4.119	0.000	3.214	9.050

Omnibus:	13730.883	Durbin-Watson:	0.566
Prob(Omnibus):	0.000	Jarque-Bera (JB):	192661.331
Skew:	3.274	Prob(JB):	0.00
Kurtosis:	16.835	Cond. No.	9.96e+04

Cross Validation

Cross Validation

In [244]:
```
#Simple K-Fold cross validation. 5 folds.
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression
accuracies = cross_val_score(estimator=LinearRegression(), X=X_test, y=Y_test,cv=10)
```

In [245]:
```
accuracies.mean()
```

Out[245]:
```
0.11888974091758367
```

In [247]:
```
#Simple K-Fold cross validation. 5 folds.
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression
accuracies = cross_val_score(estimator=LinearRegression(), X=X_train, y=Y_train,cv=10)
accuracies.mean()
```

Out[247]:
```
0.10923878824669786
```

Ridge Regression

Ridge Regression

In [251]:
```
from sklearn.linear_model import Ridge
##Training the model
ridgeReg = Ridge(alpha=0.05,normalize=True)
ridgeReg.fit(X_train,Y_train)
pred = ridgeReg.predict(X_test)
##Calculating MSE
MSE = np.mean((pred-Y_test)**2)
print('MSE = ',MSE)
##Calculating Score
print('score = ',ridgeReg.score(X_test,Y_test))
```

```
MSE =  9401.51504916
score =  0.112565229846
```

Lasso Regression

In [250]:
```
from sklearn.linear_model import Lasso
##Training the model
lassoReg = Lasso(alpha=0.005,normalize=True)
lassoReg.fit(X_train,Y_train)
pred = lassoReg.predict(X_test)
##Calculating MSE
MSE = np.mean((pred-Y_test)**2)
print('MSE = ',MSE)
##Calculating Score
print('score = ',lassoReg.score(X_test,Y_test))
```

```
MSE =  9357.74399249
score =  0.116696899839
```

Elastic Net regression

Lasso Regression

Lasso Regression

```
In [250]: from sklearn.linear_model import Lasso
          ##Training the model
          lassoReg = Lasso(alpha=0.005,normalize=True)
          lassoReg.fit(X_train,Y_train)
          pred = lassoReg.predict(X_test)
          ##Calculating MSE
          MSE = np.mean((pred-Y_test)**2)
          print('MSE = ',MSE)
          ##Calculating Score
          print('score = ',lassoReg.score(X_test,Y_test))
```

```
MSE =  9357.74399249
score =  0.116696899839
```

Elastic Net Regression

```
print('MSE = ',MSE)
##Calculating Score
print('score = ',lassoReg.score(X_test,Y_test))
```

```
MSE =  9357.74399249
score =  0.116696899839
```

Elastic Net regression

In [253]:

```
from sklearn.linear_model import ElasticNet
##Training the model
ENReg = ElasticNet(alpha=1,l1_ratio=0.5,normalize=False)
ENReg.fit(X_train,Y_train)
pred = ENReg.predict(X_test)
##Calculating MSE
MSE = np.mean((pred-Y_test)**2)
print('MSE = ',MSE)
##Calculating Score
print('score = ',ENReg.score(X_test,Y_test))
```

```
MSE =  9383.55443961
score =  0.114260581002
```

BIAS - VARIANCE TRADE OFF (LEARNING CURVE)

In [260]:

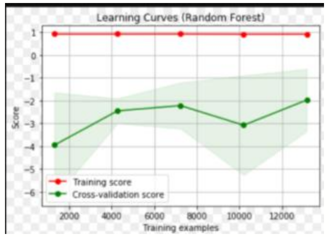
```
#print(_doc_)

import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_digits
from sklearn.model_selection import learning_curve
from sklearn.model_selection import ShuffleSplit


def plot_learning_curve(estimator, title, X, y, ylim=None, cv=None,
```

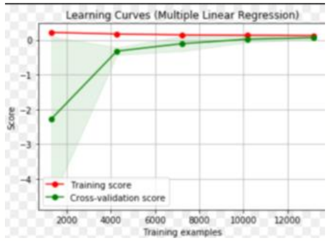
Bias Variance

BIAS - VARIANCE TRADE OFF (LEARNING CURVE)

In [260]:

```
#print(_doc_)

import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_digits
from sklearn.model_selection import learning_curve
from sklearn.model_selection import ShuffleSplit


def plot_learning_curve(estimator, title, X, y, ylim=None, cv=None,
                        n_jobs=1, train_sizes=np.linspace(.1, 1.0, 5)):
    plt.figure()
    plt.title(title)
    if ylim is not None:
        plt.ylim(*ylim)
    plt.xlabel("Training examples")
    plt.ylabel("Score")
    train_sizes, train_scores, test_scores = learning_curve(
        estimator, X, y, cv=cv, n_jobs=n_jobs, train_sizes=train_sizes)
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)
    plt.grid()

    plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                     train_scores_mean + train_scores_std, alpha=0.1,
                     color="r")
    plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                     test_scores_mean + test_scores_std, alpha=0.1, color="g")
    plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
             label="Training score")
    plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
             label="Cross-validation score")

    plt.legend(loc="best")
    return plt


digits = load_digits()
X, y = X, Y


title = "Learning Curves (Multiple Linear Regression)"
# Cross validation with 100 iterations to get smoother mean test and train
# score curves, each time with 20% data randomly selected as a validation set.
cv = ShuffleSplit(n_splits=100, test_size=0.2, random_state=0)

estimator = LinearRegression()
plot_learning_curve(estimator, title, X, y, cv=3, n_jobs=4)
plt.show()
```





Backward Selection

- By using Backward Elimination method we do not get any variable as an important feature

BACKWARD ELIMINATION METHOD

```
In [271]: import statsmodels.api as sm
          def stepwise_selection(X1, y1,
                                 initial_list=[],
                                 threshold_in=0.01,
                                 threshold_out = 0.05,
                                 verbose=True):
                  included = list(initial_list)
                  while True:
                      changed=False
          import statsmodels.api as sm
          def stepwise_selection(X1, y1,
                                 initial_list=[],
                                 threshold_in=0.01,
                                 threshold_out = 0.05,
                                 verbose=True):
              """ Perform a forward-backward feature selection
              based on p-value from statsmodels.api.OLS
              Arguments:
                  X - pandas.DataFrame with candidate features
                  y - list-like with the target
                  initial_list - list of features to start with (column names of X)
                  threshold_in - include a feature if its p-value < threshold_in
                  threshold_out - exclude a feature if its p-value > threshold_out
                  verbose - whether to print the sequence of inclusions and exclusions
              Returns: list of selected features
              Always set threshold_in < threshold_out to avoid infinite looping.
              """
              included = list(initial_list)
              while True:
                  changed=False
                   # backward step
                  model = sm.OLS(y1, sm.add_constant(pd.DataFrame(X1[included]))).fit()
                  # use all coefs except intercept
                  pvalues = model.pvalues.iloc[1:]
                  worst_pval = pvalues.max() # null if pvalues is empty
                  if worst_pval > threshold_out:
                      changed=True
                      worst_feature = pvalues.argmax()
                      included.remove(worst_feature)
                      if verbose:
                          print('Drop {:30} with p-value {:.6}'.format(worst_feature, worst_pval))
                  if not changed:
                      break
              return included

          result = stepwise_selection(X1, y1)

          print('resulting features:')
          print(result)

          resulting features:
          []
```

Forward Selection

Only one variable was obtained using forward Selection method

In [270]:

```python
import statsmodels.api as sm
def stepwise_selection(X1, y1,
                       initial_list=[],
                       threshold_in=0.01,
                       threshold_out = 0.05,
                       verbose=True):
    """ Perform a forward-backward feature selection
    based on p-value from statsmodels.api.OLS
    Arguments:
        X - pandas.DataFrame with candidate features
        y - list-like with the target
        initial_list - list of features to start with (column names of X)
        threshold_in - include a feature if its p-value < threshold_in
        threshold_out - exclude a feature if its p-value > threshold_out
        verbose - whether to print the sequence of inclusions and exclusions
    Returns: list of selected features
    Always set threshold_in < threshold_out to avoid infinite looping.
    """
    included = list(initial_list)
    while True:
        changed=False
        # forward step
        excluded = list(set(X1.columns)-set(included))
        new_pval = pd.Series(index=excluded)
        for new_column in excluded:
            model = sm.OLS(y1, sm.add_constant(X1)).fit()
            new_pval[new_column] = model.pvalues[new_column]
        best_pval = new_pval.min()
        if best_pval < threshold_in:
            best_feature = new_pval.argmin()
            included.append(best_feature)
            changed=True
            if verbose:
                print('Add  {:30} with p-value {:.6}'.format(best_feature, best_pval))
            if not changed:
                break
            return included

result = stepwise_selection(X1, y1)

print('resulting features:')
print(result)
```

```
Add  LaundryRoom_Temp               with p-value 5.4523e-129
resulting features:
['LaundryRoom_Temp']
```

Exhaustive Selection

EXHAUSTIVE SEARCH METHOD

In [272]:
```python
import statsmodels.api as sm
def stepwise_selection(X1, y1,
                       initial_list=[],
                       threshold_in=0.01,
                       threshold_out = 0.05,
                       verbose=True):
    """ Perform a forward-backward feature selection
    based on p-value from statsmodels.api.OLS
    Arguments:
        X - pandas.DataFrame with candidate features
        y - list-like with the target
        initial_list - list of features to start with (column names of X)
        threshold_in - include a feature if its p-value < threshold_in
        threshold_out - exclude a feature if its p-value > threshold_out
        verbose - whether to print the sequence of inclusions and exclusions
    Returns: list of selected features
    Always set threshold_in < threshold_out to avoid infinite looping.
    """
    included = list(initial_list)
    while True:
        changed=False
        # forward step
        excluded = list(set(X1.columns)-set(included))
        new_pval = pd.Series(index=excluded)
        for new_column in excluded:
            model = sm.OLS(y1, sm.add_constant(pd.DataFrame(X1[included+[new_column]]))).fit()
            new_pval[new_column] = model.pvalues[new_column]
        best_pval = new_pval.min()
        if best_pval < threshold_in:
            best_feature = new_pval.argmin()
            included.append(best_feature)
            changed=True
            if verbose:
                print('Add  {:30} with p-value {:.6}'.format(best_feature, best_pval))

        # backward step
        model = sm.OLS(y1, sm.add_constant(pd.DataFrame(X1[included]))).fit()
        # use all coefs except intercept
        pvalues = model.pvalues.iloc[1:]
        worst_pval = pvalues.max() # null if pvalues is empty
        if worst_pval > threshold_out:
            changed=True
            worst_feature = pvalues.argmax()
            included.remove(worst_feature)
            if verbose:
                print('Drop {:30} with p-value {:.6}'.format(worst_feature, worst_pval))
        if not changed:
            break
    return included

result = stepwise_selection(X1, y1)

print('resulting features:')
print(result)
```

```
Add  Humidity                       with p-value 2.22825e-92
Add  Kitchen_Hum                    with p-value 1.2589e-85
```


```python
                print('Drop {:30} with p-value {:.6}'.format(worst_feature, worst_pval))
        if not changed:
            break
    return included

result = stepwise_selection(X1, y1)

print('resulting features:')
print(result)
```



```
Add  Humidity                       with p-value 2.22825e-92
Add  Kitchen_Hum                    with p-value 1.2589e-85
```


```
Add  TeenagerRoom_Hum               with p-value 4.48028e-143
Add  LivingRoom_Hum                 with p-value 1.8353e-101
Add  Month_Number                   with p-value 1.90285e-35
Drop Humidity                       with p-value 0.616498
Add  LaundryRoom_Temp               with p-value 6.44506e-75
Add  ParentRoom_Temp                with p-value 4.20613e-71
Add  LivingRoom_Temp                with p-value 7.6848e-31
Add  TeenagerRoom_Temp              with p-value 1.32161e-32
Add  OutsideNorth_Temp              with p-value 7.97415e-23
Add  Outside_Temp                   with p-value 5.87504e-15
Add  Windspeed                      with p-value 1.05919e-06
Add  OfficeRoom_Temp                with p-value 9.88067e-08
Add  Tdewpoint                      with p-value 7.5834e-08
Add  LaundryRoom_Hum                with p-value 1.27303e-08
Add  ParentRoom_Hum                 with p-value 8.99433e-09
Add  IroningRoom_Hum                with p-value 2.88288e-06
Add  OfficeRoom_Hum                 with p-value 7.25068e-05
Add  BathRoom_Hum                   with p-value 0.00187506
Add  Visibility                     with p-value 0.00280772
Add  Weekday_number                 with p-value 0.00297726
resulting features:
['Kitchen_Hum', 'TeenagerRoom_Hum', 'LivingRoom_Hum', 'Month_Number', 'LaundryRoom_Temp', 'ParentRoom_Temp', 'LivingRoom_Temp', 'TeenagerRoom_Temp', 'OutsideNorth_Temp', 'Outside_Temp', 'Windspeed', 'OfficeRoom_Temp', 'Tdewpoint', 'LaundryRoom_Hum', 'ParentRoom_Hum', 'IroningRoom_Hum', 'OfficeRoom_Hum', 'BathRoom_Hum', 'Visibility', 'Weekday_number']
```

FEATURE SELECTION AFTER EXHAUSTIVE SEARCH

[278]:
```python
## X are the varibles used for predictions
## Y is the target
X = dataset.iloc[:,[7,21,9,31,10,22,8,20,16,24,27,12,29,11,23,19,13,15,28,32]].values
Y = dataset.iloc[:,3].values
Y
```

Out[278]:
```
array([ 90,  90,  80, ..., 280, 430, 440], dtype=int64)
```

In [279]:
```python
## fitting multiple Linear regression to the training set
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=np.random)
regressor.fit(X_train,Y_train)
```

Out[279]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

In [280]:
```python
import statsmodels.formula.api as sm
X = np.append(arr=np.ones((19735,1)).astype(int),values = X,axis = 1)
X_opt = X[:,0:32]
regressor_OLS = sm.OLS(endog = Y,exog = X_opt).fit()
regressor_OLS.summary()
```

Important Features of Exhaustive search

Selected Features of Exhaustive search	
Kitchen_Hum	LivingRoom_Temp
TeenagerRoom_Hum	TeenagerRoom_Temp
LivingRoom_Hum	OutsideNorth_Temp
Month_Number	Outside_Temp
LaundryRoom_Temp	Windspeed
ParentRoom_Temp	LaundryRoom_Hum
Weekday_number	BathRoom_Hum
OfficeRoom_Hum	OfficeRoom_Temp
IroningRoom_Hum	Tdewpoint ParentRoom_Hum
Visibility	

Exhaustive search result with selected features

	Train	Test
MAE	54.3393	55.4967
RMSE	95.3274	98.0463
R2	0.1581	0.1484
MAPE	121.1182	119.9960

TPot : According to Tpot RandomForest is the best model and which is already seen in the RandomForest had the best accuracy score

```
In [16]: tpot = TPOTRegressor(generations=5, population_size=50, verbosity=2)
         tpot.fit(X_train, Y_train)
         print(tpot.score(X_test, Y_test))
```

```
C:\Users\HP\Anaconda32\lib\importlib\_bootstrap.py:219: ImportWarning: can't resolve package from __spec__ or __package__, fall
ing back on __name__ and __path__
  return f(*args, **kwds)

Warning: xgboost.XGBRegressor is not available and will not be used by TPOT.

Generation 1 - Current best internal CV score: -6223.678637476058

Generation 2 - Current best internal CV score: -5605.060340102767

Generation 3 - Current best internal CV score: -5605.060340102767

Generation 4 - Current best internal CV score: -5605.060340102767

Generation 5 - Current best internal CV score: -4752.19162710271


Best pipeline: LassoLarsCV(RandomForestRegressor(input_matrix, bootstrap=False, max_features=0.1, min_samples_leaf=1, min_sampl
es_split=6, n_estimators=100), normalize=True)
-4454.4360946
```

Grid Search

Grid Search

In [31]:
```
from sklearn.grid_search import GridSearchCV
```

Finding the right parameters (like what C or gamma values to use) is a tricky task! his idea of creating a 'grid' of parameters and just trying out all the possible combinations is called a Gridsearch, this method is common enough that Scikit-learn has this functionality built in with GridSearchCV

GridSearchCV takes a dictionary that describes the parameters that should be tried and a model to train.

In [27]:
```
param_grid = {
    'n_estimators': [200, 700],
    'max_features': ['auto', 'sqrt', 'log2']}

CV_rfc = GridSearchCV(estimator=RandomForestRegressor(), param_grid=param_grid, cv= 5)
```

In [30]:
```
CV_rfc.fit(X_train, Y_train)
print(CV_rfc.best_params_)
```

```
{'max_features': 'log2', 'n_estimators': 700}
```

FINAL ANALYSIS OF ALL MODELS

	All features		Features selection(Boruta)		Features selection(Exhaustive Search)	
	Training	Testing	Traning	Testing	Training	Testing
Multiple Linear Regression	0.16	0.14	-0.16	0.13	0.16	0.15
Random Forest	0.94	0.56	0.93	0.54		
MIP Regressor	0.11	0.1	0.088	0.093		
Best Model	All features	Random Forest				

PIPELINE

```
In [28]: from sklearn.ensemble import RandomForestRegressor
         from sklearn.pipeline import Pipeline
         from sklearn import linear_model

         pipeline = Pipeline([

             #('features',feats),
             ('classifier', RandomForestRegressor(random_state = 42))

         ])

         pipeline.fit(X_train, Y_train)
         preds = pipeline.predict(X_test)
         #np.mean(preds == Y_test)
         print(preds)

         [ 377.    86.    33. ...,  187.  224.    58.]

In [29]: pipeline.score(X_test,Y_test)

Out[29]: 0.53588686568405097

In [30]: pipeline.score(X_train,Y_train)

Out[30]: 0.911762477117952
```

THANK YOU!!!!!!