hw3

March 5, 2018

```
In [138]: import numpy as np
          import xlsxwriter
          import pandas as pd
          from sklearn.linear_model import LinearRegression, Ridge, Lasso, ElasticNet
          from sklearn.ensemble import RandomForestClassifier, GradientBoostingRegressor
          from sklearn.svm import SVR
          from sklearn.feature_selection import SelectKBest, f_regression
          from sklearn.model_selection import train_test_split, cross_val_score
          from sklearn.preprocessing import StandardScaler, Imputer, PolynomialFeatures, Labell
          from sklearn.model_selection import GridSearchCV
          from sklearn.pipeline import make_pipeline


          data2015 = pd.read_excel('/Users/jingyu/Desktop/2015 FE Guide-for DOE-Mobility Ventu
          data2016 = pd.read_excel('/Users/jingyu/Desktop/2016 FE Guide for DOE-OK to release-
          data2017 = pd.read_excel('/Users/jingyu/Desktop/2017 FE Guide for DOE-release dates
          data2018 = pd.read_excel('/Users/jingyu/Desktop/2018 FE Guide for DOE-release dates


          trainData = data2015.append([data2016, data2017],ignore_index=True)

          y_train = trainData['Comb Unrd Adj FE - Conventional Fuel'].to_frame()
          X_train = trainData.drop(columns=['Comb Unrd Adj FE - Conventional Fuel'])
          y_test = data2018['Comb Unrd Adj FE - Conventional Fuel'].to_frame()
          X_test = data2018.drop(columns=['Comb Unrd Adj FE - Conventional Fuel'])
```

ClarificationBecause after onehotencoding, the categorical features got quite large amount, all models don't use GridResearch to tune the parameters due to the large time consuming.

1 Task 1

1.1 1. Four linear models are applied in this task:

1. Linear Regression
2. Ridge
3. Lasso

4. Elastic Net

The best model is Linear Regression.

1.2 2. Pre-processing

1. As required, first delete columns which have a strong correaltion with y: any columns has words ['EPA','CO2','Smog','Guzzler','FE','MPG','Cost','Rating','Range']

2. Drop columns with more than 1/3 of records are nan, since these columns don't have some useful information for our prediction.

The left columns will be divided into continuous columns and categorical columns. Continuous Columns:

Model Year Index (Model Type Index) Eng Displ # Cyl # Gears Max Ethanol % - Gasoline Intake Valves Per Cyl Exhaust Valves Per Cyl Carline Class $ You Spend over 5 years (increased amount spent in fuel costs over 5 years - on label)

For these columns:

1. Impute with mean.

2. Scaling with standardscaler.

Categorical Columns:

Mfr Name Division Carline Verify Mfr Cd Transmission Air Aspiration Method Desc Trans Trans Desc Lockup Torque Converter Trans Creeper Gear Drive Sys Drive Desc Fuel Usage - Conventional Fuel Fuel Usage Desc - Conventional Fuel Fuel Unit - Conventional Fuel Fuel Unit Desc - Conventional Fuel Descriptor - Model Type (40 Char or less) Carline Class Desc Calc Approach Desc Release Date Unique Label? Label Recalc? Suppressed? Police/Emerg? Cyl Deact? Var Valve Timing? Var Valve Timing Desc Var Valve Lift? Fuel Metering Sys Cd Fuel Metering Sys Desc Camless Valvetrain (Y or N) Oil Viscosity Stop/Start System (Engine Management System) Code Stop/Start System (Engine Management System) Description

For these columns:

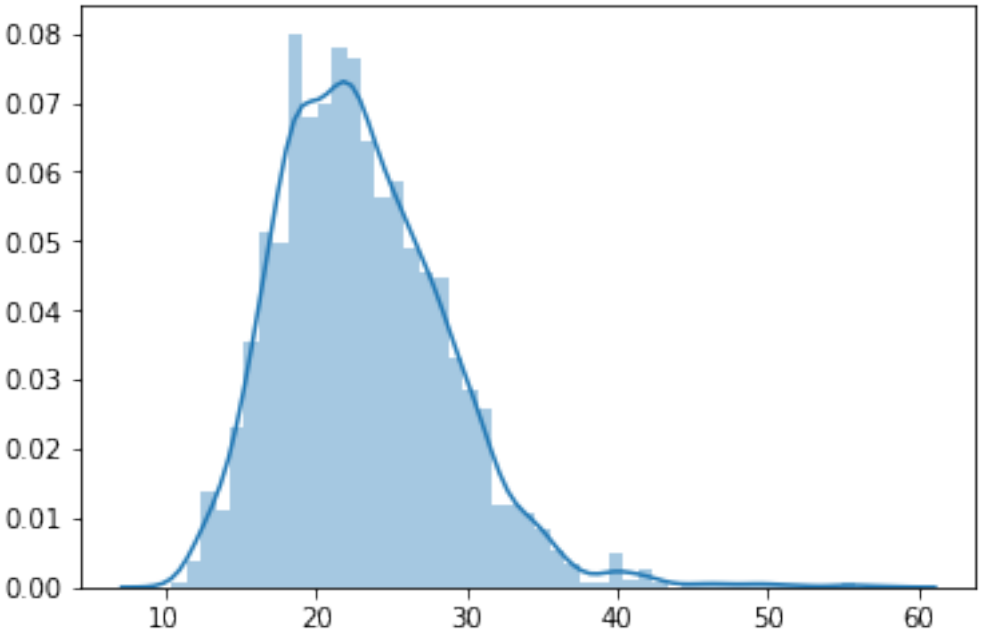
1. Use get_dummies to do onehotencoding.

```
In [139]: def ColumnsType(X_train):
              ### divide the columns into continuous and categorical
              CategoricalColumns = list()
              ContinuousColumns = list()
              for column in X_train.columns:
                  if type(X_train[column].dropna().reset_index(drop=True)[0]) != np.int64 and t
                      CategoricalColumns.append(column)
                  else:
                      ContinuousColumns.append(column)
              return ContinuousColumns,CategoricalColumns
```

2 EDVA

Show the distribution of y_train

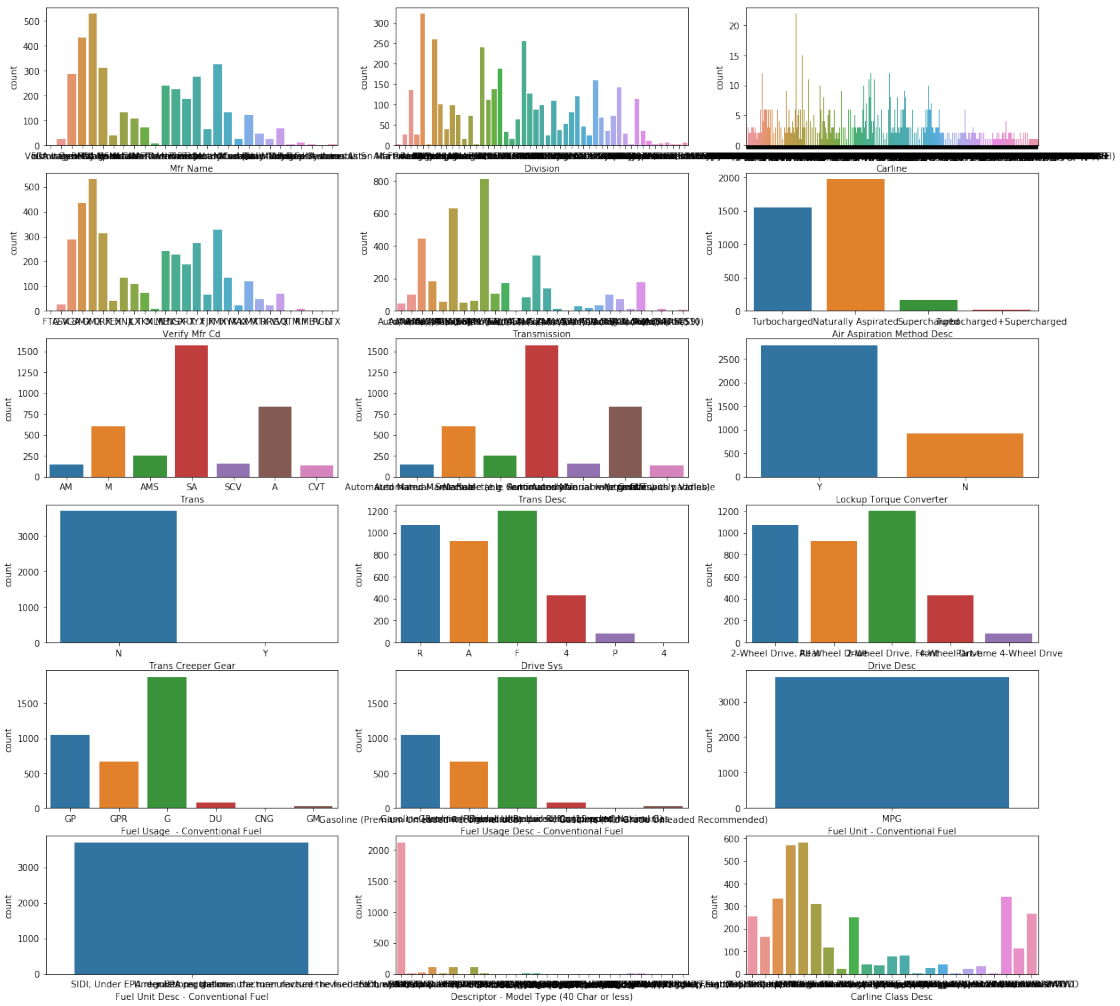
```
In [140]: import seaborn as sns
          sns.distplot(y_train)

Out[140]: <matplotlib.axes._subplots.AxesSubplot at 0x115ac26a0>
```



Then, we see the count histigram for the Categorical Features

```
In [141]: import matplotlib.pyplot as plt
          fig, axes = plt.subplots(6,3,figsize=(20,20))
          k=0
          for i in range(6):
              for j in range(3):
                  sns.countplot(X_train[CategoricalColumns[k]],ax=axes[i,j])
                  k+=1

          plt.suptitle("Categorical Features", fontsize=30)

Out[141]: Text(0.5,0.98,'Categorical Features')
```

Categorical Features



Since Continuous Columns has some missing value, we need to deal with them. so we just process the data

```
In [145]: def preprocessing(X_train):

              ### delete direct measurement columns
              keywords=['EPA','CO2','Smog','Guzzler','FE','MPG','Cost','Rating','Range']
              for word in keywords:
                  for column in X_train.columns:
                      if word in column:
                          del X_train[column]
              print(X_train.shape)

              ### drop columns with the number of value more than 2/3 times of the whole datas
```

```
    X_train = X_train.dropna(thresh=len(X_train)*(2/3), axis=1)
    ####### remove value 'Mod'


    modColumns=['MFR Calculated Gas Guzzler MPG ','FE Rating (1-10 rating on Label)'
    for column in modColumns:
        try:
            X_train[column] = X_train[column].replace(to_replace='Mod',value=np.nan)
        except:
            continue
    print('strat to detect categorical columns')
    print(X_train.shape)

    return X_train


def imputeScalOHE(X_train,X_test,ContinuousColumns,CategoricalColumns):
    '''
    OneHotEncoding on categorical columns;
    Imputation and Scaling on continuous columns
    '''
    X_train_con = X_train[ContinuousColumns]
    X_train_cat = X_train[CategoricalColumns]
    del X_train_cat['Release Date']
    X_test_con = X_test[ContinuousColumns]
    X_test_cat = X_test[CategoricalColumns]
    del X_test_cat['Release Date']
    #####onehotencoding
    X_train_cat = pd.get_dummies(X_train_cat)
    X_test_cat = pd.get_dummies(X_test_cat)
    # Get missing columns in the training test
    missing_cols = set( X_train_cat.columns ) - set(X_test_cat.columns )
    # Add a missing column in test set with default value equal to 0
    for column in missing_cols:
        X_test_cat[column] = 0
    # Ensure the order of column in the test set is in the same order than in train
    X_test_cat = X_test_cat[X_train_cat.columns]
    #####impute
    imputer = Imputer()
    imputer.fit(X_train_con)
    X_train_con_imputed = imputer.transform(X_train_con)
    X_test_con_imputed = imputer.transform(X_test_con)
    #####scaling
    scaler = StandardScaler()
    scaler.fit(X_train_con_imputed)
    X_train_con_scaled_imputed = scaler.transform(X_train_con_imputed)
    X_test_con_scaled_imputed = scaler.transform(X_test_con_imputed)
```

```
            X_train_ISO =  np.concatenate((X_train_con_scaled_imputed,X_train_cat.as_matrix()
            X_test_ISO = np.concatenate((X_test_con_scaled_imputed,X_test_cat.as_matrix()),ax

            ConColsNum, CatColsNum = X_train_con.shape[1], X_train_cat.shape[1]

            ##### ISO stands for Imputation Standardlization and Onehotencoding
            return X_train_ISO, X_test_ISO, ConColsNum, CatColsNum
```

```
In [146]: X_train_prep = preprocessing(X_train)
          X_test_prep = preprocessing(X_test)
          ContinuousColumns, CategoricalColumns = ColumnsType(X_train_prep)


          X_train_ISO,X_test_ISO,ConColsNum,CatColsNum = imputScalOHE(X_train_prep,X_test_prep


          LR=LinearRegression().fit(X_train_ISO,y_train)
          RG=Ridge().fit(X_train_ISO,y_train)
          LA=Lasso().fit(X_train_ISO,y_train)
          EN=ElasticNet().fit(X_train_ISO,y_train)

          print('LRScore:{}\nRidgeScore:{}\nLassoScore:{}\nElasticNetScore:{}'.format(LR.score
```

```
(3701, 100)
strat to detect categorical columns
(3701, 44)
(1227, 100)
strat to detect categorical columns
(1227, 44)
LRScore:0.9485945527947737
RidgeScore:0.9454906815505151
LassoScore:0.537687474274948
ElasticNetScore:0.5668373813115083
```

Without the GridResearch to find the best hyperparameters, the linear regression model already got mor than 0.94 R^2 value.

3 Task 2

Beside the features we used in task 1, in this task, we do the polynomial features on the continuous features.

```
In [147]: def polyFeatures(X, ConColsNum, CatColsNum):
              X_con = X[:,:ConColsNum]
              X_cat = X[:,:(CatColsNum+ConColsNum)]

              poly = PolynomialFeatures()
```

```
            X_con_poly = poly.fit_transform(X_con)
            scaler = StandardScaler()
            scaler.fit(X_con_poly)
            X_con_poly_scaled = scaler.transform(X_con_poly)
            X_poly =  np.concatenate((X_con_poly_scaled,X_cat),axis=1)

            return X_poly
```

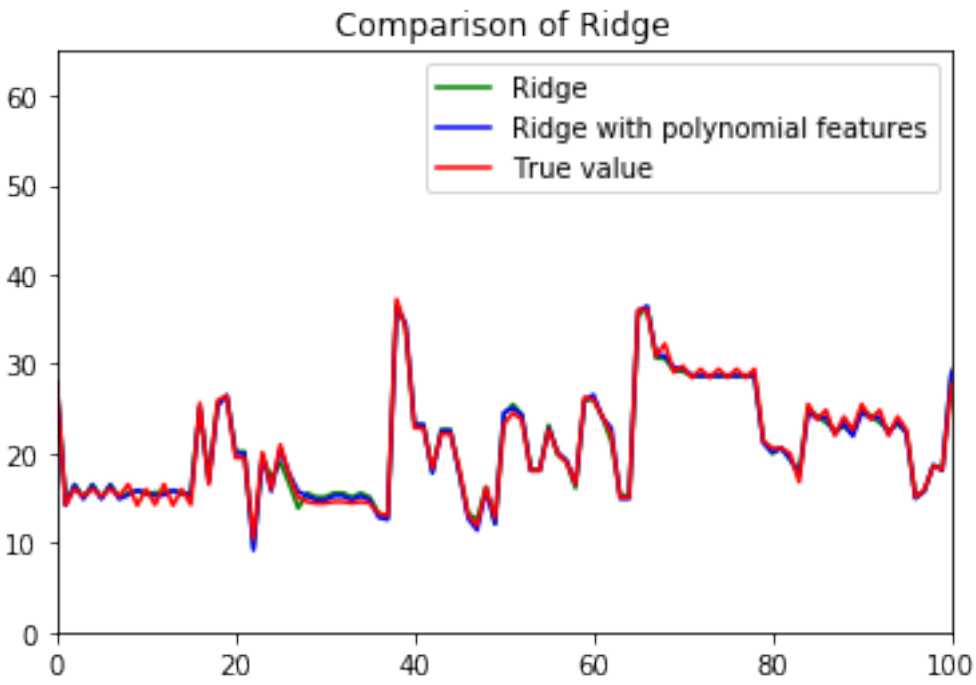
In [148]:
```
X_train_poly = polyFeatures(X_train_ISO,ConColsNum, CatColsNum)
X_test_poly = polyFeatures(X_test_ISO,ConColsNum, CatColsNum)


LR_poly=LinearRegression().fit(X_train_poly,y_train)
RG_poly=Ridge().fit(X_train_poly,y_train)
LA_poly=Lasso().fit(X_train_poly,y_train)
EN_poly=ElasticNet().fit(X_train_poly,y_train)

print('LRScore:{}\nRidgeScore:{}\nLassoScore:{}\nElasticNetScore:{}'.format(LR_poly.s
```

```
LRScore:-2.945715159844874e+22
RidgeScore:0.9458534218171113
LassoScore:0.5561793848668073
ElasticNetScore:0.6032948265062534
```

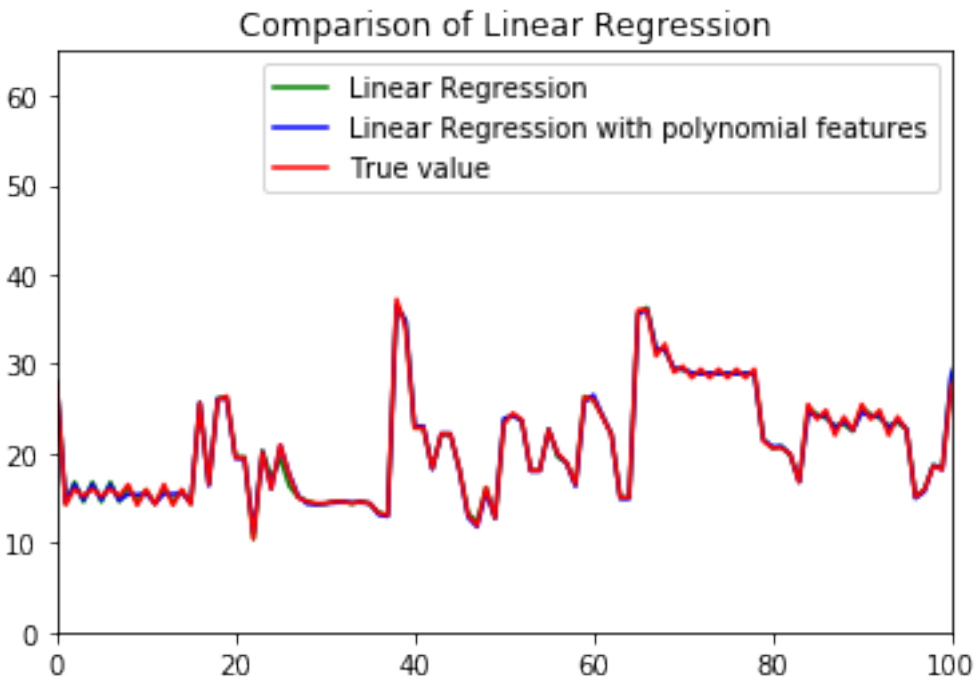
4 EDAV

We used the best score Ridge model to show which one is better.

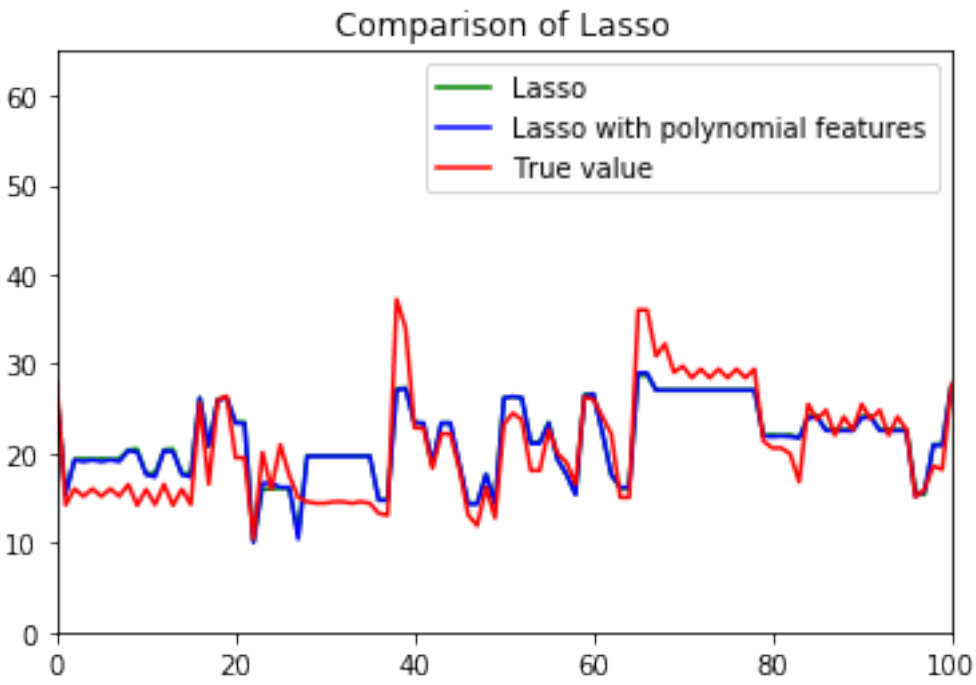
In [149]:
```
lRL = plt.plot(RG.predict(X_train_ISO),"g", label = "Ridge")
lRP = plt.plot(RG_poly.predict(X_train_poly),"b", label = "Ridge with polynomial feat
lRR = plt.plot(y_train,'r', label = "True value")
plt.title('Comparison of Ridge')
plt.xlim(0, 100)
plt.ylim(0,65)
plt.legend()
plt.show()
```



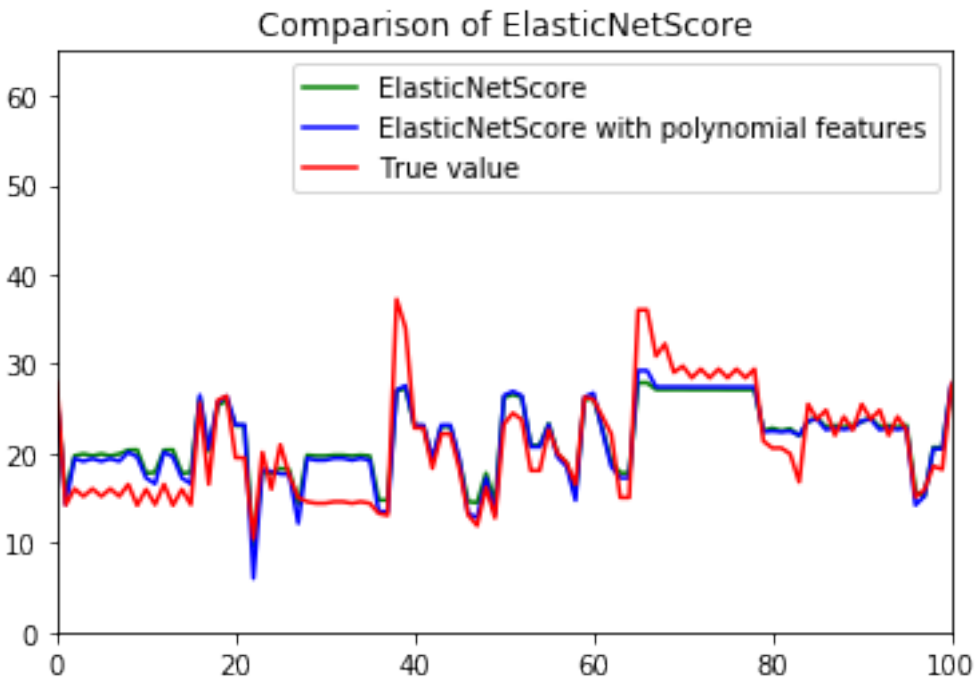

```
In [150]: lR_L = plt.plot(LR.predict(X_train_ISO),"g", label = "Linear Regression")
          lR_P = plt.plot(LR_poly.predict(X_train_poly),"b", label = "Linear Regression with po
          LRR = plt.plot(y_train,'r', label = "True value")
          plt.title('Comparison of Linear Regression')
          plt.xlim(0, 100)
          plt.ylim(0,65)
          plt.legend()
          plt.show()
```




```
In [151]: lL_L = plt.plot(LA.predict(X_train_ISO),"g", label = "Lasso")
          lL_P = plt.plot(LA_poly.predict(X_train_poly),"b", label = "Lasso with polynomial fea
          LRR = plt.plot(y_train,'r', label = "True value")
          plt.title('Comparison of Lasso')
          plt.xlim(0, 100)
          plt.ylim(0,65)
          plt.legend()
          plt.show()
```



In [152]:
```
lE_L = plt.plot(EN.predict(X_train_ISO),"g", label = "ElasticNetScore")
lE_P = plt.plot(EN_poly.predict(X_train_poly),"b", label = "ElasticNetScore with poly
LRR = plt.plot(y_train,'r', label = "True value")
plt.title('Comparison of ElasticNetScore')
plt.xlim(0, 100)
plt.ylim(0,65)
plt.legend()
plt.show()
```



From the result, the polynomial features do help to improve the perfermance of Ridge, Lasso and ElasticNetScore. But it does extremely bad on Linear Regression. The reason might be, that the Linear Regression doesn't have regularization term to help get rid of some useless features. And from the graph, there is just a little bit change between polynomial features and linear features for all of them.

5 Task 3

For this task, we still use the features we use in task 1. And model we used here are GradientBoosting and SVM. Finally, the GradientBoosting has a better perfermance.

5.1 GradientBoosting

```
In [153]: GB = GradientBoostingRegressor().fit(X_train_ISO, y_train)
          print("Gradient Boosting score: {}".format(GB.score(X_test_ISO, y_test)))
```

```
/Users/jingyu/anaconda/envs/python3/lib/python3.5/site-packages/sklearn/utils/validation.py:578
  y = column_or_1d(y, warn=True)


Gradient Boosting score: 0.9382604353309978
```

5.2 SVM

```
In [154]: SVR = SVR().fit(X_train_ISO, y_train)
          print("SVM grid search score: {}".format(SVR.score(X_test_ISO, y_test)))
```

```
/Users/jingyu/anaconda/envs/python3/lib/python3.5/site-packages/sklearn/utils/validation.py:578
  y = column_or_1d(y, warn=True)


SVM grid search score: 0.5993519608175232
```

6 Task 4

In this task, we use best model, Linear Regression to show how to select the parameters. There are different ways to do that.

1. use SelectKBest to select best features.

```
In [157]: select = SelectKBest(k=20, score_func=f_regression)
          select.fit(X_train_ISO, y_train)
          X_train_sub = select.transform(X_train_ISO)
          X_test_sub = select.transform(X_test_ISO)
          LR_selected = LinearRegression().fit(X_train_sub, y_train)
          print("Test score: {}".format(LR_selected.score(X_test_sub,y_test)))

Test score: 0.690049698820539


/Users/jingyu/anaconda/envs/python3/lib/python3.5/site-packages/sklearn/utils/validation.py:578
  y = column_or_1d(y, warn=True)
/Users/jingyu/anaconda/envs/python3/lib/python3.5/site-packages/sklearn/feature_selection/univa
  corr /= X_norms
/Users/jingyu/anaconda/envs/python3/lib/python3.5/site-packages/sklearn/feature_selection/univa
  F = corr ** 2 / (1 - corr ** 2) * degrees_of_freedom
/Users/jingyu/anaconda/envs/python3/lib/python3.5/site-packages/scipy/stats/_distn_infrastructu
  return (self.a < x) & (x < self.b)
/Users/jingyu/anaconda/envs/python3/lib/python3.5/site-packages/scipy/stats/_distn_infrastructu
  return (self.a < x) & (x < self.b)
/Users/jingyu/anaconda/envs/python3/lib/python3.5/site-packages/scipy/stats/_distn_infrastructu
  cond2 = cond0 & (x <= self.a)
```

2. Since this is a linear model, we can just choose the features whose coefficent has a high absolute value than 2.

```
In [158]: selectedColumns = (abs(LR.coef_)>2)[0,:]

In [159]: X_train_ISO,X_test_ISO,ConColsNum,CatColsNum = imputScalOHE(X_train_prep,X_test_prep
          X_train_selected = X_train_ISO[:,selectedColumns]
          X_test_selected = X_test_ISO[:,selectedColumns]

In [160]: LR_selected = LinearRegression().fit(X_train_selected,y_train)
          print(LR_selected.score(X_test_selected, y_test))
```

```
0.8716107394381741
```

Thus, for this model, remove some irrelevant features won't help.