import libraries

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.model_selection import train_test_split
import numpy as np

# import train_test_split
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestRegressor
```

In [2]:
```
import warnings
warnings.filterwarnings('ignore')

pd.set_option('display.max_rows', 20)
pd.set_option('display.max_columns', 500)
# pd.set_option('display.width', 1000)
```

In [3]:
```
classForAnalysis = 2
```

In [4]:
```
# data exploration
import pandas as pd
#df = pd.read_csv('../nhanes_input_data/newdietaryIntakeDataForClassificationAndAnalysi
df = pd.read_csv('./nhanes_output_data/classifiedGroups/2022-12-01/' + str(classForAnal
df.head()
```

Out[4]:

	RIDAGEYR_Age_in_years_at_screening	URDACT_Albumin_creatinine_ratio_mg_g	DataYear	SEQN - Respondent sequence number
0	3	3.19	2017-2018	101691.0
1	3	3.19	2017-2018	101691.0
2	3	3.19	2017-2018	101691.0
3	3	3.19	2017-2018	101691.0

	RIDAGEYR_Age_in_years_at_screening	URDACT_Albumin_creatinine_ratio_mg_g	DataYear	SEQN - Respondent sequence number
4	3	3.19	2017-2018	101691.0

In [5]:
```
'./nhanes_output_data/classifiedGroups/2022-12-01/' + str(classForAnalysis) + '_dietary
```

Out[5]:
```
'./nhanes_output_data/classifiedGroups/2022-12-01/2_dietaryIntakeDataForClassificationAnd
AnalysisData.csv'
```

In [6]:
```
df.shape
```

Out[6]:
```
(65637, 87)
```

In [7]:
```
original_acr = df['URDACT_Albumin_creatinine_ratio_mg_g']
```

In [8]:
```
# Create categories for the severity of ACR
```

In [9]:
```
df.columns[:10]
```

Out[9]:
```
Index(['RIDAGEYR_Age_in_years_at_screening',
       'URDACT_Albumin_creatinine_ratio_mg_g', 'DataYear',
       'SEQN - Respondent sequence number',
       'WTDRD1 - Dietary day one sample weight',
       'WTDR2D - Dietary two-day sample weight',
       'DR1ILINE - Food/Individual component number',
       'DR1DRSTZ - Dietary recall status', 'DR1EXMER - Interviewer ID code',
       'DRABF - Breast-fed infant (either day)'],
      dtype='object')
```

In [10]:
```
# convert str values to int using the scikit-learn encoder : acr_category
```

In [11]:
```
st = df[

    [
         'DR1IKCAL - Energy (kcal)'
        , 'DR1IPROT - Protein (gm)'
        , 'DR1ICARB - Carbohydrate (gm)'
        , 'DR1ISUGR - Total sugars (gm)'
        , 'DR1IFIBE - Dietary fiber (gm)'
        , 'DR1ITFAT - Total fat (gm)'
        , 'DR1ISFAT - Total saturated fatty acids (gm)'
        , 'DR1IMFAT - Total monounsaturated fatty acids (gm)'
        , 'DR1IPFAT - Total polyunsaturated fatty acids (gm)'
        , 'DR1ICHOL - Cholesterol (mg)'
```

```
            ,'URDACT_Albumin_creatinine_ratio_mg_g'
        ]
    ]
```

In [12]:
```
st.columns
```

Out[12]:
```
Index(['DR1IKCAL - Energy (kcal)', 'DR1IPROT - Protein (gm)',
       'DR1ICARB - Carbohydrate (gm)', 'DR1ISUGR - Total sugars (gm)',
       'DR1IFIBE - Dietary fiber (gm)', 'DR1ITFAT - Total fat (gm)',
       'DR1ISFAT - Total saturated fatty acids (gm)',
       'DR1IMFAT - Total monounsaturated fatty acids (gm)',
       'DR1IPFAT - Total polyunsaturated fatty acids (gm)',
       'DR1ICHOL - Cholesterol (mg)', 'URDACT_Albumin_creatinine_ratio_mg_g'],
      dtype='object')
```

In [13]:
```
import pandas as pd
import numpy as np

def clean_dataset(df):
    assert isinstance(df, pd.DataFrame), "df needs to be a pd.DataFrame"
    df.dropna(inplace=True)
    indices_to_keep = ~df.isin([np.nan, np.inf, -np.inf]).any(1)
    return df[indices_to_keep].astype(np.float64)
```

In [14]:
```
st = clean_dataset(st);
```

In [15]:
```
st.replace([np.inf, -np.inf], np.nan, inplace=True)
```

In [16]:
```
st.columns
```

Out[16]:
```
Index(['DR1IKCAL - Energy (kcal)', 'DR1IPROT - Protein (gm)',
       'DR1ICARB - Carbohydrate (gm)', 'DR1ISUGR - Total sugars (gm)',
       'DR1IFIBE - Dietary fiber (gm)', 'DR1ITFAT - Total fat (gm)',
       'DR1ISFAT - Total saturated fatty acids (gm)',
       'DR1IMFAT - Total monounsaturated fatty acids (gm)',
       'DR1IPFAT - Total polyunsaturated fatty acids (gm)',
       'DR1ICHOL - Cholesterol (mg)', 'URDACT_Albumin_creatinine_ratio_mg_g'],
      dtype='object')
```

In [17]:
```
acr_preserved = st['URDACT_Albumin_creatinine_ratio_mg_g']

st_norm = (st - st.mean())/ (st.max() - st.min())
st_norm = clean_dataset(st_norm);

# st_norm['acr_preserved'] = acr_preserved
```

Using ACR category as the target

Apply Machine Learning

Linear Regression

Bayesian

RandomForest Regression

Linear Regression : Actual Values

In [18]:
```
# Split data into Train and Test
# Use 20% of dataset as testing data
```

In [19]:
```
#!pip install scikit-learn numpy
```

In [20]:
```
from sklearn.linear_model import LinearRegression

y = st['URDACT_Albumin_creatinine_ratio_mg_g']
X = st.drop(columns=['URDACT_Albumin_creatinine_ratio_mg_g'])

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=
len(y_train), len(y_test), len(X_train), len(X_test)

# create the model
model = LinearRegression().fit(X_train, y_train)

train_predicted = model.predict(X_train)
test_predicted = model.predict(X_test)

train_mse = mean_squared_error(y_train, train_predicted)
test_mse = mean_squared_error(y_test, test_predicted)

print('MSE train data, MSE test data', train_mse, test_mse)
print('RMSE train data, RMSE test data', np.sqrt(np.absolute(train_mse)),  np.sqrt(np.a
print('R2 train data, R2 test data', r2_score(y_train, train_predicted), r2_score(y_tes
print('\n\n');
print('Regression Coefficients on Actual Data (not normalized), 20% as test data');
model.coef_
```

```
MSE train data, MSE test data 36.697082769282055 36.30923149244877
RMSE train data, RMSE test data 6.057811714578298 6.025714189409316
R2 train data, R2 test data 0.003712433738775389 0.0049098683039341395


Regression Coefficients on Actual Data (not normalized), 20% as test data
```

Out[20]:
```
array([-0.00402676,  0.00678259,  0.01366591, -0.00915826, -0.01712812,
       -0.05309468,  0.12367174,  0.07339351,  0.05747781, -0.00168766])
```

In [21]:
```
l = ( round(x,2) for x in list(model.coef_))
list(l)
```

Out[21]:
```
[-0.0, 0.01, 0.01, -0.01, -0.02, -0.05, 0.12, 0.07, 0.06, -0.0]
```

Linear Regression : Normalized Values

In [22]:
```
y = st_norm ['URDACT_Albumin_creatinine_ratio_mg_g']
X = st_norm.drop(columns=['URDACT_Albumin_creatinine_ratio_mg_g'])

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=
len(y_train), len(y_test)

# create the model
model = LinearRegression().fit(X_train, y_train)

train_predicted = model.predict(X_train)
test_predicted = model.predict(X_test)

train_mse = mean_squared_error(y_train, train_predicted)
test_mse = mean_squared_error(y_test, test_predicted)

print('MSE train data, MSE test data', train_mse, test_mse)
print('RMSE train data, RMSE test data', np.sqrt(np.absolute(train_mse)),  np.sqrt(np.a
print('R2 train data, R2 test data\n\n', r2_score(y_train, train_predicted), r2_score(y

print('Regression Coefficients on Normalized Data, 20% as test data');
model.coef_
```

Out[22]:
```
MSE train data, MSE test data 0.04684095538290938 0.04634589356920685
RMSE train data, RMSE test data 0.21642771398993563 0.21528096425185123
R2 train data, R2 test data

 0.003712433738775056 0.0049098683039340285
Regression Coefficients on Normalized Data, 20% as test data
array([-0.48065077,  0.05216703,  0.2411039 , -0.14960785, -0.0273536 ,
       -0.34402467,  0.37306094,  0.17164222,  0.19379202, -0.08567921])
```

In [23]:
```
l = ( round(x,2) for x in list(model.coef_))
list(l)
```

Out[23]:
```
[-0.48, 0.05, 0.24, -0.15, -0.03, -0.34, 0.37, 0.17, 0.19, -0.09]
```

Linear Regression : Actual Values : Cross Validations

In [24]:
```
print('Data Not Normalized, Absoultr ACR as the target')
y = st['URDACT_Albumin_creatinine_ratio_mg_g']
X = st.drop(columns=['URDACT_Albumin_creatinine_ratio_mg_g'])
linear_regression_cross_validation_scores = cross_val_score(LinearRegression(), X, y, c

print("Accuracy, Standard Deviations (+/- 2) :", linear_regression_cross_validation_sco

print('Regression Coefficients on Actual Data, 10 fold cross validation');
```

```
print('All Scores', linear_regression_cross_validation_scores)
print('\n\n')
linear_regression_cross_validation_scores
```

```
Data Not Normalized, Absoultr ACR as the target
Accuracy, Standard Deviations (+/- 2) : -0.15570477951774175 0.2664724030509976
Regression Coefficients on Actual Data, 10 fold cross validation
All Scores [-0.36030818 -0.22316023 -0.04725131 -0.01631    -0.00664793 -0.06289664
 -0.01075262 -0.22474334 -0.31281287 -0.29216468]
```

Out[24]:
```
array([-0.36030818, -0.22316023, -0.04725131, -0.01631   , -0.00664793,
       -0.06289664, -0.01075262, -0.22474334, -0.31281287, -0.29216468])
```

In [25]:
```
l = ( round(x,2) for x in list(linear_regression_cross_validation_scores))
list(l)
```

Out[25]:
```
[-0.36, -0.22, -0.05, -0.02, -0.01, -0.06, -0.01, -0.22, -0.31, -0.29]
```

Linear Regression : Normalized Values : Cross Validations

In [26]:
```
print('Data Normalized, ACR Value as the target')
y = st_norm['URDACT_Albumin_creatinine_ratio_mg_g']
X = st_norm.drop(columns=['URDACT_Albumin_creatinine_ratio_mg_g'])
linear_regression_cross_validation_scores = cross_val_score(LinearRegression(), X, y, c

print("Accuracy, Standard Deviations (+/- 2) :", linear_regression_cross_validation_sco
print('All Scores', linear_regression_cross_validation_scores)
print('\n\n')
linear_regression_cross_validation_scores
```

```
Data Normalized, ACR Value as the target
Accuracy, Standard Deviations (+/- 2) : -0.1557047795177419 0.2664724030509981
All Scores [-0.36030818 -0.22316023 -0.04725131 -0.01631    -0.00664793 -0.06289664
 -0.01075262 -0.22474334 -0.31281287 -0.29216468]
```

Out[26]:
```
array([-0.36030818, -0.22316023, -0.04725131, -0.01631   , -0.00664793,
       -0.06289664, -0.01075262, -0.22474334, -0.31281287, -0.29216468])
```

In [27]:
```
l = ( round(x,2) for x in list(linear_regression_cross_validation_scores))
list(l)
```

Out[27]:
```
[-0.36, -0.22, -0.05, -0.02, -0.01, -0.06, -0.01, -0.22, -0.31, -0.29]
```

Apply Polynomial Regression: Actual Values

In [28]:
```
from sklearn.preprocessing import PolynomialFeatures

print('Data Not Normalized, Absolute ACR as the target')
```

```
y = st['URDACT_Albumin_creatinine_ratio_mg_g']
X = st.drop(columns=['URDACT_Albumin_creatinine_ratio_mg_g'])

X_poly =  PolynomialFeatures(degree = 2).fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(
    X_poly, y, test_size=0.20, random_state=42)


poly_regression = LinearRegression().fit(X_train, y_train)

poly_regression_train_pred = poly_regression.predict(X_train)
poly_regression_test_pred = poly_regression.predict(X_test)

poly_regression_train_mse = mean_squared_error(y_train, poly_regression_train_pred)
poly_regression_test_mse = mean_squared_error(y_test, poly_regression_test_pred)


print('MSE train data, MSE test data', poly_regression_train_mse, poly_regression_test_
print('RMSE train data, RMSE test data', np.sqrt(np.absolute(poly_regression_train_mse)
print('R2 train data:, R2 test data', r2_score(y_train, poly_regression_train_pred), r2

print('Polynomial Regression (degree 2) Coefficients on Normalized Data, 20% as test da
poly_regression.coef_
```

Out[28]:

```
Data Not Normalized, Absolute ACR as the target
MSE train data, MSE test data 36.63273086280682 36.343214847903404
RMSE train data, RMSE test data 6.052497902751129 6.052497902751129
R2 train data:, R2 test data 0.005459520957930342 0.00397852108823038
Polynomial Regression (degree 2) Coefficients on Normalized Data, 20% as test data
array([-5.06429081e-15, -1.11421346e-02,  4.24647766e-02,  4.46222251e-02,
       -1.26006345e-02,  1.91109296e-02, -1.10565900e-01,  2.74337654e-01,
        2.04248502e-01,  1.03484141e-01, -1.19457649e-03,  9.25474502e-06,
       -4.28317892e-04, -1.27404809e-05, -4.02342613e-05,  2.10603193e-03,
        6.86305627e-04, -1.31823222e-03, -2.80452600e-03,  6.10933319e-04,
        1.03989491e-04,  1.36076643e-03,  1.15550851e-03,  1.37027699e-04,
       -5.68851960e-03,  3.49349816e-03,  3.47413863e-03,  6.66978748e-03,
       -5.14765899e-03, -3.80606044e-04, -2.55119181e-04,  6.08892144e-04,
       -6.06358295e-03, -2.04945328e-03,  4.26676157e-03,  1.06418231e-02,
       -4.27855624e-03, -4.19203881e-04, -3.03095764e-04, -2.23067612e-03,
       -1.87777340e-03,  1.37251591e-03,  1.49380180e-03,  3.93427720e-03,
        9.99775995e-05, -9.26958389e-03, -3.53280288e-02,  1.75253072e-02,
        3.87847775e-03,  2.54919019e-02, -3.29552026e-05, -7.73890555e-03,
        1.77028179e-02,  2.20002769e-02,  1.76328189e-02, -1.53296598e-03,
       -5.56904621e-03, -3.53892737e-03, -2.96742823e-02,  3.38743833e-04,
        6.27633347e-03, -1.97789027e-02,  1.01994035e-03, -2.06056442e-02,
        4.29543331e-04, -2.59958687e-06])
```

In [29]:

```
l = ( round(x,2) for x in list(poly_regression.coef_))
list(l)
```

Out[29]:

```
[-0.0,
 -0.01,
 0.04,
 0.04,
 -0.01,
 0.02,
 -0.11,
 0.27,
```

0.2,
0.1,
-0.0,
0.0,
-0.0,
-0.0,
-0.0,
0.0,
0.0,
-0.0,
-0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
-0.01,
0.0,
0.0,
0.01,
-0.01,
-0.0,
-0.0,
0.0,
-0.01,
-0.0,
0.0,
0.01,
-0.0,
-0.0,
-0.0,
-0.0,
-0.0,
0.0,
0.0,
0.0,
0.0,
-0.01,
-0.04,
0.02,
0.0,
0.03,
-0.0,
-0.01,
0.02,
0.02,
0.02,
-0.0,
-0.01,
-0.0,
-0.03,
0.0,
0.01,
-0.02,
0.0,
-0.02,
0.0,
-0.0]

Apply Polynomial Regression: Actual Values: Cross Validation Scores

In [30]:
```
print('Polynomial Regression (degree 2) Coefficients on Normalized Data, 20% as test da
print('Polynomial Regression cross_val_score');
poly_regression_cv = cross_val_score(LinearRegression(), X_poly, y, cv = 10)
print("Accuracy and Standard Deviations (+/- 2) ", poly_regression_cv.mean(), poly_regr
print ('All Scores', poly_regression_cv)
```

```
Polynomial Regression (degree 2) Coefficients on Normalized Data, 20% as test data
Polynomial Regression cross_val_score
Accuracy and Standard Deviations (+/- 2)  -0.19218257570462913 0.39704848652879715
All Scores [-0.35830244 -0.22039309 -0.04733391 -0.01625872 -0.01082209 -0.07023279
 -0.01294305 -0.22846199 -0.6637287  -0.29334898]
```

In [31]:
```
l = ( round(x,2) for x in list(poly_regression_cv))
list(l)
```

Out[31]:
```
[-0.36, -0.22, -0.05, -0.02, -0.01, -0.07, -0.01, -0.23, -0.66, -0.29]
```

Apply Polynomial Regression: Normalized Values

In [32]:
```
from sklearn.preprocessing import PolynomialFeatures

print('Data Not Normalized, Absolute ACR as the target')
y = st_norm['URDACT_Albumin_creatinine_ratio_mg_g']
X = st_norm.drop(columns=['URDACT_Albumin_creatinine_ratio_mg_g'])

X_poly =  PolynomialFeatures(degree = 2).fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(
    X_poly, y, test_size=0.20, random_state=42)


poly_regression = LinearRegression().fit(X_train, y_train)

poly_regression_train_pred = poly_regression.predict(X_train)
poly_regression_test_pred = poly_regression.predict(X_test)

poly_regression_train_mse = mean_squared_error(y_train, poly_regression_train_pred)
poly_regression_test_mse = mean_squared_error(y_test, poly_regression_test_pred)


print('MSE train data, MSE test data', poly_regression_train_mse, poly_regression_test_
print('RMSE train data, RMSE test data', np.sqrt(np.absolute(poly_regression_train_mse)
print('R2 train data:, R2 test data', r2_score(y_train, poly_regression_train_pred), r2

print('Polynomial Regression (degree 2) Coefficients on Normalized Data, 20% as test da
poly_regression.coef_
```

```
Data Not Normalized, Absolute ACR as the target
MSE train data, MSE test data 0.04675881520847199 0.04638927066396353
RMSE train data, RMSE test data 0.21623786719368093 0.21623786719368093
R2 train data:, R2 test data 0.005459520957930342 0.003978521088250919
Polynomial Regression (degree 2) Coefficients on Normalized Data, 20% as test data
```

Out[32]:

```
array([-1.37695868e-14, -1.21434512e+00,  2.77372316e-01,  7.24500919e-01,
       -1.96419916e-01,  2.94601957e-02, -6.50465070e-01,  7.48743977e-01,
        4.14610957e-01,  3.54747195e-01, -3.54543309e-02,  3.69074900e+00,
       -1.10063540e+01, -7.50979485e-01, -2.19590640e+00,  1.12368880e+01,
        1.48570520e+01, -1.32854944e+01, -2.19130291e+01,  6.88186178e+00,
        1.76383155e+01,  2.25313937e+00,  4.38876786e+00,  4.81896051e-01,
       -1.95572401e+00,  4.87307183e+00,  2.25610938e+00,  3.35801127e+00,
       -3.73636343e+00, -4.15978174e+00, -2.22268318e+00,  4.91190958e+00,
       -4.78192349e+00, -6.55760581e+00,  6.35588999e+00,  1.22899800e+01,
       -7.12363369e+00, -1.05095749e+01, -2.26394254e+00, -1.62886600e+00,
       -5.56321708e+00,  1.89308854e+00,  1.59736315e+00,  6.06519587e+00,
        2.32080206e+00, -6.61717145e-01, -1.02321180e+01,  2.36310209e+00,
        4.05448124e-01,  3.84189437e+00, -7.47864103e-02, -9.09409867e+00,
        9.68485743e+00,  9.33117045e+00,  1.07819897e+01, -1.41144904e+01,
       -1.41841189e+00, -6.98795885e-01, -8.44749810e+00,  1.45202735e+00,
        9.60819580e-01, -4.36523089e+00,  3.38949750e+00, -6.55632406e+00,
        2.05795900e+00, -1.87538135e-01])
```

In [33]:

```
l = ( round(x,2) for x in list(poly_regression.coef_))
list(l)
```

Out[33]:

```
[-0.0,
 -1.21,
 0.28,
 0.72,
 -0.2,
 0.03,
 -0.65,
 0.75,
 0.41,
 0.35,
 -0.04,
 3.69,
 -11.01,
 -0.75,
 -2.2,
 11.24,
 14.86,
 -13.29,
 -21.91,
 6.88,
 17.64,
 2.25,
 4.39,
 0.48,
 -1.96,
 4.87,
 2.26,
 3.36,
 -3.74,
 -4.16,
 -2.22,
 4.91,
 -4.78,
```

```
-6.56,
6.36,
12.29,
-7.12,
-10.51,
-2.26,
-1.63,
-5.56,
1.89,
1.6,
6.07,
2.32,
-0.66,
-10.23,
2.36,
0.41,
3.84,
-0.07,
-9.09,
9.68,
9.33,
10.78,
-14.11,
-1.42,
-0.7,
-8.45,
1.45,
0.96,
-4.37,
3.39,
-6.56,
2.06,
-0.19]
```

Apply Polynomial Regression: Normalized Values: Cross Validation

In [34]:

```
print('Polynomial Regression (degree 2) Coefficients on Normalized Data, 20% as test da
print('Polynomial Regression cross_val_score');
poly_regression_cv = cross_val_score(LinearRegression(), X_poly, y, cv = 10)
print("Accuracy and Standard Deviations (+/- 2) ", poly_regression_cv.mean(), poly_regr
print ('All Scores', poly_regression_cv)
```

```
Polynomial Regression (degree 2) Coefficients on Normalized Data, 20% as test data
Polynomial Regression cross_val_score
Accuracy and Standard Deviations (+/- 2)  -0.19218257570461 0.3970484865292476
All Scores [-0.35830244 -0.22039309 -0.04733391 -0.01625872 -0.01082209 -0.07023279
 -0.01294305 -0.22846199 -0.6637287  -0.29334898]
```

In [35]:

```
l = ( round(x,2) for x in list(poly_regression_cv))
list(l)
```

Out[35]:

```
[-0.36, -0.22, -0.05, -0.02, -0.01, -0.07, -0.01, -0.23, -0.66, -0.29]
```

Bayesian: Actual Values: Linear

In [36]:
```
from sklearn.linear_model import BayesianRidge

print('Data Not Normalized, Absolute ACR as the target')
y = st['URDACT_Albumin_creatinine_ratio_mg_g']
X = st.drop(columns=['URDACT_Albumin_creatinine_ratio_mg_g'])
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=

bayesian = BayesianRidge().fit(X_train, y_train)

bayesian_train_pred = bayesian.predict(X_train)
bayesian_test_pred = bayesian.predict(X_test)

bayesian_train_mse = mean_squared_error(y_train, bayesian_train_pred)
bayesian_test_mse = mean_squared_error(y_test, bayesian_test_pred)

print('MSE train data, MSE test data', bayesian_train_mse, bayesian_test_mse)
print('RMSE train data, RMSE test data', np.sqrt(np.absolute(bayesian_train_mse)),  np.

print('R2 train data, R2 test data', r2_score(y_train, bayesian_train_pred), r2_score(y

print('##########################################')

bayesian.score
```

```
Data Not Normalized, Absolute ACR as the target
MSE train data, MSE test data 36.70444861463874 36.32457362111485
RMSE train data, RMSE test data 6.058419646627224 6.058419646627224
R2 train data, R2 test data 0.0035124587110596517 0.0044894022073710405
##########################################
```

Out[36]:
```
<bound method RegressorMixin.score of BayesianRidge()>
```

Bayesian: Actual Values: Polynomial Degree 2

In [37]:
```
X_poly =  PolynomialFeatures(degree = 2).fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X_poly, y, test_size=0.20, random_s

bayesian = BayesianRidge().fit(X_train, y_train)

bayesian_train_pred = bayesian.predict(X_train)
bayesian_test_pred = bayesian.predict(X_test)

bayesian_train_mse = mean_squared_error(y_train, bayesian_train_pred)
bayesian_test_mse = mean_squared_error(y_test, bayesian_test_pred)

print('Bayesian on Polynomial fit')
print('MSE train data, MSE test data', bayesian_train_mse, bayesian_test_mse)
print('RMSE train data, RMSE test data', np.sqrt(np.absolute(bayesian_train_mse)),  np.

print('R2 train data, R2 test data', r2_score(y_train, bayesian_train_pred), r2_score(y
```

```
Bayesian on Polynomial fit
MSE train data, MSE test data 36.6966848155483 36.335548367168386
```

```
RMSE train data, RMSE test data 6.057778868161853 6.057778868161853
R2 train data, R2 test data 0.0037232377680596063 0.0041886285185086525
```

Bayesian with Cross Validation Polynomial X is used

In [38]:

```
bayesian_cv = cross_val_score(BayesianRidge(), X_poly, y, cv = 10)
print("Accuracy and Standard Deviations", bayesian_cv.mean(), bayesian_cv.std() * 2)
print('All Scores', bayesian_cv)
```

```
Accuracy and Standard Deviations -0.15733642407559528 0.26960963772389046
All Scores [-0.36208939 -0.22413904 -0.04759558 -0.0157379  -0.00784081 -0.06373985
 -0.01026397 -0.22504944 -0.31954659 -0.29736168]
```

In [39]:

```
l = ( round(x,2) for x in list(bayesian_cv))
list(l)
```

Out[39]:

```
[-0.36, -0.22, -0.05, -0.02, -0.01, -0.06, -0.01, -0.23, -0.32, -0.3]
```

Bayesian with Normalized Values

In [40]:

```
from sklearn.linear_model import BayesianRidge

print('Data Not Normalized, Absolute ACR as the target')
y = st_norm['URDACT_Albumin_creatinine_ratio_mg_g']
X = st_norm.drop(columns=['URDACT_Albumin_creatinine_ratio_mg_g'])
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=

bayesian = BayesianRidge().fit(X_train, y_train)

bayesian_train_pred = bayesian.predict(X_train)
bayesian_test_pred = bayesian.predict(X_test)

bayesian_train_mse = mean_squared_error(y_train, bayesian_train_pred)
bayesian_test_mse = mean_squared_error(y_test, bayesian_test_pred)

print('MSE train data, MSE test data', bayesian_train_mse, bayesian_test_mse)
print('RMSE train data, RMSE test data', np.sqrt(np.absolute(bayesian_train_mse)),  np.

print('R2 train data, R2 test data', r2_score(y_train, bayesian_train_pred), r2_score(y

print('########################################')
```

```
Data Not Normalized, Absolute ACR as the target
MSE train data, MSE test data 0.04684926675841848 0.046364236334120734
RMSE train data, RMSE test data 0.21644691441186792 0.21644691441186792
R2 train data, R2 test data 0.003535654251453413 0.004516031807361154
########################################
```

Bayesian, Normalized Data, Polynomial Degree = 2

In [41]:
```
X_poly =  PolynomialFeatures(degree = 2).fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X_poly, y, test_size=0.20, random_s

bayesian = BayesianRidge().fit(X_train, y_train)

bayesian_train_pred = bayesian.predict(X_train)
bayesian_test_pred = bayesian.predict(X_test)

bayesian_train_mse = mean_squared_error(y_train, bayesian_train_pred)
bayesian_test_mse = mean_squared_error(y_test, bayesian_test_pred)

print('Bayesian on Polynomial fit')
print('MSE train data, MSE test data', bayesian_train_mse, bayesian_test_mse)
print('RMSE train data, RMSE test data', np.sqrt(np.absolute(bayesian_train_mse)),  np.

print('R2 train data, R2 test data', r2_score(y_train, bayesian_train_pred), r2_score(y
```

```
Bayesian on Polynomial fit
MSE train data, MSE test data 0.046832574490465954 0.04635559674395354
RMSE train data, RMSE test data 0.21640835124935903 0.21640835124935903
R2 train data, R2 test data 0.0038906918222684217 0.004701531972644224
```

Bayesian Normalized Data: Cross validation: Polynomial Data used

In [42]:
```
bayesian_cv = cross_val_score(BayesianRidge(), X_poly, y, cv = 10)
print("Accuracy and Standard Deviations", bayesian_cv.mean(), bayesian_cv.std() * 2)
print('All Scores', bayesian_cv)
```

```
Accuracy and Standard Deviations -0.1556649650385757 0.2668065583180628
All Scores [-0.36062968 -0.22324443 -0.0480207  -0.01600587 -0.00770117 -0.06160748
 -0.00948674 -0.22436637 -0.31209826 -0.29348895]
```

In [43]:
```
l = ( round(x,2) for x in list(bayesian_cv))
list(l)
```

Out[43]:
```
[-0.36, -0.22, -0.05, -0.02, -0.01, -0.06, -0.01, -0.22, -0.31, -0.29]
```

RandomForestRegressor: Actual Values

In [44]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegres
from sklearn.ensemble import RandomForestRegressor

print('Data Not Normalized, Absolute ACR as the target')
y = st['URDACT_Albumin_creatinine_ratio_mg_g']
X = st.drop(columns=['URDACT_Albumin_creatinine_ratio_mg_g'])
X_poly =  PolynomialFeatures(degree = 2).fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(
    X_poly, y, test_size=0.20, random_state=42)

random_forest = RandomForestRegressor(n_estimators = 100, bootstrap=True, criterion='ms
```

```
random_forest_train_pred = random_forest.predict(X_train)
random_forest_test_pred = random_forest.predict(X_test)

random_forest_train_mse = mean_squared_error(y_train, random_forest_train_pred)
random_forest_test_mse = mean_squared_error(y_test, random_forest_test_pred)

print('MSE train data, MSE test data', random_forest_train_mse, random_forest_test_mse)
print('RMSE train data, RMSE test data', np.sqrt(np.absolute(random_forest_train_mse)),
print('R2 train data, R2 test data', r2_score(y_train, random_forest_train_pred), r2_sc
```

```
Data Not Normalized, Absolute ACR as the target
MSE train data, MSE test data 36.53909337061131 36.24884348059105
RMSE train data, RMSE test data 6.044757511316009 6.044757511316009
R2 train data, R2 test data 0.008001681319743992 0.006564861048268478
```

RandomForestRegressor: Actual Values: Cross validation

K Fold

In [45]:
```
random_forest_cv = cross_val_score(RandomForestRegressor(n_estimators = 100, bootstrap=
print("Accuracy: Mean and Standard Deviations", random_forest_cv.mean(), random_forest_
print('All scores', random_forest_cv)
```

```
Accuracy: Mean and Standard Deviations -0.15124813853610783 0.25791079717449683
All scores [-0.35485597 -0.21668029 -0.04707368 -0.01440631 -0.00844273 -0.06479159
 -0.00944252 -0.21654281 -0.29925436 -0.28099113]
```

In [46]:
```
l = ( round(x,2) for x in list(random_forest_cv))
list(l)
```

Out[46]:
```
[-0.35, -0.22, -0.05, -0.01, -0.01, -0.06, -0.01, -0.22, -0.3, -0.28]
```

In [47]:
```
random_forest_cv = cross_val_score(RandomForestRegressor(n_estimators = 100, bootstrap=
print("Accuracy: Mean and Standard Deviations", random_forest_cv.mean(), random_forest_
print('All scores', random_forest_cv)
```

```
Accuracy: Mean and Standard Deviations -0.15165397414885073 0.25836061612023087
All scores [-0.35501204 -0.21773379 -0.0473228  -0.01543925 -0.00745611 -0.0646027
 -0.00984395 -0.21773859 -0.30074174 -0.28064878]
```

In [48]:
```
l = ( round(x,2) for x in list(random_forest_cv))
list(l)
```

Out[48]:
```
[-0.36, -0.22, -0.05, -0.02, -0.01, -0.06, -0.01, -0.22, -0.3, -0.28]
```

In [49]:
```
#random_forest.score
```

RandomForestRegressor: Normalized Values

In [50]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegres


print('Data Not Normalized, Absolute ACR as the target')
y = st_norm['URDACT_Albumin_creatinine_ratio_mg_g']
X = st_norm.drop(columns=['URDACT_Albumin_creatinine_ratio_mg_g'])
X_poly =  PolynomialFeatures(degree = 2).fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(
    X_poly, y, test_size=0.20, random_state=42)

random_forest = RandomForestRegressor(n_estimators = 10, bootstrap=True, criterion='mse

random_forest_train_pred = random_forest.predict(X_train)
random_forest_test_pred = random_forest.predict(X_test)

random_forest_train_mse = mean_squared_error(y_train, random_forest_train_pred)
random_forest_test_mse = mean_squared_error(y_test, random_forest_test_pred)

print('MSE train data, MSE test data', random_forest_train_mse, random_forest_test_mse)
print('RMSE train data, RMSE test data', np.sqrt(np.absolute(random_forest_train_mse)),
print('R2 train data, R2 test data', r2_score(y_train, random_forest_train_pred), r2_sc
```

```
Data Not Normalized, Absolute ACR as the target
MSE train data, MSE test data 0.04656796561601843 0.046209642423612265
RMSE train data, RMSE test data 0.21579612048417002 0.21579612048417002
R2 train data, R2 test data 0.009518812115276831 0.007835309156875803
```

In [51]:
```
# RandomForestRegressor: Normalized Values: Cross validation K Fold
```

In [52]:
```
random_forest_cv = cross_val_score(RandomForestRegressor(n_estimators = 100, bootstrap=
print("Accuracy: Mean and Standard Deviations", random_forest_cv.mean(), random_forest_
print('All scores', random_forest_cv)
```

```
Accuracy: Mean and Standard Deviations -0.1490302061132035 0.255256630889444
All scores [-0.34793126 -0.21289198 -0.04704062 -0.01512759 -0.00684827 -0.0615237
 -0.00754446 -0.21620933 -0.29516905 -0.2800158 ]
```

In [53]:
```
l = ( round(x,2) for x in list(random_forest_cv))
list(l)
```

Out[53]:
```
[-0.35, -0.21, -0.05, -0.02, -0.01, -0.06, -0.01, -0.22, -0.3, -0.28]
```

rfr_cv

In [ ]:

In [ ]:

In [54]:
```
# interpret coeffs
```

https://www.analyticsvidhya.com/blog/2021/07/a you-need-to-know-about-polynomial-regression/

applying polynomial regression degree 2

poly = PolynomialFeatures(degree=2, include_bias=True) x_train_trans = poly.fit_transform(X_train) x_test_trans = poly.transform(X_test)

include bias parameter

lr = LinearRegression() lr.fit(x_train_trans, y_train) y_pred = lr.predict(x_test_trans) print(r2_score(y_test, y_pred))

lr.coef*[:10] print(lr.intercept*)

From: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html Ref: Example Parameters: RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=2, 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=100, n_jobs=None, oob_score=False, random_state=0, verbose=0, warm_start=False)

References

Projects mentioned on: http://sitestree.com/prediction-bayesian-regression-concepts-example-projects/

Insurance HealthCare Costs: https://github.com/techshot25/HealthCare Linear and Bayesian modeling in R: Predicting movie popularity https://towardsdatascience.com/linear-and-bayesian-modelling-in-r-predicting-movie-popularity-6c8ef0a44184

Bayesian-Stock-Price-Prediction https://github.com/lschlessinger1/Bayesian-Stock-Price-Prediction

Bayesian Prediction: Well (Oil) Production https://github.com/jpgrana/bayesian-approach-predicting-well-production

Binary Classification on Stock Market (S&P 500) using Naive Bayes and Logistic Regression https://github.com/NeilPrabhu/Stock-Prediction

Naive Bayes Weather Prediction https://github.com/husnainfareed/simple-naive-bayes-weather-prediction/blob/master/bayes.py

Regression Predict Fuel Efficiency: https://www.tensorflow.org/tutorials/keras/basic_regression

Regression-Example-Predicting-House-Prices https://github.com/andersy005/deep-learning/blob/master/keras/04-A-Regression-Example-Predicting-House-Prices.ipynb

Stock Price Prediction using Regression https://github.com/chaitjo/regression-stock-prediction

Concept: Predicting the Future with Bayes' Theorem https://fs.blog/2018/09/bayes-theorem/

Chapter 5 - Bayesian Prediction
https://www.sciencedirect.com/science/article/pii/B9780123748546000089

Books:

Bayesian Methods for Hackers https://github.com/CamDavidsonPilon/Probabilistic-Programming-and-Bayesian-Methods-for-Hackers

Multiple-linear-regression https://github.com/topics/multiple-linear-regression

Making Predictions with Regression Analysis https://statisticsbyjim.com/regression/predictions-regression/

Regression and Prediction
http://jukebox.esc13.net/untdeveloper/RM/Stats_Module_5/Stats_Module_56.html

https://towardsdatascience.com/train-test-split-and-cross-validation-in-python-80b61beca4b6

In [ ]: