import libraries

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

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 = 0
```

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	53	3.0	2017-2018	95405.0
1	53	3.0	2017-2018	95405.0
2	53	3.0	2017-2018	95405.0
3	53	3.0	2017-2018	95405.0
4	53	3.0	2017-2018	95405.0

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

Out[5]:
```
'./nhanes_output_data/classifiedGroups/2022-12-01/0_dietaryIntakeDataForClassificationAn
dAnalysisData.csv'
```

In [6]:
```
df.shape
```
Out[6]:
```
(193805, 87)
```
In [7]:
```
'''
data_folder = './nhanes_input_data/'
df = pd.read_csv( data_folder + '0_dietaryIntakeDataForClassificationAndAnalysisData.cs
df.shape
df.head(5)
'''
```
Out[7]:
```
"\ndata_folder = './nhanes_input_data/'\ndf = pd.read_csv( data_folder + '0_dietaryIntak
eDataForClassificationAndAnalysisData.csv') # import the CSV as a pandas dataframe\ndf.s
hape\ndf.head(5) \n"
```
In [8]:
```
original_acr = df['URDACT_Albumin_creatinine_ratio_mg_g']
```
In [9]:
```
# Create categories for the severity of ACR
```
In [10]:
```
df.columns[:10]
```
Out[10]:
```
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 [11]:
```
# convert str values to int using the scikit-learn encoder : acr_category
```
In [12]:
```
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 [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
```

Apply Machine Learning

Linear Regression

Bayesian

RandomForest Regression

st.columns

In [18]:
```
# Split data into Train and Test
# Use 10% of dataset as testing data
```

In [19]:
```
from sklearn.model_selection import train_test_split
```

Using ACR category as the target

finding MSE, RMSE, Accuracy error

actual Data

Have to verify and/or adjust: Was this approach correct? i.e using category numbers as the target and using classification ID as target for linear regression. The code worked as the category was a number though output probably is not accurate, and adjustments are required on the output side to get correct classification on the regression output. Or a different approach can be used for classification

Test accuracy was better when using Absolute ACR values as the target

Use ACR as the Target variable

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

In [20]:
```
# work on the normalized data
# st = st_norm
```

In [21]:
```
import numpy as np

# import train_test_split
from sklearn.model_selection import train_test_split
```

In [22]:
```
# st.fillna(0)
```

In [23]:
```
# st['URDACT_Albumin_creatinine_ratio_mg_g'][30:40], df['URDACT_Albumin_creatinine_rati
```

In [28]:
```
#st = st_norm
#st = st.reset_index()
#st.dropna();

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 = X.reset_index
```

```
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)

#X_train = X_train.reset_index
#y_train = y_train.reset_index
#X_test = X_test.reset_index
#y_test = y_test.reset_index
```

Out[28]:

```
(142529, 35633, 142529, 35633)
```

In [29]:

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

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

In [30]:

```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score


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
```

```
MSE train data, MSE test data 69631.95564473532 84432.98458862741
RMSE train data, RMSE test data 263.87867599473685 290.573544199446
R2 train data, R2 test data 0.00011137873890987304 2.0230781149543908e-05
```

In [31]:

```
print('Regression Coefficients on Actual Data (not normalized), 20% as test data');
model.coef_
```

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

Out[31]:

```
array([-0.0339911 , -0.02630866,  0.04799198,  0.00862493,  0.60546403,
        0.88157393, -0.43578233, -0.98301004, -0.66529095,  0.02553043])
```

Checking on Normalized data

In [ ]:

```
#st_norm[['URDACT_Albumin_creatinine_ratio_mg_g', 'acr_preserved', 'acr_category', 'acr_
```

In [ ]:

```
y = st_norm ['URDACT_Albumin_creatinine_ratio_mg_g'] #st_norm['acr_preserved']
X = st_norm.drop(columns=['URDACT_Albumin_creatinine_ratio_mg_g']) #, 'acr_preserved'])

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', r2_score(y_train, train_predicted), r2_score(y_tes
```

In [32]:

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

Out[32]:

```
Regression Coefficients on Normalized Data, 20% as test data
array([-0.0339911 , -0.02630866,  0.04799198,  0.00862493,  0.60546403,
        0.88157393, -0.43578233, -0.98301004, -0.66529095,  0.02553043])
```

y = st_norm['acr_preserved'] X = st_norm.drop(columns=['URDACT_Albumin_creatinine_ratio_mg_g', 'acr_preserved'])

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.10, random_state=42) 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.absolute(test_mse))) print('R2 train data, R2 test data', r2_score(y_train, train_predicted), r2_score(y_test, test_predicted))

In [33]:

```
# use cross validations using Linear Regression
```

In [34]:

```
# not normalized data
```

In [37]:

```
from sklearn.model_selection import cross_val_score

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')
```

```
Data Not Normalized, Absoultr ACR as the target
Accuracy, Standard Deviations (+/- 2) : -10384.199747228411 20112.950551759288
Regression Coefficients on Actual Data, 10 fold cross validation
All Scores [-2.25671753e+04 -2.50634576e+04 -2.24218062e+04 -1.83625663e+04
 -9.77800552e+03 -3.72712629e+03 -1.64036623e+03 -2.75230275e+02
 -6.09319102e+00 -1.70599114e-01]
```

print('Data Not Normalized, ACR Category as the target') y = st['acr_category'] X = st.drop(columns=['URDACT_Albumin_creatinine_ratio_mg_g']) linear_regression_cross_validation_scores = cross_val_score(LinearRegression(), X, y, cv = 10)

print("Accuracy, Standard Deviations (+/- 2) :", linear_regression_cross_validation_scores.mean(), linear_regression_cross_validation_scores.std() * 2) print('All Scores', linear_regression_cross_validation_scores) print('\n\n')

print('Data Normalized, ACR Value as the target') y = st_norm['URDACT_Albumin_creatinine_ratio_mg_g'] X = st_norm.drop(columns=['acr_category', 'acr_category_2', 'URDACT_Albumin_creatinine_ratio_mg_g', 'acr_preserved']) linear_regression_cross_validation_scores = cross_val_score(LinearRegression(), X, y, cv = 10)

print("Accuracy, Standard Deviations (+/- 2) :", linear_regression_cross_validation_scores.mean(), linear_regression_cross_validation_scores.std() * 2) print('All Scores', linear_regression_cross_validation_scores) print('\n\n')

print('Data Normalized, ACR Category as the target')

y = st_norm['acr_category_2'] X = st_norm.drop(columns=['acr_category', 'acr_category_2', 'URDACT_Albumin_creatinine_ratio_mg_g', 'acr_preserved']) linear_regression_cross_validation_scores = cross_val_score(LinearRegression(), X, y, cv = 10)

print("Accuracy, Standard Deviations (+/- 2) :", linear_regression_cross_validation_scores.mean(), linear_regression_cross_validation_scores.std() * 2) print('All Scores', linear_regression_cross_validation_scores) print('\n\n')

Apply Polynomial Regression

In [38]:

```python
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)
```

Data Not Normalized, Absolute ACR as the target

In [45]:
```
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
```

```
MSE train data, MSE test data 69611.79502924645 84447.76271195522
RMSE train data, RMSE test data 263.8404726899314 263.8404726899314
R2 train data:, R2 test data 0.00040087757372353483 -0.00015479352276237535
```

In [46]:
```
print('Polynomial Regression (degree 2) Coefficients on Normalized Data, 20% as test da
poly_regression.coef_
```

```
Polynomial Regression (degree 2) Coefficients on Normalized Data, 20% as test data
```
Out[46]:
```
array([ 6.93028367e-13, -6.61260380e-02, -3.16784855e-02,  3.61869794e-02,
       -1.72390926e-02,  1.45780824e+00,  2.35730610e+00, -1.28709328e+00,
       -2.22709562e+00, -1.88396799e+00,  2.37173279e-02,  5.98305606e-05,
       -2.61312334e-03, -5.22740936e-04, -3.28963440e-04,  1.79871549e-02,
        2.16657360e-02, -4.29046189e-02, -1.08681674e-02, -1.75398546e-02,
        2.37944053e-03,  1.27720130e-02,  2.10212903e-02, -4.87193204e-03,
       -1.75626257e-01, -1.91245604e-01,  2.88194142e-01,  2.04445693e-01,
        1.91751606e-01, -1.03777867e-02,  2.15293835e-03, -4.45769806e-04,
       -7.93993464e-02, -1.05065272e-01,  1.95007612e-01,  5.67896243e-02,
        8.44188267e-02, -1.12763047e-02,  2.27362324e-03, -8.79980429e-03,
       -4.05240254e-02,  4.66421085e-02,  5.97213156e-02,  5.46281499e-02,
        1.55639263e-03,  4.05278369e-02,  6.42359229e-01, -8.69729650e-01,
       -8.53741041e-01, -7.22292873e-01,  1.49452871e-02,  2.39822793e-01,
       -6.27102504e-01, -8.98521604e-01, -8.11843167e-01,  4.17758976e-03,
        5.79615937e-01,  1.10452070e+00,  1.09987864e+00, -2.44751164e-02,
        5.67381355e-01,  1.12649219e+00, -3.22306518e-02,  5.15201869e-01,
       -2.48707034e-02,  2.23856400e-05])
```

In [ ]:
```

```

In [40]:
```
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)
```

```
Accuracy and Standard Deviations (+/- 2)  -10544.733577359078 20461.55955421137
All Scores [-2.29249198e+04 -2.56259388e+04 -2.27524228e+04 -1.85444716e+04
 -9.88855344e+03 -3.76514823e+03 -1.65942150e+03 -2.79641758e+02
 -6.64725609e+00 -1.70595289e-01]
```

In [64]:
```
print('Polynomial Regression (degree 2) Coefficients on Normalized Data, 20% as test da
# poly_regression_cv.coef_
```

```
Polynomial Regression (degree 2) Coefficients on Normalized Data, 20% as test data
```

In [47]:
```
# https://www.analyticsvidhya.com/blog/2021/07/all-you-need-to-know-about-polynomial-re
#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))
```
```
-4.26766717063695
```

In [52]:
```
lr.coef_[:10]
```

Out[52]:
```
array([-3.81416865e-03, -6.69377001e+01, -3.84786753e+00,  5.80169491e+00,
        6.20199846e-01,  1.91469449e+00, -6.95167198e-01,  5.59480998e+00,
        8.49816760e-01, -5.16740292e+00])
```

In [53]:
```
print(lr.intercept_)
```
```
113.06958169897533
```

RandomForestRegressor

In [55]:
```
# 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 69330.5793802937 84297.61295586039
RMSE train data, RMSE test data 263.3070059460889 263.3070059460889
R2 train data, R2 test data 0.004439028231778108 0.0016235010522728244
```

RandomForestRegressor with cross validation

K Fold

```
In [56]:
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 -10551.873759576327 20483.13796637598
All scores [-2.32245378e+04 -2.54793971e+04 -2.27525248e+04 -1.85290858e+04
 -9.76308532e+03 -3.81447221e+03 -1.66989459e+03 -2.79091605e+02
 -6.47780365e+00 -1.70615532e-01]
```

```
In [59]:
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 -10577.372638150822 20570.28700508069
All scores [-2.33167424e+04 -2.56046658e+04 -2.28279431e+04 -1.85389502e+04
 -9.77832271e+03 -3.75463858e+03 -1.66818374e+03 -2.77584768e+02
 -6.52443473e+00 -1.70615237e-01]
```

```
In [60]:
#random_forest.score
```

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)

rfr_cv

Bayesian

```
In [61]:
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=0.10, 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('############################################')

X_poly =  PolynomialFeatures(degree = 2).fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X_poly, y, test_size=0.10, 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
```

```
Data Not Normalized, Absolute ACR as the target
MSE train data, MSE test data 71304.03661812235 84203.61150693354
RMSE train data, RMSE test data 267.02815697623043 267.02815697623043
R2 train data, R2 test data 7.173424597406441e-05 2.15890285020226e-05
############################################
Bayesian on Polynomial fit
MSE train data, MSE test data 71299.02288901998 84223.59647978879
RMSE train data, RMSE test data 267.0187687954163 267.0187687954163
R2 train data, R2 test data 0.000142044002392816 -0.00021574688906067507
```

Bayesian with Cross Validation

Polynomial X is used

In [62]:
```
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 -10408.982629315093 20199.334670222353
All Scores [-2.27900483e+04 -2.50498000e+04 -2.25729406e+04 -1.83335454e+04
 -9.71570977e+03 -3.72224820e+03 -1.62521977e+03 -2.74144907e+02
 -5.99871183e+00 -1.70606684e-01]
```

In [63]:
```
# interpret coeffs
# https://scikit-learn.org/stable/auto_examples/inspection/plot_linear_model_coefficien
```

The following can be ignored, Kfold cross validation is already considered above

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 [ ]:
```