Simple Regression Dataset - Linear Regression vs XGBoost

Model is trained with XGBoost installed in notebook instance

In the later examples, we will train using SageMaker's XGBoost algorithm.

Training on SageMaker takes several minutes (even for simple dataset).

If algorithm is supported on Python, we will try them locally on notebook instance

This allows us to quickly learn an algorithm, understand tuning options and then finally train on SageMaker Cloud

In this exercise, let's compare XGBoost and Linear Regression for simple regression dataset

In [1]:
```
#DWB#2023-07-12# Added cell
#DWB#uninstalling a previous install to show full install process
!echo "Y" | pip uninstall xgboost
```

```
Found existing installation: xgboost 1.7.6
Uninstalling xgboost-1.7.6:
  Would remove:
    /home/ec2-user/anaconda3/envs/python3/lib/python3.10/site-packages/xgboost-1.7.6.dist-info/*
    /home/ec2-user/anaconda3/envs/python3/lib/python3.10/site-packages/xgboost.libs/libgomp-a34b3233.so.1.0.0
    /home/ec2-user/anaconda3/envs/python3/lib/python3.10/site-packages/xgboost/*
Proceed (Y/n)?   Successfully uninstalled xgboost-1.7.6
```

In [2]:
```
# Install xgboost in notebook instance.
#### Command to install xgboost
!pip install xgboost
```

```
Looking in indexes: https://pypi.org/simple, https://pip.repos.neuron.amazonaws.com
Collecting xgboost
  Using cached xgboost-1.7.6-py3-none-manylinux2014_x86_64.whl (200.3 MB)
Requirement already satisfied: numpy in /home/ec2-user/anaconda3/envs/python3/lib/python3.10/site-packages (from xgb
oost) (1.22.3)
Requirement already satisfied: scipy in /home/ec2-user/anaconda3/envs/python3/lib/python3.10/site-packages (from xgb
oost) (1.10.1)
Installing collected packages: xgboost
Successfully installed xgboost-1.7.6
```

In [3]:

```
import sys
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error, mean_absolute_error


# XGBoost
import xgboost as xgb
# Linear Regression
from sklearn.linear_model import LinearRegression
```

In [4]:

```
# All data
df = pd.read_csv('linear_all.csv')
```

In [5]:

```
df.head()
```

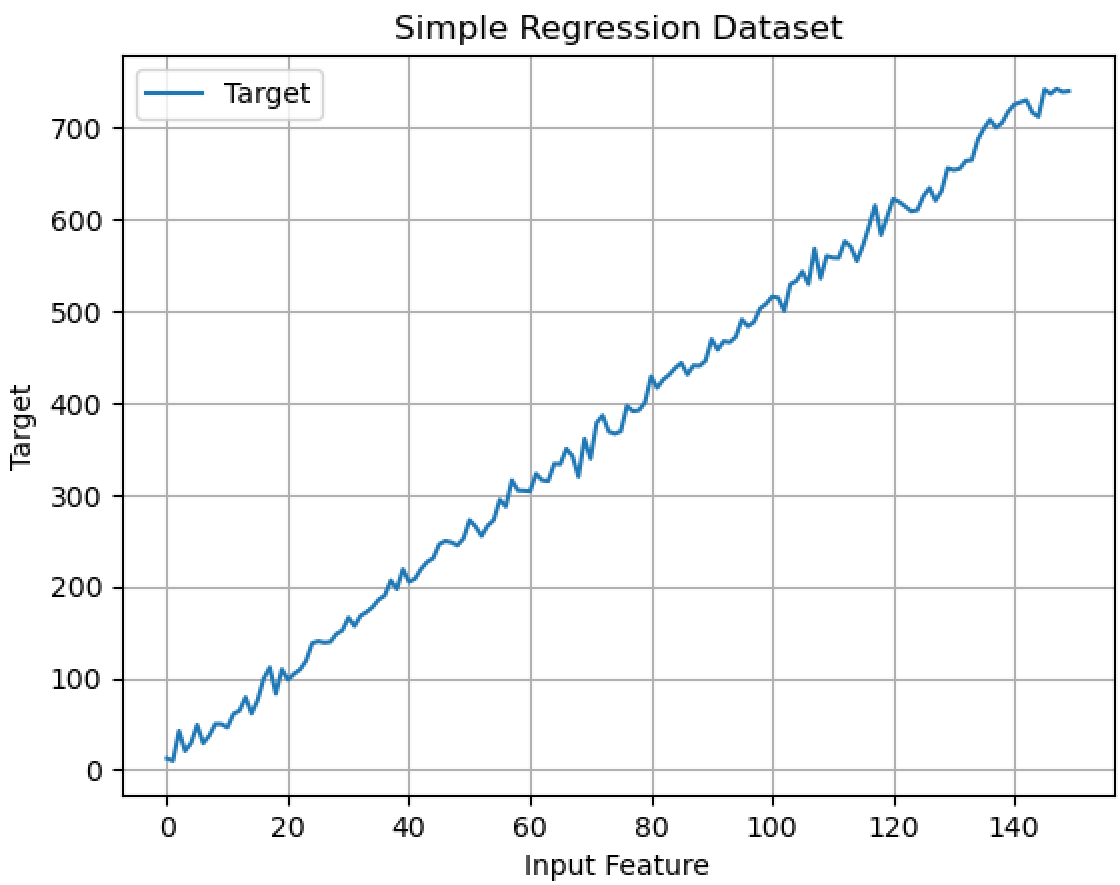
Out[5]:

	x	y
0	0	12.412275
1	1	9.691298
2	2	42.307712
3	3	20.479079
4	4	29.096098

In [6]:

```
plt.plot(df.x,df.y,label='Target')
plt.grid(True)
plt.xlabel('Input Feature')
```

```
plt.ylabel('Target')
plt.legend()
plt.title('Simple Regression Dataset')
plt.show()
```



In [7]:

```
# Let's Load Training and Validation Datasets
train_file = 'linear_train.csv'
validation_file = 'linear_validation.csv'

# Specify the column names as the file does not have column header
df_train = pd.read_csv(train_file,names=['y','x'])
df_validation = pd.read_csv(validation_file,names=['y','x'])
```

In [8]:

```
df_train.head()
```

Out[8]:

	y	x
0	425.457270	82
1	687.275162	134
2	554.643782	114
3	219.007382	42
4	560.269533	109

In [9]:
```
df_validation.head()
```

Out[9]:

	y	x
0	342.264067	67
1	60.951235	11
2	315.592889	62
3	700.097979	137
4	535.676139	108

In [10]:
```
plt.scatter(df_train.x,df_train.y,label='Training',marker='.')
plt.scatter(df_validation.x,df_validation.y,label='Validation',marker='.')
plt.grid(True)
plt.xlabel('Input Feature')
plt.ylabel('Target')
plt.title('Simple Regression Dataset')
plt.legend()
plt.show()
```

In [11]:
```
X_train = df_train.iloc[:,1:] # Features: 1st column onwards
y_train = df_train.iloc[:,0].ravel() # Target: 0th column

X_validation = df_validation.iloc[:,1:]
y_validation = df_validation.iloc[:,0].ravel()
```

In [12]:
```
# Create an instance of XGBoost Regressor
# XGBoost Training Parameter Reference:
#   https://github.com/dmlc/xgboost/blob/master/doc/parameter.md
regressor = xgb.XGBRegressor()
```

In [69]:
```
# Default Options #DWB# shown by inspecting regressor
#DWB# I don't like the format from just feeding in the name
```

```
#regressor
print(str(regressor)) # more than were there in his lecture
print(  (f"\nFor my default,\n  regression.booster = {regressor.booster},\n"
         "whereas in Chandra's lecture\n"
         "(\"Lab - Training Simple Regression\",\n"
         "at 2:08 in the video, as seen 20230720T185200-0600),\n"
         "we saw\n  regression.booster = 'gbtree'."
        )
     )
```

```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=None, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             n_estimators=100, n_jobs=None, num_parallel_tree=None,
             predictor=None, random_state=None, ...)

For my default,
  regression.booster = None,
whereas in Chandra's lecture
("Lab - Training Simple Regression",
at 2:08 in the video, as seen 20230720T185200-0600),
we saw
  regression.booster = 'gbtree'.
```

In [59]:

```
# Train the model
# Provide Training Dataset and Validation Dataset
# XGBoost reports training and validation error
regressor.fit(X_train,y_train,
              eval_set = [(X_train, y_train), (X_validation, y_validation)])
              #DWB# a semicolon prevents printing out the XGBRegressor again
```

```
[0]	validation_0-rmse:312.59224	validation_1-rmse:309.71131
[1]	validation_0-rmse:223.90627	validation_1-rmse:221.75470
[2]	validation_0-rmse:160.90971	validation_1-rmse:156.30206
[3]	validation_0-rmse:115.67703	validation_1-rmse:111.37984
[4]	validation_0-rmse:83.50769	validation_1-rmse:80.23268
[5]	validation_0-rmse:60.28321	validation_1-rmse:56.99042
[6]	validation_0-rmse:43.79396	validation_1-rmse:40.56110
[7]	validation_0-rmse:32.11387	validation_1-rmse:29.46322
[8]	validation_0-rmse:23.79751	validation_1-rmse:21.48173
[9]	validation_0-rmse:17.93956	validation_1-rmse:16.86060
[10]	validation_0-rmse:13.83357	validation_1-rmse:14.40190
[11]	validation_0-rmse:10.89259	validation_1-rmse:13.18560
[12]	validation_0-rmse:8.73792	validation_1-rmse:12.70564
[13]	validation_0-rmse:7.32471	validation_1-rmse:12.86105
[14]	validation_0-rmse:6.22650	validation_1-rmse:12.87767
[15]	validation_0-rmse:5.58807	validation_1-rmse:13.13437
[16]	validation_0-rmse:5.02747	validation_1-rmse:13.03502
[17]	validation_0-rmse:4.58230	validation_1-rmse:13.06587
[18]	validation_0-rmse:4.25840	validation_1-rmse:13.16091
[19]	validation_0-rmse:4.02088	validation_1-rmse:13.20965
[20]	validation_0-rmse:3.83337	validation_1-rmse:13.36338
[21]	validation_0-rmse:3.57168	validation_1-rmse:13.49237
[22]	validation_0-rmse:3.36921	validation_1-rmse:13.62991
[23]	validation_0-rmse:3.22752	validation_1-rmse:13.75530
[24]	validation_0-rmse:3.09152	validation_1-rmse:13.81823
[25]	validation_0-rmse:2.97103	validation_1-rmse:13.82356
[26]	validation_0-rmse:2.82581	validation_1-rmse:13.91419
[27]	validation_0-rmse:2.75075	validation_1-rmse:13.99817
[28]	validation_0-rmse:2.66672	validation_1-rmse:14.03056
[29]	validation_0-rmse:2.64305	validation_1-rmse:14.06679
[30]	validation_0-rmse:2.61180	validation_1-rmse:14.08419
[31]	validation_0-rmse:2.55397	validation_1-rmse:14.14185
[32]	validation_0-rmse:2.43906	validation_1-rmse:14.21921
[33]	validation_0-rmse:2.31654	validation_1-rmse:14.26301
[34]	validation_0-rmse:2.29145	validation_1-rmse:14.30315
[35]	validation_0-rmse:2.19485	validation_1-rmse:14.33316
[36]	validation_0-rmse:2.14014	validation_1-rmse:14.38449
[37]	validation_0-rmse:2.07183	validation_1-rmse:14.35956
[38]	validation_0-rmse:1.95956	validation_1-rmse:14.41703
[39]	validation_0-rmse:1.88531	validation_1-rmse:14.43918
[40]	validation_0-rmse:1.86735	validation_1-rmse:14.47058
[41]	validation_0-rmse:1.82048	validation_1-rmse:14.50962
```

```
[42]	validation_0-rmse:1.78803	validation_1-rmse:14.50232
[43]	validation_0-rmse:1.69333	validation_1-rmse:14.55050
[44]	validation_0-rmse:1.62249	validation_1-rmse:14.56792
[45]	validation_0-rmse:1.59631	validation_1-rmse:14.56281
[46]	validation_0-rmse:1.58324	validation_1-rmse:14.58330
[47]	validation_0-rmse:1.52950	validation_1-rmse:14.59688
[48]	validation_0-rmse:1.46826	validation_1-rmse:14.62189
[49]	validation_0-rmse:1.40424	validation_1-rmse:14.65027
[50]	validation_0-rmse:1.38862	validation_1-rmse:14.65419
[51]	validation_0-rmse:1.34541	validation_1-rmse:14.66677
[52]	validation_0-rmse:1.29872	validation_1-rmse:14.69935
[53]	validation_0-rmse:1.26157	validation_1-rmse:14.72340
[54]	validation_0-rmse:1.24904	validation_1-rmse:14.73585
[55]	validation_0-rmse:1.19405	validation_1-rmse:14.75435
[56]	validation_0-rmse:1.15900	validation_1-rmse:14.77241
[57]	validation_0-rmse:1.14578	validation_1-rmse:14.77590
[58]	validation_0-rmse:1.10874	validation_1-rmse:14.78746
[59]	validation_0-rmse:1.09985	validation_1-rmse:14.79944
[60]	validation_0-rmse:1.06107	validation_1-rmse:14.80253
[61]	validation_0-rmse:1.04570	validation_1-rmse:14.80491
[62]	validation_0-rmse:1.01881	validation_1-rmse:14.82609
[63]	validation_0-rmse:0.98739	validation_1-rmse:14.84663
[64]	validation_0-rmse:0.95160	validation_1-rmse:14.85792
[65]	validation_0-rmse:0.91730	validation_1-rmse:14.86138
[66]	validation_0-rmse:0.89162	validation_1-rmse:14.87062
[67]	validation_0-rmse:0.88262	validation_1-rmse:14.87350
[68]	validation_0-rmse:0.85606	validation_1-rmse:14.88405
[69]	validation_0-rmse:0.83039	validation_1-rmse:14.90555
[70]	validation_0-rmse:0.80217	validation_1-rmse:14.90871
[71]	validation_0-rmse:0.79483	validation_1-rmse:14.91097
[72]	validation_0-rmse:0.76935	validation_1-rmse:14.91671
[73]	validation_0-rmse:0.74789	validation_1-rmse:14.92586
[74]	validation_0-rmse:0.71865	validation_1-rmse:14.94531
[75]	validation_0-rmse:0.69811	validation_1-rmse:14.96310
[76]	validation_0-rmse:0.67721	validation_1-rmse:14.96298
[77]	validation_0-rmse:0.66666	validation_1-rmse:14.96477
[78]	validation_0-rmse:0.65925	validation_1-rmse:14.97117
[79]	validation_0-rmse:0.63809	validation_1-rmse:14.97066
[80]	validation_0-rmse:0.62310	validation_1-rmse:14.98477
[81]	validation_0-rmse:0.61763	validation_1-rmse:14.98919
[82]	validation_0-rmse:0.60214	validation_1-rmse:14.99254
[83]	validation_0-rmse:0.59266	validation_1-rmse:15.00204
```

```
[84]	validation_0-rmse:0.57398	validation_1-rmse:15.00881
[85]	validation_0-rmse:0.56816	validation_1-rmse:15.01047
[86]	validation_0-rmse:0.54846	validation_1-rmse:15.01283
[87]	validation_0-rmse:0.53247	validation_1-rmse:15.01265
[88]	validation_0-rmse:0.51886	validation_1-rmse:15.01920
[89]	validation_0-rmse:0.50600	validation_1-rmse:15.01966
[90]	validation_0-rmse:0.48054	validation_1-rmse:15.02951
[91]	validation_0-rmse:0.46820	validation_1-rmse:15.02981
[92]	validation_0-rmse:0.46465	validation_1-rmse:15.03104
[93]	validation_0-rmse:0.46061	validation_1-rmse:15.03510
[94]	validation_0-rmse:0.44996	validation_1-rmse:15.04098
[95]	validation_0-rmse:0.43824	validation_1-rmse:15.05033
[96]	validation_0-rmse:0.43266	validation_1-rmse:15.05148
[97]	validation_0-rmse:0.41700	validation_1-rmse:15.05335
[98]	validation_0-rmse:0.40495	validation_1-rmse:15.05614
[99]	validation_0-rmse:0.39714	validation_1-rmse:15.06363
```

Out[59]:

▾ XGBRegressor

```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=None, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
```

In [70]:

```
# Get the Training RMSE and Evaluation RMSE
eval_result = regressor.evals_result()
```

In [71]:

```
eval_result
```

```
Out[71]: {'validation_0': OrderedDict([('rmse',
                        [312.59224443956623,
                         223.9062718955829,
                         160.90971257530234,
                         115.67703009451913,
                         83.50768779168386,
                         60.283208020661135,
                         43.793956372626674,
                         32.113874229120576,
                         23.79751491807248,
                         17.939562072260912,
                         13.833572169121286,
                         10.892590815726312,
                         8.73791751236688,
                         7.324714292686442,
                         6.226497539093337,
                         5.588068603709141,
                         5.027468499136668,
                         4.5822954413292765,
                         4.258395317603746,
                         4.020877834191251,
                         3.8333733399107586,
                         3.571677478588978,
                         3.36920522463644,
                         3.2275216671848845,
                         3.0915225438562635,
                         2.971027800139437,
                         2.825807810626455,
                         2.7507484277234404,
                         2.6667203073602184,
                         2.643050885873713,
                         2.6118035769926244,
                         2.553967852960363,
                         2.4390632739562164,
                         2.3165403222180116,
                         2.291446259582387,
                         2.1948527673250458,
                         2.1401367936737814,
                         2.071827822077863,
                         1.959562939152379,
                         1.8853082916765906,
                         1.8673493607208607,
```

```
1.8204783491868546,
1.7880322185187443,
1.6933261114433975,
1.6224914044589827,
1.5963062073749015,
1.5832404508430933,
1.529503241432709,
1.4682567687850292,
1.4042385122381866,
1.388616387828227,
1.345411752315979,
1.2987154259514186,
1.2615678721379209,
1.2490425668623037,
1.1940456540499391,
1.1590023725766432,
1.1457797980406936,
1.1087368898821852,
1.0998508264506635,
1.061068693285586,
1.0457042356376076,
1.0188069123823136,
0.987389989262799,
0.9516035980882488,
0.9173029417581924,
0.8916179469348092,
0.8826216632316954,
0.8560644331668469,
0.8303934570446853,
0.8021701630524387,
0.7948305529140819,
0.7693522212451417,
0.7478860052711606,
0.7186500559598827,
0.698108053213934,
0.6772128009461164,
0.6666588496788669,
0.6592492224467972,
0.6380932291213642,
0.6231037456759634,
0.6176321061075527,
0.6021427058014956,
```

```
                    0.5926567748567947,
                    0.5739822687029278,
                    0.5681601507802188,
                    0.5484615859745025,
                    0.5324655994880023,
                    0.5188632917681142,
                    0.5060000368574907,
                    0.4805392226086445,
                    0.4681968181512183,
                    0.46465296783369836,
                    0.4606079476457048,
                    0.4499581862331583,
                    0.4382420984702145,
                    0.43265531470613355,
                    0.4170010581169265,
                    0.404953785052106,
                    0.3971396673124024])]),
      'validation_1': OrderedDict([('rmse',
                   [309.7113103426434,
                    221.75470293522946,
                    156.30206495363524,
                    111.3798431854292,
                    80.2326784169725,
                    56.9904249715414,
                    40.5610956503098,
                    29.46322069420898,
                    21.4817269305809,
                    16.860600728646613,
                    14.401902532617553,
                    13.185601887203395,
                    12.705641245083322,
                    12.861048823949146,
                    12.877670934572821,
                    13.134370129687452,
                    13.035022216296776,
                    13.065870118827426,
                    13.160912814988293,
                    13.209648280055221,
                    13.363376091308906,
                    13.492369966166498,
                    13.629907572417679,
                    13.755301577747202,
```

```
13.818231872257918,
13.823555478866261,
13.914188919231433,
13.998169372591173,
14.030559285959656,
14.066791072173704,
14.084190926440336,
14.141853247390847,
14.21920988095194,
14.263005610431703,
14.303151869944255,
14.33316212033954,
14.384485585860057,
14.359562443941973,
14.417026565450362,
14.439177867164002,
14.470575495877169,
14.509623271017723,
14.502322535229967,
14.55050056558313,
14.567915927658929,
14.562809743356159,
14.583296017860173,
14.596879317994112,
14.621892149599667,
14.650266744870846,
14.654193076384798,
14.66677489946595,
14.699345517303195,
14.72339585041756,
14.735849097872807,
14.754350669688955,
14.772409986699637,
14.775899194194947,
14.787459329212114,
14.799436370533469,
14.802527812393945,
14.804907837875287,
14.826087583519099,
14.846627622276364,
14.857917281479475,
14.86137772403098,
```

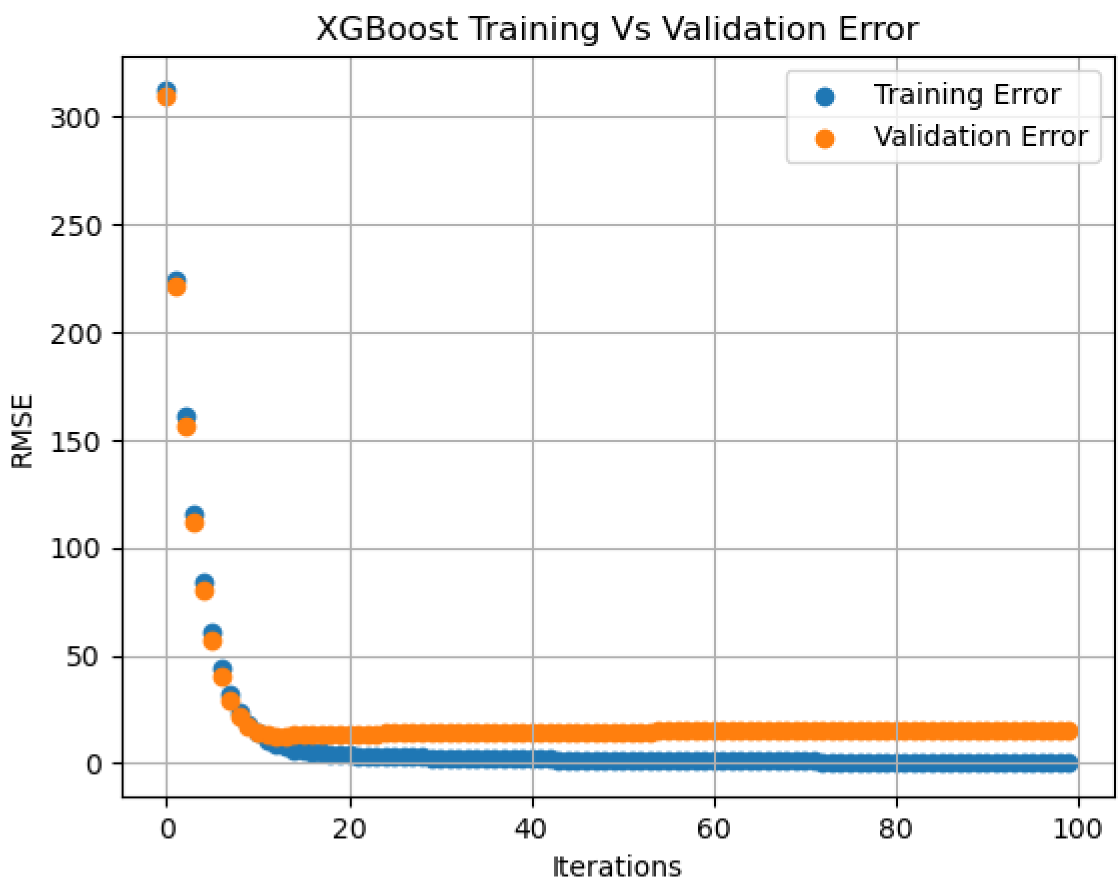
```
14.870620378120655,
14.873504909103655,
14.884045768560139,
14.905554008346687,
14.908705843302883,
14.910966943418211,
14.916711507184633,
14.925862007959259,
14.945307666564858,
14.963100046760035,
14.962978536312756,
14.964768049119506,
14.971169109091559,
14.970662692405877,
14.984769810214521,
14.98918882072408,
14.992543389907583,
15.002035020260571,
15.008813215156009,
15.010473722834888,
15.012832216402469,
15.01265271535071,
15.019196116072465,
15.019662803665737,
15.029509283926032,
15.029805031808078,
15.031042800107162,
15.03509553887574,
15.040981495452273,
15.050331122533427,
15.051477134898823,
15.05334632673122,
15.056137737917748,
15.063627994613592])])}
```

In [72]:
```
training_rounds = range(len(eval_result['validation_0']['rmse']))
```

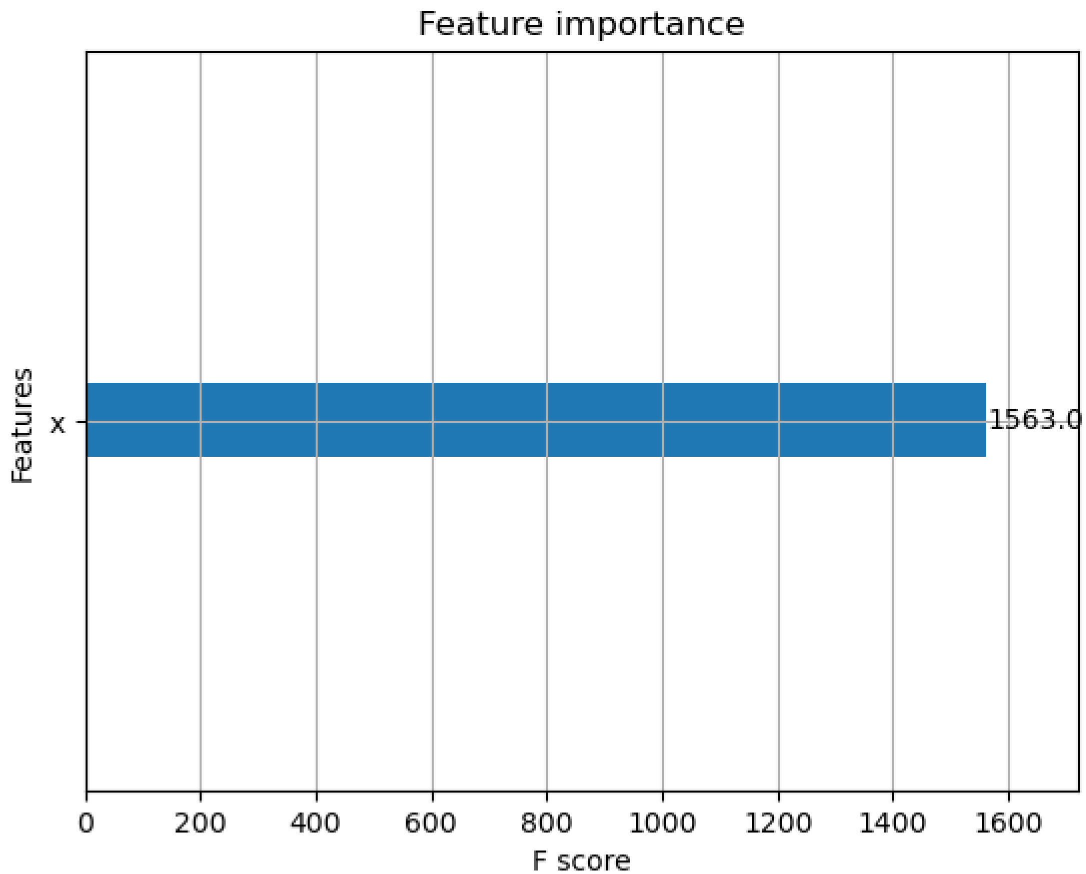
In [73]:
```
print(training_rounds)
```

```
range(0, 100)
```

In [74]:
```
#DWB#  Note that the iteration number is n_trees_used+1 (I think the '+1' is there),
#DWB#+ so we're seeing how quality improves as we add more trees
plt.scatter(x=training_rounds,y=eval_result['validation_0']['rmse'],label='Training Error')
plt.scatter(x=training_rounds,y=eval_result['validation_1']['rmse'],label='Validation Error')
plt.grid(True)
plt.xlabel('Iterations')
plt.ylabel('RMSE')
plt.title('XGBoost Training Vs Validation Error')
plt.legend()
plt.show()
```



In [75]:
```
xgb.plot_importance(regressor) #DWB# later, we will have more than one feature
plt.show()
```



Validation Dataset Compare Actual and Predicted

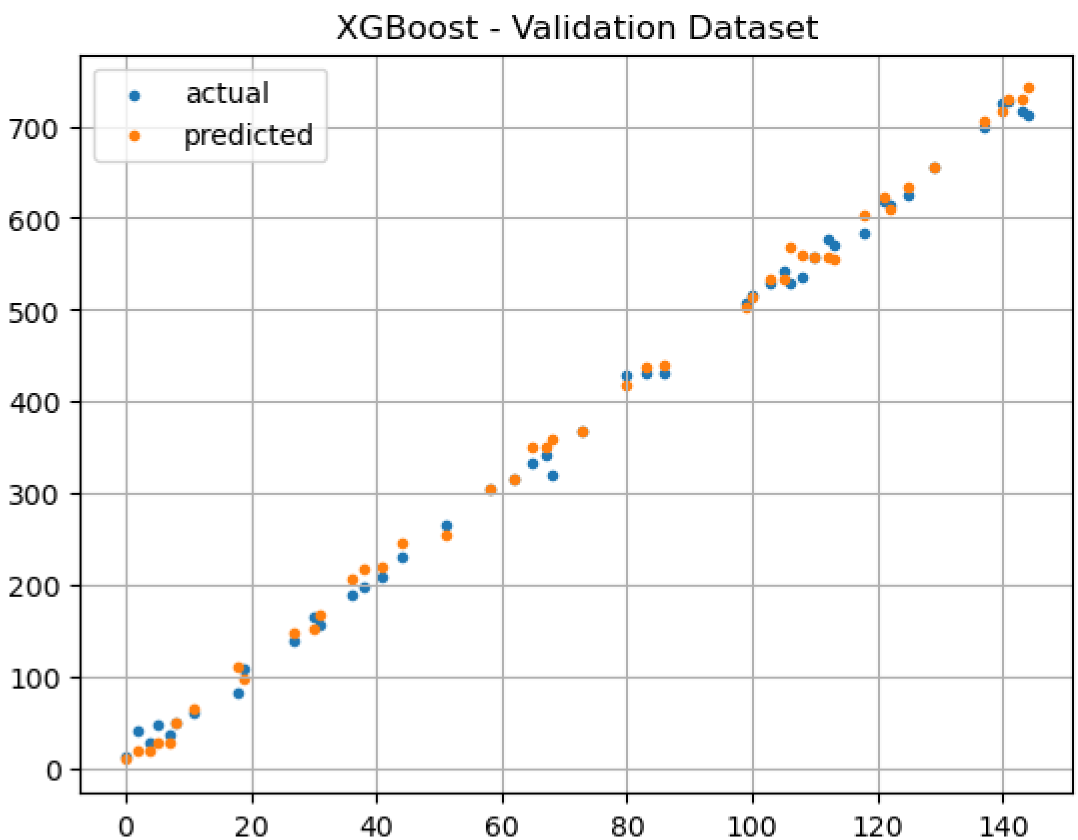
In [76]:
```
result = regressor.predict(X_validation)
```

In [77]:
```
result[:5]
```

Out[77]:
```
array([350.00354 ,  64.406364, 314.84644 , 705.8229  , 560.227   ],
      dtype=float32)
```

In [78]:
```
plt.title('XGBoost - Validation Dataset')
plt.scatter(df_validation.x,df_validation.y,label='actual',marker='.')
plt.scatter(df_validation.x,result,label='predicted',marker='.')
```

```
plt.grid(True)
plt.legend()
plt.show()
```

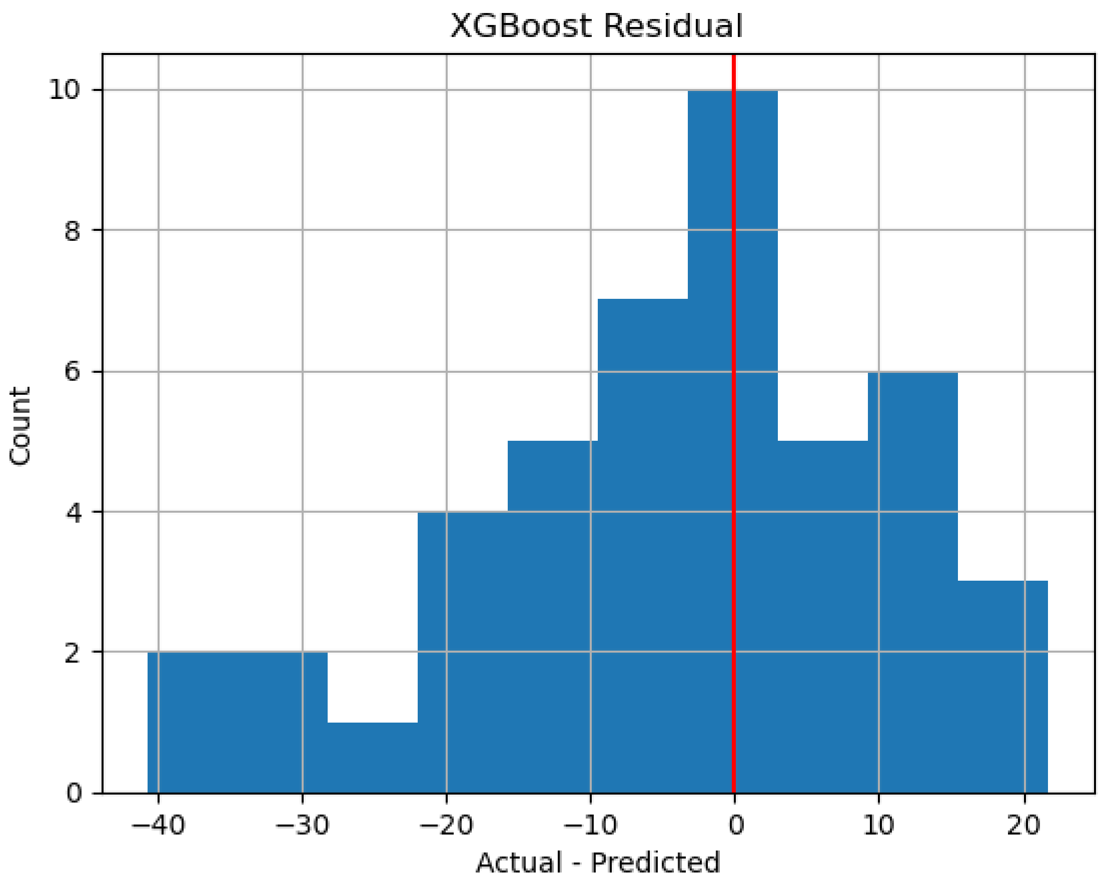


In [79]:
```
# RMSE Metrics
print('XGBoost Algorithm Metrics')
mse = mean_squared_error(df_validation.y,result)
print(" Mean Squared Error: {0:.2f}".format(mse))
print(" Root Mean Square Error: {0:.2f}".format(mse**.5))
```

```
XGBoost Algorithm Metrics
 Mean Squared Error: 226.91
 Root Mean Square Error: 15.06
```

In [80]:
```
# Residual
# Over prediction and Under Prediction needs to be balanced
```

```
# Training Data Residuals
residuals = df_validation.y - result
plt.hist(residuals)
plt.grid(True)
plt.xlabel('Actual - Predicted')
plt.ylabel('Count')
plt.title('XGBoost Residual')
plt.axvline(color='r')
plt.show()
```

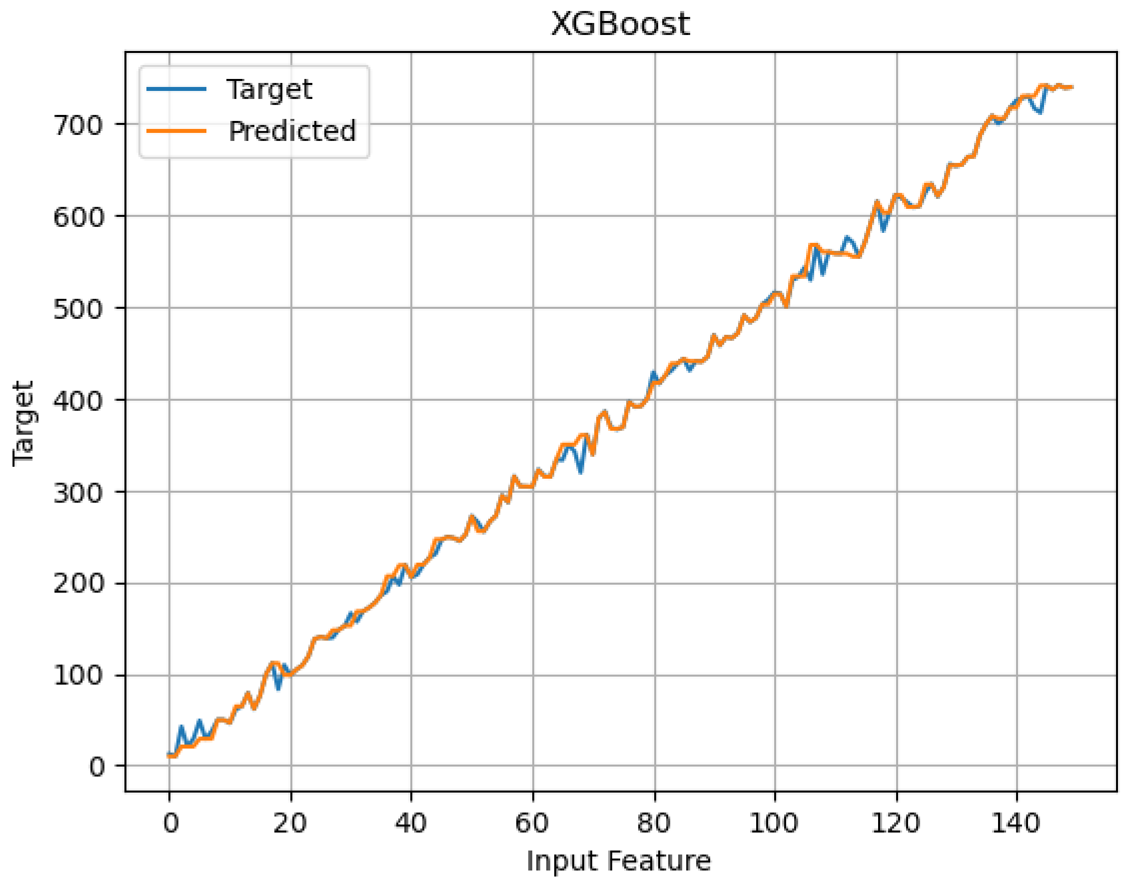


In [31]:
```
# Count number of values greater than zero and less than zero
value_counts = (residuals > 0).value_counts(sort=False)
```

```
print(' Under Estimation: {0}'.format(value_counts[True]))
print(' Over  Estimation: {0}'.format(value_counts[False]))
```

```
 Under Estimation: 22
 Over  Estimation: 23
```

In [32]:
```
# Plot for entire dataset
plt.plot(df.x,df.y,label='Target')
plt.plot(df.x,regressor.predict(df[['x']]) ,label='Predicted')
plt.grid(True)
plt.xlabel('Input Feature')
plt.ylabel('Target')
plt.legend()
plt.title('XGBoost')
plt.show()
```



Linear Regression Algorithm

In [33]:
```
lin_regressor = LinearRegression()
```

In [35]:
```
lin_regressor.fit(X_train,y_train); #semicolon prevents signature of object
```

Compare Weights assigned by Linear Regression.

Original Function: 5*x + 8 + some noise

In [36]:
```
lin_regressor.coef_
```

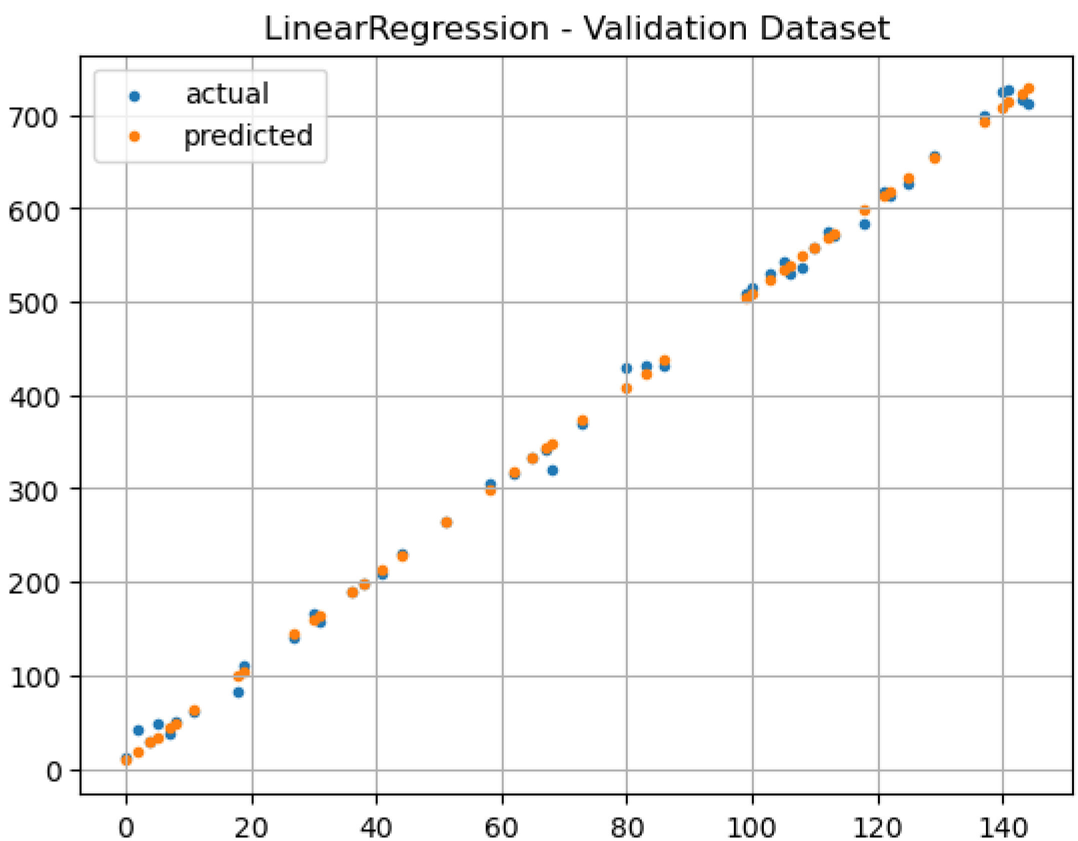
Out[36]: array([4.99777227])

In [37]:
```
lin_regressor.intercept_
```

Out[37]: 8.683965388503225

In [38]:
```
result = lin_regressor.predict(df_validation[['x']])
```

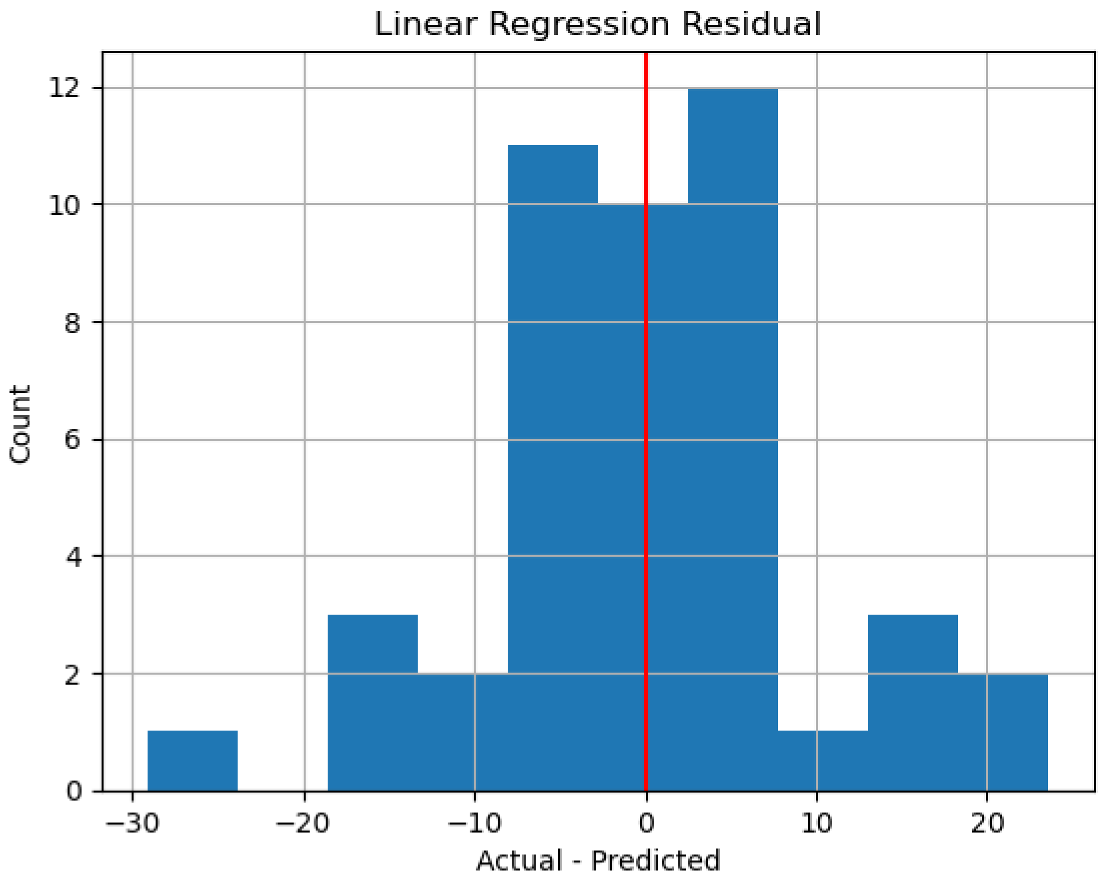
In [39]:
```
plt.title('LinearRegression - Validation Dataset')
plt.scatter(df_validation.x,df_validation.y,label='actual',marker='.')
plt.scatter(df_validation.x,result,label='predicted',marker='.')
plt.grid(True)
plt.legend()
plt.show()
```



In [40]:
```
# RMSE Metrics
print('Linear Regression Metrics')
mse = mean_squared_error(df_validation.y,result)
print(" Mean Squared Error: {0:.2f}".format(mse))
print(" Root Mean Square Error: {0:.2f}".format(mse**.5))
```

```
Linear Regression Metrics
 Mean Squared Error: 99.10
 Root Mean Square Error: 9.95
```

In [41]:
```
# Residual
# Over prediction and Under Prediction needs to be balanced
# Training Data Residuals
residuals = df_validation.y - result
plt.hist(residuals)
plt.grid(True)
plt.xlabel('Actual - Predicted')
plt.ylabel('Count')
plt.title('Linear Regression Residual')
plt.axvline(color='r')
plt.show()
```

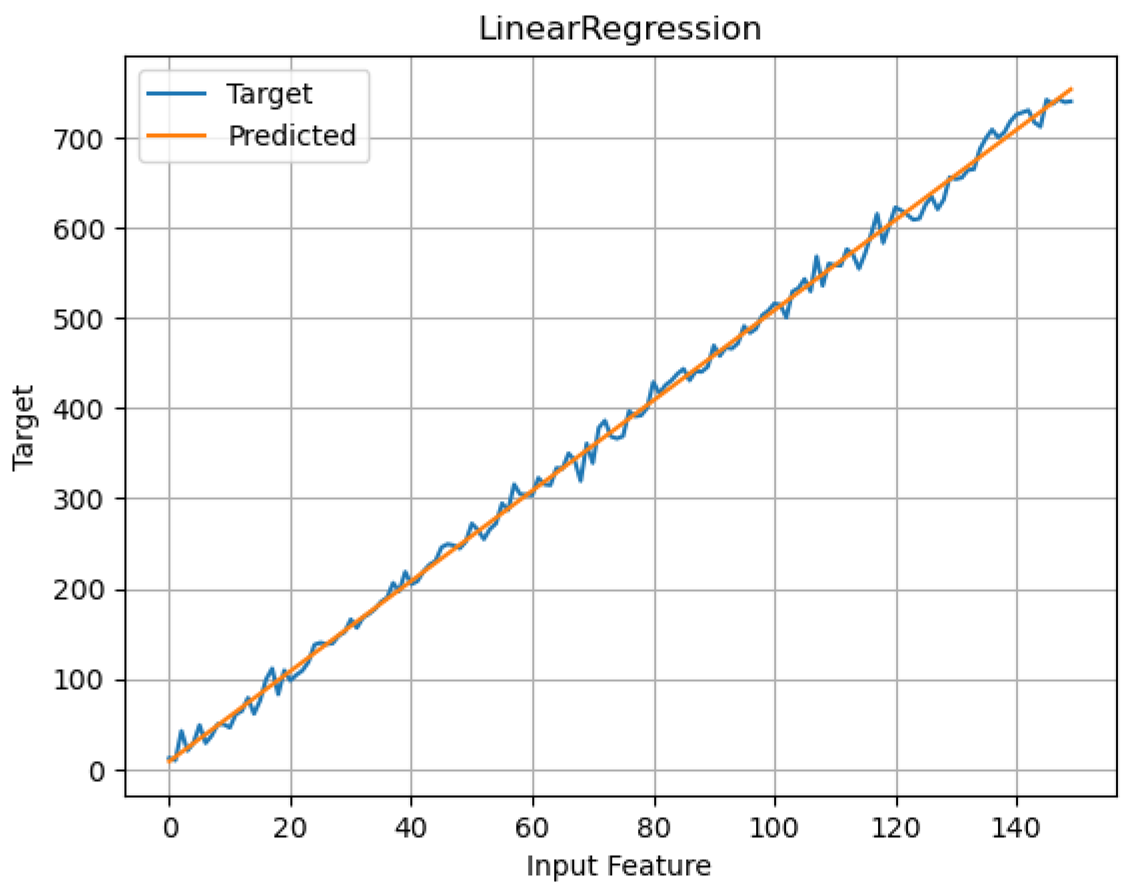


In [42]:
```
# Count number of values greater than zero and less than zero
value_counts = (residuals > 0).value_counts(sort=False)

print(' Under Estimation: {0}'.format(value_counts[True]))
print(' Over  Estimation: {0}'.format(value_counts[False]))
```

```
 Under Estimation: 24
 Over  Estimation: 21
```

In [43]:
```
# Plot for entire dataset
plt.plot(df.x,df.y,label='Target')
plt.plot(df.x,lin_regressor.predict(df[['x']]) ,label='Predicted')
plt.grid(True)
plt.xlabel('Input Feature')
```

```
plt.ylabel('Target')
plt.legend()
plt.title('LinearRegression')
plt.show()
```



Input Features - Outside range used for training

XGBoost Prediction has an upper and lower bound (applies to tree based algorithms)

Linear Regression extrapolates

In [44]:
```
# True Function
def straight_line(x):
```

```
    return 5*x + 8
```

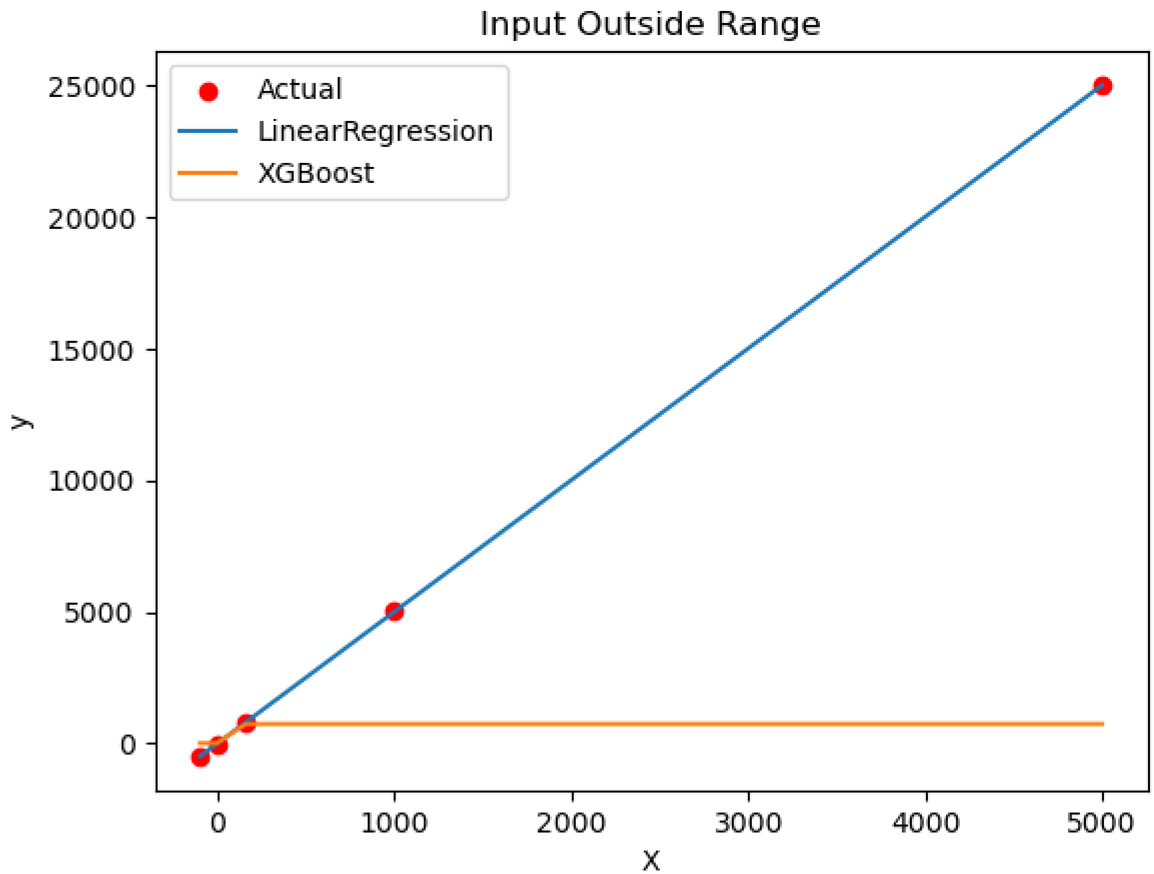
In [45]:
```
# X is outside range of training samples
X = np.array([-100,-5,160,1000,5000])
y = straight_line(X)

df_tmp = pd.DataFrame({'x':X,'y':y})
df_tmp['xgboost']=regressor.predict(df_tmp[['x']])
df_tmp['linear']=lin_regressor.predict(df_tmp[['x']])
```

In [46]:
```
df_tmp
```

Out[46]:

	x	y	xgboost	linear
0	-100	-492	9.905086	-491.093262
1	-5	-17	9.905086	-16.304896
2	160	808	739.950562	808.327528
3	1000	5008	739.950562	5006.456235
4	5000	25008	739.950562	24997.545312

In [47]:
```
# XGBoost Predictions have an upper bound and lower bound
# Linear Regression Extrapolates
plt.scatter(df_tmp.x,df_tmp.y,label='Actual',color='r')
plt.plot(df_tmp.x,df_tmp.linear,label='LinearRegression')
plt.plot(df_tmp.x,df_tmp.xgboost,label='XGBoost')
plt.legend()
plt.xlabel('X')
plt.ylabel('y')
plt.title('Input Outside Range')
plt.show()
```



In [48]:
```
# X is inside range of training samples
X = np.array([0,1,3,5,7,9,11,15,18,125])
y = straight_line(X)

df_tmp = pd.DataFrame({'x':X,'y':y})
df_tmp['xgboost']=regressor.predict(df_tmp[['x']])
df_tmp['linear']=lin_regressor.predict(df_tmp[['x']])
```

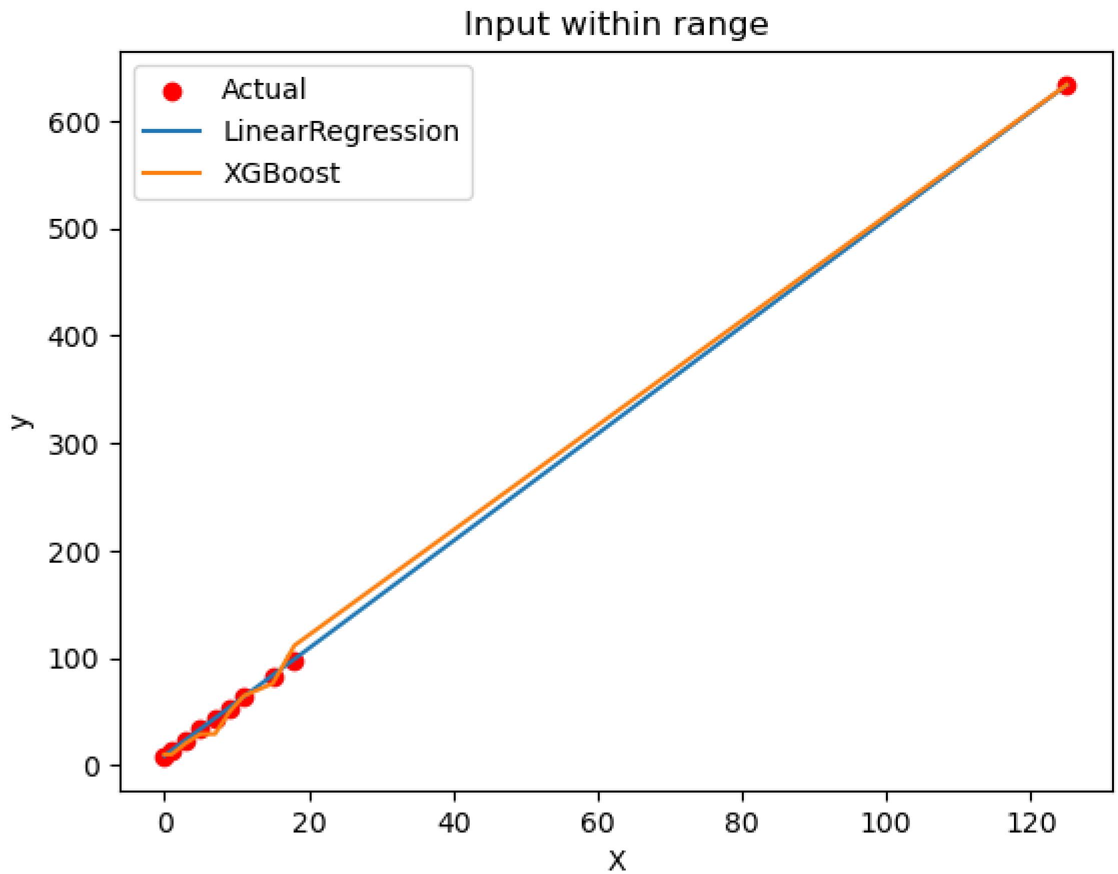
In [49]:
```
df_tmp
```

Out[49]:

	x	y	xgboost	linear
0	0	8	9.905086	8.683965
1	1	13	9.905086	13.681738
2	3	23	20.523394	23.677282
3	5	33	28.935297	33.672827
4	7	43	28.935297	43.668371
5	9	53	49.514168	53.663916
6	11	63	64.406364	63.659460
7	15	83	75.930733	83.650549
8	18	98	111.305298	98.643866
9	125	633	633.698364	633.405499

In [50]:

```
# XGBoost Predictions have an upper bound and lower bound
# Linear Regression Extrapolates
plt.scatter(df_tmp.x,df_tmp.y,label='Actual',color='r')
plt.plot(df_tmp.x,df_tmp.linear,label='LinearRegression')
plt.plot(df_tmp.x,df_tmp.xgboost,label='XGBoost')
plt.legend()
plt.xlabel('X')
plt.ylabel('y')
plt.title('Input within range')
plt.show()
```



Summary

1. Use sagemaker notebook as your own server on the cloud
2. Install python packages
3. Train directly on SageMaker Notebook (for small datasets, it takes few seconds).
4. Once happy with algorithm and performance, you can train on sagemaker cloud (takes several minutes even for small datasets)
5. Not all algorithms are available for installation (for example: AWS algorithms like DeepAR are available only in SageMaker)
6. In this exercise, we installed XGBoost and compared performance of XGBoost model and Linear Regression