# Quadratic 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 Quadratic regression dataset

In [1]:

```
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 [2]:

```
df = pd.read_csv(r'C:\Users\309962\Desktop\quadratic_all.csv')
```

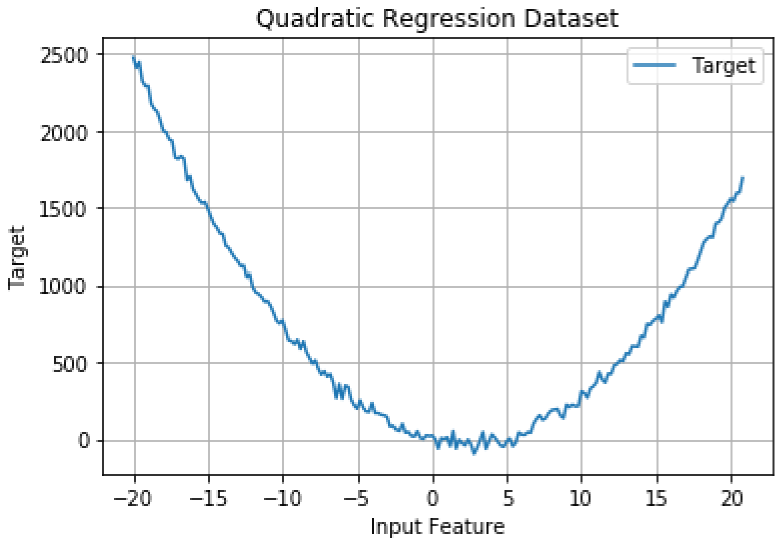
In [3]:

```
df.head()
```

Out[3]:

| | x | y |
|---|---|---|
| **0** | -20.0 | 2473.236825 |
| **1** | -19.8 | 2405.673895 |
| **2** | -19.6 | 2444.523136 |
| **3** | -19.4 | 2320.437236 |
| **4** | -19.2 | 2288.088295 |

In [4]:
```
plt.plot(df.x,df.y,label='Target')
plt.grid(True)
plt.xlabel('Input Feature')
plt.ylabel('Target')
plt.legend()
plt.title('Quadratic Regression Dataset')
plt.show()
```




In [6]:
```
train_file = r'C:\Users\309962\Desktop\quadratic_train.csv'
validation_file = r'C:\Users\309962\Desktop\quadratic_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 [7]:
```
df_train.head()
```

Out[7]:

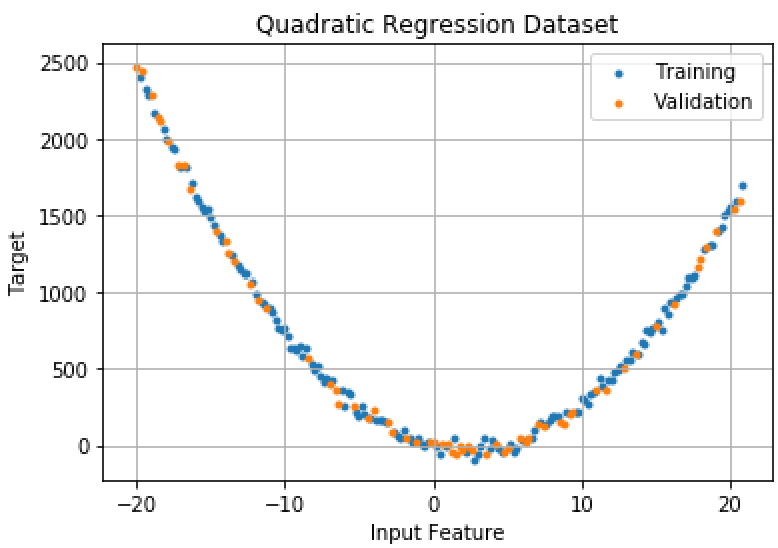
| | y | x |
|---|---|---|
| **0** | 343.968005 | 10.8 |
| **1** | 1585.894405 | -15.8 |
| **2** | 1497.303317 | 19.6 |
| **3** | 769.909912 | -10.4 |
| **4** | 1173.230755 | -13.2 |

In [8]:
```
df_validation.head()
```

Out[8]:

| | y | x |
|---|---|---|
| 0 | 1824.856344 | -17.2 |
| 1 | 16.997917 | -1.2 |
| 2 | 1832.141730 | -16.8 |
| 3 | 1395.206684 | 19.0 |
| 4 | 145.840543 | -3.0 |

In [9]:
```
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('Quadratic Regression Dataset')
plt.legend()
plt.show()
```




In [10]:
```

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 [11]:
```
# Create an instance of XGBoost Regressor
# XGBoost Training Parameter Reference:
#   https://github.com/dmlc/xgboost/blob/master/doc/parameter.md
regressor = xgb.XGBRegressor()
```

In [12]:
```
regressor
```

Out[12]:
```
XGBRegressor(base_score=None, booster=None, colsample_bylevel=None,
             colsample_bynode=None, colsample_bytree=None, gamma=None,
             gpu_id=None, importance_type='gain', interaction_constraints=None,
             learning_rate=None, max_delta_step=None, max_depth=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             n_estimators=100, n_jobs=None, num_parallel_tree=None,
             objective='reg:squarederror', random_state=None, reg_alpha=None,
             reg_lambda=None, scale_pos_weight=None, subsample=None,
             tree_method=None, validate_parameters=False, verbosity=None)
```

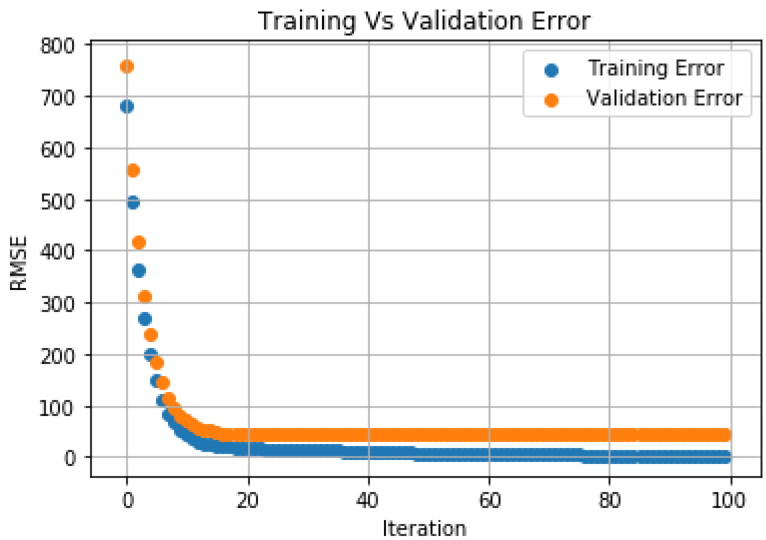
In [13]:
```
regressor.fit(X_train,y_train, eval_set = [(X_train, y_train), (X_validation, y_v
```

```
[0]	validation_0-rmse:680.75653	validation_1-rmse:759.28186
[1]	validation_0-rmse:496.64975	validation_1-rmse:558.76227
[2]	validation_0-rmse:364.40195	validation_1-rmse:416.74503
[3]	validation_0-rmse:268.61850	validation_1-rmse:314.03879
[4]	validation_0-rmse:198.73166	validation_1-rmse:239.39935
[5]	validation_0-rmse:148.15569	validation_1-rmse:184.01250
[6]	validation_0-rmse:111.41606	validation_1-rmse:143.64578
[7]	validation_0-rmse:85.12823	validation_1-rmse:114.83409
[8]	validation_0-rmse:66.19106	validation_1-rmse:95.02868
[9]	validation_0-rmse:52.48116	validation_1-rmse:80.46168
[10]	validation_0-rmse:42.81858	validation_1-rmse:70.20042
[11]	validation_0-rmse:35.82252	validation_1-rmse:62.60704
[12]	validation_0-rmse:30.72047	validation_1-rmse:57.81083
[13]	validation_0-rmse:27.04723	validation_1-rmse:53.74323
[14]	validation_0-rmse:24.51246	validation_1-rmse:50.83495
[15]	validation_0-rmse:22.54053	validation_1-rmse:48.28755
[16]	validation_0-rmse:20.98230	validation_1-rmse:46.41356
[17]	validation_0-rmse:19.73797	validation_1-rmse:45.18608
[18]	validation_0-rmse:18.49679	validation_1-rmse:44.70341
```

In [14]:
```
eval_result = regressor.evals_result()
```

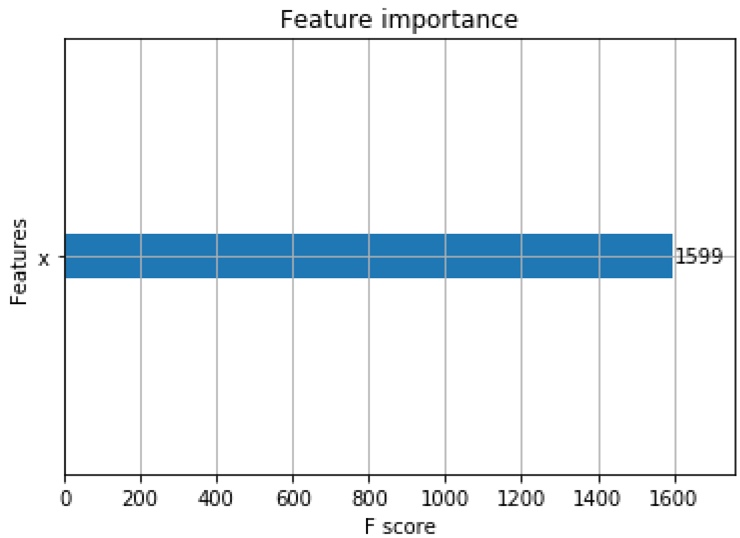
In [15]:
```
training_rounds = range(len(eval_result['validation_0']['rmse']))
```

In [16]:
```
plt.scatter(x=training_rounds,y=eval_result['validation_0']['rmse'],label='Traini
plt.scatter(x=training_rounds,y=eval_result['validation_1']['rmse'],label='Valida
plt.grid(True)
plt.xlabel('Iteration')
plt.ylabel('RMSE')
plt.title('Training Vs Validation Error')
plt.legend()
plt.show()
```



In [17]:

```
xgb.plot_importance(regressor)
plt.show()
```




# # Validation Dataset Compare Actual and Predicted

In [18]:

```
result = regressor.predict(X_validation)
```

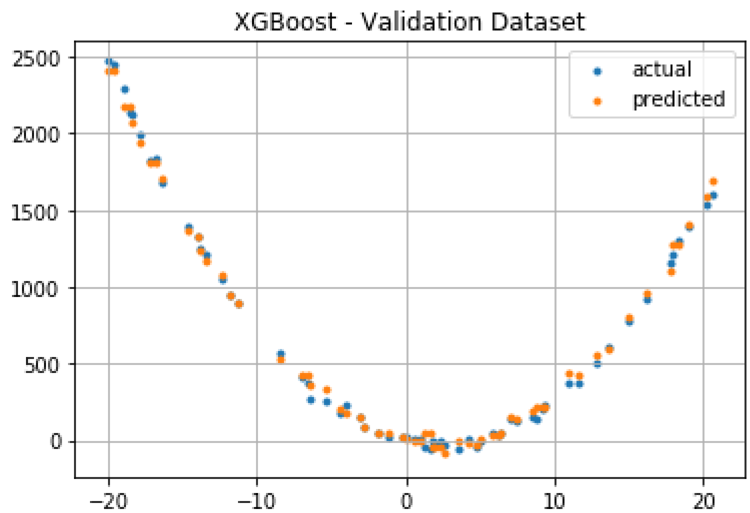
In [19]:

```
result[:5]
```

Out[19]:

```
array([1815.7225 ,   46.51924, 1815.7225 , 1400.9963 ,  156.46053],
      dtype=float32)
```

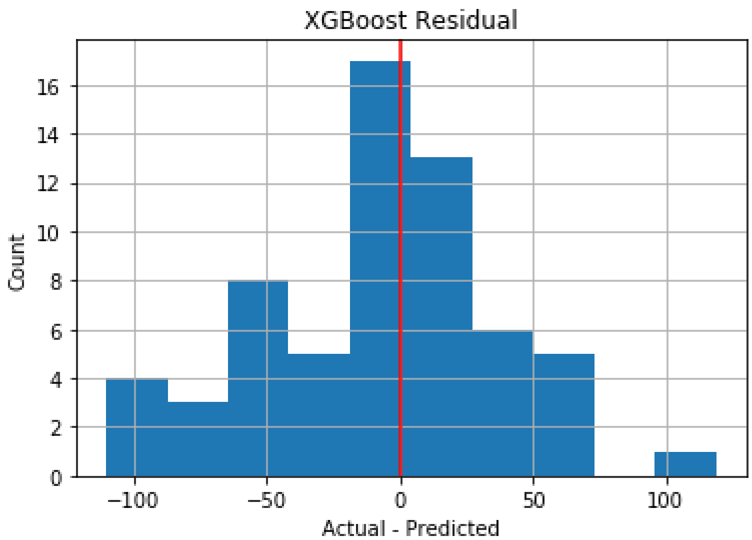
In [20]:
```
plt.title('XGBoost - Validation Dataset')
plt.scatter(df_validation.x,df_validation.y,label='actual',marker='.')
plt.scatter(df_validation.x,result,label='predicted',marker='.')
plt.grid(True)
plt.legend()
plt.show()
```




In [21]:
```
# 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: 2060.76
 Root Mean Square Error: 45.40
```

In [22]:
```
# 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()
```

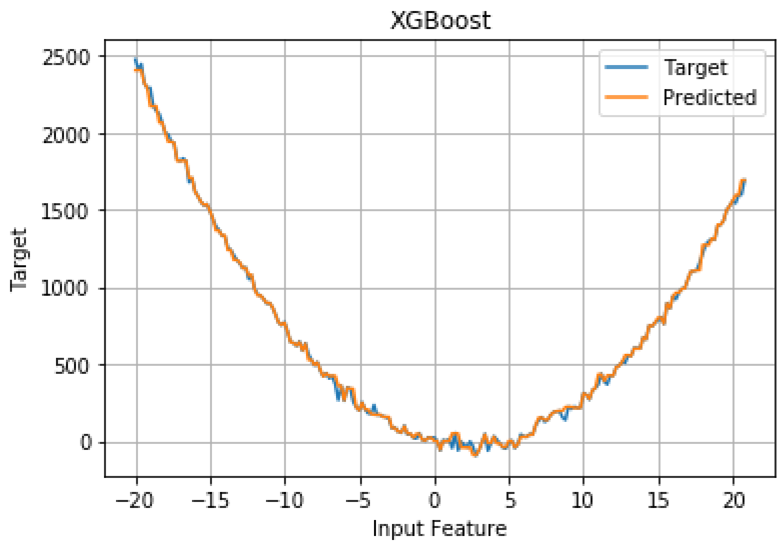



In [23]:
```
# Count number of values greater than zero and less than zero
value_counts = (residuals > 0).value_counts(sort=False)

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

```
 Under Estimation: 27
 Over  Estimation: 35
```

In [24]:
```
# 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 [25]:
```
lin_regressor = LinearRegression()
```

In [26]:
```
lin_regressor.fit(X_train,y_train)
```

Out[26]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)

In [27]:
```
lin_regressor.coef_
```

Out[27]: array([-15.07800272])

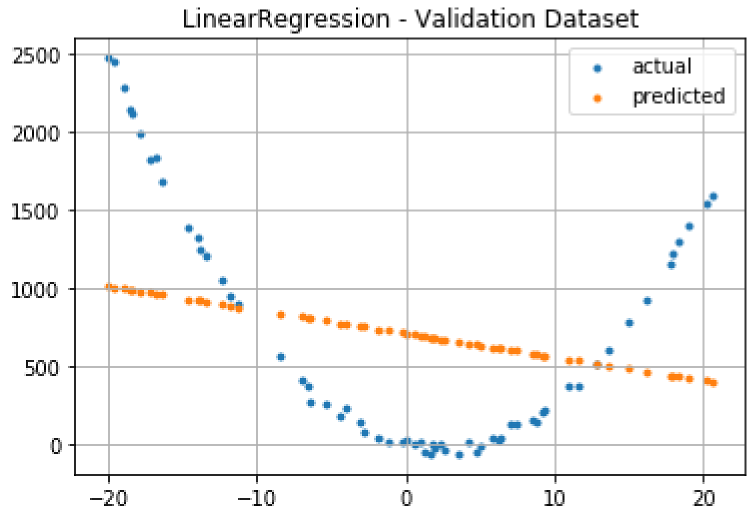
In [28]:
```

lin_regressor.intercept_
```

Out[28]: 709.8622001903116

In [29]:
```
result = lin_regressor.predict(df_validation[['x']])
```

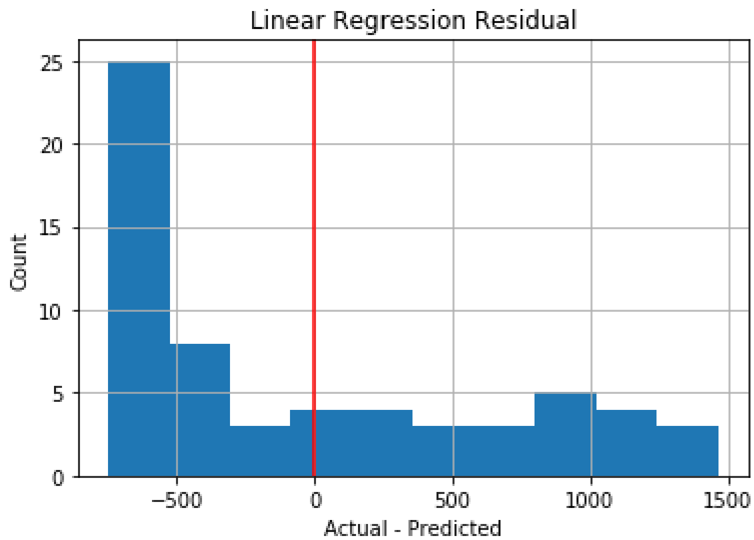
In [30]:
```
plt.title('LinearRegression - Validation Dataset')
plt.scatter(df_validation.x,df_validation.y,label='actual',marker='.')
plt.scatter(df_validation.x,result,label='predicted',marker='.')
plt.grid(True)
plt.legend()
plt.show()
```




In [31]:
```
# 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: 488269.59
 Root Mean Square Error: 698.76
```

In [32]:
```
# 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()
```

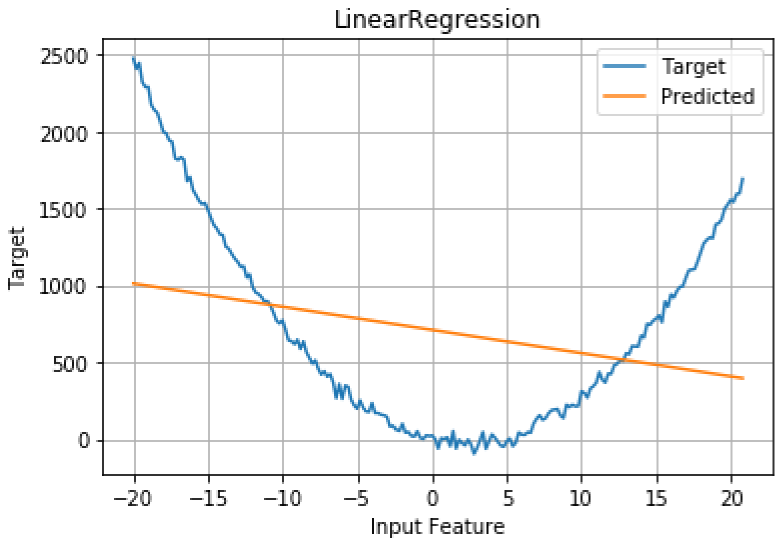



In [33]:
```
# Count number of values greater than zero and less than zero
value_counts = (residuals > 0).value_counts(sort=False)

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

```
 Under Estimation: 25
 Over  Estimation: 37
```

In [34]:
```
# 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()
```




```
Linear Regression is showing clear symptoms of under-fitting

Input Features are not sufficient to capture complex relationship
```

# Your Turn

You can correct this under-fitting issue by adding relavant features. 1.What feature will you add and why? 2.Complete the code and Test 3.What performance do you see now?

In [36]:
```
# 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'])
df = pd.read_csv(r'C:\Users\309962\Desktop\quadratic_all.csv')
```

# # Add new features

In [ ]:
```
# Place holder to add new features to df_train, df_validation and df
# if you need help, scroll down to see the answer
# Add your code
```

In [37]:

```
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 [38]:

```
lin_regressor.fit(X_train,y_train)
```

Out[38]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

In [39]:

```
lin_regressor.coef_
```

Out[39]:

```
array([-15.07800272])
```

In [40]:

```
lin_regressor.intercept_
```

Out[40]:

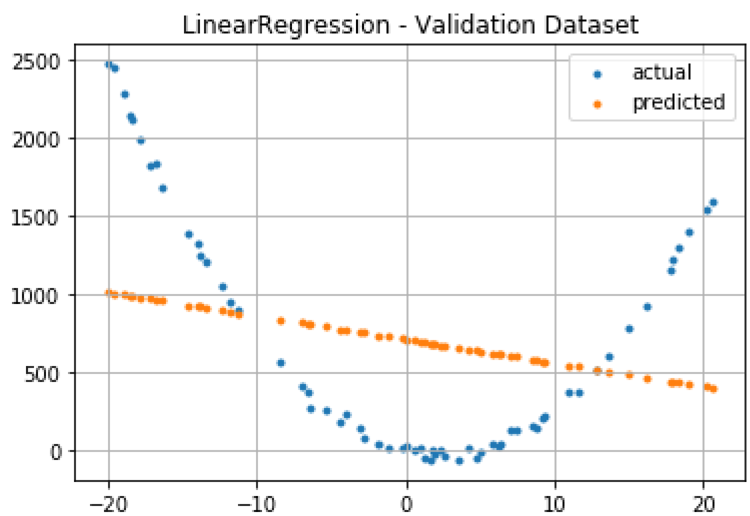
```
709.8622001903116
```

In [41]:

```
result = lin_regressor.predict(X_validation)
```

In [42]:

```
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 [43]:
```
# 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))

print("***You should see an RMSE score of 30.45 or less")
```

```
Linear Regression Metrics
 Mean Squared Error: 488269.59
 Root Mean Square Error: 698.76
***You should see an RMSE score of 30.45 or less
```

In [44]:
```
df.head()
```

Out[44]:

| | x | y |
|---|---|---|
| **0** | -20.0 | 2473.236825 |
| **1** | -19.8 | 2405.673895 |
| **2** | -19.6 | 2444.523136 |
| **3** | -19.4 | 2320.437236 |
| **4** | -19.2 | 2288.088295 |

In [45]:
```
# Plot for entire dataset
plt.plot(df.x,df.y,label='Target')
plt.plot(df.x,lin_regressor.predict(df[['x','x2']]) ,label='Predicted')
plt.grid(True)
plt.xlabel('Input Feature')
plt.ylabel('Target')
plt.legend()
plt.title('LinearRegression')
plt.show()
```

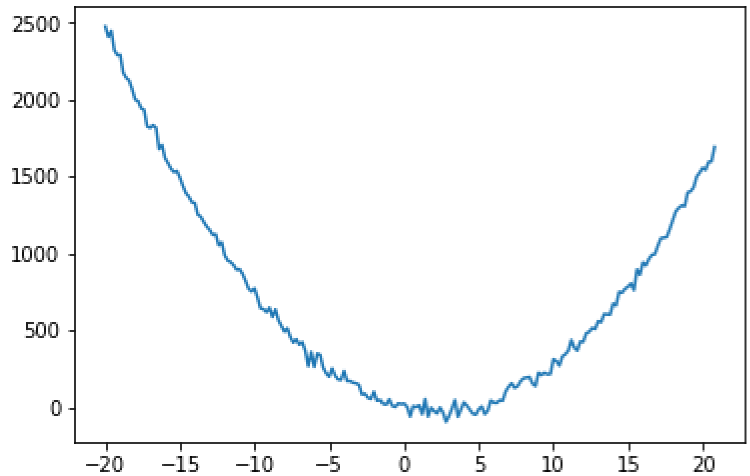
```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-45-69fac097b630> in <module>()
      1 # Plot for entire dataset
      2 plt.plot(df.x,df.y,label='Target')
----> 3 plt.plot(df.x,lin_regressor.predict(df[['x','x2']]) ,label='Predicted')
      4 plt.grid(True)
      5 plt.xlabel('Input Feature')

C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\frame.py in __getitem__
(self, key)
   2131         if isinstance(key, (Series, np.ndarray, Index, list)):
   2132             # either boolean or fancy integer index
-> 2133             return self._getitem_array(key)
   2134         elif isinstance(key, DataFrame):
   2135             return self._getitem_frame(key)

C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\frame.py in _getitem_arr
ay(self, key)
   2175             return self._take(indexer, axis=0, convert=False)
   2176         else:
-> 2177             indexer = self.loc._convert_to_indexer(key, axis=1)
   2178             return self._take(indexer, axis=1, convert=True)
   2179 

C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\indexing.py in _convert_
to_indexer(self, obj, axis, is_setter)
   1267                 if mask.any():
   1268                     raise KeyError('{mask} not in index'
-> 1269                                    .format(mask=objarr[mask]))
   1270 
   1271                 return _values_from_object(indexer)

KeyError: "['x2'] not in index"
```




In [ ]: