Part 1: Manual Implementation of SGD Regressor

CS6375 - Machine Learning

Assignment 1

Team:

Siddhant Suresh Medar - SSM200002 and Adithya Iyer - ASI200000

Dataset used:

Computer Hardware Data Set

Data Set Information:

The estimated relative performance values were estimated by the authors using a linear regression method. See their article (pp 308-313) for more details on how the relative performance values were set.

Attribute Information:

1. vendor name: 30 (adviser, amdahl,apollo, basf, bti, burroughs, c.r.d, cambex, cdc, dec, dg, formation, four-phase, gould, honeywell, hp, ibm, ipl, magnuson, microdata, nas, ncr, nixdorf, perkin-elmer, prime, siemens, sperry, sratus, wang)
2. Model Name: many unique symbols
3. MYCT: machine cycle time in nanoseconds (integer)
4. MMIN: minimum main memory in kilobytes (integer)
5. MMAX: maximum main memory in kilobytes (integer)
6. CACH: cache memory in kilobytes (integer)
7. CHMIN: minimum channels in units (integer)
8. CHMAX: maximum channels in units (integer)
9. PRP: published relative performance (integer)
10. ERP: estimated relative performance from the original article (integer)

Relevant Papers:

Ein-Dor and Feldmesser (CACM 4/87, pp 308-317)

Kibler,D. & Aha,D. (1988). Instance-Based Prediction of Real-Valued Attributes. In Proceedings of the CSCSI (Canadian AI) Conference.

Target Variable:

ERP = estimated relative performance from the original article (integer)

Import required libraries

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, cross_val_predict
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_absolute_error, mean_squared_error
from sklearn.metrics import r2_score, explained_variance_score
from sklearn.linear_model import LinearRegression, SGDRegressor
import seaborn as sns
from scipy import stats
import warnings
warnings.filterwarnings('ignore')
import sys
import io
%matplotlib inline
```

Load the dataset

In [2]:

```
columns = ['VENDOR NAME','MODEL NAME','MYCT','MMIN','MMAX','CACH','CHMIN',
           'CHMAX','PRP','ERP']
df = pd.read_csv(
    'https://archive.ics.uci.edu/ml/machine-learning-databases/cpu-performance/machine.
data'
    , names = columns)
```

Peform Exploratory Data Analysis

In [3]:

```
#peek through the data
df.head()
```

Out[3]:

	VENDOR NAME	MODEL NAME	MYCT	MMIN	MMAX	CACH	CHMIN	CHMAX	PRP	ERP
0	adviser	32/60	125	256	6000	256	16	128	198	199
1	amdahl	470v/7	29	8000	32000	32	8	32	269	253
2	amdahl	470v/7a	29	8000	32000	32	8	32	220	253
3	amdahl	470v/7b	29	8000	32000	32	8	32	172	253
4	amdahl	470v/7c	29	8000	16000	32	8	16	132	132

Pre-processing the data

In [4]:

```
#check for NULL values
print (df.isnull().sum())
```

```
VENDOR NAME    0
MODEL NAME     0
MYCT           0
MMIN           0
MMAX           0
CACH           0
CHMIN          0
CHMAX          0
PRP            0
ERP            0
dtype: int64
```

In [5]:

```
#remove missing values - no missing values were found
df.dropna( inplace = True )
```

In [6]:

```
#drop duplicate items - no duplicate items were found
df.drop_duplicates()
```

Out[6]:

	VENDOR NAME	MODEL NAME	MYCT	MMIN	MMAX	CACH	CHMIN	CHMAX	PRP	ERP
0	adviser	32/60	125	256	6000	256	16	128	198	199
1	amdahl	470v/7	29	8000	32000	32	8	32	269	253
2	amdahl	470v/7a	29	8000	32000	32	8	32	220	253
3	amdahl	470v/7b	29	8000	32000	32	8	32	172	253
4	amdahl	470v/7c	29	8000	16000	32	8	16	132	132
...	...	...	...	...	...	...	...	...	...	...
204	sperry	80/8	124	1000	8000	0	1	8	42	37
205	sperry	90/80-model-3	98	1000	8000	32	2	8	46	50
206	sratus	32	125	2000	8000	0	2	14	52	41
207	wang	vs-100	480	512	8000	32	0	0	67	47
208	wang	vs-90	480	1000	4000	0	0	0	45	25

209 rows × 10 columns

In [7]:

```
#information about the data
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 209 entries, 0 to 208
Data columns (total 10 columns):
 #   Column       Non-Null Count  Dtype 
---  ------       --------------  ----- 
 0   VENDOR NAME  209 non-null    object
 1   MODEL NAME   209 non-null    object
 2   MYCT         209 non-null    int64 
 3   MMIN         209 non-null    int64 
 4   MMAX         209 non-null    int64 
 5   CACH         209 non-null    int64 
 6   CHMIN        209 non-null    int64 
 7   CHMAX        209 non-null    int64 
 8   PRP          209 non-null    int64 
 9   ERP          209 non-null    int64 
dtypes: int64(8), object(2)
memory usage: 18.0+ KB
```

In [8]:

```
#explore data
df.describe()
```

Out[8]:

	MYCT	MMIN	MMAX	CACH	CHMIN	CHMAX	
count	209.000000	209.000000	209.000000	209.000000	209.000000	209.000000	209.00
mean	203.822967	2867.980861	11796.153110	25.205742	4.698565	18.267943	105.62
std	260.262926	3878.742758	11726.564377	40.628722	6.816274	25.997318	160.83
min	17.000000	64.000000	64.000000	0.000000	0.000000	0.000000	6.00
25%	50.000000	768.000000	4000.000000	0.000000	1.000000	5.000000	27.00
50%	110.000000	2000.000000	8000.000000	8.000000	2.000000	8.000000	50.00
75%	225.000000	4000.000000	16000.000000	32.000000	6.000000	24.000000	113.00
max	1500.000000	32000.000000	64000.000000	256.000000	52.000000	176.000000	1150.00

In [9]:

```
#explore target variable
df["ERP"].describe()
```

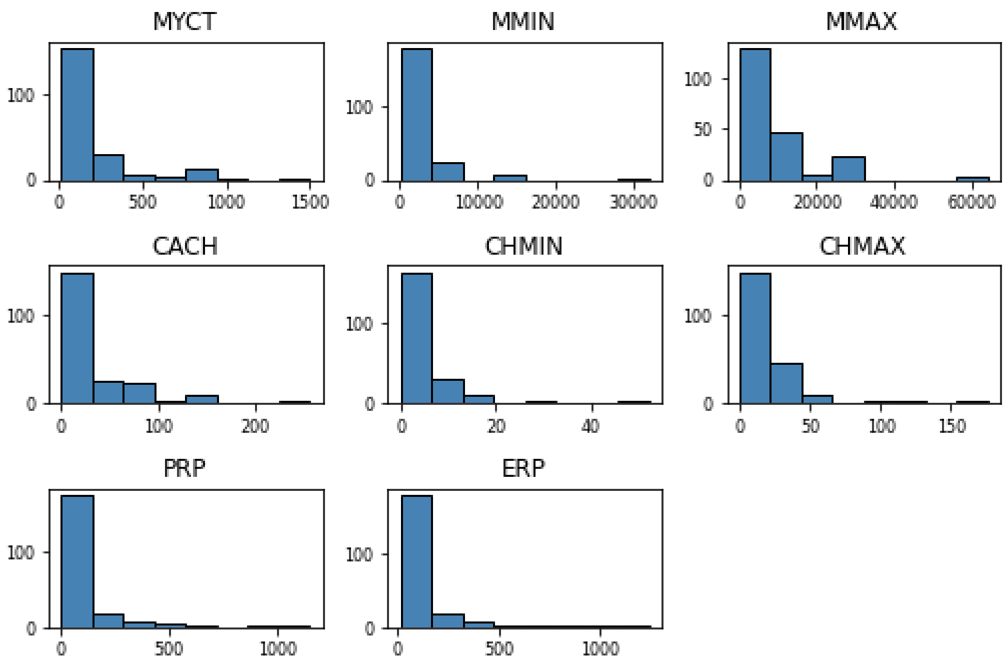
Out[9]:

```
count     209.000000
mean       99.330144
std       154.757102
min        15.000000
25%        28.000000
50%        45.000000
75%       101.000000
max      1238.000000
Name: ERP, dtype: float64
```

Feature Engineering

In [10]:

```
#histogram plots for each column
df.hist(bins=8, color='steelblue', edgecolor='black', linewidth=1.0, xlabelsize=8, ylab
elsize=8, grid=False)
plt.tight_layout(rect=(0, 0, 1.2, 1.2))
```



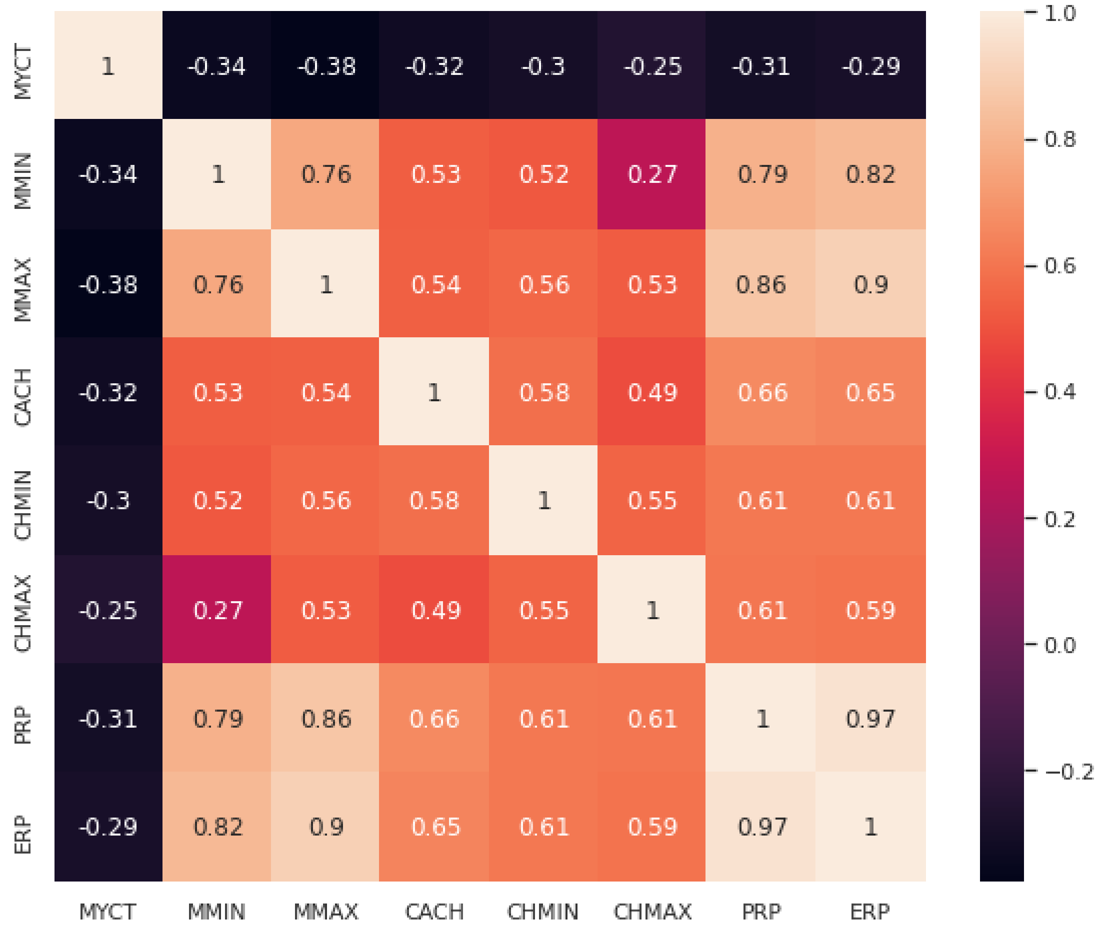
In [11]:

```
#calculate correlation between the attributes
correlations = df.corr()
print(correlations)
```

```
           MYCT      MMIN      MMAX  ...     CHMAX       PRP       ERP
MYCT   1.000000 -0.335642 -0.378561  ... -0.250502 -0.307099 -0.288396
MMIN  -0.335642  1.000000  0.758157  ...  0.266907  0.794931  0.819292
MMAX  -0.378561  0.758157  1.000000  ...  0.527246  0.863004  0.901202
CACH  -0.321000  0.534729  0.537990  ...  0.487846  0.662641  0.648620
CHMIN -0.301090  0.517189  0.560513  ...  0.548281  0.608903  0.610580
CHMAX -0.250502  0.266907  0.527246  ...  1.000000  0.605209  0.592156
PRP   -0.307099  0.794931  0.863004  ...  0.605209  1.000000  0.966472
ERP   -0.288396  0.819292  0.901202  ...  0.592156  0.966472  1.000000

[8 rows x 8 columns]
```

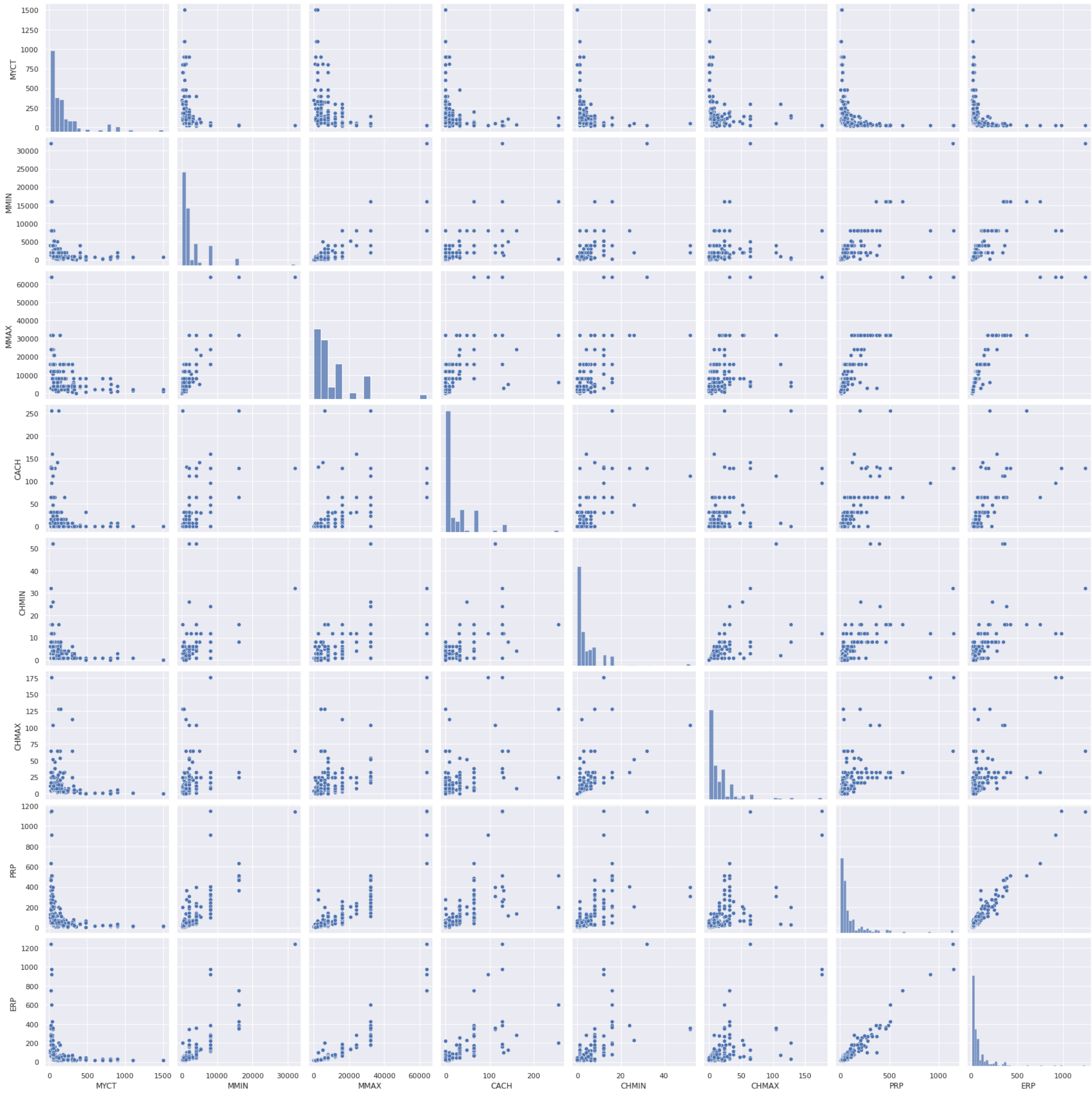
In [12]:

```
#plot heatmap to visualize the correlation
sns.set(rc = {'figure.figsize':(10,8)})
sns.heatmap(correlations, annot=True,square=True)
plt.show()
```



In [13]:

```
#pairplot to check the pairwise relationships in the dataset
sns.set()
sns.pairplot(df, size=3)
plt.show()
```



Conclusion: We observed during exploratory analysis that the first two attributes - Vendor name and Model Name does not contribute to the end result.

Decision : Drop two attributes - Vendor name and Model Name

PART 1 - Impelementing SGD regressor manually

Splitting the data into independent and dependent variables - X and y

In [14]:

```
data = df.iloc[:,2:]

X = data.iloc[:,:-1]
y = data.iloc[:,-1]
```

In [15]:

```
# Splitting the data into training and testing samples
#Using 80/20 split for training and testing respectively
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size = 0.20, random_state = 42, shuffle = True)
```

In [16]:

```
#shape of X_Train and X_Test
X_train.shape, X_test.shape
```

Out[16]:

```
((167, 7), (42, 7))
```

In [17]:

```
#shape of Y_Train and Y_Test
y_train.shape, y_test.shape
```

Out[17]:

```
((167,), (42,))
```

In [18]:

```
#standardize features by removing the mean and scaling to unit variance
sc = StandardScaler()
sc.fit(X_train)
```

Out[18]:

```
StandardScaler()
```

In [19]:

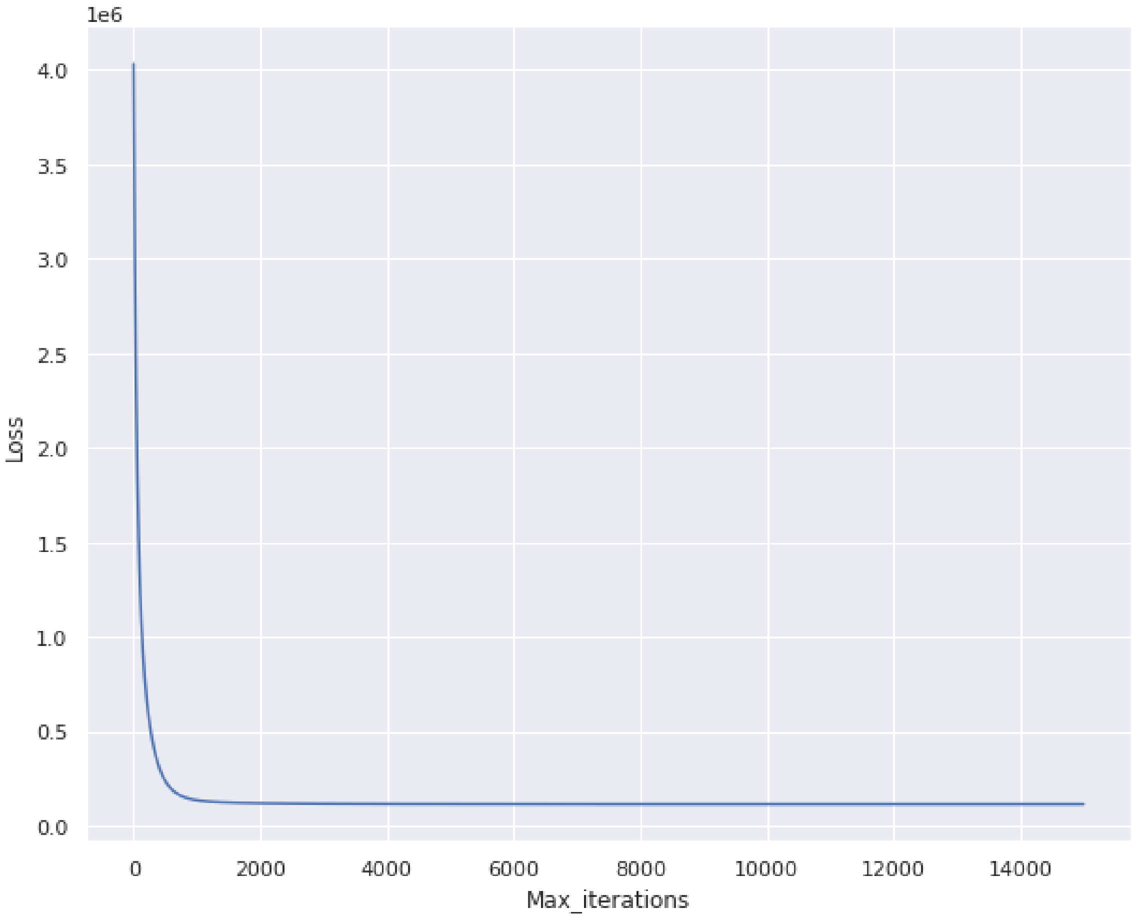
```
#apply the calculations performed earlier in fit()
#to every data point in feature using transform()
X_train_sc = sc.transform(X_train)
X_test_sc = sc.transform(X_test)
```

In [20]:

```
class ManualSGD:
    def __init__(self,learning_rate=0.001,max_iterations=500,threshold=None):
        #constructor
        self.learning_rate = learning_rate
        self.max_iterations = max_iterations
        self.threshold = threshold

    def predict(self,X):
        #function to predict the values using newly created model
        X=np.insert(X.T,0,np.ones(X.shape[0]),axis=0)
        return np.dot(self.weights,X)

    def Rsquared(self,X,Y):
        #function to calculate r2 score
        return 1-(((Y - self.predict(X))**2).sum()/((Y - Y.mean())**2).sum())

    def loss_function(self,x,y,category='mse'):
        if category == 'mse':
            loss=np.sum(np.square(x.reshape(-1, 1) - y.reshape(-1, 1)))
            /(2*x.shape[0])
        return np.round(loss,3)

    def fit(self,X,y):
        #function to fit the data on the model
        self.losses=[] #list to track the losses
        self.X=X
        self.y=y
        #initialize weights and biases
        self.weights = np.random.rand(self.X.shape[1]+1).reshape(1,-1)
        #pad with ones for bias
        self.feature_vector = np.insert(self.X.T, 0,
                                        np.ones(self.X.shape[0]), axis=0)
        dw=0

        while self.max_iterations>=0:
            self.hyp = np.dot(self.weights,self.feature_vector)
            self.losses.append(self.loss_function(self.hyp,y))
            # @ is matrix multiplication
            dw = (self.feature_vector@(self.hyp-self.y).T)
            dw /= self.X.shape[0]        #average it
            self.weights -= (self.learning_rate*dw.reshape(1,-1))
            #update weights
            self.max_iterations -= 1 #decrement iterations count by 1
```

In [21]:

```
#Optimum LR and Itrs
lr, itrs = 0.003, 15000

#create object of class
model = ManualSGD(learning_rate=lr,max_iterations=itrs)

#fit the training data on the model
model.fit(X_train_sc,np.array(y_train))

#predict
y_pred=model.predict(X_test_sc)

loss=list(model.losses)

#visualize loss
plt.plot(loss)
plt.xlabel("Max_iterations")
plt.ylabel("Loss")
plt.show()
```



In [22]:

```
r2 = model.Rsquared(X_train_sc,np.array(y_train))
mae = mean_absolute_error(y_test, y_pred[0])
rmse = mean_squared_error(y_test, y_pred[0], squared=False)
evs = explained_variance_score(y_test, y_pred[0])
```

In [23]:

```
weights = list(model.weights)
print("Weights: ",weights)
```

```
Weights:  [array([92.22754491,  8.61135461, 12.15708748, 44.29981758, 13.3
9012229,
       -1.37289294,  5.42281678, 66.24193932])]
```

In [24]:

```
print()
print("For LR: "+str(lr)+", Iterations= "+str(itrs))
print("======================================")
print("R2 Score: ", r2)
print("Mean absolute error: ", mae)
print("Root Mean squared error: ", rmse)
print("Explained Variance Score: ", evs)
```

```
For LR: 0.003, Iterations= 15000
======================================
R2 Score:  0.9571726139921094
Mean absolute error:  25.526649239429617
Root Mean squared error:  54.87979243324654
Explained Variance Score:  0.9463933128634727
```

In [25]:

```
file = open("Manual_SGD_log.txt","a")
file.write("LR = " + str(lr) + ", max_iterations = " + str(itrs) +
           ", R^2 = "+str(r2) + ", MAE = "+str(mae) + ", RMSE = " + str(rmse) +
           ", Explained-Variance = " + str(evs) + " \n")
file.close()
print("Wrote to file sucessfully.")
```

```
Wrote to file sucessfully.
```

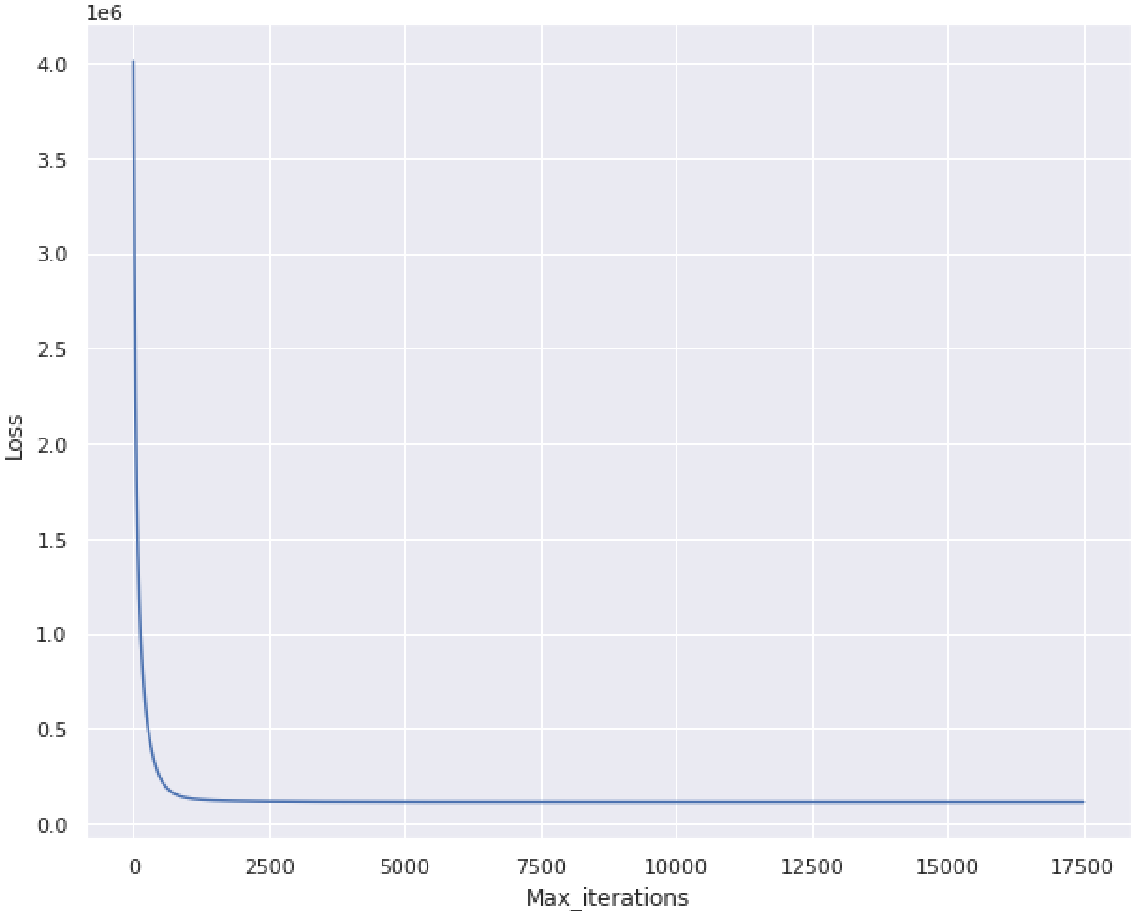
Observation:

For a learning rate = 0.003 and n_iterations = 15000, we get a r2 score of 95.71%

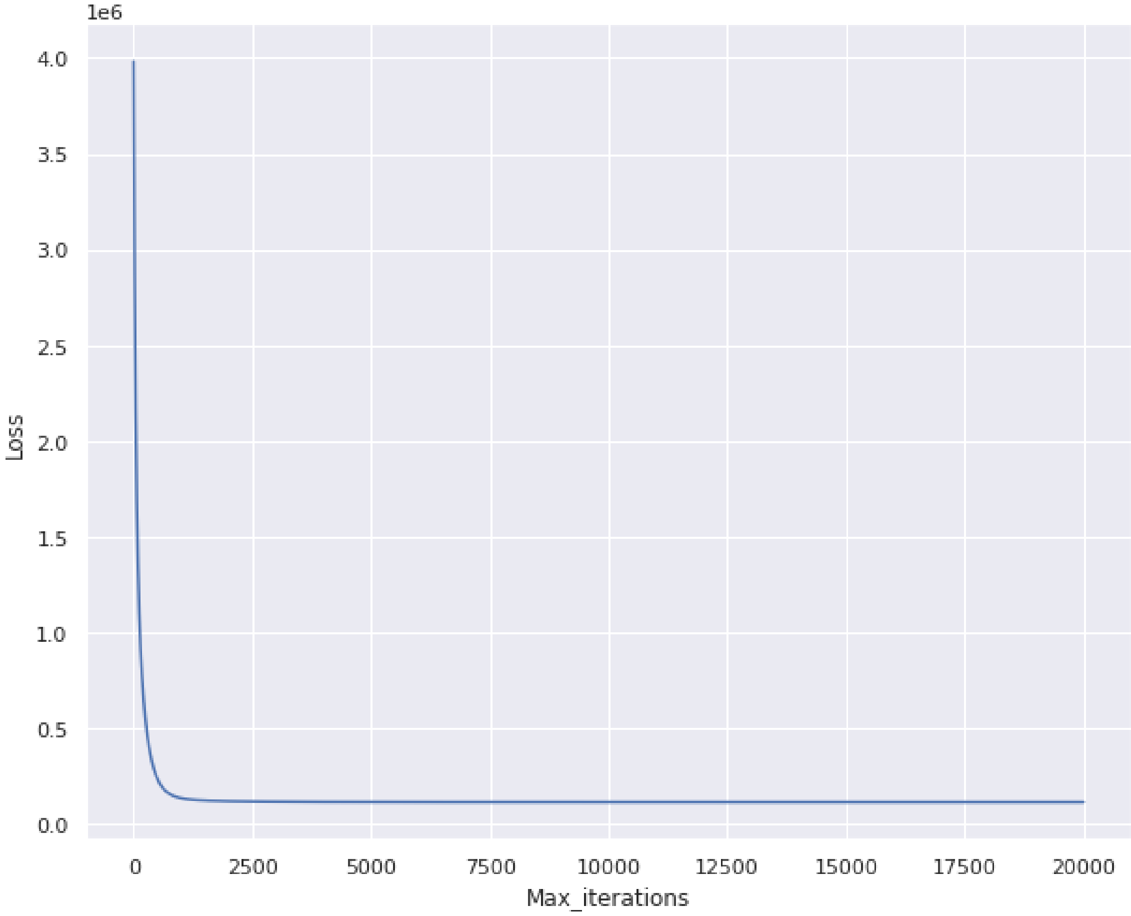
Now increasing iterations to check if it increases the accuracy further

In [26]:

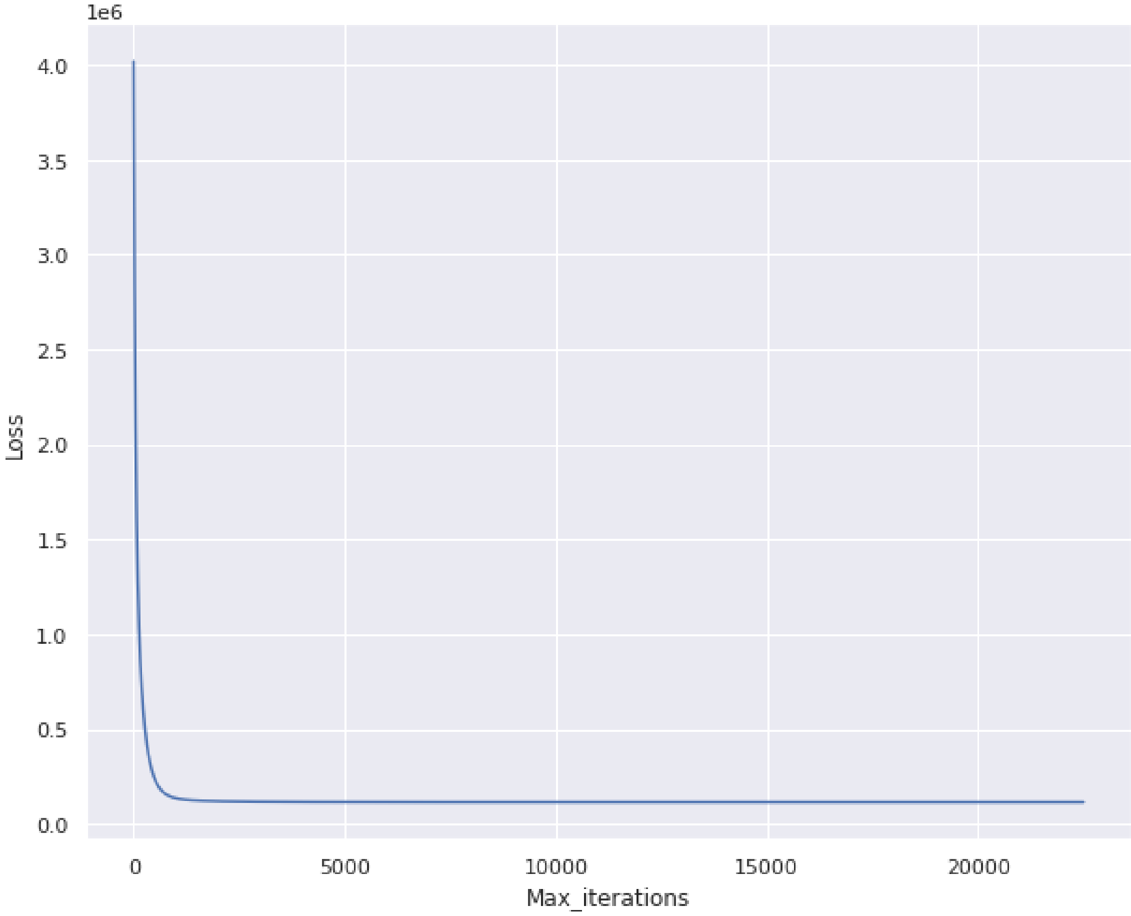
```
lr,r2_lst = 0.003,[]

itrs = 17500
while itrs<=50000:
    model = ManualSGD(learning_rate=lr,max_iterations=itrs)

    #fit the training data on the model
    model.fit(X_train_sc,np.array(y_train))

    #predict
    y_pred=model.predict(X_test_sc)

    loss=list(model.losses)

    #visualize loss
    plt.plot(loss)
    plt.xlabel("Max_iterations")
    plt.ylabel("Loss")
    plt.show()

    r2 = model.Rsquared(X_train_sc,np.array(y_train))
    mae = mean_absolute_error(y_test, y_pred[0])
    rmse = mean_squared_error(y_test, y_pred[0], squared=False)
    evs = explained_variance_score(y_test, y_pred[0])

    print("\nFor LR: "+str(lr)+", Iterations= "+str(itrs))
    print("======================================")
    print("R2 Score: ", r2)
    print("Mean absolute error: ", mae)
    print("Root Mean squared error: ", rmse)
    print("Explained Variance Score: ", evs)

    file = open("Manual_SGD_log.txt","a")
    file.write("LR = " + str(lr) + ", max_iterations = " + str(itrs) +
               ", R^2 = " + str(r2) + ", MAE = " + str(mae) + ", RMSE = " +
               str(rmse) + ", Explained-Variance = " + str(evs) + " \n")
    file.close()
    print("Wrote to file sucessfully.")

    r2_lst.append(np.around(r2,7))
    itrs+=2500
```



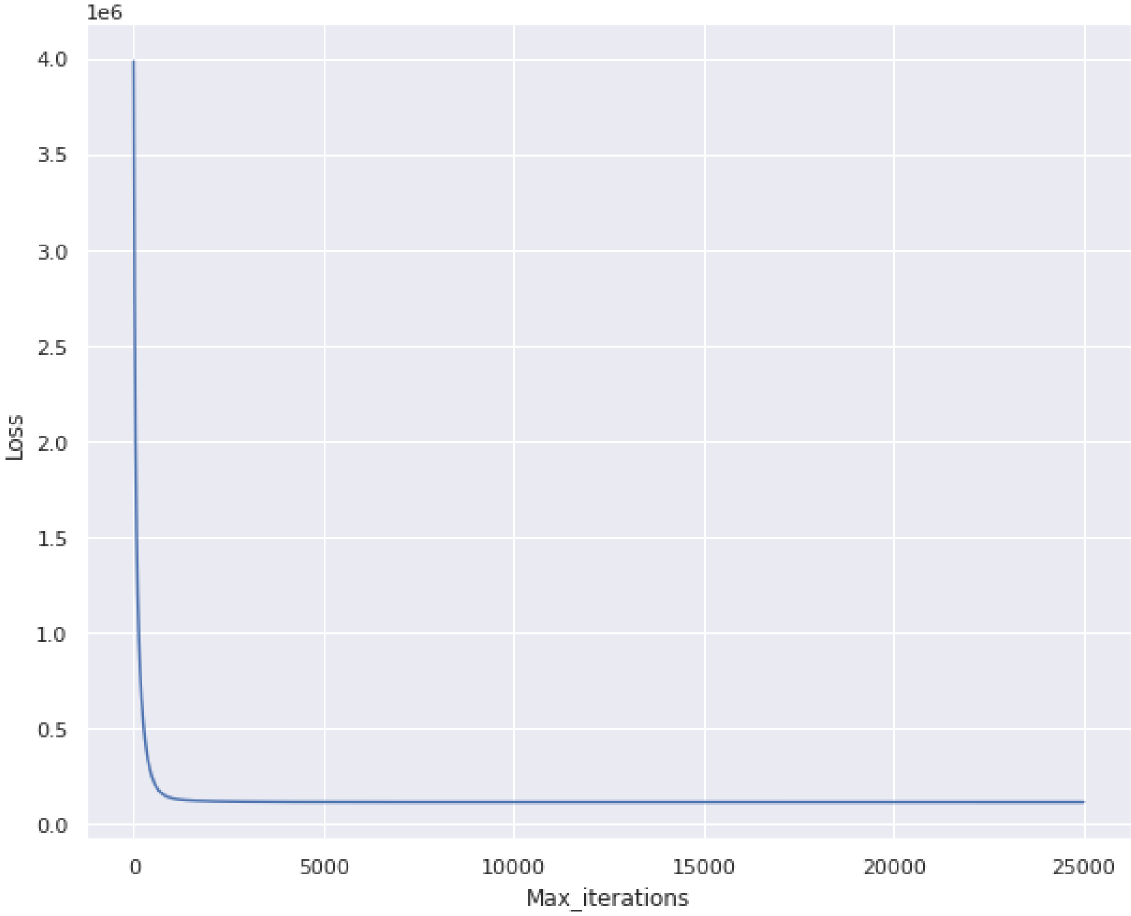

```
For LR: 0.003, Iterations= 17500
======================================
R2 Score:  0.9571727799167745
Mean absolute error:  25.52037231231766
Root Mean squared error:  54.85932675325127
Explained Variance Score:  0.9464330519974609
Wrote to file sucessfully.
```



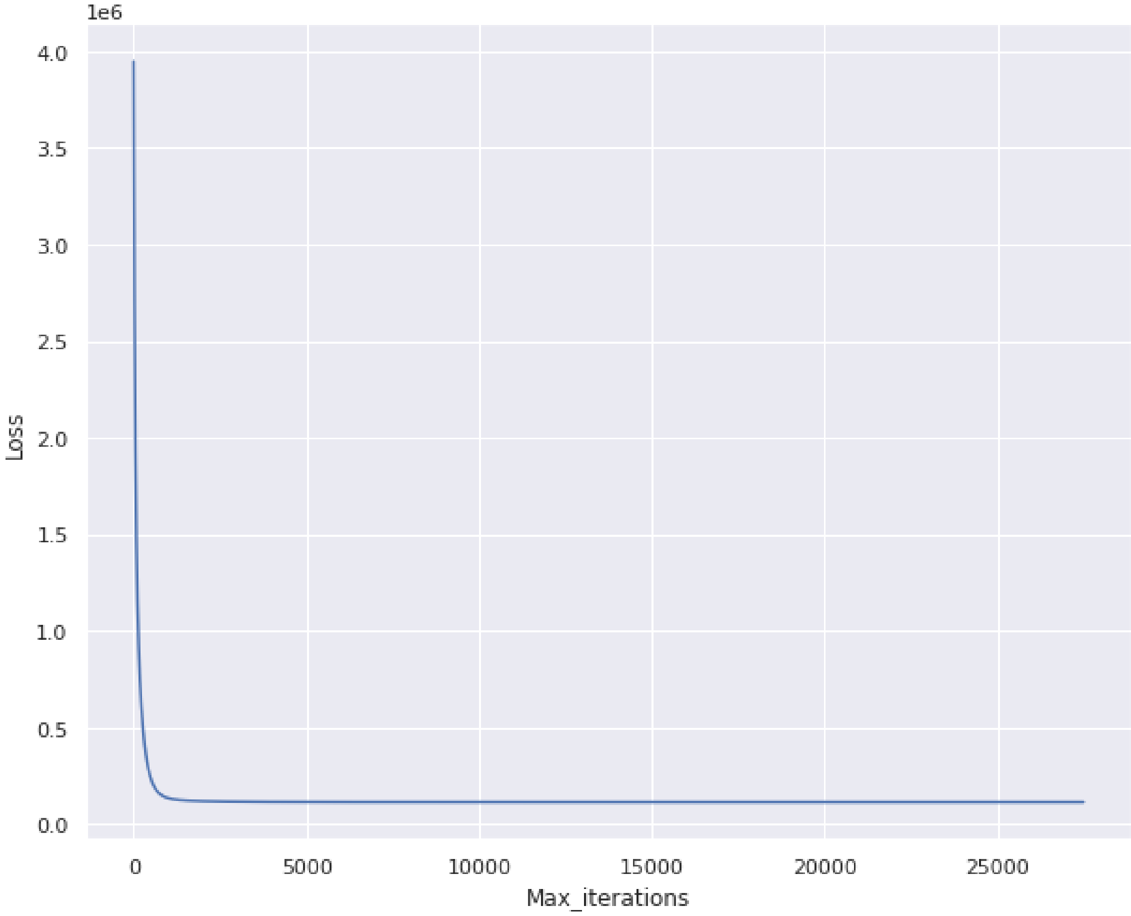

```
For LR: 0.003, Iterations= 20000
======================================
R2 Score:  0.957172808760194
Mean absolute error:  25.51773619243657
Root Mean squared error:  54.850767011135126
Explained Variance Score:  0.9464496877979258
Wrote to file sucessfully.
```



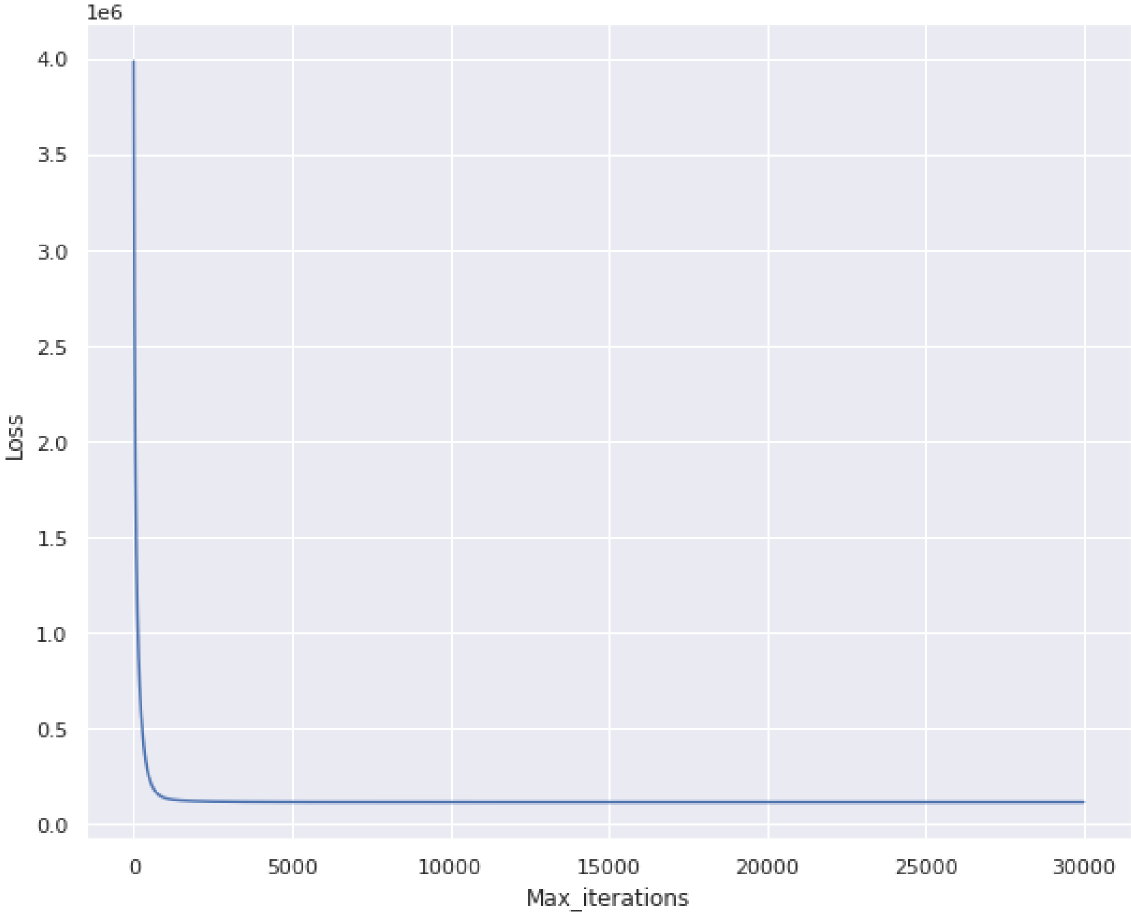

```
For LR: 0.003, Iterations= 22500
======================================
R2 Score:  0.957172814091838
Mean absolute error:  25.51658697346695
Root Mean squared error:  54.84704422786904
Explained Variance Score:  0.9464569275093573
Wrote to file sucessfully.
```



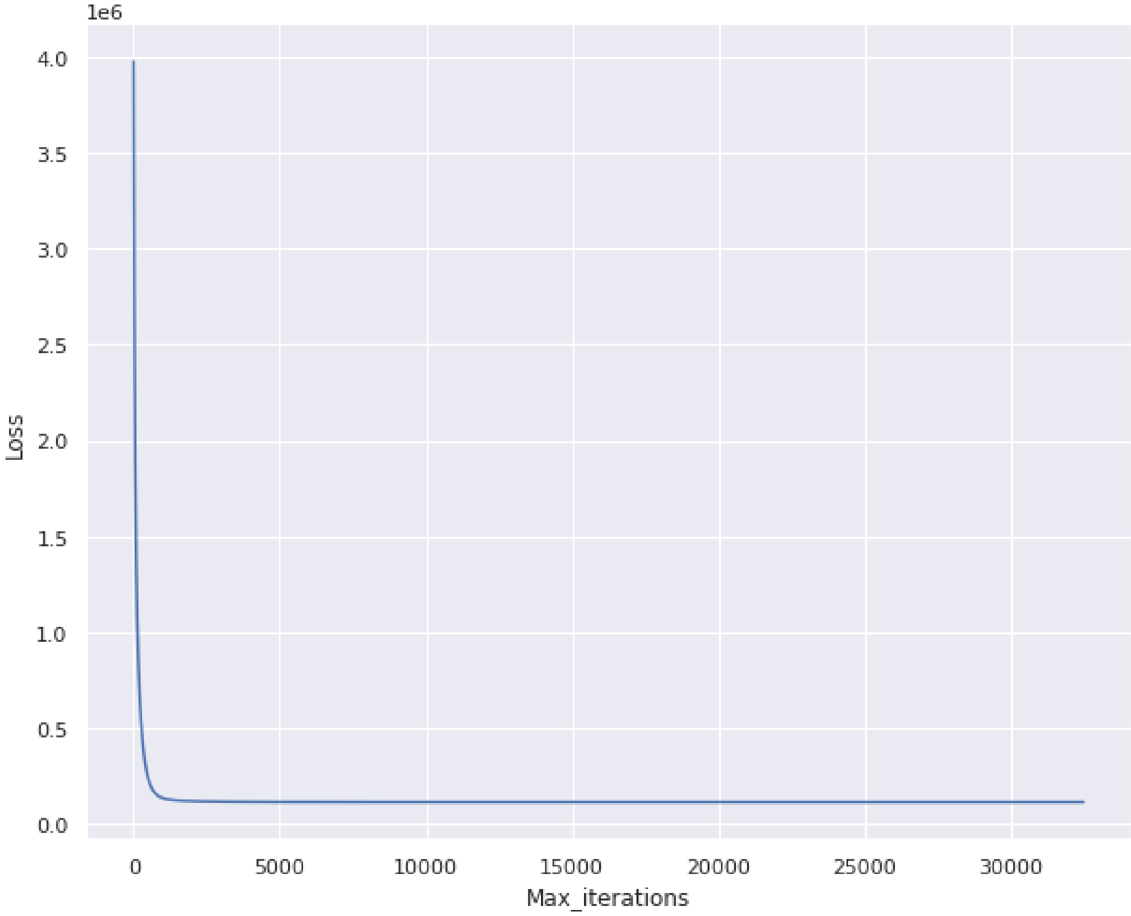

```
For LR: 0.003, Iterations= 25000
======================================
R2 Score:  0.9571728150608146
Mean absolute error:  25.51609973310397
Root Mean squared error:  54.845467757556975
Explained Variance Score:  0.9464599943109608
Wrote to file sucessfully.
```



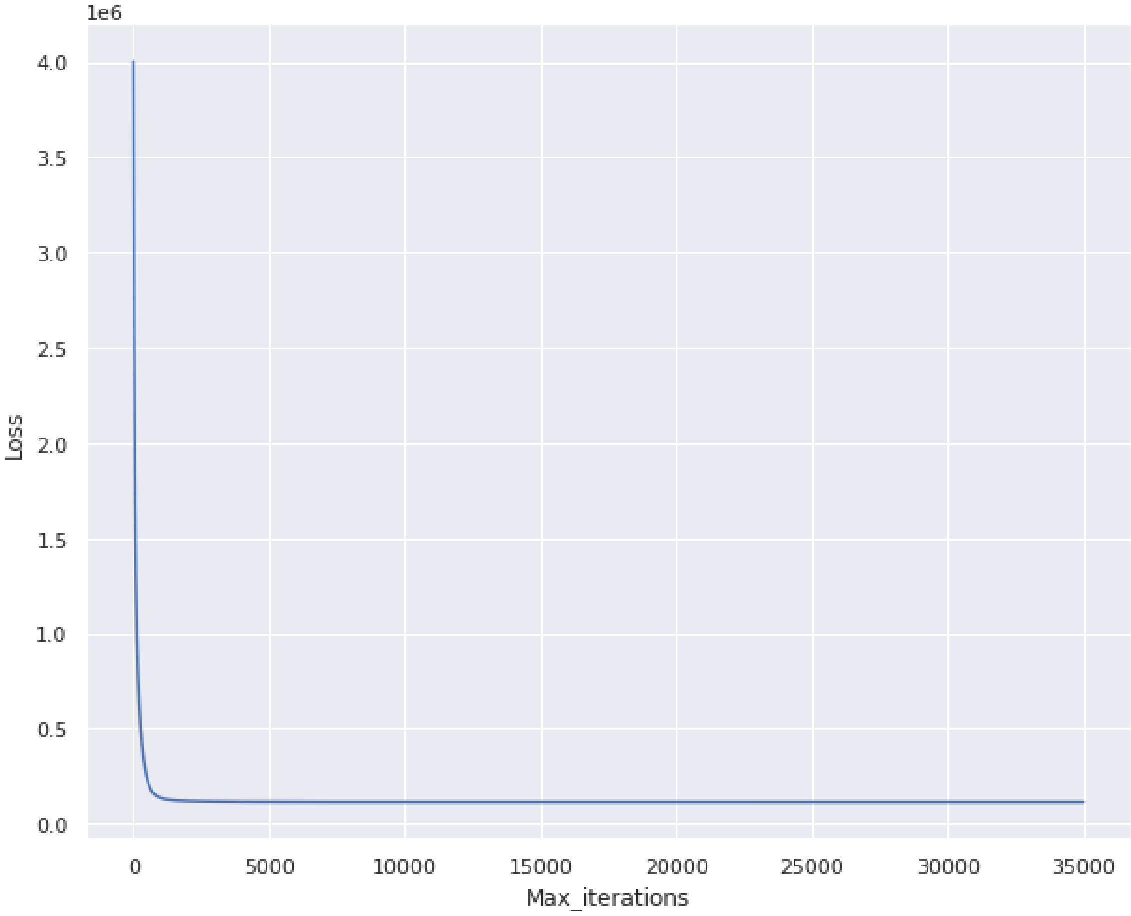

```
For LR: 0.003, Iterations= 27500
======================================
R2 Score:  0.9571728152486338
Mean absolute error:  25.5158821444334
Root Mean squared error:  54.84476423389084
Explained Variance Score:  0.9464613632050234
Wrote to file sucessfully.
```



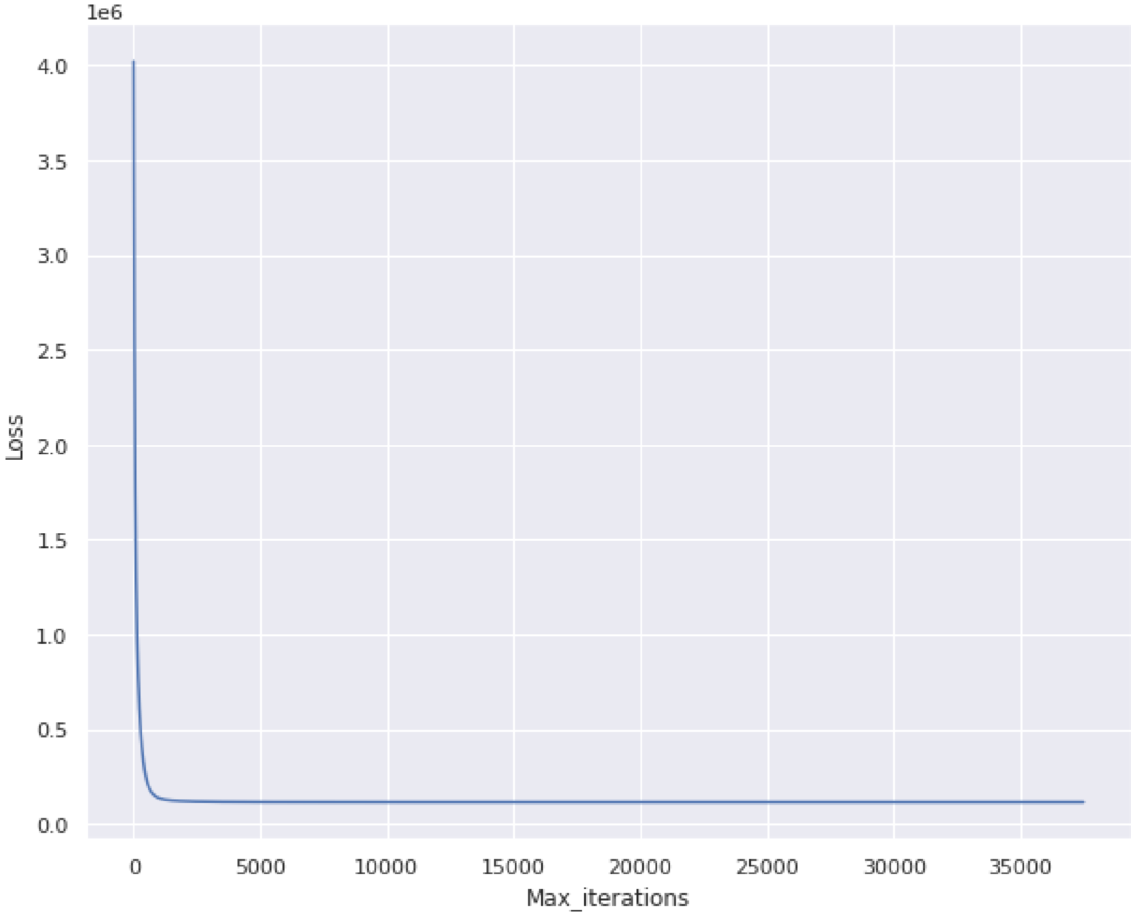

```
For LR: 0.003, Iterations= 30000
======================================
R2 Score:  0.9571728152832856
Mean absolute error:  25.51578865526772
Root Mean squared error:  54.84446207818008
Explained Variance Score:  0.9464619512050638
Wrote to file sucessfully.
```



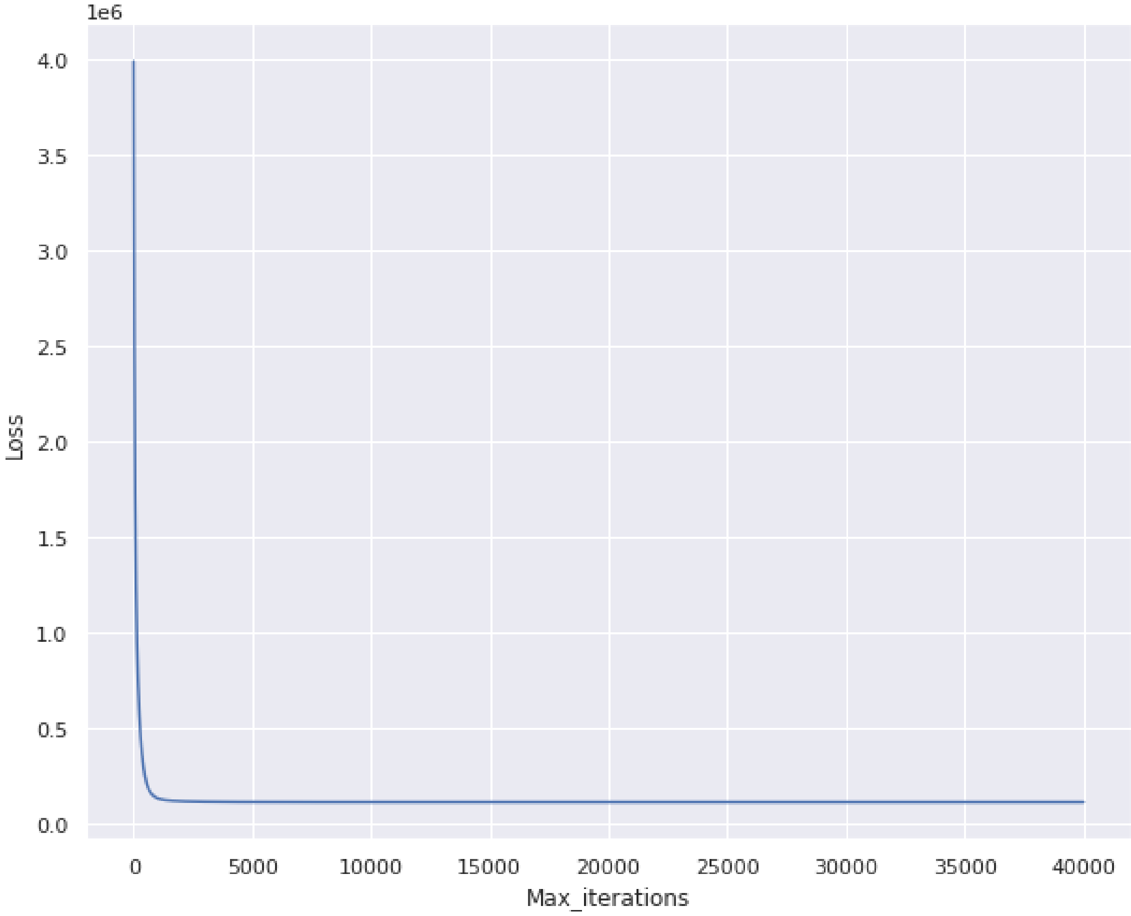

```
For LR: 0.003, Iterations= 32500
======================================
R2 Score:  0.9571728152897906
Mean absolute error:  25.515747715431345
Root Mean squared error:  54.84432978683186
Explained Variance Score:  0.9464622086625277
Wrote to file sucessfully.
```



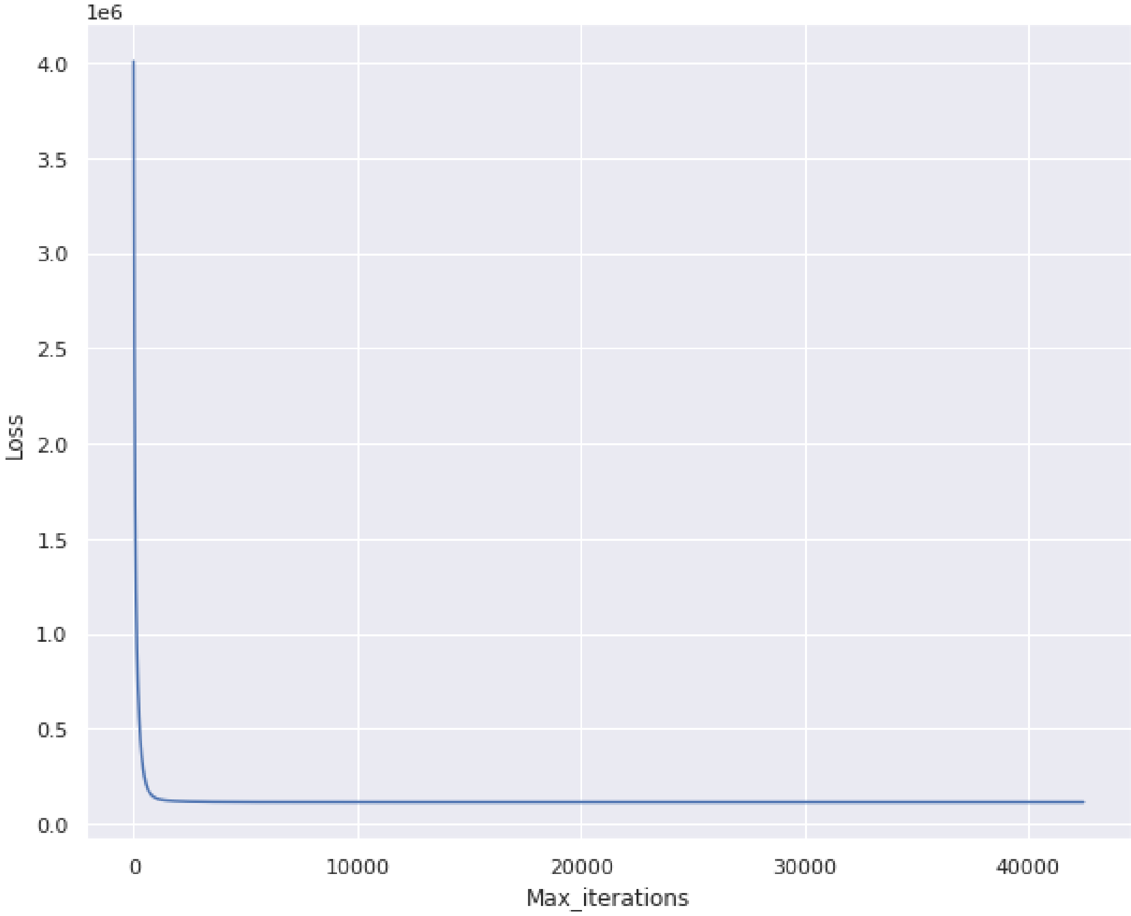

```
For LR: 0.003, Iterations= 35000
======================================
R2 Score:  0.9571728152909479
Mean absolute error:  25.515730621148805
Root Mean squared error:  54.844274554797856
Explained Variance Score:  0.9464623161555452
Wrote to file sucessfully.
```



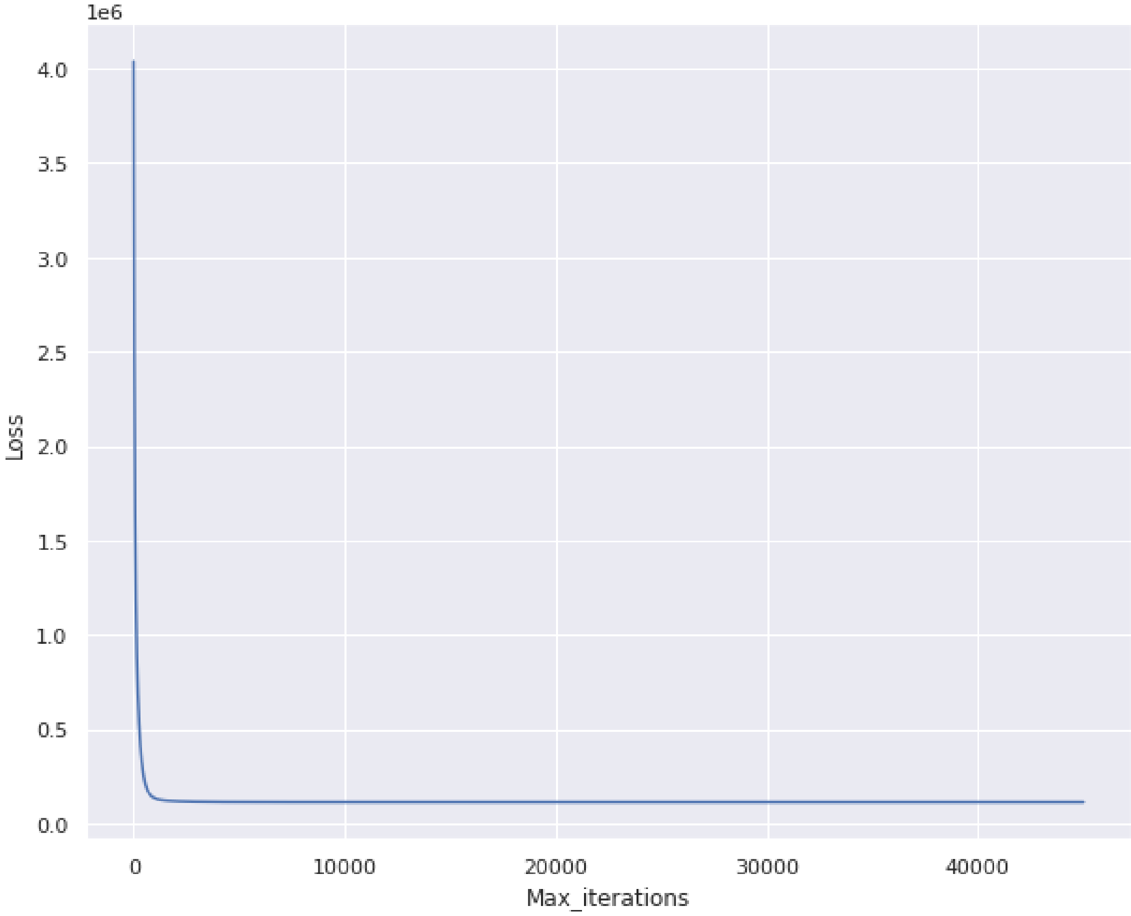

```
For LR: 0.003, Iterations= 37500
======================================
R2 Score:  0.9571728152911645
Mean absolute error:  25.515723221121945
Root Mean squared error:  54.84425064668902
Explained Variance Score:  0.9464623626867467
Wrote to file sucessfully.
```



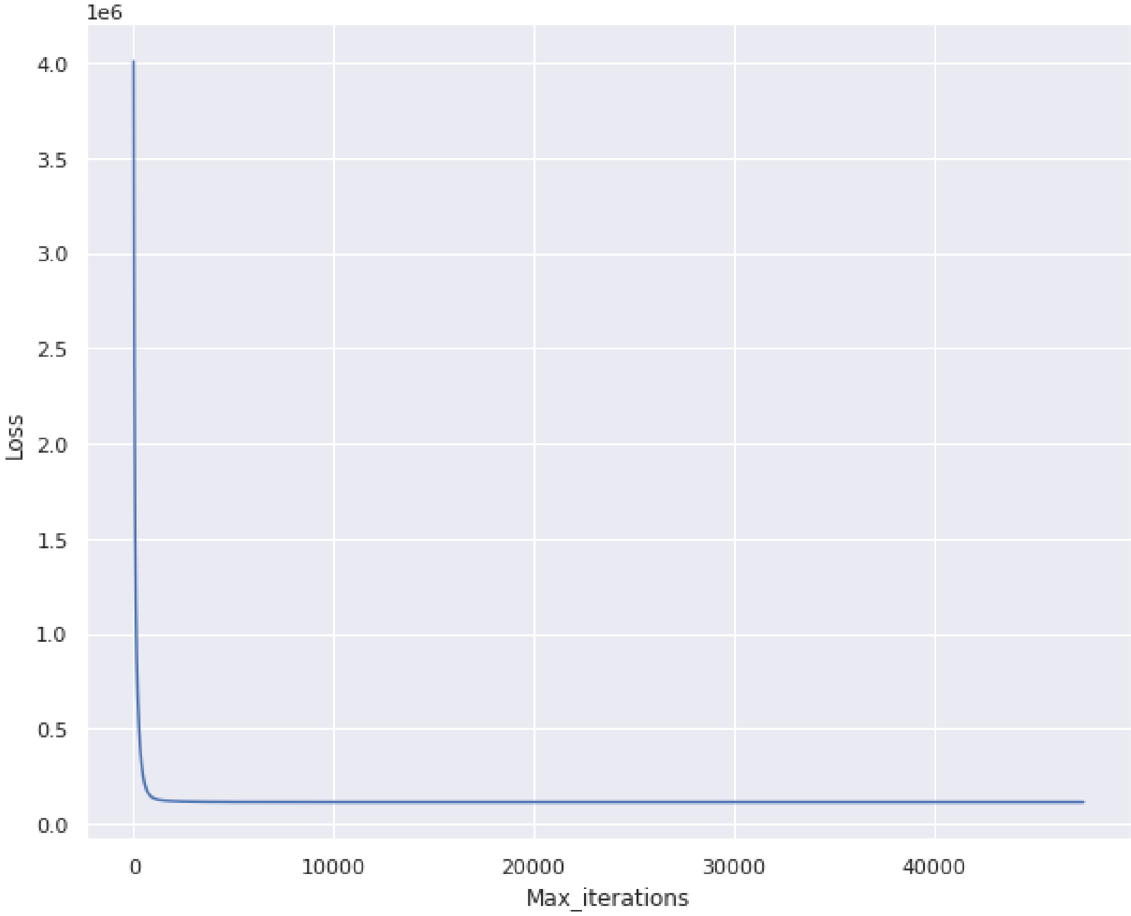

```
For LR: 0.003, Iterations= 40000
======================================
R2 Score:  0.9571728152912051
Mean absolute error:  25.51571999010045
Root Mean squared error:  54.844240208227276
Explained Variance Score:  0.9464623830028744
Wrote to file sucessfully.
```



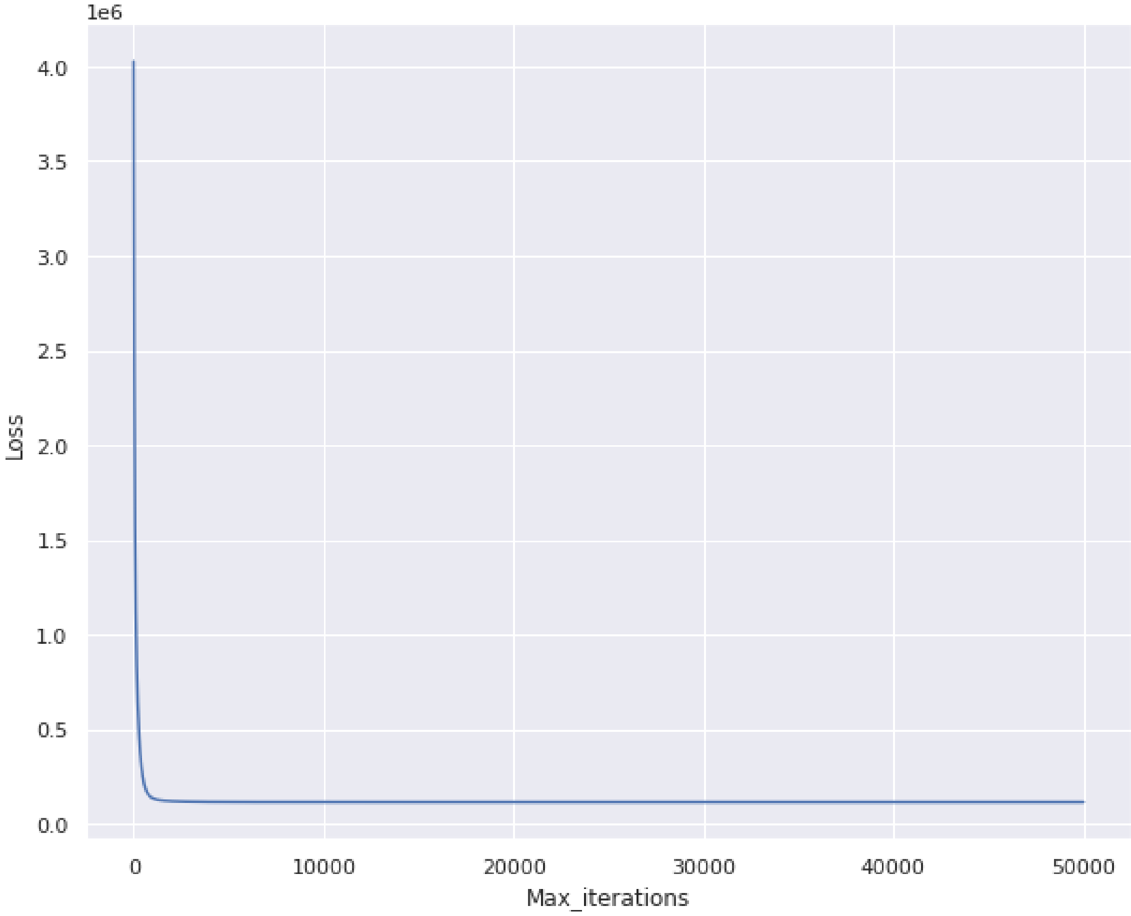

```
For LR: 0.003, Iterations= 42500
====================================
R2 Score:  0.9571728152912126
Mean absolute error:  25.5157186218835
Root Mean squared error:  54.84423578801216
Explained Variance Score:  0.9464623916058913
Wrote to file sucessfully.
```




```
For LR: 0.003, Iterations= 45000
======================================
R2 Score:  0.9571728152912139
Mean absolute error:  25.51571804087901
Root Mean squared error:  54.844233911015074
Explained Variance Score:  0.9464623952590836
Wrote to file sucessfully.
```




```
For LR: 0.003, Iterations= 47500
======================================
R2 Score:  0.9571728152912141
Mean absolute error:  25.51571777851117
Root Mean squared error:  54.84423306341335
Explained Variance Score:  0.9464623969087715
Wrote to file sucessfully.
```



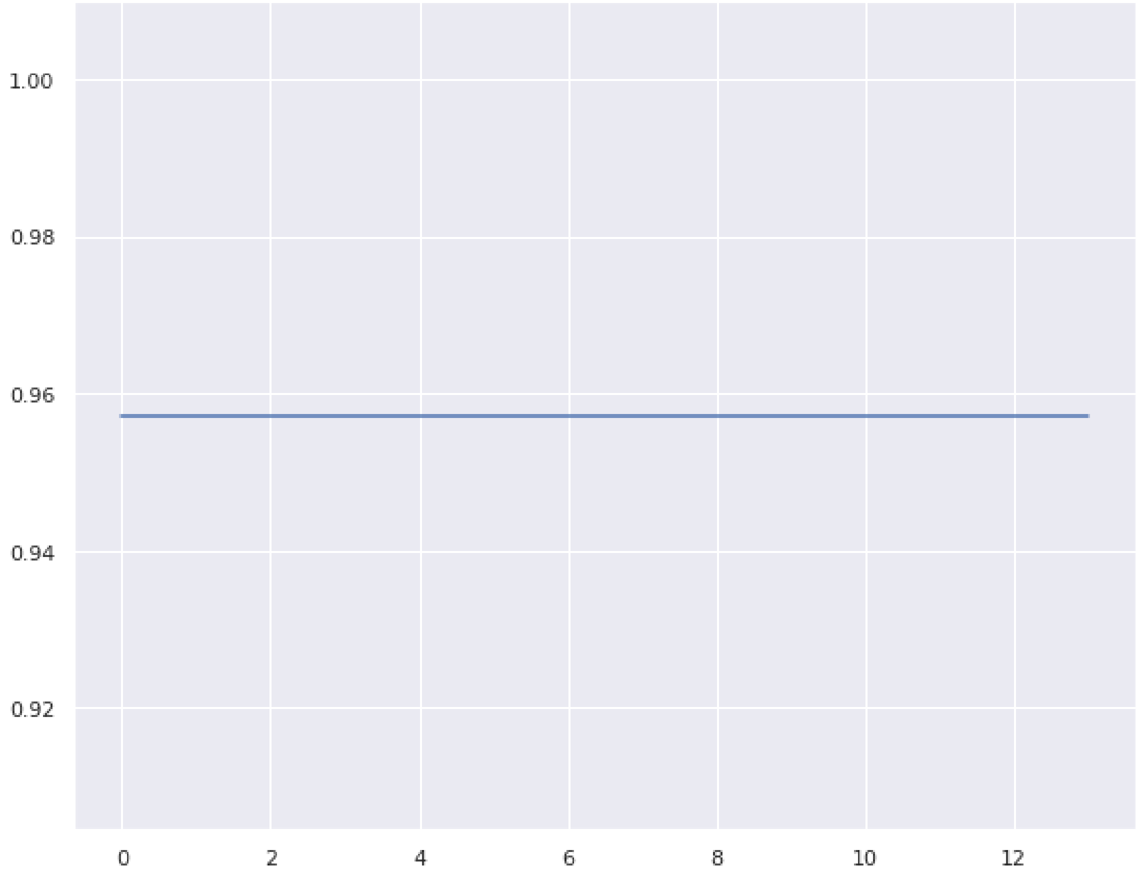

```
For LR: 0.003, Iterations= 50000
======================================
R2 Score:  0.9571728152912142
Mean absolute error:  25.51571767046275
Root Mean squared error:  54.844232714354575
Explained Variance Score:  0.9464623975881455
Wrote to file sucessfully.
```

In [27]:

```
plt.plot(r2_lst)
```

Out[27]:

```
[<matplotlib.lines.Line2D at 0x7f922f9eacd0>]
```



Conclusion:

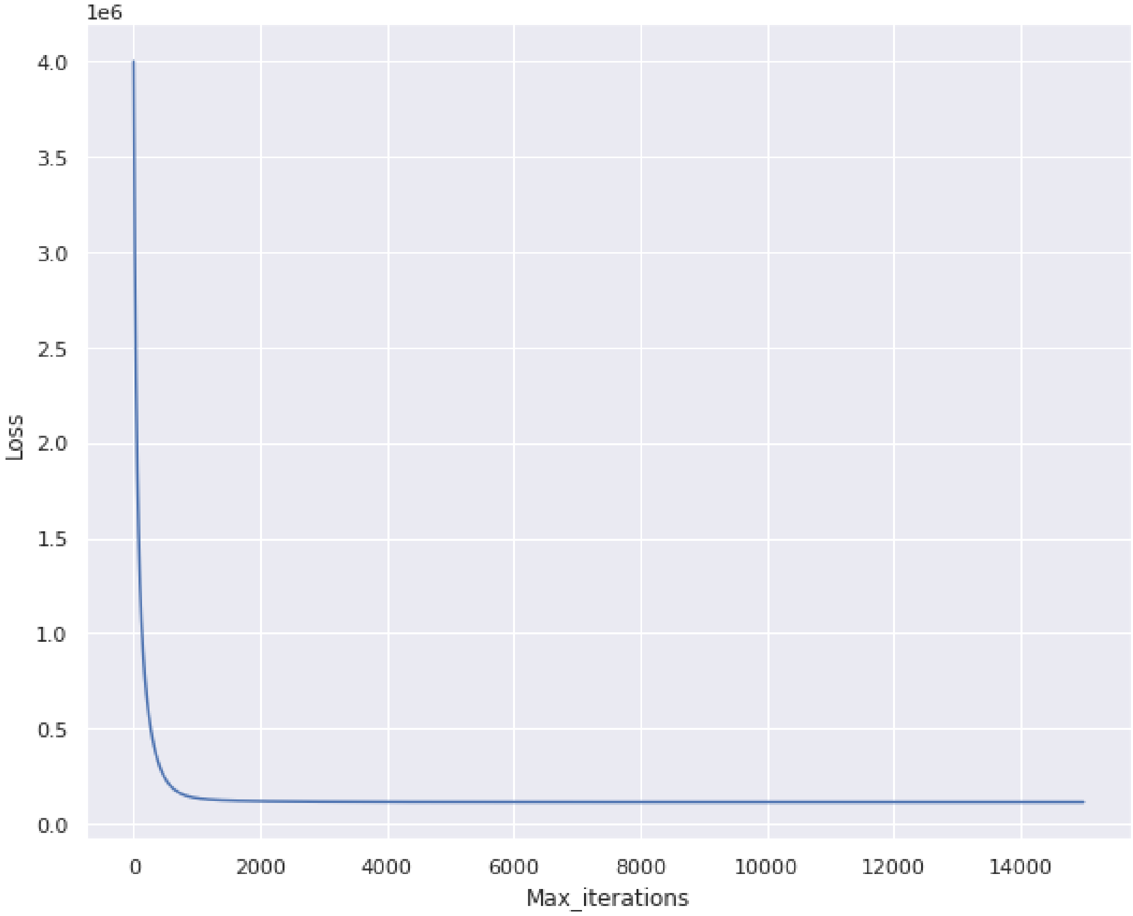
From the above plot, it can observed that increasing n_iterations above 15000 does not improve r2_score

The increase is of the order of 10^-7, which is very insignificant for so many iterations

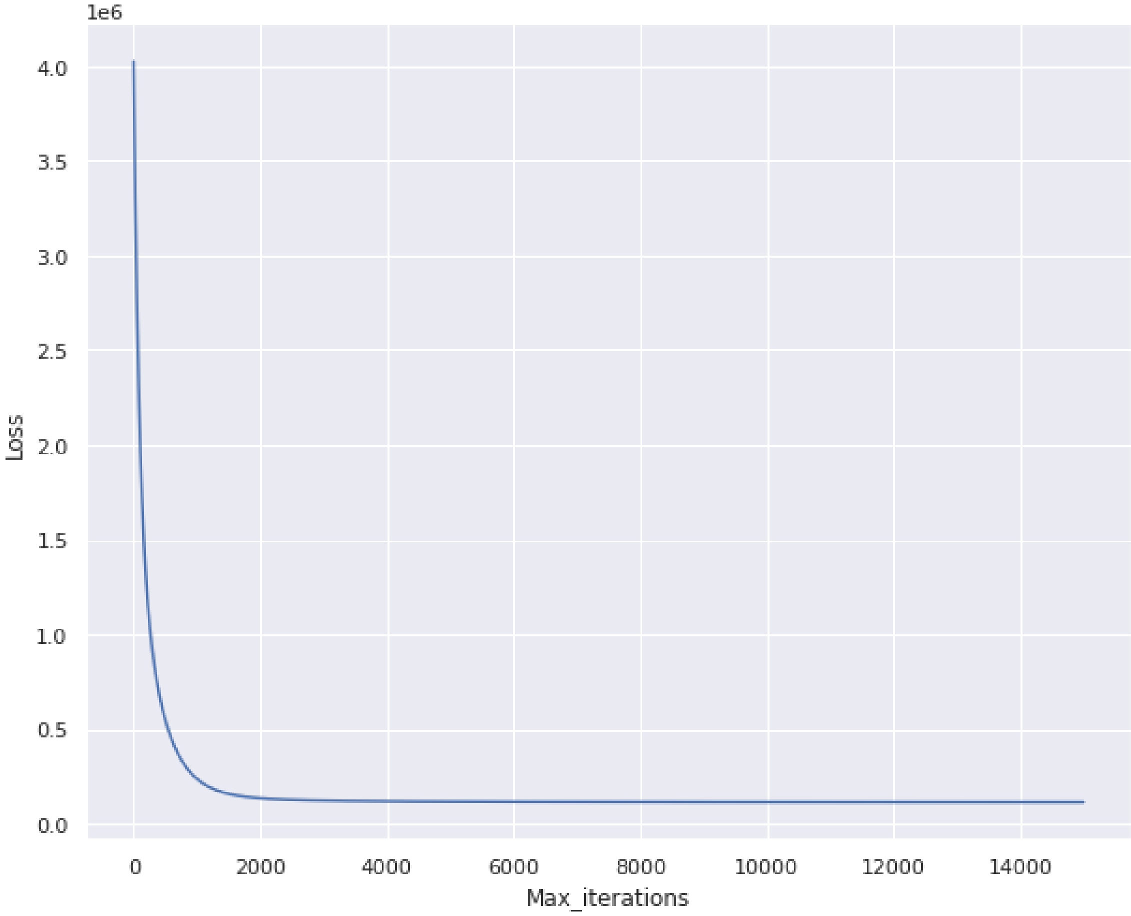
Now we can try altering learning_rate

In [28]:

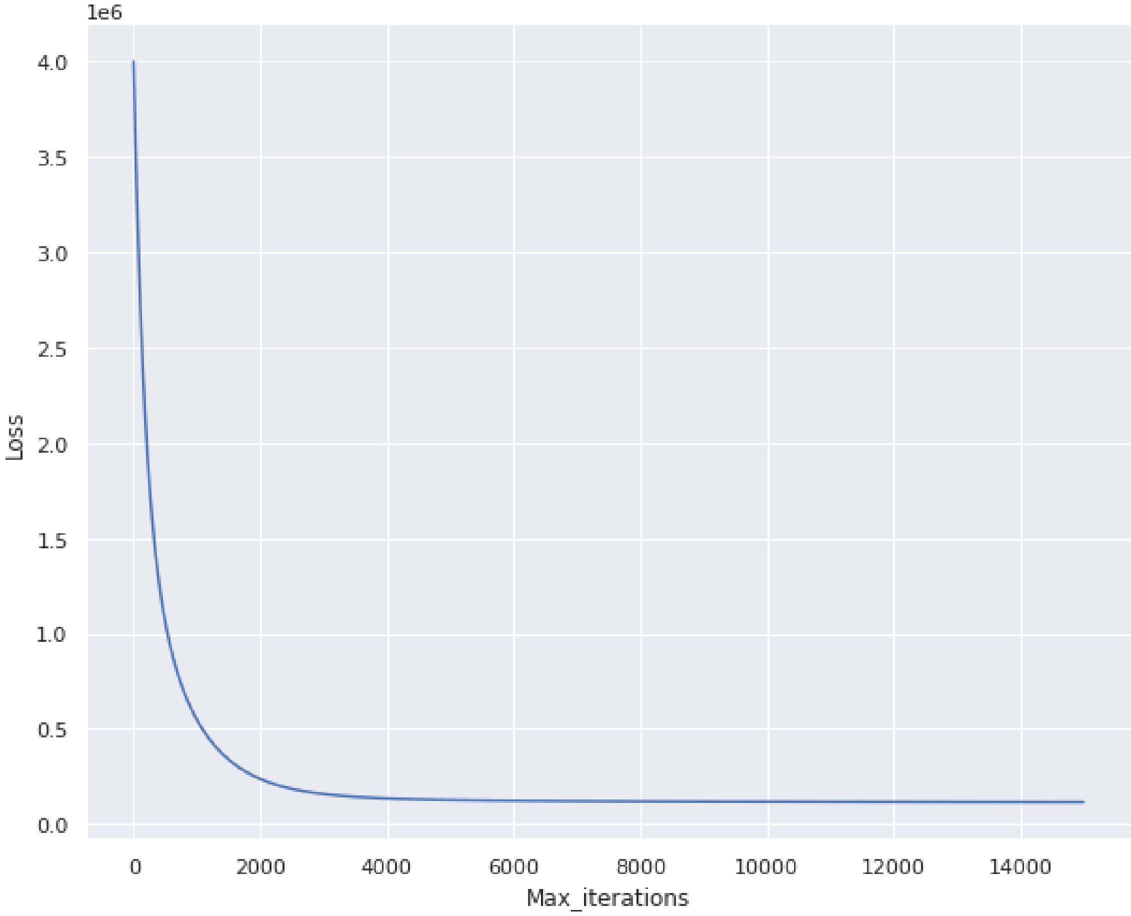
```
#try combination of lr by keeping n_iterations constant i.e 15000
lr = 0.003
r2_lst = []

while lr >= 0.0001:
    model = ManualSGD(learning_rate=lr,max_iterations=15000)

    #fit the training data on the model
    model.fit(X_train_sc,np.array(y_train))

    #predict
    y_pred=model.predict(X_test_sc)

    loss=list(model.losses)

    #visualize loss
    plt.plot(loss)
    plt.xlabel("Max_iterations")
    plt.ylabel("Loss")
    plt.show()

    r2 = model.Rsquared(X_train_sc,np.array(y_train))
    mae = mean_absolute_error(y_test, y_pred[0])
    rmse = mean_squared_error(y_test, y_pred[0], squared=False)
    evs = explained_variance_score(y_test, y_pred[0])

    print("\nFor LR: "+str(lr)+", Iterations= "+str(itrs))
    print("======================================")
    print("R2 Score: ", r2)
    print("Mean absolute error: ", mae)
    print("Root Mean squared error: ", rmse)
    print("Explained Variance Score: ", evs)

    file = open("Manual_SGD_log.txt","a")
    file.write("LR = " + str(lr) + ", max_iterations = " + str(itrs) +
               ", R^2 = " + str(r2) + ", MAE = " + str(mae) + ", RMSE = " +
               str(rmse) + ", Explained-Variance = " + str(evs) + " \n")
    file.close()
    print("Wrote to file sucessfully.")

    r2_lst.append(np.around(r2,2))
    lr/=2
```



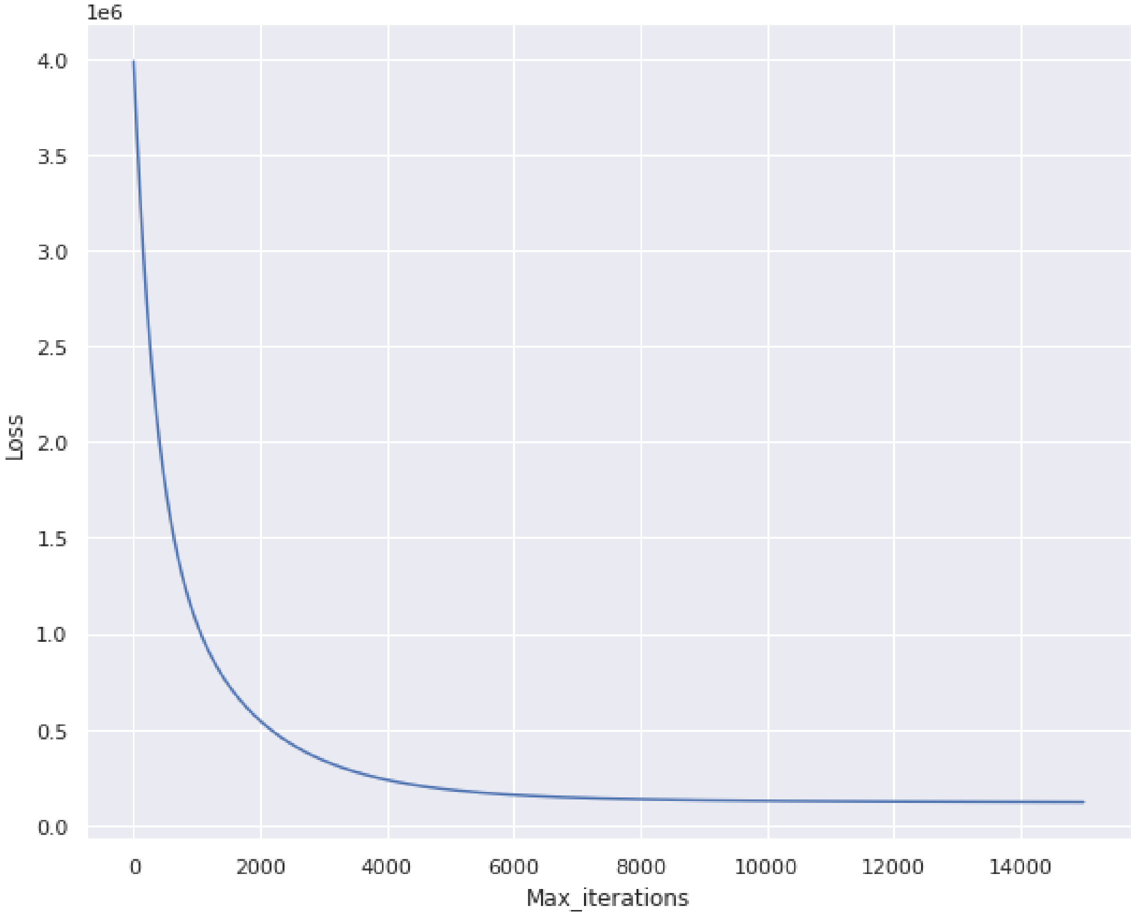

```
For LR: 0.003, Iterations= 52500
======================================
R2 Score:  0.9571726216743162
Mean absolute error:  25.526426996845554
Root Mean squared error:  54.87907491997556
Explained Variance Score:  0.9463947117469326
Wrote to file sucessfully.
```



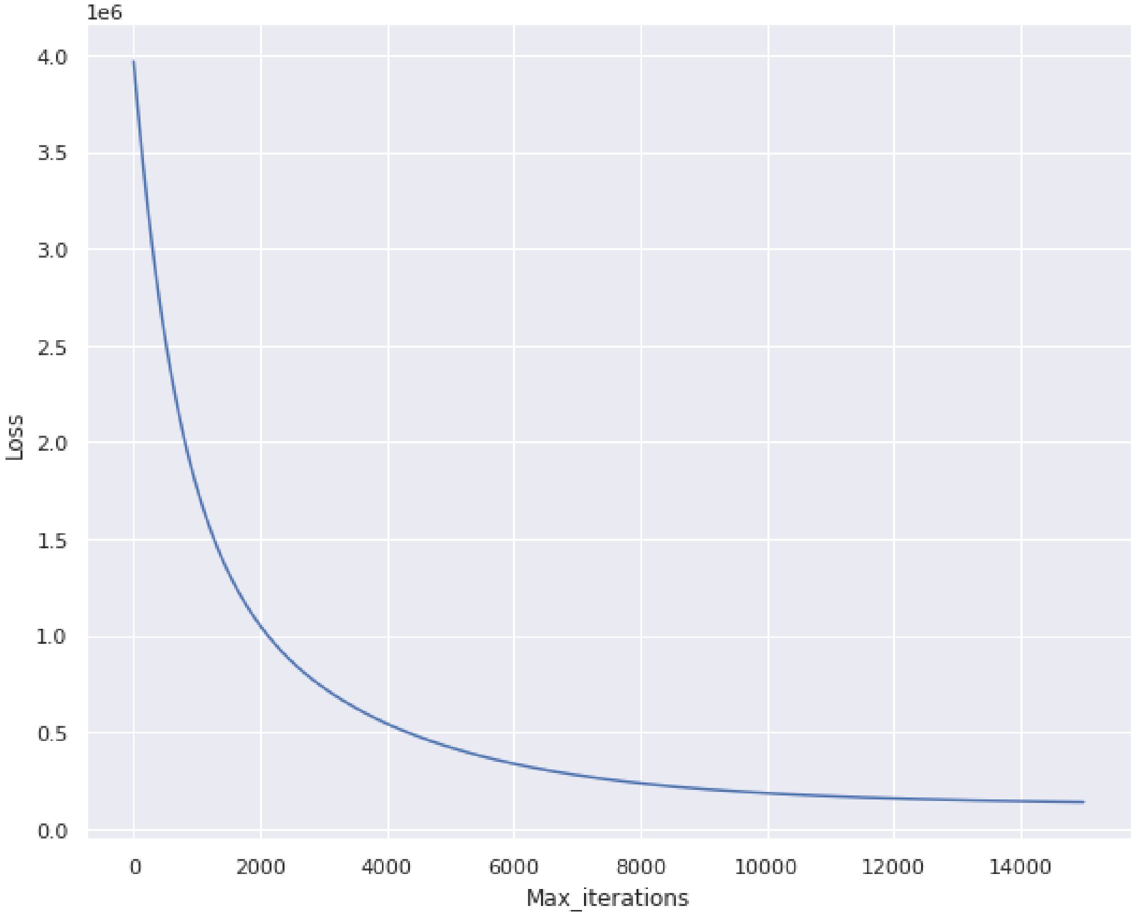

```
For LR: 0.0015, Iterations= 52500
======================================
R2 Score:  0.9571405656048058
Mean absolute error:  25.629076926462837
Root Mean squared error:  55.23074618035433
Explained Variance Score:  0.9457174268021795
Wrote to file sucessfully.
```




```
For LR: 0.00075, Iterations= 52500
======================================
R2 Score:  0.9567020917804513
Mean absolute error:  25.75930566780692
Root Mean squared error:  55.900820262828724
Explained Variance Score:  0.9444579482298912
Wrote to file sucessfully.
```




```
For LR: 0.000375, Iterations= 52500
======================================
R2 Score:  0.9549677341191553
Mean absolute error:  25.897145650679754
Root Mean squared error:  56.662044289847884
Explained Variance Score:  0.9430712074590496
Wrote to file sucessfully.
```



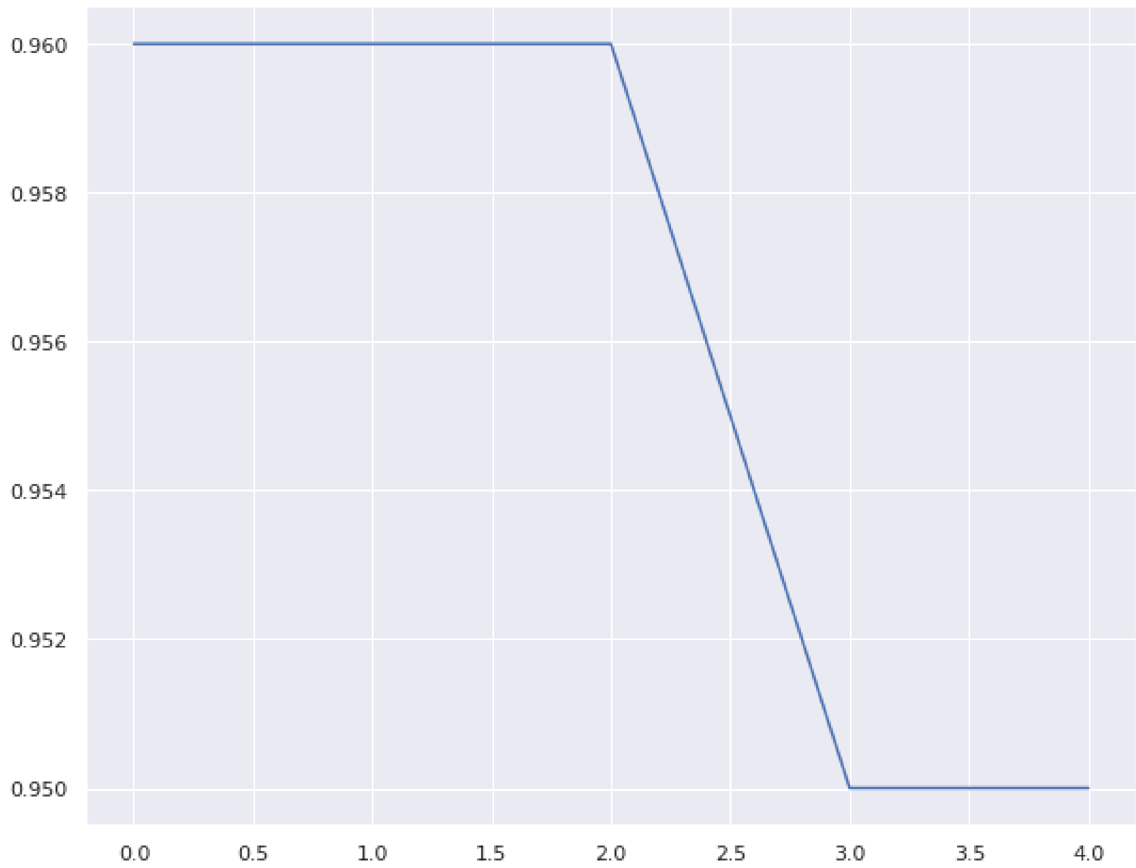

```
For LR: 0.0001875, Iterations= 52500
======================================
R2 Score:  0.9480591086311615
Mean absolute error:  27.518134885629756
Root Mean squared error:  60.72328322993885
Explained Variance Score:  0.9367437165109976
Wrote to file sucessfully.
```

In [29]:

```
plt.plot(r2_lst)
```

Out[29]:

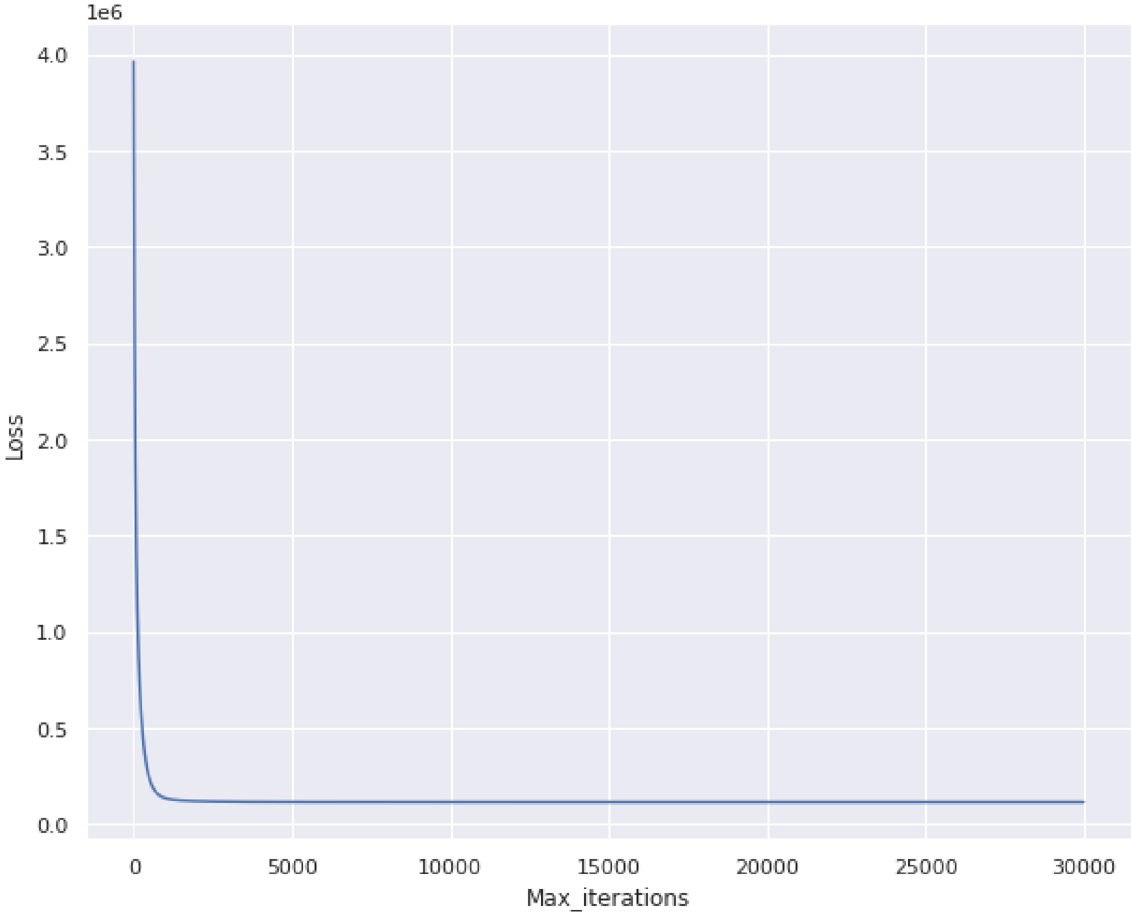
```
[&lt;matplotlib.lines.Line2D at 0x7f922e516550&gt;]
```



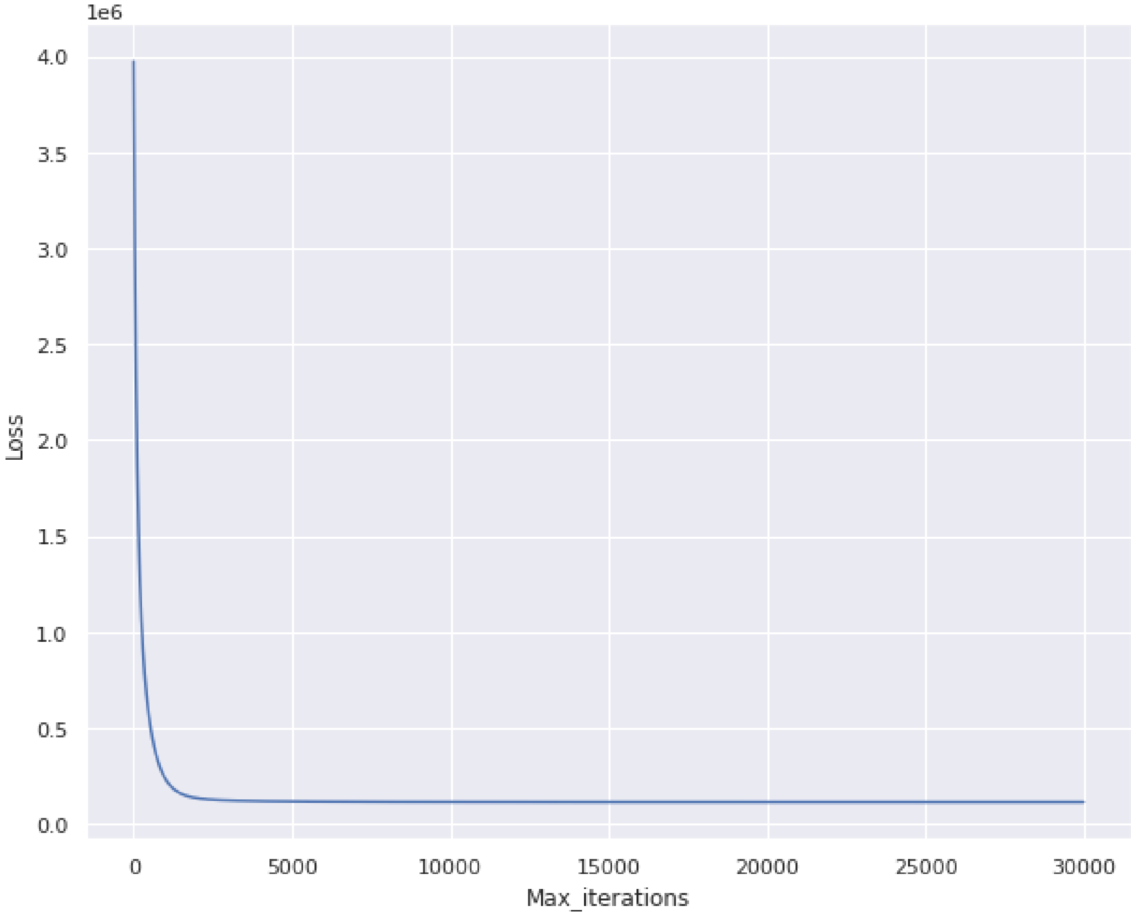
The accuracy falls when learning rate is altered while keeping the n_iterations same

In [30]:

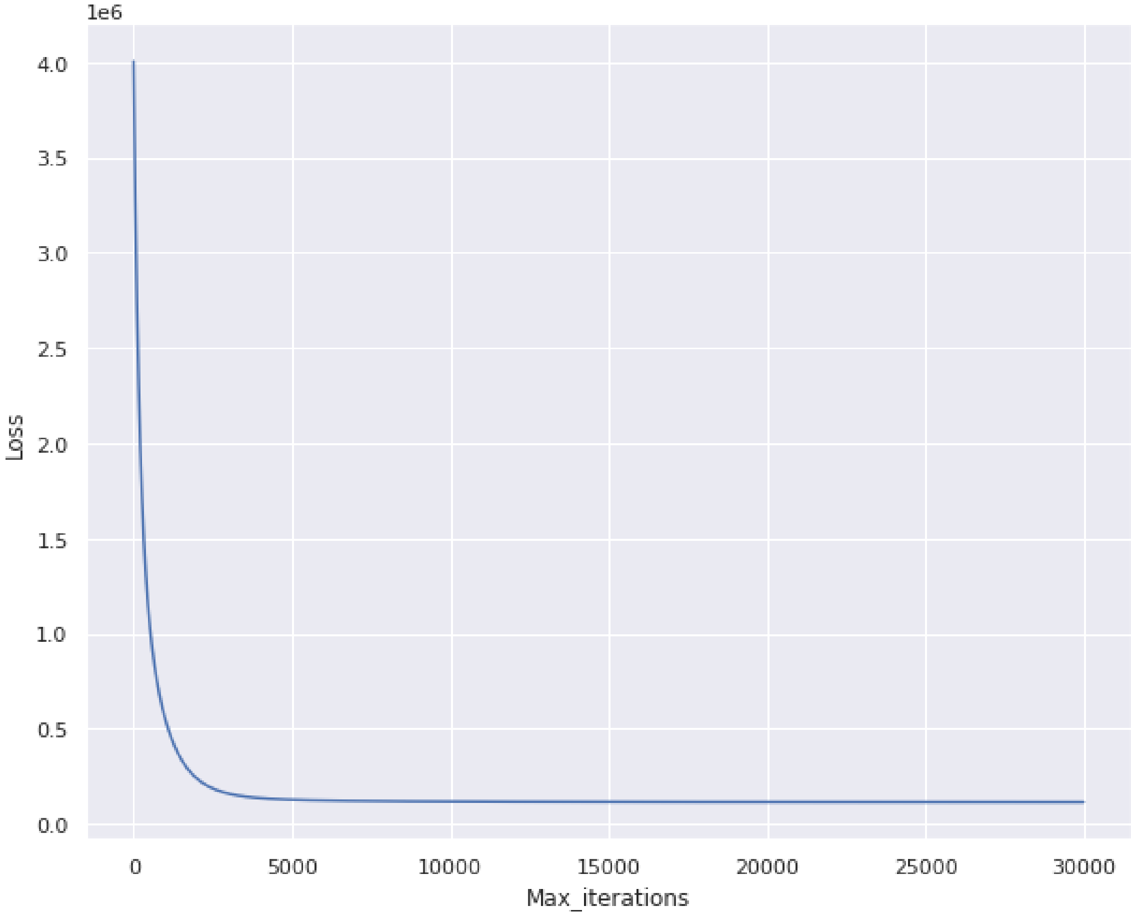
```
lr = 0.003
r2_lst = []

while lr >= 0.0001:
    model = ManualSGD(learning_rate=lr,max_iterations=30000)

    #fit the training data on the model
    model.fit(X_train_sc,np.array(y_train))

    #predict
    y_pred=model.predict(X_test_sc)

    loss=list(model.losses)

    #visualize loss
    plt.plot(loss)
    plt.xlabel("Max_iterations")
    plt.ylabel("Loss")
    plt.show()

    r2 = model.Rsquared(X_train_sc,np.array(y_train))
    mae = mean_absolute_error(y_test, y_pred[0])
    rmse = mean_squared_error(y_test, y_pred[0], squared=False)
    evs = explained_variance_score(y_test, y_pred[0])

    print("\nFor LR: "+str(lr)+", Iterations= "+str(itrs))
    print("======================================")
    print("R2 Score: ", r2)
    print("Mean absolute error: ", mae)
    print("Root Mean squared error: ", rmse)
    print("Explained Variance Score: ", evs)

    file = open("Manual_SGD_log.txt","a")
    file.write("LR = " + str(lr) + ", max_iterations = " + str(itrs) +
               ", R^2 = " + str(r2) + ", MAE = " + str(mae) + ", RMSE = " +
               str(rmse) + ", Explained-Variance = " + str(evs) + " \n")
    file.close()
    print("Wrote to file sucessfully.")

    r2_lst.append(r2)
    lr/=2
```



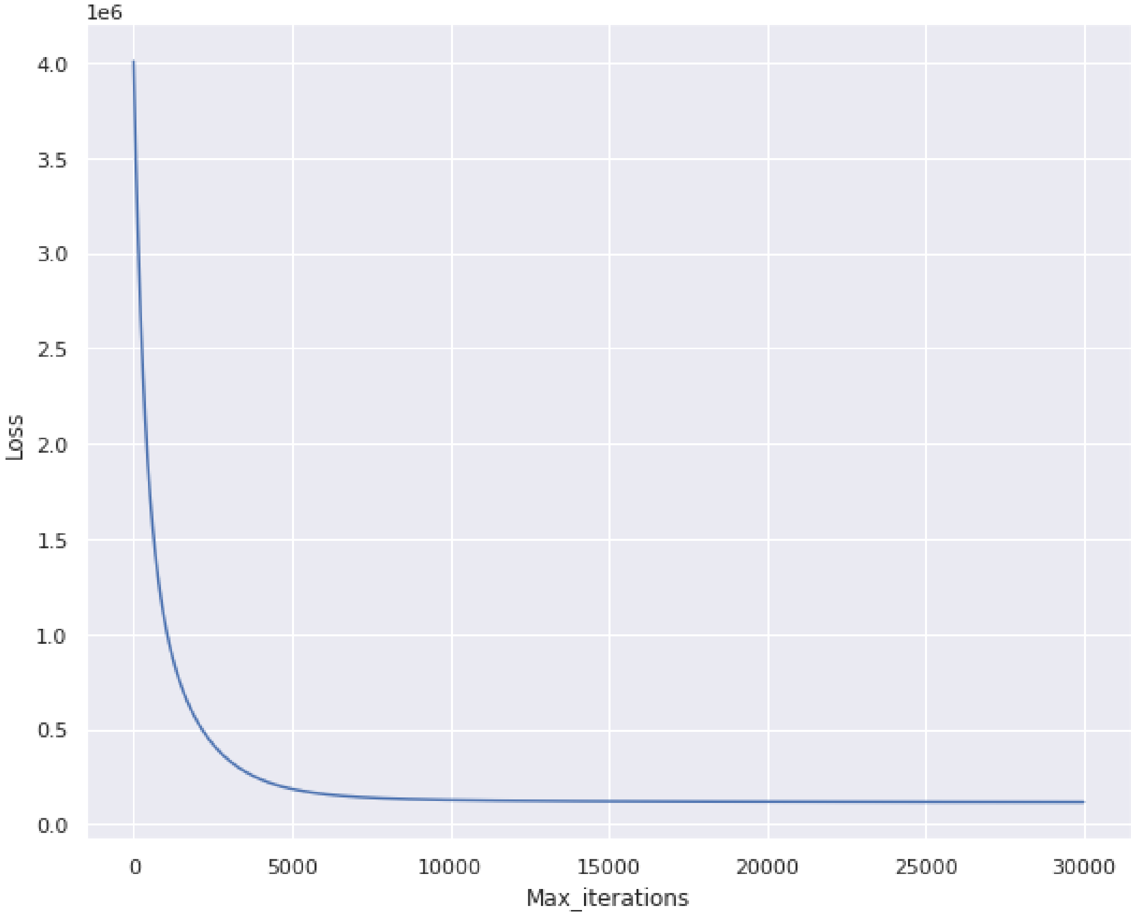

```
For LR: 0.003, Iterations= 52500
======================================
R2 Score:  0.957172815283322
Mean absolute error:  25.515788493252806
Root Mean squared error:  54.84446155383323
Explained Variance Score:  0.9464619522249244
Wrote to file sucessfully.
```



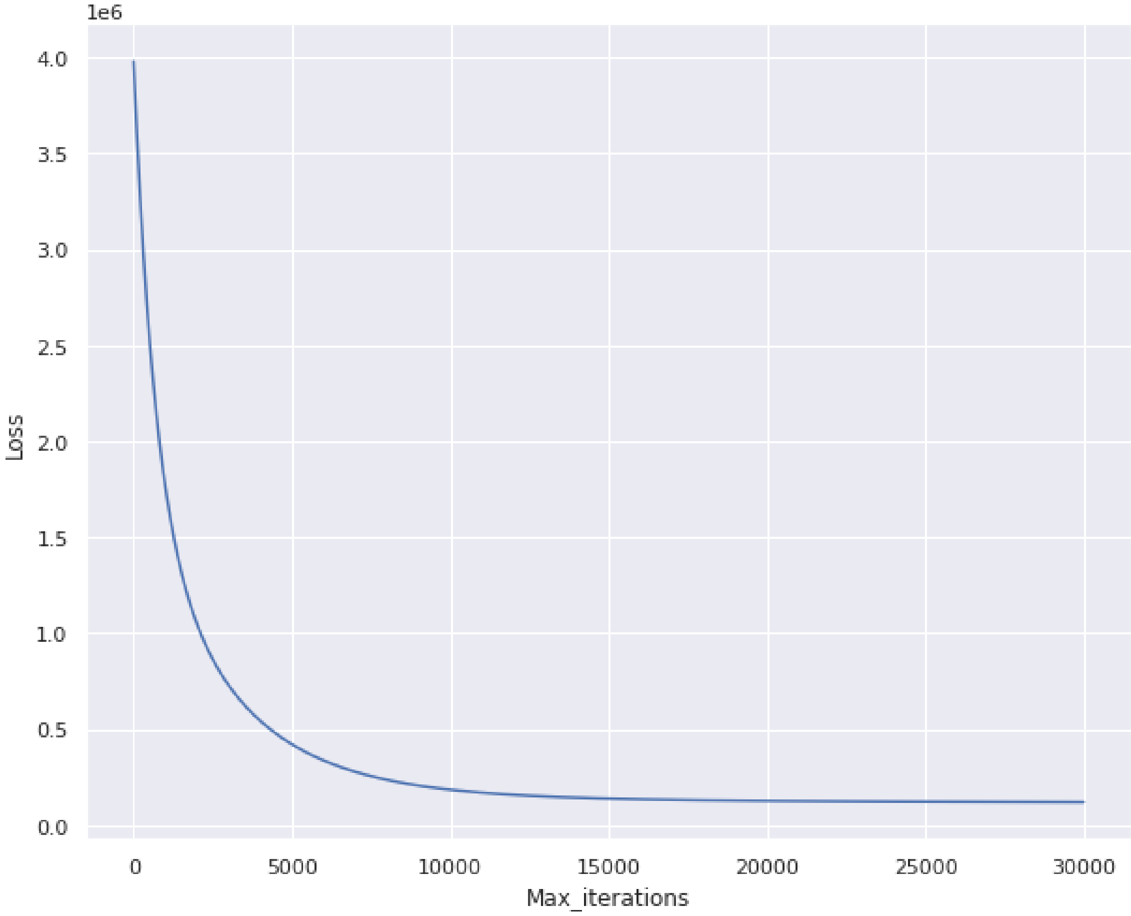

```
For LR: 0.0015, Iterations= 52500
======================================
R2 Score:  0.9571726168483918
Mean absolute error:  25.526572660020747
Root Mean squared error:  54.87954240570421
Explained Variance Score:  0.9463937983172975
Wrote to file sucessfully.
```




```
For LR: 0.00075, Iterations= 52500
======================================
R2 Score:  0.9571397434917757
Mean absolute error:  25.62997047277682
Root Mean squared error:  55.234194838556036
Explained Variance Score:  0.9457109680083321
Wrote to file sucessfully.
```




```
For LR: 0.000375, Iterations= 52500
======================================
R2 Score:  0.9567126289489319
Mean absolute error:  25.7473564715255
Root Mean squared error:  55.86300771982175
Explained Variance Score:  0.944534172478918
Wrote to file sucessfully.
```



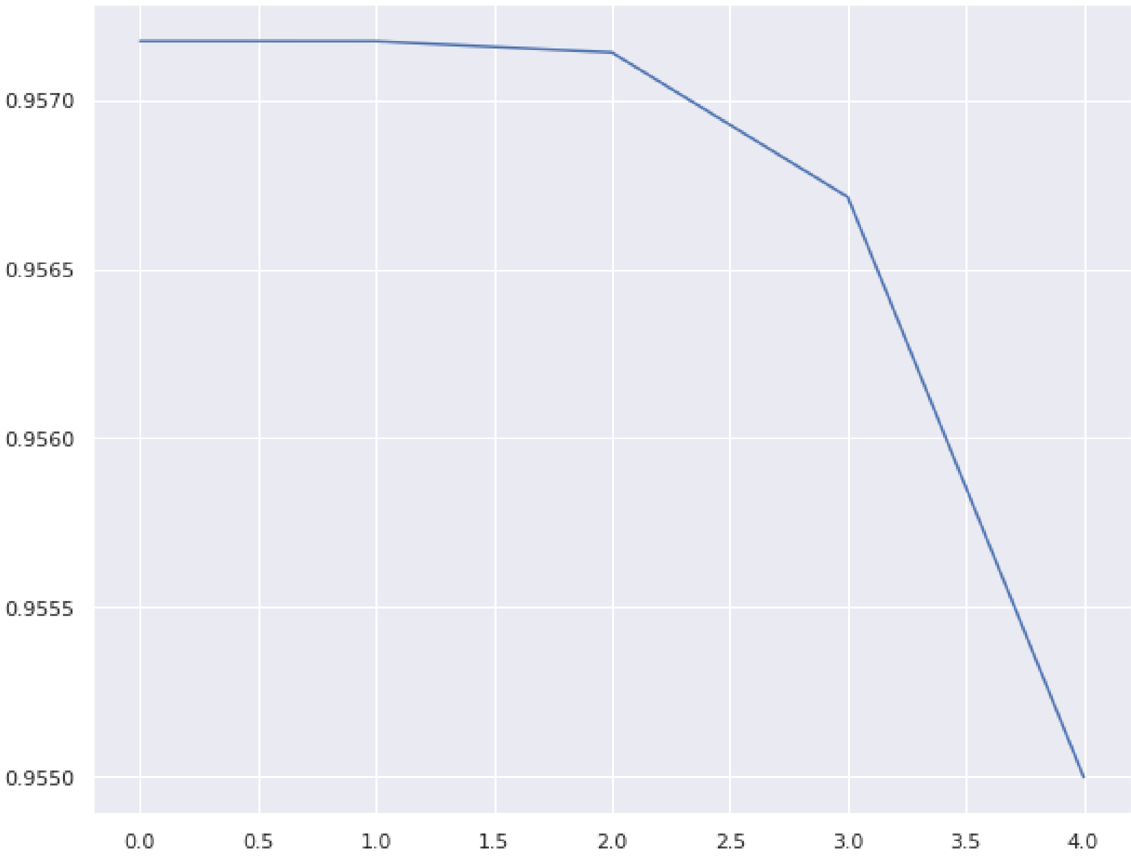

```
For LR: 0.0001875, Iterations= 52500
======================================
R2 Score:  0.9549991121300001
Mean absolute error:  25.87462331295083
Root Mean squared error:  56.63143046728613
Explained Variance Score:  0.9431276147057309
Wrote to file sucessfully.
```

In [31]:

```
plt.plot(r2_lst)
```

Out[31]:

```
[<matplotlib.lines.Line2D at 0x7f922e78ce90>]
```



From the above plot we can observe that increasing iterations also doesn't help much.

PART 2 - Impelementing SGD regressor using Scikit-learn library

In [32]:

```
#use GridSearchCV to loop through predefined hyperparameters and
#    fit your estimator (model) on your training set to find
#    the best parameters from the listed hyperparameters

p={'learning_rate': ['constant'], 'eta0': [0.001, 0.01, 0.02, 0.05, 0.08, 0.1],
   'max_iter':[500, 1000, 2000, 5000, 7000, 10000, 15000, 20000, 25000, 30000,
               35000, 40000, 45000, 50000]}

sgd=SGDRegressor()

model=GridSearchCV(sgd,param_grid=p)
```

In [33]:

```
#fit the data on the model created
model.fit(X_train_sc,y_train)
```

Out[33]:

```
GridSearchCV(estimator=SGDRegressor(),
             param_grid={'eta0': [0.001, 0.01, 0.02, 0.05, 0.08, 0.1],
                         'learning_rate': ['constant'],
                         'max_iter': [500, 1000, 2000, 5000, 7000, 10000,
15000,
                                      20000, 25000, 30000, 35000, 40000, 4
5000,
                                      50000]})
```

In [34]:

```
#print best estimators found by the model
model.best_estimator_
```

Out[34]:

```
SGDRegressor(eta0=0.02, learning_rate='constant', max_iter=30000)
```

In [35]:

```
#print best parameters found by the model
print(model.best_params_)
```

```
{'eta0': 0.02, 'learning_rate': 'constant', 'max_iter': 30000}
```

In [36]:

```
#print best score found by the model
print(model.best_score_)
```

```
0.9400688104995936
```

In [37]:

```
#predict using the model
y_pred=model.predict(X_test_sc)
```

In [38]:

```
#calculate metrics
r2 = model.score(X_train_sc,y_train)
mae = mean_absolute_error(y_test, y_pred)
rmse = mean_squared_error(y_test, y_pred, squared=False)
evs = explained_variance_score(y_test, y_pred)

print("R2 Score: ", r2)
print("Mean absolute error: ", mae)
print("Root Mean squared error: ", rmse)
print("Explained Variance Score: ", evs)
```

```
R2 Score:  0.9426297591728554
Mean absolute error:  30.693156077522872
Root Mean squared error:  66.38940030040736
Explained Variance Score:  0.9225820701019357
```

Conclusion: We obtained a r2 score of 95.16% using scikit library which is less than that of our custom SGD regressor's score using GridSearchCV

Comparing model performance with same set of parameters as that of our custom model

In [39]:

```
#Function to display loss curve during each iteration
class DisplayLossCurve(object):
  def __init__(self, print_loss=False):
    self.print_loss = print_loss

  """Make sure the model verbose is set to 1"""
  def __enter__(self):
    self.old_stdout = sys.stdout
    sys.stdout = self.mystdout = io.StringIO()

  def __exit__(self, *args, **kwargs):
    sys.stdout = self.old_stdout
    loss_history = self.mystdout.getvalue()
    loss_list = []
    for line in loss_history.split('\n'):
      if(len(line.split("loss: ")) == 1):
        continue
      loss_list.append(float(line.split("loss: ")[-1]))
    plt.figure()
    plt.plot(np.arange(len(loss_list)), loss_list)
    plt.xlabel("Epoch")
    plt.ylabel("Loss")

    if self.print_loss:
      print("=============== Loss Array ===============")
      print(np.array(loss_list))

    return True
```

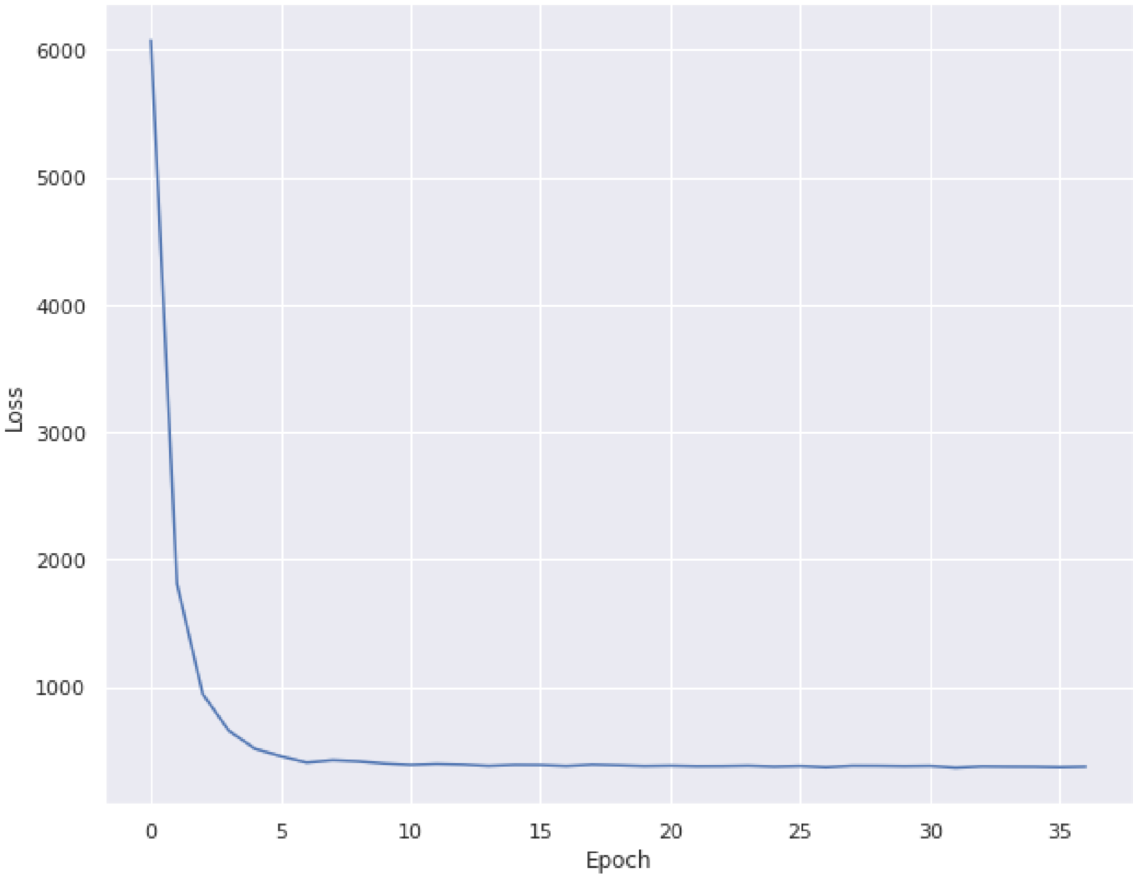
In [40]:

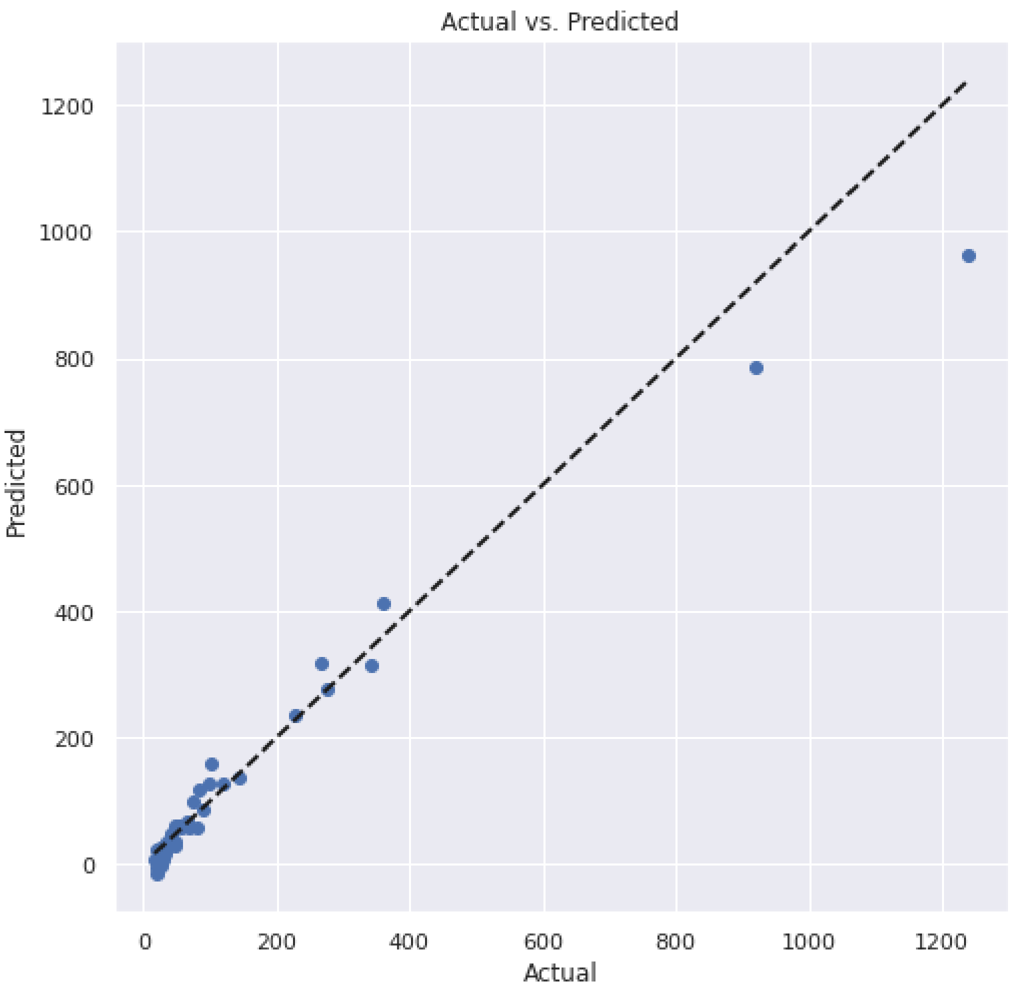
```
lr, itrs = 0.003, 15000
model=SGDRegressor(learning_rate='constant', eta0=lr, max_iter=itrs, verbose=1)
```

In [41]:

```
with DisplayLossCurve(print_loss=True):
    model.fit(X_train_sc,y_train)

y_pred=model.predict(X_test_sc)

r2 = model.score(X_train_sc,y_train)
mae = mean_absolute_error(y_test, y_pred)
rmse = mean_squared_error(y_test, y_pred, squared=False)
evs = explained_variance_score(y_test, y_pred)

r2_lst.append(r2)

print()
print("LR: "+str(lr)+" Iterations= "+str(15000))
print("R2 Score: ", r2)
print("Mean absolute error: ", mae)
print("Root Mean squared error: ", rmse)
print("Explained Variance Score: ", evs)

file = open("Scikit_SGD_log.txt","a")
file.write("LR = " + str(lr) + ",max_iterations = " + str(itrs) + " R^2 = " +
           str(r2) + ", MAE = " + str(mae) + ", RMSE = " + str(rmse) +
           ",  Explained-Variance = " + str(evs) + " \n")
file.close()

plt.figure(figsize = (8,8))
plt.scatter(y_test, y_pred)
plt.plot([y_test.min(),y_test.max()],[y_test.min(),y_test.max()],'k--',lw=2)
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.title('Actual vs. Predicted')
plt.show()

file = open("Scikit_SGD_log.txt","a")
file.write("LR = " + str(lr) + ", max_iterations = " + str(itrs) + ", R^2 = " +
           str(r2) + ", MAE = " + str(mae) + ", RMSE = " + str(r2) +
           ", Explained-Variance = " + str(evs) + " \n")
file.close()
print("Wrote to file sucessfully.")
```

```
=============== Loss Array ===============
[6074.141682 1812.777835  943.7183    657.620291  516.093557  457.495796
  408.347118  426.823423  417.63173   399.898885  389.158239  396.61693
  391.665313  381.198069  389.29442   388.180112  380.120248  391.028287
  386.078642  379.787734  382.843997  378.615876  379.319834  382.701272
  375.815245  380.435192  370.899655  382.11838   381.610133  379.115317
  381.152597  367.206297  377.523637  375.750139  375.159381  371.724194
  375.716747]

LR: 0.003 Iterations= 15000
R2 Score:  0.9565421203737108
Mean absolute error:  25.47748191911172
Root Mean squared error:  51.82665486241274
Explained Variance Score:  0.9521341027181411
```






```
Wrote to file sucessfully.
```

In [42]:

```
#weights obtained
model.coef_
```

Out[42]:

```
array([ 8.80955631, 13.13337567, 46.21912906, 13.49819145, -1.2626479 ,
        6.67382171, 65.88985961])
```

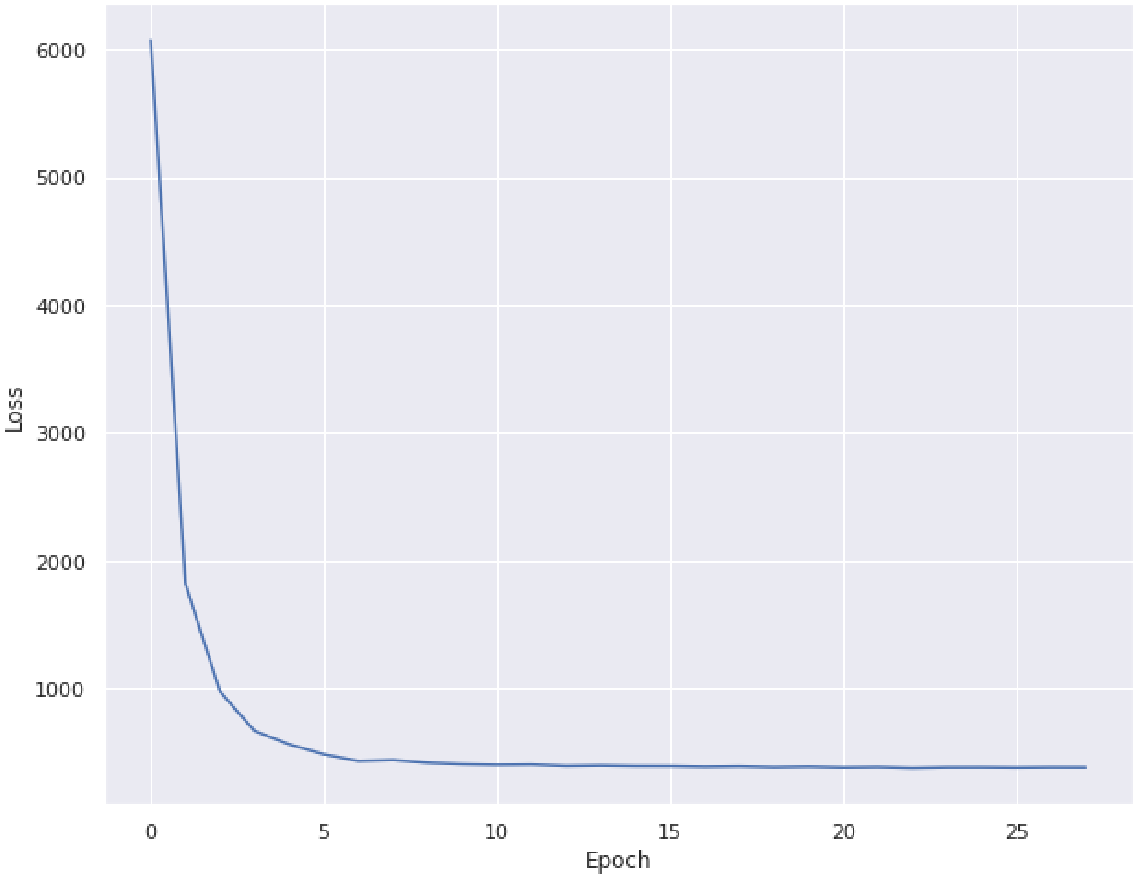
We obtained a r2 score of 95.16% using scikit library which is less than that of our custom SGD regressor's score

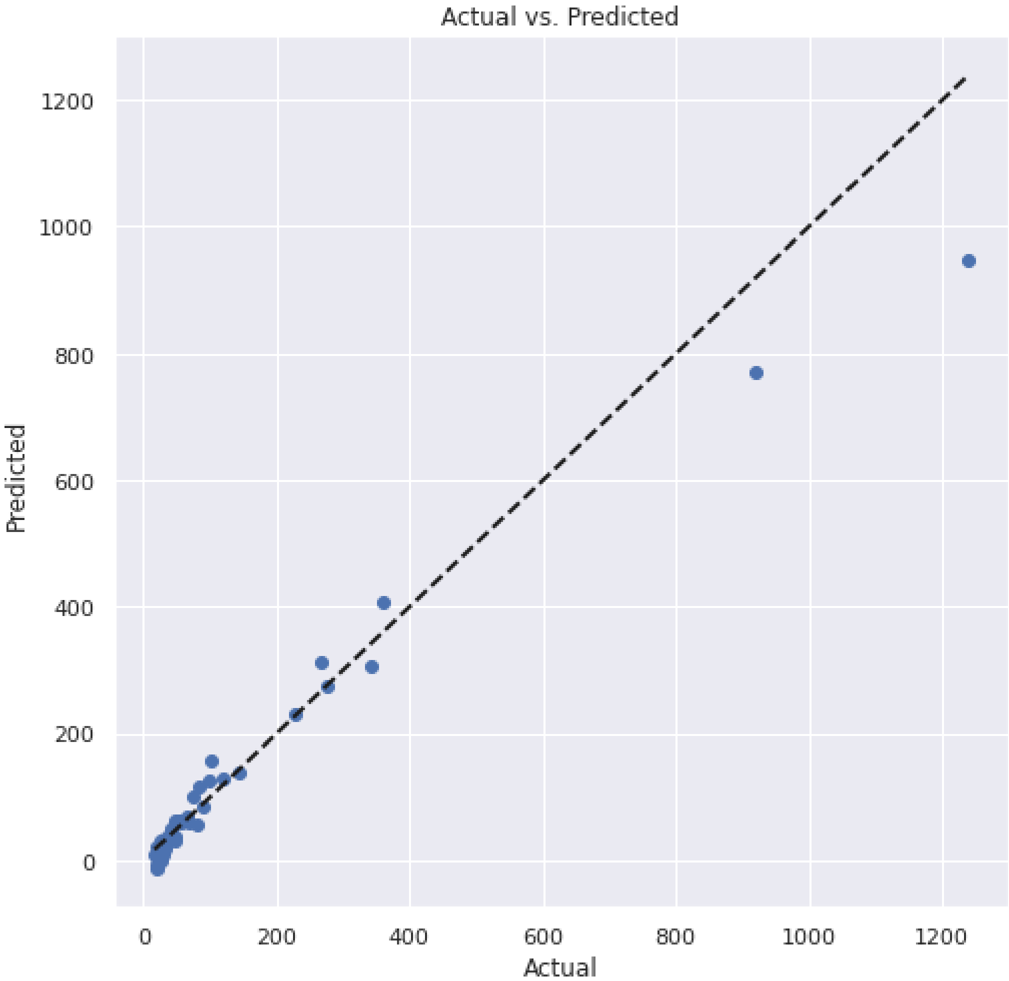
Now we can try altering iterations to observe how it affects r2 score like we did in part 1

In [43]:

```
itrs = 17500
lr =0.003
r2_lst = []

while itrs <= 50000:
    model=SGDRegressor(learning_rate='constant', eta0=lr, max_iter=itrs, verbose=1)

    with DisplayLossCurve(print_loss=True):
        model.fit(X_train_sc,y_train)

    y_pred=model.predict(X_test_sc)

    r2 = model.score(X_train_sc,y_train)
    mae = mean_absolute_error(y_test, y_pred)
    rmse = mean_squared_error(y_test, y_pred, squared=False)
    evs = explained_variance_score(y_test, y_pred)

    r2_lst.append(r2)

    print()
    print("For LR: "+str(lr)+", Iterations= "+str(itrs))
    print("======================================")
    print("R2 Score: ", r2)
    print("Mean absolute error: ", mae)
    print("Root Mean squared error: ", rmse)
    print("Explained Variance Score: ", evs)

    file = open("Scikit_SGD_log.txt","a")
    file.write("LR = " + str(lr) + ",max_iterations = " + str(itrs) + " R^2 = "
               + str(r2) + ", MAE = " + str(mae) + ", RMSE = " + str(rmse) +
               ",  Explained-Variance = " + str(evs) + " \n")
    file.close()

    plt.figure(figsize = (8,8))
    plt.scatter(y_test, y_pred)
    plt.plot([y_test.min(),y_test.max()],[y_test.min(),y_test.max()],'k--',lw=2)
    plt.xlabel('Actual')
    plt.ylabel('Predicted')
    plt.title('Actual vs. Predicted')
    plt.show()

    itrs+=2500
```

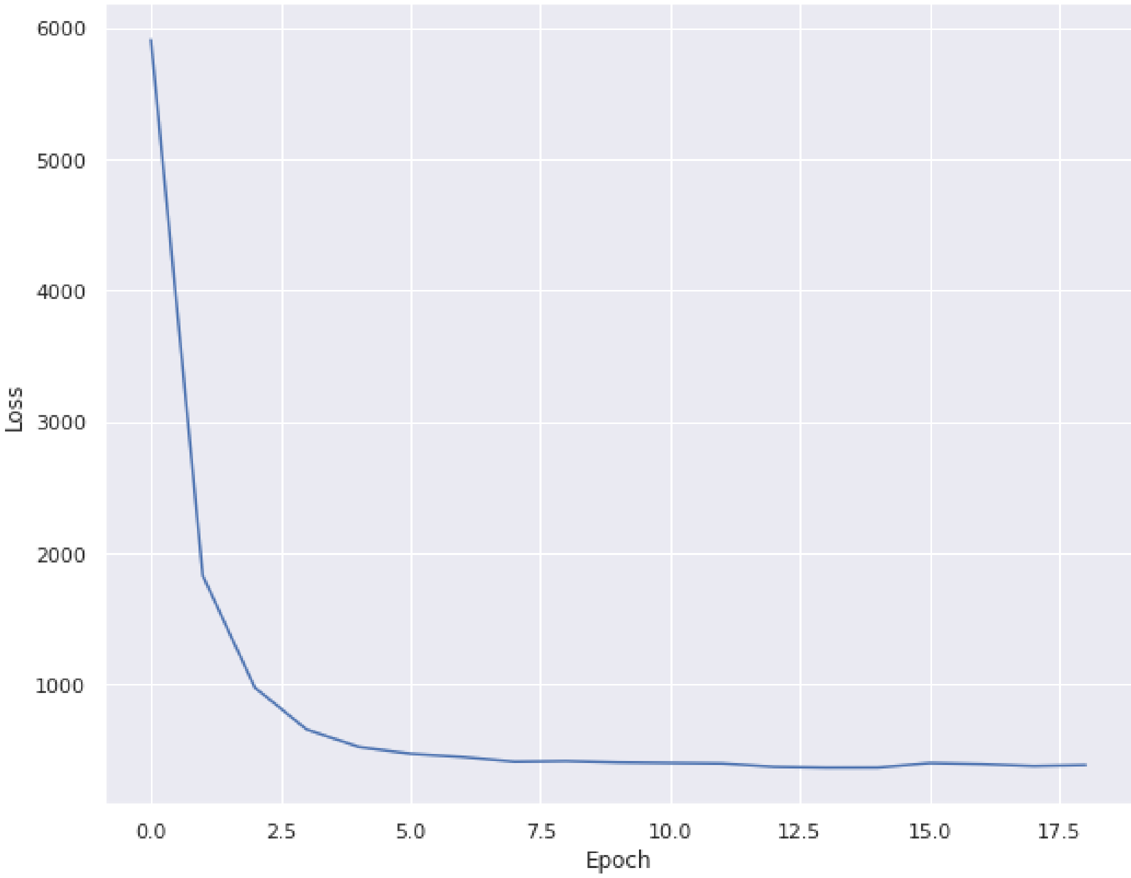
```
=============== Loss Array ===============
[6071.038953 1821.46348   973.892193  664.369026  559.125936  482.120173
  429.556781  437.611557  415.324092  404.906168  400.036022  402.486903
  391.537424  394.709427  390.736637  390.237003  385.34474   388.301266
  381.873369  385.593655  379.811441  382.929266  375.63037   380.796415
  381.15503   378.901254  381.270332  380.773594]

For LR: 0.003, Iterations= 17500
======================================
R2 Score:  0.9569525128096922
Mean absolute error:  25.593928071452304
Root Mean squared error:  54.18557019096956
Explained Variance Score:  0.9476412193179591
```

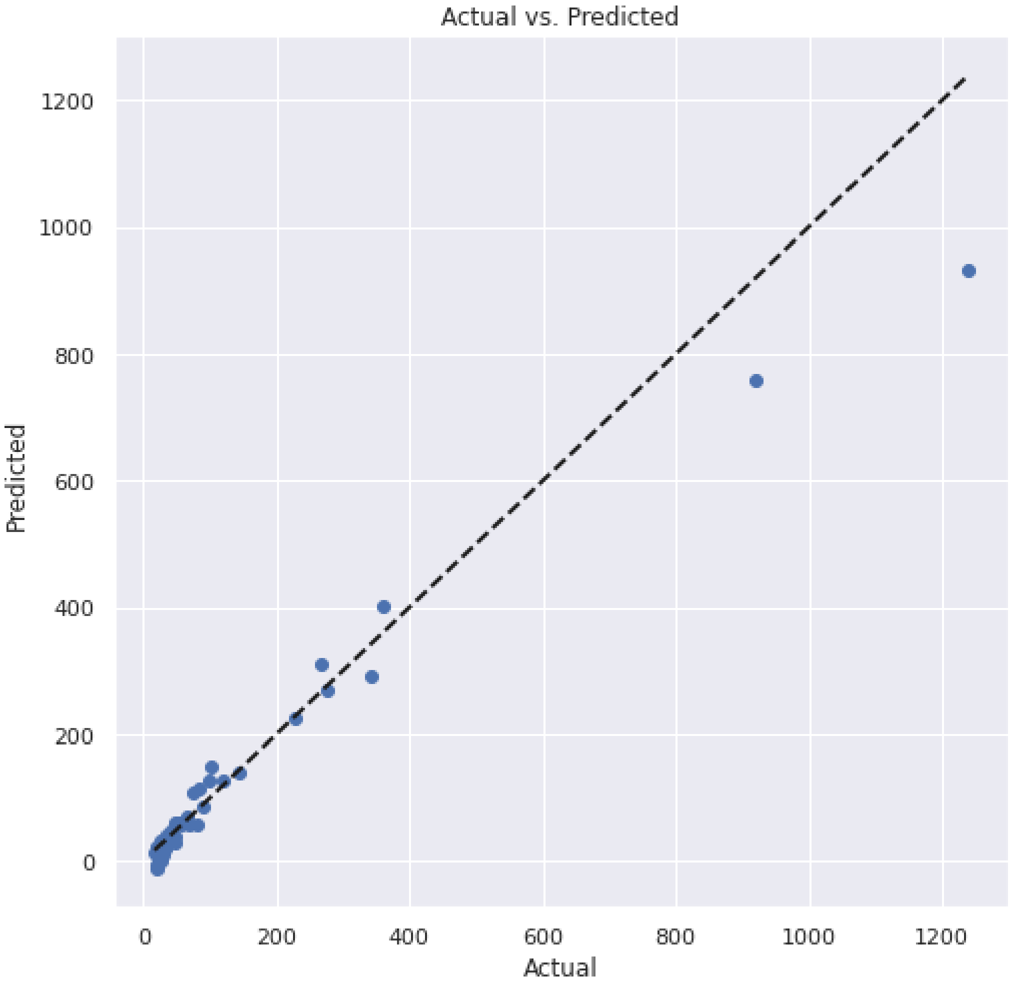




Actual vs. Predicted
Predicted
Actual
0
200
400
600
800
1000
1200

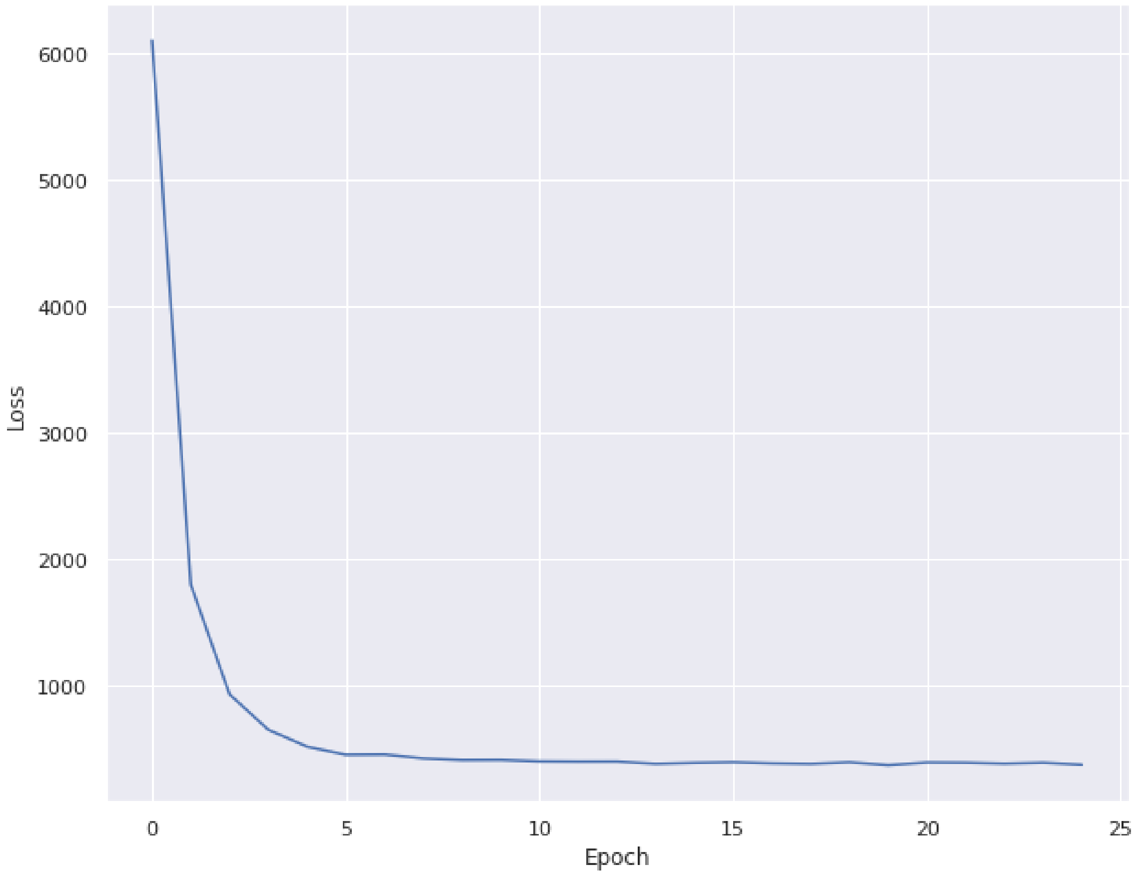
```
=============== Loss Array ===============
[5906.547144 1826.898242  976.253072  656.777009  524.13179   471.988561
  447.617112  412.162862  417.062677  406.351722  401.935151  397.854157
  372.587751  366.416922  366.942491  401.05991   392.835361  378.366674
  385.796903]

For LR: 0.003, Iterations= 20000
======================================
R2 Score:  0.9564715192706554
Mean absolute error:  25.993513216655533
Root Mean squared error:  56.834751041375036
Explained Variance Score:  0.9426841997438798
```

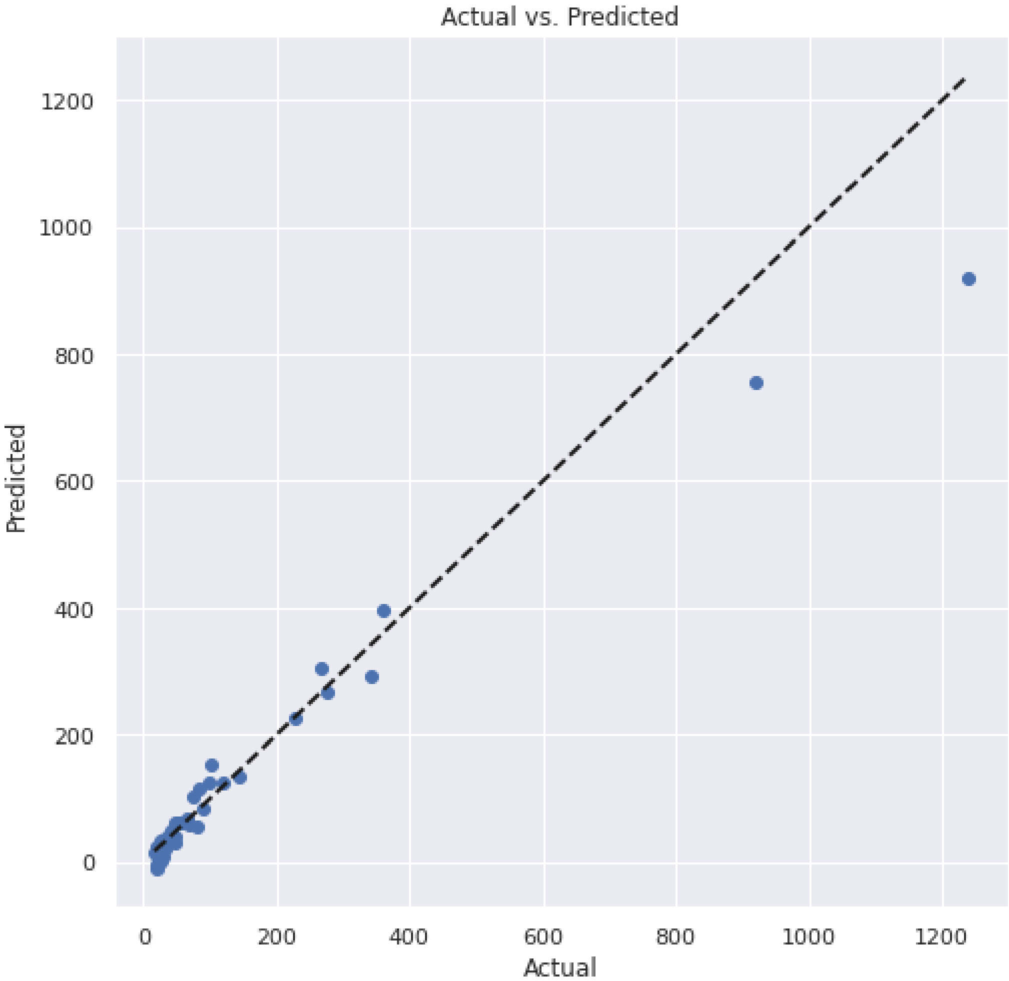




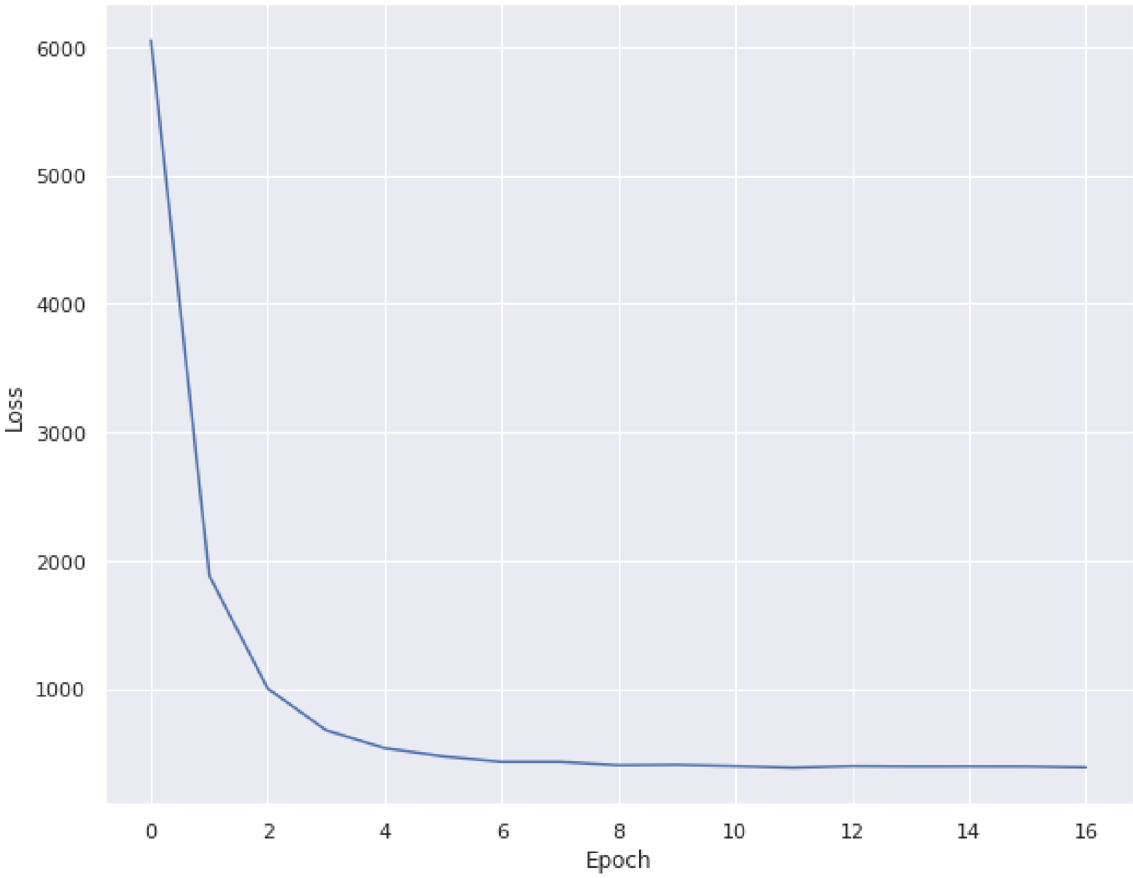
Actual vs. Predicted
Predicted
Actual
0
200
400
600
800
1000
1200

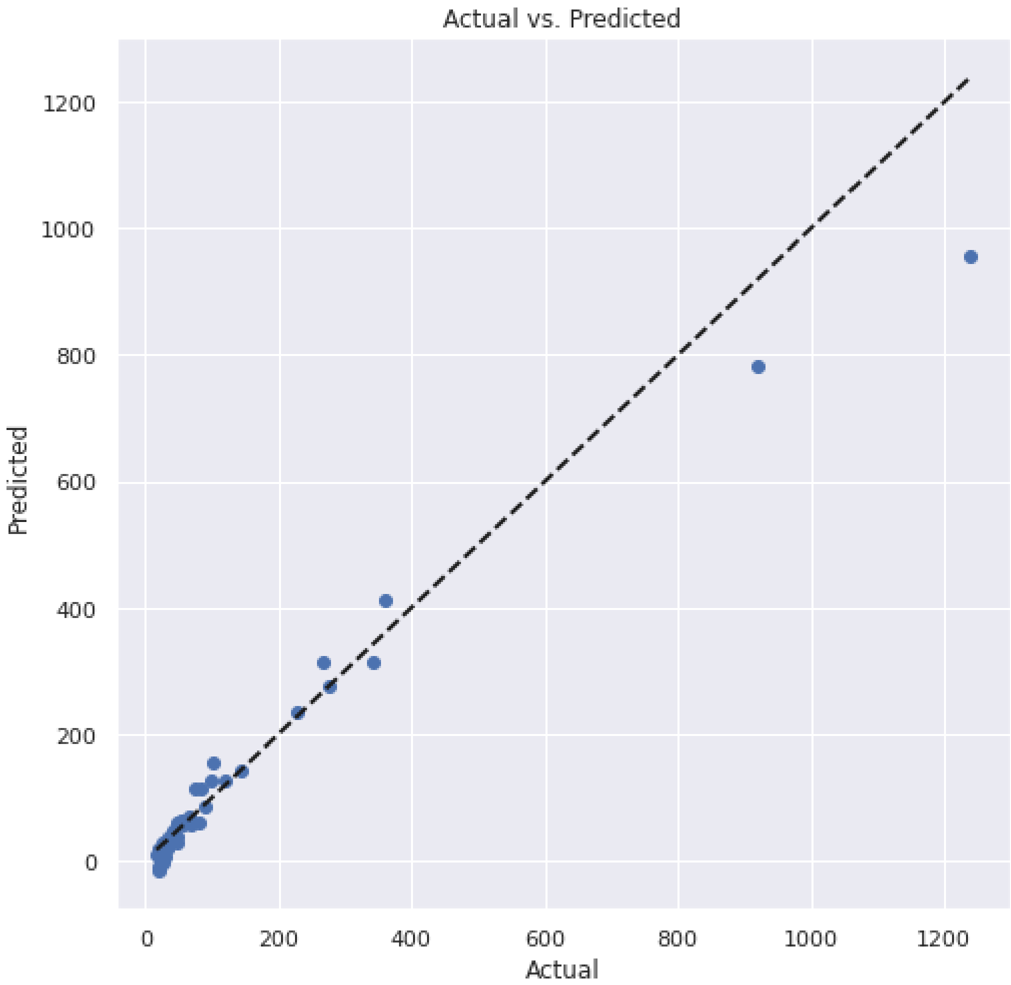
```
=============== Loss Array ===============
[6096.447596 1793.893198  925.862227  645.240837  510.88905   447.1667
  449.083601  417.887773  405.117625  406.122132  394.534524  392.78531
  393.183216  375.369149  383.105309  387.866191  379.203045  375.180851
  387.64674   364.966117  386.003181  384.360348  376.80902   384.318818
  368.103971]

For LR: 0.003, Iterations= 22500
======================================
R2 Score:  0.9564057837620915
Mean absolute error:  26.186848094822363
Root Mean squared error:  58.51796819209976
Explained Variance Score:  0.9392368141621991
```




Actual vs. Predicted
Predicted
Actual
0
200
400
600
800
1000
1200

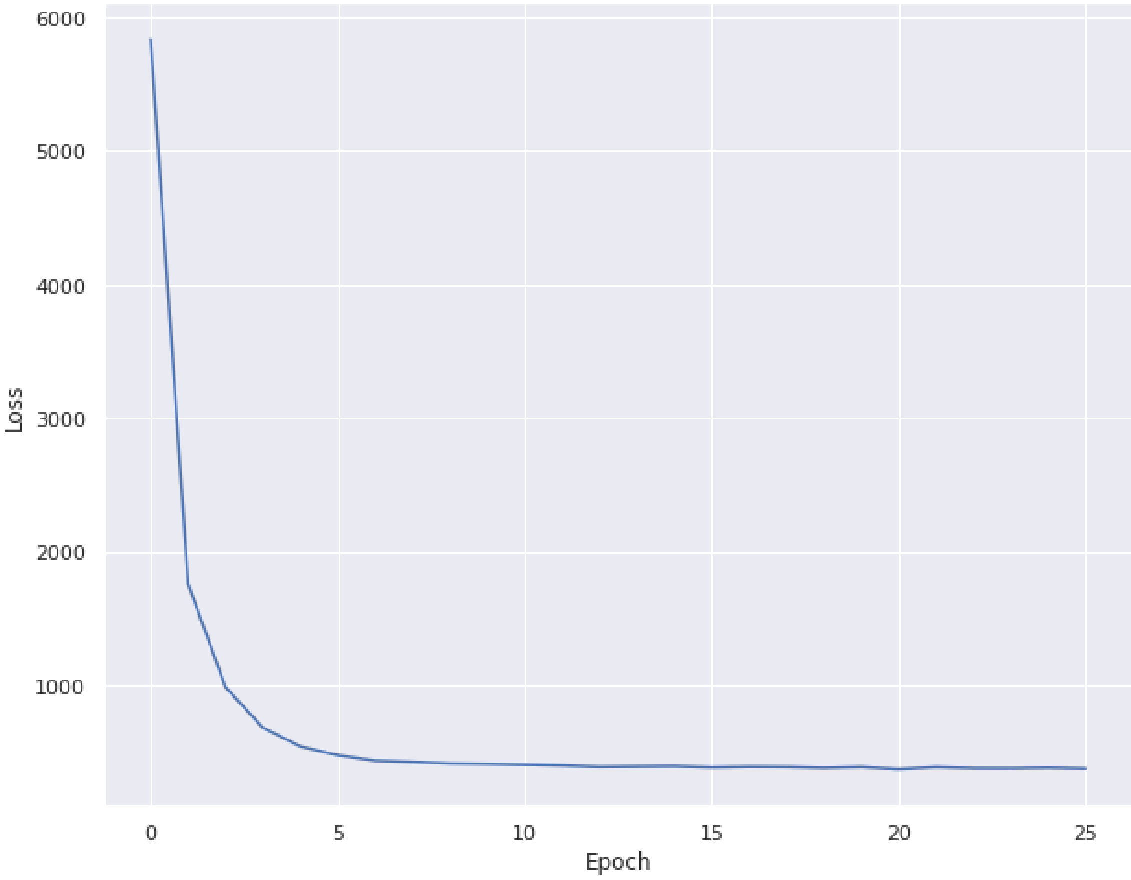
```
=============== Loss Array ===============
[6052.134523 1879.08436  1000.19496   675.163091  536.09849   472.562687
  429.830041  429.705088  402.545202  405.863005  396.354501  383.624499
  396.50898   393.883529  394.369029  393.786972  386.556191]

For LR: 0.003, Iterations= 25000
======================================
R2 Score:  0.9558656333959009
Mean absolute error:  25.684611303190415
Root Mean squared error:  52.607414218179954
Explained Variance Score:  0.9507696284538113
```

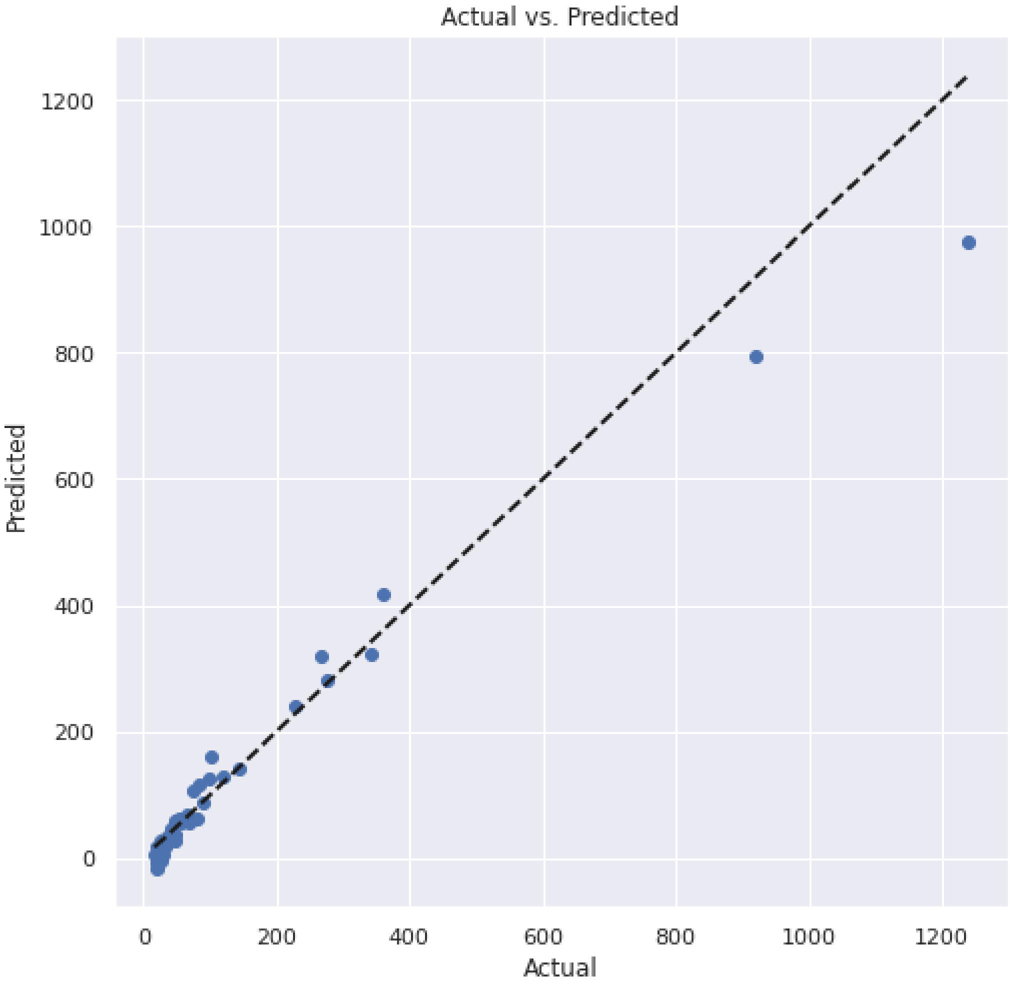




Actual vs. Predicted
Predicted
Actual
1200
1000
800
600
400
200
0
0
200
400
600
800
1000
1200

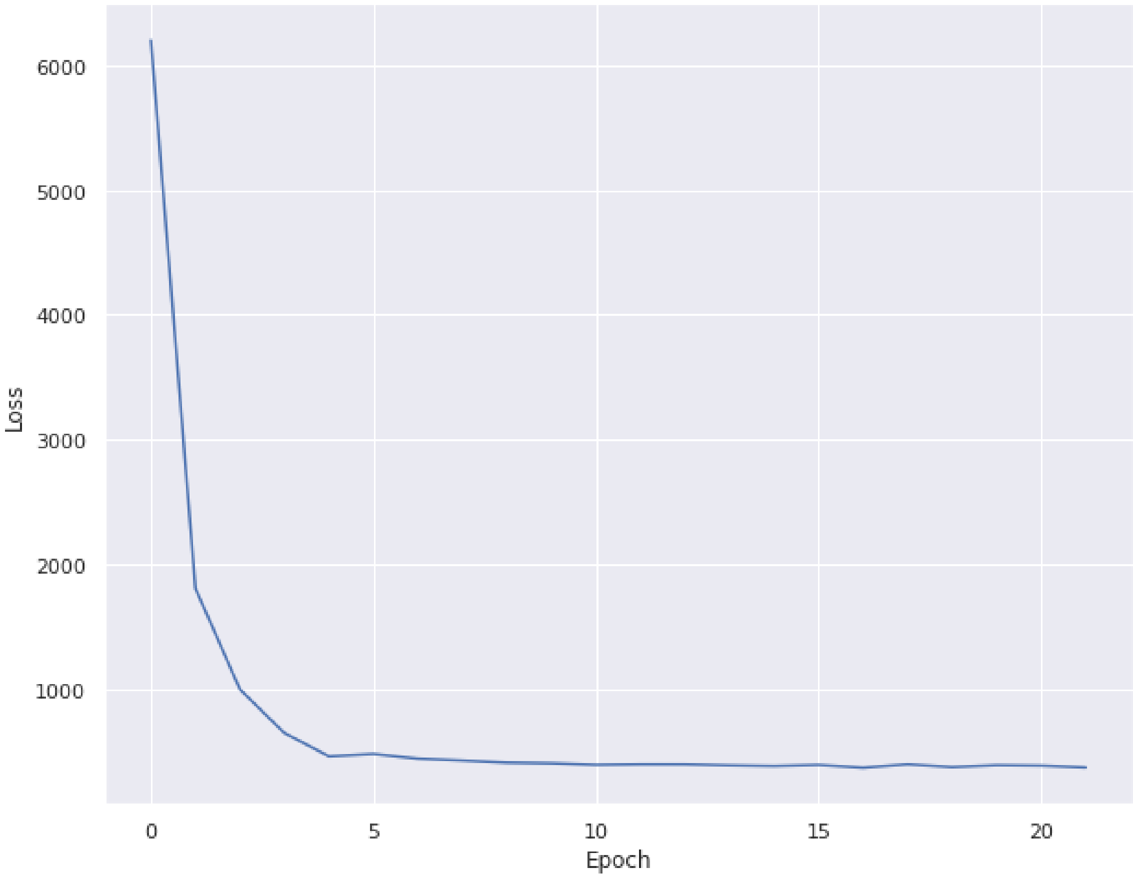
```
=============== Loss Array ===============
[5830.707788 1761.382614  986.562924  682.651564  542.682423  476.883997
  435.936778  426.66701   414.531863  411.479976  405.665293  399.532188
  389.445052  392.294487  394.601544  385.490266  390.479165  389.026362
  382.151276  388.630133  372.719143  387.928429  380.046137  379.208342
  383.043095  377.32672 ]

For LR: 0.003, Iterations= 27500
======================================
R2 Score:  0.955032806788986
Mean absolute error:  25.629971705641825
Root Mean squared error:  50.220674721027855
Explained Variance Score:  0.954925957154724
```

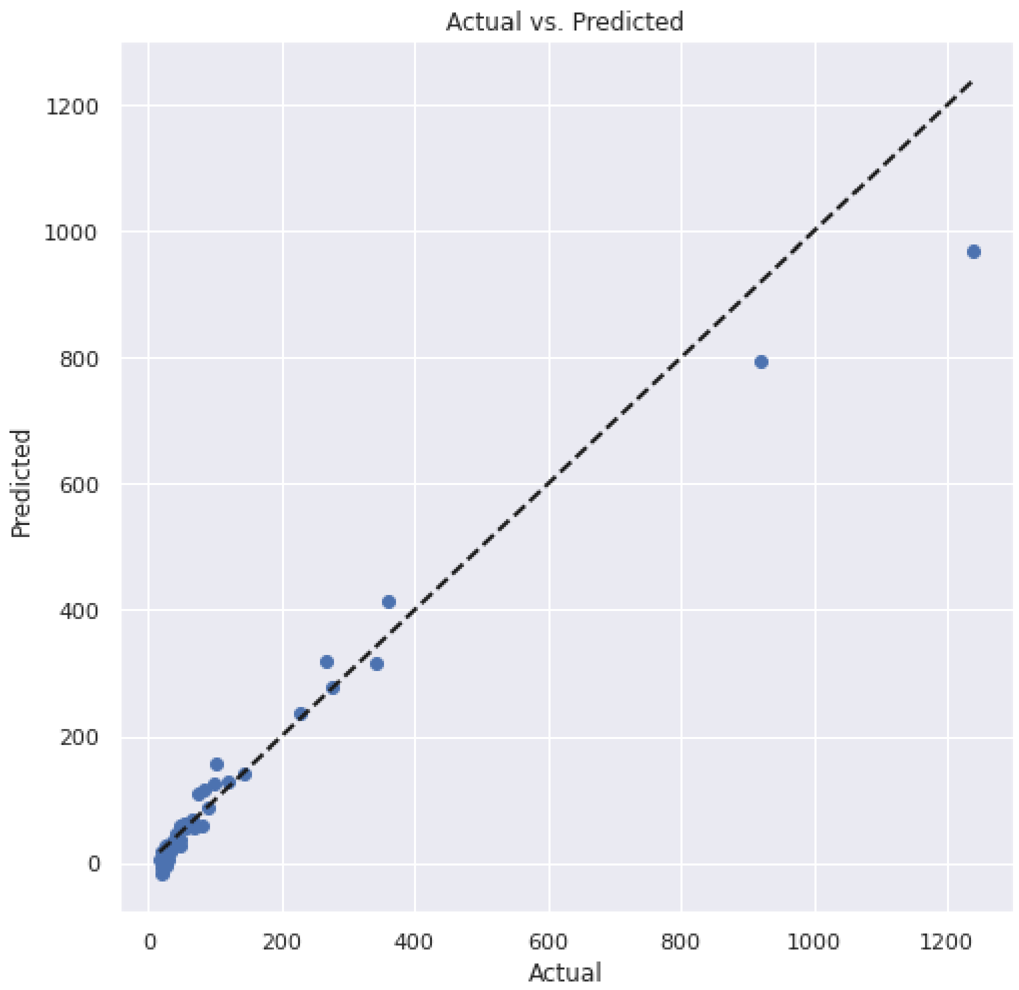




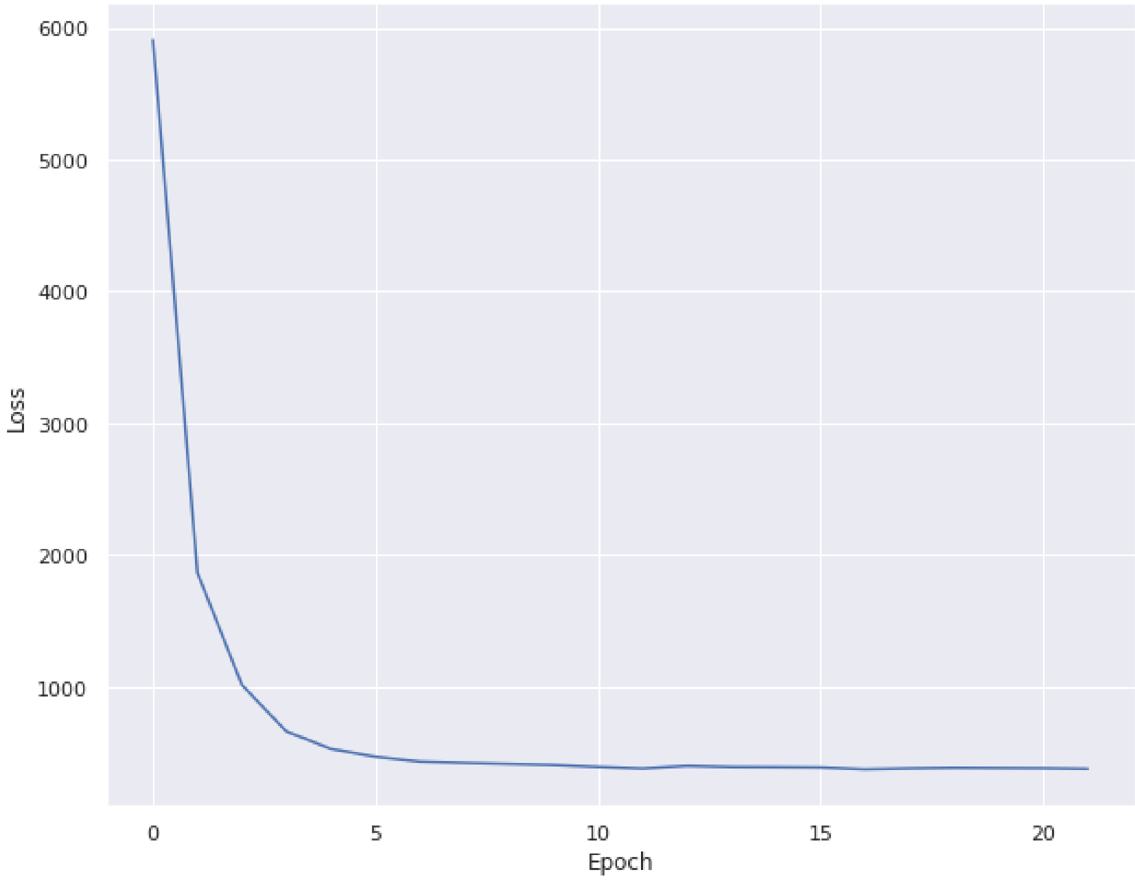
Actual vs. Predicted
1200
1000
800
600
400
200
0
Predicted
0
200
400
600
800
1000
1200
Actual

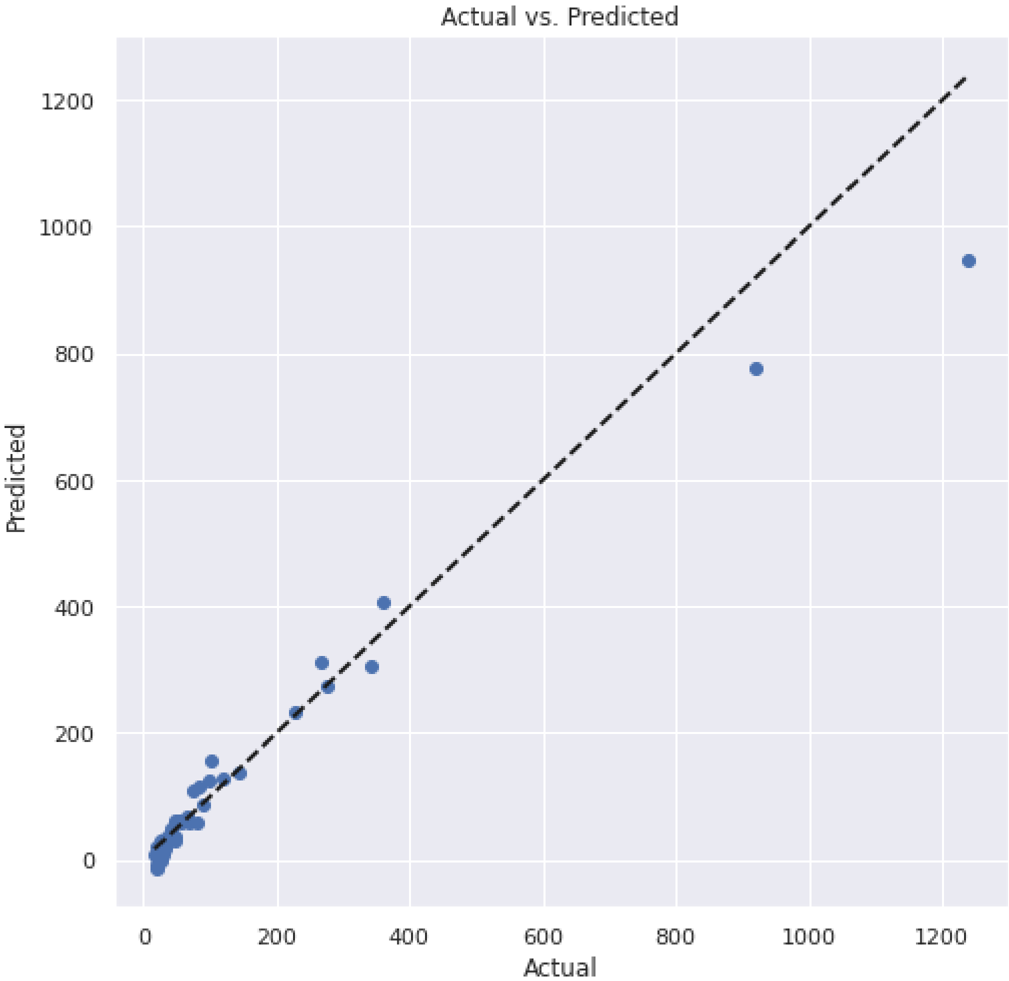
```
=============== Loss Array ===============
[6204.046307 1807.413653  996.250912  645.939958  459.477398  478.429006
  440.856977  426.138591  408.815937  403.47205   391.002035  395.347381
  395.105999  386.832274  381.584367  390.287382  368.203463  395.203078
  373.096769  388.051086  384.656754  370.029701]

For LR: 0.003, Iterations= 30000
======================================
R2 Score:  0.9552505530824827
Mean absolute error:  25.794163019174274
Root Mean squared error:  50.76103845609937
Explained Variance Score:  0.9541893934235873
```



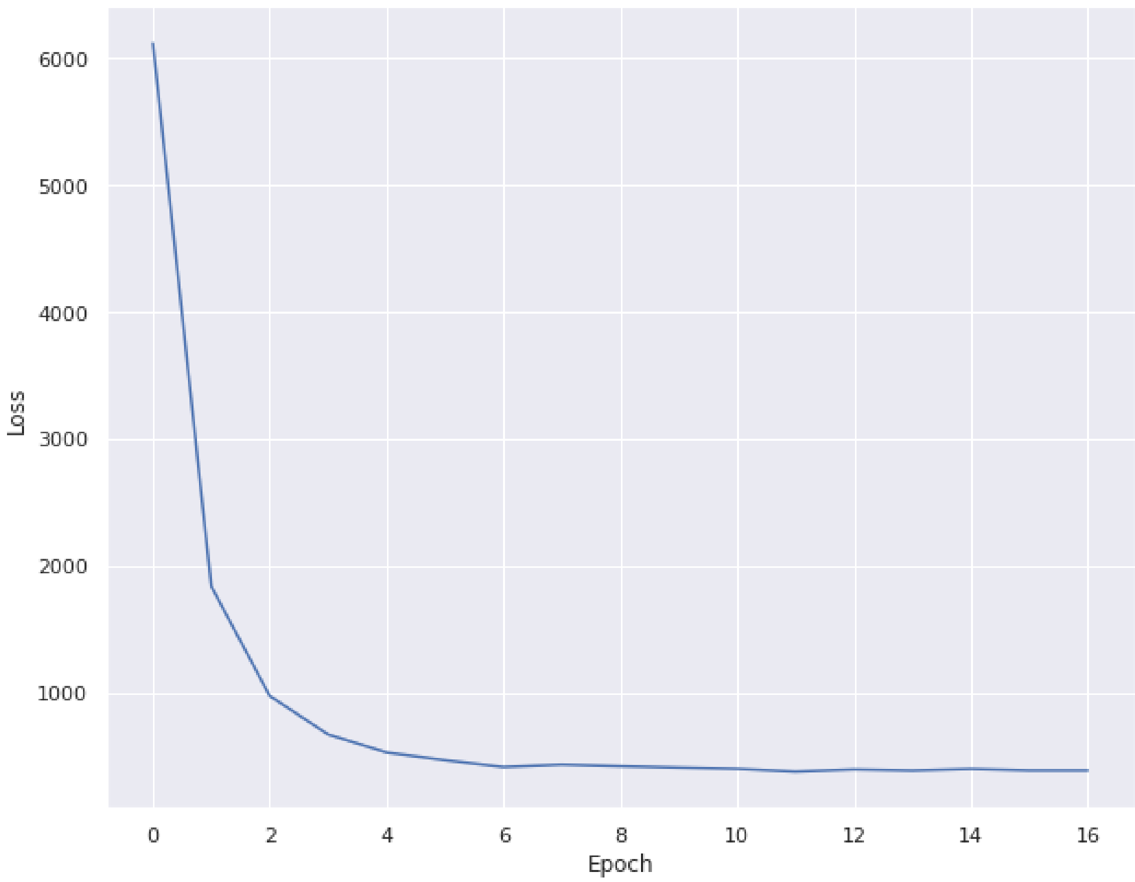

Actual vs. Predicted
Predicted
Actual
0
200
400
600
800
1000
1200

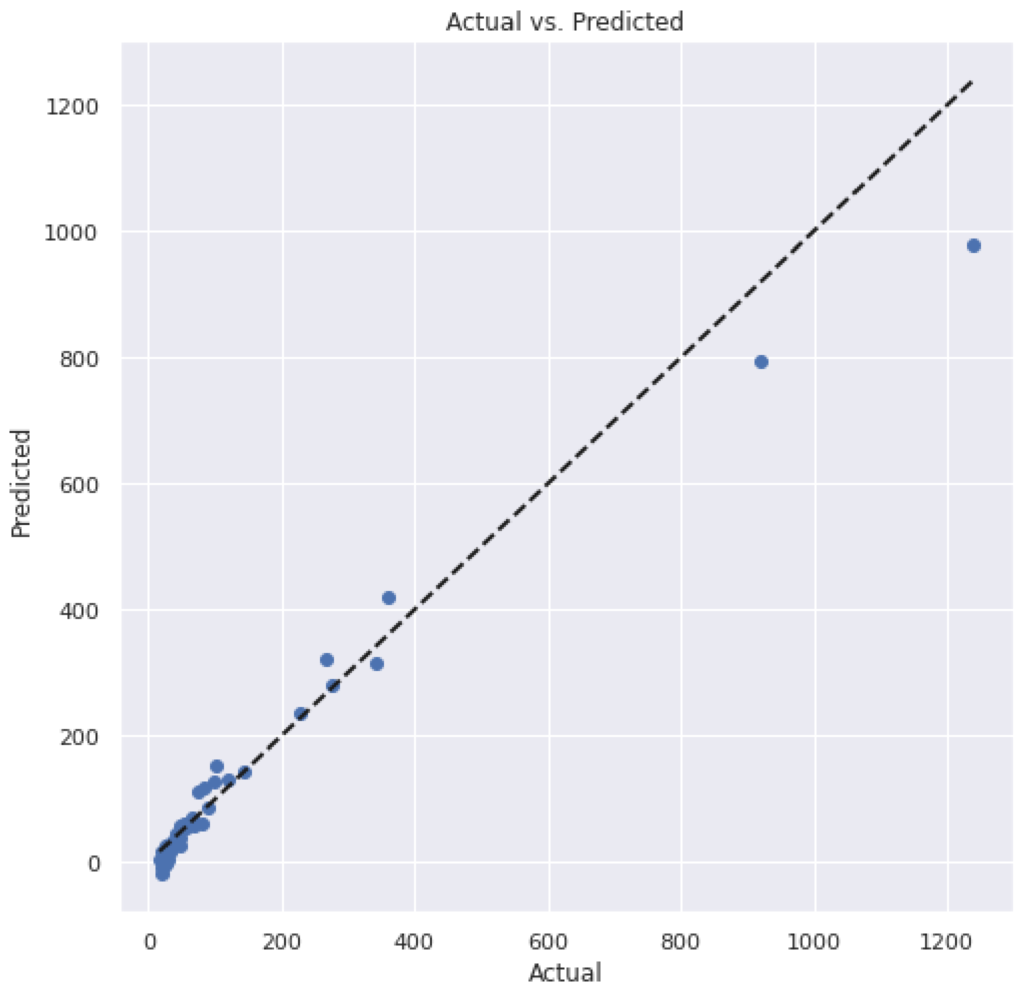
```
=============== Loss Array ===============
[5907.299189 1866.590586 1014.465163  661.908058  530.383385  470.713947
  433.128972  423.363264  415.548895  407.70089   392.888741  380.466107
  400.851326  392.307717  390.866922  388.807004  374.630622  381.517537
  385.724192  383.886231  382.382666  378.015864]

For LR: 0.003, Iterations= 32500
======================================
R2 Score:  0.9566990363408159
Mean absolute error:  25.636142042489062
Root Mean squared error:  53.82061946605013
Explained Variance Score:  0.9484475696761622
```



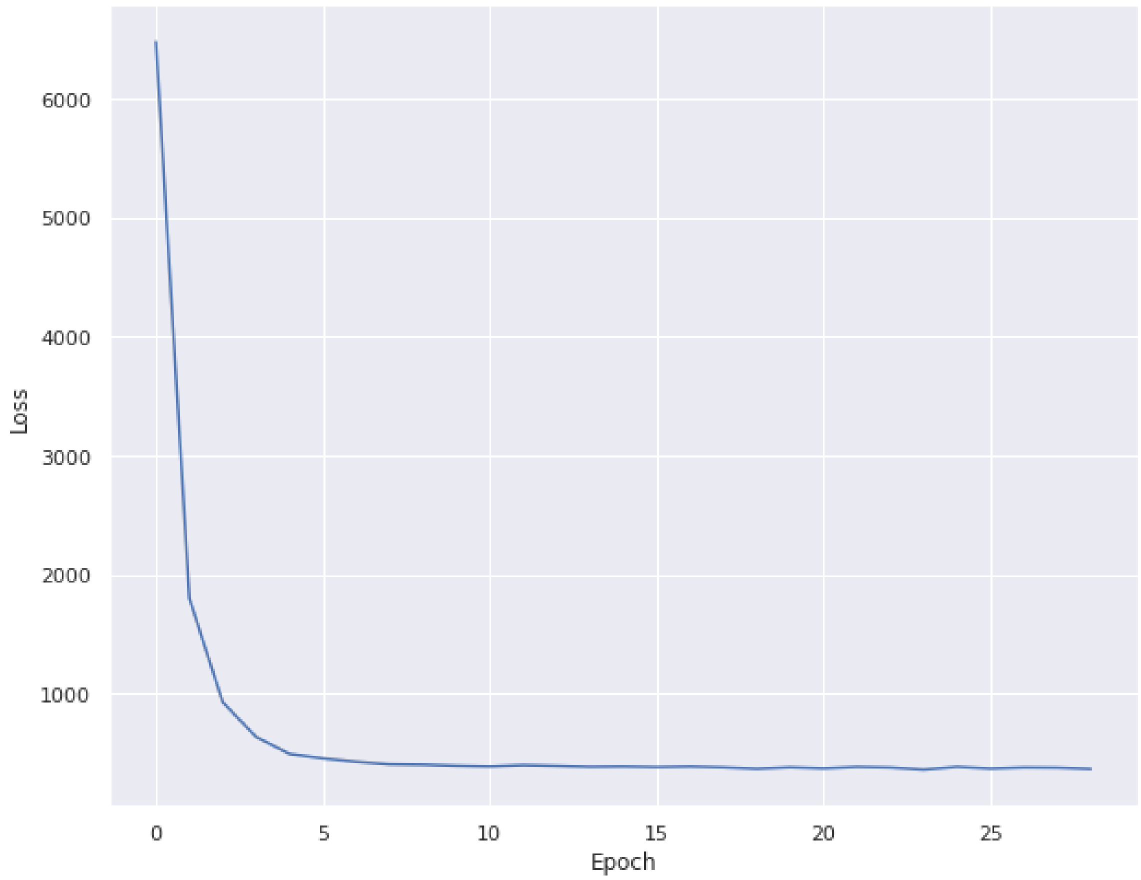

Actual vs. Predicted
1200
1000
800
600
400
200
0
Predicted
0
200
400
600
800
1000
1200
Actual

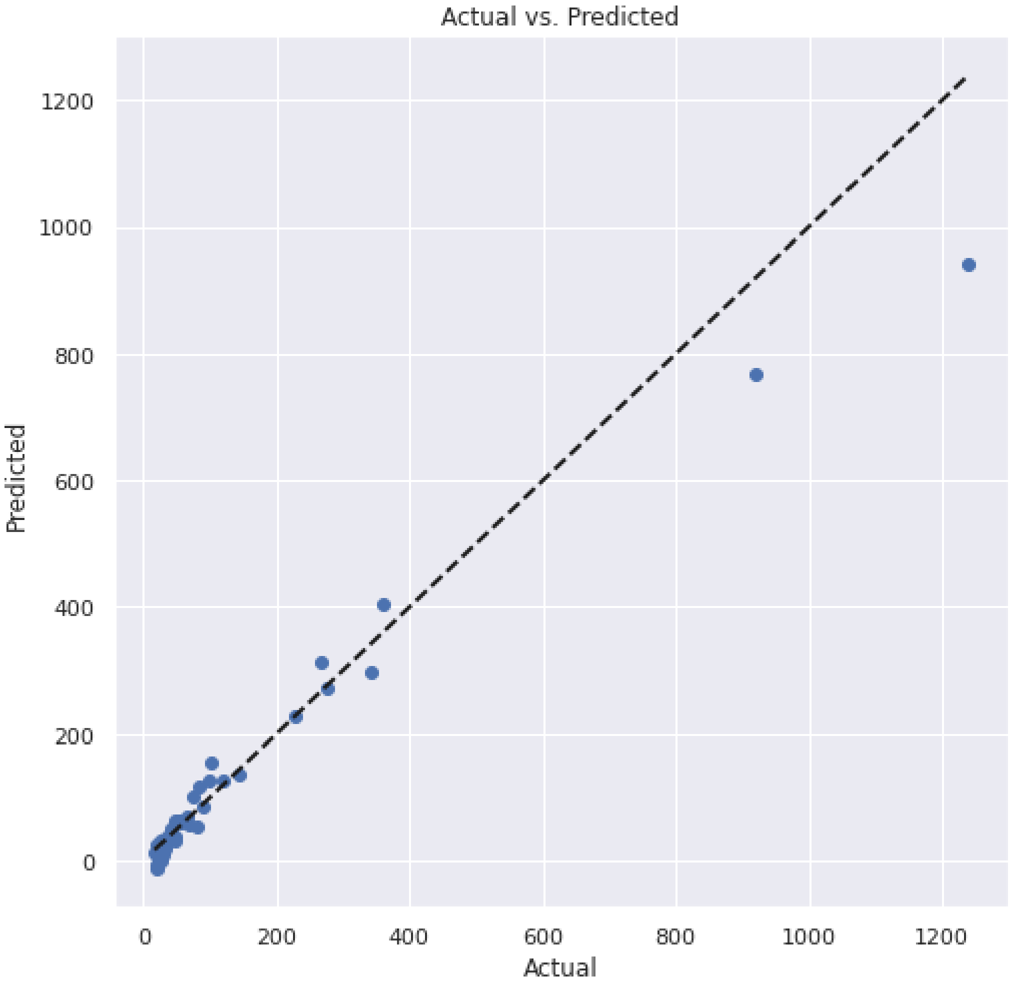
```
=============== Loss Array ===============
[6111.760357 1834.067037  972.325188  668.85642   527.17214   466.812454
  413.372797  432.490988  419.821175  408.236906  398.427521  376.929799
  393.599281  385.084264  397.565561  386.014441  386.088415]

For LR: 0.003, Iterations= 35000
======================================
R2 Score:  0.9542250437494495
Mean absolute error:  26.053640569434826
Root Mean squared error:  49.93717463993683
Explained Variance Score:  0.955671697044926
```



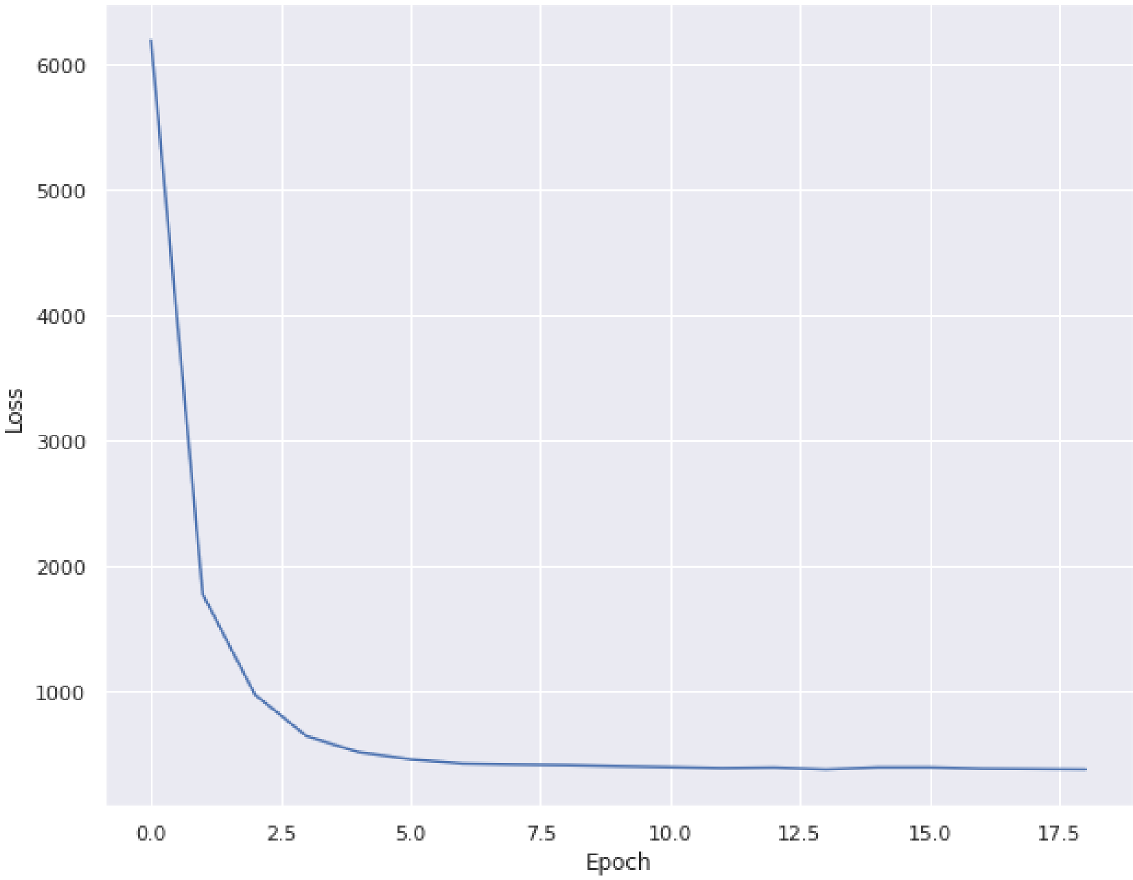

Actual vs. Predicted
Predicted
Actual
0
200
400
600
800
1000
1200

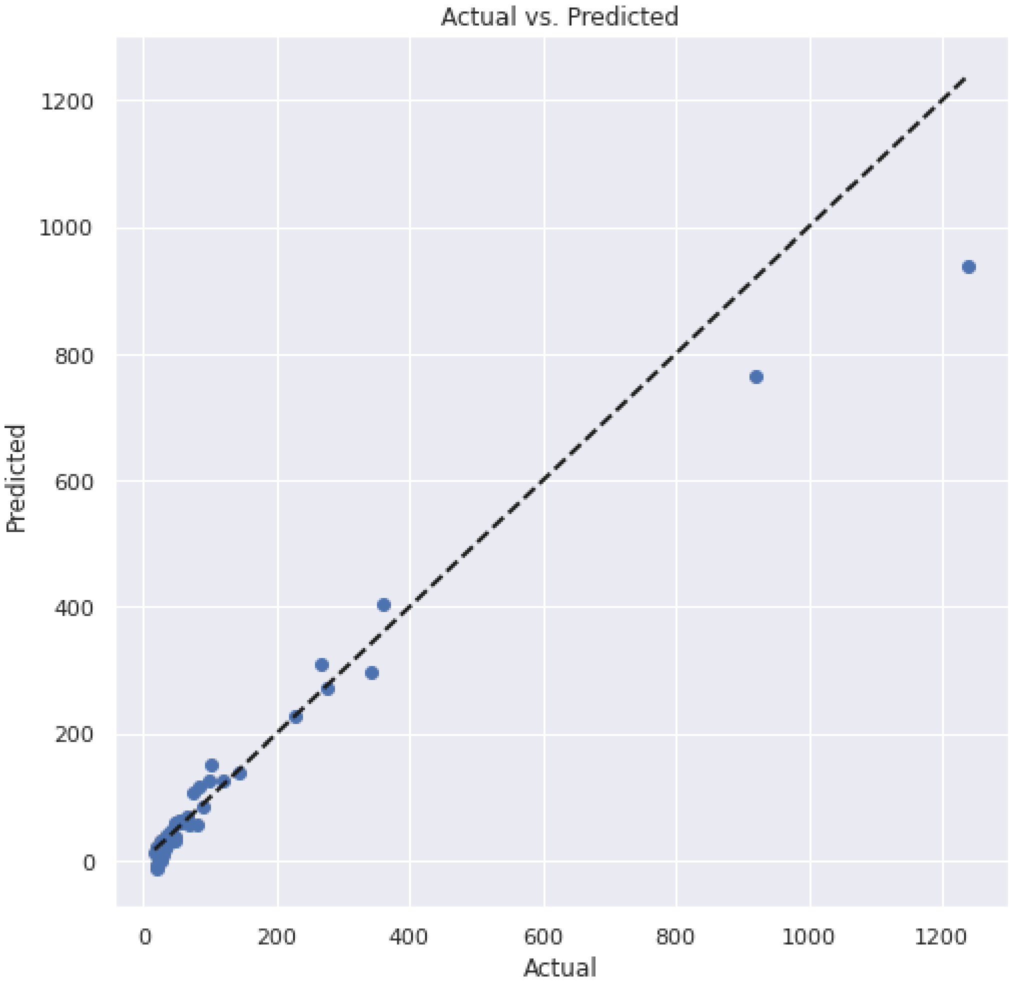
```
=============== Loss Array ===============
[6476.577702 1805.52203   930.310708  638.233898  495.32372   458.317041
  429.650098  408.76382   405.127087  395.742094  389.95153   400.908282
  394.536975  387.016521  389.288496  384.795717  389.065059  382.587836
  370.698213  383.17613   373.929683  386.340009  381.179205  363.347404
  386.931132  372.233529  381.647778  379.753876  369.400992]

For LR: 0.003, Iterations= 37500
======================================
R2 Score:  0.9570331556178285
Mean absolute error:  25.834645247715542
Root Mean squared error:  55.14027004551649
Explained Variance Score:  0.9460272566314591
```



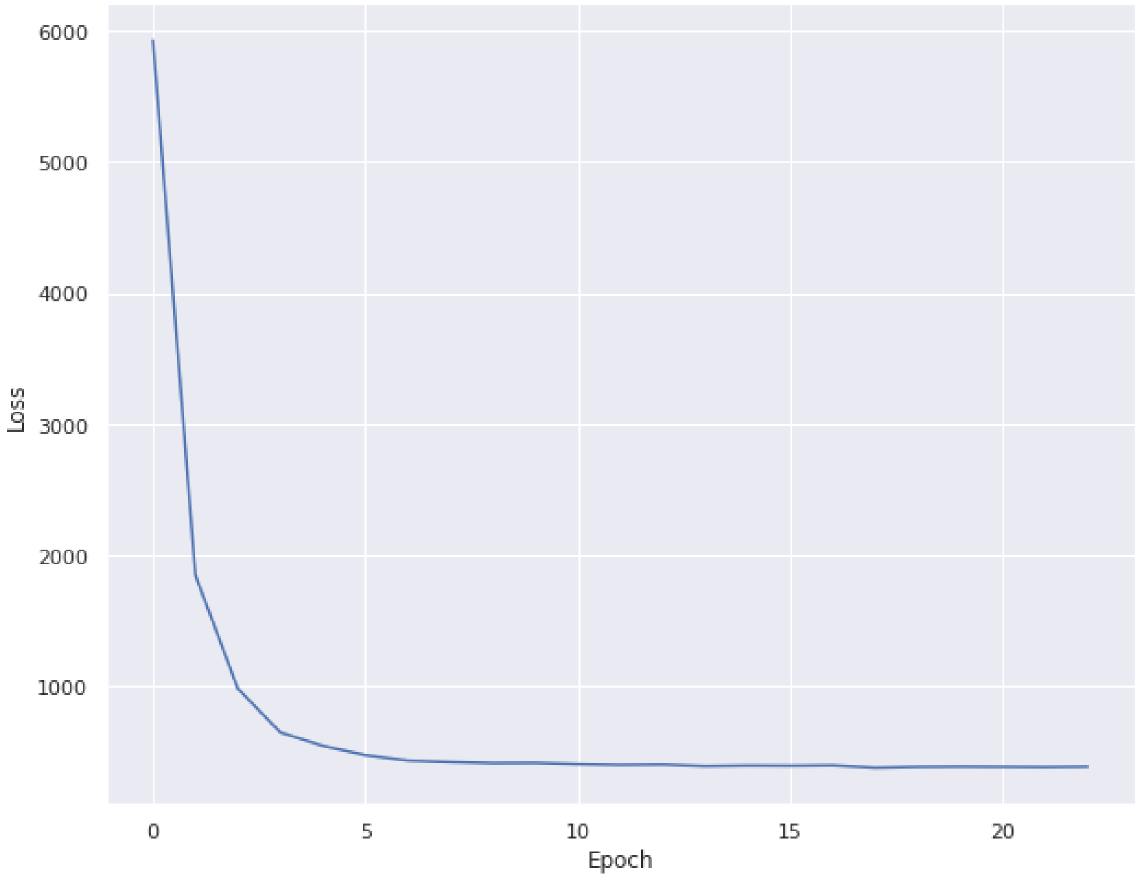

Actual vs. Predicted
Predicted
Actual
0
200
400
600
800
1000
1200

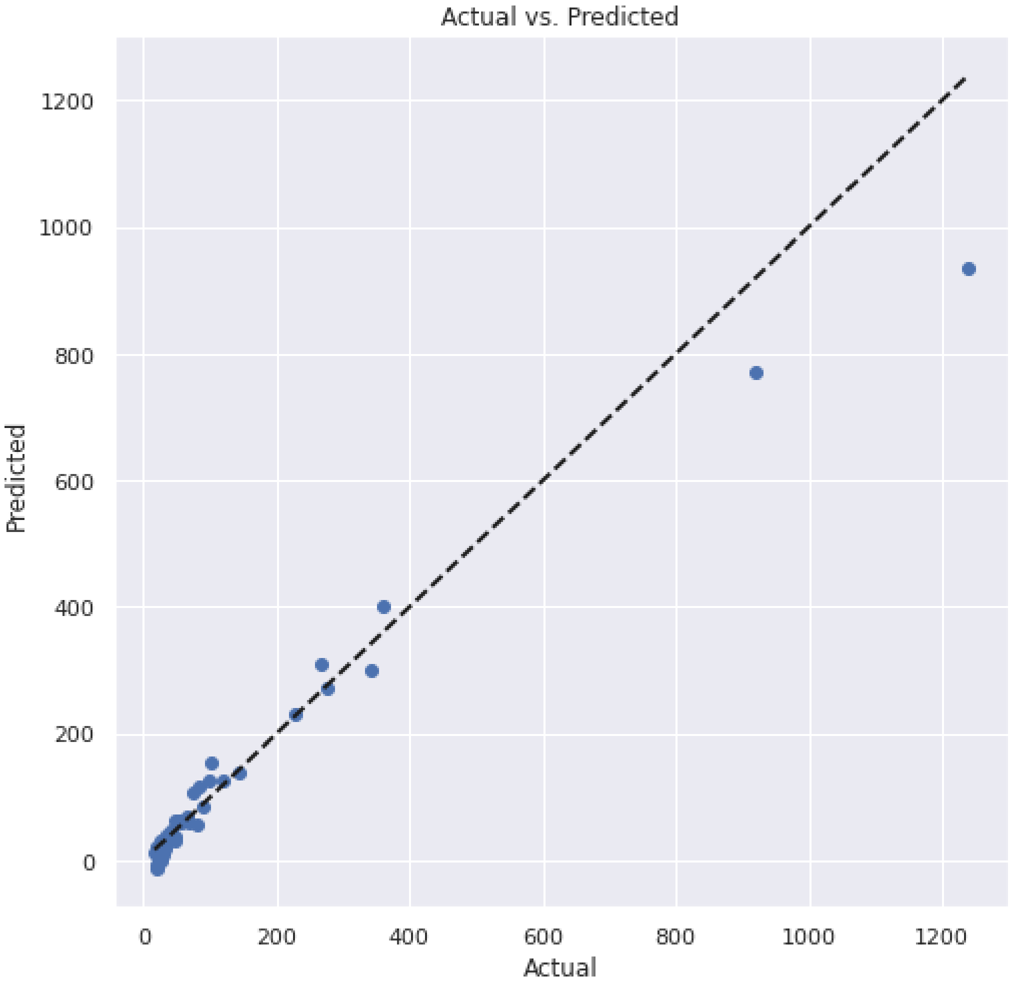
```
=============== Loss Array ===============
[6195.382774 1774.837708  977.021793  642.892604  516.470932  459.118306
  424.089397  417.104461  412.278844  403.832088  396.342753  388.106257
  392.800053  378.092448  394.414422  394.332958  384.050477  381.398468
  378.419033]

For LR: 0.003, Iterations= 40000
======================================
R2 Score:  0.9566647235221852
Mean absolute error:  25.909959523625055
Root Mean squared error:  55.9377796991934
Explained Variance Score:  0.9445006623158345
```



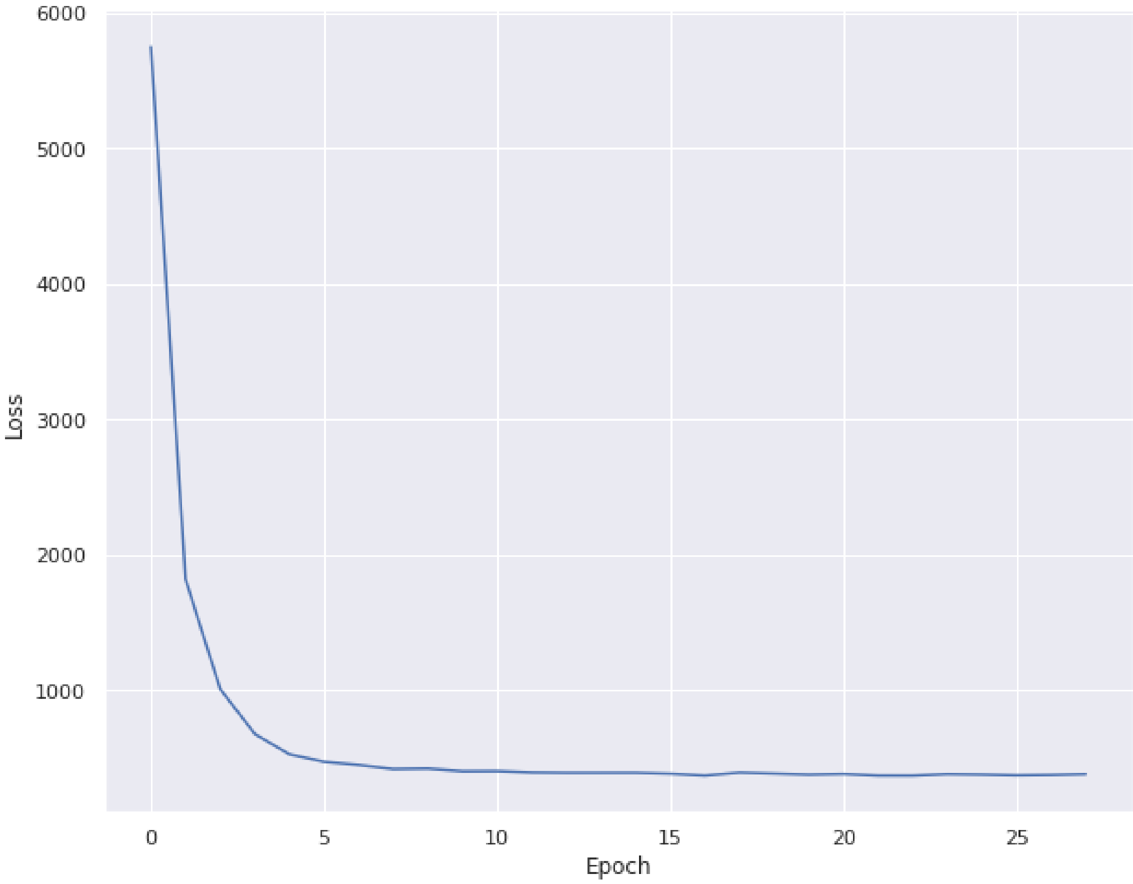

Actual vs. Predicted
Predicted
Actual
0
200
400
600
800
1000
1200

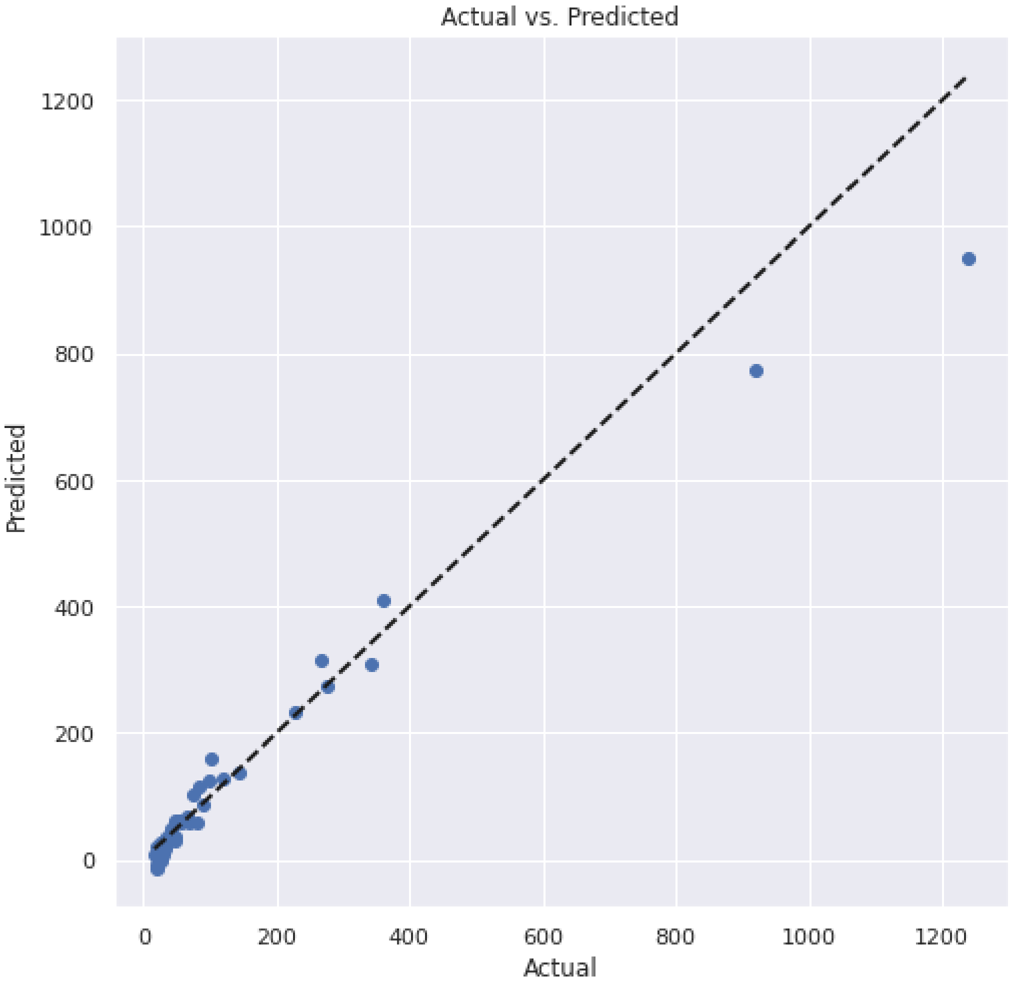
```
=============== Loss Array ===============
[5931.948851 1845.411816  977.159554  643.605537  540.156043  469.747342
  428.68818   417.530246  408.9001    409.967692  400.856153  394.931902
  397.661046  386.079199  390.098957  388.776689  391.780314  373.786714
  380.310676  381.970939  380.806399  378.914404  381.818643]

For LR: 0.003, Iterations= 42500
======================================
R2 Score:  0.9568491101605828
Mean absolute error:  25.860204125493688
Root Mean squared error:  55.64029768059053
Explained Variance Score:  0.9449354691593166
```




Actual vs. Predicted
Predicted
Actual
0
200
400
600
800
1000
1200

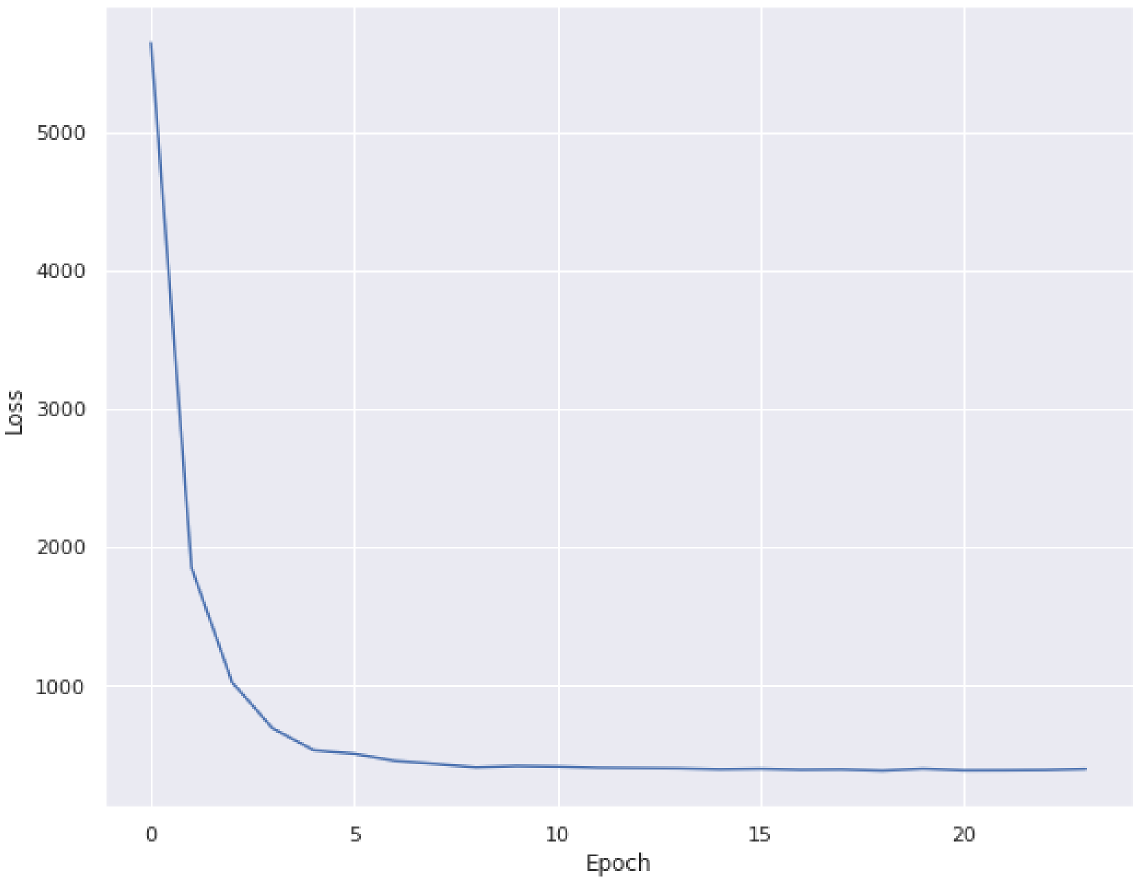
```
=============== Loss Array ===============
[5747.831681 1822.128942 1012.042009  680.195172  530.261813  474.657204
  451.248602  421.104639  424.55032   405.514453  406.387654  395.74894
  393.49897   393.582813  393.722083  387.257962  373.883022  395.140613
  388.265062  380.852024  385.449055  373.797278  373.515822  384.382492
  381.300611  375.863974  378.559833  384.2538  ]

For LR: 0.003, Iterations= 45000
======================================
R2 Score:  0.9567899977652793
Mean absolute error:  25.655154853845044
Root Mean squared error:  53.71324042685213
Explained Variance Score:  0.9487351288265826
```

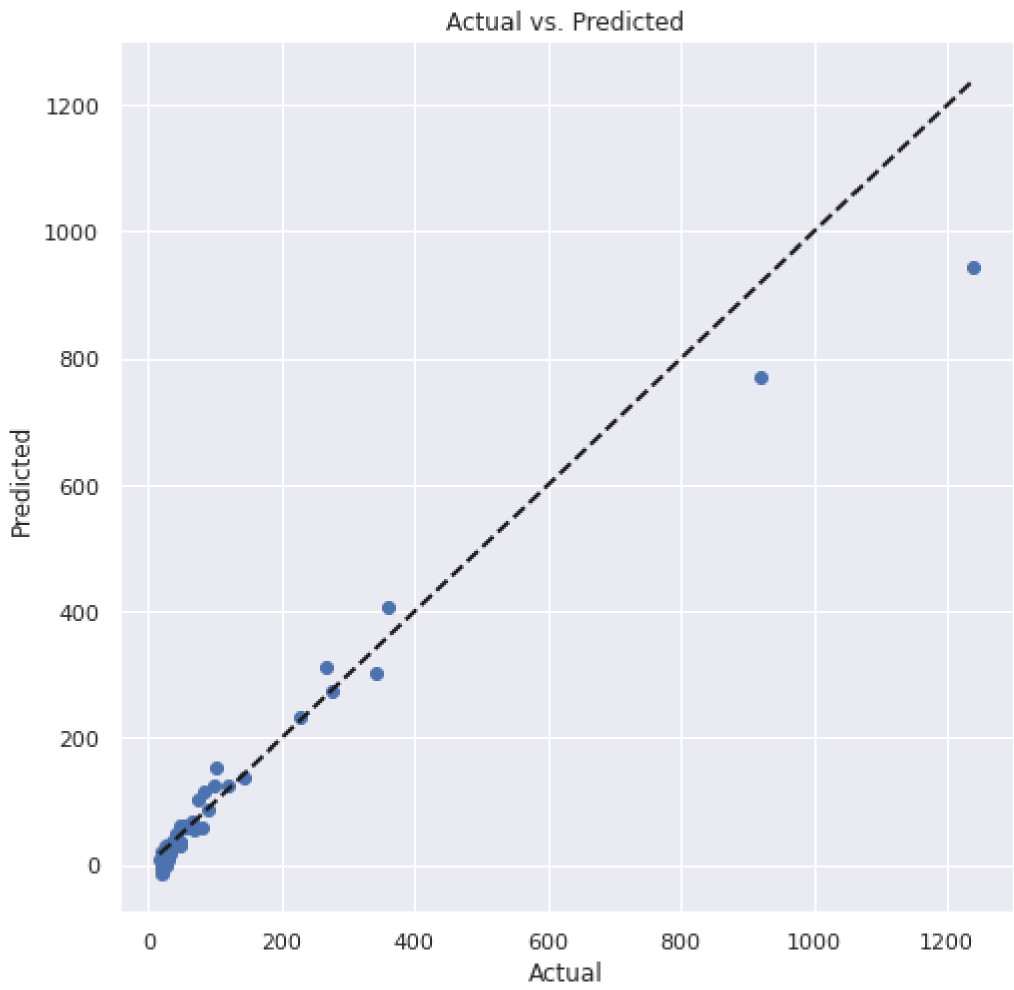




Actual vs. Predicted
Predicted
Actual
1200
1000
800
600
400
200
0
0
200
400
600
800
1000
1200

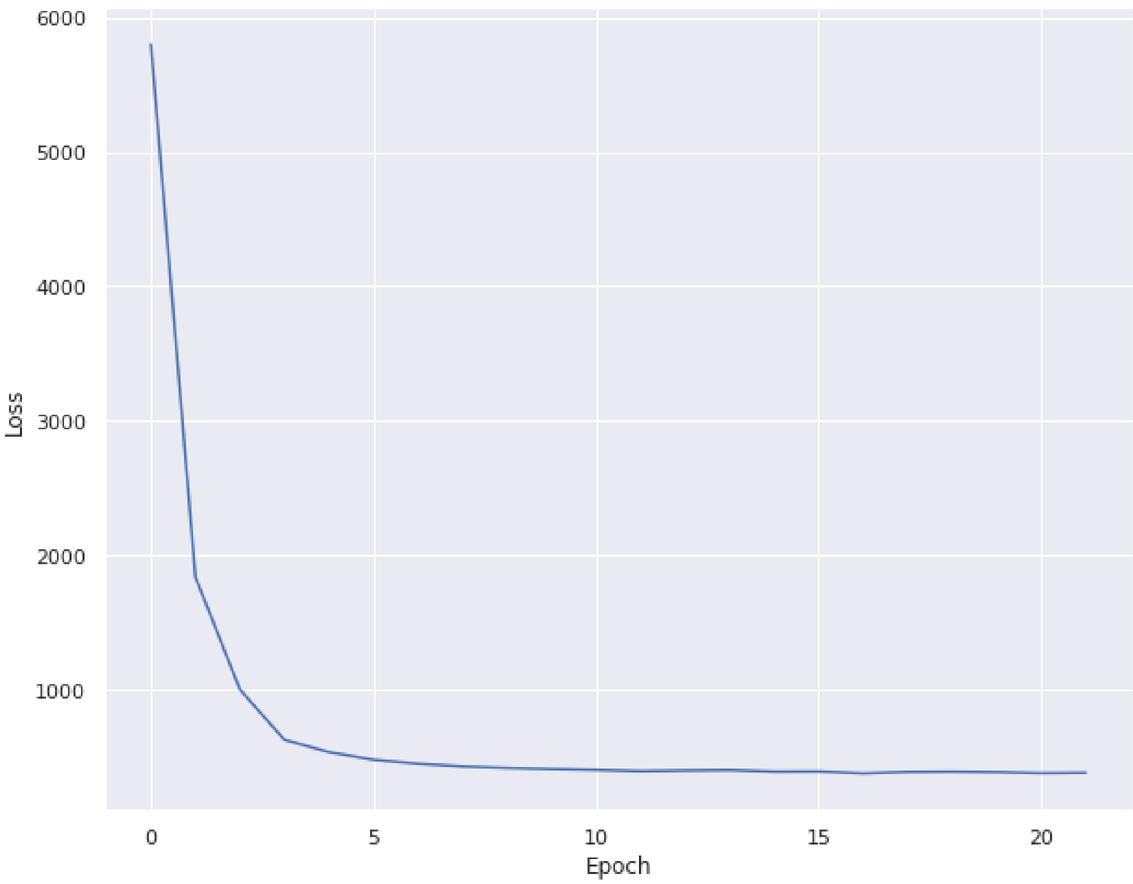
```
=============== Loss Array ===============
[5645.207909 1846.200446 1014.940983  681.437753  524.634897  498.519746
  447.692104  425.900404  400.155864  410.144059  405.674843  396.981665
  394.881354  392.787651  385.909442  389.79229   382.666516  385.088456
  376.459122  390.379087  378.570756  379.207998  381.134206  388.452244]

For LR: 0.003, Iterations= 47500
======================================
R2 Score:  0.9568829099720658
Mean absolute error:  25.720574961774712
Root Mean squared error:  54.52494263400941
Explained Variance Score:  0.947204113484203
```

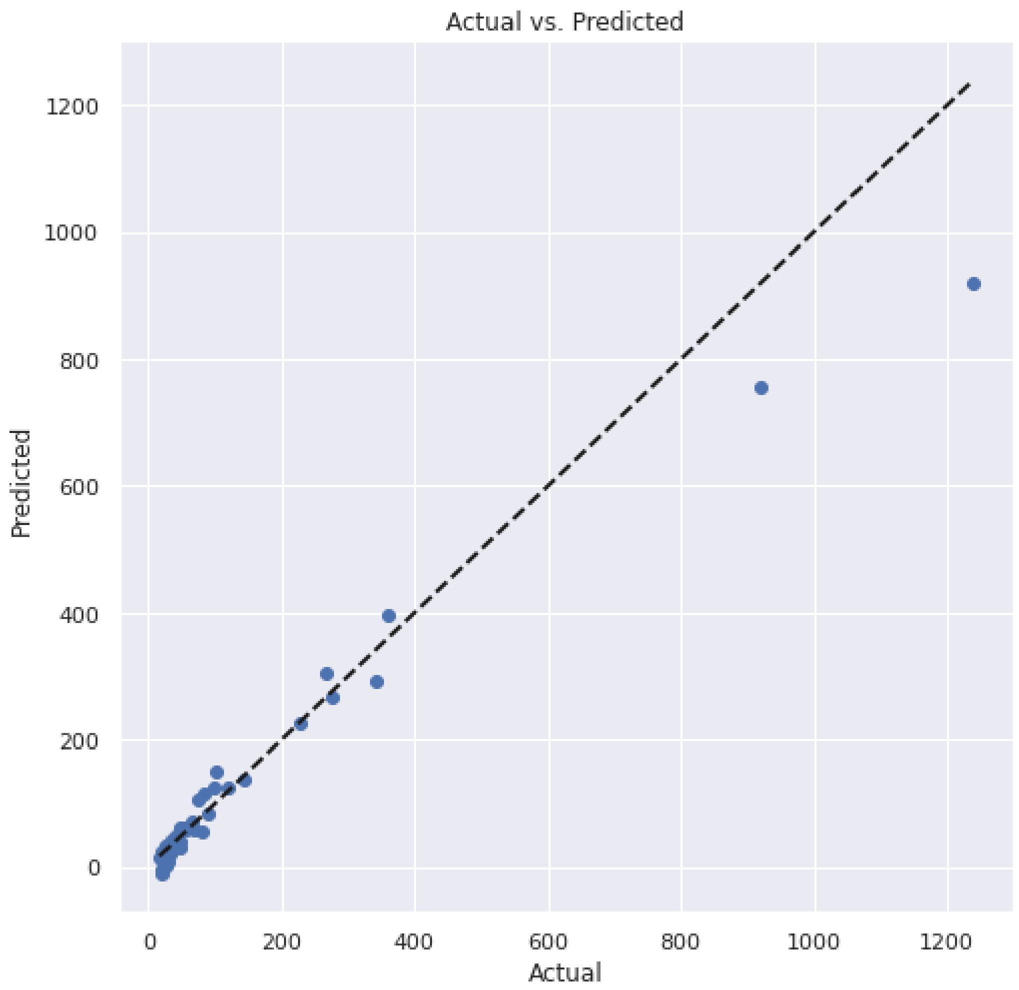




Actual vs. Predicted
Predicted
Actual
0
200
400
600
800
1000
1200

```
=============== Loss Array ===============
[5800.290095 1837.393835  999.105939  625.432006  534.837653  477.529847
  447.712456  427.209632  415.696279  407.798011  401.250287  391.814045
  396.699091  399.994286  388.094605  389.368516  375.399397  385.706252
  387.717773  384.510674  376.798202  379.419198]

For LR: 0.003, Iterations= 50000
======================================
R2 Score:  0.9562520712689068
Mean absolute error:  26.1479768005052
Root Mean squared error:  58.4076909011371
Explained Variance Score:  0.9393890313013623
```





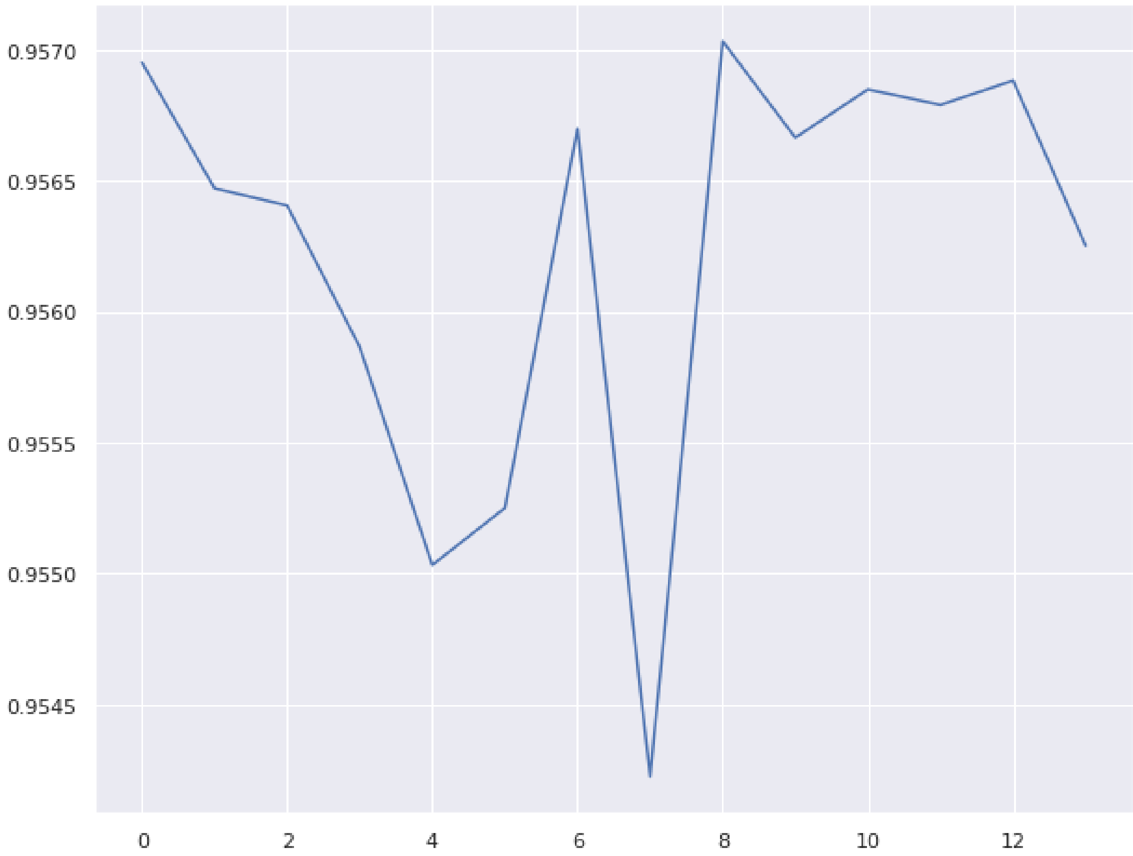
Actual vs. Predicted
1200
1000
800
600
400
200
0
Predicted
0
200
400
600
800
1000
1200
Actual

In [44]:

```
plt.plot(r2_lst)
```

Out[44]:

```
[<matplotlib.lines.Line2D at 0x7f9232205f10>]
```



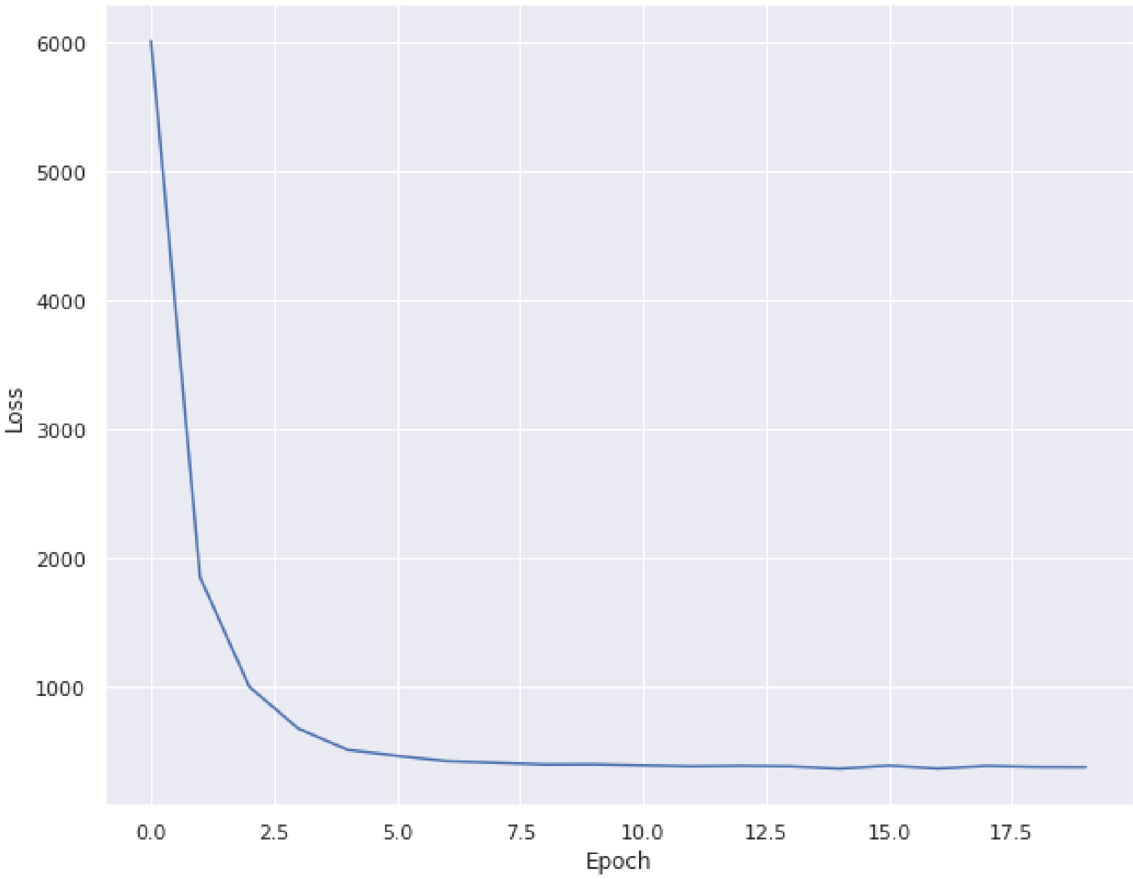
R2 score fluctuates as seen from the above plot

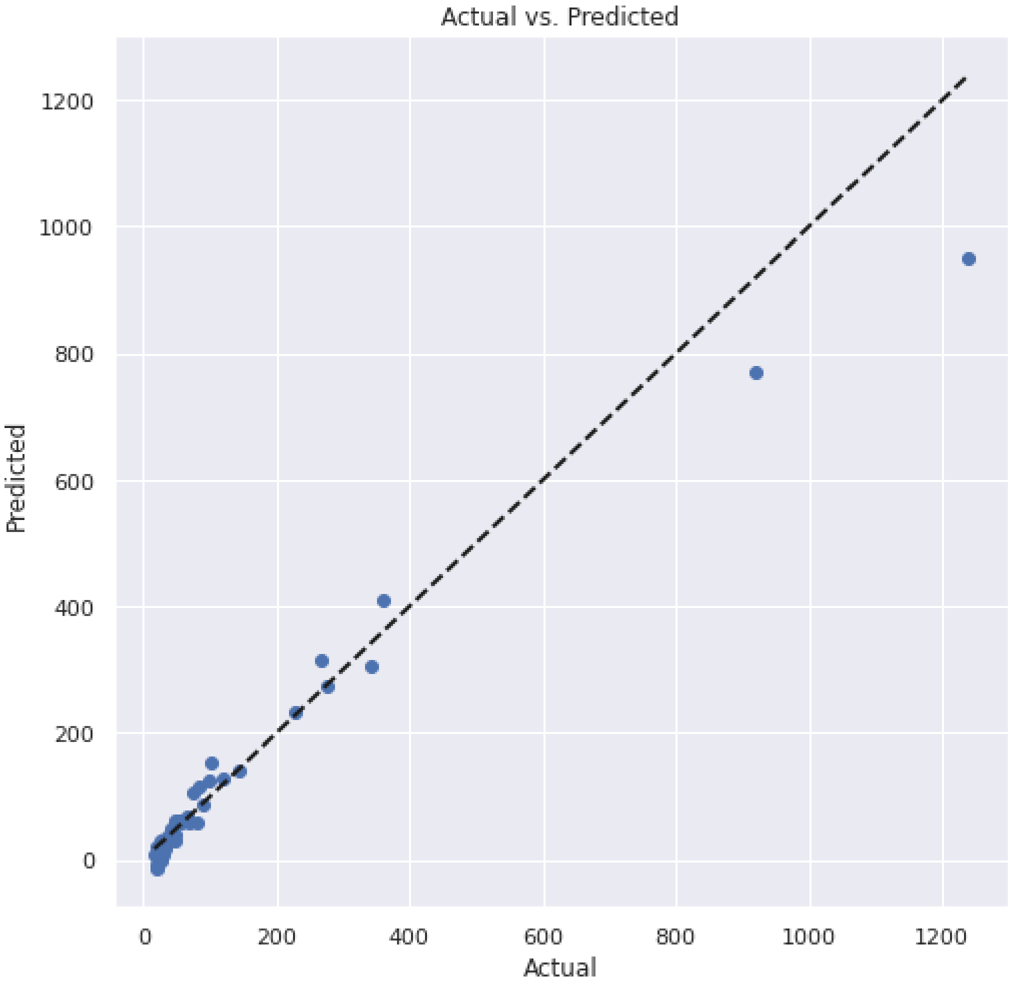
Now we try altering learning rate while keeping iterations same

In [45]:

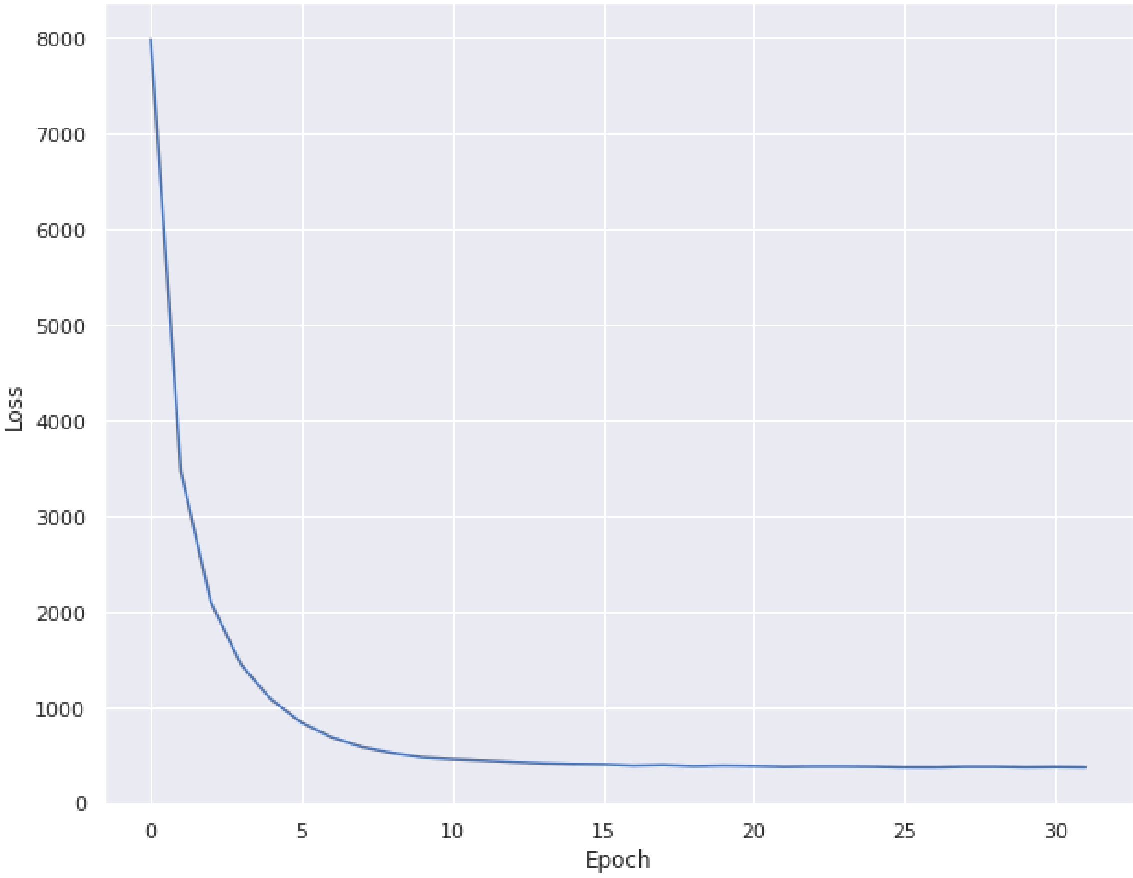
```
lr = 0.003
itrs=15000
r2_lst = []

while lr >= 0.0001:
    model=SGDRegressor(learning_rate='constant', eta0=lr, max_iter=itrs, verbose=1)

    with DisplayLossCurve(print_loss=True):
        model.fit(X_train_sc,y_train)

    y_pred=model.predict(X_test_sc)

    r2 = model.score(X_train_sc,y_train)
    mae = mean_absolute_error(y_test, y_pred)
    rmse = mean_squared_error(y_test, y_pred, squared=False)
    evs = explained_variance_score(y_test, y_pred)

    r2_lst.append(r2)

    print()
    print("For LR: "+str(lr)+", Iterations= "+str(itrs))
    print("======================================")
    print("R2 Score: ", r2)
    print("Mean absolute error: ", mae)
    print("Root Mean squared error: ", rmse)
    print("Explained Variance Score: ", evs)

    file = open("Scikit_SGD_log.txt","a")
    file.write("LR = " + str(lr) + ",max_iterations = " + str(itrs) + " R^2 = "
    + str(r2) + ", MAE = " + str(mae) + ", RMSE = " + str(rmse) +
    ",  Explained-Variance = " + str(evs) + " \n")
    file.close()

    plt.figure(figsize = (8,8))
    plt.scatter(y_test, y_pred)
    plt.plot([y_test.min(),y_test.max()],[y_test.min(),y_test.max()],'k--',lw=2)
    plt.xlabel('Actual')
    plt.ylabel('Predicted')
    plt.title('Actual vs. Predicted')
    plt.show()

    r2_lst.append(np.around(r2,2))
    lr/=2
```

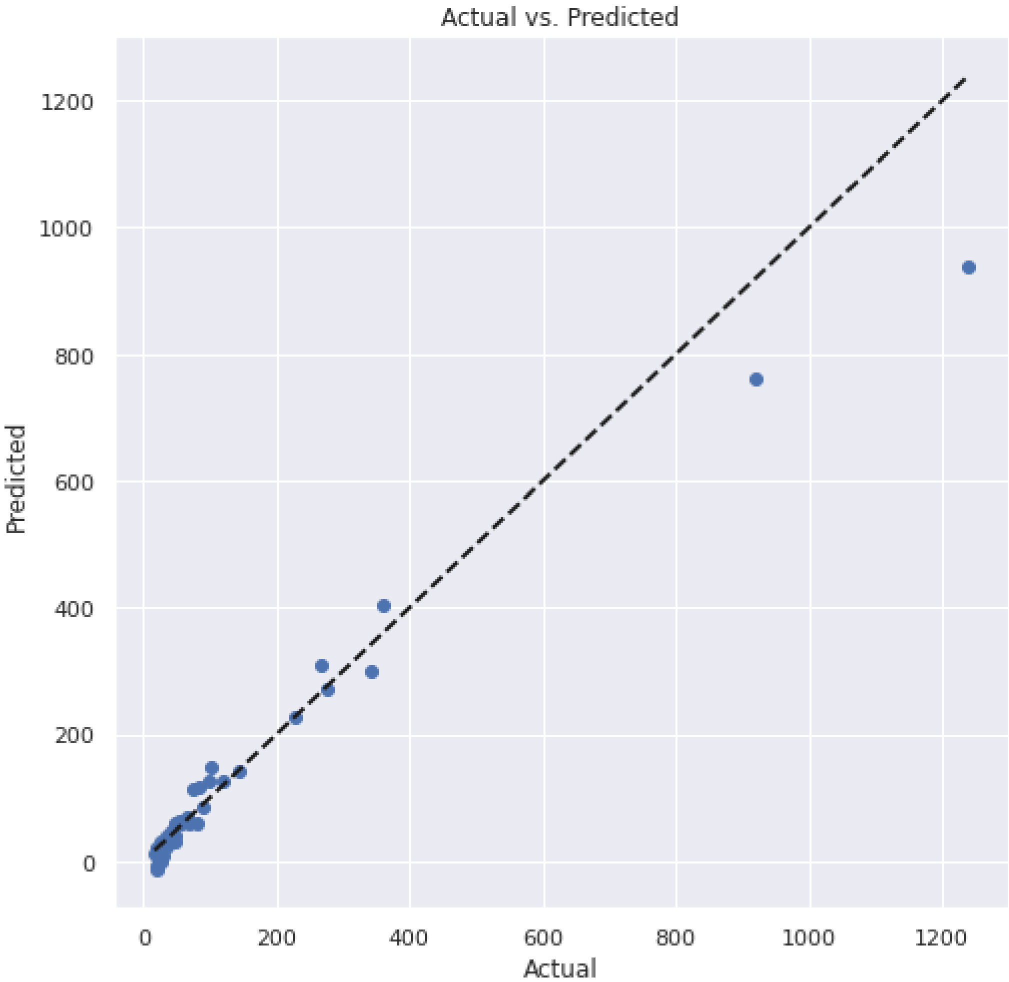
```
=============== Loss Array ===============
[6015.146368 1854.920047 1005.66227   680.52103   516.332742  470.87379
  429.795186  417.463099  403.323333  404.499752  396.304011  388.507066
  392.880095  388.668367  371.00475   394.635587  371.99243   392.242392
  384.02059   382.682865]

For LR: 0.003, Iterations= 15000
======================================
R2 Score:  0.9566408256712335
Mean absolute error:  25.697409320563235
Root Mean squared error:  54.13518275609155
Explained Variance Score:  0.9478915269635809
```



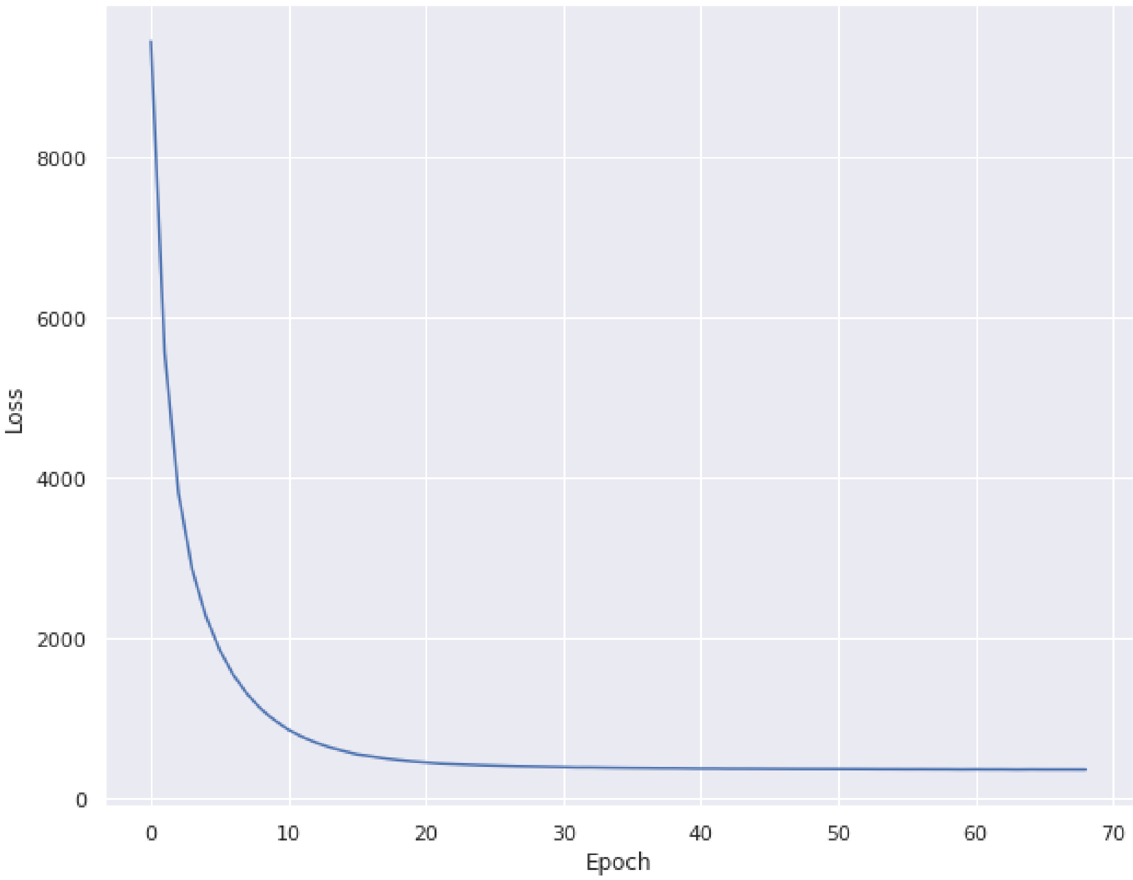


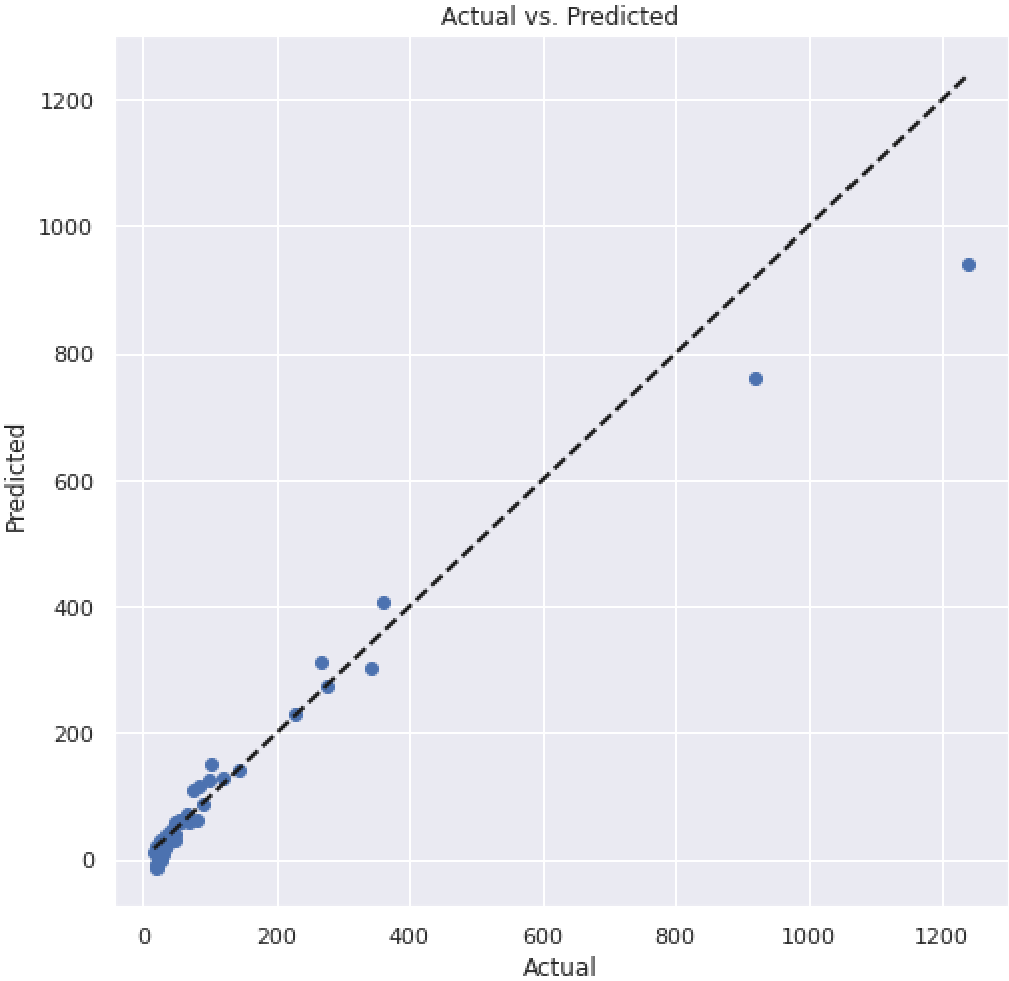

```
=============== Loss Array ===============
[7980.253836 3466.44824  2092.961973 1441.982482 1076.956843  836.690396
  683.201      582.687852  520.814663  474.05785   454.786814  440.190777
  424.445419  410.864052  402.683016  399.44356   386.715796  393.249806
  381.369186  387.279186  383.102059  376.41361   379.660507  379.521856
  376.762057  368.909093  368.452196  376.783579  376.634316  370.026732
  373.02014   369.639408]

For LR: 0.0015, Iterations= 15000
======================================
R2 Score:  0.9562639207774449
Mean absolute error:  25.77364799237426
Root Mean squared error:  56.027133351783746
Explained Variance Score:  0.9442619610234873
```


Loss
Epoch
8000
7000
6000
5000
4000
3000
2000
1000
0
0
5
10
15
20
25
30

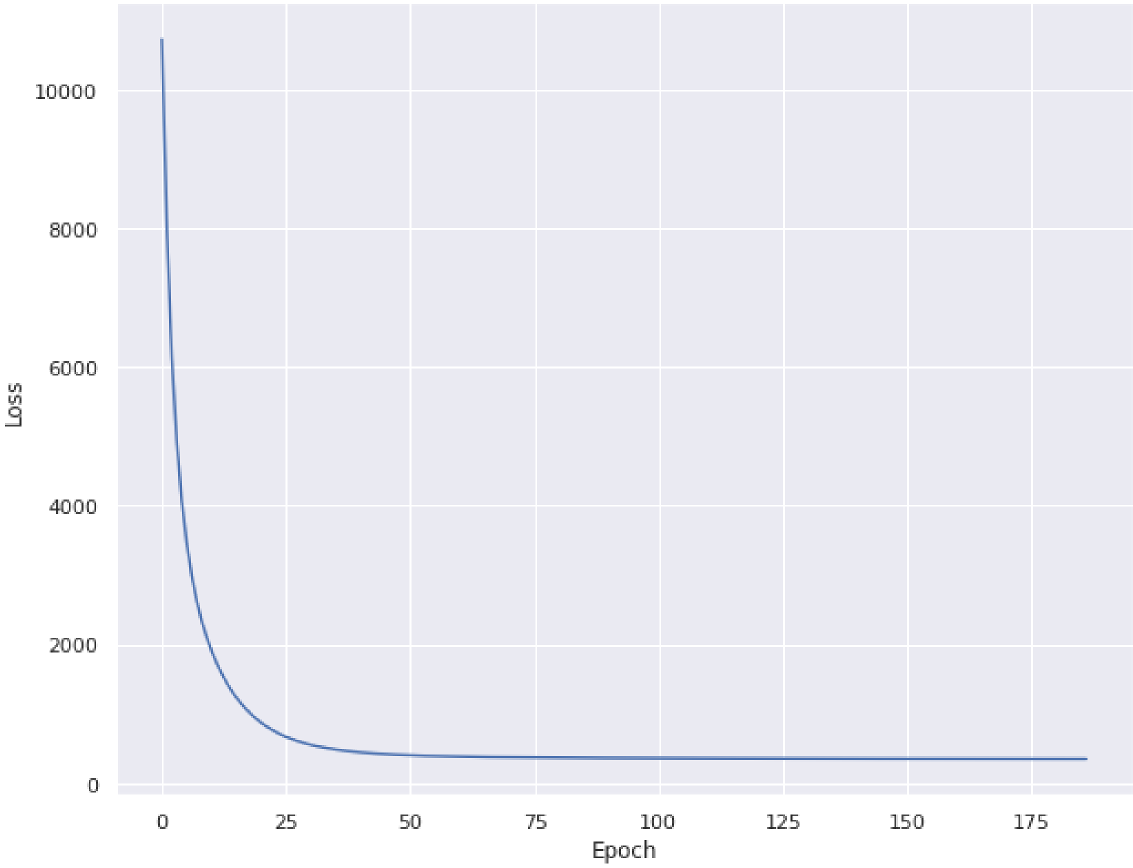

Actual vs. Predicted
Predicted
Actual
1200
1000
800
600
400
200
0
0
200
400
600
800
1000
1200

```
=============== Loss Array ===============
[9451.296956 5570.606424 3811.658445 2861.659506 2273.808451 1853.491014
 1541.16412  1302.985703 1116.741081  975.826888  857.438271  767.516946
  695.69767   637.83999   591.727992  549.301174  525.265444  500.117404
  480.102181  462.731169  449.932203  436.305751  428.833063  421.588054
  414.763695  409.37831   404.065292  399.913127  396.808468  393.343137
  390.171617  386.187598  386.449187  383.778331  381.782235  379.83988
  379.163404  377.197536  377.037522  375.156403  373.251189  373.657398
  371.555454  371.227435  370.978065  370.039332  369.165123  368.197961
  368.033575  367.890565  367.367457  366.728871  366.127781  364.956217
  364.985444  363.861655  364.359898  363.530161  363.327944  361.447514
  362.964871  361.74748   362.104168  359.908239  361.726581  360.622588
  360.60792   360.557518  360.014851]

For LR: 0.00075, Iterations= 15000
======================================
R2 Score:  0.9563311566052329
Mean absolute error:  25.773529889205754
Root Mean squared error:  55.64362966699465
Explained Variance Score:  0.9449732437305846
```

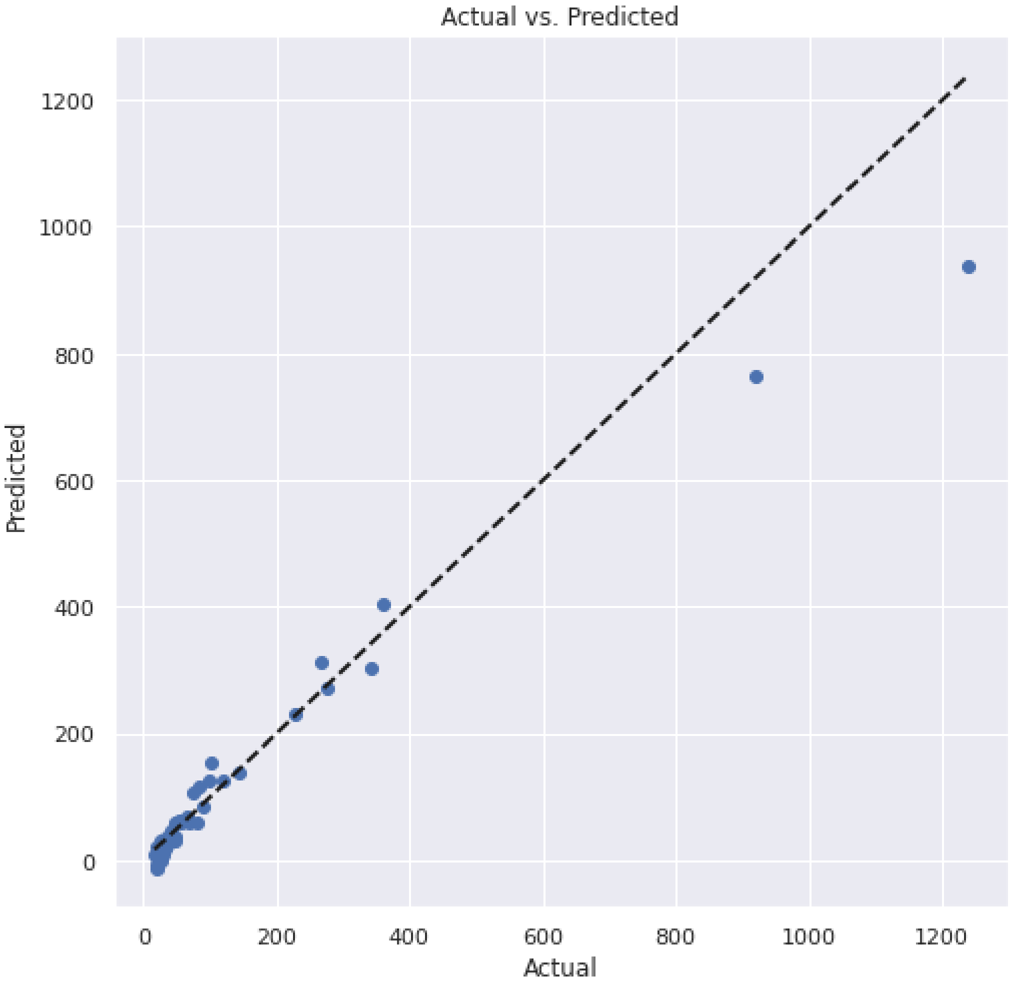



Actual vs. Predicted
Predicted
Actual
0
200
400
600
800
1000
1200

```
=============== Loss Array ===============
[10726.133807  7957.040124  6146.560817  4923.714031  4076.35851
  3456.396651  2993.445406  2634.31445   2346.482136  2110.031356
  1907.576369  1734.207063  1584.064634  1452.616766  1337.057452
  1234.492739  1144.422574  1064.284742   992.886575   928.825883
   872.393306   822.060684   777.355525   737.269891   701.436561
   669.490178   641.226095   615.915955   593.161166   572.088837
   553.410774   536.970427   522.090016   509.111926   496.857578
   486.234702   476.275296   467.210809   459.465176   451.929242
   445.422169   439.446743   434.042084   429.085355   424.528674
   420.488949   416.442961   413.173855   410.027335   406.969978
   403.727552   401.596792   399.592305   397.011144   395.478526
   393.678167   391.899986   390.019922   388.622683   387.310017
   385.721779   384.762138   383.52493    382.141382   381.443719
   380.332708   379.345622   378.642814   377.624823   376.892059
   376.223929   375.414479   374.73644    373.995048   373.238748
   372.708009   371.362932   371.161713   370.977564   370.405202
   370.023259   369.566562   369.054104   368.561507   368.022139
   367.720297   367.324106   366.719726   366.490115   365.916269
   365.762763   365.259159   364.884864   364.648449   364.431984
   364.110223   363.782097   363.478556   363.185247   362.872636
   362.606796   362.012608   361.946871   361.420587   361.4662
   361.265367   361.030005   360.894189   360.459739   360.267994
   360.062948   359.90168    359.554261   359.441896   359.263527
   359.054633   358.946608   358.695622   358.509345   358.01839
   358.155637   358.041624   357.838041   357.515183   357.417811
   357.333233   357.221166   356.991224   356.686513   356.503019
   356.645862   356.330314   356.233988   355.990161   355.954913
   355.901061   355.679073   355.675926   355.545782   355.429663
   355.23091    354.795061   355.082001   354.878696   354.751759
   354.502119   354.614467   354.24162    354.407746   354.23963
   354.175765   354.007244   354.021368   353.857619   353.680729
   353.5365     353.471581   353.538678   353.473499   353.404425
   353.160611   353.116925   352.852488   353.046609   352.635247
   352.861355   352.806959   352.570409   352.704282   352.461524
   352.485123   352.494196   352.38609    352.272872   352.224502
   352.167347   352.125477   351.925305   352.023746   351.773646
   351.767509   351.160457   351.767302   351.755749   351.568529
   351.60563    351.554254]

For LR: 0.000375, Iterations= 15000
======================================
R2 Score:  0.9567852037197244
Mean absolute error:  25.749909607335184
Root Mean squared error:  55.67365493672688
Explained Variance Score:  0.9448989118890614
```



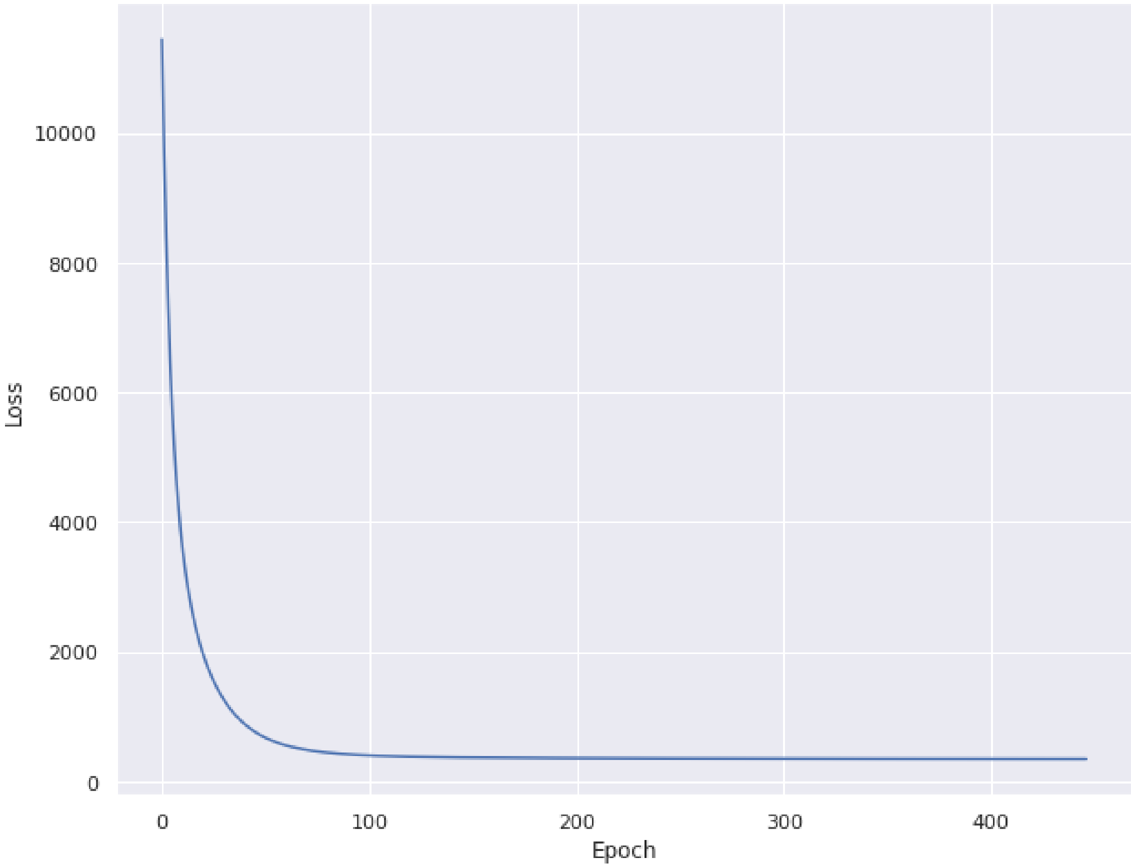
Loss
10000
8000
6000
4000
2000
0
0
25
50
75
100
125
150
175
Epoch

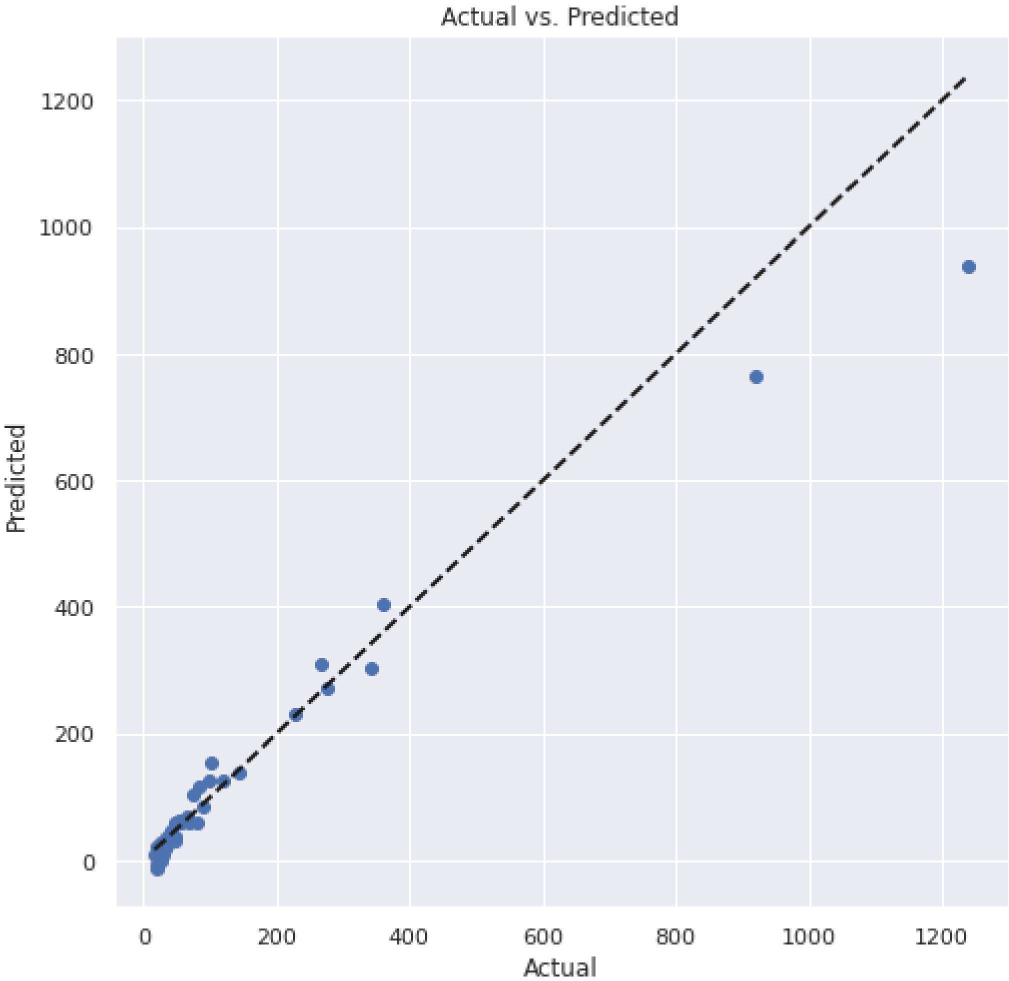


Actual vs. Predicted
Predicted
Actual
0
200
400
600
800
1000
1200

```
=============== Loss Array ===============
[11442.417137  9762.735825  8420.868082  7339.085987  6458.489765
  5736.585365  5141.526216  4646.250051  4231.039617  3880.770635
  3579.945004  3318.963756  3092.520799  2893.410457  2716.365169
  2557.393524  2412.925176  2282.002887  2161.845874  2051.924423
  1951.493929  1857.838509  1770.943002  1690.198903  1614.765312
  1544.339335  1478.712088  1417.026728  1359.403524  1304.960293
  1253.983114  1205.98973   1160.911222  1118.301702  1078.376113
  1040.749013  1005.149787   971.873722   940.268342   910.420536
   882.2203     855.645778   830.721726   806.719645   784.708182
   763.51225    743.584725   724.780394   706.948421   690.133568
   674.05697    659.110309   644.97103    631.30099    618.625829
   606.60685    595.235994   584.369337   574.176741   564.49645
   555.336276   546.636501   538.360754   530.308686   523.052177
   516.012198   509.438663   502.900277   497.075068   491.372966
   485.93419    480.84149    475.888187   471.159111   466.846922
   462.650451   458.591001   454.836578   451.209408   447.729749
   444.439285   441.295971   438.381357   435.523129   432.734888
   430.224652   427.755924   425.387289   422.992173   420.885961
   418.733417   416.961255   414.92322    413.188927   411.463754
   409.780987   408.228885   406.548312   405.221656   403.802024
   402.440696   401.174578   399.913631   398.717006   397.514356
   396.438169   395.398776   394.193892   393.370606   392.409265
   391.486512   390.551805   389.72514    388.902656   388.095217
   387.218871   386.517781   385.706457   385.043541   384.388686
   383.797184   383.058778   382.483916   381.93888    381.301012
   380.800866   380.220405   379.646998   379.115009   378.690547
   378.168225   377.745206   377.266316   376.794496   376.250818
   375.918123   375.510182   375.051079   374.712489   374.307648
   373.93256    373.532274   373.043154   372.827535   372.447705
   372.151023   371.772057   371.491767   371.098741   370.807043
   370.557094   370.111278   369.943382   369.696068   369.385961
   369.125158   368.860535   368.527458   368.277289   368.00567
   367.816036   367.512397   367.334817   367.05769    366.82155
   366.595777   366.394592   366.118616   365.954754   365.659707
   365.519576   365.188199   365.079999   364.901595   364.709005
   364.438772   364.22714    364.138541   363.936848   363.752382
   363.580838   363.326103   363.190891   362.999334   362.857578
   362.685646   362.529214   362.383817   362.12963    362.04889
   361.913796   361.746454   361.595493   361.44823    361.279172
   361.123706   360.845597   360.827808   360.664159   360.563636
   360.41139    360.297267   360.173433   359.990128   359.909499
   359.624684   359.646821   359.476038   359.345461   359.261996
   359.136366   358.984704   358.865209   358.774639   358.668779
   358.480122   358.440773   358.317544   358.207657   357.995882
   357.997354   357.878918   357.765406   357.645783   357.543284
   357.373181   357.360735   357.231947   357.126083   357.018443
   356.945356   356.831329   356.697864   356.65561    356.59447
   356.421658   356.37988    356.313895   356.196788   356.136949
   356.040199   355.943167   355.7696     355.777102   355.708168
   355.606602   355.539931   355.459168   355.374169   355.272398
   355.126221   355.11149    355.047558   354.981796   354.898446
   354.832963   354.745369   354.641883   354.609991   354.473139
   354.414032   354.332677   354.29746    354.244972   354.17571
   354.032147   354.0085     353.909805   353.91752    353.755598
   353.783238   353.697827   353.650508   353.522742   353.50033
   353.378164   353.365468   353.335103   353.262717   353.157575
   353.148009   353.042474   353.019036   352.933789   352.89371
   352.861596   352.751789   352.750844   352.676208   352.58479
   352.573596   352.511569   352.454407   352.413671   352.349475
   352.272539   352.159854   352.150341   352.174374   352.081359
```

```
   352.001231   352.021516   351.950414   351.880582   351.760893
   351.806932   351.723504   351.733813   351.683535   351.6312
   351.564474   351.465825   351.512435   351.407545   351.397004
   351.324143   351.300389   351.266185   351.229378   351.14827
   351.129068   351.115161   351.04127    350.884402   350.970983
   350.917642   350.902589   350.83102    350.83382    350.765419
   350.712564   350.7143     350.660666   350.532989   350.586898
   350.524002   350.529384   350.416064   350.437331   350.424165
   350.353245   350.34473    350.306717   350.220435   350.235508
   350.143086   350.1214     350.146512   350.015796   350.068828
   349.986582   350.014866   349.982222   349.934063   349.898481
   349.889604   349.858484   349.801821   349.733798   349.77108
   349.6913     349.707171   349.683698   349.636244   349.599007
   349.509083   349.572806   349.498543   349.519423   349.486767
   349.428432   349.414855   349.380995   349.361814   349.327356
   349.279952   349.271195   349.198559   349.240282   349.215997
   349.18783    349.128945   349.15672    349.113988   349.078001
   349.072007   349.045267   348.912773   348.990813   348.940211
   348.952092   348.916677   348.848981   348.889485   348.857411
   348.825105   348.817681   348.783939   348.763157   348.680399
   348.708009   348.70172    348.646558   348.659292   348.557517
   348.629398   348.576258   348.525712   348.530048   348.532098
   348.508576   348.494367   348.398747   348.461907   348.432059
   348.387227   348.411757   348.366811   348.337662   348.29823
   348.33899    348.276246   348.311866   348.268196   348.225441
   348.247123   348.203655   348.207378   348.160951   348.142629
   348.174827   348.07888    348.133884   348.11431    348.086763
   348.057971   348.057341   347.916287   347.975304   348.012707
   348.006122   347.872796   347.967232   347.939577   347.931791
   347.898294   347.879045]

For LR: 0.0001875, Iterations= 15000
======================================
R2 Score:  0.9569457924227388
Mean absolute error:  25.734194145432177
Root Mean squared error:  55.62758308756151
Explained Variance Score:  0.9449738918758595
```


Loss
10000
8000
6000
4000
2000
0
0
100
200
300
400
Epoch



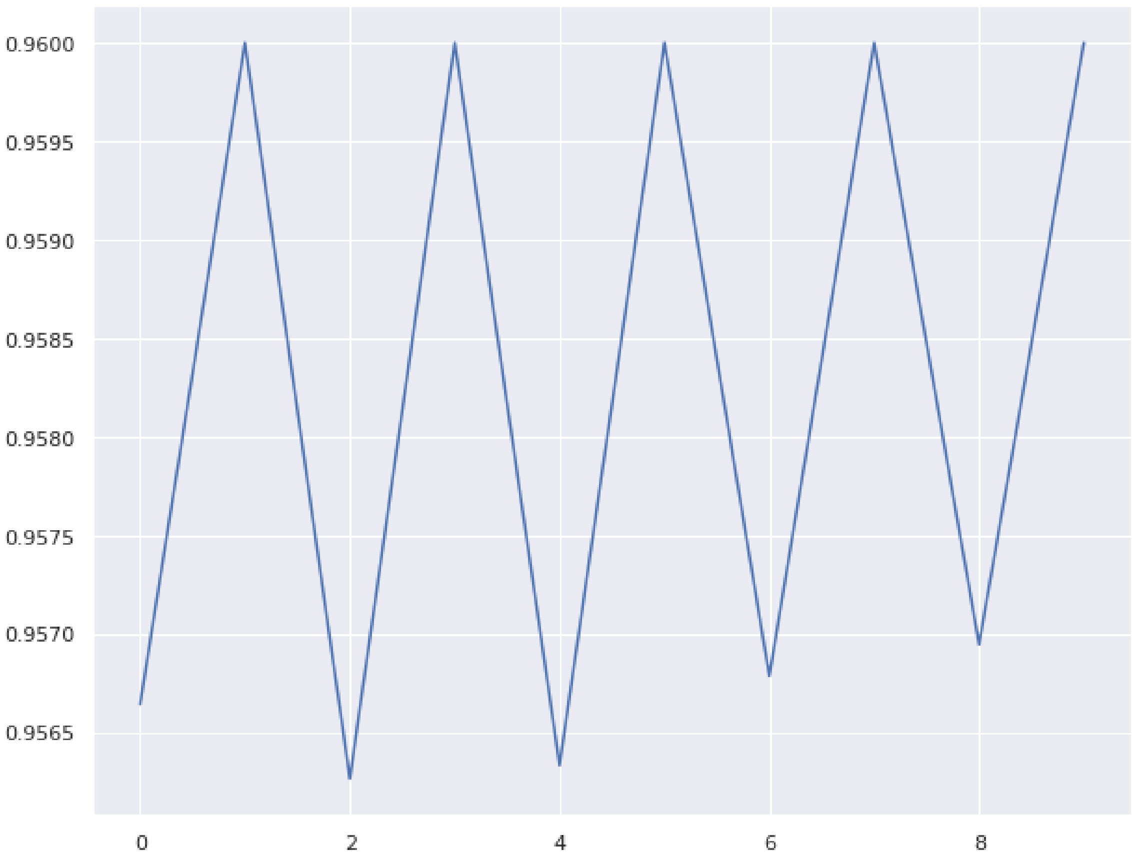
Actual vs. Predicted
Predicted
Actual
0
200
400
600
800
1000
1200

In [46]:

```
plt.plot(r2_lst)
```

Out[46]:

```
[<matplotlib.lines.Line2D at 0x7f922e11ac90>]
```



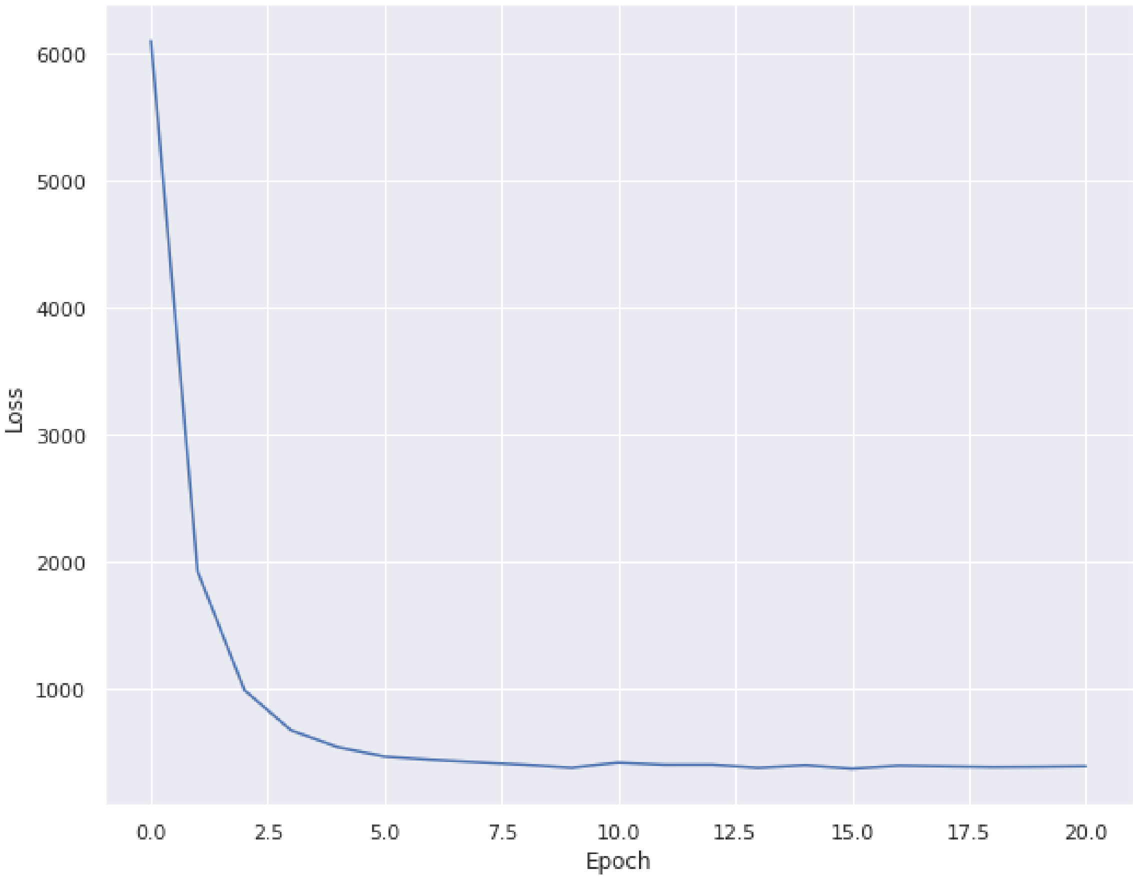
This does not improves much

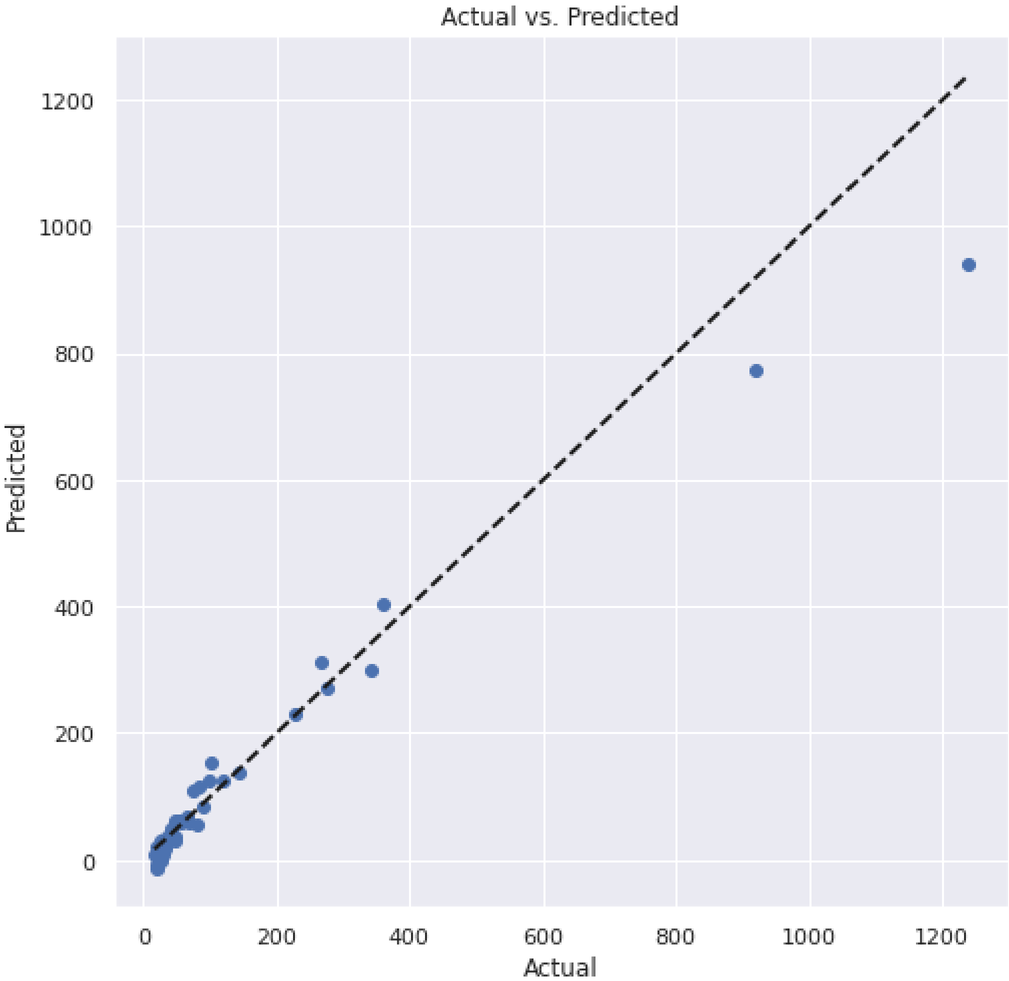
We can try to improve the above plot by increasing iterations say 150000

In [47]:

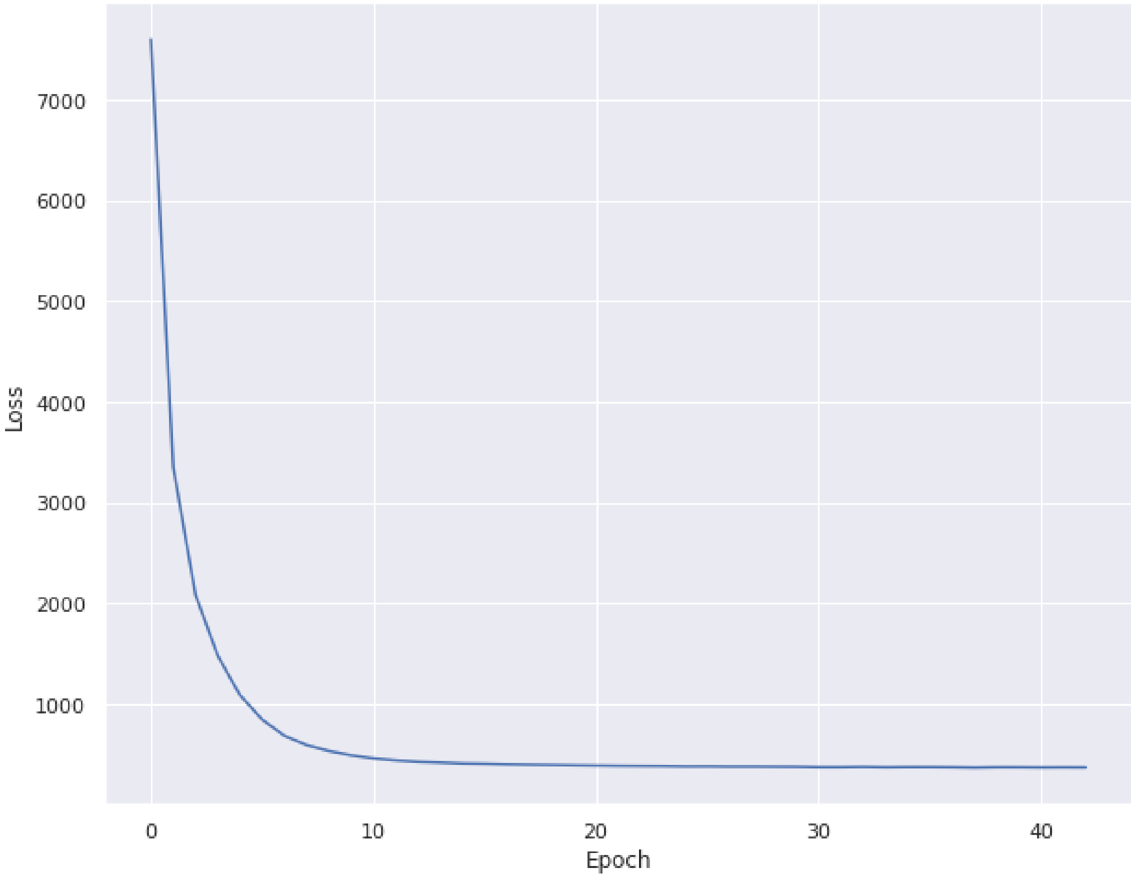
```
lr = 0.003
itrs = 150000
r2_lst = []

while lr >= 0.0001:
    model=SGDRegressor(learning_rate='constant', eta0=lr, max_iter=itrs, verbose=1)

    with DisplayLossCurve(print_loss=True):
        model.fit(X_train_sc,y_train)

    y_pred=model.predict(X_test_sc)

    r2 = model.score(X_train_sc,y_train)
    mae = mean_absolute_error(y_test, y_pred)
    rmse = mean_squared_error(y_test, y_pred, squared=False)
    evs = explained_variance_score(y_test, y_pred)

    r2_lst.append(r2)

    print()
    print("For LR: "+str(lr)+", Iterations= "+str(itrs))
    print("======================================")
    print("R2 Score: ", r2)
    print("Mean absolute error: ", mae)
    print("Root Mean squared error: ", rmse)
    print("Explained Variance Score: ", evs)

    file = open("Scikit_SGD_log.txt","a")
    file.write("LR = " + str(lr) + ",max_iterations = " + str(itrs) +
               " R^2 = " + str(r2) + ", MAE = " + str(mae) + ", RMSE = "
               + str(rmse) + ",  Explained-Variance = " + str(evs) + " \n")
    file.close()

    plt.figure(figsize = (8,8))
    plt.scatter(y_test, y_pred)
    plt.plot([y_test.min(),y_test.max()],[y_test.min(),y_test.max()],'k--',lw=2)
    plt.xlabel('Actual')
    plt.ylabel('Predicted')
    plt.title('Actual vs. Predicted')
    plt.show()

    r2_lst.append(r2)
    lr/=2
```

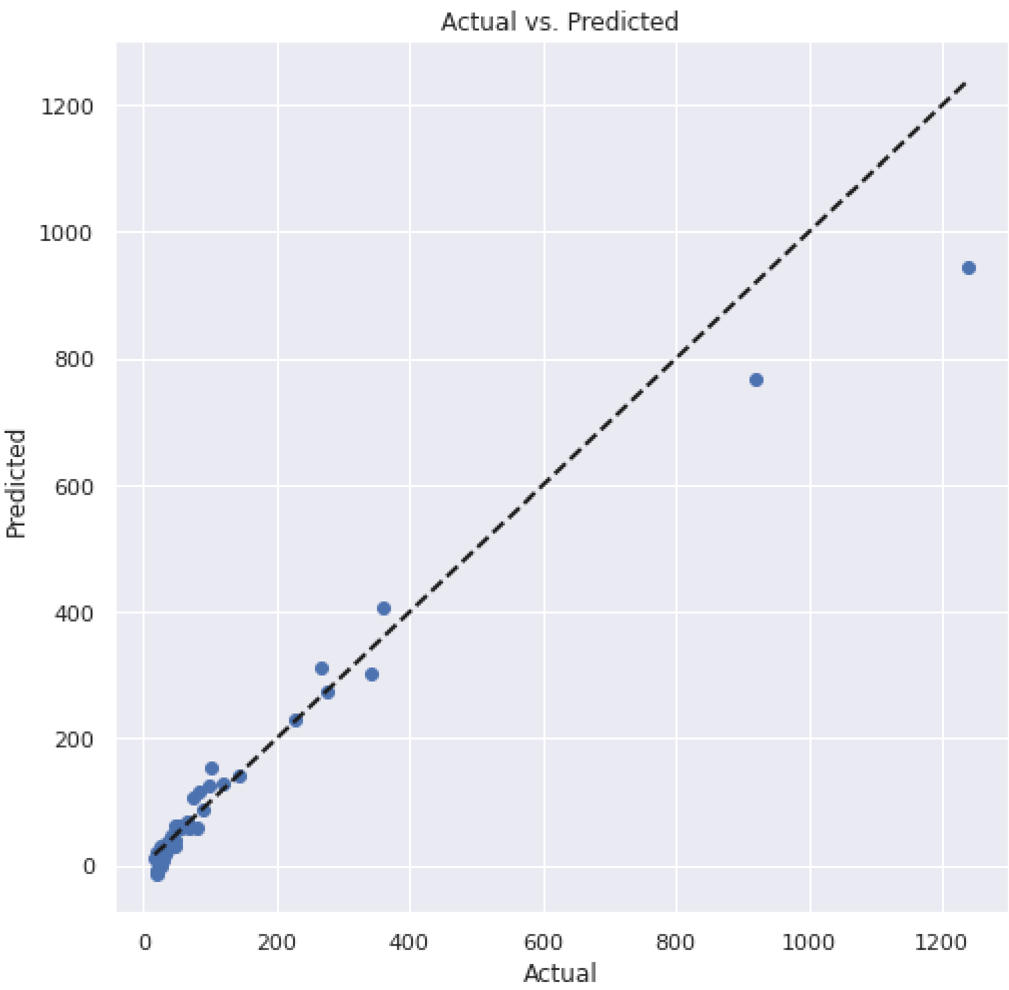
```
=============== Loss Array ===============
[6101.127778 1920.397509  986.023617  668.986477  536.153178  463.047835
  437.70771   417.881743  397.510388  374.673511  415.276391  397.470233
  397.993757  374.03565   394.157473  367.410717  390.748602  385.170346
  379.807997  381.814094  386.132441]

For LR: 0.003, Iterations= 150000
======================================
R2 Score:  0.9567803684647529
Mean absolute error:  25.81688007988116
Root Mean squared error:  54.965241828486015
Explained Variance Score:  0.9463760450717472
```



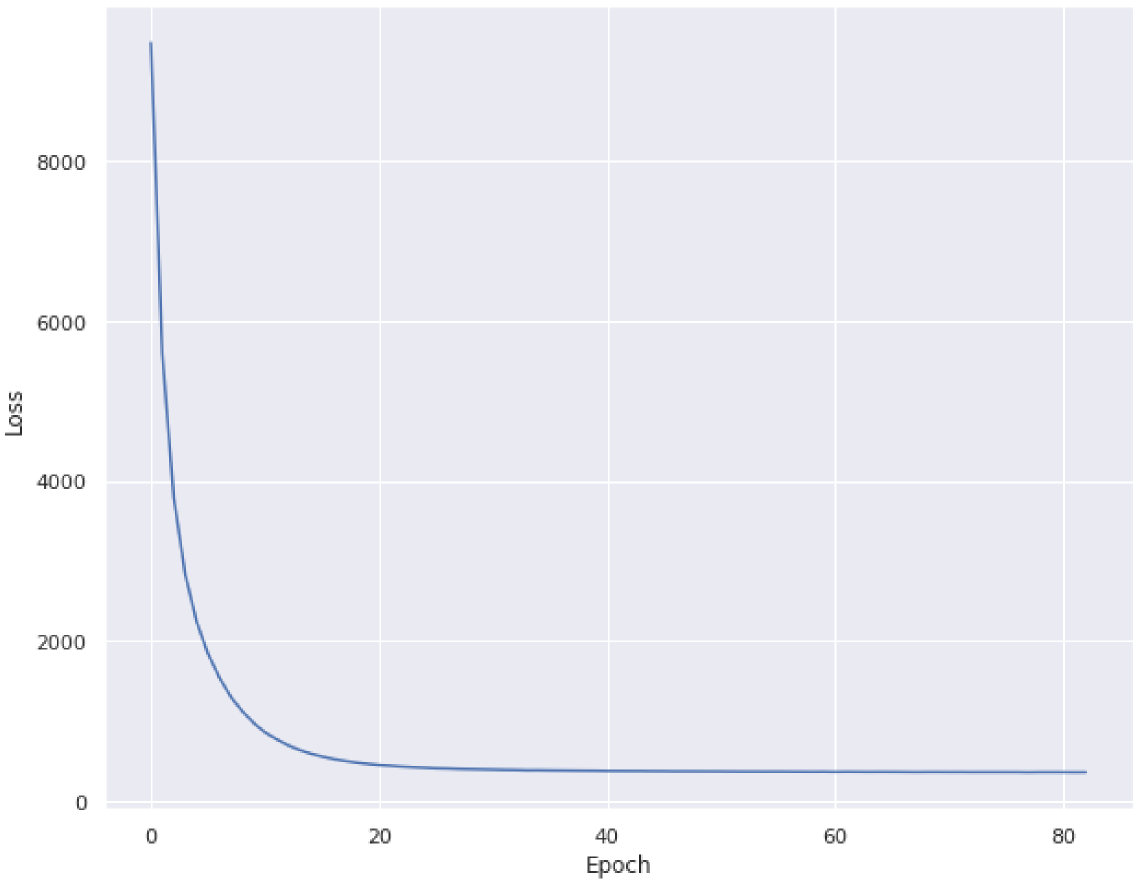

Actual vs. Predicted
1200
1000
800
600
400
200
0
Predicted
0
200
400
600
800
1000
1200
Actual

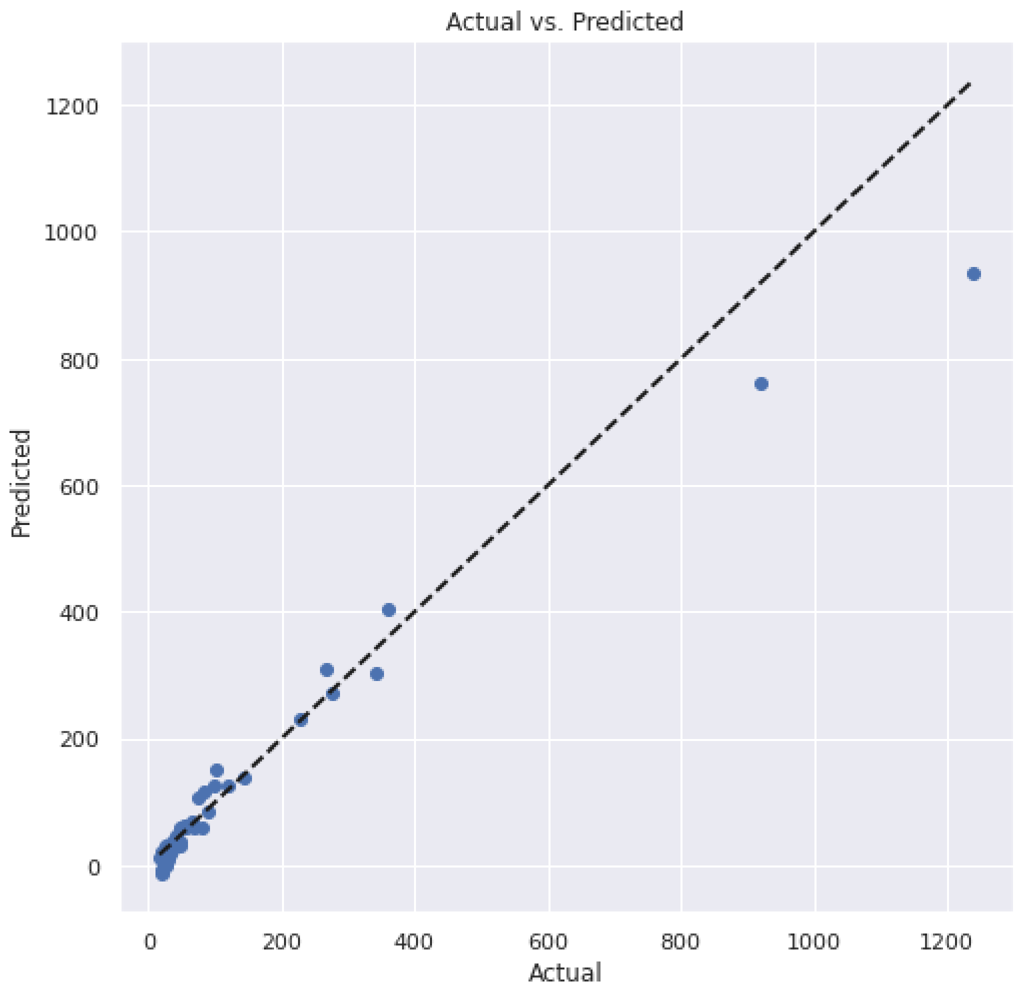
```
=============== Loss Array ===============
[7602.141366 3352.955513 2085.6795   1477.466611 1083.903796  841.726795
  679.585864  586.768109  529.124737  486.246253  455.977275  436.403824
  422.27966   414.368216  404.749089  401.3991    395.994507  392.371342
  389.828976  386.531204  384.017599  381.377076  379.960215  379.019211
  375.898018  376.447234  374.391748  374.57004   373.504381  372.578393
  368.275868  368.244992  370.937849  367.273792  369.270062  369.008374
  367.603578  364.018306  367.619334  367.22897   365.440626  366.521477
  365.647689]

For LR: 0.0015, Iterations= 150000
======================================
R2 Score:  0.9567160799377619
Mean absolute error:  25.73949250215789
Root Mean squared error:  55.00590281397207
Explained Variance Score:  0.9462444504604843
```



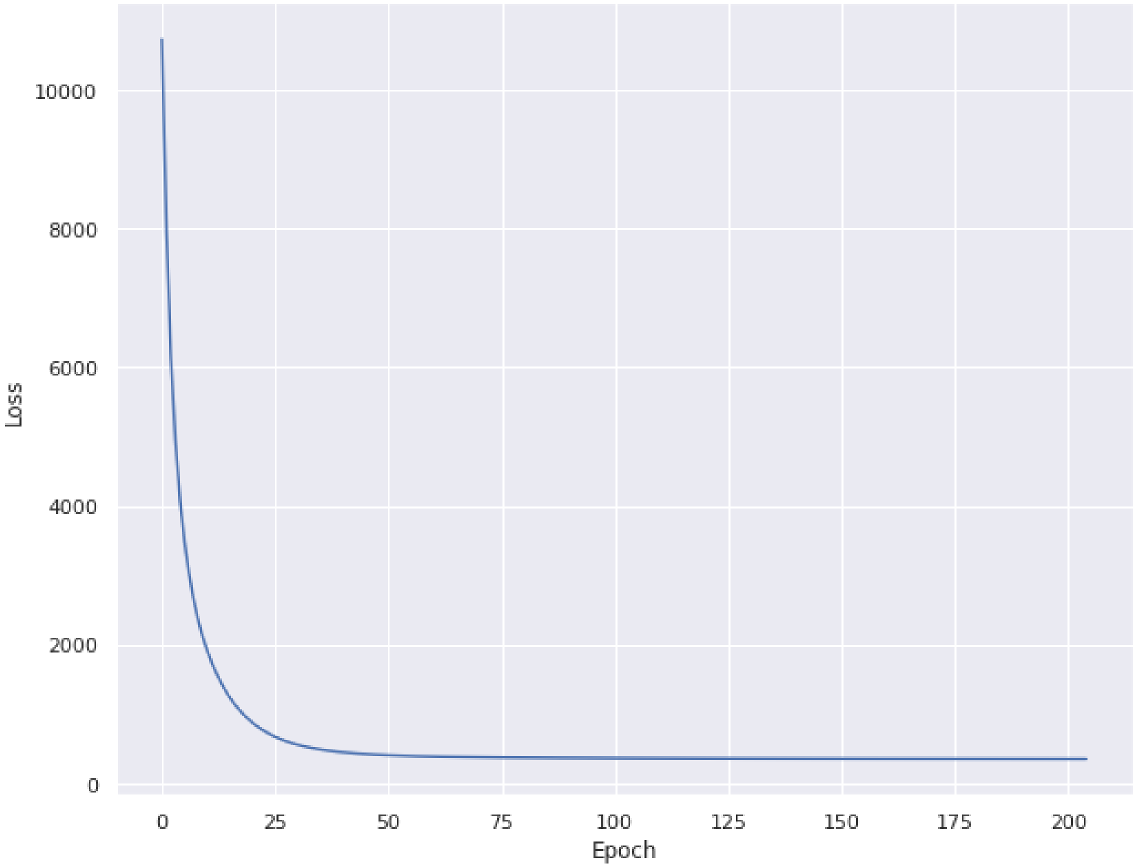

Actual vs. Predicted
Predicted
Actual
0
200
400
600
800
1000
1200

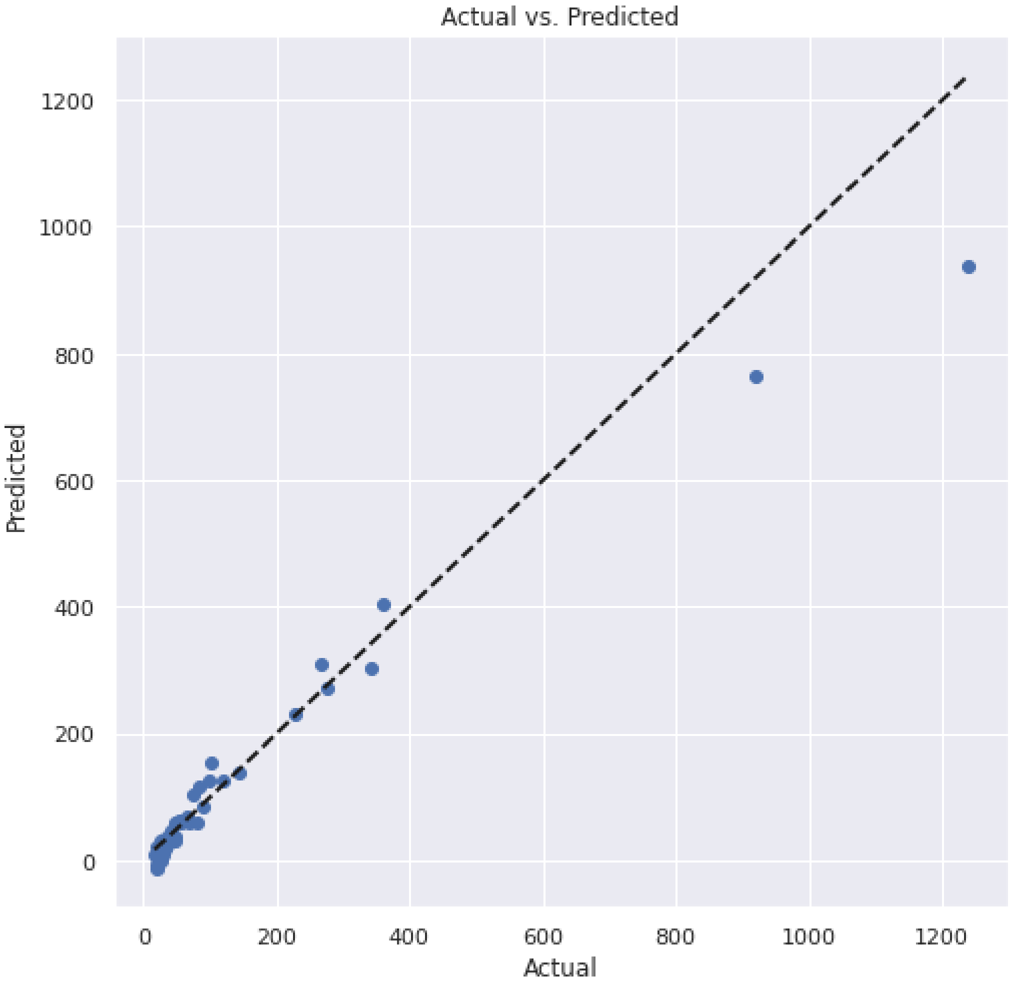
```
=============== Loss Array ===============
[9478.626259 5599.557043 3794.836287 2837.354771 2248.555956 1842.961546
 1543.039038 1304.435353 1121.851984  976.595814  858.157711  772.224106
  696.706397  639.663398  591.459517  555.034794  523.846702  499.663367
  479.335427  462.806947  447.656179  438.730334  428.436091  420.950289
  414.501633  407.163705  404.56903   399.110141  396.634481  393.66823
  390.767787  387.454035  385.955385  381.717237  382.521471  380.224055
  379.307286  377.970689  376.723749  376.090926  373.537877  373.141855
  372.330289  372.158612  370.530918  370.28877   368.114548  368.626735
  368.2984    367.572427  367.22201   366.032784  365.355059  365.448591
  364.312437  365.057487  363.993187  363.120054  362.403925  363.121994
  361.206704  362.667463  361.946252  360.930285  361.570644  361.05835
  360.391412  358.858677  359.661491  359.933114  359.147885  359.267063
  358.352207  358.785916  358.754228  358.410486  358.169107  356.464461
  358.16112   358.113302  357.731624  356.559539  357.233042]

For LR: 0.00075, Iterations= 150000
======================================
R2 Score:  0.9566452241351361
Mean absolute error:  25.79057871112131
Root Mean squared error:  56.19109022983336
Explained Variance Score:  0.9438778939745015
```




Actual vs. Predicted
1200
1000
800
600
400
200
0
Predicted
0
200
400
600
800
1000
1200
Actual

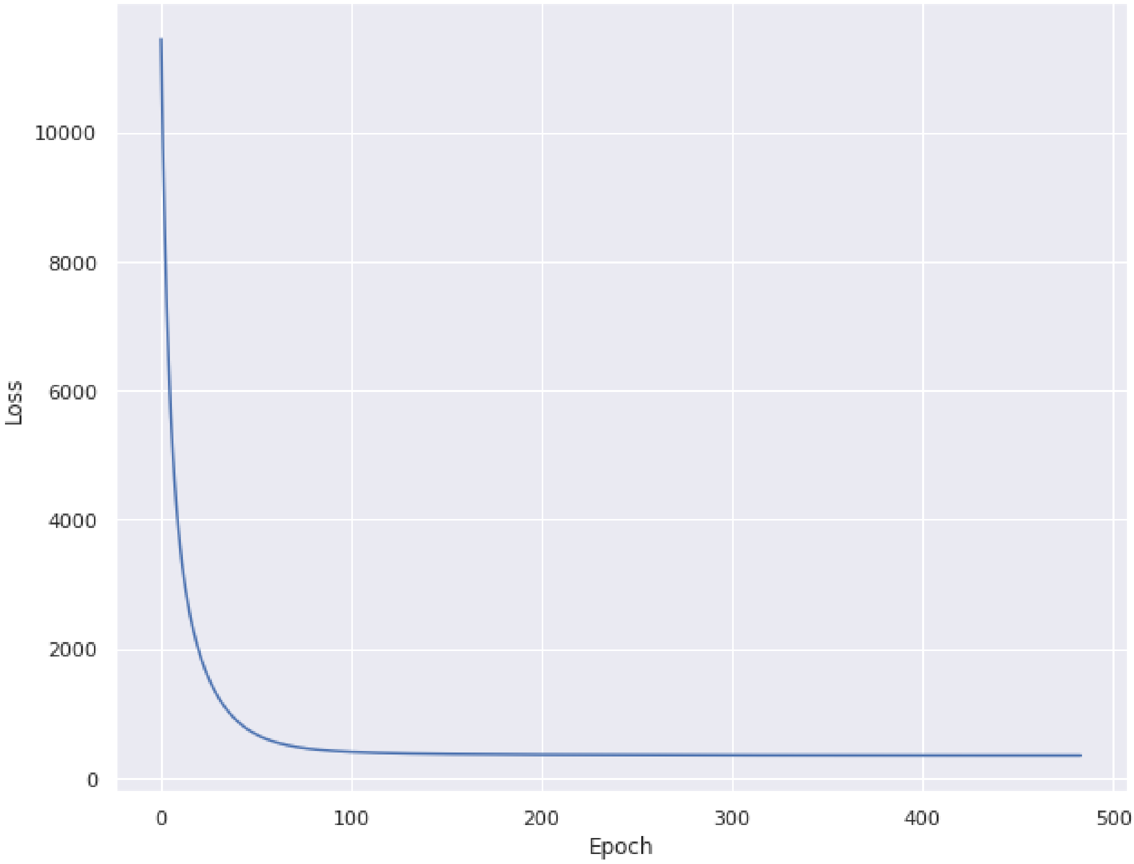
```
=============== Loss Array ===============
[10729.434839  7952.853512  6142.11981   4932.262106  4088.387702
  3478.295707  3013.509436  2649.765813  2356.581786  2115.443041
  1913.376857  1739.151191  1587.574691  1455.547172  1338.629532
  1235.135264  1144.208403  1063.945854   992.409866   928.549775
   872.584273   822.214542   776.594537   737.569914   701.873307
   669.214656   641.016059   615.231883   592.049007   571.735391
   554.107965   537.416102   522.382763   508.98459    496.936999
   485.997498   475.854824   467.613759   459.237388   451.969408
   445.636898   439.384346   433.959378   429.084744   424.382883
   420.05543    416.566294   413.17319    409.468894   406.2405
   404.337035   401.805348   399.405735   397.366362   395.094836
   393.639196   391.606731   389.875532   388.761232   387.369273
   385.860842   384.741778   383.526382   382.479335   381.424142
   379.730328   379.590444   378.709502   377.837387   376.967868
   376.23574    375.338771   374.779673   374.005166   373.413344
   372.751741   372.056686   371.624251   371.040413   370.491539
   369.86922    369.508281   368.891517   368.721663   368.240812
   367.301268   367.3995     366.639449   366.523602   366.024653
   365.840657   365.475492   364.929255   364.580058   364.477778
   364.153462   363.855536   363.440599   363.225063   362.900527
   362.660698   362.045746   362.135442   361.887205   361.491753
   361.361112   361.11294    360.850261   360.619904   360.316238
   360.168178   359.885131   359.721881   359.318119   359.333826
   359.128986   358.756872   358.740474   358.572539   358.412341
   357.921428   358.070391   357.886422   357.718657   357.360119
   356.954202   357.293124   357.022182   356.671245   356.438929
   356.429923   356.409961   356.325906   356.141953   356.013922
   355.834338   355.348439   355.635903   355.530375   355.408502
   355.311156   355.087519   354.885567   354.949679   354.615946
   354.605454   354.432895   354.515536   354.3348     354.206313
   353.952582   354.167442   354.003155   353.944291   353.657533
   353.688575   353.612218   353.537912   353.360713   353.016043
   353.206945   353.180445   352.886513   353.037052   352.907915
   352.713492   352.67277    352.740294   352.6877     352.553704
   352.474935   352.397852   351.966605   352.275036   352.29629
   351.955467   352.070532   351.839185   351.76993    351.730184
   351.942548   351.858893   351.741148   351.621874   351.646133
   351.507424   351.300601   351.538879   351.347036   351.184224
   351.30928    351.155576   351.190116   351.129619   351.03477
   350.80829    350.923952   350.953027   350.645757   350.355221
   350.85496    350.779234   350.714471   350.679544   350.425844]

For LR: 0.000375, Iterations= 150000
======================================
R2 Score:  0.9568818909482312
Mean absolute error:  25.746670589200694
Root Mean squared error:  55.639660926615406
Explained Variance Score:  0.9449590225787773
```


10000
8000
6000
4000
2000
0
Loss
0
25
50
75
100
125
150
175
200
Epoch

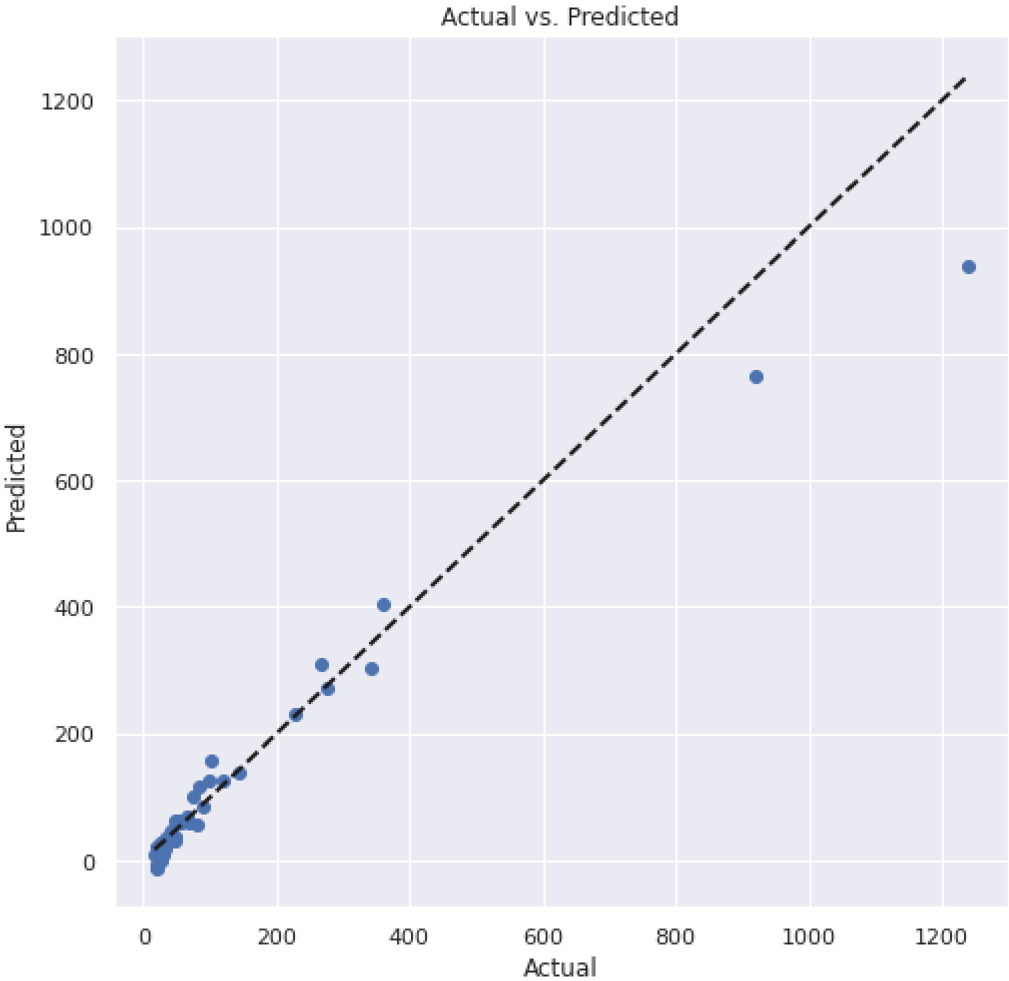

Actual vs. Predicted
Predicted
Actual
0
200
400
600
800
1000
1200

```
=============== Loss Array ===============
[11444.99693   9768.871772  8427.487031  7341.793629  6458.26324
  5738.423387  5143.589745  4650.118154  4236.664048  3885.537172
  3584.136533  3323.239894  3096.838352  2895.226956  2715.86521
  2555.274513  2411.098325  2280.339403  2160.647667  2050.531294
  1949.25845   1855.844519  1769.145925  1688.617551  1613.366358
  1543.122786  1477.500848  1415.954606  1358.430682  1304.257033
  1253.41576   1205.304943  1160.250812  1118.006689  1078.022054
  1040.229672  1004.797045   971.25722    939.871934   910.051939
   881.914702   855.345344   830.277642   806.612883   784.277103
   762.999434   743.054499   724.350796   706.593519   689.726725
   673.807736   658.790529   644.575723   631.108796   618.348304
   606.253794   594.80719    583.930648   573.815119   564.164966
   555.017562   546.255836   537.842361   530.306272   522.878243
   515.934809   509.108968   502.820632   496.917995   490.800061
   485.805795   480.496982   475.783441   471.106389   466.718494
   462.539412   458.513322   454.680752   451.066203   447.661377
   444.318455   441.213584   438.221588   435.393142   432.65917
   430.059079   427.50372    425.227393   422.916706   420.837978
   418.725422   416.795002   414.887856   413.060698   411.356755
   409.689677   408.086564   406.503081   405.125367   403.726003
   402.333344   401.092788   399.843463   398.622413   397.492551
   396.38381    395.303586   394.26946    393.298478   392.341023
   391.365266   390.513439   389.64067    388.818257   388.022148
   387.234212   386.378979   385.781526   385.075783   384.407052
   383.715256   383.094021   382.473935   381.885751   381.3079
   380.737999   380.113596   379.625649   379.074715   378.629563
   378.15471    377.66624    377.164653   376.606601   376.325778
   375.903784   375.426846   375.059022   374.653743   374.06372
   373.895168   373.471823   373.159191   372.805879   372.460066
   372.099812   371.779357   371.452407   371.111857   370.823659
   370.459277   370.230452   369.916669   369.628513   369.370901
   369.088081   368.784676   368.511773   368.252642   368.047982
   367.759172   367.537705   367.300523   367.075478   366.842507
   366.561448   366.301422   366.121543   365.926902   365.726558
   365.491173   365.297554   365.10158    364.881637   364.691975
   364.499863   364.295835   364.114635   363.927272   363.686692
   363.538993   363.352172   363.194877   363.027479   362.849939
   362.627123   362.509313   362.352744   362.195423   361.961478
   361.855459   361.726926   361.539405   361.424529   361.265617
   361.111286   360.950599   360.7585     360.690228   360.568024
   360.418886   360.237655   360.132061   359.951334   359.894235
   359.773637   359.623087   359.50261    359.383042   359.19586
   359.132157   359.005667   358.86902    358.780144   358.652022
   358.48195    358.413669   358.28164    358.153019   358.090395
   357.942958   357.840906   357.688639   357.675083   357.559096
   357.464097   357.299061   357.25297    357.14485    357.030476
   356.935531   356.853142   356.763775   356.665574   356.540525
   356.459086   356.294505   356.255894   356.202948   356.120036
   356.025987   355.934156   355.854313   355.768385   355.68971
   355.593583   355.505551   355.440032   355.326291   355.266569
   355.19518    355.104558   355.02876    354.918761   354.866226
   354.801147   354.729604   354.662823   354.585427   354.51946
   354.400754   354.361062   354.304877   354.228538   354.163785
   354.088595   354.024508   353.900059   353.861853   353.81937
   353.747972   353.671747   353.617968   353.537711   353.46595
   353.431346   353.376807   353.284726   353.24417    353.142676
   353.11185    353.009496   353.020913   352.953825   352.907395
   352.85068    352.74322    352.724037   352.642642   352.619942
   352.554721   352.510973   352.456206   352.403818   352.351417
   352.293462   352.245848   352.17985    352.153578   352.087594
```

```
   352.043721   351.983196   351.906008   351.867183   351.855881
   351.787984   351.744035   351.690327   351.654889   351.62319
   351.557099   351.460006   351.421317   351.415206   351.389921
   351.32531    351.273143   351.25015    351.181472   351.17093
   351.128011   351.098423   351.049059   350.992571   350.981071
   350.849999   350.832361   350.852192   350.821983   350.757178
   350.737877   350.69638    350.669584   350.617358   350.59611
   350.553812   350.477528   350.466776   350.422922   350.427644
   350.374394   350.337297   350.24141    350.269517   350.16233
   350.205071   350.165134   350.089526   350.113812   349.990981
   350.025904   350.023343   349.981228   349.950495   349.920267
   349.841086   349.855318   349.778798   349.776421   349.767438
   349.709054   349.670322   349.670308   349.633837   349.621346
   349.585922   349.562662   349.446629   349.482266   349.478188
   349.452482   349.43311    349.384846   349.371371   349.351103
   349.315997   349.293034   349.231521   349.224171   349.194868
   349.172976   349.167759   349.096247   349.098717   349.088439
   349.055273   349.026215   349.009724   348.9757     348.973509
   348.9408     348.926487   348.904461   348.873067   348.854216
   348.748476   348.797048   348.785211   348.744833   348.668032
   348.698894   348.705006   348.654401   348.644478   348.632124
   348.62628    348.569294   348.564841   348.511188   348.548477
   348.501074   348.497142   348.471172   348.434534   348.357418
   348.43665    348.408652   348.393582   348.352467   348.331305
   348.327854   348.23534    348.288461   348.256117   348.267193
   348.248785   348.212642   348.127258   348.174336   348.170904
   348.166454   348.114648   348.112377   348.116334   348.09543
   348.082986   348.06526    348.050929   347.998169   348.005866
   347.982001   347.971555   347.942718   347.951898   347.941604
   347.906646   347.900441   347.89082    347.849964   347.858735
   347.830331   347.793047   347.818335   347.784767   347.795315
   347.765582   347.732647   347.739422   347.745328   347.721512
   347.713586   347.701908   347.681571   347.636244   347.660277
   347.625904   347.628328   347.603403   347.567754   347.605348
   347.56748    347.521276   347.525051   347.548813   347.544475
   347.51644    347.510295   347.478021   347.370705   347.461829
   347.468652   347.453394   347.419027   347.434788]

For LR: 0.0001875, Iterations= 150000
======================================
R2 Score:  0.9570015304090982
Mean absolute error:  25.71783795648453
Root Mean squared error:  55.53317057728711
Explained Variance Score:  0.9451487248002817
```



Loss
Epoch
10000
8000
6000
4000
2000
0
0
100
200
300
400
500

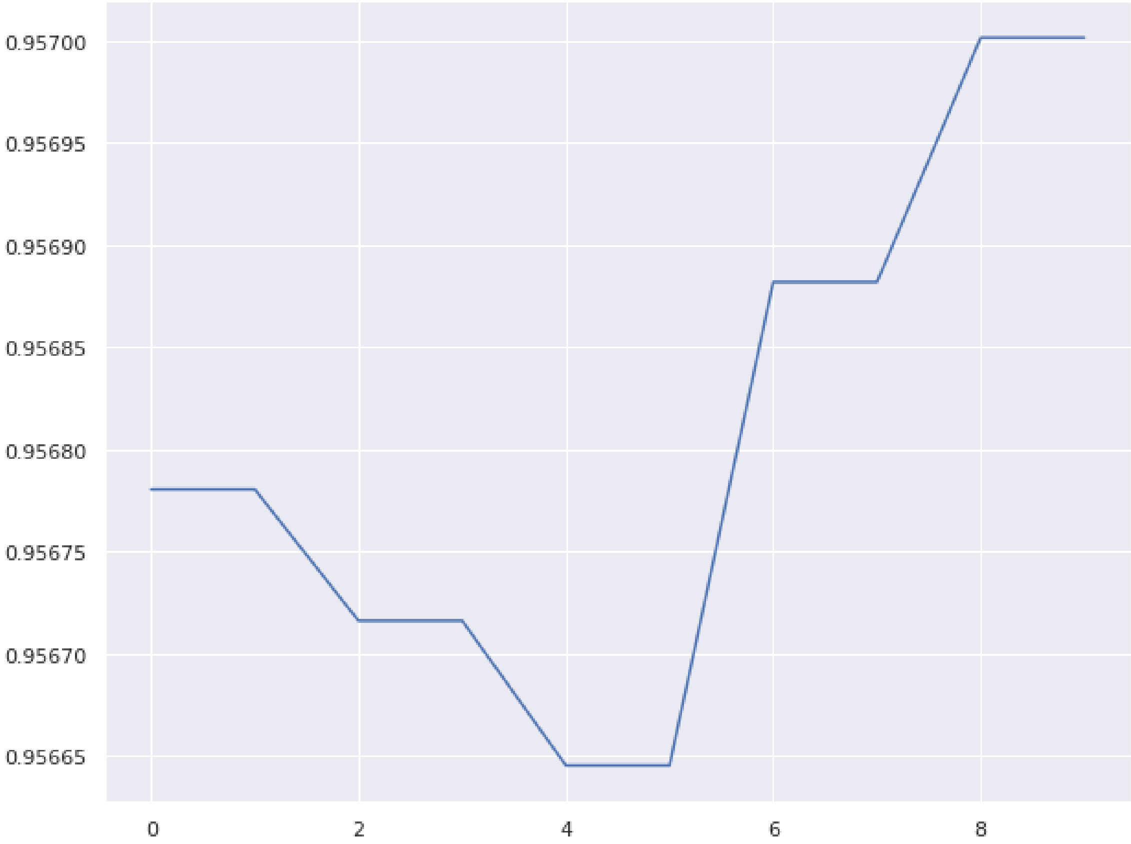

Actual vs. Predicted
1200
1000
800
600
400
200
0
Predicted
0
200
400
600
800
1000
1200
Actual

In [48]:

```
plt.plot(r2_lst)
```

Out[48]:

```
[<matplotlib.lines.Line2D at 0x7f922e6d0310>]
```



We can observe this also doesn't help much

Final Conclusion:

Hence we conclude that our custom SGD regressor works better than Scikit's SGD regressor