Group Name : Data_Wizards

Group Members

Pushkar Jain

Tanuj Pancholi

DATAWIZ TASK 3 - Machine learning

Defining the problem statement

Implement all 3 using sklearn/your own code in Python on the dataset that we have provided, to predict y

In [147]:
```
import pandas as pd
import numpy as np
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
import seaborn as seabornInstance
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import RidgeCV
from sklearn.preprocessing import PolynomialFeatures
from sklearn import metrics
from sklearn.metrics import r2_score
%matplotlib inline
import bokeh
from bokeh.plotting import figure, output_file, show
from bokeh.io import output_notebook
from bokeh.io import push_notebook, show, output_notebook
from bokeh.layouts import row
from bokeh.plotting import figure

output_notebook()
```

(https://bokeh.org)

BokehJS 1.4.0 successfully loaded.

Step 1: Gather the data

In [148]:
```
df = pd.read_csv("datasets_1256_2242_train.csv")
```

In [149]:
```
df.head()
```

Out[149]:

	x	y
0	24.0	21.549452
1	50.0	47.464463
2	15.0	17.218656
3	38.0	36.586398
4	87.0	87.288984

Step 2: Prepare the Data

Checking for null values

In [150]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 700 entries, 0 to 699
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   x       700 non-null    float64
 1   y       699 non-null    float64
dtypes: float64(2)
memory usage: 11.1 KB
```

As the number of x and y values are not equal there is one null value

Removing Null Values

In [151]:
```
df.dropna(inplace = True)
```

Rechecking for Null Values

In [152]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 699 entries, 0 to 699
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   x       699 non-null    float64
 1   y       699 non-null    float64
dtypes: float64(2)
memory usage: 16.4 KB
```

As the number of x and y values are equal there is no null value

Finding Co-relation between data

In [153]:
```
df.corr()
```

Out[153]:

	x	y
x	1.00000	0.99534
y	0.99534	1.00000

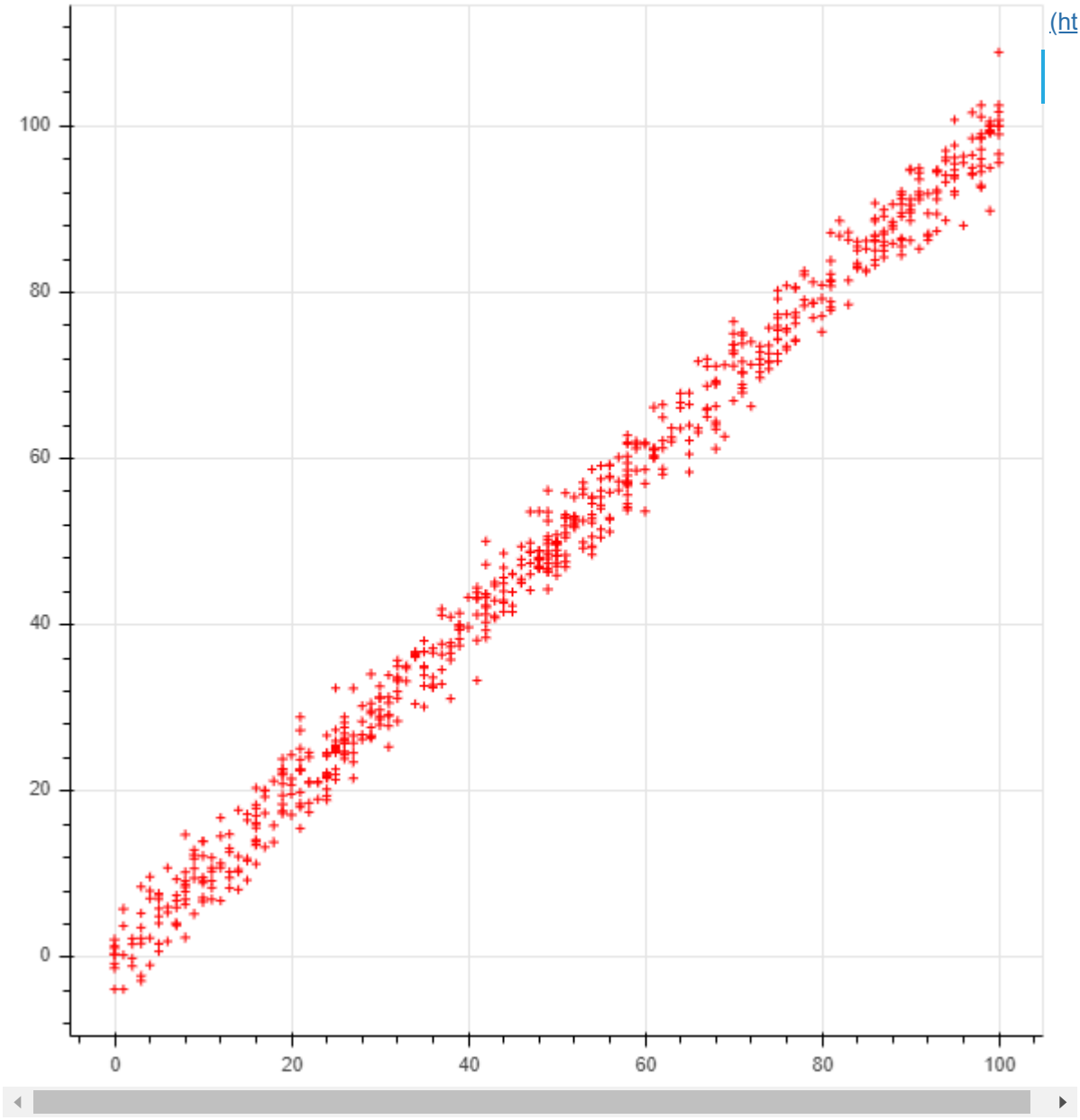
As x vs y is 0.99 the data is varying linearly

Reshaping the data

In [154]:
```
X = df['x'].values.reshape(-1,1)
Y = df['y'].values.reshape(-1,1)
```

Plotting the data

In [155]:
```
p = figure()
p.cross(X.flatten(),Y.flatten(),color="red",fill_alpha=0.2, size=5)
show(p)
```



Splitting the data in train and test dataset

In [156]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, rand
```

Step 3: Selecting Model - Linear Regression

In [157]:
```
regressor = LinearRegression()
```

Step 3.1: Training the Model

In [158]:
```
regressor.fit(X_train, Y_train)
```

Out[158]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fa
lse)
```

In [159]:
```
#To retrieve the intercept:
print(regressor.intercept_)
```

```
[0.0283478]
```

In [160]:
```
print(regressor.coef_)
```

```
[[0.99831656]]
```

Step 3.2: Evaluate the Model

In [161]:
```
regressor.score(X_test,Y_test)
```

Out[161]:
```
0.9916592070219102
```

In [162]:
```
print('Mean Absolute Error:', metrics.mean_absolute_error(Y_test, regressor.p
print('Mean Squared Error:', metrics.mean_squared_error(Y_test, regressor.pre
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(Y_test,
```

```
Mean Absolute Error: 2.2074272654412033
Mean Squared Error: 7.551490700376495
Root Mean Squared Error: 2.747997580125662
```

Step 3.3: Get predictions

In [163]:
```
Y_pred = regressor.predict(X_test)
df1 = pd.DataFrame({'Actual': Y_test.flatten(), 'Predicted': Y_pred.flatten()})
df1
```

Out[163]:

	Actual	Predicted
0	29.667360	28.979528
1	56.687188	57.930708
2	85.027790	83.886939
3	73.138500	75.900406
4	48.437538	53.937442
...	...	...
135	14.002263	16.001413
136	94.151492	96.865054
137	40.831821	42.955960
138	19.471008	18.996362
139	17.609462	18.996362

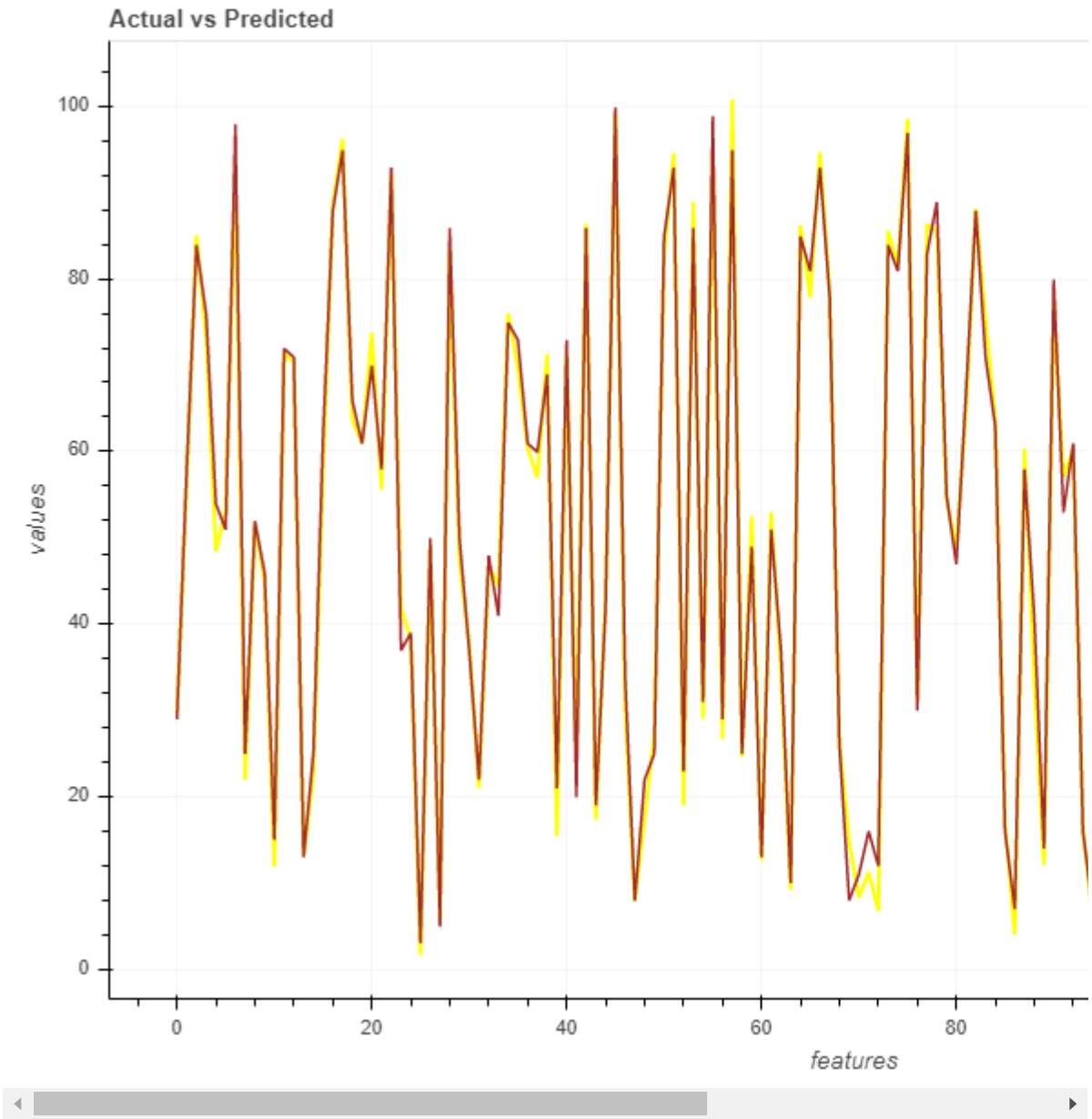
140 rows × 2 columns

In [164]:

```
p1 = figure(title = "Actual vs Predicted", width=900)
p1.grid.grid_line_alpha=0.3
p1.xaxis.axis_label = 'features'
p1.yaxis.axis_label = 'values'

p1.line(df1.index,df1['Actual'],color = 'yellow',line_width = 2,legend_label
p1.line(df1.index,df1['Predicted'] , color = 'brown',line_width = 1.5,legend_

show(p1)
```

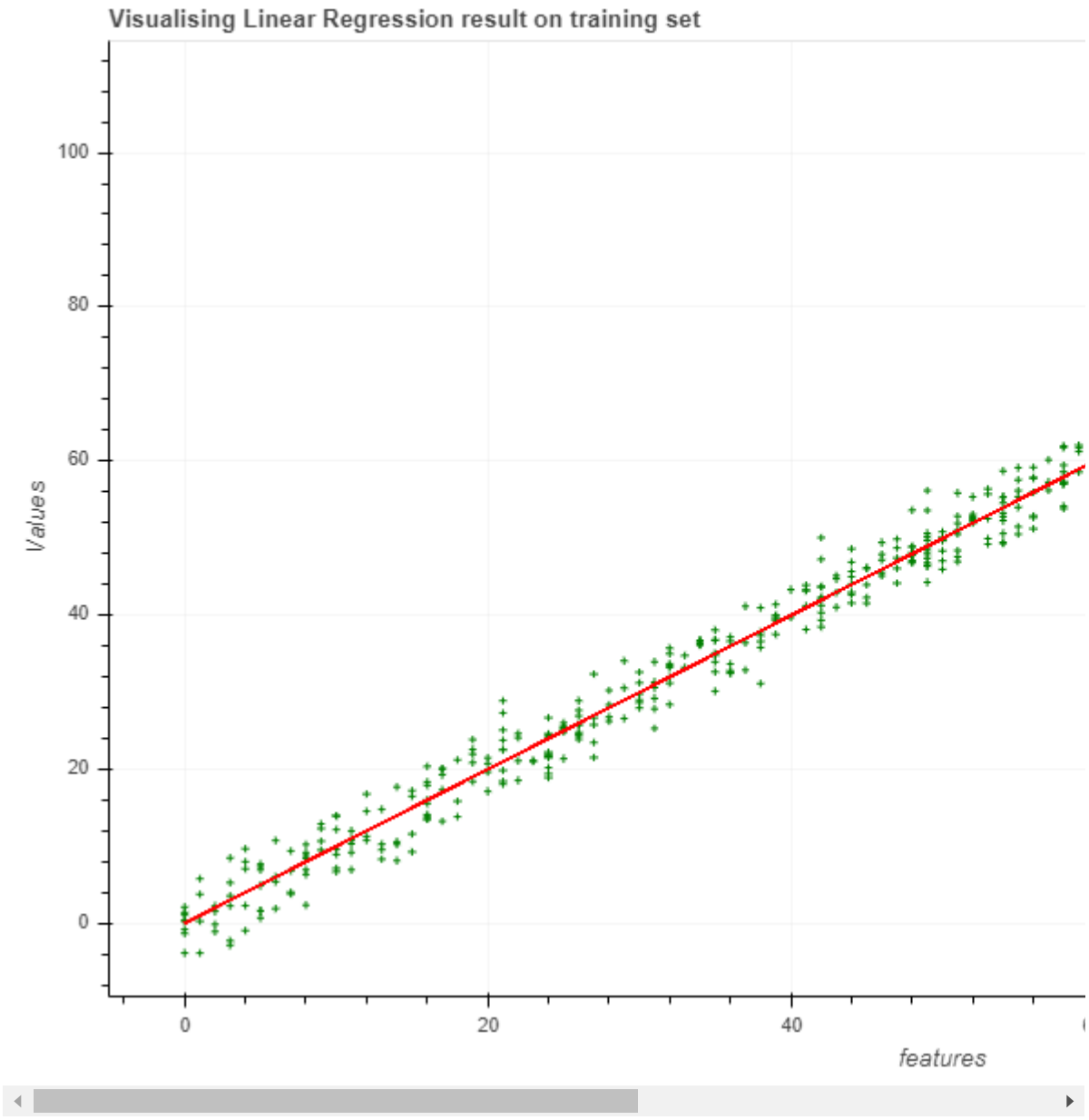


In [165]:
```
p1 = figure(title = "Visualising Linear Regression result on training set", w
p1.grid.grid_line_alpha=0.3
p1.xaxis.axis_label = 'features'
p1.yaxis.axis_label = 'Values'

p1.cross(X_train.flatten(),Y_train.flatten(), color='green')
p1.line(X_train.flatten(),(regressor.predict(X_train)).flatten(), color='red'



show(p1)
```

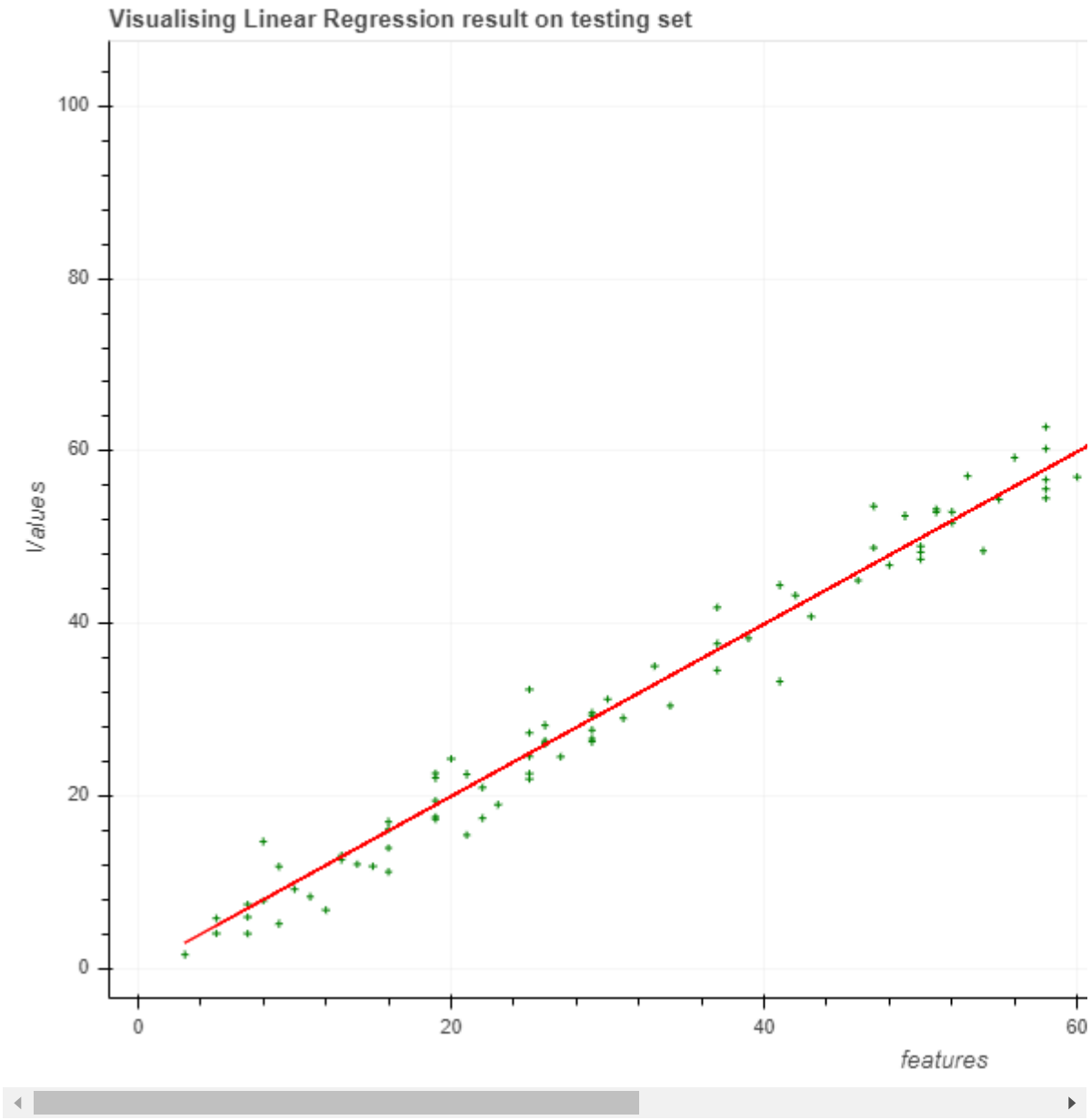


In [166]:
```
p1 = figure(title = "Visualising Linear Regression result on testing set", wi
p1.grid.grid_line_alpha=0.3
p1.xaxis.axis_label = 'features'
p1.yaxis.axis_label = 'Values'

p1.cross(X_test.flatten(),Y_test.flatten(), color='green')
p1.line(X_test.flatten(),(regressor.predict(X_test)).flatten(), color='red')



show(p1)
```



Step 4: Selecting Model - Polynomial Regression

In [167]:
```
def polyRegression(X_train,X_test,Y_train,Y_test,degree):
    polynom = PolynomialFeatures(degree=degree)
    X_polynom = polynom.fit_transform(X_train)
    ployRegr = LinearRegression()
    ployRegr.fit(X_polynom, Y_train)
    return ployRegr.score(polynom.fit_transform(X_test), Y_test)
```

Finding the maximum score of polynomial regression for degrees 2 to 100

In [168]:
```
lst=[]
for i in range(2,100):
    lst.append(polyRegression(X_train,X_test,Y_train,Y_test,i))
print("The maximum score is:",max(lst),"for degree:",lst.index(max(lst))+2)
```

```
The maximum score is: 0.9915964583277461 for degree: 2
```

We found the maximum score of polynomial regression is for degree 2

In [169]:
```
polynom = PolynomialFeatures(degree=2)
X_polynom = polynom.fit_transform(X_train)
ployRegr = LinearRegression()
```

Step 4.1: Training the Model

In [170]:
```
ployRegr.fit(X_polynom, Y_train) #training the algorithm
```

Out[170]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fa
lse)
```

Step 4.2: Evaluate the Model

In [171]:
```
ployRegr.score(polynom.fit_transform(X_test), Y_test)
```

Out[171]:
```
0.9915964583277461
```

In [172]:
```
print('Mean Absolute Error:', metrics.mean_absolute_error(Y_test, ployRegr.pr
print('Mean Squared Error:', metrics.mean_squared_error(Y_test, ployRegr.pred
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(Y_test,
```

```
Mean Absolute Error: 2.2287655433468787
Mean Squared Error: 7.608301387524093
Root Mean Squared Error: 2.758314954374154 3
```

Step 4.3: Get predictions

In [173]:
```
Y_predPoly = ployRegr.predict( polynom.fit_transform(X_test))
df2 = pd.DataFrame({'Actual': Y_test.flatten(), 'Predicted': Y_pred.flatten()
df2
```

Out[173]:

	Actual	Predicted
0	29.667360	28.979528
1	56.687188	57.930708
2	85.027790	83.886939
3	73.138500	75.900406
4	48.437538	53.937442
...	...	...
135	14.002263	16.001413
136	94.151492	96.865054
137	40.831821	42.955960
138	19.471008	18.996362
139	17.609462	18.996362

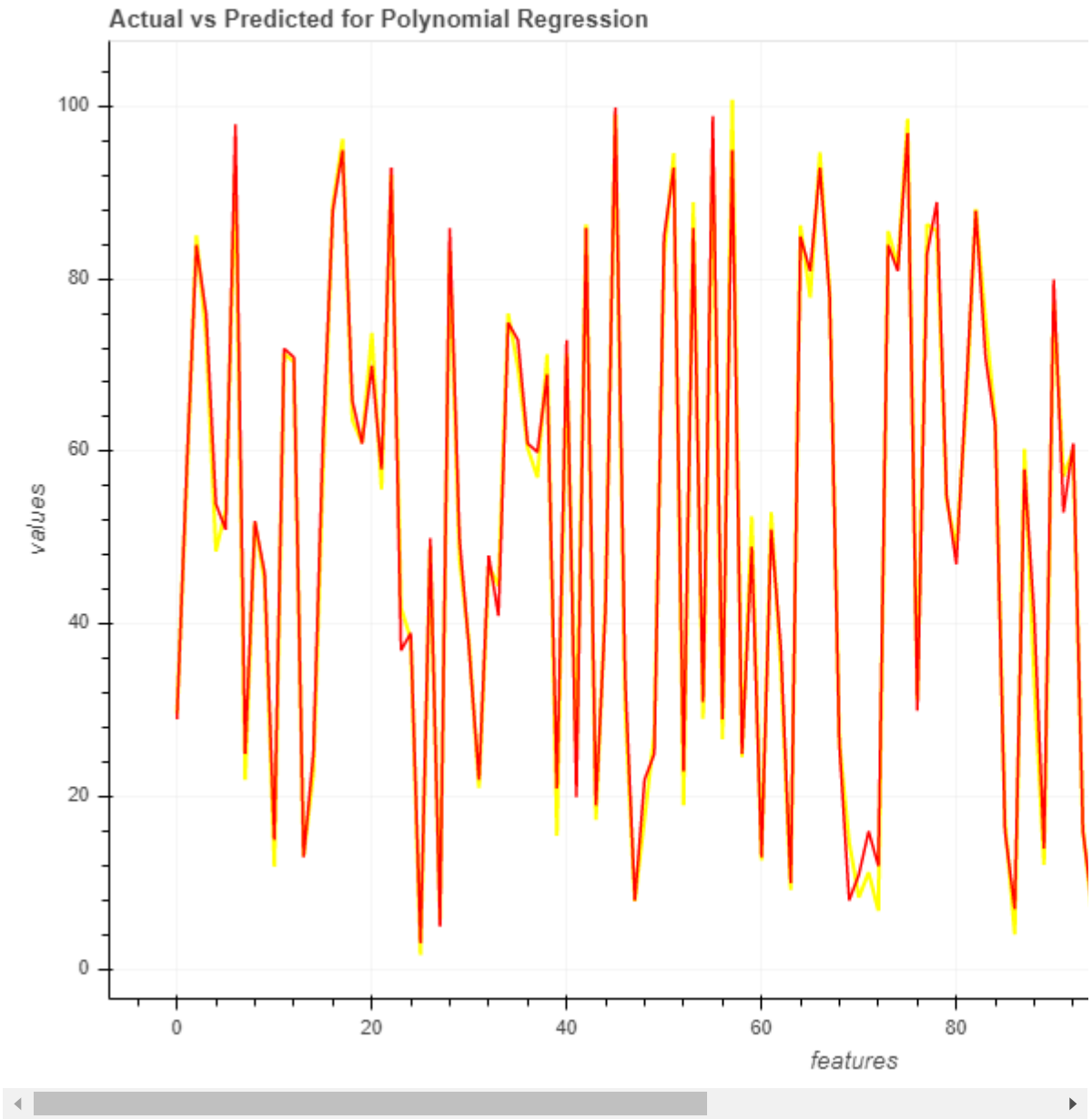
140 rows × 2 columns

In [174]:

```
p1 = figure(title = "Actual vs Predicted for Polynomial Regression", width=90
p1.grid.grid_line_alpha=0.3
p1.xaxis.axis_label = 'features'
p1.yaxis.axis_label = 'values'

p1.line(df2.index,df2['Actual'],color = 'yellow',line_width = 2,legend_label
p1.line(df2.index,df2['Predicted'] , color = 'red',line_width = 1.5,legend_la

show(p1)
```



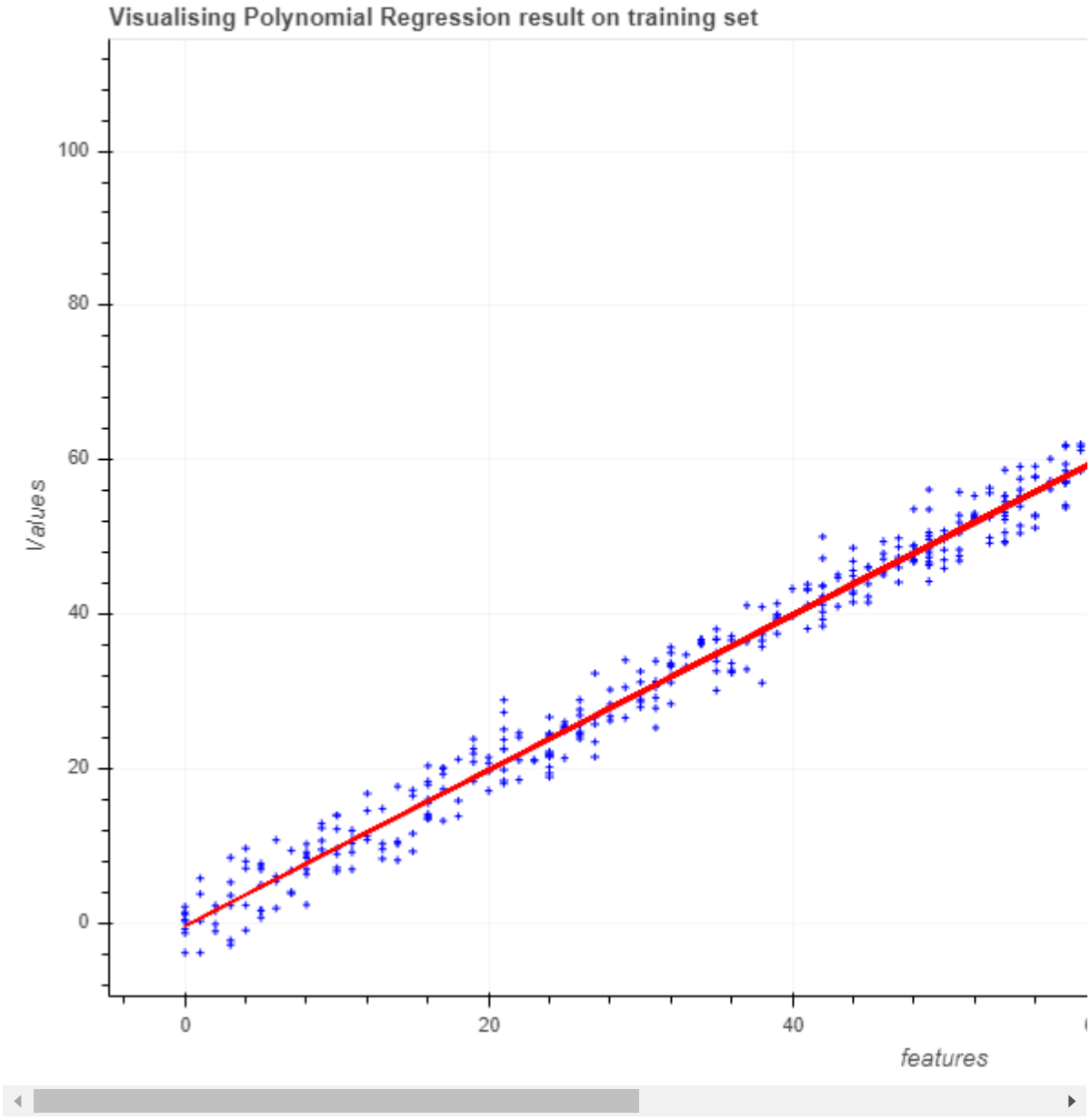
In [175]:

```
p1 = figure(title = "Visualising Polynomial Regression result on training set
p1.grid.grid_line_alpha=0.3
p1.xaxis.axis_label = 'features'
p1.yaxis.axis_label = 'Values'

p1.cross(X_train.flatten(),Y_train.flatten(), color='blue')
p1.line(X_train.flatten(),y = ployRegr.predict(X_polynom).flatten(), color='r



show(p1)
```

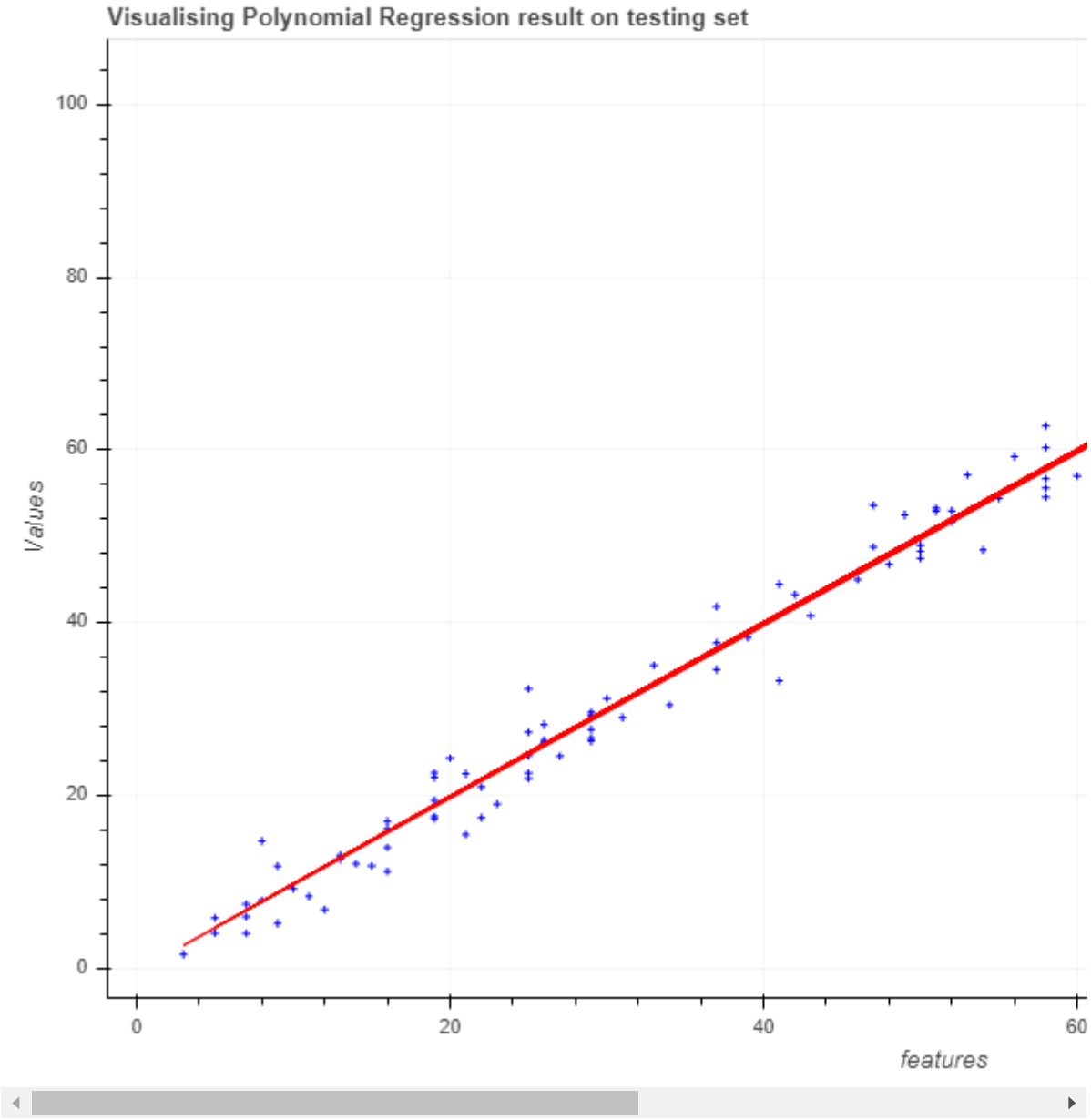


In [176]:
```
p1 = figure(title = "Visualising Polynomial Regression result on testing set"
p1.grid.grid_line_alpha=0.3
p1.xaxis.axis_label = 'features'
p1.yaxis.axis_label = 'Values'

p1.cross(X_test.flatten(),Y_test.flatten(), color='blue')
p1.line(X_test.flatten(),y = ployRegr.predict(polynom.fit_transform(X_test)).



show(p1)
```



Because the data is linearly varying, the graph of polynomial regression is also linear instead of being a a curve

Step 5: Selecting Model - Ridge Regression

In [177]:
```
ridgeRegr = RidgeCV(alphas=(1e-15,1e-10,1e-5,1e-2,0.1, 1.0, 5,10,20,30,40,50,
```

Step 5.1: Training the Model

In [178]:
```
ridgeRegr.fit(X_train,Y_train)
```

Out[178]:
```
RidgeCV(alphas=array([1.e-15, 1.e-10, 1.e-05, 1.e-02, 1.e-01, 1.e+00, 5.e+0
0, 1.e+01,
       2.e+01, 3.e+01, 4.e+01, 5.e+01, 6.e+01, 7.e+01, 8.e+01, 9.e+01,
       1.e+02]),
        cv=None, fit_intercept=True, gcv_mode=None, normalize=True,
        scoring=None, store_cv_values=False)
```

Step 5.2: Evaluate the Model

In [179]:
```
ridgeRegr.alpha_
```

Out[179]:
```
1e-05
```

In [180]:
```
ridgeRegr.score(ridgeRegr.predict(X_train),Y_train)
```

Out[180]:
```
0.9904033259765133
```

In [181]:
```
print('Mean Absolute Error:', metrics.mean_absolute_error(Y_test,  ridgeRegr.
print('Mean Squared Error:', metrics.mean_squared_error(Y_test,  ridgeRegr.pr
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(Y_test,
```

```
Mean Absolute Error: 2.207480868963139
Mean Squared Error: 7.551684850184807
Root Mean Squared Error: 2.74803290558625
```

Step 5.3: Get predictions

In [182]:
```
Y_predRidge = ridgeRegr.predict(X_test)
df3 = pd.DataFrame({'Actual': Y_test.flatten(), 'Predicted': Y_predRidge.flat
df3
```

Out[182]:

	Actual	Predicted
0	29.667360	28.979731
1	56.687188	57.930622
2	85.027790	83.886593
3	73.138500	75.900141
4	48.437538	53.937396
...	...	...
135	14.002263	16.001746
136	94.151492	96.864579
137	40.831821	42.956023
138	19.471008	18.996666
139	17.609462	18.996666

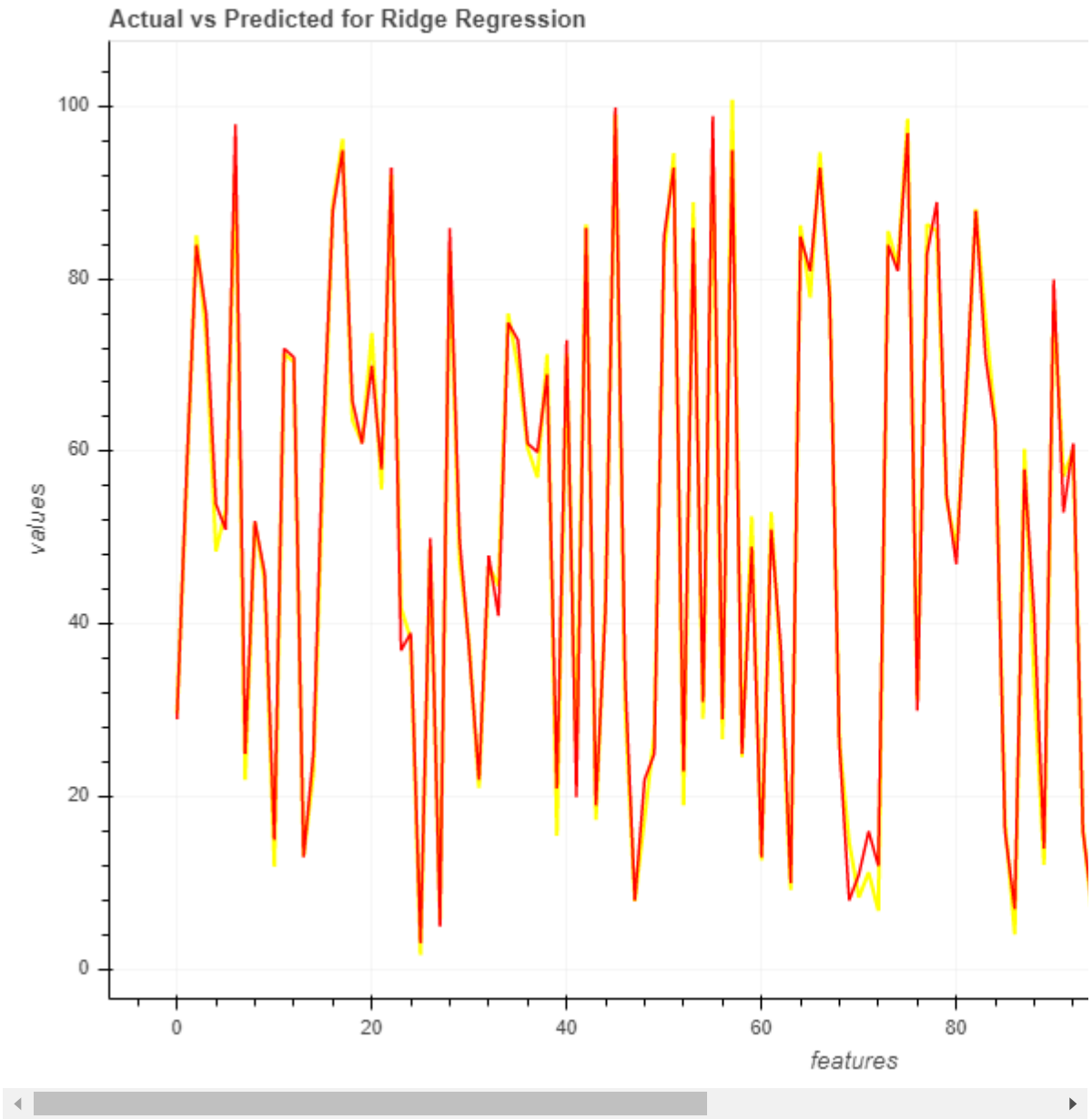
140 rows × 2 columns

In [183]:

```
p1 = figure(title = "Actual vs Predicted for Ridge Regression", width=900)
p1.grid.grid_line_alpha=0.3
p1.xaxis.axis_label = 'features'
p1.yaxis.axis_label = 'values'

p1.line(df3.index,df3['Actual'],color = 'yellow',line_width = 2,legend_label
p1.line(df3.index,df3['Predicted'] , color = 'red',line_width = 1.5,legend_la

show(p1)
```



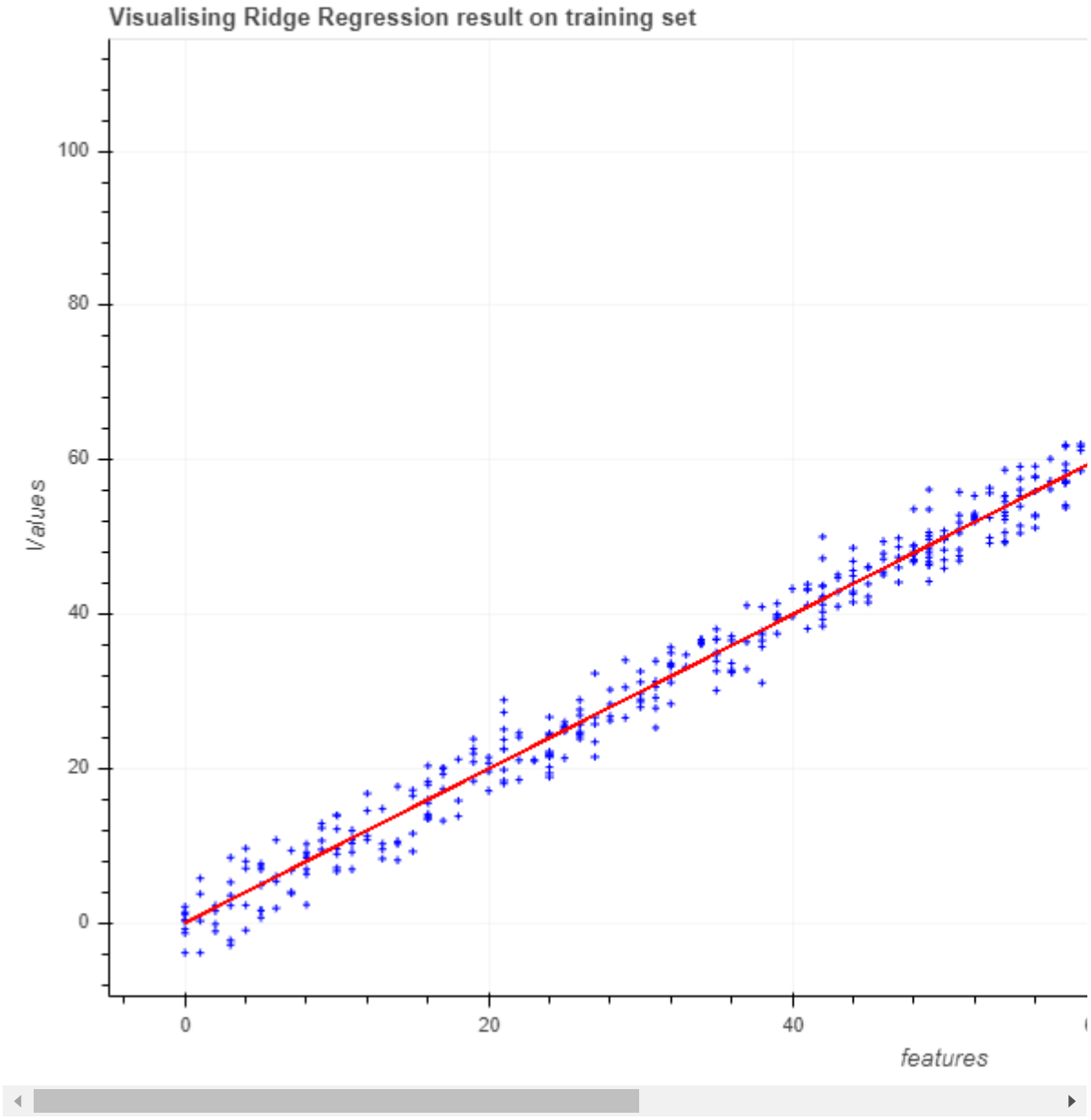
In [184]:

```
p1 = figure(title = "Visualising Ridge Regression result on training set", wi
p1.grid.grid_line_alpha=0.3
p1.xaxis.axis_label = 'features'
p1.yaxis.axis_label = 'Values'

p1.cross(X_train.flatten(),Y_train.flatten(), color='blue')
p1.line(X_train.flatten(),y = ridgeRegr.predict(X_train).flatten(), color='re



show(p1)
```

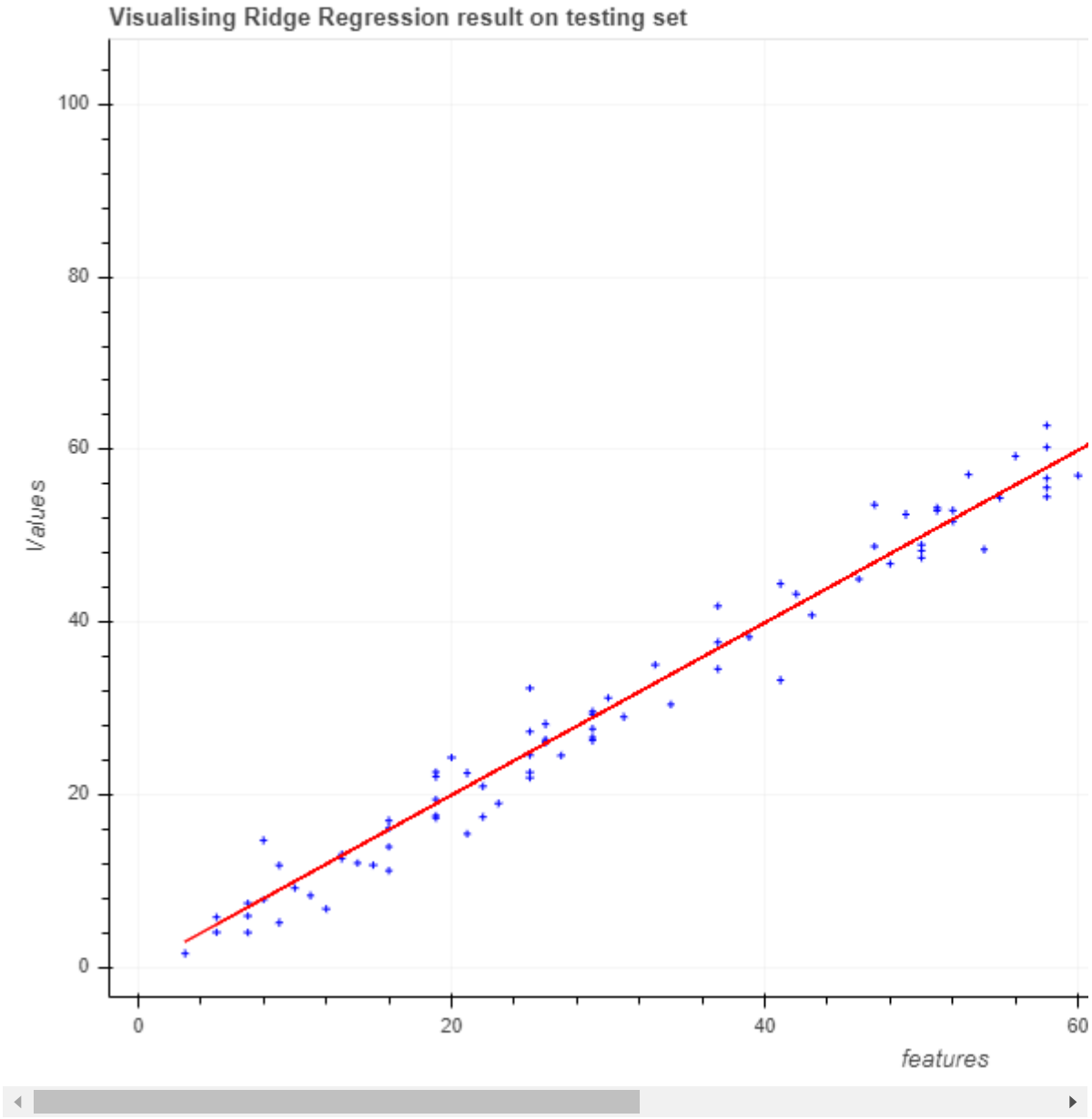


In [185]:
```
p1 = figure(title = "Visualising Ridge Regression result on testing set", wid
p1.grid.grid_line_alpha=0.3
p1.xaxis.axis_label = 'features'
p1.yaxis.axis_label = 'Values'

p1.cross(X_test.flatten(),Y_test.flatten(), color='blue')
p1.line(X_test.flatten(),y =  ridgeRegr.predict(X_test).flatten(), color='red



show(p1)
```



Comparing the three models

In [186]:
```
print("Score of Linear Regression Model",regressor.score(X_test,Y_test))
print("Score of Polynomial Regression Model",ployRegr.score(polynom.fit_trans
print("Score of Ridge Regression Model",ridgeRegr.score(ridgeRegr.predict(X_t
```

```
Score of Linear Regression Model 0.9916592070219102
Score of Polynomial Regression Model 0.9915964583277461
Score of Ridge Regression Model 0.9904033259765133
```

Final Insight:

As in the given dataset, data was linear varying so Linear Regression gave the best score of all the 3 Models

In [ ]:
```

```