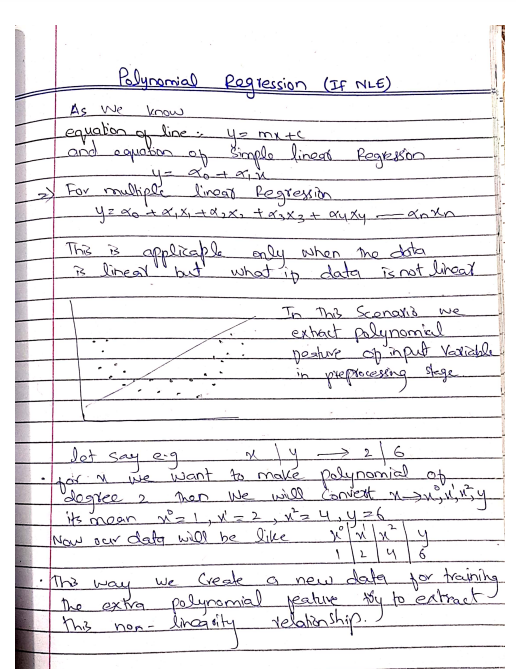
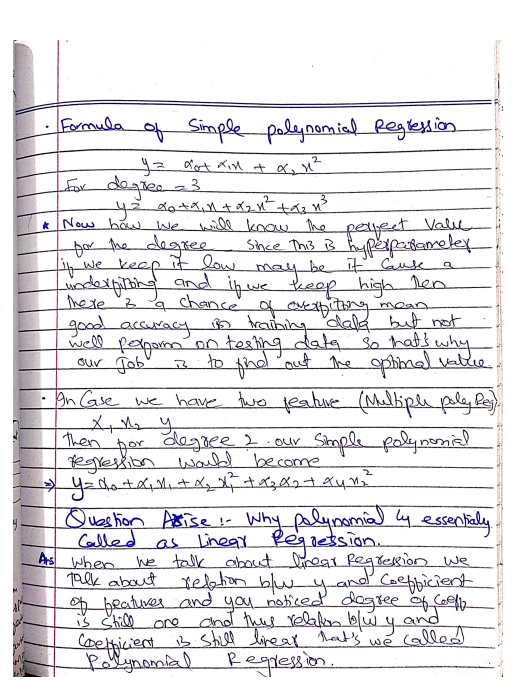
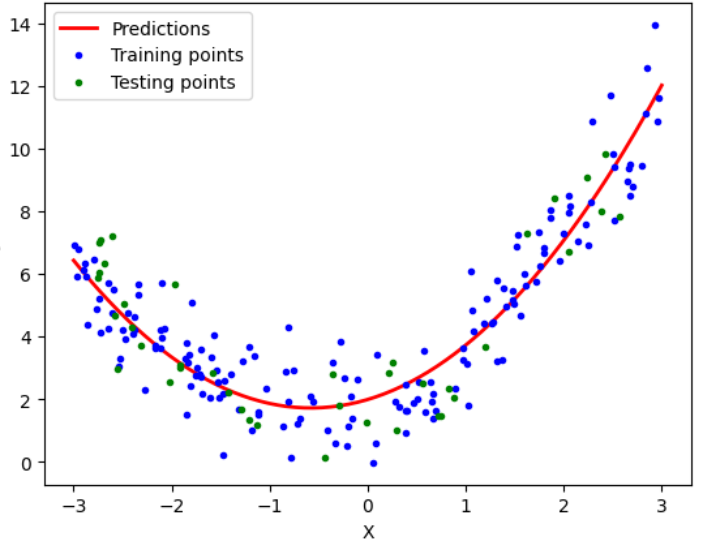
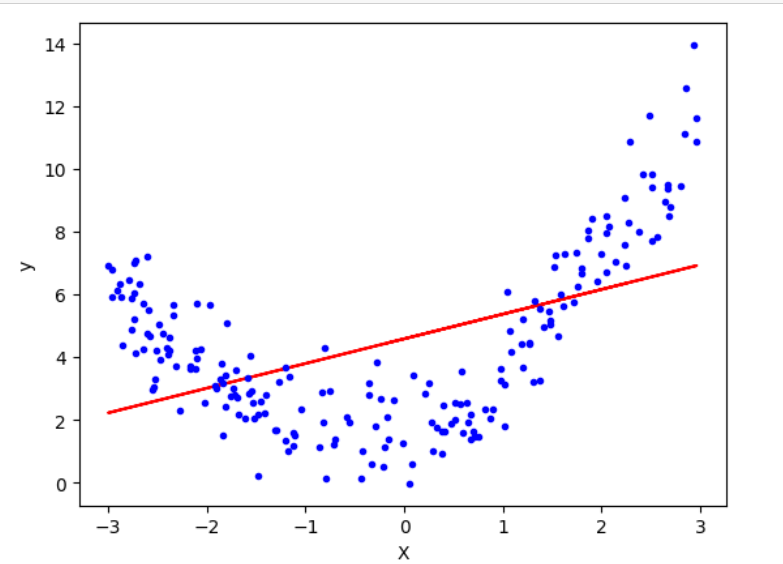
**Polynomial Regression (Simple & Multiple)**

**Explanations:**

****

****

**Difference Between Linear Regression & Polynomial Regression**



**Code:**

import numpy as np

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression,SGDRegressor

from sklearn.preprocessing import PolynomialFeatures,StandardScaler

from sklearn.metrics import r2\_score

from sklearn.pipeline import Pipeline

X = 6 \* np.random.rand(200, 1) - 3

y = 0.8 \* X\*\*2 + 0.9 \* X + 2 + np.random.randn(200, 1)

# y = 0.8x^2 + 0.9x + 2

plt.plot(X, y,'b.')

plt.xlabel("X")

plt.ylabel("y")

plt.show()

# Train test split

X\_train,X\_test,y\_train,y\_test = train\_test\_split(X,y,test\_size=0.2,random\_state=2)

# Applying linear regression

lr = LinearRegression()

lr.fit(X\_train,y\_train)

y\_pred = lr.predict(X\_test)

r2\_score(y\_test,y\_pred)

plt.plot(X\_train,lr.predict(X\_train),color='r')

plt.plot(X, y, "b.")

plt.xlabel("X")

plt.ylabel("y")

plt.show()

# Applying Polynomial Linear Regression

# degree 2

poly = PolynomialFeatures(degree=2,include\_bias=True)

X\_train\_trans = poly.fit\_transform(X\_train)

X\_test\_trans = poly.transform(X\_test)

print(X\_train[0])

print(X\_train\_trans[0])

lr = LinearRegression()

lr.fit(X\_train\_trans,y\_train)

y\_pred = lr.predict(X\_test\_trans)

r2\_score(y\_test,y\_pred)

print(lr.coef\_)

print(lr.intercept\_)

X\_new=np.linspace(-3, 3, 200).reshape(200, 1)

X\_new\_poly = poly.transform(X\_new)

y\_new = lr.predict(X\_new\_poly)

plt.plot(X\_new, y\_new, "r-", linewidth=2, label="Predictions")

plt.plot(X\_train, y\_train, "b.",label='Training points')

plt.plot(X\_test, y\_test, "g.",label='Testing points')

plt.xlabel("X")

plt.ylabel("y")

plt.legend()

plt.show()

def Apna\_Polynomial(degree):

X\_new=np.linspace(-3, 3, 100).reshape(100, 1)

X\_new\_poly = poly.transform(X\_new)

polybig\_features = PolynomialFeatures(degree=degree, include\_bias=False)

std\_scaler = StandardScaler()

lin\_reg = LinearRegression()

Apna\_Polynomial = Pipeline([

("poly\_features", polybig\_features),

("std\_scaler", std\_scaler),

("lin\_reg", lin\_reg),

])

Apna\_Polynomial.fit(X, y)

y\_newbig = Apna\_Polynomial.predict(X\_new)

plt.plot(X\_new, y\_newbig,'r', label="Degree " + str(degree), linewidth=2)

plt.plot(X\_train, y\_train, "b.", linewidth=3)

plt.plot(X\_test, y\_test, "g.", linewidth=3)

plt.legend(loc="upper left")

plt.xlabel("X")

plt.ylabel("y")

plt.axis([-3, 3, 0, 10])

plt.show()

Apna\_Polynomial(350)

# 3D polynomial regression

x = 7 \* np.random.rand(100, 1) - 2.8

y = 7 \* np.random.rand(100, 1) - 2.8

z = x\*\*2 + y\*\*2 + 0.2\*x + 0.2\*y + 0.1\*x\*y +2 + np.random.randn(100, 1)

# z = x^2 + y^2 + 0.2x + 0.2y + 0.1xy + 2

import plotly.express as px

df = px.data.iris()

fig = px.scatter\_3d(df, x=x.ravel(), y=y.ravel(), z=z.ravel())

fig.show()

lr = LinearRegression()

lr.fit(np.array([x,y]).reshape(100,2),z)

x\_input = np.linspace(x.min(), x.max(), 10)

y\_input = np.linspace(y.min(), y.max(), 10)

xGrid, yGrid = np.meshgrid(x\_input,y\_input)

final = np.vstack((xGrid.ravel().reshape(1,100),yGrid.ravel().reshape(1,100))).T

z\_final = lr.predict(final).reshape(10,10)

import plotly.graph\_objects as go

fig = px.scatter\_3d(df, x=x.ravel(), y=y.ravel(), z=z.ravel())

fig.add\_trace(go.Surface(x = x\_input, y = y\_input, z =z\_final ))

fig.show()

X\_multi = np.array([x,y]).reshape(100,2)

X\_multi.shape

poly = PolynomialFeatures(degree=30)

X\_multi\_trans = poly.fit\_transform(X\_multi)

print("Input",poly.n\_features\_in\_)

print("Ouput",poly.n\_output\_features\_)

print("Powers\n",poly.powers\_)

X\_multi\_trans.shape

lr = LinearRegression()

lr.fit(X\_multi\_trans,z)

X\_test\_multi = poly.transform(final)

z\_final = lr.predict(X\_multi\_trans).reshape(10,10)

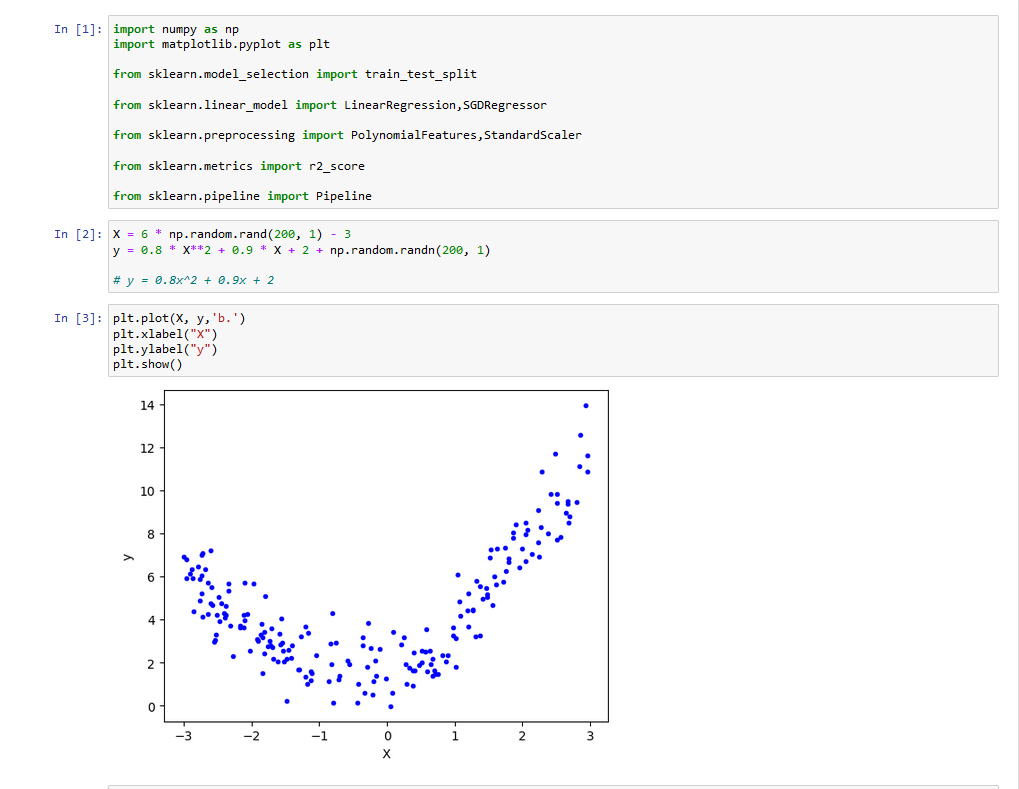
fig = px.scatter\_3d(x=x.ravel(), y=y.ravel(), z=z.ravel())

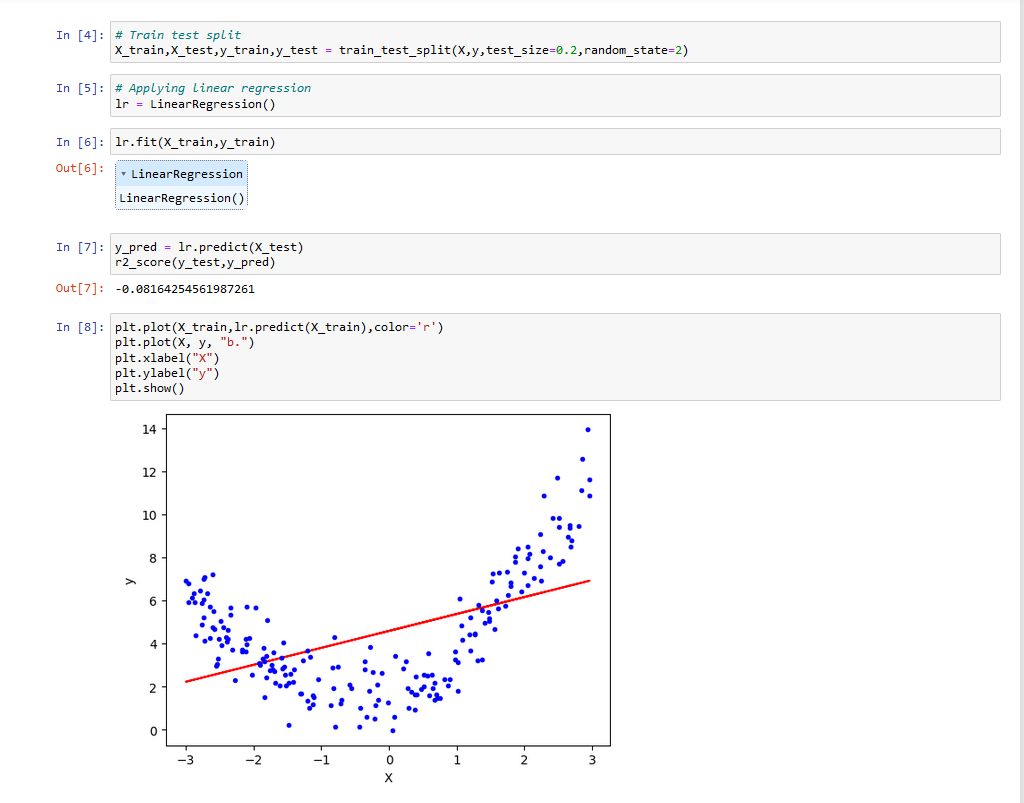
fig.add\_trace(go.Surface(x = x\_input, y = y\_input, z =z\_final))

fig.update\_layout(scene = dict(zaxis = dict(range=[0,35])))

fig.show()

**Screenshot:**

****

****

