```
import pandas as pd
import numpy as np
import seaborn as sns



from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt

data=pd.read_csv("/content/winequality-red - winequality-red.csv.csv")


data.head()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH
0	7.4	0.70	0.00	1.9	0.076	11.0	34	0.9978	3.51
1	7.8	0.88	0.00	2.6	0.098	25.0	67	0.9968	3.20
2	7.8	0.76	0.04	2.3	0.092	15.0	54	0.9970	3.26
3	11.2	0.28	0.56	1.9	0.075	17.0	60	0.9980	3.16

```
data.shape
```

```
(399, 12)
```

```
data.iloc[0:300
          ]
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	p
0	7.4	0.70	0.00	1.9	0.076	11.0	34	0.99780	3.5
1	7.8	0.88	0.00	2.6	0.098	25.0	67	0.99680	3.2
2	7.8	0.76	0.04	2.3	0.092	15.0	54	0.99700	3.2
3	11.2	0.28	0.56	1.9	0.075	17.0	60	0.99800	3.1
4	7.4	0.70	0.00	1.9	0.076	11.0	34	0.99780	3.5
...	...	...	...	...	...	...	...	...	.
295	10.8	0.50	0.46	2.5	0.073	5.0	27	1.00010	3.0
296	10.6	0.83	0.37	2.6	0.086	26.0	70	0.99810	3.1
297	7.1	0.63	0.06	2.0	0.083	8.0	29	0.99855	3.6
298	7.2	0.65	0.02	2.3	0.094	5.0	31	0.99930	3.6
299	6.9	0.67	0.06	2.1	0.080	8.0	33	0.99845	3.6

```
# Assuming 'data' is your DataFrame object
data.dropna(axis=1, how='all', inplace=True)



data.describe()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	t su dio

```
data.isnull().sum()
```

```
fixed acidity           0
volatile acidity        0
citric acid             0
residual sugar          0
chlorides               0
free sulfur dioxide     0
total sulfur dioxide    0
density                 0
pH                      0
sulphates               0
alcohol                 0
quality                 0
dtype: int64
```

```
data.head()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH
0	7.4	0.70	0.00	1.9	0.076	11.0	34	0.9978	3.51
1	7.8	0.88	0.00	2.6	0.098	25.0	67	0.9968	3.20
2	7.8	0.76	0.04	2.3	0.092	15.0	54	0.9970	3.26
3	11.2	0.28	0.56	1.9	0.075	17.0	60	0.9980	3.16

```
data.shape
```

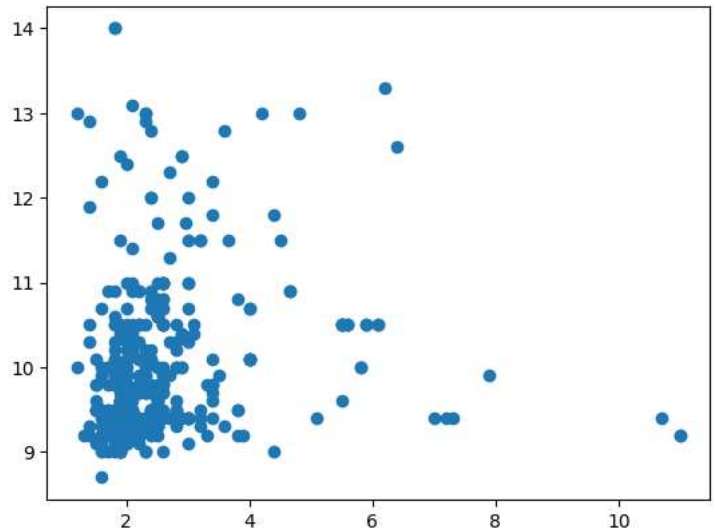
```
(399, 12)
```

```
data.iloc[0:50]
```

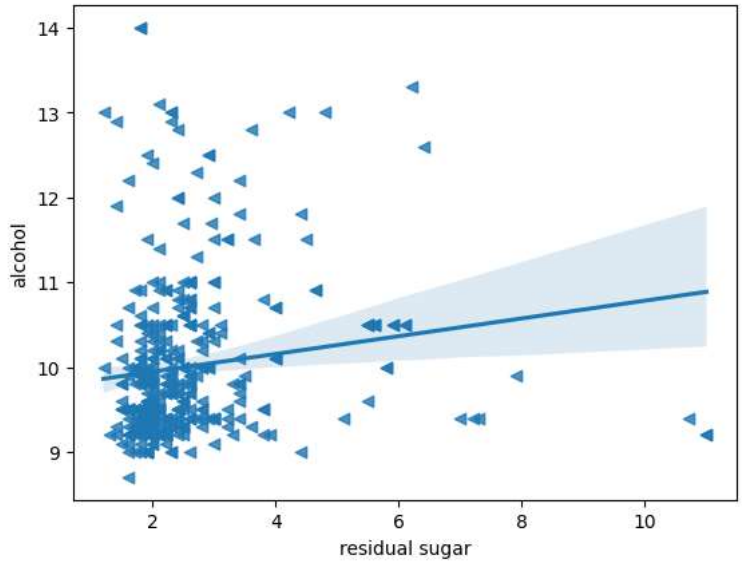
18	7.4	0.590	0.08	4.4	0.086	6.0	29	0.9974	3
19	7.9	0.320	0.51	1.8	0.341	17.0	56	0.9969	3
20	8.9	0.220	0.48	1.8	0.077	29.0	60	0.9968	3
21	7.6	0.390	0.31	2.3	0.082	23.0	71	0.9982	3
22	7.9	0.430	0.21	1.6	0.106	10.0	37	0.9966	3
23	8.5	0.490	0.11	2.3	0.084	9.0	67	0.9968	3
24	6.9	0.400	0.14	2.4	0.085	21.0	40	0.9968	3
25	6.3	0.390	0.16	1.4	0.080	11.0	23	0.9955	3
26	7.6	0.410	0.24	1.8	0.080	4.0	11	0.9962	3
27	7.9	0.430	0.21	1.6	0.106	10.0	37	0.9966	3
28	7.1	0.710	0.00	1.9	0.080	14.0	35	0.9972	3
29	7.8	0.645	0.00	2.0	0.082	8.0	16	0.9964	3
30	6.7	0.675	0.07	2.4	0.089	17.0	82	0.9958	3
31	6.9	0.685	0.00	2.5	0.105	22.0	37	0.9966	3
32	8.3	0.655	0.12	2.3	0.083	15.0	113	0.9966	3
33	6.9	0.605	0.12	10.7	0.073	40.0	83	0.9993	3
34	5.2	0.320	0.25	1.8	0.103	13.0	50	0.9957	3
35	7.8	0.645	0.00	5.5	0.086	5.0	18	0.9986	3
36	7.8	0.600	0.14	2.4	0.086	3.0	15	0.9975	3
37	8.1	0.380	0.28	2.1	0.066	13.0	30	0.9968	3
38	5.7	1.130	0.09	1.5	0.172	7.0	19	0.9940	3
39	7.3	0.450	0.36	5.9	0.074	12.0	87	0.9978	3
40	7.3	0.450	0.36	5.9	0.074	12.0	87	0.9978	3
41	8.8	0.610	0.30	2.8	0.088	17.0	46	0.9976	3
42	7.5	0.490	0.20	2.6	0.332	8.0	14	0.9968	3
43	8.1	0.660	0.22	2.2	0.069	9.0	23	0.9968	3
44	6.8	0.670	0.02	1.8	0.050	5.0	11	0.9962	3
45	4.6	0.520	0.15	2.1	0.054	8.0	65	0.9934	3
46	7.7	0.935	0.43	2.2	0.114	22.0	114	0.9970	3
47	8.7	0.290	0.52	1.6	0.113	12.0	37	0.9969	3

```
x=data["residual sugar"]
y=data["alcohol"]

plt.plot(x,y,'o')
```

```
[<matplotlib.lines.Line2D at 0x7e78dad14280>]
```



```
sns.regplot(x=x,y=y,data=data,marker='<')
plt.show()
```




```
type(x)
```

```
pandas.core.series.Series
```

```
x.shape
```

```
(399,)
```

```
type(y)
```

```
pandas.core.series.Series
```

```
y.shape
```

```
(399,)
```

```
data.iloc[0:100]
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH
0	7.4	0.700	0.00	1.9	0.076	11.0	34	0.9978	3.51
1	7.8	0.880	0.00	2.6	0.098	25.0	67	0.9968	3.20
2	7.8	0.760	0.04	2.3	0.092	15.0	54	0.9970	3.26
3	11.2	0.280	0.56	1.9	0.075	17.0	60	0.9980	3.16
4	7.4	0.700	0.00	1.9	0.076	11.0	34	0.9978	3.51
...	...	...	...	...	...	...	...	...	...
95	4.7	0.600	0.17	2.3	0.058	17.0	106	0.9932	3.85
96	6.8	0.775	0.00	3.0	0.102	8.0	23	0.9965	3.45
97	7.0	0.500	0.25	2.0	0.070	3.0	22	0.9963	3.25
98	7.6	0.900	0.06	2.5	0.079	5.0	10	0.9967	3.39
99	8.1	0.545	0.18	1.9	0.080	13.0	35	0.9972	3.30

```
x_array = x.to_numpy()
x=x_array.reshape(399,1)
```

```
x_train, x_test, y_train, y_test = train_test_split(x, y,test_size=0.25)
```

```
print(f"x Training dataset: {x_train.shape}")
print(f"y Training dataset: {y_train.shape}")
print(f"x test dataset: {x_test.shape}")
print(f"y test dataset: {y_test.shape}")
```

```
x Training dataset: (299, 1)
y Training dataset: (299,)
x test dataset: (100, 1)
y test dataset: (100,)
```

```
from sklearn.linear_model import LinearRegression
from sklearn.impute import SimpleImputer

# Assuming you have 'x_train' and 'y_train' defined
imputer = SimpleImputer(strategy='mean')
x_train_imputed = imputer.fit_transform(x_train)

model = LinearRegression()
model.fit(x_train_imputed, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
model.coef_
```

```
array([0.10372951])
```

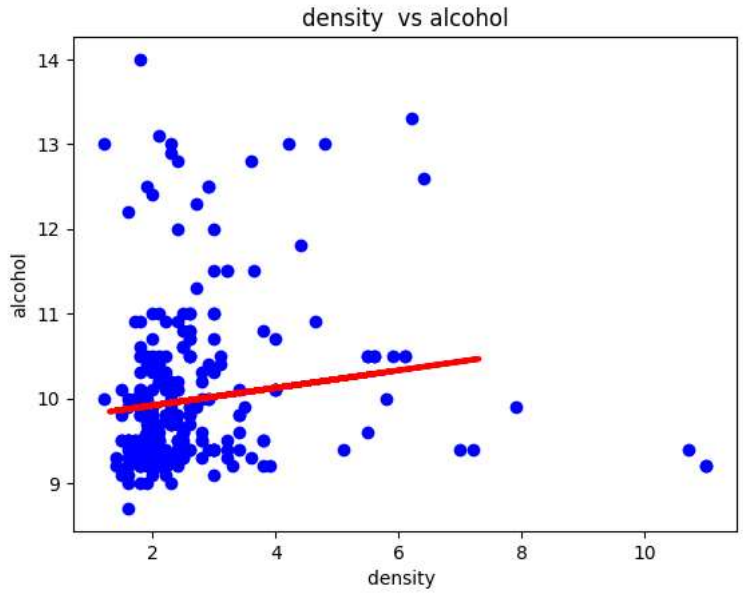
```
model.intercept_
```

```
9.707489856685877
```

```
y_pred = model.predict(x_test)
y_pred.shape
```

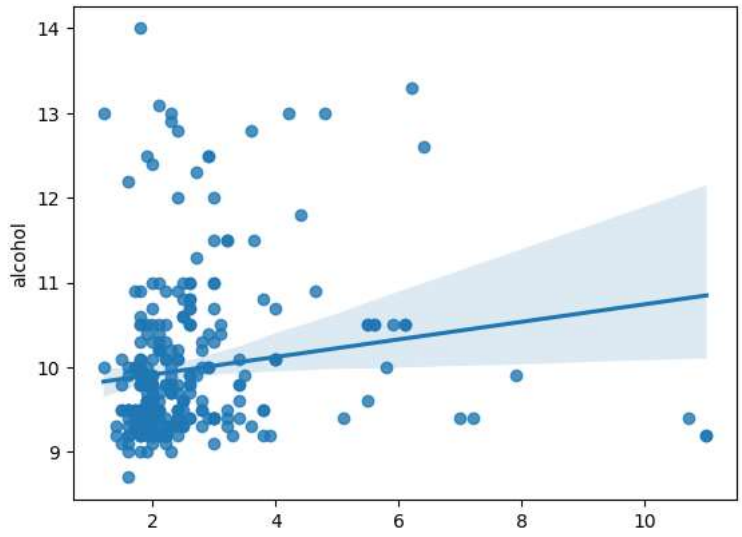
```
(100,)
```

```
plt.scatter(x_train, y_train, color='blue')
plt.plot(x_test, y_pred, color='red', linewidth=3)
plt.title("density  vs alcohol")
plt.xlabel("density ")
plt.ylabel("alcohol")
plt.show()
```



```
sns.regplot(data=data,x=x_train,y=y_train,)
```

```
<Axes: ylabel='alcohol'>
```



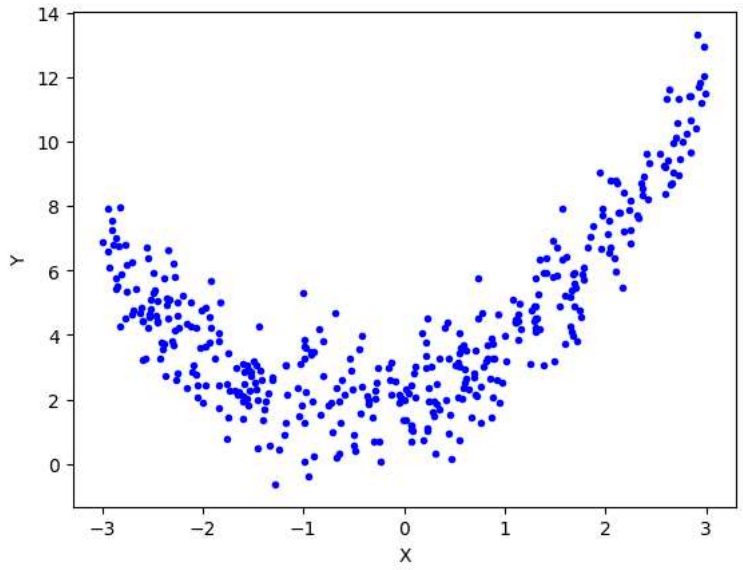

```
import numpy as np

nan_indices = np.isnan(y_pred)
has_nan = nan_indices.any()

if has_nan:
    print("y_pred contains NaN values.")
```

```
import numpy as np
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# Check for NaN values in y_pred
nan_indices = np.isnan(y_pred)
has_nan = nan_indices.any()

if has_nan:
    print("y_pred contains NaN values. Removing corresponding rows...")
    y_test = y_test[~nan_indices]
    y_pred = y_pred[~nan_indices]

# Check for NaN values in y_test after removing corresponding rows
nan_indices = np.isnan(y_test)
has_nan = nan_indices.any()

if has_nan:
    print("y_test contains NaN values. Removing corresponding rows...")
    y_test = y_test[~nan_indices]
    y_pred = y_pred[~nan_indices]

# Calculate evaluation metrics
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"MSE: {mse}")
print(f"MAE: {mae}")
print(f"R-Square: {r2}")
```

```
MSE: 0.9458092867624106
MAE: 0.7106866516634475
R-Square: 0.0009735745819978714
```

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import r2_score
```

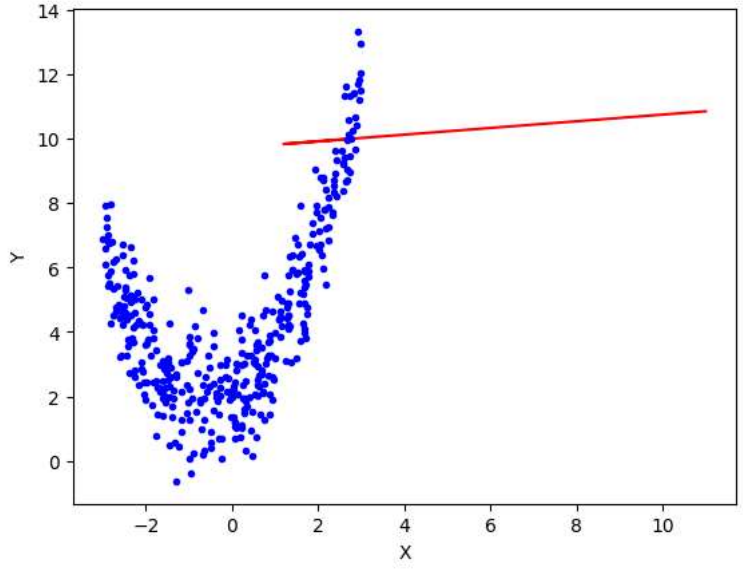
```
import numpy as np
import matplotlib.pyplot as plt
X = 6 * np.random.rand(399, 1) - 3
y = 0.8 * X**2 + 0.9*X + 2 + np.random.randn(399, 1)
#equation used -> y = 0.8x^2 + 0.9x + 2
#visualize the data
plt.plot(X, y, 'b.')
plt.xlabel("X")
plt.ylabel("Y")
plt.show()
```




```
lr = LinearRegression()
lr.fit(x_train, y_train)
y_pred = lr.predict(x_test)
print(r2_score(y_test, y_pred))
```

```
0.0009735745819978714
```

```
plt.plot(x_train, lr.predict(x_train), color="r")
plt.plot(X, y, "b.")
plt.xlabel("X")
plt.ylabel("Y")
plt.show()
```




```
poly = PolynomialFeatures(degree=2, include_bias=True)
x_train_trans = poly.fit_transform(x_train)
x_test_trans = poly.transform(x_test)
```

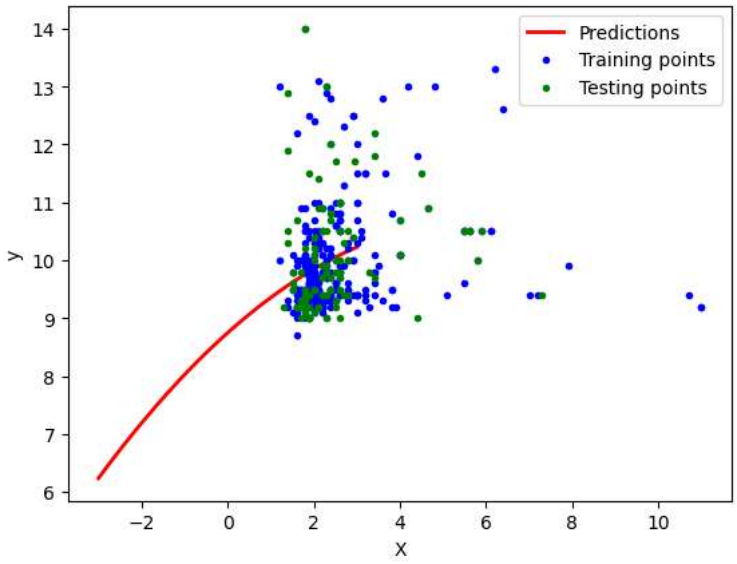
```
#include bias parameter
lr = LinearRegression()
lr.fit(x_train_trans, y_train)
y_pred = lr.predict(x_test_trans)
print(r2_score(y_test, y_pred))
```

```
0.00406409082852377
```

```
print(lr.coef_)
print(lr.intercept_)
```

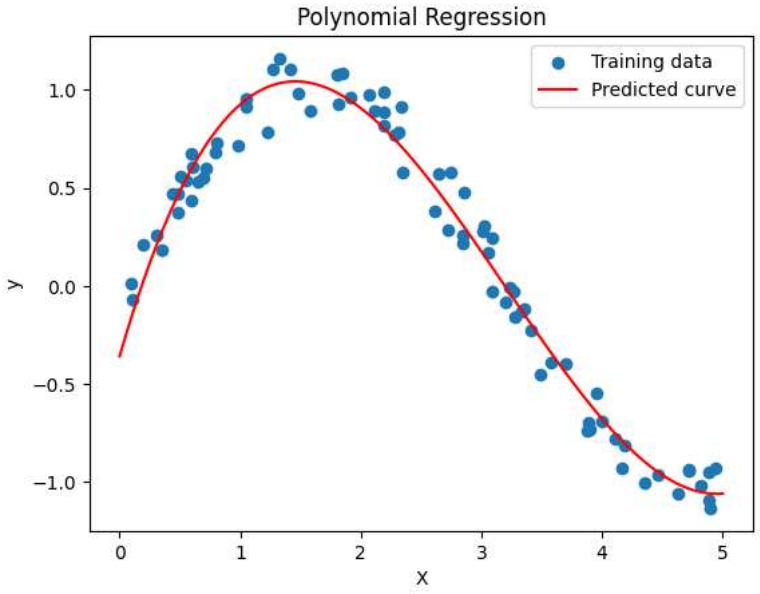
```
[ 0.          0.66533594 -0.05737749]
8.746798489342982
```

```
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()
```



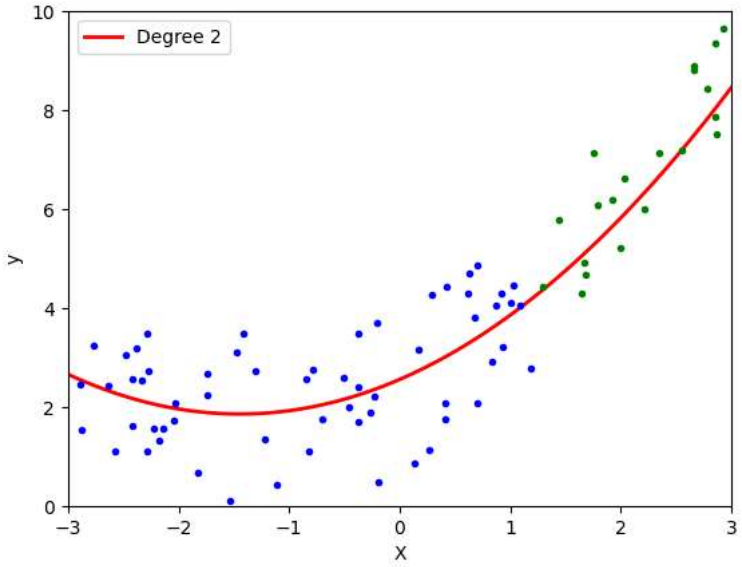

```
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline
import matplotlib.pyplot as plt

# Generate some sample data
np.random.seed(0)
X = np.sort(5 * np.random.rand(80, 1), axis=0)
y = np.sin(X).ravel() + np.random.normal(0, 0.1, X.shape[0])

# Create a pipeline for polynomial regression
degree = 3  # You can set the degree of the polynomial
model = make_pipeline(PolynomialFeatures(degree), LinearRegression())

# Fit the model to the data
model.fit(X, y)

# Generate new data points for prediction
X_new = np.linspace(0, 5, 100)[:, np.newaxis]

# Predict using the fitted model
y_new = model.predict(X_new)

# Plot the results
plt.scatter(X, y, label='Training data')
```

```
plt.plot(X_new, y_new, label='Predicted curve', color='r')
plt.legend()
plt.xlabel('X')
plt.ylabel('y')
plt.title('Polynomial Regression')
plt.show()
```




```
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
import numpy as np
import matplotlib.pyplot as plt

def polynomial_regression(degree, X, y):
    X_new = np.linspace(-3, 3, 100).reshape(100, 1)
    polybig_features = PolynomialFeatures(degree=degree, include_bias=False)
    std_scaler = StandardScaler()
    lin_reg = LinearRegression()
    polynomial_regression = Pipeline([
        ("poly_features", polybig_features),
        ("std_scaler", std_scaler),
        ("lin_reg", lin_reg),
    ])
    polynomial_regression.fit(X, y)
    y_newbig = polynomial_regression.predict(X_new)
    return X_new, y_newbig

# Generate some sample data
np.random.seed(0)
X = np.sort(6 * np.random.rand(80, 1) - 3, axis=0)
y = 0.5 * X**2 + X + 2 + np.random.randn(80, 1)

x_train = X[:60]
y_train = y[:60]
x_test = X[60:]
y_test = y[60:]

# Degree of the polynomial
degree = 2

X_new, y_newbig = polynomial_regression(degree, x_train, y_train)

# Plotting prediction line
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()
```



✓ 0s completed at 10:25 AM