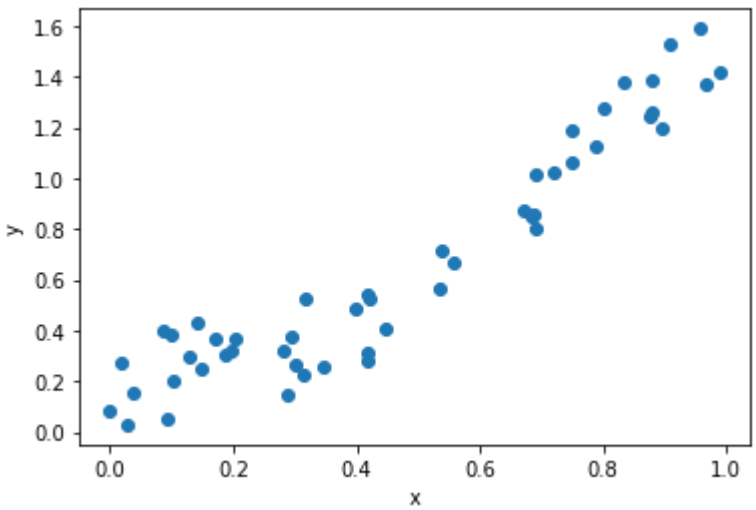
```
# Let us see this in action with one variable => easy to visualize

import numpy as np
import matplotlib.pyplot as plt

np.random.seed(1)
X = np.random.rand(50,1)
y = 0.7*(X**5)·-·\
····2.1*(X**4)·+·\
····2.3*(X**3)·+·\
····0.2*(X**2)·+·\
····0.3*·X·+·\
····0.4*np.random.rand(50,1) # no data in world is perfect
fig = plt.figure()
plt.scatter(X, y)
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```




```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X, y)
output = model.predict(X)

fig = plt.figure()
plt.scatter(X, y, label="samples")
plt.scatter(X, output, label="prediction")
plt.xlabel("X")
plt.ylabel("Y")
plt.show()
display(model.score(X, y))
```

```
from sklearn.preprocessing import StandardScaler

X_deg2 = np.hstack([X, X**2])
model_deg2 = LinearRegression()
model_deg2.fit(X_deg2, y)
output = model_deg2.predict(X_deg2)

fig = plt.figure()
plt.scatter(X, y, label="samples")
plt.scatter(X, output, label="prediction")
plt.xlabel("X")
plt.ylabel("Y")
plt.show()
display(model_deg2.score(X_deg2, y))
```

```
X_deg3 = np.hstack([X, X**2, X**3])
model_deg3 = LinearRegression()
model_deg3.fit(X_deg3, y)
output = model_deg3.predict(X_deg3)

fig = plt.figure()
plt.scatter(X, y, label="samples")
plt.scatter(X, output, label="prediction")
plt.xlabel("x")
plt.ylabel("y")
plt.show()
display(model_deg3.score(X_deg3, y))
```

```
from sklearn.preprocessing import PolynomialFeatures

poly = PolynomialFeatures(3)
X_poly = poly.fit_transform(X)
display(X_poly.shape) # shape of the generated features

n_features = X_poly.shape[1]

for degree in range(n_features):
    fig = plt.figure()
    plt.scatter(X, X_poly[:,degree])
    plt.xlabel("X")
    plt.ylabel(f"X^{degree}")
    plt.show()
```

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.preprocessing import StandardScaler

scores = []
for i in range(1, 10):
    poly = PolynomialFeatures(i)
    X_poly = poly.fit_transform(X)
    scaler = StandardScaler()
    scaler.fit(X_poly)
    X_poly_scaled = scaler.transform(X_poly)
    model = LinearRegression()
    model.fit(X_poly_scaled, y)
    output = model.predict(X_poly_scaled)

    fig = plt.figure()
    plt.scatter(X, y, label="samples")
    plt.scatter(X, output, label="prediction")
    plt.xlabel("X")
    plt.ylabel("Y")
    plt.title(f"Degree {i}")
    plt.show()
```

```
        display(model.score(X_poly_scaled, y))
        scores.append(model.score(X_poly_scaled, y))

    print(scores)
```

```
#overfit like crazy
from sklearn.preprocessing import PolynomialFeatures
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression

degree = 50 # max-degree
scores = []
for i in range(0, degree):
    poly = PolynomialFeatures(i)
    X_poly = poly.fit_transform(X)
    scaler = StandardScaler()
    scaler.fit(X_poly)
    X_poly_scaled = scaler.transform(X_poly)
    model = LinearRegression()
    model.fit(X_poly_scaled, y)
    output = model.predict(X_poly_scaled)
    scores.append(model.score(X_poly_scaled, y))
```

```
max_idx = np.argmax(scores)
print(max_idx, scores[max_idx])

fig = plt.figure()
plt.scatter(range(0,degree), scores, label="samples")
plt.grid()
plt.show()
```

```
degree = 31
poly = PolynomialFeatures(degree)
X_poly = poly.fit_transform(X)
scaler = StandardScaler()
scaler.fit(X_poly)
X_poly_scaled = scaler.transform(X_poly)
model = LinearRegression()
model.fit(X_poly_scaled, y)
output = model.predict(X_poly_scaled)
fig = plt.figure()
plt.scatter(X, y, label="ground truth")
plt.scatter(X, output, label="prediction")
plt.xlabel("X")
plt.ylabel("Y")
plt.title(f"Degree {degree}")
plt.show()
display(model.score(X_poly_scaled, y))
scores.append(model.score(X_poly_scaled, y))
```

```
# lets first generate a dataset of 1000 points this time
# train, Val and test set
np.random.seed(1)
X = np.random.rand(100,1)
y = 0.7*(X**5) - \
    2.1*(X**4) + \
    2.3*(X**3) + \
    0.2*(X**2) + \
    0.3* X + \
    0.4*np.random.rand(100,1)

from sklearn.model_selection import train_test_split
#0.6, 0.2, 0.2 split
X_tr_cv, X_test, y_tr_cv, y_test = train_test_split(X, y, test_size=0.2, random_sta

X_train, X_val, y_train, y_val = train_test_split(X_tr_cv, y_tr_cv, test_size=0.25,


fig = plt.figure()
plt.scatter(X_train, y_train)
plt.xlabel("x")
plt.ylabel("y")
plt.title("Training Data")
plt.show()

fig = plt.figure()
plt.scatter(X_val, y_val)
plt.xlabel("x")
plt.ylabel("y")
plt.title("Validation Data")
plt.show()

fig = plt.figure()
plt.scatter(X_test, y_test)
plt.xlabel("x")
plt.ylabel("y")
plt.title("Test Data")
plt.show()
```

```
# Train and Validatation without hyper param tuning. Just by controlling the degree
from sklearn.preprocessing import PolynomialFeatures
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import make_pipeline

max_degree = 32 # max polynomial degree
train_scores = []
val_scores = []
scaler = StandardScaler()
for degree in range(1, max_degree):
    polyreg_scaled = make_pipeline(PolynomialFeatures(degree), scaler, LinearRegres
    polyreg_scaled.fit(X_train, y_train)
    train_score = polyreg_scaled.score(X_train, y_train)
    val_score = polyreg_scaled.score(X_val, y_val)
    train_scores.append(train_score)
    val_scores.append(val_score)
```

```
plt.figure()
plt.plot(list(range(1, 32)), train_scores, label="train")
plt.plot(list(range(1, 32)), val_scores, label="val")
plt.legend(loc='lower right')
plt.xlabel("degree")
plt.ylabel("R-score")
plt.grid()
plt.show()
```

```
# Ridge regression with L2-regularization
```

```
from sklearn.linear_model import Ridge
train_scores = []
val_scores = []
scaler = StandardScaler()

for alpha in range(1,20):
  polyreg_scaled = make_pipeline(PolynomialFeatures(32), scaler, Ridge(alpha))
  polyreg_scaled.fit(X_train, y_train)
  train_score = polyreg_scaled.score(X_train, y_train)
  val_score = polyreg_scaled.score(X_val, y_val)
  train_scores.append(train_score)
  val_scores.append(val_score)
```

```
print(val_scores)
```

```
[0.9191998246630931, 0.921663634745074, 0.9225261379655693, 0.9228107199506557
```

```
# train the best model with alpha=4

polyreg_scaled = make_pipeline(PolynomialFeatures(32), scaler, Ridge(4))
polyreg_scaled.fit(X_train, y_train)
train_score = polyreg_scaled.score(X_train, y_train)
```