

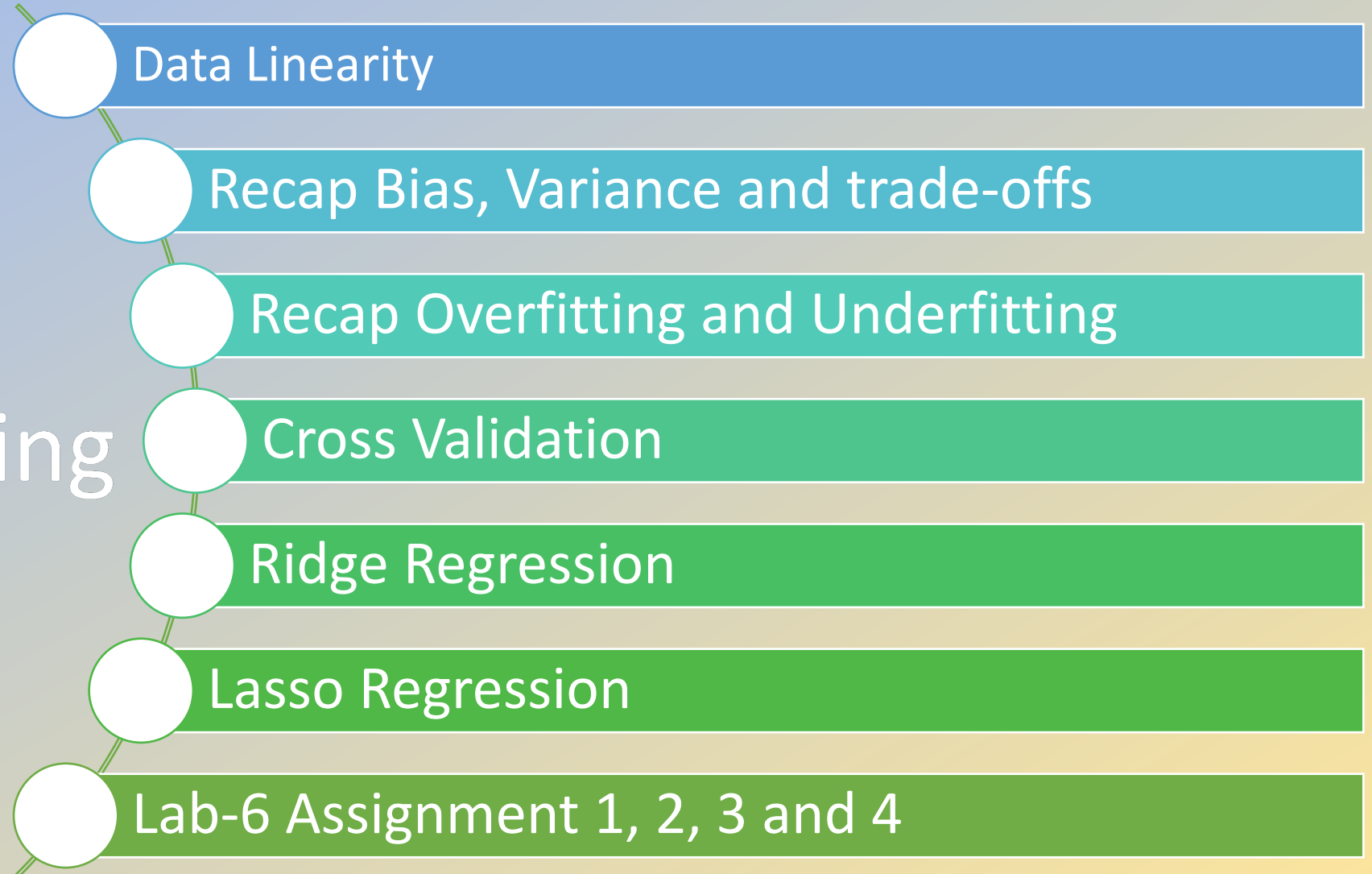
Regularization in Machine Learning

Machine Learning in Python

Discussion

Regularization in Machine Learning

Machine Learning in Python



Data Linearity

Regression

Data Linearity

Classification

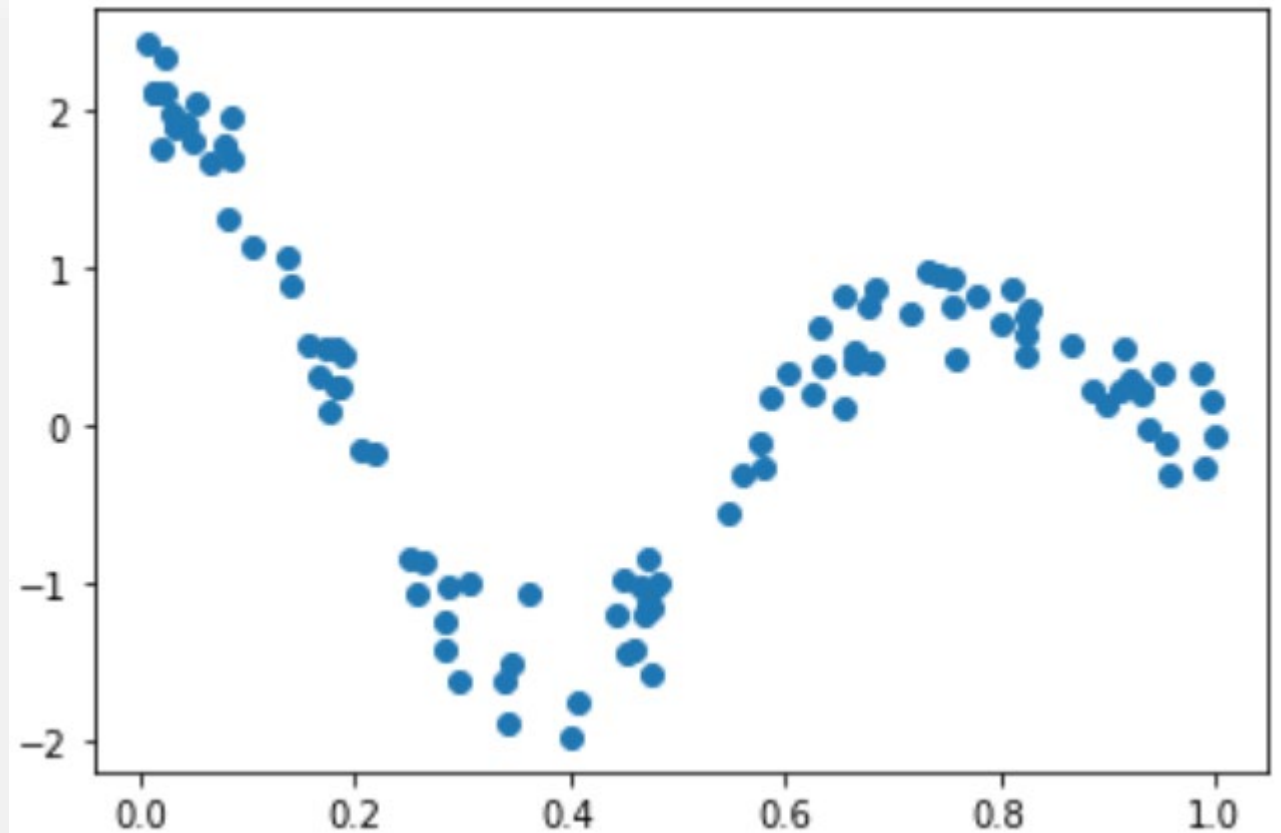
Data Linearity

Hands On

```
import pandas as pd

df = pd.read_csv('./dataset 1.csv')

import matplotlib.pyplot as plt
plt.scatter(df.X1, df.X2)
```



Bias and Variance

Recap

Overfitting and Underfitting

Recap



Overfitting and Underfitting

Hands On

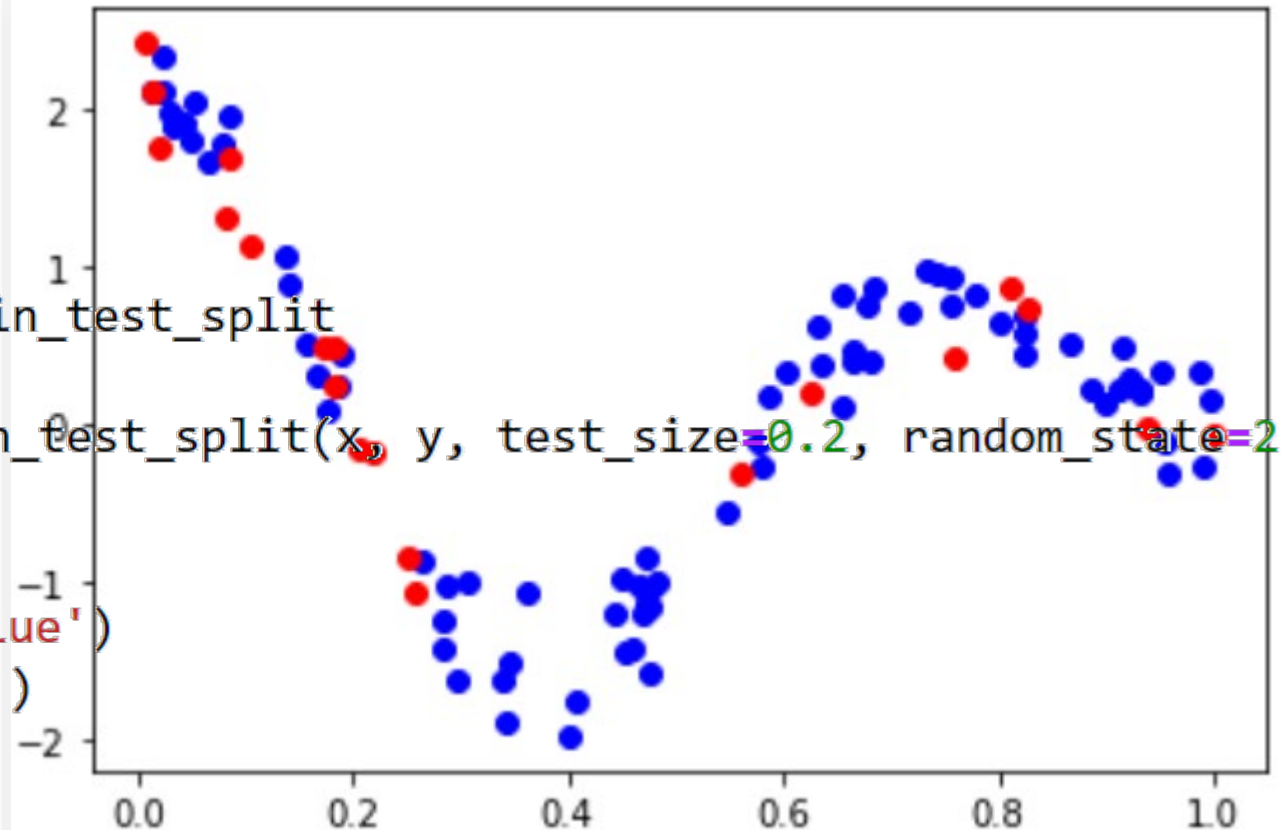
```
import numpy as np

x = df[['X1']].to_numpy()
y = df[['X2']].to_numpy()

from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=2)

plt.scatter(x_train, y_train, color='blue')
plt.scatter(x_test, y_test, color='red')
```



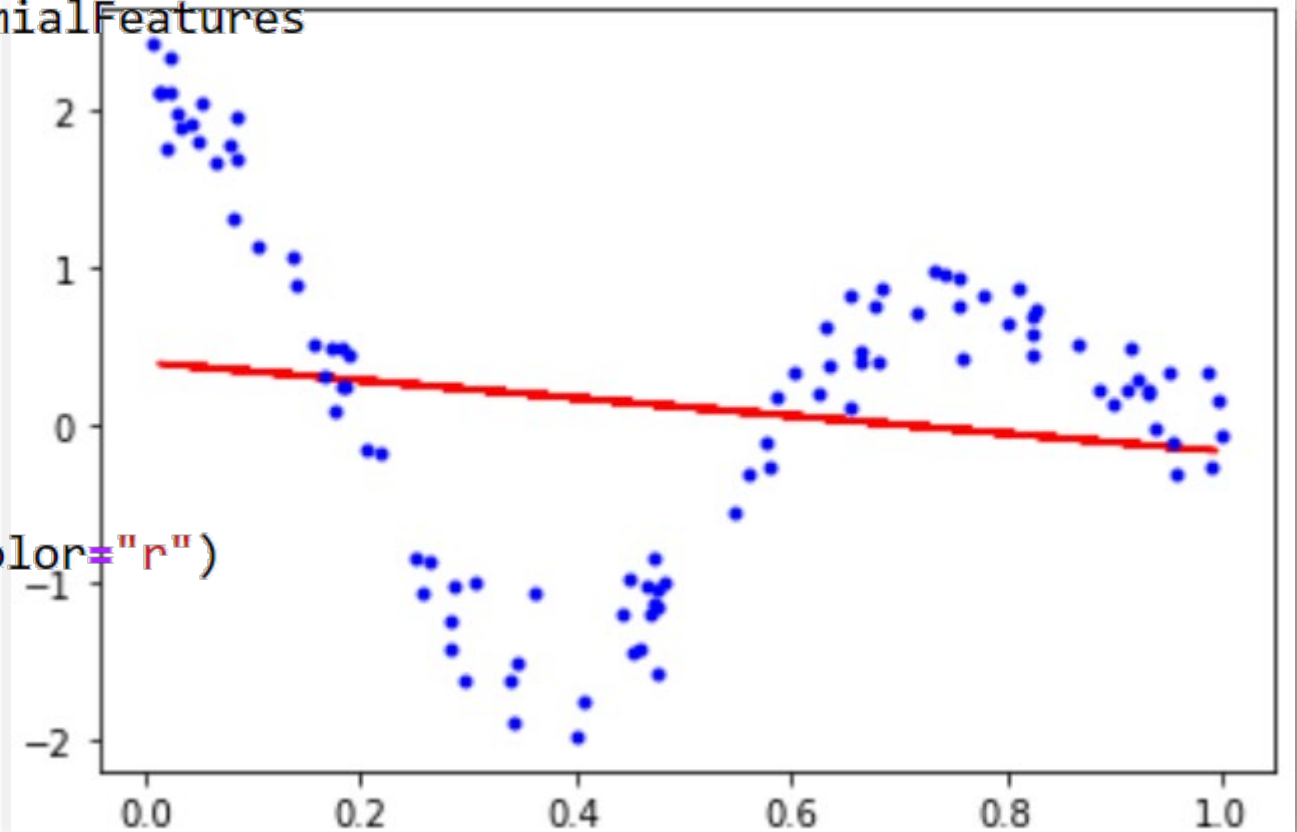
Overfitting and Underfitting

Hands On

```
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import r2_score
```

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

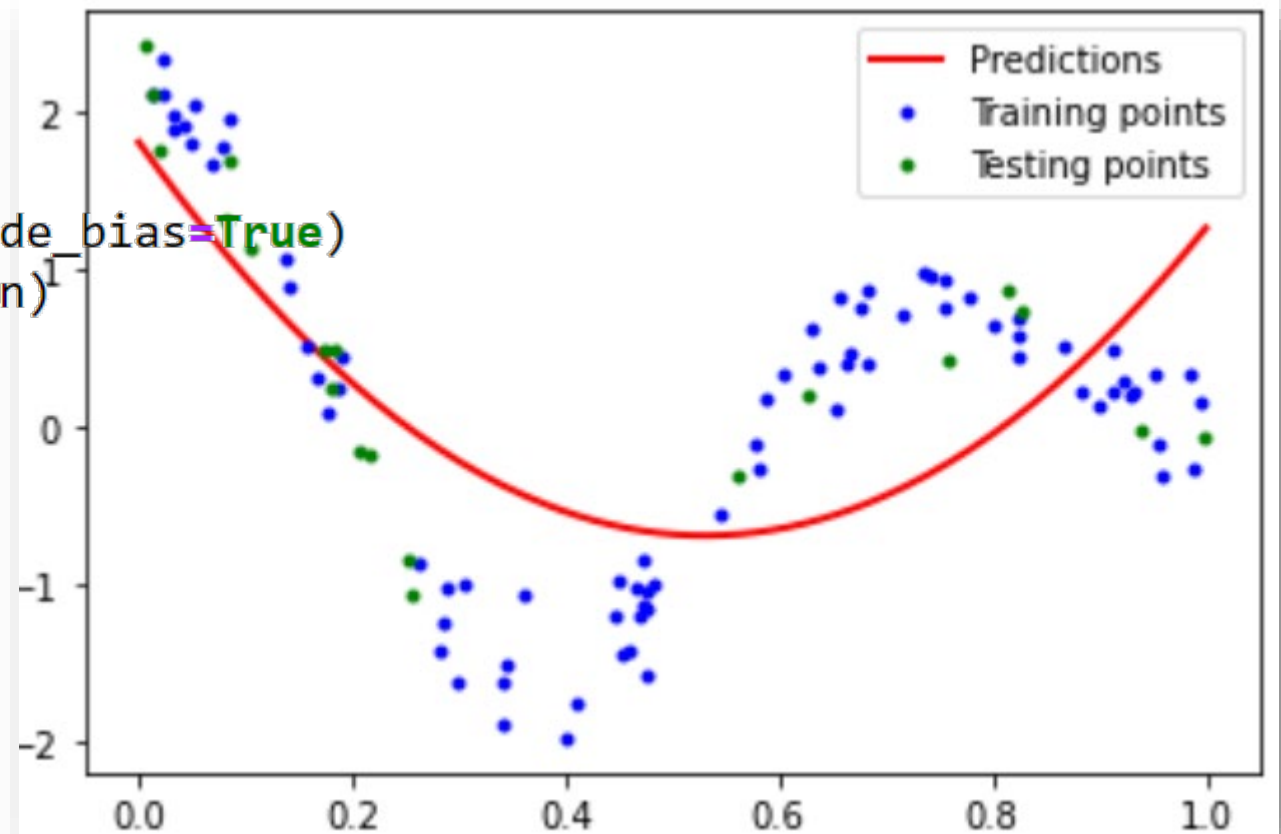
```
plt.plot(x_train, lr.predict(x_train), color="r")
plt.plot(x, y, "b.")
plt.xlabel("X")
plt.ylabel("Y")
plt.show()
```



Overfitting and Underfitting

Hands On

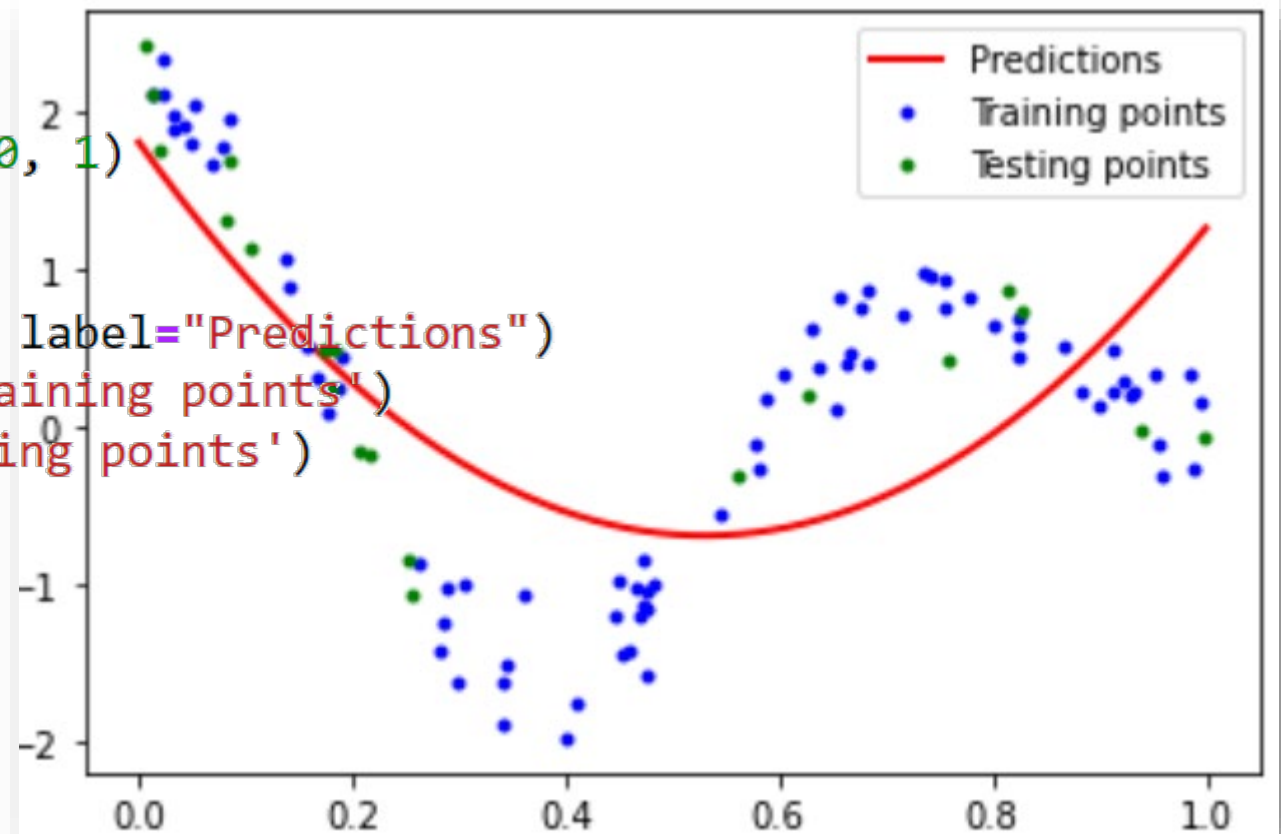
```
#applying polynomial regression degree 2  
poly = PolynomialFeatures(degree=2, include_bias=True)  
x_train_trans = poly.fit_transform(x_train)  
x_test_trans = poly.transform(x_test)  
#include bias parameter  
lr = LinearRegression()  
lr.fit(x_train_trans, y_train)  
y_pred = lr.predict(x_test_trans)  
print(r2_score(y_test, y_pred))
```



Overfitting and Underfitting

Hands On

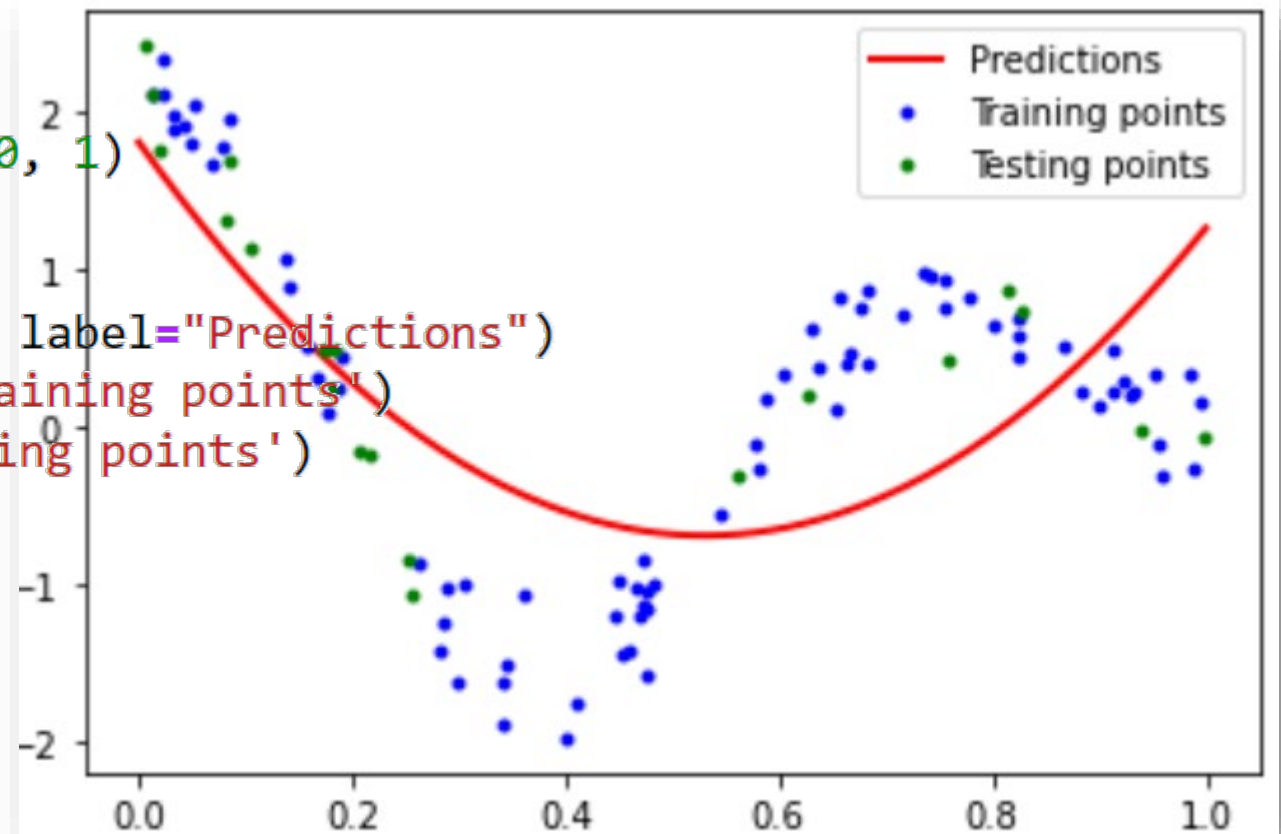
```
X_new = np.linspace(0, 1, 200).reshape(200, 1)
X_new_poly = poly.transform(X_new)
y_new = lr.predict(X_new_poly)
plt.plot(X_new, y_new, "r-", linewidth=2, label="Predictions")
plt.plot(x_train, y_train, "b.", label='Training points')
plt.plot(x_test, y_test, "g.", label='Testing points')
plt.xlabel("X")
plt.ylabel("y")
plt.legend()
plt.show()
```



Overfitting and Underfitting

Hands On

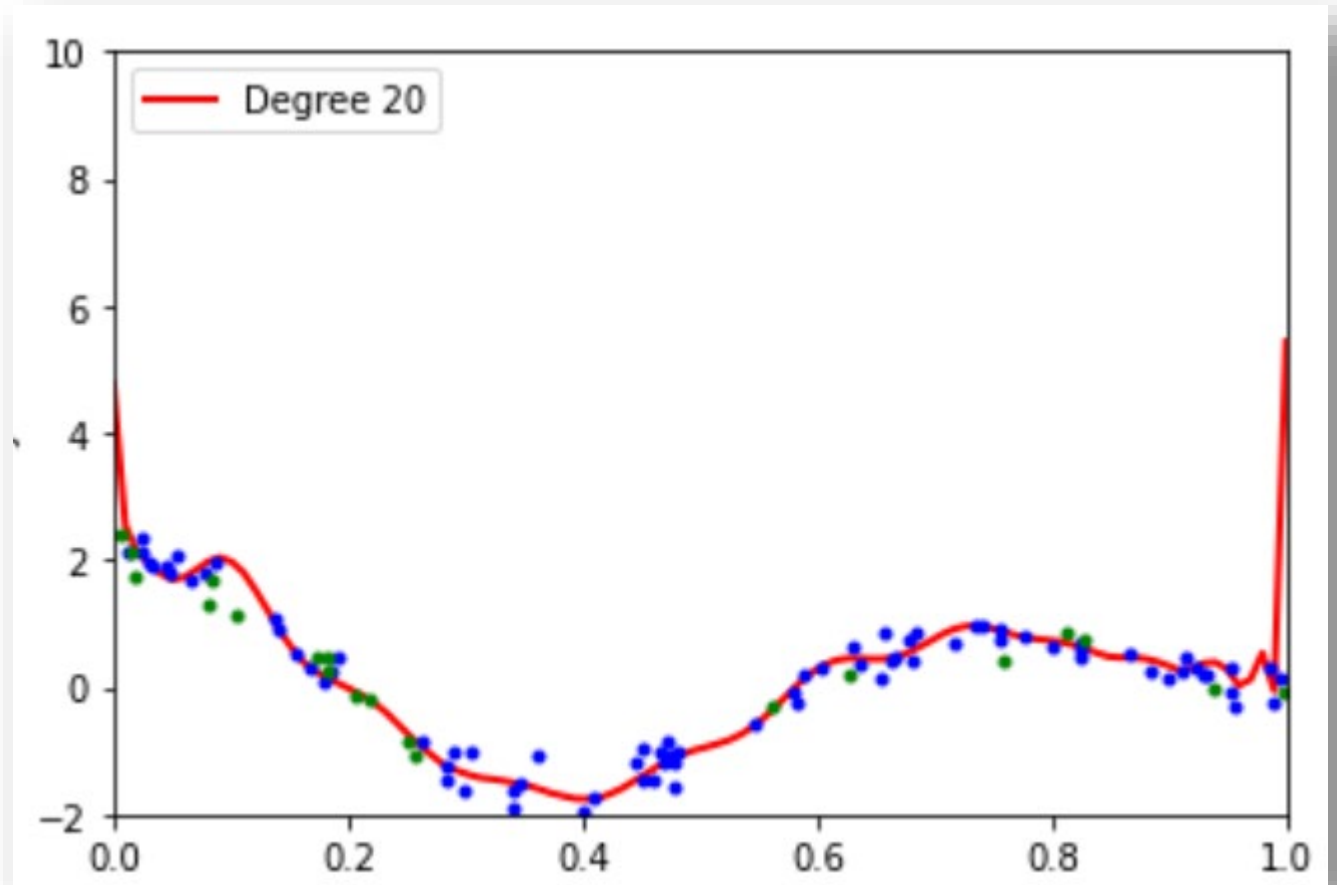
```
X_new = np.linspace(0, 1, 200).reshape(200, 1)
X_new_poly = poly.transform(X_new)
y_new = lr.predict(X_new_poly)
plt.plot(X_new, y_new, "r-", linewidth=2, label="Predictions")
plt.plot(x_train, y_train, "b.", label='Training points')
plt.plot(x_test, y_test, "g.", label='Testing points')
plt.xlabel("X")
plt.ylabel("y")
plt.legend()
plt.show()
```



Overfitting and Underfitting

Hands On

```
polynomial_regression(20)
```



Overfitting and Underfitting

Hands On

```
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline

def polynomial_regression(degree):
    X_new=np.linspace(0, 1, 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()
    polynomial_regression = Pipeline([
        ("poly_features", polybig_features),
        ("std_scaler", std_scaler),
        ("lin_reg", lin_reg),
    ])
    polynomial_regression.fit(x_train_trans, y_train)

    y_newbig = polynomial_regression.predict(X_new_poly)
```


Overfitting and Underfitting

Hands On

#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([0, 1, -2, 10])
plt.show()
```

Cross Validation

The Problem with Train-Test Split

Cross Validation

Hands On

```
from sklearn.model_selection import KFold
kf = KFold(n_splits=2, random_state=None)

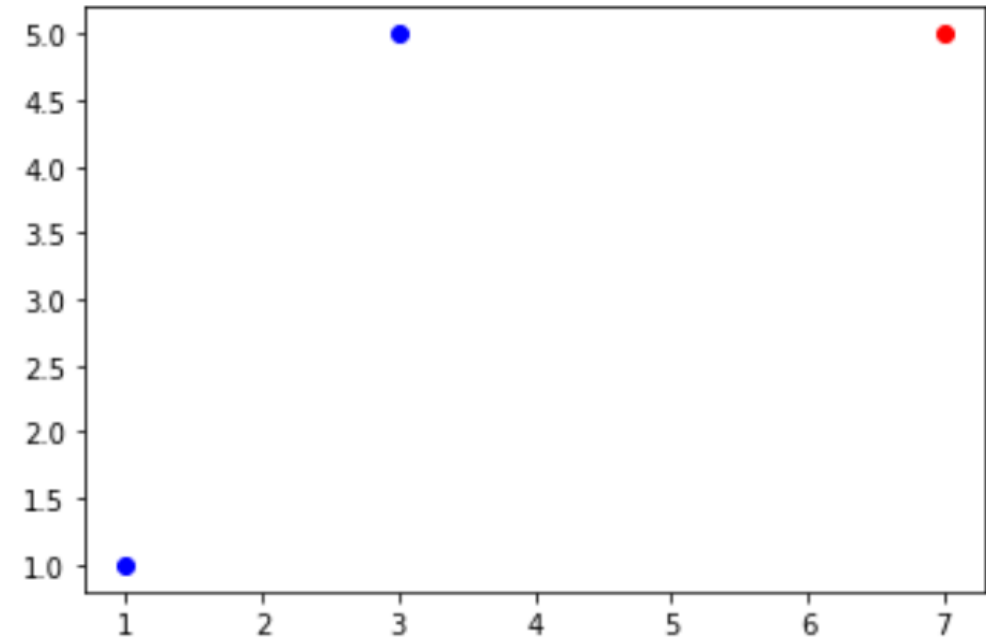
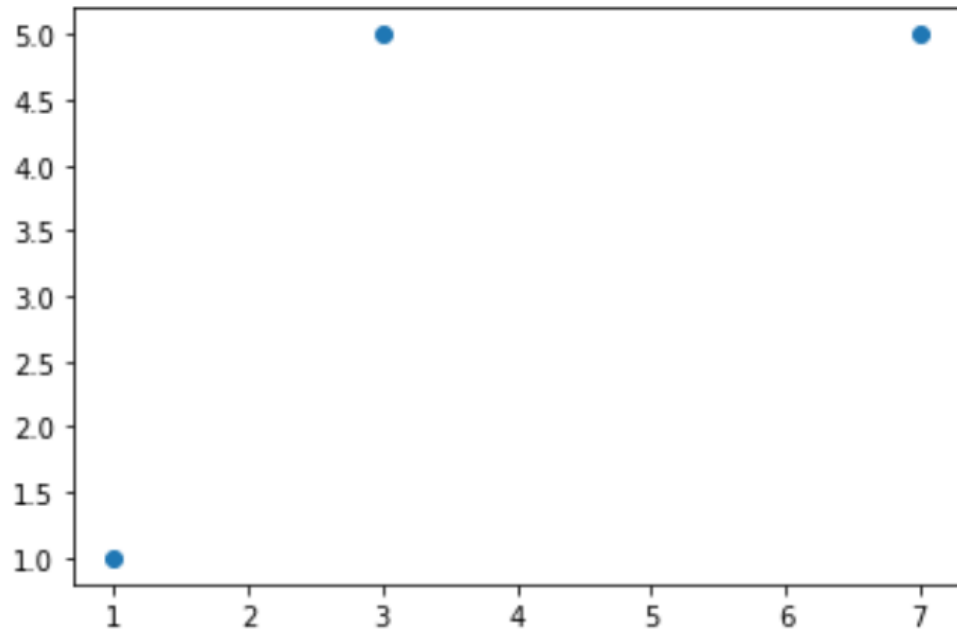
for train_index, test_index in kf.split(x_train_trans):
    print("Train:", train_index, "Validation:", test_index)
    x_train, x_test = x[train_index], x[test_index]
    y_train, y_test = y[train_index], y[test_index]
```

Ridge Regression

Solution to reduce variance

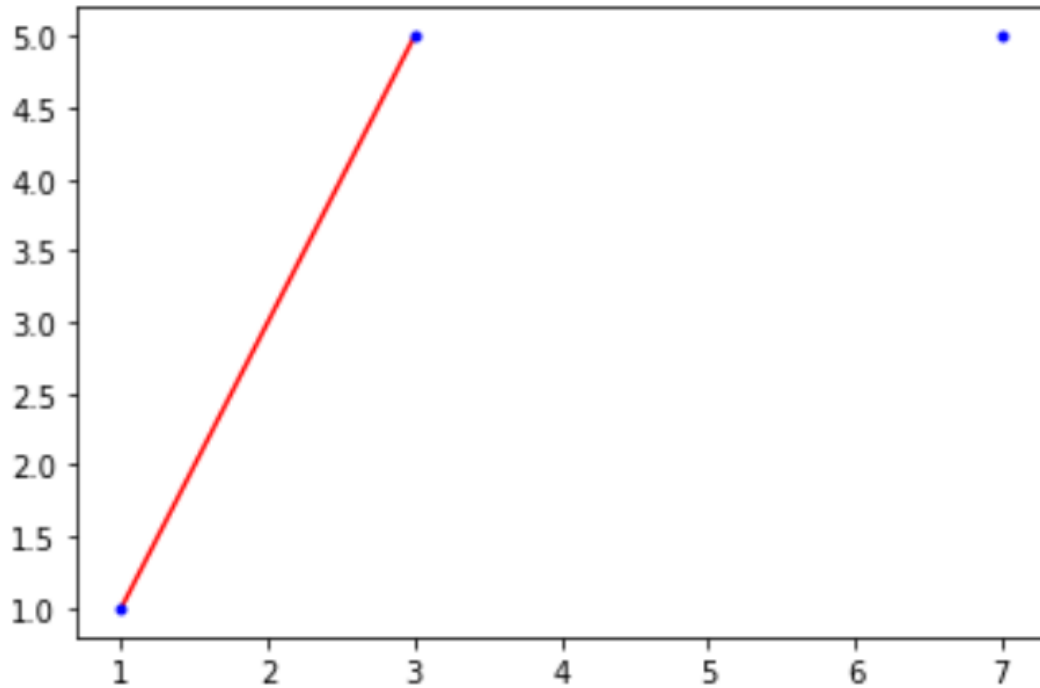
Ridge Regression

Hands On



Ridge Regression

Hands On

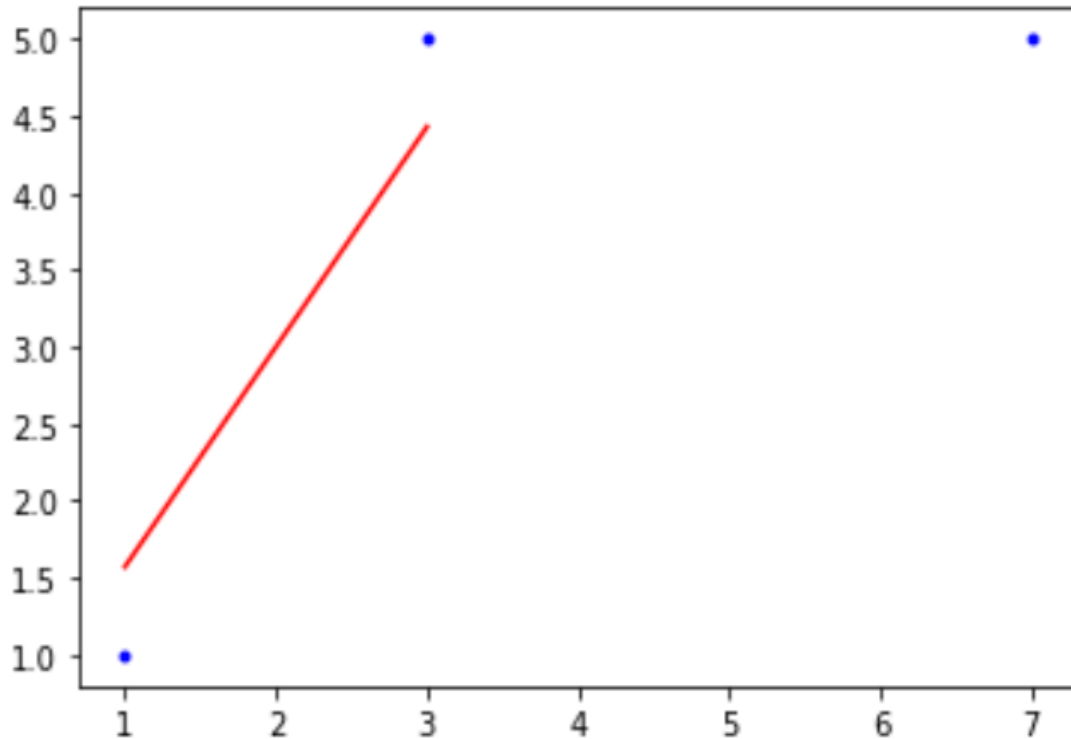


```
print(mean_squared_error(y_test, y_pred))
```

63.999999999999994

Ridge Regression

Hands On



```
from sklearn import linear_model  
  
ridge = linear_model.Ridge(alpha=.8)  
ridge.fit(x_train, y_train)  
y_pred_ridge = ridge.predict(x_test)  
print(mean_squared_error(y_test, y_pred_ridge))
```

26.44897959183673

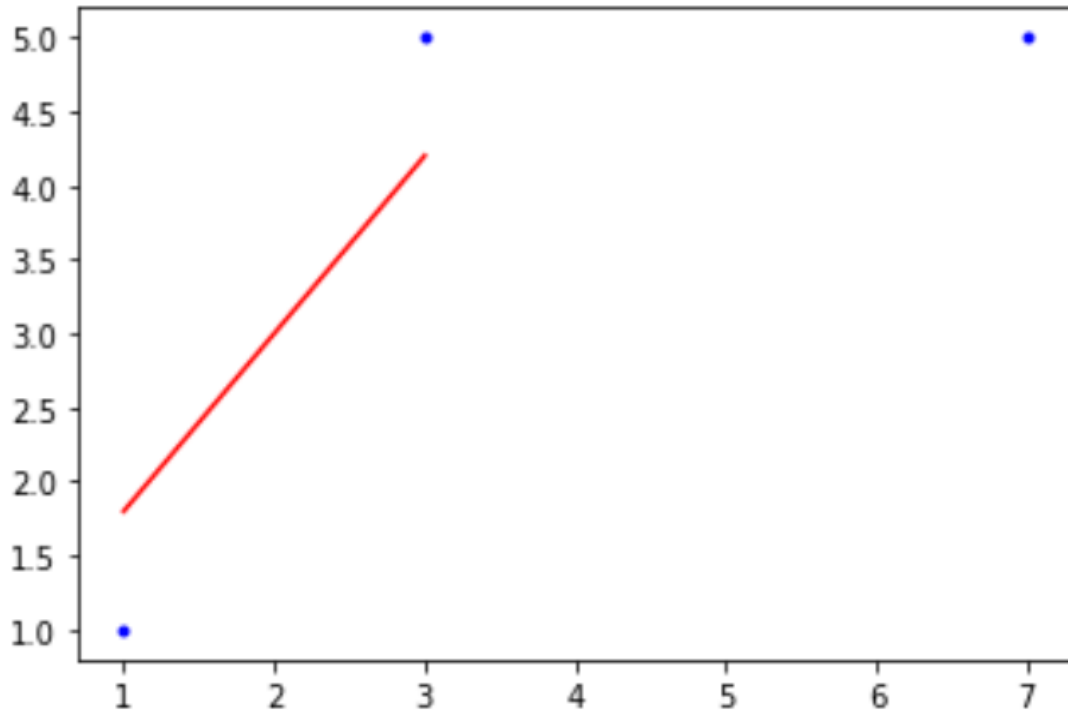
```
plt.plot(x_train, ridge.predict(x_train), color="r")  
plt.plot(x, y, "b.")  
plt.xlabel("X")  
plt.ylabel("Y")  
plt.show()
```

LASSO Regression

Another solution to reduce variance

LASSO Regression

Hands On



```
from sklearn import linear_model  
  
lasso = linear_model.Lasso(alpha=.8)  
lasso.fit(x_train, y_train)  
y_pred_lasso = lasso.predict(x_test)  
print(mean_squared_error(y_test, y_pred_lasso))
```

16.0

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

Lab Assignment

Assignment 1

Use **K-Fold cross validation** technique to find the **optimal degree** of the **polynomial regression function**.

```
polynomial_regression(degree)
```


Lab Assignment

Assignment 2

Use **K-Fold cross validation** technique to find the best classification algorithm among **KNN**, **Naïve Bayes**, **SVM** and **Decision Tree** for any categorical dataset.

Lab Assignment

Assignment 3

Use **K-Fold cross validation** technique to find the **optimal alpha** for **Ridge** and **Lasso** regression for **dataset 0**.

Lab Assignment

Assignment 4

Give a **detailed analysis** on what happens with **Ridge** and **Lasso** regression for **dataset 0**.

Note: *Use alpha from Assignment 3.*