NUID: 002320268 Name: Zankhana Mehta Email: mehta.zan@northeastern.edu

Problem 1 [12pts] For each of parts (a) through (c), indicate whether we would generally expect the performance of a flexible statistical learning method to be better or worse than an inflexible method. Justify your answer.

(a) The sample size n is extremely large, and the number of predictors p is small. Answer: Models flexibilty is its ability to adapt, evolve and learn from input data. In the above case where sample size is large, a flexible statistical method would perform better than an inflexible one because it can capture the true underlying relationship between the data and predictors. An inflexible model might be too restrictive to accurately fit the data.

(b) The number of predictors p is extremely large, and the number of observations n is small. Answer: In this case the inflexible statistical method will perform better since it can restrict the complexity and prevent overfitting that the flexible methods are prone to.

(c) The relationship between the predictors and response is highly non-linear. Answer: Flexible statistical learning methods can capture non-linear relationships in data without making assumptions about the data's underlying distribution. While inflexible model might lead to prediction errors. So in this case flexible statistical methods will perform better.

Problem 2

a) [4pts] Use Figure 1a to predict medv given lstat=25 with K = 1 and K = 5. **lstat=25 k=1 nearest neighbour is (24.39,8.3) Answer: medv=8.3 k=5 Answer: medv=12.1**

(b) [4pts] Repeat (a) for lstat=27. **lstat=27 k=1 nearest neighbour is (26.82,13.4) Answer: medv=13.4 k=5 neartest neighbours are (24.39,8.3),(25.68,9.7),(29.68,8.1),(26.82,13.4),(26.4,17.2) Answer: medv=11.34**

(c) [4pts] Use Figure 1b to predict medv_cat given lstat=25 with K = 1 and K = 5. **lstat=25 k=1 nearest neighbour is (24.39,0) Answer:medv=0 k=5 nearest neighbour is (24.39,0),(25.68,0),(23.6,1.4),(24.16,2),(26.4,2) frequency of 0:2, 1:1,2:2 taking the least median with highest frequency Answer:medv=0**

(d) [4pts] Repeat (c) for lstat=27. **lstat=27 k=1 nearest neighbour is (26.82,1) Answer: medv=1 k=5 neartest neighbours are (24.39,0),(25.68,0),(29.68,0),(26.82,1),(26.4,2) frequency of 0:3, 1:1,2:1 Answer: medv=0**

(e) [4pts] If we increase K in KNN, is the model more flexible or less flexible? Explain why. **Answer: Let's run thorugh the case where initially k is small,the decision boundary is highly flexible, that is the model reacts to even small variations in the data. But it can closely follow the training data, which can lead to overfitting. As we increase k the model takes into account more neighbors when**

classifying a new data point. This leads to a smoother decision boundary, as the prediction for any point now depends on a larger set of neighbors. But the model becomes less sensitive to noise or individual outliers.Increasing k reduces variance (making the model more stable and less prone to overfitting), but it increases bias (the model may become too simple and unable to capture the complexity of the data). So, higher k results in a less flexible model.

(f) [6pts] How do the square of bias, variance, training MSE, test MSE, and irreducible error change with K for KNN regression? Explain why. **Square of bias: When k is small the model closely follows the training data, leading to low bias. However, as k increases, the predictions are based on more neighbors, making the model's decision boundary smoother and less fitted to specific patterns in the data. This leads to an increased bias as the model becomes less flexible and may not capture the underlying patterns in the data as well.**

Variance: It is inversley proportional to the value of k. A small k makes the model very flexible, highly sensitive to variations in the training data, and prone to overfitting (high variance). As k increases, the predictions are based on more points, leading to more stable and smoother estimates. The model becomes less sensitive to changes in individual training points, reducing variance.

Training MSE: For small k, the model is overfitting and tends to closely match the training data, leading to a low training MSE. As k increases, the model fits the training data less accurately, so the training MSE increases.

Test MSE: With very small k, the model is prone to overfitting, resulting in high test MSE because it is capturing noise in the data (high variance). As k increases, the model generalizes better, reducing overfitting, and test MSE decreases. However, after a certain point, increasing k too much leads to underfitting (the model becomes too simple), and the test MSE starts increasing again due to high bias. This creates a Ushaped curve for test MSE. Irreducible Error: This remains constant. It represents the noise inherent in the data that cannot be modeled or reduced by any model, including KNN. It's due to random variability or factors not captured by the available features, so it does not change with k.

Problem 3

Part a

Number of observations(p): The sample size is specified in the function as 100, therefore n=100.

Number of features(p): As mentioned in the question different powers of x are different features, since in this equation there are 3 x variables with different powers number of features is 3(p=3).

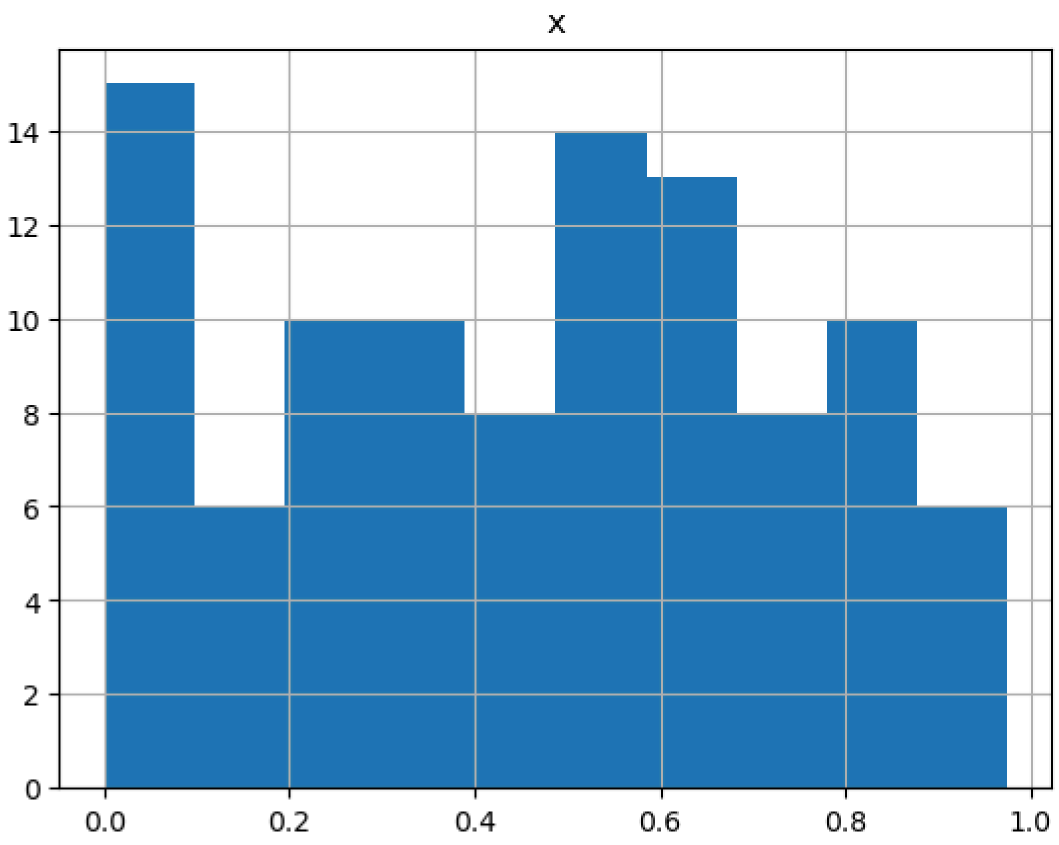
Model used in this question to generate the data is y=x^5 - 2 * x^4 + x^3

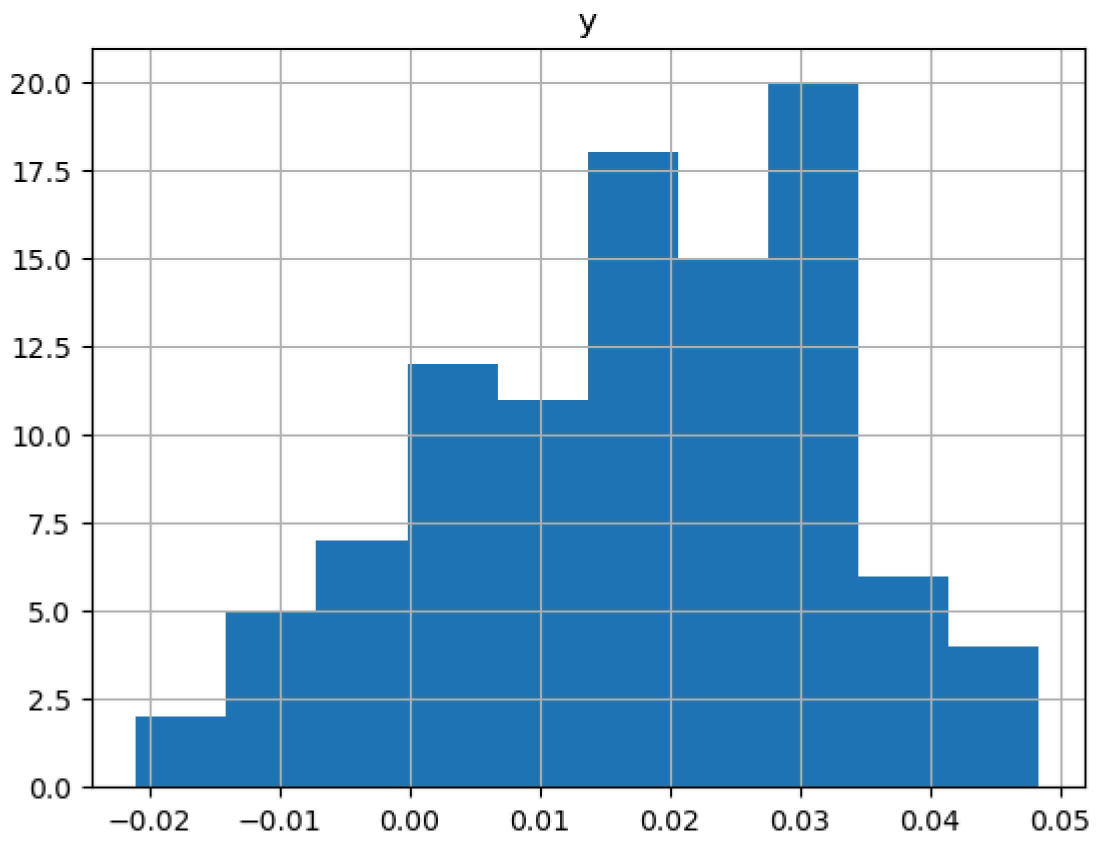
In [10]:
```
import numpy as np
import pandas as pd
```

In [12]:
```
def f(x):
 return x ** 5 - 2 * x ** 4 + x ** 3 #true function
def get_sim_data(f, sample_size, std):
 x = np.random.uniform(0, 1, sample_size)
 y = f(x) + np.random.normal(0, std, sample_size)
 df = pd.DataFrame({'x': x, 'y': y})
 return df
df = get_sim_data(f,100,0.01)
#print(get_sim_data(f))
```

In [14]:
```
df.hist('x') #uniformly distributed
df.hist('y') #normally distributed
```

Out[14]:
```
array([[<Axes: title={'center': 'y'}>]], dtype=object)
```





In [16]:
```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
```

In [18]:
```
df.head(5)
```

Out[18]:

	x	y
0	0.683214	0.033149
1	0.004226	-0.014879
2	0.342745	0.014779
3	0.251840	0.014984
4	0.669353	0.022133

In [20]:
```
df.shape
```

Out[20]:
```
(100, 2)
```

Part b and c

In [24]:
```
X = df['x'].values.reshape(-1, 1)
Y = df['y'].values
for i in range(16):
 poly=PolynomialFeatures(degree=i)
 degree_x=poly.fit_transform(X)#training set
 model=LinearRegression()
 model.fit(degree_x,Y)
 x0=0.18
 x0_fit=poly.transform([[x0]])#test set
 print(model.predict(x0_fit))
```

```
[0.01767839]
[0.01243638]
[0.0123963]
[0.00684651]
[0.00699582]
[0.00799499]
[0.00800465]
[0.00603613]
[0.00620828]
[0.00619444]
[0.00586755]
[0.00530537]
[0.00411163]
[0.00416807]
[0.00413081]
[0.00211689]
```

Part d

In [30]:
```
# iterating for 250 times
predictions_per_degree = {j: [] for j in range(16)} #for part e
for i in range(250):
     def f(x):
         return x ** 5 - 2 * x ** 4 + x ** 3
     def get_sim_data(f, sample_size, std):
         x = np.random.uniform(0, 1, sample_size)
         y = f(x) + np.random.normal(0, std, sample_size)
         df = pd.DataFrame({'x': x, 'y': y})
         return df
     df = get_sim_data(f,100,0.01)
     X = df['x'].values.reshape(-1, 1)
     Y = df['y'].values
     for j in range(16):
         poly=PolynomialFeatures(degree=j)
         degree_x=poly.fit_transform(X)#training set
         model=LinearRegression()
         model.fit(degree_x,Y)
         y_pred=model.predict(poly.transform([[0.18]]))
         predictions_per_degree[j].append(y_pred)
```

Part e

In [33]:
```
bias_squared = {}
y_true = f(0.18)
for j in range(16):
 average = np.mean(predictions_per_degree[j])
 bias = average- y_true
 bias_squared[j] = bias ** 2
 print(f"Degree {j}: Bias Squared = {bias_squared[j]}")
```

```
Degree 0: Bias Squared = 0.00016340939284580186
Degree 1: Bias Squared = 7.114051374559911e-05
Degree 2: Bias Squared = 2.8683778681132882e-05
Degree 3: Bias Squared = 4.3240568498671105e-06
Degree 4: Bias Squared = 9.291226839446423e-07
Degree 5: Bias Squared = 1.4323114578246049e-08
Degree 6: Bias Squared = 1.8565648299011434e-08
Degree 7: Bias Squared = 4.425165338366436e-09
Degree 8: Bias Squared = 2.0299597089602324e-09
Degree 9: Bias Squared = 2.351206141546605e-09
Degree 10: Bias Squared = 1.1081813894659267e-09
Degree 11: Bias Squared = 2.3594601441624568e-08
Degree 12: Bias Squared = 3.8679247529445e-08
Degree 13: Bias Squared = 7.30168256541323e-08
Degree 14: Bias Squared = 7.458350625064033e-08
Degree 15: Bias Squared = 7.924088230195677e-08
```

Part f

In [36]:
```
variance = {}
for j in range(16):
     avg = np.mean(predictions_per_degree[j])
     variance[j] = np.mean((predictions_per_degree[j] - avg) ** 2)
     print(f"Degree {j}: Variance = {variance[j]}")
```

```
Degree 0: Variance = 2.5354433379671205e-06
Degree 1: Variance = 4.240572692102544e-06
Degree 2: Variance = 3.5497188527788246e-06
Degree 3: Variance = 3.1081128190596366e-06
Degree 4: Variance = 4.579659407803415e-06
Degree 5: Variance = 5.7102862135557875e-06
Degree 6: Variance = 5.836802853843687e-06
Degree 7: Variance = 7.896756351334372e-06
Degree 8: Variance = 9.052928298209146e-06
Degree 9: Variance = 9.411404498448661e-06
Degree 10: Variance = 9.529829833015183e-06
Degree 11: Variance = 1.1318754412689825e-05
Degree 12: Variance = 1.337533786987028e-05
Degree 13: Variance = 1.401296199282676e-05
Degree 14: Variance = 1.5659124894239428e-05
Degree 15: Variance = 1.796367822770741e-05
```

Part g

In [39]:
```
std=0.01
irr_err=std**2
```

Part h

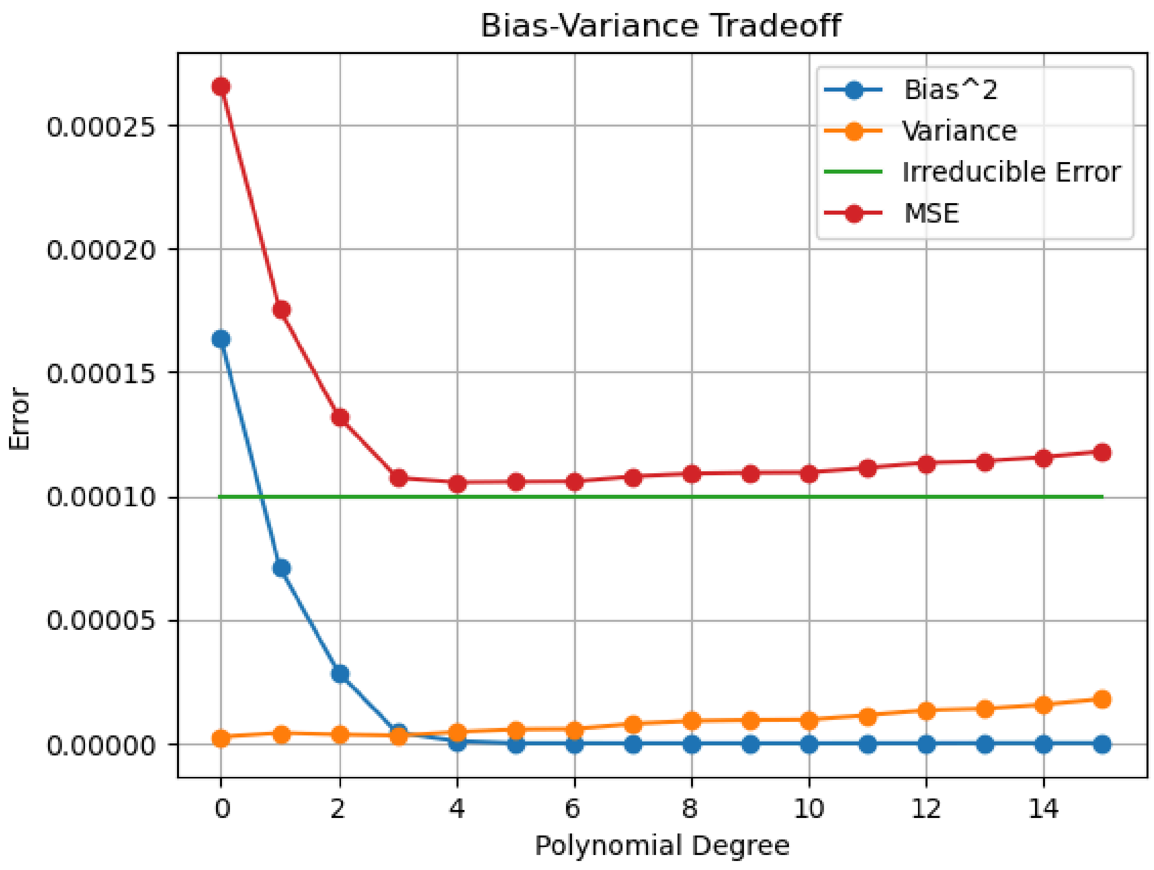
In [42]:
```
mse=[]
for i in range(16):
     mse_degree=bias_squared[i]+variance[i]+irr_err
     mse.append(mse_degree)
print(mse)
```

```
[0.0002659448361837690, 0.00017538108643770168, 0.00013223349753391170, 0.00
010743216966892675, 0.00010550878209174806, 0.00010572460932813404, 0.0001
0585536850214271, 0.00010790118151667274, 0.00010905495825791811, 0.000109
41375570459022, 0.00010953093801440465, 0.00011134234901413146, 0.00011341
401711739972, 0.00011408597881848090, 0.00011573370840049007, 0.00011804291
911000937]
```

Part i

In [46]:
```python
import matplotlib.pyplot as plt
```

In [48]:
```python
degrees=range(16)
plt.plot(degrees,[bias_squared[i] for i in degrees], label='Bias^2', mark
plt.plot(degrees, [variance[i] for i in degrees], label='Variance', marke
plt.plot(degrees, [irr_err for _ in degrees], label='Irreducible Error')#
plt.plot(degrees, [mse[i] for i in degrees], label='MSE', marker='o')
plt.xlabel('Polynomial Degree')
plt.ylabel('Error')
plt.title('Bias-Variance Tradeoff')
plt.legend()
plt.grid(True)
plt.show()
```



Bias: As the polynomial degree increases, the bias squared tends to decrease. A low-degree polynomial (underfitting) cannot capture the complexity of the true function, leading to high bias.

Variance: The variance increases with the degree of the polynomial because higher-degree polynomials are more flexible and tend to fit the noise in the data (overfitting). Irreducible Error: This is constant across all degrees, representing the inherent noise in the data.

MSE (Mean Squared Error): The MSE initially decreases as we reduce bias by increasing the degree of the polynomial. However, as the degree becomes too high, the variance dominates, causing the MSE to increase again. The optimal degree minimizes the MSE.

Bias-Variance Tradeoff: Lower-degree polynomials have higher bias and lower variance, while higher-degree polynomials have lower bias but higher variance. There is a tradeoff between bias and variance. The optimal model lies somewhere between too simple (high bias, low variance) and too complex (low bias, high variance). In this case, it looks like around degree 5 may provide the best tradeoff since it's closest to the true function and minimizes MSE.

Problem 4

Part a

In [54]:
```
import numpy as np
import pandas as pd
```

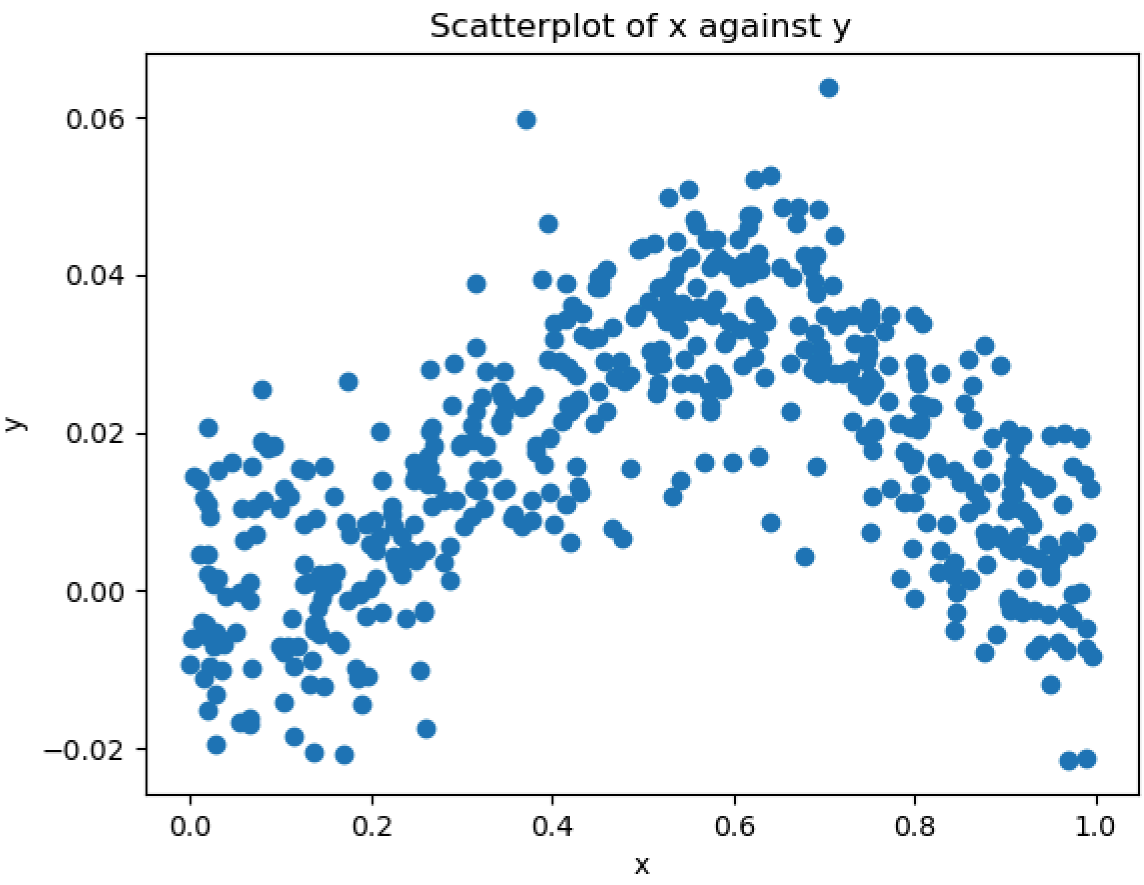
In [56]:
```
def f(x):
 return x ** 5 - 2 * x ** 4 + x ** 3
np.random.seed(1)
x = np.random.uniform(0, 1, size=500)
y = f(x) + np.random.normal(0, 0.01, size=500)
df= pd.DataFrame({'x': x, 'y': y})
df.head()
```

Out[56]:

	x	y
0	0.417022	0.023488
1	0.720324	0.027480
2	0.000114	-0.009339
3	0.302333	0.008121
4	0.146756	-0.011964

Part b

In [59]:
```
import matplotlib.pyplot as plt
plt.scatter(x, y)
plt.title("Scatterplot of x against y")
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```



The scatterplot has a parabolic pattern, which corresponds to the shape of the underlying function f(x). The curve starts low at x=0, increases to a peak near x=0.5, and then declines again toward x=1. This is typical behavior for polynomials of degree 5

Part c

In [62]:
```
from sklearn.model_selection import LeaveOneOut, cross_val_score
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
```

In [69]:
```
np.random.seed(123)
x = np.random.uniform(0, 1, size=500)
y = f(x) + np.random.normal(0, 0.01, size=500)
df= pd.DataFrame({'x': x, 'y': y})
#df.head()
errors1=[]
loo = LeaveOneOut()
degrees = np.arange(1, 8)
for degree in degrees:
     poly = PolynomialFeatures(degree)
     X_poly = poly.fit_transform(x.reshape(-1, 1))

     model = LinearRegression()

     neg_mse = cross_val_score(model, X_poly, y, cv=loo, scoring='neg_mea
     mse = -np.mean(neg_mse) # Mean squared error
     errors1.append(mse)
# Print LOOCV errors for each model
```

```
for i, mse in enumerate(errors1):
     print(f"Degree {i+1} polynomial LOOCV error: {mse}")
```

```
Degree 1 polynomial LOOCV error: 0.00023872667496951678
Degree 2 polynomial LOOCV error: 0.00012610526499073603
Degree 3 polynomial LOOCV error: 0.00011087924643876155
Degree 4 polynomial LOOCV error: 0.00010574321537933433
Degree 5 polynomial LOOCV error: 0.00010460992336824871
Degree 6 polynomial LOOCV error: 0.00010463444234900385
Degree 7 polynomial LOOCV error: 0.0001050017444614648
```

Part d

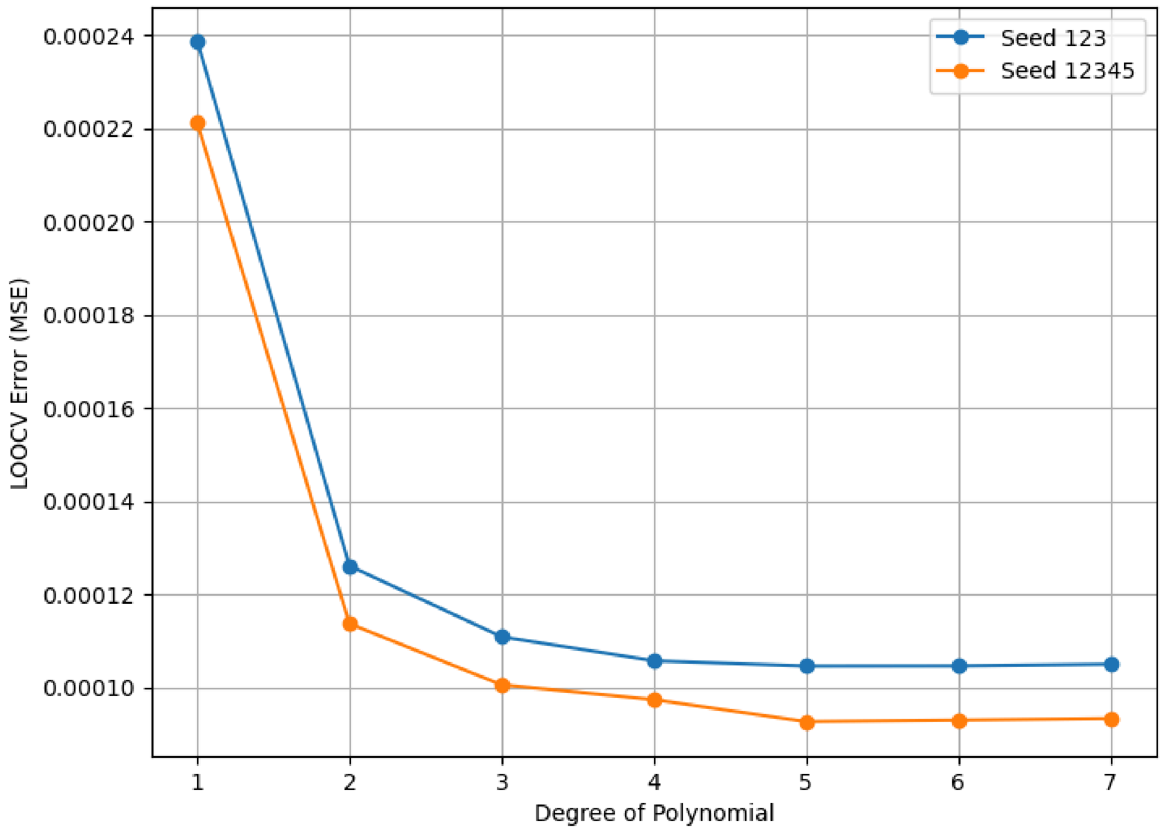
In [83]:
```
np.random.seed(12345)
x = np.random.uniform(0, 1, size=500)
y = f(x) + np.random.normal(0, 0.01, size=500)
df= pd.DataFrame({'x': x, 'y': y})
#df.head()
errors2=[]
loo = LeaveOneOut()
# Fit polynomial models from degree 1 to 7
degrees = np.arange(1, 8)
for degree in degrees:
     # Create polynomial features
     poly = PolynomialFeatures(degree)
     X_poly = poly.fit_transform(x.reshape(-1, 1))

     # Fit the model using linear regression
     model = LinearRegression()

     # Perform LOOCV
     neg_mse = cross_val_score(model, X_poly, y, cv=loo, scoring='neg_mea
     mse = -np.mean(neg_mse) # Mean squared error
     errors2.append(mse)
# Print LOOCV errors for each model
for i, mse in enumerate(errors2):
     print(f"Degree {i+1} polynomial LOOCV error: {mse}")
```

```
Degree 1 polynomial LOOCV error: 0.00022114215496333624
Degree 2 polynomial LOOCV error: 0.00011371176055646527
Degree 3 polynomial LOOCV error: 0.00010057119012530982
Degree 4 polynomial LOOCV error: 9.740695636902108e-05
Degree 5 polynomial LOOCV error: 9.268909299573358e-05
Degree 6 polynomial LOOCV error: 9.30073090859531e-05
Degree 7 polynomial LOOCV error: 9.335757987940265e-05
```

In [85]:
```
plt.figure(figsize=(8, 6))
plt.plot(degrees, errors1, marker='o', label='Seed 123')
plt.plot(degrees, errors2, marker='o', label='Seed 12345')
plt.xlabel('Degree of Polynomial')
plt.ylabel('LOOCV Error (MSE)')
plt.legend()
plt.grid(True)
plt.show()
```



No, the results with seed 12345 differ from the results with seed 123 due to the randomness in generating the x values and the noise in y. Changing the seed affects the initial random generation of x and the random noise added to y. But the curve or the pattern of the graph remians somewhat the same. This is beacuse The cross-validation itself doesn't involve randomness as LeaveOneOut (LOOCV) uses all data points except one for each validation step. However, the difference arises from the different datasets generated due to different seeds

Part e

In [88]:
```
min_error1 = np.argmin(errors1)
print(f"Polynomial degree with the smallest error: {min_error1 + 1}")
min_error2 = np.argmin(errors2)
print(f"Polynomial degree with the smallest error: {min_error2 + 1}")
```

```
Polynomial degree with the smallest error: 5
Polynomial degree with the smallest error: 5
```

Yes, this is what I expected beaucse lower-degree Polynomials may underfit the data because the true relationship between x and y appears to be nonlinear and somewhat complex. Higher-degree Polynomials may overfit the data, capturing the noise in addition to the true pattern. 5-degree polynomial gives a right balance between these two by giving the smallest LOOCV.

Part f

In [97]:
```
import statsmodels.api as sm
from sklearn.preprocessing import PolynomialFeatures
```

```python
degree = 5
poly = PolynomialFeatures(degree)
X_poly = poly.fit_transform(x.reshape(-1, 1))
X_poly = sm.add_constant(X_poly)
# Fit the model using Ordinary Least Squares (OLS)
model = sm.OLS(y, X_poly)
results = model.fit()
print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
====
Dep. Variable:                      y   R-squared:
0.615
Model:                            OLS   Adj. R-squared:
0.611
Method:                 Least Squares   F-statistic:                     1
57.7
Date:                Fri, 27 Sep 2024   Prob (F-statistic):          6.80e
-100
Time:                        20:51:29   Log-Likelihood:                 16
18.0
No. Observations:                 500   AIC:                            -3
224.
Df Residuals:                     494   BIC:                            -3
199.
Df Model:                           5
Covariance Type:            nonrobust
==============================================================================
====
                 coef    std err          t      P>|t|      [0.025      0.
975]
------------------------------------------------------------------------------
----
const         -0.0053      0.002     -2.130      0.034      -0.010      -
0.000
x1             0.1436      0.051      2.813      0.005       0.043
0.244
x2            -0.8814      0.321     -2.749      0.006      -1.511      -
0.252
x3             3.1448      0.819      3.838      0.000       1.535
4.754
x4            -4.2798      0.909     -4.708      0.000      -6.066      -
2.494
x5             1.8827      0.364      5.172      0.000       1.167
2.598
==============================================================================
====
Omnibus:                        1.498   Durbin-Watson:
2.076
Prob(Omnibus):                  0.473   Jarque-Bera (JB):
1.536
Skew:                           0.131   Prob(JB):
0.464
Kurtosis:                       2.932   Cond. No.                     3.91
e+03
==============================================================================
====

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is cor
rectly specified.
[2] The condition number is large, 3.91e+03. This might indicate that ther
e are
strong multicollinearity or other numerical problems.
```

A low p-value (typically less than 0.05) suggests that the coefficient is statistically significant, meaning there's strong evidence that the

corresponding term contributes meaningfully to explaining the variation in y. In this case the x3,x4,x5 polynomial term are very significant while the constant,x1 and x2 terms are significant too.

In [ ]:

```
```