DATA/STAT/BIOSTAT 558
SPRING QUARTER 2025

Homework # 1
Online Submission to Canvas: due Wednesday April 16th, 5pm PST

Instructions: You may discuss the homework problems in small groups, but you must write up the final solutions and code yourself. Please turn in your code used for the problems that involve coding. However, code without written answers will receive no credit. To receive credit, you must explain your answers and show your work. All plots should be appropriately labeled and legible, with axis labels, legends, etc., as needed.

Problem 1 This problem has to do with the bias-variance trade-off and related ideas. It's okay to submit hand-sketched plots: you are not supposed to actually compute the quantities referred to below on data.

(a) Make a plot, like the one we saw in class, with "flexibility" (sometimes called complexity) of the function $\hat{f}$ on the x-axis. Sketch the following curves on the y-axis: squared bias, variance, irreducible error, reducible error, and expected prediction error. Be sure to label each curve. Indicate which level of flexibility is "best". Justify your answer (i.e., explain why this level of flexibility is best).
Ans: Figure 1

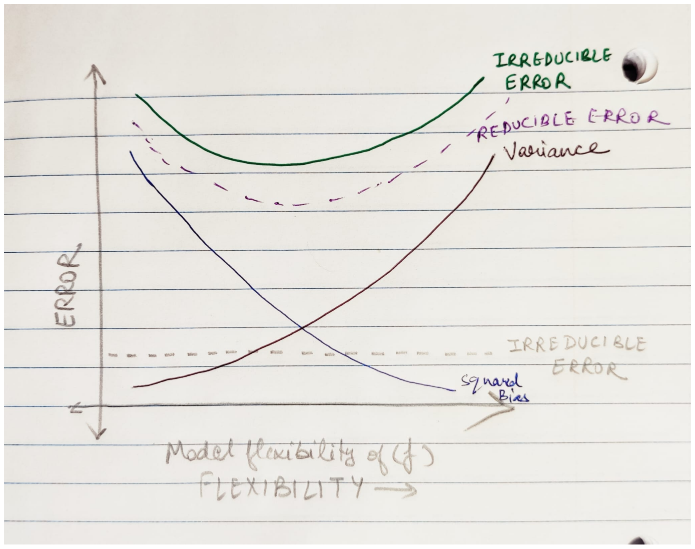

Figure 1: (1.a) Flexibility Curve

(b) For each of the following pairs of models, which do you expect to have higher bias and lower variance versus lower bias and higher variance?

   (a) Linear regression with just an intercept versus linear regression with an intercept and one predictor?
   Ans: Linear regression with just an intercept will have higher bias and lower variance because it is less flexible. The variance would be low as there are no parameters to fit. Linear regression with an intercept and one predictor will have lower bias and higher variance as it can better fit the data but may vary more across different samples. Now we have the slope parameter, it would increase model complexity ans estimates would vary from one training set to other which would result in higher variance.

   (b) A model that has a training MSE of 100 versus a model with a training MSE of 200?
   Ans: We cannot say. Just comparing training MSEs is not enough to determine which model has higher bias or variance. To truly assess bias and variance, we would need to look at how both models perform on test data.

   (c) A model that interpolates the training data versus a model that does not interpolate the training data?
   Ans: A model that interpolates the training data will have lower bias and higher variance because of overfitting, it fits every training point. A model that does not interpolate the training data is less flexible and will have higher bias and lower variance due to underfitting. This reflects the bias-variance trade-off, interpolation reduces bias at the cost of increased sensitivity to training data variation while non-interpolation smooths over data patterns but may miss the true function.

   Explain your answers.

(c) Consider two models: one has an expected prediction error of 2398, and the other has an expected prediction error of 1212. Based on this information, can you determine which model has higher squared bias and which model has higher variance? Explain your answer.
Ans: No
$EPE = Irreducible\_Error + Bias^2 + Variance$
Based on the equation we do not know the breakdown of EPE among the three terms in the equation. Hence, we cannot determine which model has higher squared bias or higher variance based on just the expected prediction errors (2398 vs. 1212).

Problem 2 In this problem, we will consider constructing a k-nearest-neighbors classifier in a setting with $p = 2$ features.

Let the first feature $X_1 \sim \text{Unif}[-1, 1]$ and second feature $X_2 \sim \text{Unif}[-1, 1]$, i.e. the observations for each feature are independent and identically distributed (i.i.d.)

from a uniform distribution.

Each observation belongs to one of two classes: the red class or the blue class. Given some value of $\tau \in (0, 1)$ to be specified below, the value of Y for a given observation is as follows:

- If $|X_1| < 0.2$ and $|X_2| < 0.2$, then $Y =$ red.
- If $|X_1| \geq 0.2$ and/or $|X_2| \geq 0.2$, $Y =$ red with probability τ, and $Y =$ blue with probability $1 - \tau$.

(a) Letting $\tau = 0.4$, generate a training set of $n = 200$ observations as described above. Plot the training data. Make sure that the axes are properly labeled, and that each observation is colored according to its class label.

Ans: Figure 2 : The training data points are plotted and are colored according to their true class labels. (Training set with $n = 200$ and $\tau = 0.4$)

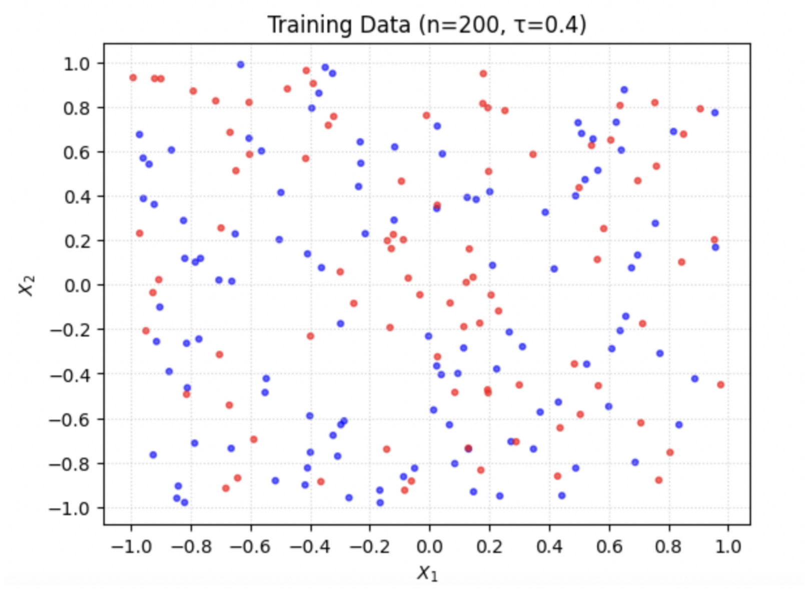

Figure 2: (2.a) Training set with $n = 200$ and $\tau = 0.4$

(b) The Bayes classifier assigns each observation to the most probable class, based on its (X_1, X_2) values. Write out a mathematical expression for the Bayes classifier for part (a).

(Hint: this should not require algebra. It should require geometry and logic.)

Ans: The Bayes classifier always chooses the class with the highest conditional probability. A point inside the square where $|X_1| < 0.2$ and $|X_2| < 0.2$ is red with a probability of 1, so the Bayes classifier predicts red. Outside that region class is blue with probability of 0.6 and red with probability of 0.4, hence the classier predicts blue in this region.

$$\hat{Y}(X_1, X_2) = \begin{cases} \text{red} & \text{if } |X_1| < 0.2 \text{ and } |X_2| < 0.2 \\ \text{blue} & \text{otherwise} \end{cases}$$

(c) On the plot from (a), color or shade each possible value of (X_1, X_2) either red or blue, depending on the prediction for Y arising from the Bayes classifier for that (X_1, X_2) pair.

(See Figure 2.13 of the textbook for guidance on what this might look like. You are being asked to color/shade EVERY possible value of (X_1, X_2), not just the pairs that happen to be in your training set.)

Ans: Figure 3 : The training data points are plotted and are colored according to their true class labels. The red shaded square at the center corresponds to the region where the Bayes classifier always predicts red. The outer blue region is where the Bayes classifier always predicts blue.

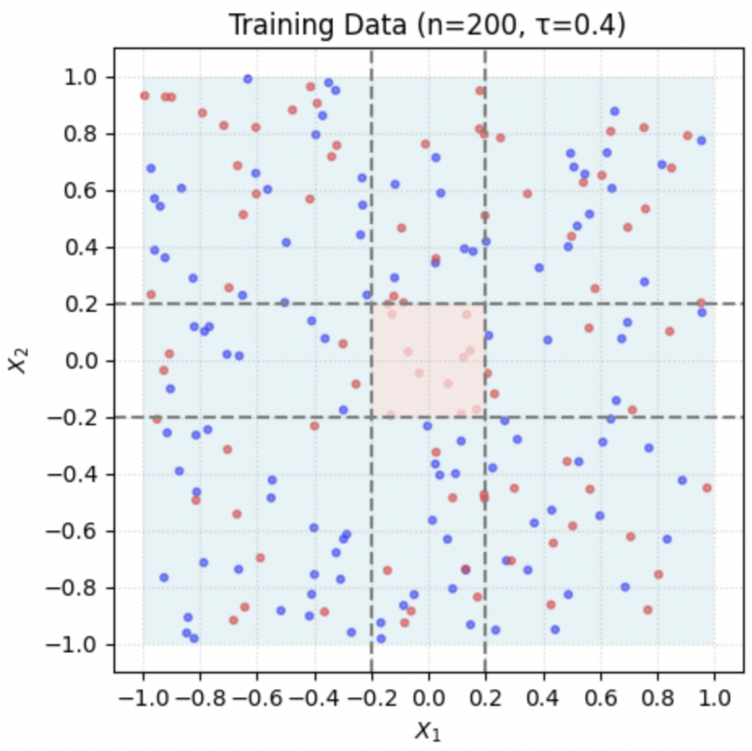

Figure 3: (2.c) Training set with shaded region based on prediction

(d) For each possible value of (X_1, X_2), provide a mathematical expression for the classification error (i.e., the probability of misclassification) of the Bayes classifier.

Ans: The Bayes classification error:

$$Error = \begin{cases} 0 & \text{if } |X_1| < 0.2 \text{ and } |X_2| < 0.2 \\ 0.4 & \text{otherwise} \end{cases}$$

When $|X_1| < 0.2$ and $|X_2| < 0.2$, the point is certainly red and the classifier also predicts red, hence the probability of error is 0. For the remaining region the classifier predicts blue. The point is red with probability τ (0.4), so the error is only when the point is red.

(e) The expected classification error of the Bayes classifier is called the Bayes error. Compute the Bayes error.

(Hint: The Bayes error is the *expected* classification error, where the expectation is computed over the distribution of the X's. Thus, to compute the Bayes error, average the errors you computed in (d) over the distribution of (X_1, X_2). To do this, you can take advantage of the fact that X_1 and X_2 are uniformly distributed on $[-1, 1]$.)
Ans: B$ayesError = E[$C$lassificationError]$
Proportion of area of inner square: $0.4 * 0.4/4 = 0.16/4 = 0.4$
Bayes Error in inner square: $0 * 0.4 = 0$
Proportion of area of remaining square: $(2 * 2 - 0.16)/4 = 3.84/4$
Bayes Error in remaining region: $0.4 * 3.84/4 = 0.384$
Bayes Error $= 0 + 3.84 = 0.384$

(f) Now generate a test set consisting of another 200 observations. Fit a k-nearest neighbors model on the training set, for a range of values of k from 1 to $n/2$. Make a plot that displays the value of $1/k$ on the x-axis, and classification error (both training error and test error) on the y-axis. Display the Bayes error rate (computed in (e)) as a horizontal line. Make sure all axes and curves are properly labeled. Explain your results.
Ans: Figure 4: The Bayes error remains constant across different values of k as it reflects the irreducible error in the data. As k increases the model becomes less flexible, the training error increases as the model becomes more regularized and is less likely to overfit. For higher k, the model generalizes better to unseen data. The test error initially decreases and then increases. This pattern shows the classic bias-variance trade off suggesting that there is optimal balance between bias and variance. At low k, the model overfits capturing noise in the training data. At high k, it underfits and fails to capture the relevant structure.

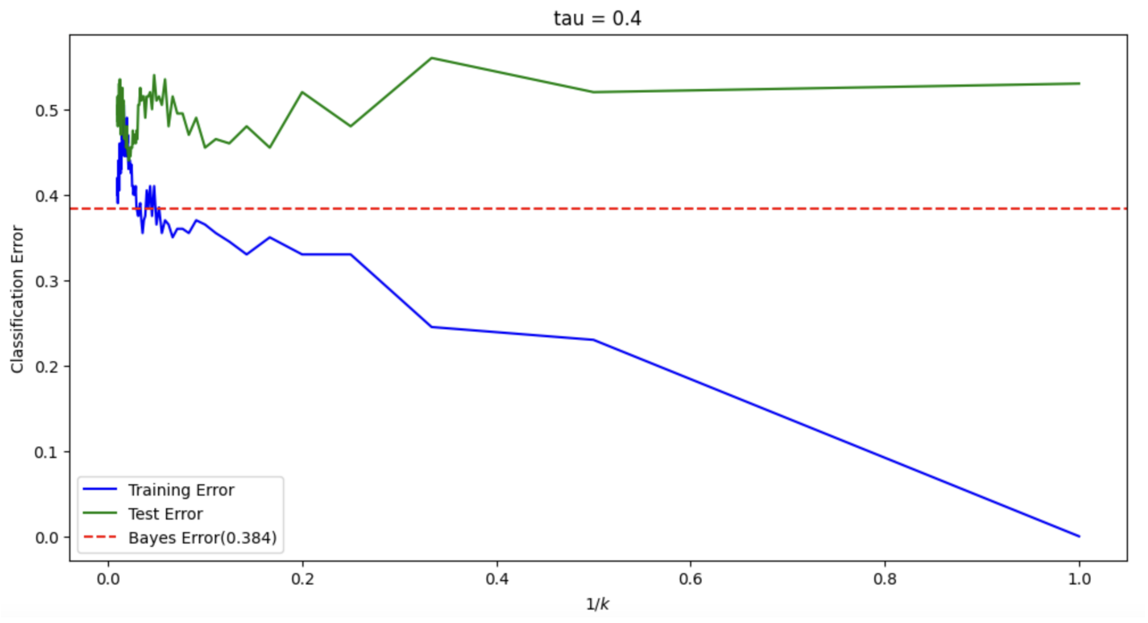

Figure 4: (2.f) Classification Error for K-NN Model

(g) Repeat the above steps (a)-(f), but with $\tau = 0.1$. How do your results differ? Explain your answer.
Ans: (a) Figure 5 : The training data points are plotted and are colored according to their true class labels. (Training set with $n = 200$ and $\tau = 0.1$)

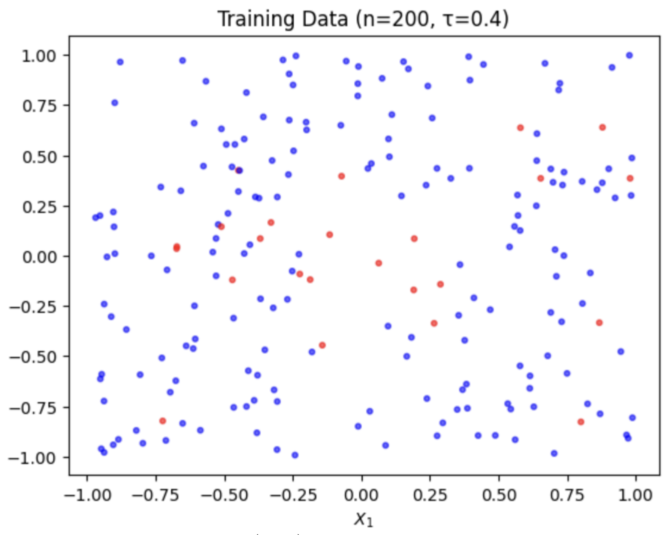

Figure 5: (2.f) Traning data points

(b)

$$\hat{Y}(X_1, X_2) = \begin{cases} \text{red} & \text{if } |X_1| < 0.2 \text{ and } |X_2| < 0.2 \\ \text{blue} & \text{otherwise} \end{cases}$$

(c) Figure 6 The training data points are plotted and are colored according to their true class labels. The red shaded square at the center corresponds to the region where the Bayes classifier always predicts red. The outer blue region is where the Bayes classifier always predicts blue.

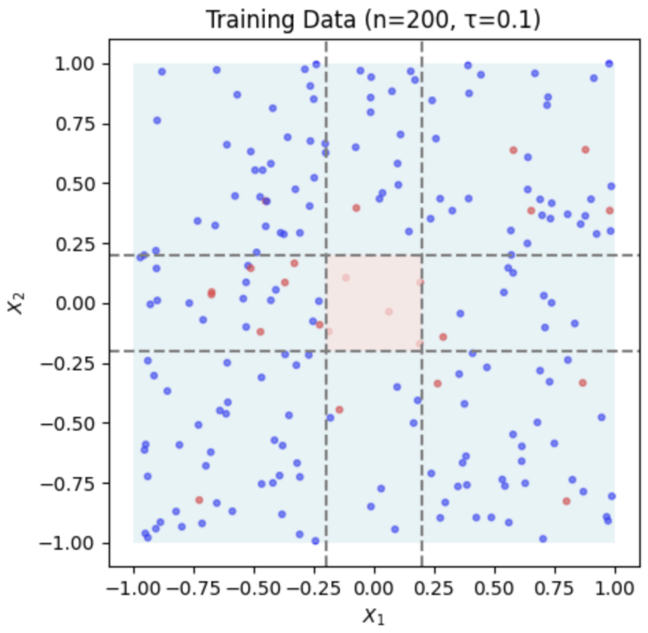

Figure 6: (2.f) Training set with shaded region based on prediction

(d)The Bayes classification error:

$$Error = \begin{cases} 0 & \text{if } |X_1| < 0.2 \text{ and } |X_2| < 0.2 \\ 0.1 & \text{otherwise} \end{cases}$$

(e) B$ayesError = E[$C$lassificationError]$
Proportion of area of inner square: $0.4 * 0.4/4 = 0.16/4 = 0.4$
Bayes Error in inner square: $0 * 0.4 = 0$
Proportion of area of remaining square: $(2 * 2 - 0.16)/4 = 3.84/4$
Bayes Error in remaining region: $0.1 * 3.84/4 = 0.096$
Bayes Error $= 0 + 3.84 = 0.096$

(f) Figure 7: The Bayes error remains constant across different values of k as it reflects the irreducible error in the data. The model becomes more flexible with decreasing k ultimately interpolating the training data when k=1, resulting in near-zero training error. However, this flexibility also leads to overfitting which increases the test error due to high variance. The test error curve exhibits the classic U-shape, with an optimal value of k that minimizes the error by balancing bias and variance. The red dashed line represents the Bayes error (0.096) which is the theoretical lower bound on classification error. With $\tau = 0.1$, the data is relatively clean so the test error approaches the

Bayes error achieving near-optimal performance highlighting the effectiveness of k-NN when appropriately regularized.

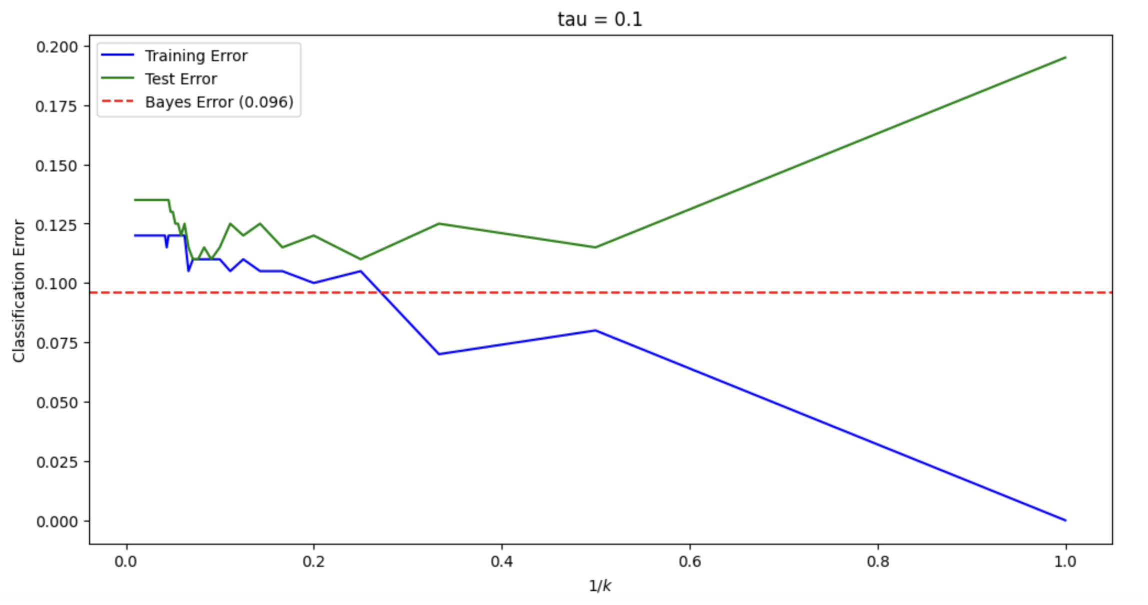

Figure 7: (2.f) Classification Error for K-NN Model

Problem 3 Consider patient data consisting of the measured amount of an antibody (covariate X) and whether the patient became infected with the flu within a year after this measurement (outcome Y).

(a) Given an example of a prediction task involving this data. What quantities might be reported in an analysis involving this prediction task?
Ans: Example: Given a patients antibody level, predict whether a patient will become infected with the flu within a year based on the measured amount of antibody.
Quantities that might be reported in the prediction task would be

- Prediction probability (eg. "there is a 10% chance of a patient getting the flu")
- Model accuracy: measures how often the model makes correct predictions $P(y = \text{infected}|X = x_i)$
- Predicted class label (0, 1) for each patient
- ROC curve and AUC: The ROC curve visualizes the trade-off between true positive rate and false positive rate at various thresholds.
- Cross-validation performance: This estimates the performance by evaluating the model across multiple training and validation sets.

(b) Given an example of an inference task involving this data. What quantities might be reported in an analysis involving this inference task?

Ans: Example for inference the data: Given a patients antibody level, determine whether higher levels of antibody are associated with a lower probability of flu infection within a year.
Quantities that might be reported in the inference task would be:

- Model coefficients or regression coefficients
- Confidence Intervals: getting a range of values within which the true estimate lies
- p-values testing null hypothesis of no association
- Assumptions behind the model: the conditions that must hold for the inference to be true.

Problem 4 Consider two functions, $f_1(X) := 1+X$ and $f_2(X) := 1+X+X^2+X^3$, where $X \in \mathbb{R}$. Suppose that for $k = 1, 2$, we observe $Y^{(k)} = f_k(X) + \varepsilon$, where $\varepsilon \sim \text{Norm}(0, 1)$. Our goal is to estimate f_1 and f_2 using $Y^{(1)}$ and $Y^{(2)}$, respectively.

In what follows, we will generate the observations of X according to $X \sim \text{Norm}(0, 1)$.

(a) During Lecture #1, we talked about the "irreducible error" associated with the expected prediction error, in the context of our discussion of the bias-variance trade-off. Write down the mathematical expression for the "irreducible error" that we saw in lecture.
Ans: Irreducible Error = $Var(\varepsilon)$

(b) What is the irreducible error associated with estimating $f_1(\cdot)$ from $Y^{(1)}$? What is the irreducible error associated with estimating $f_2(\cdot)$ from $Y^{(2)}$?
Ans: Irreducible Error (estimating $f_1(\cdot)$ from $Y^{(1)}$) = $Var(\varepsilon) = 1$
Irreducible Error (estimating $f_2(\cdot)$ from $Y^{(2)}$) = $Var(\varepsilon) = 1$

(c) For each $k = 1, 2$:

- Simulate a test dataset consisting of 10,000 independent $(X_i, Y_i^{(k)})$ pairs.
- Simulate $1,000$ training datasets, each of which consists of 5,000 independent $(X_i, Y_i^{(k)})$ pairs.

  (Note: in a real data analysis, we would of course only have access to one training dataset. However, in this homework problem we will work in an imagined scenario where we have access to a HUGE number of training datasets, in order to strengthen our understanding of the bias-variance trade-off.)
- On each training dataset, fit a linear model $\hat{f}^{(k)}$ to predict $Y^{(k)}$ (you can use, e.g., the `lm` function in **R** to fit this model), with an intercept and the single feature X.
- For each value x_0 in the test dataset, estimate the squared bias $(\text{E}[\hat{f}^{(k)}(x_0)] - f_k(x_0))^2$, using the 1,000 training sets.

- For each value x_0 in the test dataset, estimate the variance $\mathrm{Var}[\hat{f}^{(k)}(x_0)]$, using the 1,000 training sets.
- Recall that the expected prediction error at a given point x_0 is the sum of the (i) irreducible error, (ii) squared bias of $\hat{f}^{(k)}(x_0)$, and (iii) variance of $\hat{f}^{(k)}(x_0)$. Combine your answers to the previous subproblems in order to estimate the expected prediction error at each value x_0 in the test dataset. Plot the expected prediction error as a function of x_0.

Ans: Figure 8 : In the left plot, where the true function is linear, the linear model is perfectly specified. As a result, the EPE is nearly constant and close to 1 across the entire range. The error is almost due to the irreducible error. There is effectively no bias because the model matches the true function form and the variance is also minimal due to the simplicity of the model and ample training data.
In contrast, the right plot corresponds to the case where the true function is nonlinear. Here, the linear model is a poor fit especially for values of far from zero. The EPE increases rapidly as moves away from the origin, illustrating the high bias introduced by trying to approximate a cubic function with a linear model. Even though the variance remains small the squared bias grows significantly, particularly because the higher-degree terms dominate the behavior of at the tails.
When the model is appropriately chosen, EPE is low and stable but when the model is too simple for a more complex true function, EPE can become large due to high bias.

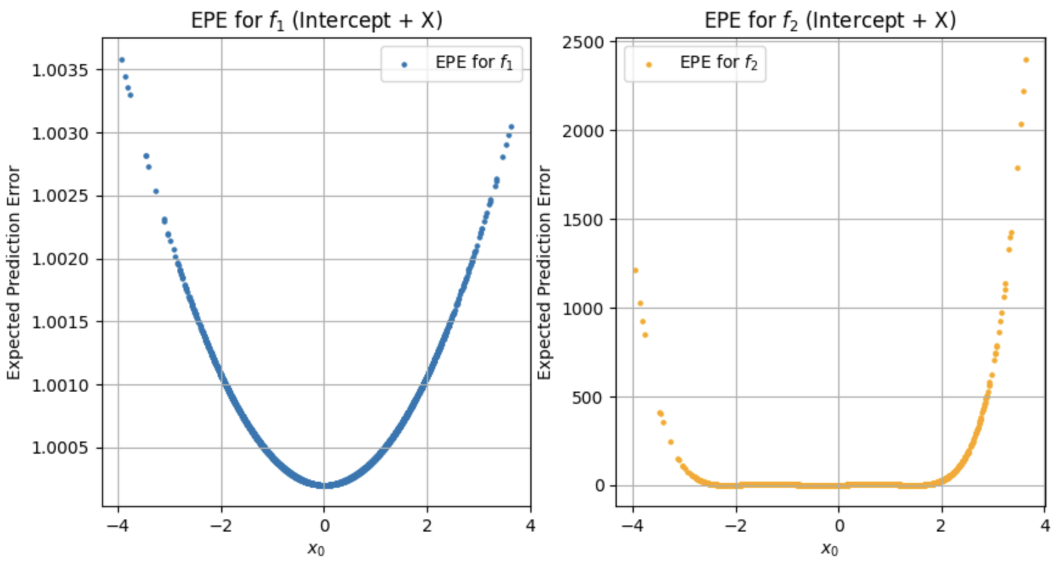

Figure 8: (4.c) Expected Prediction Error vs x_0 (Intercept + X)

(d) Repeat (c), but this time fit a linear model containing an intercept, and features X, X^2, and X^3.

Ans: Figure 9 : In the left plot, where the true function is linear, we are overfitting by using a model that includes unnecessary higher-order terms. Despite this, the EPE remains very close to 1 across the entire range, reflecting mostly the irreducible error. There is some slight increase in EPE as compared to the part (c) which comes from the increase in variance introduced by fitting a more flexible model than necessary. However, the bias remains very low because the model still includes the correct linear term.
In the right plot, where the true function is non linear, the cubic model is correctly specified as it matches the form of the true function exactly. As expected, the EPE is extremely close to 1 for the entire range suggesting that the model is both unbiased and has low variance. The remaining error is almost entirely due to the irreducible error. This reflects a much better fit.

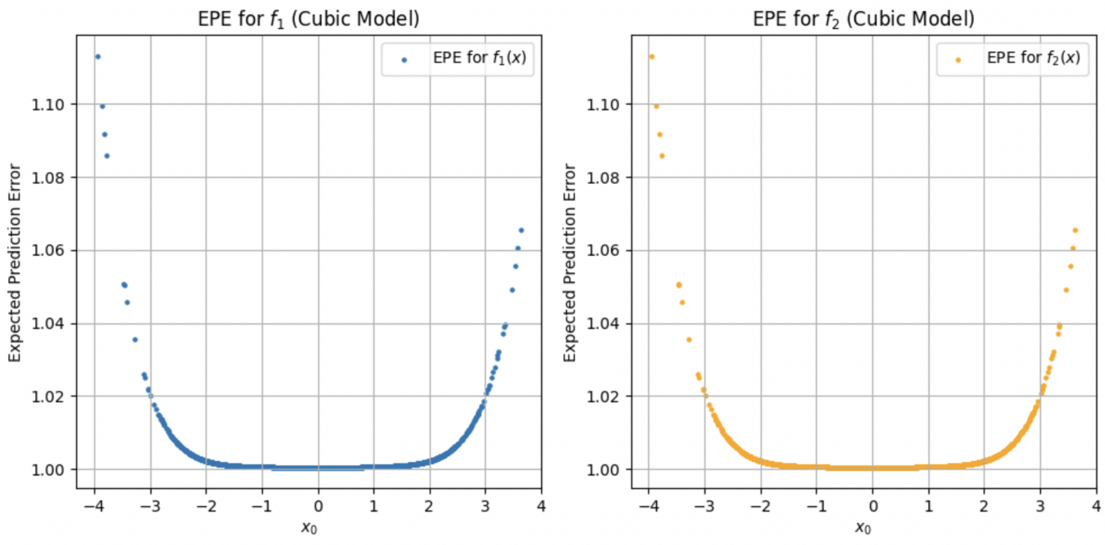

Figure 9: (4.d) Expected Prediction Error vs x_0 (Cubic)

(e) Repeat (c), but this time fit a linear model containing just an intercept.
Ans: Figure 10 : In the left plot, where the true function is linear, the model is highly biased because it tries to approximate a linear function with a constant. As a result, the EPE is U-shaped increasing rapidly as $|x_0|$ increases.
In the right plot, where the true function is non linear, the situation is even more extreme. The result is a rapid explosion in EPE for large $|x_0|$, reaching values above 4000 in the tails.

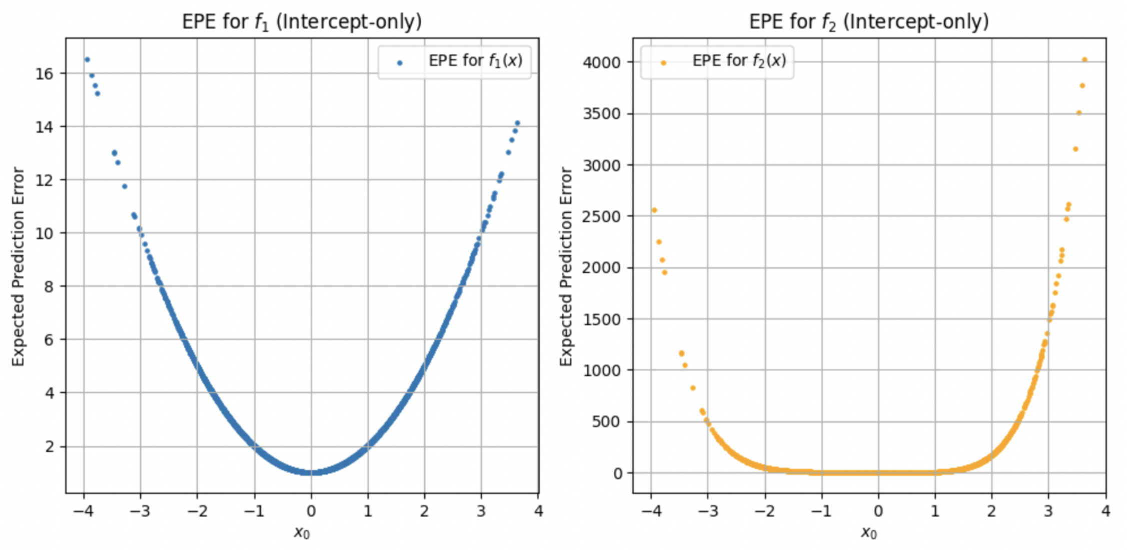

Figure 10: (4.e) Expected Prediction Error vs x_0 (Intercept-Only)

(f) Compare and contrast your results in (c)–(e). Interpret your results in terms of the bias-variance trade-off.
Ans: The results demonstrate the bias-variance trade-off in model selection. As model complexity increases from intercept-only to cubic, bias generally decreases while variance increases.
For f1: As model complexity increases from intercept only model to linear and then to the cubic, The average squared bias decreases and average variance increases. This follows the expected bias-variance trade off.

Model	**f1: Mean Squared Bias**	**f1: Mean Var**	**f1: Mean EPE**
Intercept-only	$9.861484e^{-1}$	0.000394	1.986542
Linear (X)	$4.172067e^{-7}$	0.000416	1.000416
Cubic	$3.952306e^{-7}$	0.000834	1.000834

For f2: As model complexity increases from intercept only model to linear and then to the cubic, the average squared bias decreases indicating a better fit for the true function. The average variance slightly increases as more complex models tend to me more sensitive to training data.

Model	**f2: Mean Squared Bias**	**f2: Mean Var**	**f2: Mean EPE**
Intercept-only	$2.415825e^{01}$	0.004909	25.163161
Linear (X)	$8.291678e^{0}$	0.013280	9.304958
Cubic	$3.952306e^{-7}$	0.000834	1.000834

For f1, the linear model is a good fit where as for f2 the cubic model is a better fit.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.patches as patches
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
```

```
np.random.seed(10)
def genData(tau):
  n=200
  X1 = np.random.uniform(-1, 1, n)
  X2 = np.random.uniform(-1, 1, n)
  Y = np.empty(n, dtype=object)
  for i in range(n):
    if abs(X1[i]) < 0.2 and abs(X2[i]) < 0.2:
        Y[i] = 'red'
    else:
        Y[i] = 'red' if np.random.rand() <= tau else 'blue'
  return X1, X2, Y

X1_train, X2_train, Y_train = genData(0.4)
```

```
#Question 2.a
plt.scatter(X1_train, X2_train, s=10, c=Y_train, alpha=0.6)
plt.title('Training Data (n=200, τ=0.4)')
plt.xlabel('$X_1$')
plt.ylabel('$X_2$')
plt.xticks(np.arange(-1.0, 1.1, 0.2))
plt.yticks(np.arange(-1.0, 1.1, 0.2))
plt.grid(True, linestyle=':', alpha=0.5)
plt.show()
```

```
#Question 2.c
fig, ax = plt.subplots(figsize=(5, 5))
ax.scatter(X1_train, X2_train, s=10, c=Y_train, alpha=0.6)
square1 = patches.Rectangle((-1, -1), 2, 2, facecolor='lightblue', alpha=0.3)
square2 = patches.Rectangle((-0.2, -0.2), 0.4, 0.4, facecolor='mistyrose',
                            alpha=0.7)
ax.add_patch(square1)
ax.add_patch(square2)
plt.title('Training Data (n=200, τ=0.4)')
plt.xlabel('$X_1$')
plt.ylabel('$X_2$')
plt.axvline(x=0.2, color='gray', linestyle='--')
plt.axvline(x=-0.2, color='gray', linestyle='--')
plt.axhline(y=0.2, color='gray', linestyle='--')
plt.axhline(y=-0.2, color='gray', linestyle='--')
ax.set_xticks(np.arange(-1.0, 1.1, 0.2))
ax.set_yticks(np.arange(-1.0, 1.1, 0.2))
plt.grid(True, linestyle=':', alpha=0.5)

plt.show()


X1_test, X2_test, Y_test = genData(0.4)


train_errors = []
test_errors = []
k_range = range(1, 101)
Y_train_numeric = np.array([1 if y == 'red' else 0 for y in Y_train])
Y_test_numeric = np.array([1 if y == 'red' else 0 for y in Y_test])
for k in k_range:
  knn = KNeighborsClassifier(n_neighbors=k)
  knn.fit(np.array([X1_train, X2_train]).T, Y_train_numeric)

  Y_train_pred = knn.predict(np.array([X1_train, X2_train]).T)
  Y_test_pred = knn.predict(np.array([X1_test, X2_test]).T)

  train_error = 1 - accuracy_score(Y_train_numeric, Y_train_pred)
  test_error = 1 - accuracy_score(Y_test_numeric, Y_test_pred)

  train_errors.append(train_error)
  test_errors.append(test_error)
```

```
#Question 2.f
fig, ax = plt.subplots(figsize=(12, 6))
ax.plot(1 / np.array(k_range), train_errors, label='Training Error',
        color='blue')
ax.plot(1 / np.array(k_range), test_errors, label='Test Error', color='green')
ax.axhline(y=0.384, color='red', linestyle='--', label='Bayes Error(0.384)')
ax.set_xlabel(r'$1/k$')
ax.set_ylabel('Classification Error')
ax.set_title(f'tau = 0.4')
ax.legend()
plt.show()


#Question 2.g
np.random.seed(30)
X1_train, X2_train, Y_train = genData(0.1)
plt.scatter(X1_train, X2_train, s=10, c=Y_train, alpha=0.6)
plt.title('Training Data (n=200, τ=0.4)')
plt.xlabel('$X_1$')
plt.ylabel('$X_2$')
plt.show()


fig, ax = plt.subplots(figsize=(5, 5))
ax.scatter(X1_train, X2_train, s=10, c=Y_train, alpha=0.6)
square1 = patches.Rectangle((-1, -1), 2, 2, facecolor='lightblue', alpha=0.3)
square2 = patches.Rectangle((-0.2, -0.2), 0.4, 0.4, facecolor='mistyrose',
                            alpha=0.7)
ax.add_patch(square1)
ax.add_patch(square2)
plt.title('Training Data (n=200, τ=0.1)')
plt.xlabel('$X_1$')
plt.ylabel('$X_2$')
plt.axvline(x=0.2, color='gray', linestyle='--')
plt.axvline(x=-0.2, color='gray', linestyle='--')
plt.axhline(y=0.2, color='gray', linestyle='--')
plt.axhline(y=-0.2, color='gray', linestyle='--')
plt.show()


X1_test, X2_test, Y_test = genData(0.1)
```

```
train_errors = []
test_errors = []
k_range = range(1, 101)
Y_train_numeric = np.array([1 if y == 'red' else 0 for y in Y_train])
Y_test_numeric = np.array([1 if y == 'red' else 0 for y in Y_test])
for k in k_range:
  knn = KNeighborsClassifier(n_neighbors=k)
  knn.fit(np.array([X1_train, X2_train]).T, Y_train_numeric)

  Y_train_pred = knn.predict(np.array([X1_train, X2_train]).T)
  Y_test_pred = knn.predict(np.array([X1_test, X2_test]).T)

  train_error = 1 - accuracy_score(Y_train_numeric, Y_train_pred)
  test_error = 1 - accuracy_score(Y_test_numeric, Y_test_pred)

  train_errors.append(train_error)
  test_errors.append(test_error)


#2.f
fig, ax = plt.subplots(figsize=(12, 6))
ax.plot(1 / np.array(k_range), train_errors, label='Training Error',
        color='blue')
ax.plot(1 / np.array(k_range), test_errors, label='Test Error', color='green')
ax.axhline(y=0.096, color='red', linestyle='--', label='Bayes Error (0.096)')
ax.set_xlabel(r'$1/k$')
ax.set_ylabel('Classification Error')
ax.set_title(f'tau = 0.1')
ax.legend()
plt.show()


#Question 4


np.random.seed(50)
```

```
def f1(X):
    return 1 + X

def f2(X):
    return 1 + X + X**2 + X**3

num_test = 10000
num_train_sets = 1000
train_size = 5000
X_test = np.random.normal(0, 1, num_test)
epsilon_test = np.random.normal(0, 1, num_test)
Y1_test = f1(X_test) + epsilon_test
Y2_test = f2(X_test) + epsilon_test
X_train_sets = np.random.normal(0, 1, (num_train_sets, train_size))
epsilon_train_sets = np.random.normal(0, 1, (num_train_sets, train_size))
Y1_train_sets = f1(X_train_sets) + epsilon_train_sets
Y2_train_sets = f2(X_train_sets) + epsilon_train_sets



#Question 4.c
preds_f1 = np.zeros((num_train_sets, num_test))
preds_f2 = np.zeros((num_train_sets, num_test))
X_test_reshaped = X_test.reshape(-1, 1)

for i in range(num_train_sets):
    X_train = X_train_sets[i].reshape(-1, 1)
    y_train_f1 = Y1_train_sets[i]
    model_f1 = LinearRegression().fit(X_train, y_train_f1)
    preds_f1[i] = model_f1.predict(X_test_reshaped)

    y_train_f2 = Y2_train_sets[i]
    model_f2 = LinearRegression()
    model_f2.fit(X_train, y_train_f2)
    preds_f2[i] = model_f2.predict(X_test_reshaped)

mean_preds_f1 = np.mean(preds_f1, axis=0)
mean_preds_f2 = np.mean(preds_f2, axis=0)
squared_bias_f1 = (mean_preds_f1 - f1(X_test)) ** 2
squared_bias_f2 = (mean_preds_f2 - f2(X_test)) ** 2
variance_f1 = np.var(preds_f1, axis=0)
variance_f2 = np.var(preds_f2, axis=0)
irreducible_error = np.array([1]*10000)
expected_error_f1 = irreducible_error + squared_bias_f1 + variance_f1
expected_error_f2 = irreducible_error + squared_bias_f2 + variance_f2


sorted_idx = np.argsort(X_test)
```

```python
x_vals = X_test[sorted_idx]
epe_f1_sorted = expected_error_f1[sorted_idx]
epe_f2_sorted = expected_error_f2[sorted_idx]

# Plotting
fig, axes = plt.subplots(1, 2, figsize=(10, 5))
axes[0].scatter(x_vals, epe_f1_sorted,s=5, label='EPE for $f_1$')
axes[0].set_title('EPE for $f_1$ (Intercept + X)')
axes[0].set_xlabel('$x_0$')
axes[0].set_ylabel('Expected Prediction Error')
axes[0].grid(True)
axes[0].legend()

axes[1].scatter(x_vals, epe_f2_sorted,s=5, label='EPE for $f_2$',
                color='orange')
axes[1].set_title('EPE for $f_2$ (Intercept + X)')
axes[1].set_xlabel('$x_0$')
axes[1].set_ylabel('Expected Prediction Error')
axes[1].grid(True)
axes[1].legend()
plt.show()


#Question 4d
poly = PolynomialFeatures(degree=3, include_bias=False)
X_test_poly = poly.fit_transform(X_test.reshape(-1, 1))
preds_f1_cubic = np.zeros((num_train_sets, num_test))
preds_f2_cubic = np.zeros((num_train_sets, num_test))

for i in range(num_train_sets):
    X_train = X_train_sets[i].reshape(-1, 1)
    X_train_poly = poly.transform(X_train)

    # Fit model on f1 training data
    y_train_f1 = Y1_train_sets[i]
    model_f1 = LinearRegression().fit(X_train_poly, y_train_f1)
    preds_f1_cubic[i] = model_f1.predict(X_test_poly)

    # Fit model on f2 training data
    y_train_f2 = Y2_train_sets[i]
    model_f2 = LinearRegression().fit(X_train_poly, y_train_f2)
    preds_f2_cubic[i] = model_f2.predict(X_test_poly)

# Compute EPE components
mean_preds_f1_cubic = np.mean(preds_f1_cubic, axis=0)
mean_preds_f2_cubic = np.mean(preds_f2_cubic, axis=0)

squared_bias_f1_cubic = (mean_preds_f1_cubic - f1(X_test))**2
```

```
squared_bias_f2_cubic = (mean_preds_f2_cubic - f2(X_test))**2

variance_f1_cubic = np.var(preds_f1_cubic, axis=0)
variance_f2_cubic = np.var(preds_f2_cubic, axis=0)

irreducible_error = np.ones_like(X_test)

expected_error_f1_cubic = irreducible_error + squared_bias_f1_cubic +
 variance_f1_cubic
expected_error_f2_cubic = irreducible_error + squared_bias_f2_cubic +
 variance_f2_cubic



sorted_idx = np.argsort(X_test)
x_vals = X_test[sorted_idx]
epe_f1_sorted = expected_error_f1_cubic[sorted_idx]
epe_f2_sorted = expected_error_f2_cubic[sorted_idx]

# Plot EPE side-by-side with different axes
fig, axes = plt.subplots(1, 2, figsize=(10, 5))

axes[0].scatter(x_vals, epe_f1_sorted,s=5, label='EPE for $f_1(x)$')
axes[0].set_title('EPE for $f_1$ (Cubic Model)')
axes[0].set_xlabel('$x_0$')
axes[0].set_ylabel('Expected Prediction Error')
axes[0].grid(True)
axes[0].legend()

axes[1].scatter(x_vals, epe_f2_sorted,s=5, label='EPE for $f_2(x)$',
                color='orange')
axes[1].set_title('EPE for $f_2$ (Cubic Model)')
axes[1].set_xlabel('$x_0$')
axes[1].set_ylabel('Expected Prediction Error')
axes[1].grid(True)
axes[1].legend()

plt.tight_layout()
plt.show()


#Question 4e
preds_f1_const = np.zeros((num_train_sets, num_test))
preds_f2_const = np.zeros((num_train_sets, num_test))

# Create a dummy input of all 1s (since we want to fit only an intercept)
X_dummy_train = np.ones((train_size, 1))
X_dummy_test = np.ones((num_test, 1))

for i in range(num_train_sets):
```

```
for i in range(num_train_sets):
    # f1 training
    y_train_f1 = Y1_train_sets[i]
    model_f1 = LinearRegression(fit_intercept=False)
    model_f1.fit(X_dummy_train, y_train_f1)
    preds_f1_const[i] = model_f1.predict(X_dummy_test)

    # f2 training
    y_train_f2 = Y2_train_sets[i]
    model_f2 = LinearRegression(fit_intercept=False)
    model_f2.fit(X_dummy_train, y_train_f2)
    preds_f2_const[i] = model_f2.predict(X_dummy_test)

# Compute mean predictions
mean_preds_f1_const = np.mean(preds_f1_const, axis=0)
mean_preds_f2_const = np.mean(preds_f2_const, axis=0)

# Compute squared bias, variance, and EPE
squared_bias_f1_const = (mean_preds_f1_const - f1(X_test))**2
squared_bias_f2_const = (mean_preds_f2_const - f2(X_test))**2

variance_f1_const = np.var(preds_f1_const, axis=0)
variance_f2_const = np.var(preds_f2_const, axis=0)

irreducible_error = np.ones_like(X_test)

expected_error_f1_const = irreducible_error + squared_bias_f1_const +
 variance_f1_const
expected_error_f2_const = irreducible_error + squared_bias_f2_const +
 variance_f2_const

# Sort for plotting
sorted_idx = np.argsort(X_test)
x_vals = X_test[sorted_idx]
epe_f1_sorted = expected_error_f1_const[sorted_idx]
epe_f2_sorted = expected_error_f2_const[sorted_idx]

# Plot side-by-side
fig, axes = plt.subplots(1, 2, figsize=(10, 5))

axes[0].scatter(x_vals, epe_f1_sorted,s=5, label='EPE for $f_1(x)$')
axes[0].set_title('EPE for $f_1$ (Intercept-only)')
axes[0].set_xlabel('$x_0$')
axes[0].set_ylabel('Expected Prediction Error')
axes[0].grid(True)
axes[0].legend()

axes[1].scatter(x_vals, epe_f2_sorted,s=5, label='EPE for $f_2(x)$',
               color='orange')
```

```
axes[1].set_title('EPE for $f_2$ (Intercept-only)')
axes[1].set_xlabel('$x_0$')
axes[1].set_ylabel('Expected Prediction Error')
axes[1].grid(True)
axes[1].legend()

plt.tight_layout()
plt.show()
```

```
# Question 4f
irreducible_error=1
mean_bias_f1_linear = np.mean(squared_bias_f1)
mean_var_f1_linear = np.mean(variance_f1)
mean_epe_f1_linear = mean_bias_f1_linear + mean_var_f1_linear +
 irreducible_error

mean_bias_f1_cubic = np.mean(squared_bias_f1_cubic)
mean_var_f1_cubic = np.mean(variance_f1_cubic)
mean_epe_f1_cubic = mean_bias_f1_cubic + mean_var_f1_cubic + irreducible_error

mean_bias_f1_const = np.mean(squared_bias_f1_const)
mean_var_f1_const = np.mean(variance_f1_const)
mean_epe_f1_const = mean_bias_f1_const + mean_var_f1_const + irreducible_error

# Compute means for f2
mean_bias_f2_linear = np.mean(squared_bias_f2)
mean_var_f2_linear = np.mean(variance_f2)
mean_epe_f2_linear = mean_bias_f2_linear + mean_var_f2_linear +
 irreducible_error

mean_bias_f2_cubic = np.mean(squared_bias_f2_cubic)
mean_var_f2_cubic = np.mean(variance_f2_cubic)
mean_epe_f2_cubic = mean_bias_f2_cubic + mean_var_f2_cubic + irreducible_error

mean_bias_f2_const = np.mean(squared_bias_f2_const)
mean_var_f2_const = np.mean(variance_f2_const)
mean_epe_f2_const = mean_bias_f2_const + mean_var_f2_const + irreducible_error

df_summary = pd.DataFrame({
    'Model': ['Intercept-only', 'Linear', 'Cubic'],
    'f1: Bias²': [mean_bias_f1_const, mean_bias_f1_linear, mean_bias_f1_cubic],
    'f1: Var':   [mean_var_f1_const, mean_var_f1_linear, mean_var_f1_cubic],
    'f1: EPE':   [mean_epe_f1_const, mean_epe_f1_linear, mean_epe_f1_cubic],
    'f2: Bias²': [mean_bias_f2_const, mean_bias_f2_linear, mean_bias_f2_cubic],
    'f2: Var':   [mean_var_f2_const, mean_var_f2_linear, mean_var_f2_cubic],
    'f2: EPE':   [mean_epe_f2_const, mean_epe_f2_linear, mean_epe_f2_cubic],
})

print(df_summary)
```

Start coding or generate with AI.