P2

P2 :

Proof: Number that minimizes mean squared error is the mean of the dataset.

$$f(c) = \frac{1}{N}\left[(x_1 - c)^2 + (x_2 - c)^2 + \ldots + (x_N - c)^2\right]$$

$$f'(c) = \left[\frac{d}{dc}\left[(x_1 - c)^2\right] + \frac{d}{dc}\left[(x_2 - c)^2\right] \cdots\right)\frac{1}{N}$$

$$\downarrow$$

$$\frac{d}{dc}\left[x_1^2 - 2x_1c + c^2\right]$$

$$= \frac{1}{N}\left(\left[2c - 2x_1\right] + \left[2c - 2x_2\right] + \ldots \left[2c - 2x_n\right]\right)$$

$$f'(c) = \frac{1}{N}\left(2Nc - 2\left[x_1 + x_2 + \ldots + x_N\right]\right)$$

$$f'(c) = 2c - \frac{2}{N}[x_1 + x_2 + \dots + x_N]$$

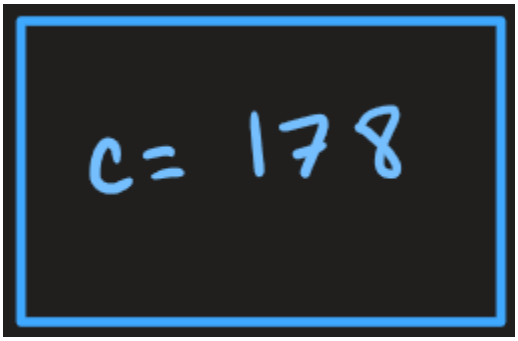
$$f'(c) = 2\left[c - \frac{1}{N}[x_1 + x_2 + \dots + x_N]\right]$$

$$f'(c) = 0 = 2\left[c - \frac{1}{N}[x_1 + x_2 + \dots + x_N]\right]$$

$$0 = c - \frac{1}{N}[x_1 + x_2 + \dots + x_N]$$

$$\boxed{c = \frac{1}{N}[x_1 + x_2 + \dots + x_N]}$$

$$c = \frac{1}{4}(175 + 172 + 180 + 185)$$


c= 178

P3

$Y = f(X) + \varepsilon$, ε is independent noise

Minimize

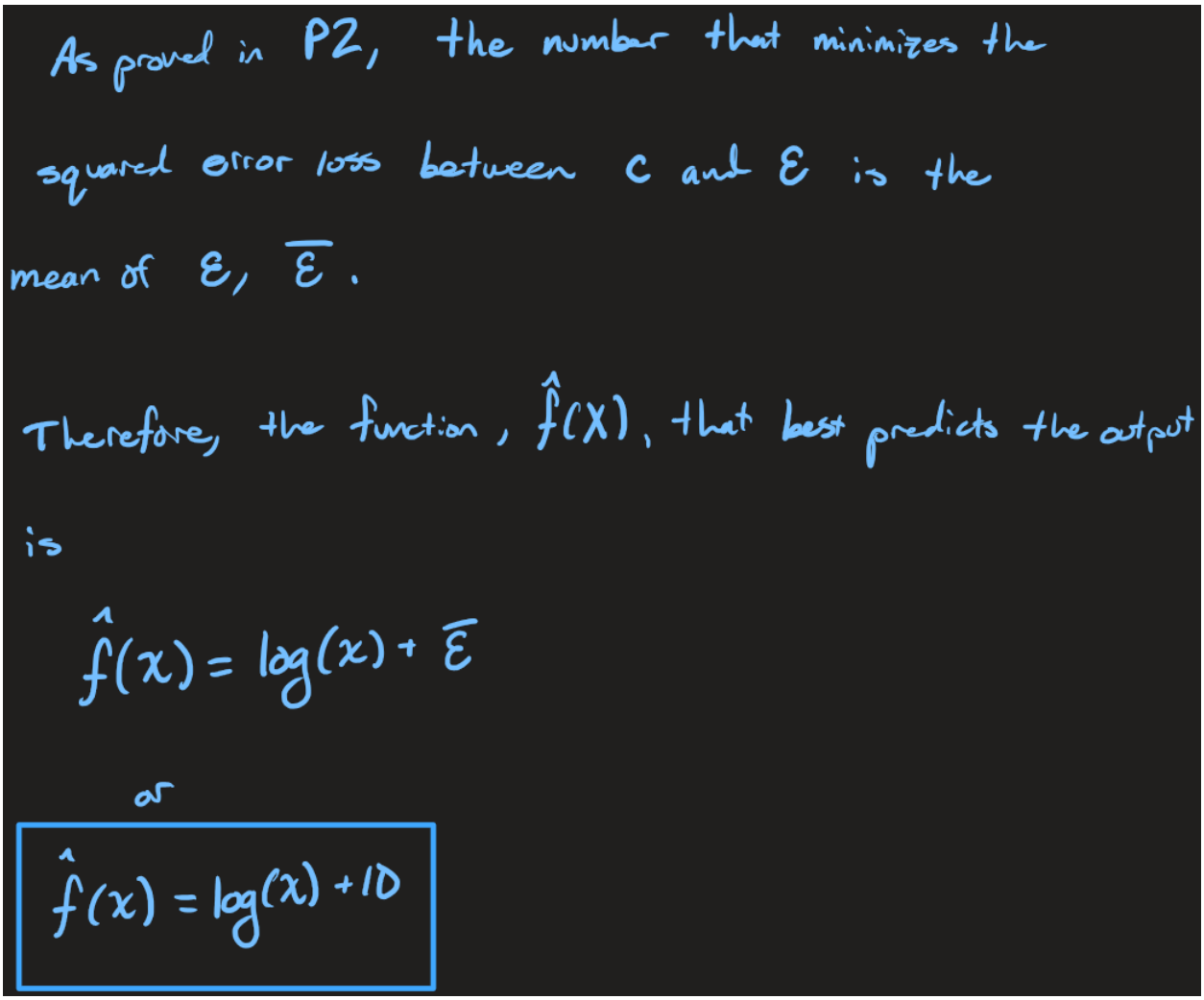
$$E\left[(Y - \hat{f}(x))^2\right]$$

$$E\left[\left(\log(x) + \varepsilon - \hat{f}(x)\right)^2\right] \qquad \hat{Y} = \log X + c$$

$$E\left[\left(\log(x) + \varepsilon - (\log(x) + c)\right)\right]$$

$$E\left[(\varepsilon - c)^2\right]$$

What value of c best describes random noise of ε?



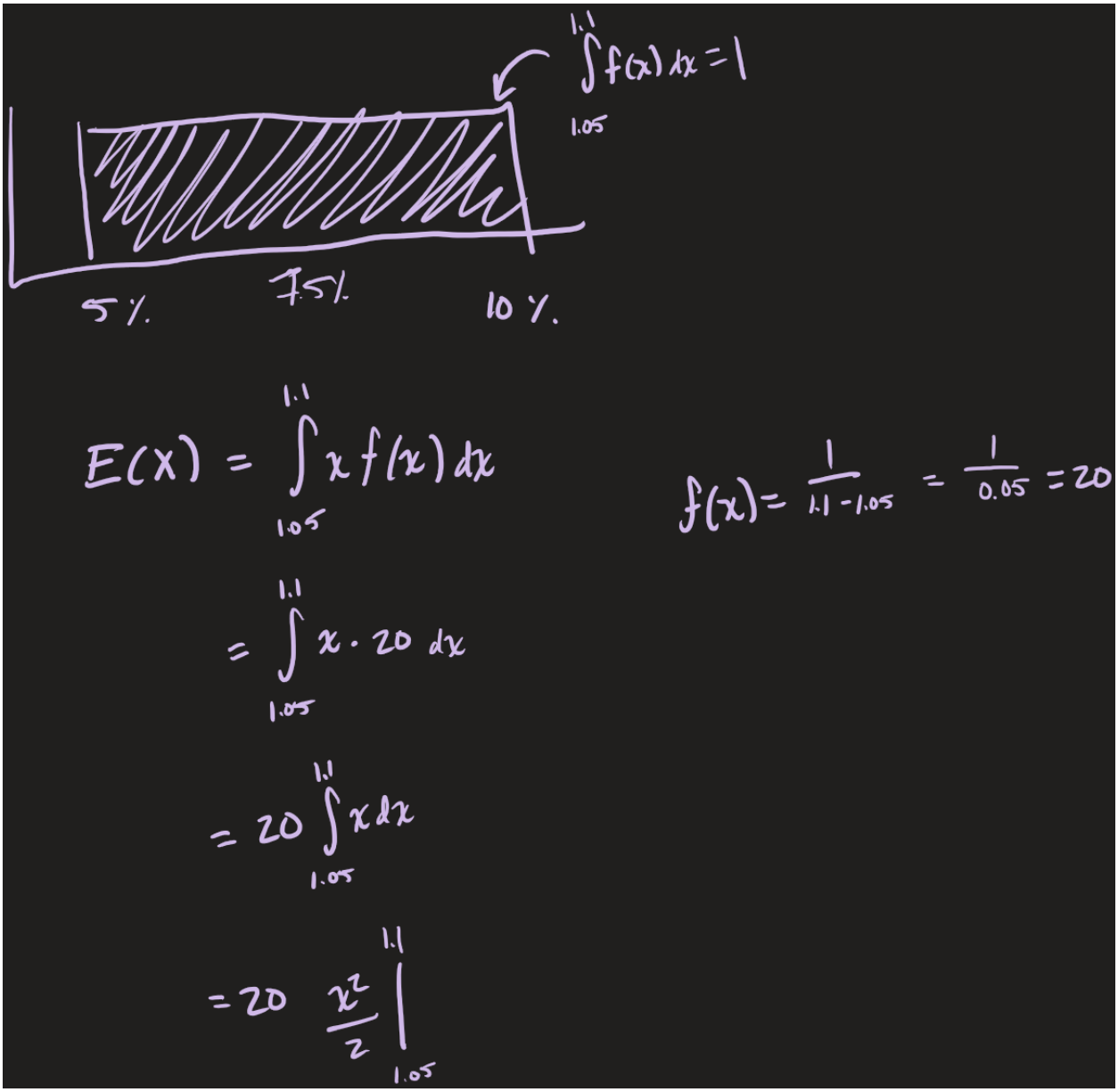
P4

With more features comes a higher dimensionality in the data. This presents issues in the K-Nearest Neighbors algorithm, because the algorithm is reliant on the Euclidean distance formula for calculating the nearest neighbors to a given point. Because the formula adds another term for each dimension of the vector, when distance is calculated for high dimensional inputs, the nearest neighbors to a given data point may not be close at all.

P5

Stratified sampling is not needed for sufficiently large enough datasets. This is because the Law of Large Numbers will allow for uniform sampling to represent features equally. However, this does **not** apply to small datasets due to the high variance that will appear in small datasets.

P6


$\int_{1.05}^{1.1} f(x)\,dx = 1$
5%
7.5%
10%
$E(X) = \int_{1.05}^{1.1} x f(x)\,dx$
$f(x) = \frac{1}{1.1 - 1.05} = \frac{1}{0.05} = 20$
$= \int_{1.05}^{1.1} x \cdot 20\,dx$
$= 20 \int_{1.05}^{1.1} x\,dx$
$= 20 \left. \frac{x^2}{2} \right|_{1.05}^{1.1}$

$$= 10\left[x^2 \Big|_{1.05}^{1.1}\right]$$

$$= 10[1.1^2 - 1.05^2]$$

$$= 10[0.1075]$$

$$= 1.075$$

Expected inflation is the average, or 7.5%.

Because there is an equal chance of staying 1 or 2 years, we can find the expected value by using the probabilities and expected costs like so:

$$\left(\tfrac{1}{2}\right)(1.075)(10{,}000) + \left(\tfrac{1}{2}\right)\left[(1.075)(10{,}000) + (1.075)^2(10{,}000)\right] = \text{Expected Total Cost}$$

$5{,}375 + 11{,}153.125 = \text{Expected Total Cost}$

$16,528.125 = Expected Total Cost

Part 7:

You are given the following training data (input-output (x, y) pairs), sorted according to the input: (1.2, 14.6), (2.3, 36.3), (3.1, 43.0), (4.2, 19.6), (8.3, 80.6), (8.4, 89.0), (9.4, 81.0), (9.9, 93.1), (11.2, 116.3), (13.0, 134.9). Now, what is the K-Nearest-Neighbor prediction for an input x0 = 6 for K = 2? How about for K = 4?

In [ ]:
```
import pandas as pd
from sklearn.neighbors import KNeighborsRegressor
```

In [ ]:
```
data = [[1.2, 14,6],
        [2.3, 36.3],
        [3.1, 43.0],
        [4.2, 19.6],
        [8.3, 80.6],
        [8.4, 89.0],
        [9.4, 81.0],
        [9.9, 93.1],
        [11.2, 116.3],
        [13.0, 134.9]]

data = pd.DataFrame(data).drop(columns=2)
data.columns = ["X", "Y"]
data
```

Out[ ]:

	X	Y
0	1.2	14.0
1	2.3	36.3
2	3.1	43.0
3	4.2	19.6
4	8.3	80.6
5	8.4	89.0
6	9.4	81.0
7	9.9	93.1
8	11.2	116.3
9	13.0	134.9

In [ ]:

```
X = data["X"].values.reshape(-1, 1)  # Reshape X to 2D array
Y = data["Y"].values  # No need to reshape Y

knn = KNeighborsRegressor(n_neighbors=2, weights="uniform", algorithm="brute")
knn.fit(X, Y)
k_2 = knn.predict([[6]])[0]

knn = KNeighborsRegressor(n_neighbors=4, weights="uniform", algorithm="brute")
knn.fit(X, Y)
k_4 = knn.predict([[6]])[0]

print("Input X_0 = 6")
print(f"K = 2: {k_2:.2f}, K = 4: {k_4:.2f}")
```

```
Input X_0 = 6
K = 2: 50.10, K = 4: 58.05
```

Part 8

You are given the following input-output (x, y) pairs, sorted according to the input: (8.4, 89.0), (9.4, 81.0), (9.9, 93.1), (11.2, 116.3). For the above data, please give an example of regression that has a coefficient of determination, i.e., R2, less than 0. Give the predictions from your regression and show your calculation steps and the R2 value. You may use Python to do the actual calculations.

The Coefficient of Determination is negative when the test set cannot be explained by the model that was trained on the training data at all. To achieve this, I will train on the input data given and test on data that does not contain this data.

In [ ]:
```
train_data = [[8.4, 89.0],
              [9.4, 81.0],
              [9.9, 93.1],
              [11.2, 116.3]]
train_data = pd.DataFrame(train_data)
# train_data = pd.DataFrame(train_data).drop(columns=2)
train_data.columns = ["X", "Y"]

train_X = train_data["X"].values.reshape(-1, 1)  # Reshape X to 2D array
train_Y = train_data["Y"].values  # No need to reshape Y

knn = KNeighborsRegressor(n_neighbors=2, weights="uniform", algorithm="brute")
knn.fit(train_X, train_Y)
```

Out[ ]:
```
KNeighborsRegressor
KNeighborsRegressor(algorithm='brute', n_neighbors=2)
```

In [ ]:
```
import numpy as np
# knn.score(X, Y)
test_X = np.concatenate([X[0:5], X[9:]], axis = None)
test_Y = np.concatenate([Y[0:5], Y[9:]], axis = None)
test_X = test_X.reshape(-1,1)

knn.score(test_X, test_Y)
```

Out[ ]:
```
-0.374370310168221
```

Part 9

Suppose you are given an array of values for a scalar- valued feature: [89.0, 81.0, 93.1, 116.3]. You will do feature scaling on it. What does the min-max scaling give you? How about standardization? Show you calculation steps and the results. You can use Python to do the actual calculation.

Min-max scaling:

In [ ]:
```
from sklearn.preprocessing import MinMaxScaler

features = np.array([89.0, 81.0, 93.1, 116.3])
features = features.reshape(-1,1)
```

```
scaler = MinMaxScaler()

scaler.fit(features)
scaler.transform(features)
```

Out[ ]:
```
array([[0.2266289],
       [0.       ],
       [0.3427762],
       [1.       ]])
```

Standardization:

In [ ]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

scaler.fit(features)
scaler.transform(features)
```

Out[ ]:
```
array([[-0.44566659],
       [-1.05512517],
       [-0.13331907],
       [ 1.63411083]])
```

Part 10

Consider a categorical variable representing the modes of transportation with four categories: CAR, PLANE, TRAIN, OTHER. Suppose in the one-hot encoding, the order of new categorical variables (aka dummy variables) is the same as listed. What will be the result of one-hot encoding of the following instances? instance 1: TRAIN instance 2: CAR instance 3: OTHER.

Instance	Car	Plane	Train	Other
TRAIN	0	0	1	0
Car	1	0	0	0
Other	0	0	0	1

Part 11

Discuss the pros and cons of 30-fold cross-validation versus 3-fold cross-validation with respect to: computation time, how prone it is to overfitting, and how confident you are about the validation score on each split.

30-fold cross-validation will split the data into 30 different data and test sets, with 29/30 of the data being training data and 1/30 being test data. It will then cycle 30 different times to train the model. On the other hand, 3-fold cross-validation will only do this 3 times, with a 2/3 : 1/3 split for training and test data.

This means that a 30-Fold cross-validation will take significantly more computing time as it needs to train the model 27 more times than the 3-Fold cross-validation.

With respect to overfitting, higher fold values will force the model to generalize more to other cases, as it prevents it from overfitting to the training set since there will be 30 cases with (N-1)/N of the training data inside the training set and 1/N of the data is used for evaluating model performance. This means that between a 3-Fold and 30-Fold cross-validation, the 30-Fold cross-validation should have better prevention against overfitting than the 3-Fold and 30-Fold should find a more generalized model.

With respect to the validation scores on each split, cross-validation allows you to get the score for each model, but it also allows you to get a mean and standard-deviation since there are N data points for an N-Fold cross-validation. This means we can be more confident in our model assessment metrics for higher fold cross-validations; therefore, 30-Fold cross-validation will give us more confidence in our validation score than the 3-Fold cross-validation.

Part 12

Consider binary classification. The table below shows 20 instances and their scores assigned by the classifier. We assume 1 corre- sponds to the positive class, 0 to the negative class. We take the convention that the classifier declares an instance positive if and only if the instance's score is greater than or equal to the threshold value.

Actual Class	Scores
0	0.05
0	0.09
0	0.12
0	0.18
0	0.23
0	0.25
0	0.31
1	0.35
0	0.39

Actual Class	Scores
0	0.41
0	0.45
0	0.47
1	0.48
1	0.75
1	0.83
1	0.88
1	0.92
0	0.95
1	0.97
1	0.99

A)

Confusion Matrix: Threshold = 0.5

	Predicted Negative	Predicted Positive
Actual Negative	11	1
Actual Positive	2	6

B)

True Positive: 6
True Negative: 11
False Positive: 1
False Negative: 2

C)

Precision : 6/(6+1) = **0.86**

Recall/Sensitivity/True Positive Rate: 6/(6+2) = **0.75**

False Negative Rate: 1- 0.75 = **0.25**

Specificity: TN / (FP + TN) = 11/12 = **0.92**

False Positive Rate: FP / (FP + TN) = **0.08**

F_1 Score = [2*(Precision * Recall)]/[Precision + Recall]
= *[2*(0.85 * 0.75)]/[0.85 + 0.75]
F_1 Score = **0.80**

D)

Recall : (TP) / (TP + FN)

To get 100% recall, there must be no false negatives. This can only be achieved with threshold = **0.35**.
At this threshold, the precision, defined as (TP) / (TP + FP), is equal to (7) / (7+5) = (7/12) = **0.58**.

E)

Example that precision is not an increasing function of threshold:

At threshold = 0.48, the precision is 7/8 = 0.875. However, at threshold = 0.95, the precision is 2/3 = 0.66. This shows that precision may not always increase as the threshold increases.