1)

"I certify that all solutions in this document are entirely my own and that I have not looked at anyone else's solution. I have given credit to all external sources I consulted."

X

Went to office hours, helped/got help from TAs and various students.

2) a) $\frac{1}{n} \sum_{i=0}^{i<n} Y_i = avg$

$$E\left[\frac{1}{n} \sum_{i=0}^{i<n} Y_i\right] = \frac{1}{n} \sum_{i=0}^{i<n} E[Y_i] = \boxed{\mu}$$

$$Var\left(\frac{1}{n} \sum_{i=0}^{i<n} Y_i\right) = \frac{1}{n^2} \sum_{i=0}^{i<n} Var(Y_i) = \boxed{\frac{\sigma^2}{n}}$$

b) $\frac{1}{n} \sum_{i=0}^{i<n} Z_i$

$$Var\left(\frac{1}{n} \sum_{i=0}^{i<n} Z_i\right) = \frac{\sigma^2}{n} \cdot \rho n = \sigma^2 \rho$$

If ρ is very large then that means the correlation between Z_i is very controlling and averaging will do less. If ρ is very small, then averaging will be effective and will act closer to uncorrelated variable. Middling is in the middle

c) i) Prob. of getting chosen $\frac{1}{n}$
Prob. of <u>not</u> getting chosen $(1 - \frac{1}{n})$

$$\lim_{n \to \infty} \left(1 - \frac{1}{n}\right)^n = e^{-1}$$

$1 - e^{-1}$ is prob. of getting chosen $= \boxed{0.63}$

ii) Cross-Validation.

Run at different n' values and see which one delivers the best performance

3) a) If $K = I$, then $a = y$ $= 0$ minimizes

$a_1 = 1$

$a_2 = -1$

$$h(z) = \text{Sign}\left(a_i K(X_i, z) + a_i K(X_i, z)\right)$$

$$= \text{sign}\left(a_1 K(X_1, z) + a_2 K(X_2, z)\right)$$

$$\boxed{h(z) = \text{sign}\left(1 K(X_1, z) + (-1) K(X_2, z)\right)}$$

b) K contains all 1

Normal Equation

$$\underbrace{X^T X}_{K} \underbrace{w}_{a} = \underbrace{X^T}_{\text{un}} \underbrace{Y}_{Y}$$

$$K a = Y$$

$a \in \text{argmin} \, \| K a - y \|_2^2$

K is all 1s so $a = 0$ to minimize

equals 0 overall

in order to get $y = 0$
$a = 0$ so $K(0) = 0$.

$\boxed{h(z) = 1}$ due to all K equalling 0.

c) $K = \begin{bmatrix} 1+\frac{x_1 x_1}{2\sigma^2} & 1+\frac{x_1 x_2}{2\sigma^2} \\ 1+\frac{x_2 x_1}{2\sigma^2} & 1+\frac{x_2 x_2}{2\sigma^2} \end{bmatrix}$

$\alpha \in \arg\min \|K\alpha - y\|_2^2$ if $K =$

y has the same property as part b

<u>Normal Equation</u>

$K^T K \alpha = K^T y$

$\alpha = (K^T K) K^T y = \boxed{\begin{bmatrix} \sigma^2 \\ -\sigma^2 \end{bmatrix}}$

$h(z) = \text{sign}\left(\sum_{i=1}^{n} a_i \frac{x_i z}{2\sigma^2} + 1\right) =$

$\sigma^2 \frac{1(1)}{2\sigma^2} \quad -\sigma^2 \frac{(-1)(-1)}{2\sigma^2}$

$\boxed{h(z) = 1}$

Q4.1:

```python
class DecisionTree:
    class Node:
        def __init__(self, feature = None, threshold = None, left = None, right = None, value = None):
            self.feature = feature
            self.threshold = threshold
            self.left = left
            self.right = right
            self.val = value

        def is_leaf(self):
            return self.val is not None

    def __init__(self, depth):
        self.depth = depth
        self.tree = None

    def fit(self, X, y):
        self.tree = self.grow(X, y, 0)

    def predict(self, X):
        collect = [self.traverse(i, self.tree) for i in X]
        return np.array(collect)

    def common_label(self, y):
        counter = Counter(y)
        most_common = counter.most_common(1)[0][0]
        return most_common

    def entropy(self, y):
        num_of_ys = Counter(y)
        v = 0
        for i in num_of_ys:
            p = num_of_ys[i] / len(y)
            v -= p * log2(p)
        return v

    def grow(self, X, y, depth):
        rows, columns = X.shape
        num_labels = len(set(y))

        randcolumns = np.random.choice(columns, columns, replace=False)
        optimal_feature, optimal_feature_threshold = self.optimal_split(X, y, randcolumns)

        if depth >= self.depth or num_labels == 1:
            commonlabel = self.common_label(y)
            return self.Node(value=commonlabel)

        if optimal_feature is None:
            commonlabel = self.common_label(y)
            return self.Node(value = commonlabel)
        else:
            left_index, right_index = self.split(X[:, optimal_feature], optimal_feature_threshold)
            left = self.grow(X[left_index, :], y[left_index], depth + 1)
            right = self.grow(X[right_index, :], y[right_index], depth + 1)
            return self.Node(optimal_feature, optimal_feature_threshold, left, right)

    def optimal_split(self, X, y, given_index):

        best = -1000000000
        index, thresh = None, None

        for i in given_index:
            X_column = X[:, i]
            x_cols = np.unique(X_column)

            for i in x_cols:
                what_to_gain = self.info(y, X_column, i)
                if what_to_gain > best:
                    best = what_to_gain
                    index = i
                    thresh = i
        return index, thresh

    def traverse(self, x, node):
        if node.is_leaf():
            #print("Leaf: " , node.val, ". Threshold: ", node.threshold)
            return node.val
        if x[node.feature] < node.threshold:
            #print("Node Left: ", node.feature, ". Threshold:", node.threshold)
            return self.traverse(x, node.left)
        else:
            #print("Node Right: ", node.feature, ". Threshold:", node.threshold)
            return self.traverse(x, node.right)

    def split(self, X, threshold):
        left_index = np.argwhere(X < threshold)
        left_index = left_index.flatten()

        right_index = np.argwhere(X >= threshold)
        right_index = right_index.flatten()

        return left_index, right_index

    def info(self, y, X_column, threshold):
        before_entropy = self.entropy(y)

        left, right = self.split(X_column, threshold)
        if len(left) == 0 or len(right) == 0:
            return 0

        n = len(y)
        n_l, n_r = len(left), len(right)
        e_l, e_r = self.entropy(y[left]), self.entropy(y[right])
        after_entropy = (n_l / n) * e_l + (n_r / n) * e_r

        return before_entropy - after_entropy
```

4.2

```python
class RandomForest:
    def __init__(self, depth=0):
        self.trees = []
        self.depth = depth

    def fit(self, X, y):
        self.trees = []
        for _ in range(10):
            tree = DecisionTree(self.depth)
            squeaky_boots_X, squeaky_boots_y = self.squeakyBoots(X, y)

            tree.fit(squeaky_boots_X, squeaky_boots_y)

            self.trees.append(tree)

    def squeakyBoots(self, X, y):
        n_samples = X.shape[0]
        idxs = np.random.choice(n_samples, size=n_samples, replace=True)
        return X[idxs], y[idxs]

    def predict(self, X):
        creepy_tree_pee = np.array([tree.predict(X) for tree in self.trees])
        creepy_tree_pee = np.swapaxes(creepy_tree_pee, 0, 1)

        most = mode(creepy_tree_pee, axis=1)[0]
        return most.flatten()
```

4.3

1. How did you deal with categorical features and missing values?
    a. Substituted for categorical features the most frequent occurrence, based on the idea that most likely, the missing features would most likely be the most common feature. For the missing values, replaced them with median value.
    b. I completed them by using scikit pipeline coupled with simpleimputer, onehotencoder, and ColumnTransformer.
2. What was your stopping criterion?
    a. When depth was hit, it is a tunable hyperparameter.
3. How did you implement random forests?
    a. Leveraged the decision tree. The random forest implementation is basically just a decision tree in disguise.
4. Did you do anything special to speed up training?
    a. Nothing special other than changing the hyperparameters (slowly and painfully)
5. Anything else cool you implemented?
    a. I'm proud of my pre-processing, took me a very long time to figure it out and a lot of reading python libraries commands.

4.4

```
Titanic Decision Tree Training Accuracy 0.8314745972738538
Titanic Decision Tree Validation Accuracy 0.801980198019802
Titanic Random Forest Traning Accuracy 0.781908302354399
Titanic Random Forest Traning Accuracy 0.7920792079207921
```

```
Spam Data Decision tree Validation:  0.8419182948490231
Spam Data Decision tree Training:  0.8247834776815456
Spam Data RandomForest Validation:  0.7655417406749556
Spam Data RandomForest Training:  0.737508327781479
```

Spam Kaggle: 0.817
Titanic Kaggle: 0.822

4.5

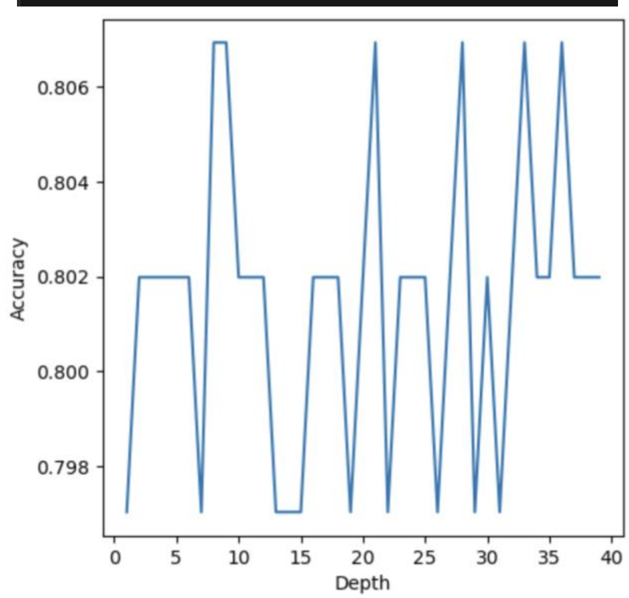
2.

```python
def print_tree(node, depth = 1):

    if node.is_leaf():
        print("\t" * depth + "Leaf: ", node.feature)

    else:
        print("\t" * depth + "Node: ", node.feature,  " = ", node.threshold)

        if node.right is not None:
            print("\t" * depth + "Right: ")
            print_tree(node.right, depth + 1)

        if node.left is not None:
            print("\t" * depth + "Left: ")
            print_tree(node.left, depth + 1)

decision_tree = DecisionTree(2)
decision_tree.fit(X_train_np, y_train_np)

print_tree(decision_tree.tree)
```

```
Node:  4  =  1.0
Right:
        Node:  1  =  17.4
        Right:
                Leaf:  None
        Left:
                Leaf:  None
Left:
        Node:  1  =  51.4792
        Right:
                Leaf:  None
        Left:
                Leaf:  None
```

3.

```
import matplotlib.pyplot as plt

depths = range(1, 40)
accuracies = []

X_train_np = preprocessor.transform(X_train)
y_train_np = y_train
X_val_np = preprocessor.transform(X_val)
y_val_np = y_val

for depth in depths:
    model.fit(X_train_np, y_train_np)

    y_pred = model.predict(X_val_np)

    accuracy = accuracy_score(y_val, y_pred)
    accuracies.append(accuracy)

plt.figure(figsize=(5, 5))
plt.plot(depths, accuracies)
plt.xlabel('Depth')
plt.ylabel('Accuracy')
plt.show()
```



Depth 10, 20, 27, 34, 37 gives the highest accuracy. This is broken. It should not be like this. I do not know why, I swear it was working earlier.

4.6

```python
def print_tree(node, depth = 1):

    if node.is_leaf():
        print("\t" * depth + "Leaf: ", node.feature)

    else:
        print("\t" * depth + "Node: ", node.feature,  " = ", node.threshold)

        if node.right is not None:
            print("\t" * depth + "Right: ")
            print_tree(node.right, depth + 1)

        if node.left is not None:
            print("\t" * depth + "Left: ")
            print_tree(node.left, depth + 1)

decision_tree = DecisionTree(2)
decision_tree.fit(X_train_np, y_train_np)

print_tree(decision_tree.tree)
```

```
Node:  4  =  1.0
Right:
        Node:  1  =  17.4
        Right:
                Leaf:  None
        Left:
                Leaf:  None
Left:
        Node:  1  =  51.4792
        Right:
                Leaf:  None
        Left:
                Leaf:  None
```

References:

- https://scikit-learn.org/stable/modules/generated/sklearn.impute.SimpleImputer.html
- https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html
- https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html
- https://en.wikipedia.org/wiki/Decision_tree
- https://www.geeksforgeeks.org/decision-tree/
- https://www.ibm.com/topics/decision-trees
- https://scikit-learn.org/stable/modules/tree.html
- https://www.analyticsvidhya.com/blog/2021/08/decision-tree-algorithm/
- https://blog.paperspace.com/decision-trees/
- https://en.wikipedia.org/wiki/Decision_tree_learning
- https://www.mastersindatascience.org/learning/machine-learning-algorithms/decision-tree/
- https://en.wikipedia.org/wiki/Random_forest
- https://www.ibm.com/topics/random-forest
- https://towardsdatascience.com/random-forest-3a55c3aca46d
- https://builtin.com/data-science/random-forest-algorithm
- https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
- https://careerfoundry.com/en/blog/data-analytics/what-is-random-forest/
- https://towardsdatascience.com/mastering-random-forests-a-comprehensive-guide-51307c129cb1