In [1]:
```
import matplotlib.pyplot as plt
import numpy as np
from sklearn import tree, ensemble
from sklearn.model_selection import cross_val_score
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import KFold
from collections import defaultdict
import pprint
```

In [2]:
```
# Assignment Constants
RANDOM_STATE = 10
FIGSIZE = (12,8)
#### Use the following line before plt.plot(....) to increase the plot size ####
# plt.figure(figsize=FIGSIZE)
```

Question 1

Use the breast cancer data set from Homework 0 to create a training set. Recall that the label is 0 if the patient's data indicates a malignant cancer and 1 otherwise. Compute the base rate of malignant cancer occurrence over the entire data set. In other words, what would be your best guess for the probability of malignant cancer of a single example using only the labels in the training set? This question is very simple, so try not to overthink it.

In [3]:
```
cancer = load_breast_cancer()
p_malignant = np.mean(cancer.target)
base_rate = min(p_malignant, 1 - p_malignant)
print(f"Base Rate: {base_rate:.4f}")
```

```
Base Rate: 0.3726
```

2

The goal is to build a decision tree that, based on the other features in the set, predicts whether or not a patient has malignant cancer. So this is a classification problem. Using `tree.DecisionTreeClassifier` and other functions in the scikit-learn library, one can build a decision tree and calculate both its training accuracy when fitted to the entire data set as well as its accuracy using 10-fold cross validation (which gives a better idea of true accuracy). In this question you will need to complete two sub-components:

(a)

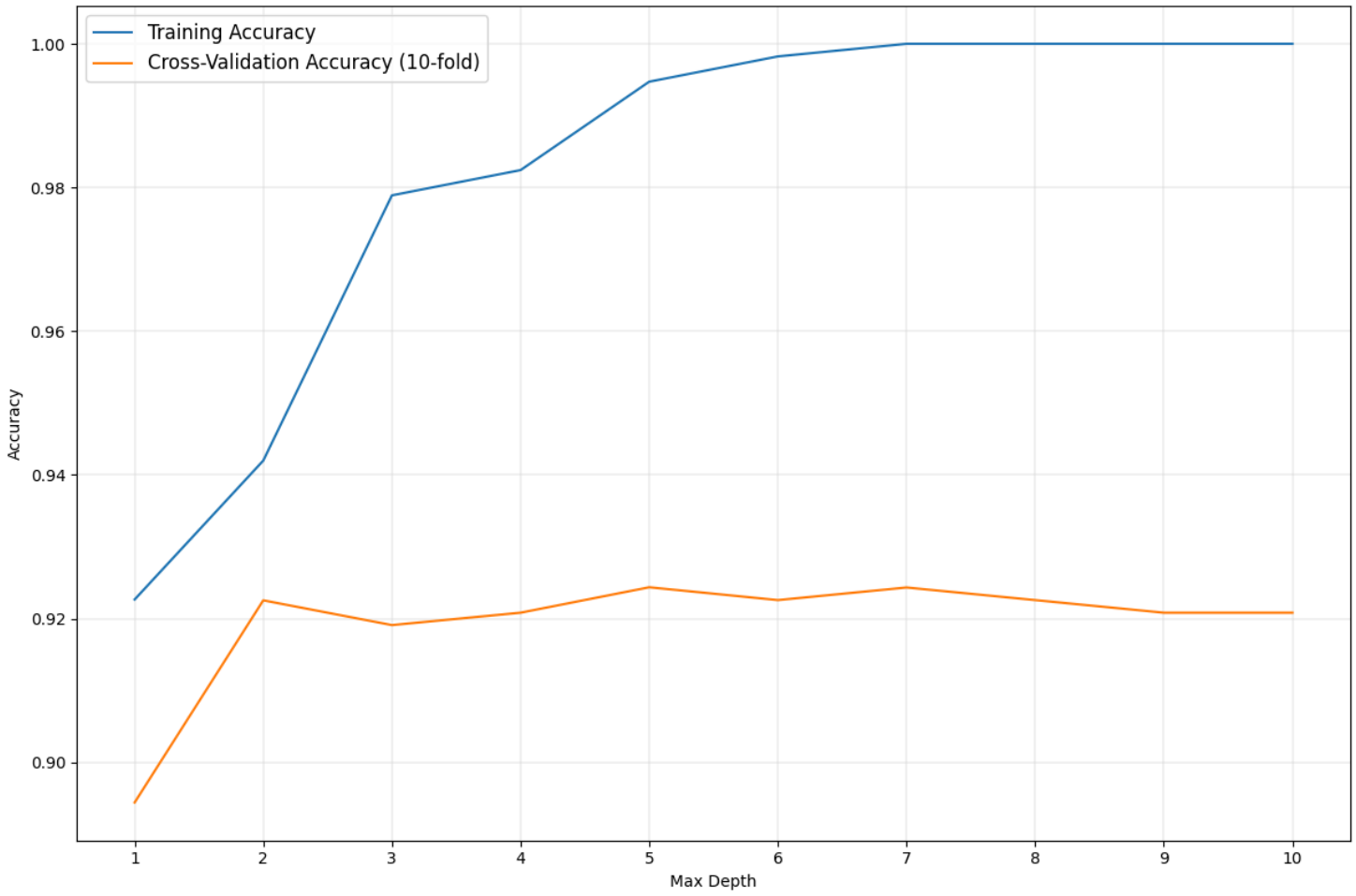
(a) Make a plot visualizing the performance of a `tree.DecisionTreeClassifier` as you search for an optimal `max_depth` parameter. Vary the depth of your decision tree using max depth = 1,2,. . . ,10 and record the results from the following evaluation procedures for each setting:

- The accuracy when training and testing on the full dataset.
- 10-fold cross-validated accuracy.

Plot the results of both evaluation procedures on the same plot with evaluation scores on the y-axis and max depth values on the x-axis. Use 10 as your random seed/state for the decision tree and the cross-validation. Use a legend to label both evaluation procedures.

In [4]:
```
# Part 2:

x, y = cancer.data, cancer.target # Features and Labels
max_depths = range(1, 11) # Max depths to try for Decision Trees

# accuracy storage
training_accuracy = []
cross_validation_accuracy = []

# K-Fold Cross Validation
kf = KFold(n_splits=10, shuffle=True, random_state=RANDOM_STATE)

for depth in max_depths:
    # Decision Tree Classifier
    decision_tree_clf = tree.DecisionTreeClassifier(max_depth=depth, random_state=RANDOM_STATE)

    # Train on full dataset and record training accuracy
    decision_tree_clf.fit(x, y)
    training_accuracy.append(decision_tree_clf.score(x, y)) # Training accuracy
```

```
    # perform K-Fold Cross Validation (10 folds)
    cv_scores = cross_val_score(decision_tree_clf, x, y, cv=kf)
    cross_validation_accuracy.append(np.mean(cv_scores))

# Plot Results
plt.figure(figsize=FIGSIZE)
plt.plot(max_depths, training_accuracy, label="Training Accuracy")
plt.plot(max_depths, cross_validation_accuracy, label="Cross-Validation Accuracy (10-fold)")
plt.xlabel("Max Depth")
plt.xticks(max_depths)
plt.ylabel("Accuracy")
plt.legend(fontsize='large')
plt.grid(True, linewidth=0.3, alpha=0.7)
plt.tight_layout()
plt.show()
```



In [5]:

```
# Part 2b and 2c: Best accuracies and depths

print("\nDepth\tTraining Acc.\tCV Acc.")
for depth, train_acc, cv_acc in zip(max_depths, training_accuracy, cross_validation_accuracy):
    print(f"{depth}\t{train_acc:.4f}\t\t{cv_acc:.4f}")

best_full_accuracy = max(training_accuracy)
best_cv_accuracy = max(cross_validation_accuracy)
best_full_depth = max_depths[training_accuracy.index(best_full_accuracy)]
best_cv_depth = max_depths[cross_validation_accuracy.index(best_cv_accuracy)]

print(f"\nBest Full Set Training Accuracy: {best_full_accuracy:.4f} at depth {best_full_depth}"
      f"\nBest Cross-Validation Accuracy: {best_cv_accuracy:.4f} at depth {best_cv_depth}"
      f"\nBase Rate: {base_rate:.4f}")

best_full_set_depths = [depth for depth, acc in zip(max_depths, training_accuracy) if acc == best_full_accuracy]
best_cv_set_depths = [depth for depth, acc in zip(max_depths, cross_validation_accuracy) if acc == best_cv_accurac
print(f"Depths with Best Full Set Training Accuracy: {best_full_set_depths}")
print(f"Depths with Best Cross-Validation Accuracy: {best_cv_set_depths}")
```

```
Depth   Training Acc.   CV Acc.
1       0.9227          0.8944
2       0.9420          0.9226
3       0.9789          0.9191
4       0.9824          0.9208
5       0.9947          0.9244
6       0.9982          0.9226
7       1.0000          0.9243
8       1.0000          0.9226
9       1.0000          0.9208
10      1.0000          0.9208

Best Full Set Training Accuracy: 1.0000 at depth 7
Best Cross-Validation Accuracy: 0.9244 at depth 5
Base Rate: 0.3726
Depths with Best Full Set Training Accuracy: [7, 8, 9, 10]
Depths with Best Cross-Validation Accuracy: [5]
```

(b)

Answer the questions below based on the results of 2a. Write your answers in the corresponding field in the markdown cell that is present in the HW1 template notebook. Do this by double clicking the markdown cell and writing your answer directly in the cell. Pressing enter will re-render the markdown.

(i.)

What setting of `max_depth` gave the best accuracy w.r.t. the **full-dataset** accuracy? If more than one setting equaled the best accuracy, list each of the best settings.

Student answer here: [7,8,9,10]

(ii.)

What setting of `max_depth` gave the best accuracy w.r.t. the **cross- validated** accuracy? If more than one setting equaled the best accuracy, list each of the best settings.

Student answer here: [5]

3.

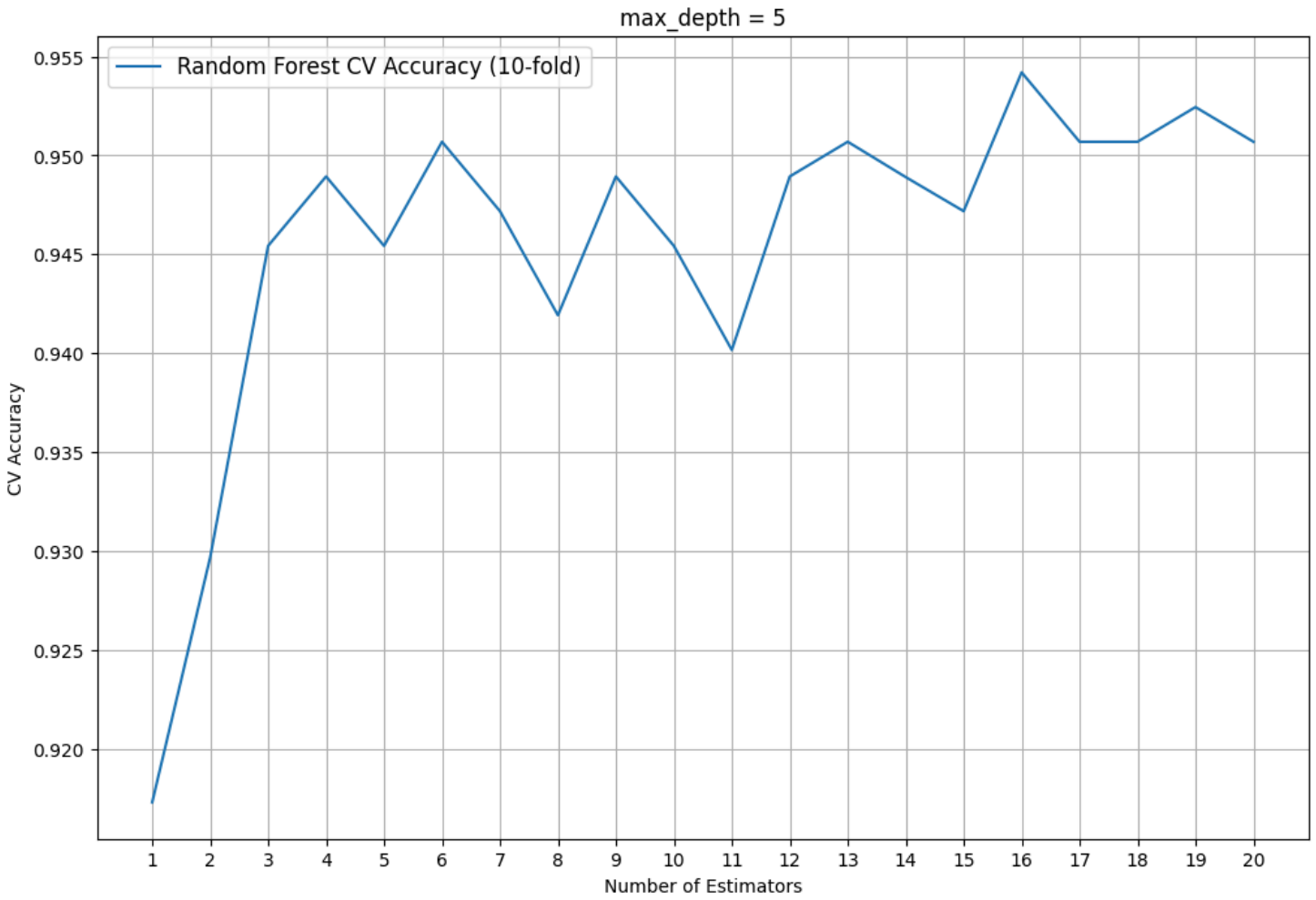
This question explores random forest classifiers by using scikit-learn's `ensemble.RandomForestClassifier`. You will make two plots and answer questions about them.

(a)

For the first plot, use a `ensemble.RandomForestClassifier` and the best depth you found 2(b)ii as `max_depth`. We will now find the optimal setting of a second parameter, n estimators. Vary the number of trees in the forest via the parameter `n_estimators` and plot its 10-fold cross-validated accuracy (use `n_estimators` = 1, 2, . . . , 20). Again, use 10 as your random seed for your classifier and cross-validation.

In [6]:
```
# Part 3:

# we will use the variable best_cv_depth from above as max_depth
n_estimators_range = range(1, 21)
cv_accuracy_rf = []

for n_estimators in n_estimators_range:
    rf_clf = ensemble.RandomForestClassifier(n_estimators=n_estimators, max_depth=best_cv_depth, random_state=RAND

    # Calculate 10-fold cross-validated accuracy
    cv_scores_rf = cross_val_score(rf_clf, x, y, cv=kf)
    cv_accuracy_rf.append(np.mean(cv_scores_rf))

# Plot Random Forest Results
plt.figure(figsize=FIGSIZE)
plt.plot(n_estimators_range, cv_accuracy_rf, label="Random Forest CV Accuracy (10-fold)")
plt.xlabel("Number of Estimators")
plt.xticks(n_estimators_range)
plt.ylabel("CV Accuracy")
plt.title("max_depth = {}".format(best_cv_depth))
```

```
plt.legend(fontsize='large')
plt.grid()
plt.show()
```



In [7]:
```
best_rf_accuracy = max(cv_accuracy_rf)
best_rf_n_estimator = n_estimators_range[cv_accuracy_rf.index(best_rf_accuracy)]
best_rf_full_set_n_estimators = [n_estimators for n_estimators, acc in zip(n_estimators_range, cv_accuracy_rf) if
print(f"\nBest Random Forest CV Accuracy: {best_rf_accuracy:.4f} at n_estimators {best_rf_n_estimator}")
print(f"n_estimators with Best Random Forest CV Accuracy: {best_rf_full_set_n_estimators}")

print("\nN_Estimators\tCV Acc.")
for n_estimators, cv_acc in zip(n_estimators_range, cv_accuracy_rf):
    print(f"{n_estimators}\t\t{cv_acc:.4f}")
```

```
Best Random Forest CV Accuracy: 0.9542 at n_estimators 16
n_estimators with Best Random Forest CV Accuracy: [16]

N_Estimators    CV Acc.
1               0.9173
2               0.9297
3               0.9454
4               0.9489
5               0.9454
6               0.9507
7               0.9472
8               0.9419
9               0.9489
10              0.9454
11              0.9402
12              0.9489
13              0.9507
14              0.9489
15              0.9472
16              0.9542
17              0.9507
18              0.9507
19              0.9524
20              0.9507
```

(b)

Do you see an improvement using random forests versus using a single tree? (Note: use the `n_estimators` =1 result as the result for a single tree.)

Student answer here: Yes, there is an improvement in accruacy of ~0.03 when comparing random forest to single tree.

(c)

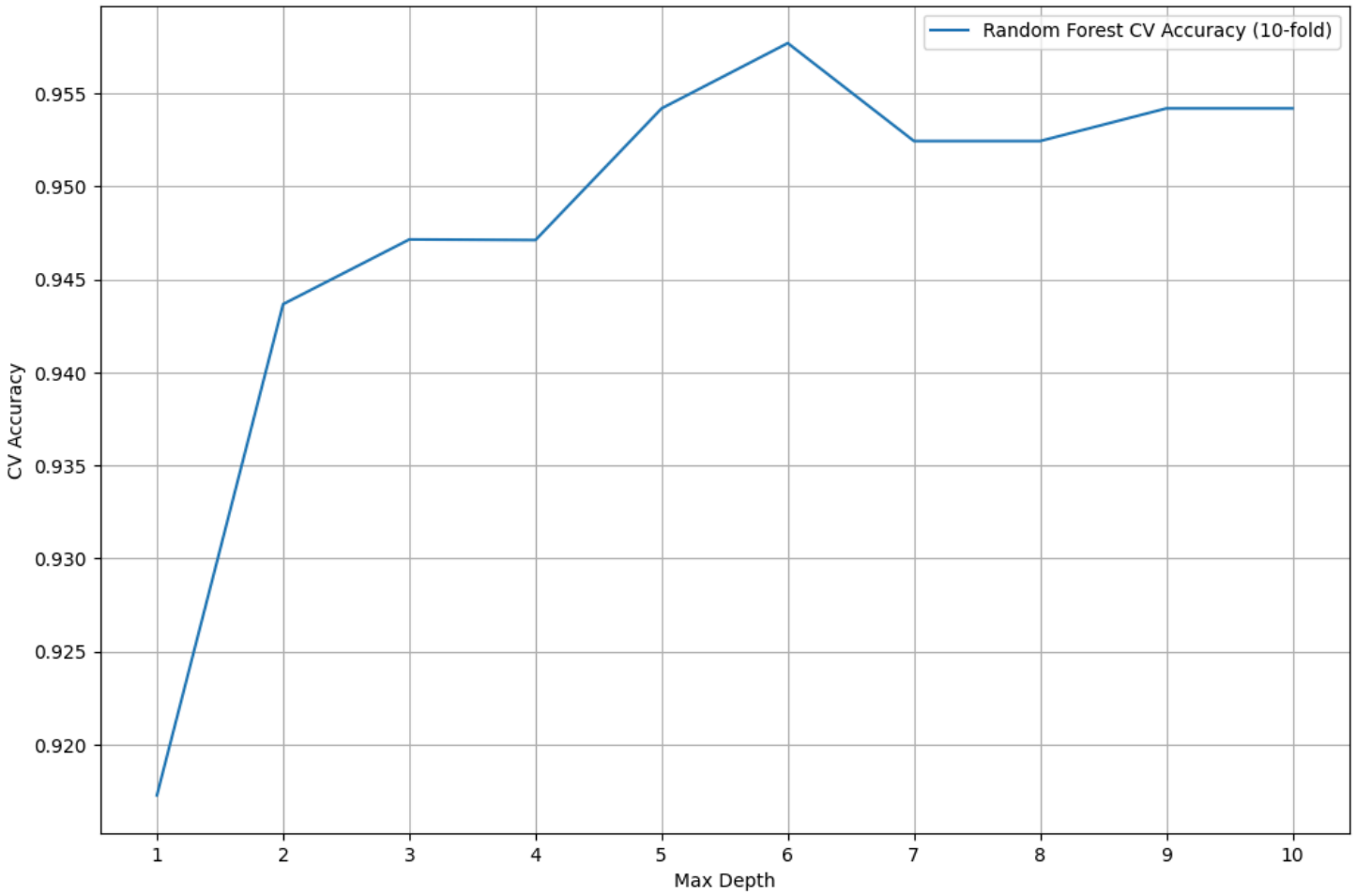
What setting of `n_estimators` gave the best accuracy w.r.t. the cross-validated ac- curacy?

Student answer here: 16

(d)

For the second plot, again use a `ensemble.RandomForestClassifier`, but this time you will fix the `n_estimators` parameter and again attempt to find the optimal setting of a `max_depth`. Use your answer to 3c as the setting for `n_estimators` and follow the procedure from 2a to find the best setting for max depth. This time, only plot the results from cross-validation and not the full set, but the plot should be the same structure as in 2a otherwise (use `max_depth` = 1,2,. . . ,10). Again, use 10 as your random seed.

In [8]:
```
# Part 3d:

# we will use the best_rf_n_estimator variable from above as n_estimators
cv_accuracy_rf = []
for depth in max_depths:
    rf_clf = ensemble.RandomForestClassifier(n_estimators=best_rf_n_estimator, max_depth=depth, random_state=RANDO

    # Calculate 10-fold cross-validated accuracy
    cv_scores_rf = cross_val_score(rf_clf, x, y, cv=kf)
    cv_accuracy_rf.append(np.mean(cv_scores_rf))

# Plot Results
plt.figure(figsize=FIGSIZE)
plt.plot(max_depths, cv_accuracy_rf, label="Random Forest CV Accuracy (10-fold)")
plt.xlabel("Max Depth")
plt.xticks(max_depths)
plt.ylabel("CV Accuracy")
plt.legend()
plt.grid()
plt.show()
```



In [9]:
```
best_rf_accuracy = max(cv_accuracy_rf)
best_rf_n_estimator = n_estimators_range[cv_accuracy_rf.index(best_rf_accuracy)]
best_rf_full_set_n_estimators = [n_estimators for n_estimators, acc in zip(n_estimators_range, cv_accuracy_rf) if
print(f"\nBest Random Forest CV Accuracy: {best_rf_accuracy:.4f} at n_estimators {best_rf_n_estimator}")
print(f"n_estimators with Best Random Forest CV Accuracy: {best_rf_full_set_n_estimators}")

print("\nN_Estimators\tCV Acc.")
for n_estimators, cv_acc in zip(n_estimators_range, cv_accuracy_rf):
    print(f"{n_estimators}\t\t{cv_acc:.4f}")
```

```
Best Random Forest CV Accuracy: 0.9577 at n_estimators 6
n_estimators with Best Random Forest CV Accuracy: [6]

N_Estimators    CV Acc.
1               0.9173
2               0.9437
3               0.9471
4               0.9471
5               0.9542
6               0.9577
7               0.9524
8               0.9524
9               0.9542
10              0.9542
```

(e)

In the plot in 3d, is the optimal setting of `max_depth` the same as in 2(b)ii? If not, what is the new optimal setting of `max_depth`?

Student answer here: No, The best setting now is [6].

4.

For this last question, we will explore the dependability of our estimates.

(a)

Make a plot using the following procedure:

i.

Using random state values from 0, 1, · · · , 99 calculate the 10-fold cross-validation accuracy of different `tree.DecisionTreeClassifiers` with max depth settings from 1, 2, · · · , 10. As before, you should use the same random state value for your classifier and cross-validation.

ii.

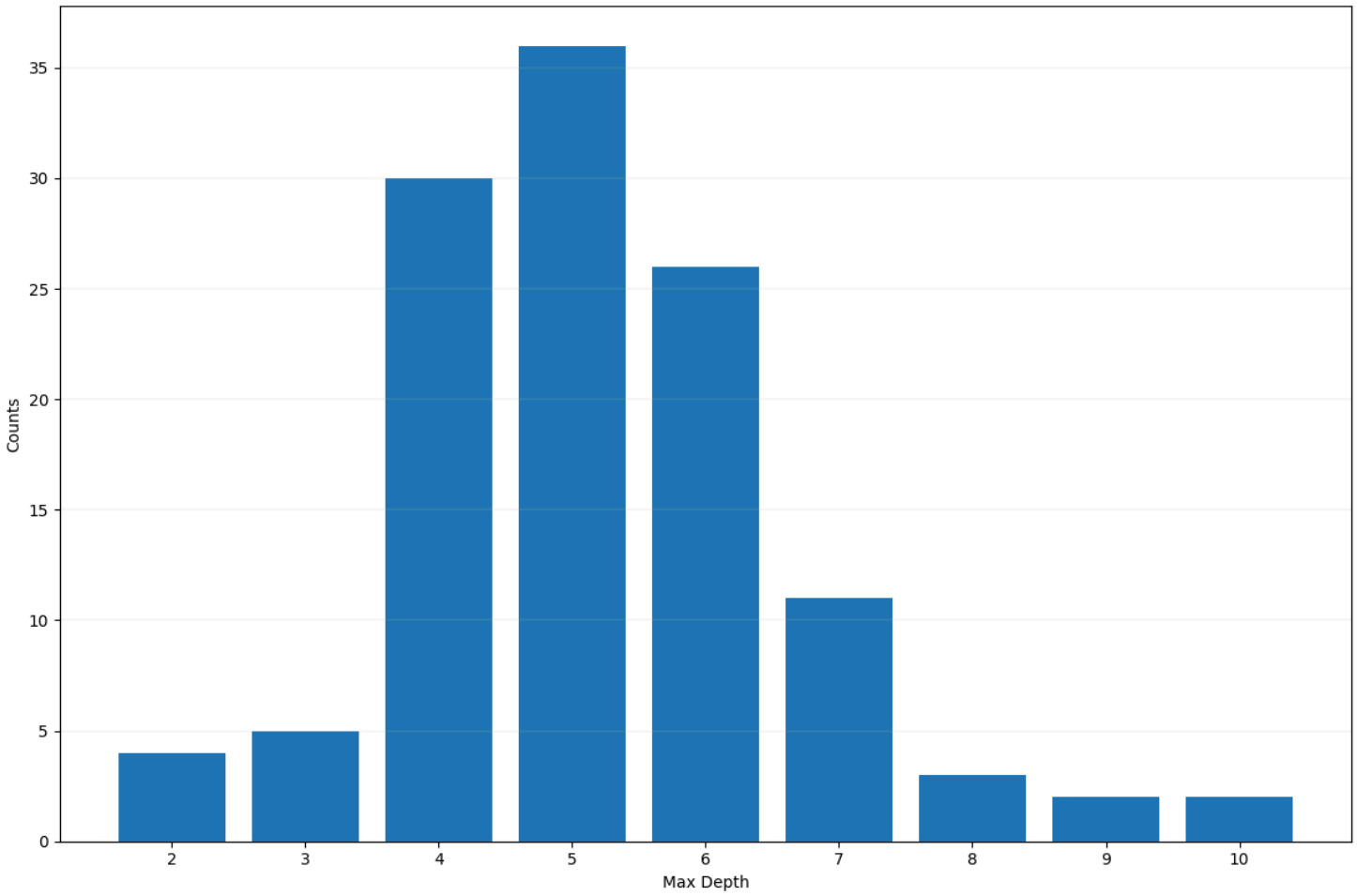
Then record the best max depth settings for each random state. Be sure to check whether multiple settings achieve the best accuracy.

Plot the counts for the best max depth settings as a bar chart with the max depth settings on the x-axis and the 'best parameter counts' on the y-axis (number of times that parameter was selected as the best max depth setting).

Note: this calculation might take some time. For debugging, try a smaller range of random states.

In [10]:
```
# Part 4:

random_states = range(100)
# we will use max_depths variable from above
best_max_depth_results = defaultdict(list)

for i, rs in enumerate(random_states):
    # kfold with current random state
    kf_rs = KFold(n_splits=10, shuffle=True, random_state=rs)

    # store cross validation means for each depth
    cv_means = []
    for depth in max_depths:
        decision_tree_clf = tree.DecisionTreeClassifier(max_depth=depth, random_state=rs)
        cv_scores = cross_val_score(decision_tree_clf, x, y, cv=kf_rs)
        cv_means.append(np.mean(cv_scores))

    # find the best accuracy and corresponding depths for this random state
    best_acc = max(cv_means)
    best_depth = [depth for depth, acc in zip(max_depths, cv_means) if acc == best_acc]

    # record each winning depth for this random state
    for depth in best_depth:
        best_max_depth_results[depth].append(best_acc)
    print(f"{i+1}/{len(random_states)} states completed\r", end="")

# convert
depths = list(best_max_depth_results.keys())
counts = [len(best_max_depth_results[d]) for d in depths]

plt.figure(figsize=FIGSIZE)
plt.bar(depths, counts)
plt.xlabel('Max Depth')
plt.ylabel('Counts')
plt.xticks(depths)
plt.grid(axis='y', linewidth=0.3, alpha=0.6)
plt.tight_layout()
plt.show()
```

```
100/100 states completed
```



In [11]:
```python
print("\nDepth\tCount\tAvg Acc.")
for depth in sorted(best_max_depth_results.keys()):
    count = len(best_max_depth_results[depth])
    avg_acc = np.mean(best_max_depth_results[depth])
    print(f"{depth}\t{count}\t{avg_acc:.4f}")

top_depths = sorted(best_max_depth_results.keys(), key=lambda d: len(best_max_depth_results[d]), reverse=True)[:2]
print("\nTop 2 Depths by Count:")
for depth in top_depths:
    count = len(best_max_depth_results[depth])
    avg_acc = np.mean(best_max_depth_results[depth])
    print(f"Depth: {depth}, Count: {count}, Avg Acc: {avg_acc:.4f}")
```

```
Depth   Count   Avg Acc.
2       4       0.9293
3       5       0.9308
4       30      0.9333
5       36      0.9357
6       26      0.9363
7       11      0.9359
8       3       0.9362
9       2       0.9350
10      2       0.9350

Top 2 Depths by Count:
Depth: 5, Count: 36, Avg Acc: 0.9357
Depth: 4, Count: 30, Avg Acc: 0.9333
```

(b)

What are the top two most frequent parameter settings?

Student answer here:

Depth: 5, Count: 36, Avg Acc: 0.9357

Depth: 4, Count: 30, Avg Acc: 0.9333