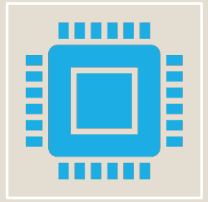
CAPSTONE PROJECT

Name: ALLEN MWANDUNGA

Matriculation No: 7219535

Streamlit NoteBook



StrimlitNoteBook is an API that allows you to convert Jupyter Notebooks to identical Streamlit components which you can embed right into your scripts where you build your Streamlit apps.

Streamlit apps were, simply, apps - a web page to showcase dashboards, try out models, etc. But those apps lacked a way to share how their creators built them and what their approaches were.

Even if the creators wanted to do that, they had to turn to crude methods like linking to external links or docs where they outlined their approach.

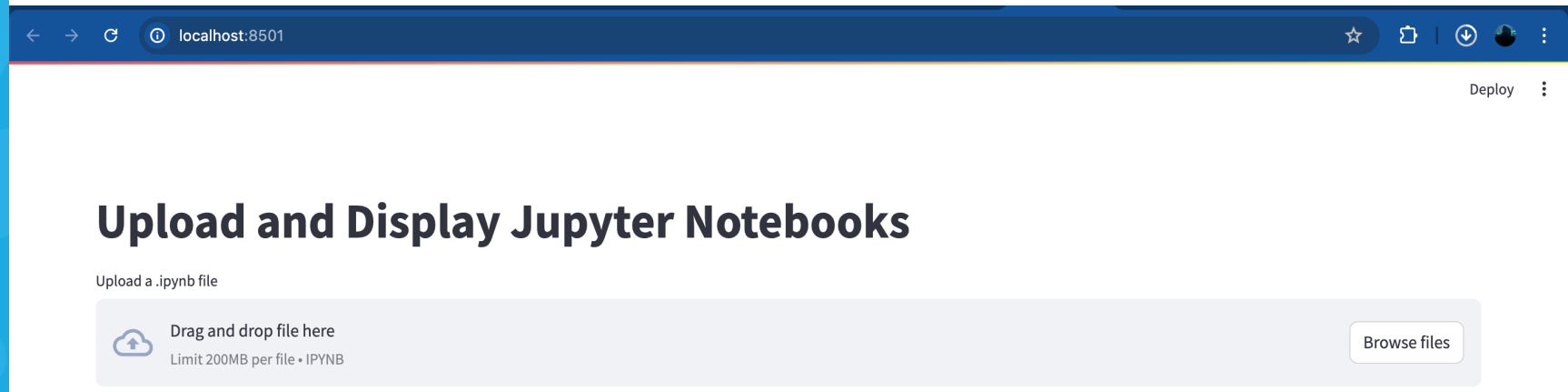
Strimlit NoteBook idea was that developers should be able to embed notebooks into the web app scripts. Furthermore, The process should be simple and require as little code as possible.

Running a web app

```
The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
(.venv) (base) Ndungas-MBP-2:strimlitbook ndungajr$ streamlit run main.py

  You can now view your Streamlit app in your browser.

  Local URL: http://localhost:8501
  Network URL: http://192.168.178.21:8501
```

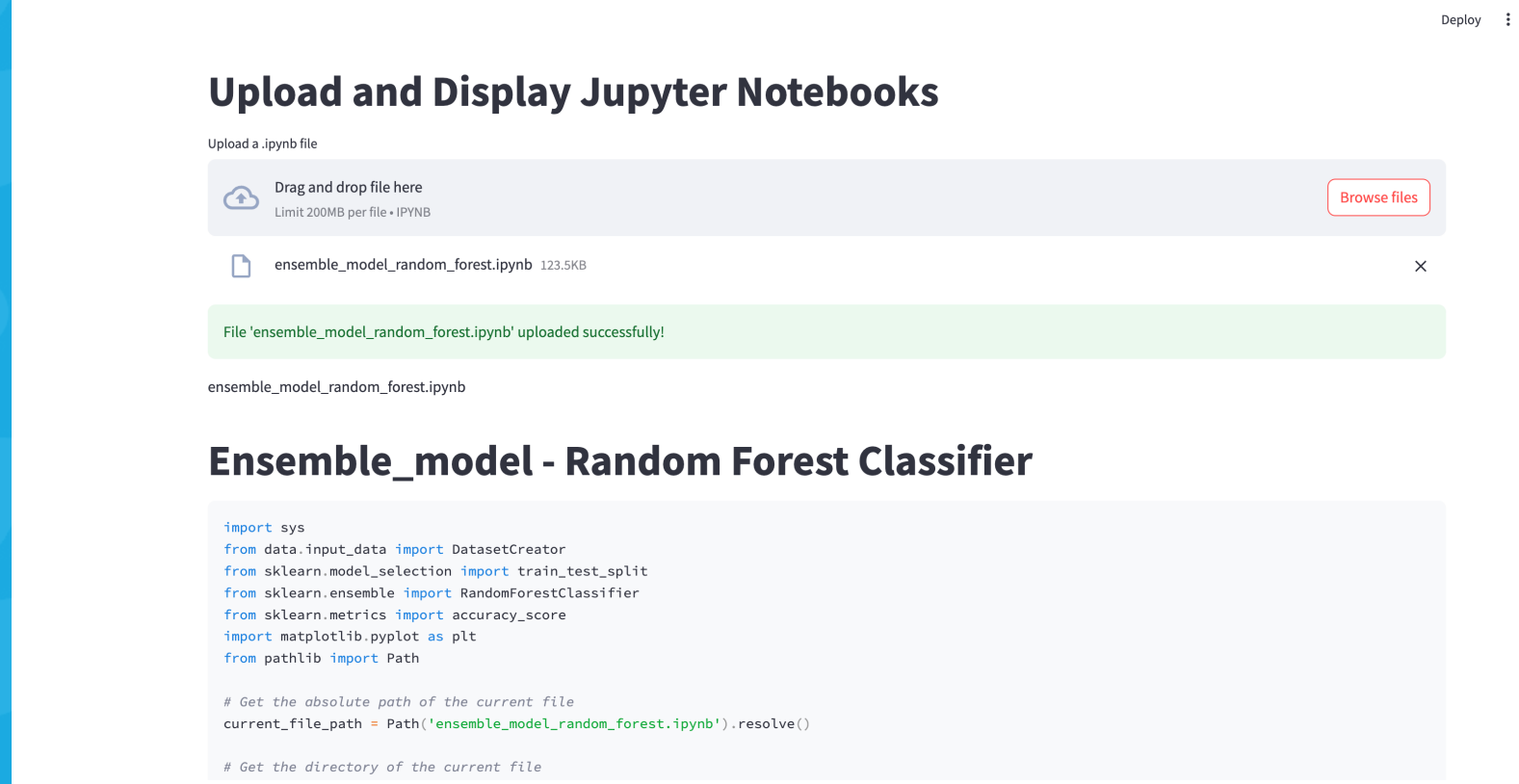
WEB APP WITHOUT NOTEBOOK FILES

It comprises a button for Browse Files and once clicked you can upload your Jupyter Notebook from local file



Uploaded File

- This is how the uploaded file looks like.



Uploaded File

Deploy

```
# Get the directory of the current file
project_dir = current_file_path.parent

# Add the project directory to sys.path
sys.path.insert(0, str(project_dir))

# Step 1: Create Datasets
dataset_creator = DatasetCreator()
blob_dataset = dataset_creator.create_blob_dataset()
circles_dataset = dataset_creator.create_make_circles_dataset()

# Step 2: Split Data into Training, Validation, and Test Sets
X_blob, y_blob = blob_dataset['X'], blob_dataset['y']
X_circles, y_circles = circles_dataset['X'], circles_dataset['y']

# Split blob dataset into training and temporary (remaining) data
X_blob_train_temp, X_blob_test, y_blob_train_temp, y_blob_test = train_test_split(X_blob, y_blob, test_size=0.2,
                                                                                  random_state=42)
X_blob_train, X_blob_val, y_blob_train, y_blob_val = train_test_split(X_blob_train_temp, y_blob_train_temp,
                                                                      test_size=0.25, random_state=42)

print(f"Blob Dataset:")
print(f"Train set: {X_blob_train.shape}, Validation set: {X_blob_val.shape}, Test set: {X_blob_test.shape}")

# Split circles dataset into training and temporary (remaining) data
X_circles_train_temp, X_circles_test, y_circles_train_temp, y_circles_test = train_test_split(X_circles, y_circles,
                                                                                              test_size=0.2,
                                                                                              random_state=42)
X_circles_train, X_circles_val, y_circles_train, y_circles_val = train_test_split(X_circles_train_temp,
                                                                                  y_circles_train_temp, test_size=0.25,
                                                                                  random_state=42)

print(f"\nCircles Dataset:")
print(f"Train set: {X_circles_train.shape}, Validation set: {X_circles_val.shape}, Test set: {X_circles_test.shape}")
```

```
Blob Dataset:
Train set: (660, 2), Validation set: (220, 2), Test set: (220, 2)

Circles Dataset:
Train set: (300, 2), Validation set: (100, 2), Test set: (100, 2)
```

```python
# Step 3: Define a function to train and evaluate RandomForest models with different hyperparameters
def train_evaluate_rf(X_train, y_train, X_val, y_val, n_estimators_list, max_depth_list):
    results = []

    for n_estimators in n_estimators_list:
        for max_depth in max_depth_list:
            rf = RandomForestClassifier(n_estimators=n_estimators, max_depth=max_depth, random_state=42)
            rf.fit(X_train, y_train)
            val_pred = rf.predict(X_val)
            accuracy = accuracy_score(y_val, val_pred)
            results.append((n_estimators, max_depth, accuracy))
            print(f"n_estimators: {n_estimators}, max_depth: {max_depth}, Validation Accuracy: {accuracy:.4f}")

    return results


# Hyperparameters to test
n_estimators_list = [10, 50, 100, 200]
max_depth_list = [None, 10, 20, 30]

# Train and evaluate on blob dataset
blob_results = train_evaluate_rf(X_blob_train, y_blob_train, X_blob_val, y_blob_val, n_estimators_list, max_depth_list)

# Train and evaluate on circles dataset
circles_results = train_evaluate_rf(X_circles_train, y_circles_train, X_circles_val, y_circles_val, n_estimators_list, max_depth_list)
```

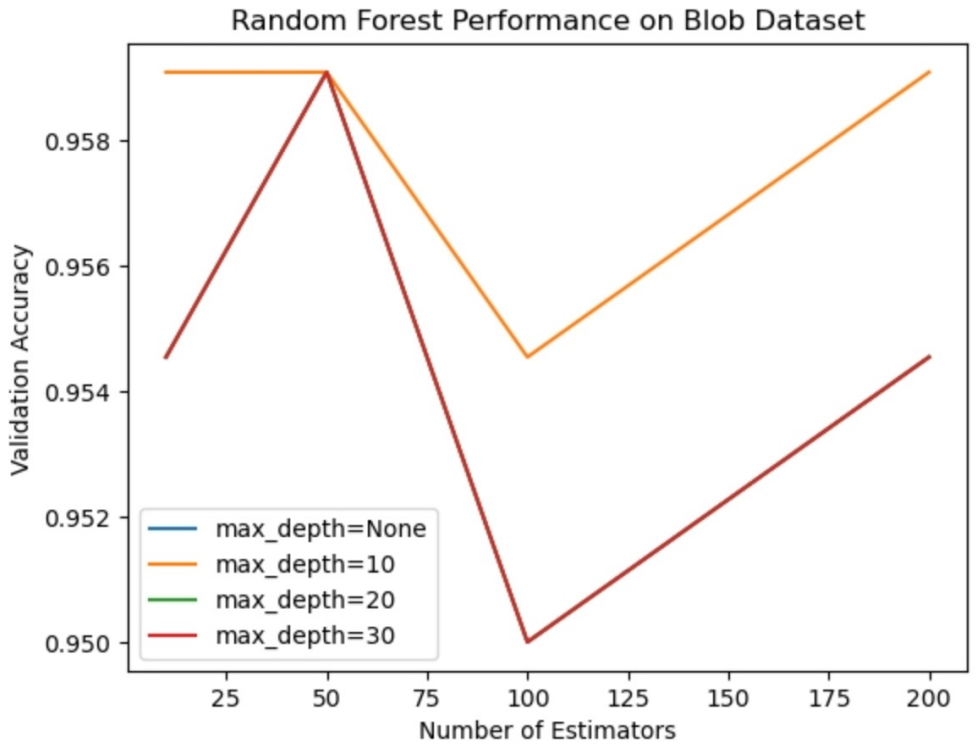
Uploaded File

```python
# Step 4: Plot the results
def plot_results(results, dataset_name):
    n_estimators_values = list(set([r[0] for r in results]))
    n_estimators_values.sort()

    for max_depth in max_depth_list:
        accuracies = [r[2] for r in results if r[1] == max_depth]
        plt.plot(n_estimators_values, accuracies, label=f"max_depth={m

    plt.xlabel('Number of Estimators')
    plt.ylabel('Validation Accuracy')
    plt.title(f'Random Forest Performance on {dataset_name} Dataset')
    plt.legend()
    plt.show()
```

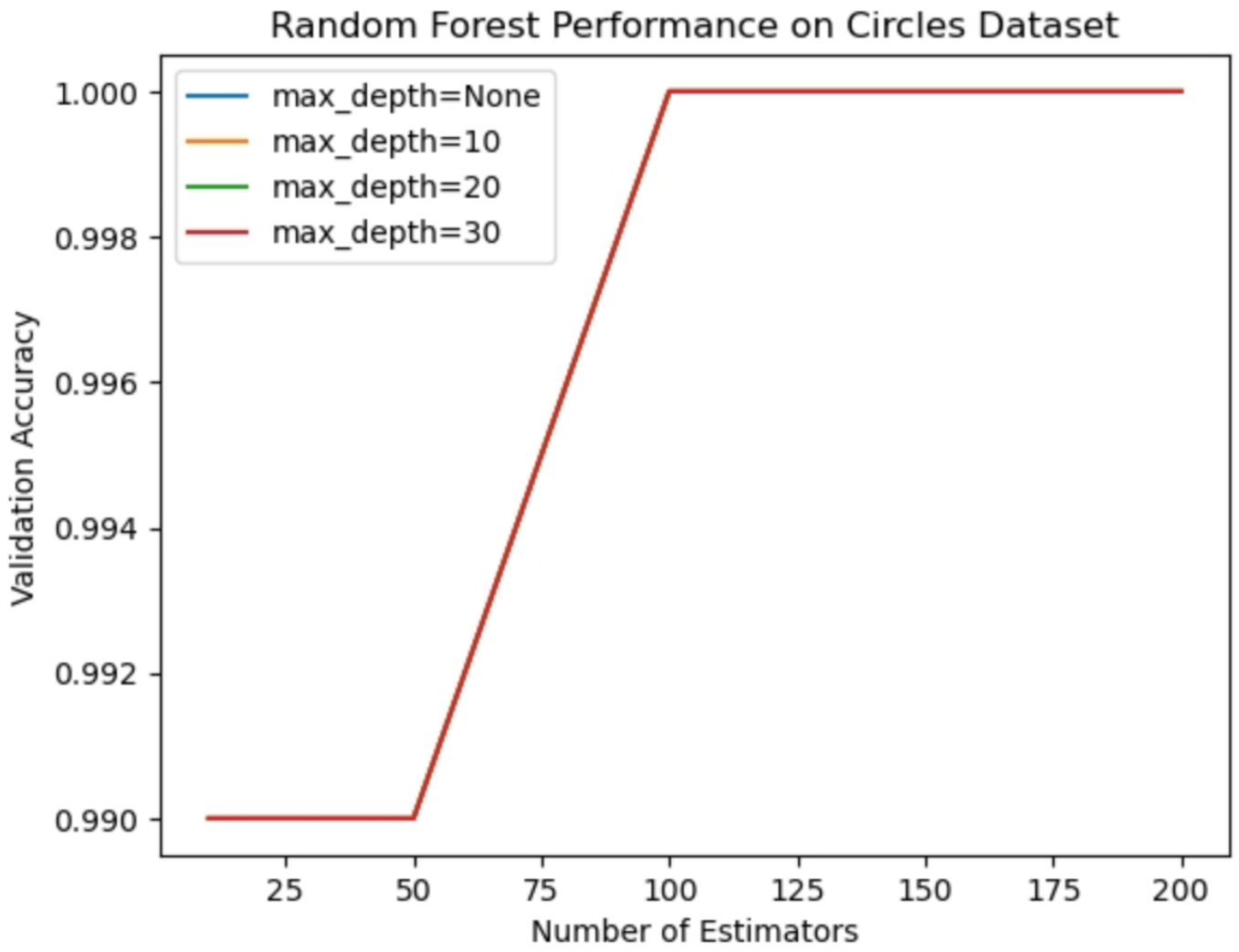
```python
plot_results(blob_results, "Blob")
plot_results(circles_results, "Circles")
```

Uploaded File

Visualisation on Blob Dataset




```
<Figure size 640x480 with 1 Axes>
```




```
<Figure size 640x480 with 1 Axes>
```

Visualization on Circle Dataset

```python
# Step 5: Evaluate the best ensemble on the test set
def evaluate_on_test_set(X_train, y_train, X_test, y_test, best_params):
    rf = RandomForestClassifier(n_estimators=best_params[0], max_depth=best_params[1], random_state=42)
    rf.fit(X_train, y_train)
    test_pred = rf.predict(X_test)
    test_accuracy = accuracy_score(y_test, test_pred)
    print(f"Test Accuracy: {test_accuracy:.4f}")


# Assuming best_params are (n_estimators, max_depth)
best_params_blob = max(blob_results, key=lambda x: x[2])
best_params_circles = max(circles_results, key=lambda x: x[2])

print("\nEvaluating the best ensemble on the test set for Blob dataset")
evaluate_on_test_set(X_blob_train, y_blob_train, X_blob_test, y_blob_test, best_params_blob)

print("\nEvaluating the best ensemble on the test set for Circles dataset")
evaluate_on_test_set(X_circles_train, y_circles_train, X_circles_test, y_circles_test, best_params_circle
```

```
Evaluating the best ensemble on the test set for Blob dataset
Test Accuracy: 0.9636

Evaluating the best ensemble on the test set for Circles dataset
Test Accuracy: 1.0000
```

Delete 'ensemble_model_random_forest.ipynb'

Evaluation of ensemble model

- Evaluation of ensemble model to test set from Blob and Circle
- We can also click the **delete** button to eliminate the notebook from the web app

Conclusion

- The study demonstrates that increasing the number of estimators generally improves model performance until it plateaus. The effect of max_depth is less pronounced but still contributes to fine-tuning performance.
- For both datasets (i.e., Blob and Circles), the best performance was achieved with **n_estimators=200** and **max_depth=None,** highlighting the importance of leveraging the full depth of trees in Random Forest models for complex data structures.

Input data

- To be used by ensemble_model_random_forest model

```python
from sklearn.datasets import make_blobs, make_circles


Allen Mwandunga
class DatasetCreator:
    Allen Mwandunga
    @staticmethod
    def create_blob_dataset() -> dict:
        n_samples_1 = 1000
        n_samples_2 = 100
        centers = [[0.0, 0.0], [2.0, 2.0]]
        cluster_std = [1.5, 0.5]

        X, y = make_blobs(n_samples=[n_samples_1, n_samples_2],
                          centers=centers,
                          cluster_std=cluster_std,
                          random_state=0,
                          shuffle=False)

        return {'X': X, 'y': y}

    Allen Mwandunga
    @staticmethod
    def create_make_circles_dataset() -> dict:
        X, y = make_circles( n_samples: 500, factor=0.1, noise=0.1)
        return {'X': X, 'y': y}
```

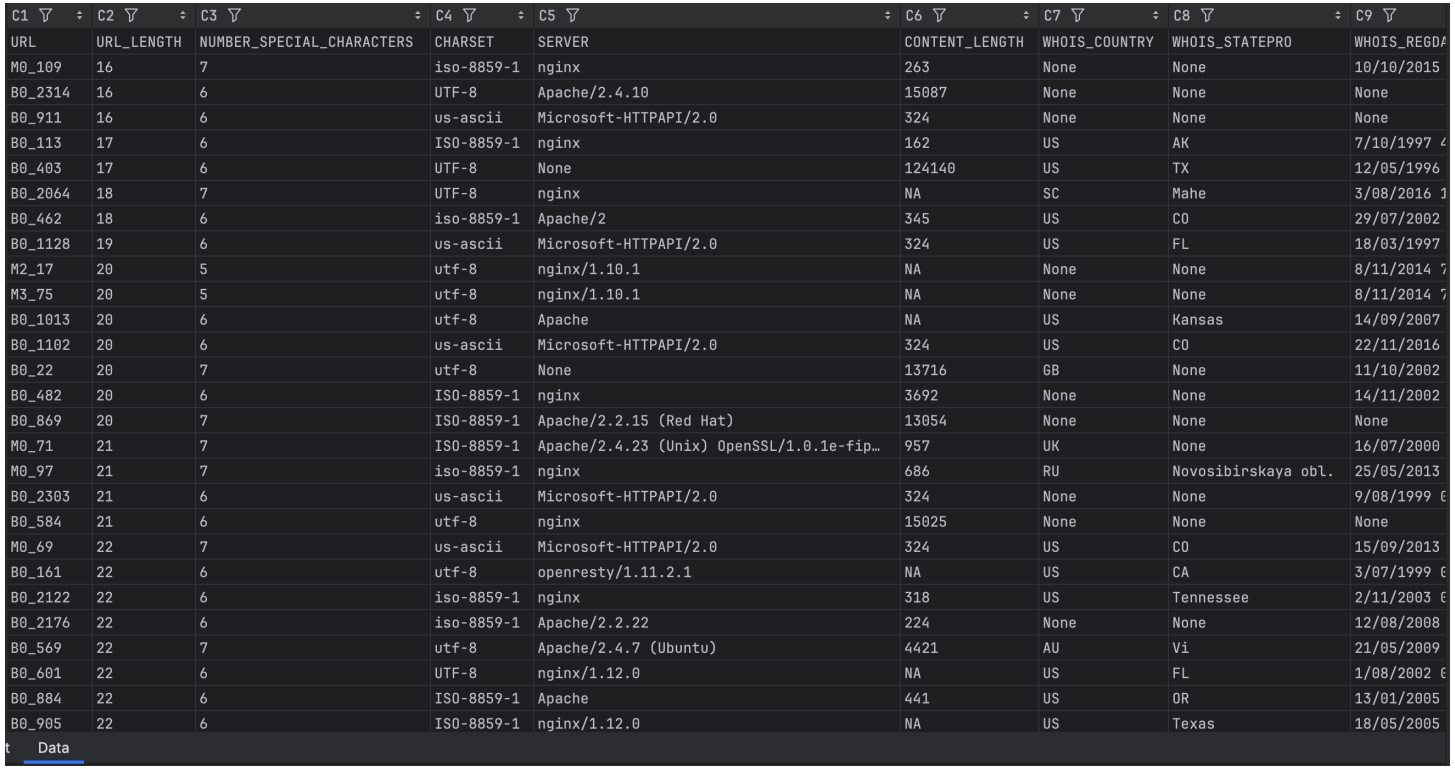
input data

- To be used by

```python
class DatasetCreator:
    Allen Mwandunga
    @staticmethod
    def create_blob_dataset() -> dict:
        n_samples_1 = 1000
        n_samples_2 = 100
        centers = [[0.0, 0.0], [2.0, 2.0]]
        cluster_std = [2.0, 1.0]

        X, y = make_blobs(n_samples=[n_samples_1, n_samples_2],
                          centers=centers,
                          cluster_std=cluster_std,
                          random_state=0,
                          shuffle=False)

        return {'X': X, 'y': y}



    Allen Mwandunga
    @staticmethod
    def create_points_dataset() -> dict:
        np.random.seed(0)
        n_points_per_cluster = 30000
        c1 = [-6, -2] + 0.7 * np.random.randn(n_points_per_cluster, 2)
        c2 = [-2, 2] + 0.3 * np.random.randn(n_points_per_cluster, 2)
        c3 = [1, -2] + 0.2 * np.random.randn(n_points_per_cluster, 2)
        c4 = [4, 4] + 0.1 * np.random.randn(n_points_per_cluster, 2)
        c5 = [5, 0] + 1.4 * np.random.randn(n_points_per_cluster, 2)
        c6 = [5, 6] + 2.0 * np.random.randn(n_points_per_cluster, 2)
        x = np.vstack((c1, c2, c3, c4, c5, c6))
        return {'X': x}
```

Malicious and Benign dataset

- Input dataset to be used by Decision Tree – Classification model

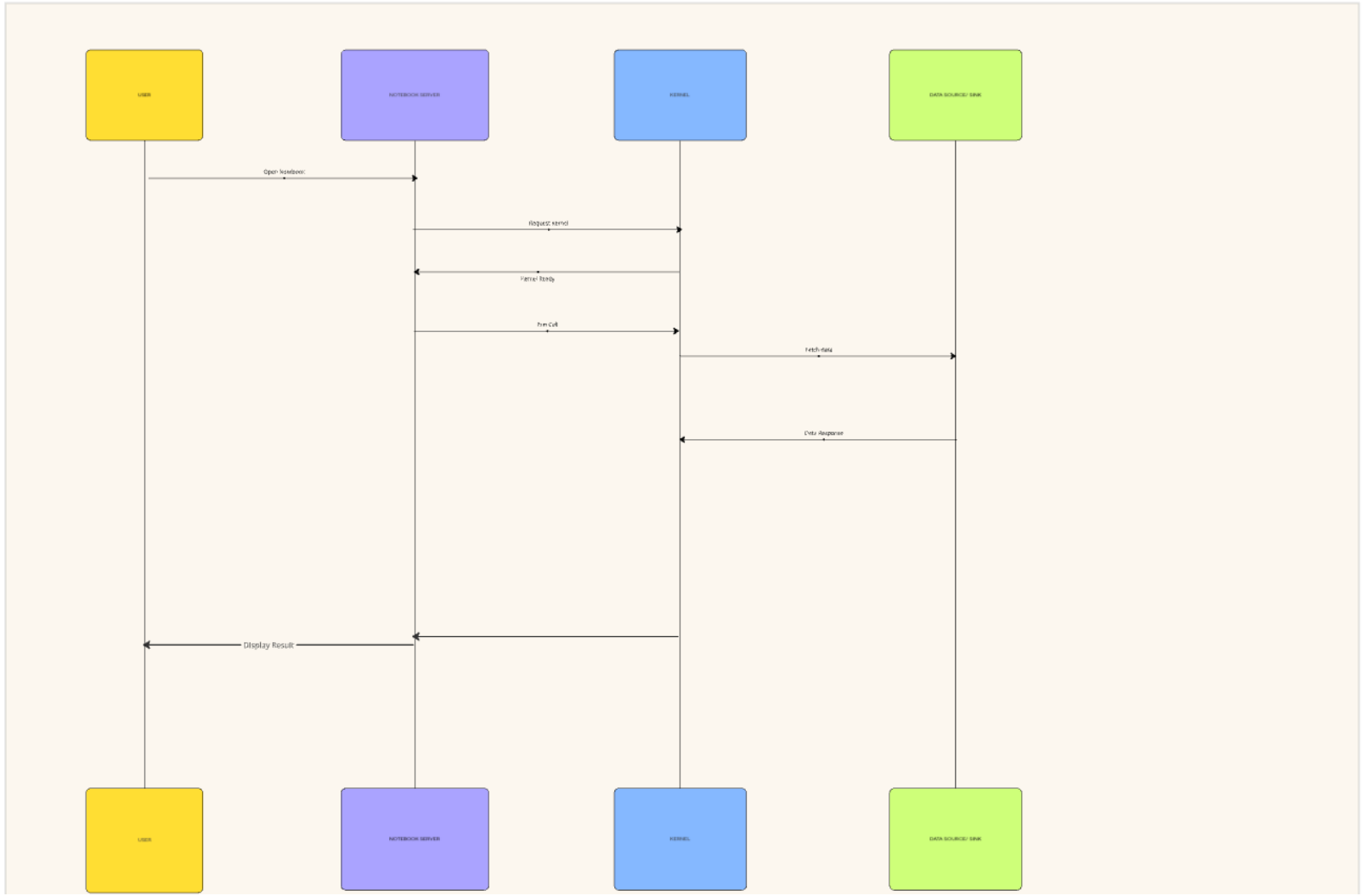


C1	C2	C3	C4	C5	C6	C7	C8	C9
URL	URL_LENGTH	NUMBER_SPECIAL_CHARACTERS	CHARSET	SERVER	CONTENT_LENGTH	WHOIS_COUNTRY	WHOIS_STATEPRO	WHOIS_REGDA
M0_109	16	7	iso-8859-1	nginx	263	None	None	10/10/2015
B0_2314	16	6	UTF-8	Apache/2.4.10	15087	None	None	None
B0_911	16	6	us-ascii	Microsoft-HTTPAPI/2.0	324	None	None	None
B0_113	17	6	ISO-8859-1	nginx	162	US	AK	7/10/1997 4
B0_403	17	6	UTF-8	None	124140	US	TX	12/05/1996
B0_2064	18	7	UTF-8	nginx	NA	SC	Mahe	3/08/2016 1
B0_462	18	6	iso-8859-1	Apache/2	345	US	CO	29/07/2002
B0_1128	19	6	us-ascii	Microsoft-HTTPAPI/2.0	324	US	FL	18/03/1997
M2_17	20	5	utf-8	nginx/1.10.1	NA	None	None	8/11/2014 7
M3_75	20	5	utf-8	nginx/1.10.1	NA	None	None	8/11/2014 7
B0_1013	20	6	utf-8	Apache	NA	US	Kansas	14/09/2007
B0_1102	20	6	us-ascii	Microsoft-HTTPAPI/2.0	324	US	CO	22/11/2016
B0_22	20	7	utf-8	None	13716	GB	None	11/10/2002
B0_482	20	6	ISO-8859-1	nginx	3692	None	None	14/11/2002
B0_869	20	7	ISO-8859-1	Apache/2.2.15 (Red Hat)	13054	None	None	None
M0_71	21	7	ISO-8859-1	Apache/2.4.23 (Unix) OpenSSL/1.0.1e-fip…	957	UK	None	16/07/2000
M0_97	21	7	iso-8859-1	nginx	686	RU	Novosibirskaya obl.	25/05/2013
B0_2303	21	6	us-ascii	Microsoft-HTTPAPI/2.0	324	None	None	9/08/1999 0
B0_584	21	6	utf-8	nginx	15025	None	None	None
M0_69	22	7	us-ascii	Microsoft-HTTPAPI/2.0	324	US	CO	15/09/2013
B0_161	22	6	utf-8	openresty/1.11.2.1	NA	US	CA	3/07/1999 0
B0_2122	22	6	iso-8859-1	nginx	318	US	Tennessee	2/11/2003 0
B0_2176	22	6	iso-8859-1	Apache/2.2.22	224	None	None	12/08/2008
B0_569	22	7	utf-8	Apache/2.4.7 (Ubuntu)	4421	AU	Vi	21/05/2009
B0_601	22	6	UTF-8	nginx/1.12.0	NA	US	FL	1/08/2002 0
B0_884	22	6	ISO-8859-1	Apache	441	US	OR	13/01/2005
B0_905	22	6	ISO-8859-1	nginx/1.12.0	NA	US	Texas	18/05/2005

t Data

UML Diagram (Sequence)

Sequence Diagram



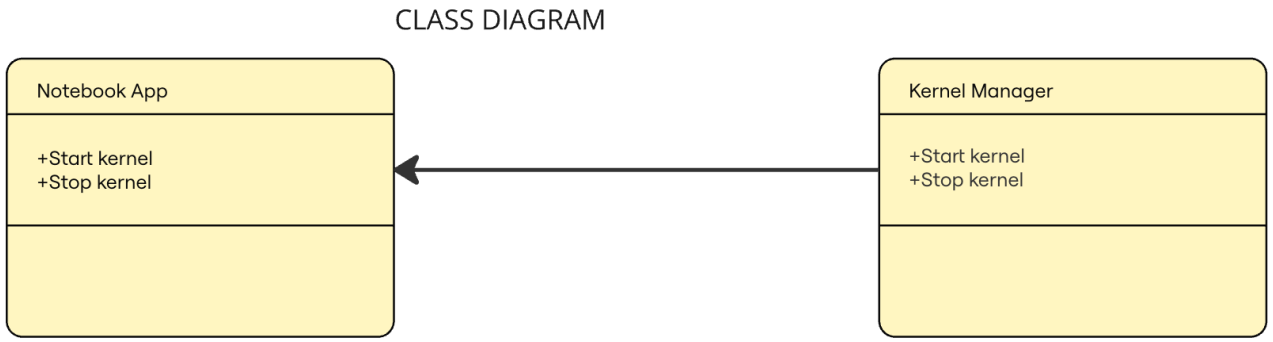
Architecture of the Jupyter Notebook

```
n_estimators: 10, max_depth: None, Validation Accuracy: 0.9545
n_estimators: 10, max_depth: 10, Validation Accuracy: 0.9591
n_estimators: 10, max_depth: 20, Validation Accuracy: 0.9545
n_estimators: 10, max_depth: 30, Validation Accuracy: 0.9545
n_estimators: 50, max_depth: None, Validation Accuracy: 0.9591
n_estimators: 50, max_depth: 10, Validation Accuracy: 0.9591
n_estimators: 50, max_depth: 20, Validation Accuracy: 0.9591
n_estimators: 50, max_depth: 30, Validation Accuracy: 0.9591
n_estimators: 100, max_depth: None, Validation Accuracy: 0.9500
n_estimators: 100, max_depth: 10, Validation Accuracy: 0.9545
n_estimators: 100, max_depth: 20, Validation Accuracy: 0.9500
n_estimators: 100, max_depth: 30, Validation Accuracy: 0.9500
n_estimators: 200, max_depth: None, Validation Accuracy: 0.9545
n_estimators: 200, max_depth: 10, Validation Accuracy: 0.9591
n_estimators: 200, max_depth: 20, Validation Accuracy: 0.9545
n_estimators: 200, max_depth: 30, Validation Accuracy: 0.9545
n_estimators: 10, max_depth: None, Validation Accuracy: 0.9900
n_estimators: 10, max_depth: 10, Validation Accuracy: 0.9900
n_estimators: 10, max_depth: 20, Validation Accuracy: 0.9900
n_estimators: 10, max_depth: 30, Validation Accuracy: 0.9900
n_estimators: 50, max_depth: None, Validation Accuracy: 0.9900
n_estimators: 50, max_depth: 10, Validation Accuracy: 0.9900
n_estimators: 50, max_depth: 20, Validation Accuracy: 0.9900
n_estimators: 50, max_depth: 30, Validation Accuracy: 0.9900
n_estimators: 100, max_depth: None, Validation Accuracy: 1.0000
n_estimators: 100, max_depth: 10, Validation Accuracy: 1.0000
n_estimators: 100, max_depth: 20, Validation Accuracy: 1.0000
n_estimators: 100, max_depth: 30, Validation Accuracy: 1.0000
n_estimators: 200, max_depth: None, Validation Accuracy: 1.0000
n_estimators: 200, max_depth: 10, Validation Accuracy: 1.0000
n_estimators: 200, max_depth: 20, Validation Accuracy: 1.0000
n_estimators: 200, max_depth: 30, Validation Accuracy: 1.0000
```

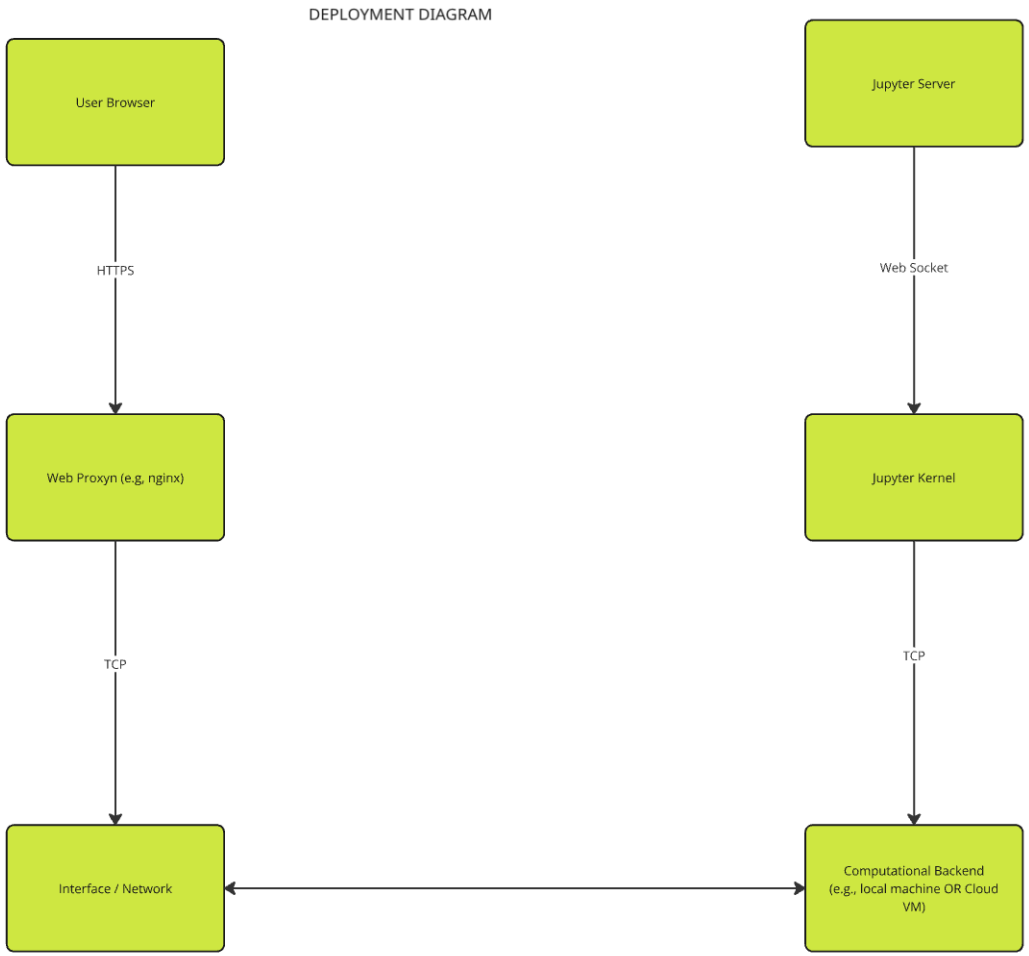
Results from train_evaluate_rf method

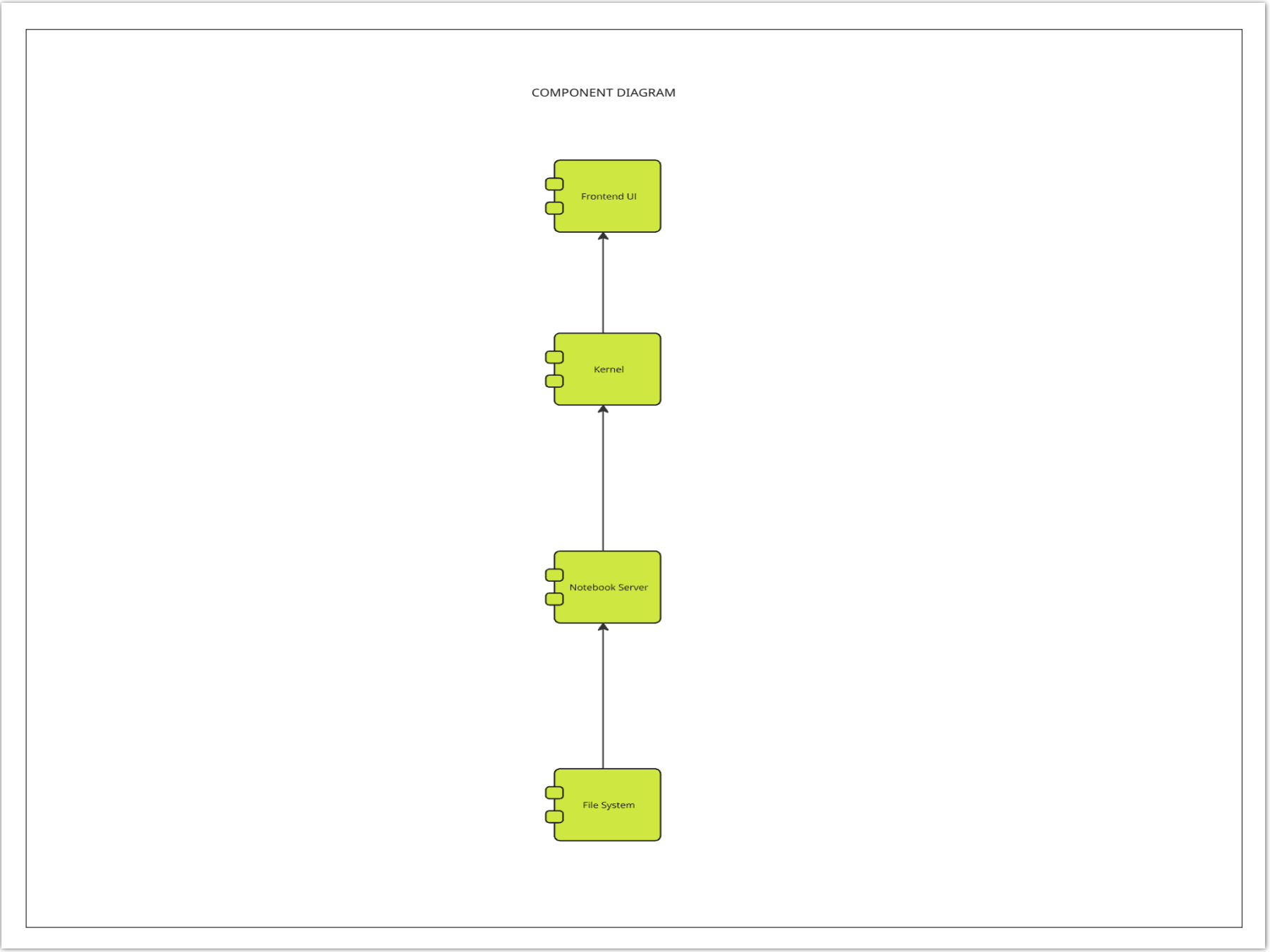
- This is an example of the output

Architecture of the Jupyter Notebook



Architecture of the Jupyter Notebook





Architecture of the Jupyter Notebook