Let's create a Flask APP to demonstrate a simple AutoML pipeline

The Web-App will perform the following things -

1. Take a csv file as an input
2. Perform some Exploratory data analysis - data sanity checks
3. Perform certain cleaning operations
4. Allows users to select required features for building model
5. Builds machine learning models based on the users selected
6. Displays model performance on test data

Let's create the index.html page - that takes in form input to upload a csv file

```
<html>
<head>
<title> Data Science with Flask </title>
<body>
    <div class="container">
        <h1>Data Science with Flask</h1>
        <form action="/upload" method="POST" enctype="multipart/form-data">
            <label for="csv_file" class="custom-file-upload">
                Select a CSV File
            </label>
            <input type="file" name="csv_file" id="csv_file" accept=".csv">
            <input type="submit" value="Upload CSV" >
        </form>
    </div>
</body>
</html>
```

Adding some CSS to make the page better looking

```
<!DOCTYPE html>
<html>
<head>
    <title>Data Science with Flask</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            background-color: #f4f4f4;
            text-align: center;
        }

        h1 {
            color: #333;
        }

        .container {
            max-width: 400px;
            margin: 0 auto;
            background-color: #fff;
            padding: 20px;
            border-radius: 5px;
            box-shadow: 0 0 10px rgba(0, 0, 0, 0.2);
        }

        input[type="file"] {
            display: none;
        }

        .custom-file-upload {
            border: 1px solid #ccc;
            display: inline-block;
            padding: 6px 12px;
            cursor: pointer;
            background-color: #4CAF50;
            color: white;
            border-radius: 5px;
        }

        .custom-file-upload:hover {
            background-color: #45a049;
        }
    </style>
</head>
<body>
    <div class="container">
        <h1>Data Science with Flask</h1>
        <form action="/upload" method="POST" enctype="multipart/form-data">
            <label for="csv_file" class="custom-file-upload">
                Select a CSV File
            </label>
            <input type="file" name="csv_file" id="csv_file" accept=".csv">
            <input type="submit" value="Upload CSV" >
        </form>
    </div>
</body>
</html>
```

Once we have created the front end - we'll need to create a Flask route to render this page

```
@app.route('/')
def index():
    return render_template('index.html')
```

Now let's create the python code

```
from flask import Flask, render_template, request, redirect, url_for, send_file
app = Flask(__name__)
@app.route('/')
def index():
    return render_template('index.html')



if __name__ == '__main__':
    app.run(debug=True)
```

We initially had the form which has takes a form input which has the action /upload. Now let's write the python script for what happens when the file has been uploaded

```
import pandas as pd
```

```
@app.route('/upload', methods=['POST'])
def upload():
    if 'csv_file' not in request.files:
        return redirect(request.url)

    file = request.files['csv_file']
    if file.filename == '':
        return redirect(request.url)

    if file:
        data = pd.read_csv(file)
        uploaded_data = data
        print(uploaded_data)

    return "None"
```

Once we've verified that the dataframe is being uploaded properly let's do some pandas operations and plot some interesting visuals

```
import plotly.graph_objects as go
import plotly.express as px
```

```
data = data.iloc[:,1:]
# Separate columns by data type
numeric_columns = data.select_dtypes(include=['number']).columns
categorical_columns = data.select_dtypes(exclude=['number']).columns
```

```python
# Task 2: Count null values for each variable
null_counts = data.isnull().sum()

# Task 3: Categorical distribution
categorical_distributions = []
for col in categorical_columns:
    plot = px.bar(data, x=col, title=f'Distribution of {col}')
    categorical_distributions.append({'plot': plot.to_json(), 'name': col})

# Task 4: Histogram distribution for numeric columns
histogram_plots = []
for col in numeric_columns:
    plot = px.histogram(data, x=col, title=f'Histogram of {col}')
    histogram_plots.append({'plot': plot.to_json(), 'name': col})

# Task 5: Correlation heatmap for numeric columns
correlation_heatmap = go.Figure(data=go.Heatmap(
    z=data[numeric_columns].corr(),
    x=numeric_columns,
    y=numeric_columns,
    colorscale='Viridis'
))

# Task 6: Contingency plot for the last column as the target variable
# Create a contingency table
contingency_plots = []
df = pd.DataFrame(data)

# Create separate contingency plots for each variable
variables_to_plot = ['Gender', 'Married', 'Education', 'Self_Employed', 'Property_Area']

for variable in variables_to_plot:
    contingency_table = pd.crosstab(df['Loan_Status'], df[variable])

    fig = px.imshow(contingency_table, labels=dict(x=variable, y='Loan Status', color='Count'),
                    x=contingency_table.columns,
                    y=contingency_table.index,
                    color_continuous_scale='YlGnBu',
                    title=f"Contingency Plot of Loan Status with {variable}")

    contingency_plots.append({'plot': fig.to_json(), 'name': variable})
```

Now we have some basic level code let's pass these visuals to the front end. For this we render the template as result.html and pass the necessary components through the code

```
return render_template('results.html',
                               numeric_columns=numeric_columns,
                               categorical_columns=categorical_columns,
                               null_counts=null_counts,
                               categorical_distributions=categorical_distributions,
                               histogram_plots=histogram_plots,
                               correlation_heatmap=correlation_heatmap.to_json(),
                               contingency_plots=contingency_plots)
```

Now lets create some front end for this

```
<!DOCTYPE html>

<html>
<head>
    <title>Data Analysis Results</title>
    <script src="https://cdn.plot.ly/plotly-latest.min.js"></script>
    <style>
        /* Modal Styles */
        .modal {
            display: none;
            position: fixed;
            z-index: 1;
            left: 0;
            top: 0;
            width: 100%;
            height: 100%;
            background-color: rgba(0, 0, 0, 0.7);
        }

        .modal-content {
            background-color: #fff;
            margin: 15% auto;
            padding: 20px;
            border: 1px solid #888;
            width: 60%;
            text-align: center;
        }
    </style>
</head>
<html>
<body>
    <h1>Data Analysis Results</h1>
    <h2>Null Value Counts</h2>
    <table>
        <tr>
            <th>Variable</th>
            <th>Null Count</th>
        </tr>
        {% for variable, null_count in null_counts.items() %}
        <tr>
            <td>{{ variable }}</td>
            <td>{{ null_count }}</td>
        </tr>
        {% endfor %}
    </table>

    <h2>Categorical Type Columns</h2>
    {% for categorical_distribution in categorical_distributions %}
        <h3>{{ categorical_distribution['name'] }}</h3>
        <div id="categorical-{{ loop.index }}"></div>
    {% endfor %}

    <h2>Numeric Type Columns</h2>
    {% for histogram_plot in histogram_plots %}
        <h3>{{ histogram_plot['name'] }}</h3>
        <div id="numeric-{{ loop.index }}"></div>
    {% endfor %}

    <h2>Correlation Heatmap for Numeric Columns</h2>
    <div id="correlation-heatmap"></div>

    <h2>Contingency Plots</h2>
    <div class="chart-pair">
        {% for contingency_plot in contingency_plots %}
```

```
            <div>
                <h3>{{ contingency_plot['name'] }}</h3>
                <div id="contingency-{{ loop.index }}"></div>
            </div>
            {% if loop.index is divisibleby(2) %}
                </div><div class="chart-pair">
            {% endif %}
        {% endfor %}
    </div>

    <script>
        var correlation_heatmap_div = document.getElementById('correlation-heatmap');
        var contingency_plot_div = document.getElementById('contingency-plot');
        var correlation_heatmap_data = {{ correlation_heatmap|safe }};
        var contingency_plot_div = document.getElementById('contingency-plot');
        Plotly.newPlot(correlation_heatmap_div, correlation_heatmap_data);

        {% for categorical_distribution in categorical_distributions %}
            var categorical_plot{{ loop.index }} = document.getElementById('categorical-{{ loop.index }}');
            var categorical_plot{{ loop.index }}_data = {{ categorical_distribution['plot']|safe }};
            Plotly.newPlot(categorical_plot{{ loop.index }}, categorical_plot{{ loop.index }}_data);
        {% endfor %}

        {% for histogram_plot in histogram_plots %}
            var numeric_plot{{ loop.index }} = document.getElementById('numeric-{{ loop.index }}');
            var numeric_plot{{ loop.index }}_data = {{ histogram_plot['plot']|safe }};
            Plotly.newPlot(numeric_plot{{ loop.index }}, numeric_plot{{ loop.index }}_data);
        {% endfor %}

        {% for contingency_plot in contingency_plots %}
            var contingency_plot{{ loop.index }} = document.getElementById('contingency-{{ loop.index }}');
            var contingency_plot{{ loop.index }}_data = {{ contingency_plot['plot']|safe }};
            Plotly.newPlot(contingency_plot{{ loop.index }}, contingency_plot{{ loop.index }}_data);
        {% endfor %}
    </script>
</body>
</html>
```

What Next?

We may want to Perform Cleaning Operations on this data. So let's add some sort of a functionality where on a click of a button, some cleaning operations are performed in the backend.

Add this block after the </script>

```
<button id="cleaning-button">Perform Cleaning Operations</button>

    <!-- Modal dialog for displaying cleaning operation messages -->
    <div id="cleaning-modal" class="modal">
        <div class="modal-content">
            <h2>Cleaning Operations</h2>
            <p id="cleaning-message">Cleaning in progress...</p>
        </div>
    </div>
```

Let's add the functionality for this button

```
<script>
    // Get the cleaning button and modal
    var cleaningButton = document.getElementById('cleaning-button');
    var cleaningModal = document.getElementById('cleaning-modal');
    var cleaningMessage = document.getElementById('cleaning-message');

    // When the cleaning button is clicked, show the modal and trigger cleaning operations
    cleaningButton.addEventListener('click', function() {
        cleaningModal.style.display = 'block';

        // Perform cleaning operations via a POST request to the server
        fetch('/clean_data', {
            method: 'POST',
        })
        .then(response => response.json())
        .then(data => {
            // Update the cleaning message with the response
            cleaningMessage.innerText = data.message;

            // If cleaning is successful, provide a download link
            if (data.success) {
                var downloadLink = document.createElement('a');
                downloadLink.href = data.download_link;
                downloadLink.innerText = 'Download Cleaned Data';
                cleaningMessage.appendChild(downloadLink);
            }
        })
        .catch(error => {
            console.error(error);
        });
    });
</script>
```

So, in the above code we've added the Button and the modal. Let's now add some functionality to the button. On clicking the perform cleaning operations we want the python backend to perform the appropriate steps and allow the user to download the data

```
## The final results.html will look like this

<!DOCTYPE html>

<html>
<head>
    <title>Data Analysis Results</title>
    <script src="https://cdn.plot.ly/plotly-latest.min.js"></script>
    <style>
        /* Modal Styles */
        .modal {
            display: none;
            position: fixed;
            z-index: 1;
            left: 0;
            top: 0;
            width: 100%;
            height: 100%;
            background-color: rgba(0, 0, 0, 0.7);
        }

        .modal-content {
            background-color: #fff;
            margin: 15% auto;
            padding: 20px;
            border: 1px solid #888;
            width: 60%;
            text-align: center;
        }
    </style>
</head>
<body>
    <h1>Data Analysis Results</h1>
    <h2>Null Value Counts</h2>
    <table>
        <tr>
            <th>Variable</th>
            <th>Null Count</th>
        </tr>
        {% for variable, null_count in null_counts.items() %}
        <tr>
            <td>{{ variable }}</td>
            <td>{{ null_count }}</td>
        </tr>
        {% endfor %}
    </table>

    <h2>Categorical Type Columns</h2>
    {% for categorical_distribution in categorical_distributions %}
        <h3>{{ categorical_distribution['name'] }}</h3>
        <div id="categorical-{{ loop.index }}"></div>
    {% endfor %}

    <h2>Numeric Type Columns</h2>
    {% for histogram_plot in histogram_plots %}
        <h3>{{ histogram_plot['name'] }}</h3>
        <div id="numeric-{{ loop.index }}"></div>
    {% endfor %}
    <h2>Correlation Heatmap for Numeric Columns</h2>
    <div id="correlation-heatmap"></div>

    <h2>Contingency Plots</h2>
    <div class="chart-pair">
        {% for contingency_plot in contingency_plots %}
            <div>
                <h3>{{ contingency_plot['name'] }}</h3>
                <div id="contingency-{{ loop.index }}"></div>
            </div>
            {% if loop.index is divisibleby(2) %}
                </div><div class="chart-pair">
            {% endif %}
        {% endfor %}
    </div>

    <script>
        var correlation_heatmap_div = document.getElementById('correlation-heatmap');
        var contingency_plot_div = document.getElementById('contingency-plot');
        var correlation_heatmap_data = {{ correlation_heatmap|safe }};
        var contingency_plot_div = document.getElementById('contingency-plot');
        Plotly.newPlot(correlation_heatmap_div, correlation_heatmap_data);

        {% for categorical_distribution in categorical_distributions %}
            var categorical_plot{{ loop.index }} = document.getElementById('categorical-{{ loop.index }}');
            var categorical_plot{{ loop.index }}_data = {{ categorical_distribution['plot']|safe }};
            Plotly.newPlot(categorical_plot{{ loop.index }}, categorical_plot{{ loop.index }}_data);
        {% endfor %}

        {% for histogram_plot in histogram_plots %}
            var numeric_plot{{ loop.index }} = document.getElementById('numeric-{{ loop.index }}');
            var numeric_plot{{ loop.index }}_data = {{ histogram_plot['plot']|safe }};
            Plotly.newPlot(numeric_plot{{ loop.index }}, numeric_plot{{ loop.index }}_data);
        {% endfor %}

        {% for contingency_plot in contingency_plots %}
            var contingency_plot{{ loop.index }} = document.getElementById('contingency-{{ loop.index }}');
            var contingency_plot{{ loop.index }}_data = {{ contingency_plot['plot']|safe }};
            Plotly.newPlot(contingency_plot{{ loop.index }}, contingency_plot{{ loop.index }}_data);
        {% endfor %}
    </script>
    <!-- Add this button to trigger the cleaning operations -->
    <button id="cleaning-button">Perform Cleaning Operations</button>

    <!-- Modal dialog for displaying cleaning operation messages -->
    <div id="cleaning-modal" class="modal">
        <div class="modal-content">
            <h2>Cleaning Operations</h2>
            <p id="cleaning-message">Cleaning in progress...</p>
            <a href="{{ url_for('modelling') }}">Go to Modeling</a>
        </div>
    </div>
    <script>
    // Get the cleaning button and modal
    var cleaningButton = document.getElementById('cleaning-button');
    var cleaningModal = document.getElementById('cleaning-modal');
    var cleaningMessage = document.getElementById('cleaning-message');

    // When the cleaning button is clicked, show the modal and trigger cleaning operations
    cleaningButton.addEventListener('click', function() {
        cleaningModal.style.display = 'block';

        // Perform cleaning operations via a POST request to the server
        fetch('/clean_data', {
            method: 'POST',
        })
        .then(response => response.json())
        .then(data => {
            // Update the cleaning message with the response
            cleaningMessage.innerText = data.message;

            // If cleaning is successful, provide a download link
            if (data.success) {
                var downloadLink = document.createElement('a');
                downloadLink.href = data.download_link;
                downloadLink.innerText = 'Download Cleaned Data';
                cleaningMessage.appendChild(downloadLink);
            }
        })
        .catch(error => {
            console.error(error);
        });
    });
</script>
```

```
</body>
</html>
```

Now let's create the backend for Cleaning the data

```
import io
import json
```

```
@app.route('/clean_data', methods=['POST'])
def clean_data():
    # Perform your cleaning operations on the uploaded data
    # For example, remove null values, check for data inconsistencies, etc.
    data = pd.read_csv("data/LoanApprovalPrediction.csv")
    cleaned_data = data  # Replace with your cleaning logic

    # Save the cleaned data to a CSV file in memory
    cleaned_data_csv = io.StringIO()
    cleaned_data.to_csv(cleaned_data_csv, index=False)
    cleaned_data_csv.seek(0)

    # Provide a download link for the user
    download_link = url_for('download_cleaned_data')

    response = {
        'success': True,
        'message': 'Cleaning operations completed successfully.',
        'download_link': download_link,
    }

    return json.dumps(response)
```

Now, the functionality for downloading the cleaned data

```
@app.route('/download_cleaned_data')
def download_cleaned_data():
    # Retrieve the cleaned data and provide it as a downloadable file
    # You should have the cleaned data available in your cleaning logic
    data = pd.read_csv("data/LoanApprovalPrediction.csv")
    cleaned_data = data.dropna()
    cleaned_data_csv = io.StringIO()
    cleaned_data.to_csv(cleaned_data_csv, index=False)
    cleaned_data_csv.seek(0)

    # Save the cleaned data CSV file in the current directory
    cleaned_data_filename = 'data/cleaned_data.csv'
    cleaned_data_csv_path = os.path.join(os.getcwd(), cleaned_data_filename)

    print(cleaned_data_csv_path)


    cleaned_data.to_csv(cleaned_data_csv_path, index=False)

    return send_file(cleaned_data_csv_path,
                     as_attachment=True,
                     download_name=cleaned_data_filename,
                     mimetype='text/csv')
```

Now that the User has the cleaned data, let's move on to the modelling part of it.

Add this piece of code to the modal in results.html

```
<a href="{{ url_for('modelling') }}">Go to Modeling</a>
```

Initially, the modelling part handles only the GET request.

That is, gets the data from the previous step and renders the html

```
@app.route('/modelling', methods=['GET', 'POST'])
def modelling():
    data = pd.read_csv("data/cleaned_data.csv")

    if data is None:
        # Handle the case where data is not available
        return "Data not available. Please upload data first."

    # Get the list of available features
    available_features = data.columns.tolist()

    return render_template('modelling.html', features=available_features)
```

Now let's write the html page for the modelling part

```
<html>
<head>
<body>
    <h1>Data Modeling</h1>
    <form method="POST">
        <div>
            <label for="features">Select Features:</label>
            <select name="features[]" id="features" multiple>
                {% for feature in features %}
                <option value="{{ feature }}">{{ feature }}</option>
                {% endfor %}
            </select>
        </div>
        <div>
            <label for="model">Select Model:</label>
            <select name="model" id="model">
                <option value="LogisticRegression">Logistic Regression</option>
                <option value="DecisionTree">Decision Tree</option>
            </select>
        </div>
        <button type="submit">Build Model</button>
    </form>
</body>
```

```
</html>
```

Now once the features are selected and the model is selected we shall move on to the python backend which build the model and produces the metrics

```
if request.method == 'POST':
        # Get selected features and model choice from the form
        selected_features = request.form.getlist('features[]')
        print(selected_features)
        selected_model = request.form['model']


        # Prepare the data with selected features
        X = data[selected_features]
        y = data['Loan_Status']

        # Encode categorical variables if needed
        encoder = LabelEncoder()
        for col in X.select_dtypes(include='object'):
            X[col] = encoder.fit_transform(X[col])

        # Split the data into train and test sets
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

        # Initialize the selected model
        if selected_model == 'LogisticRegression':
            model = LogisticRegression()
        elif selected_model == 'DecisionTree':
            model = DecisionTreeClassifier()

        # Train the model
        model.fit(X_train, y_train)

        # Make predictions
        y_pred = model.predict(X_test)

        # Calculate accuracy
        accuracy = accuracy_score(y_test, y_pred)

        # Generate a classification report
        report = classification_report(y_test, y_pred)

        return render_template('metrics.html', accuracy=accuracy, classification_report=report)
```

The overall modelling code looks like this

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score, classification_report
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
```

```
@app.route('/modelling', methods=['GET', 'POST'])
def modelling():

    data = pd.read_csv("data/cleaned_data.csv")

    if data is None:
        # Handle the case where data is not available
        return "Data not available. Please upload data first."
    if request.method == 'POST':
        # Get selected features and model choice from the form
        selected_features = request.form.getlist('features[]')
        print(selected_features)
        selected_model = request.form['model']


        # Prepare the data with selected features
        X = data[selected_features]
        y = data['Loan_Status']

        # Encode categorical variables if needed
        encoder = LabelEncoder()
        for col in X.select_dtypes(include='object'):
            X[col] = encoder.fit_transform(X[col])

        # Split the data into train and test sets
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
        # Initialize the selected model
        if selected_model == 'LogisticRegression':
            model = LogisticRegression()
        elif selected_model == 'DecisionTree':
            model = DecisionTreeClassifier()

        # Train the model
        model.fit(X_train, y_train)

        # Make predictions
        y_pred = model.predict(X_test)

        # Calculate accuracy
        accuracy = accuracy_score(y_test, y_pred)

        # Generate a classification report
        report = classification_report(y_test, y_pred)

        return render_template('metrics.html',  selected_features =selected_features, accuracy=accuracy,
classification_report=report)

    # Get the list of available features
    available_features = data.columns.tolist()

    return render_template('modelling.html', features=available_features)
```

Now let's build the metrics.html page

```
<!DOCTYPE html>
<html>
<head>
    <title>Model Metrics</title>
</head>
<body>
    <h1>Model Metrics</h1>
    <p> Selected Features: {{selected_features}}
    <p>Accuracy: {{ accuracy }}</p>
    <h2>Classification Report:</h2>
    <pre>{{ classification_report }}</pre>
</body>
</html>
```