Data Intake Report

Name: Cloud and API deployment
Report date: 7th July 2024
Internship Batch: LISUM34
Version:1.0
Data intake by: Biswadip Bhattacharyya
Data intake reviewer:<intern who reviewed the report>
Data storage location: <location URL eg: github, cloud>

Tabular data details:

Total number of observations	1000
Total number of files	1
Total number of features	7
Base format of the file	.csv
Size of the data	20 kb

1.Dataset:

The Dataset provides a comprehensive view into the dynamics of online matchmaking interactions. It captures essential variables that influence the likelihood of successful matches across different genders.

Variables:

- **Gender:** 0 (Male), 1 (Female)
- **PurchasedVIP:** 0 (No), 1 (Yes)
- **Income:** Annual income in USD
- **Children:** Number of children
- **Age:** Age of the user
- **Attractiveness:** Subjective rating of attractiveness (1-10)
- **Matches:** Number of matches obtained based on criteria

Target Variable:

- **Matches:** Number of matches received, indicative of success rate in online dating

2. Investigation of data:

```
import pandas as pd

df=pd.read_csv("C:/ads/DataGlacier/week4/Online_Dating_Behavior_Dataset.csv")
df
```

✓ 1.0s

	Gender	PurchasedVIP	Income	Children	Age	Attractiveness	Matches
0	0	1	51777	3	47	5	70
1	1	0	36646	0	42	7	130
2	0	0	53801	1	25	5	0
3	0	0	56105	0	35	8	0
4	0	0	55597	1	36	6	0
...	...	...	...	...	...	...	...
995	0	0	36799	0	28	1	0
996	0	1	43882	2	46	9	70
997	1	0	49629	2	49	6	120
998	1	0	45706	1	22	8	140
999	0	0	43075	0	43	3	0

1000 rows × 7 columns

```
df.info()
```

[2] ✓ 0.0s

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 7 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Gender          1000 non-null   int64
 1   PurchasedVIP    1000 non-null   int64
 2   Income          1000 non-null   int64
 3   Children        1000 non-null   int64
 4   Age             1000 non-null   int64
 5   Attractiveness  1000 non-null   int64
 6   Matches         1000 non-null   int64
dtypes: int64(7)
memory usage: 54.8 KB
```

```
df.isna().sum()
```

[3] ✓ 0.0s

```
Gender          0
PurchasedVIP    0
Income          0
Children        0
Age             0
Attractiveness  0
Matches         0
dtype: int64
```

```
duplicates=df.duplicated()
duplicates
```

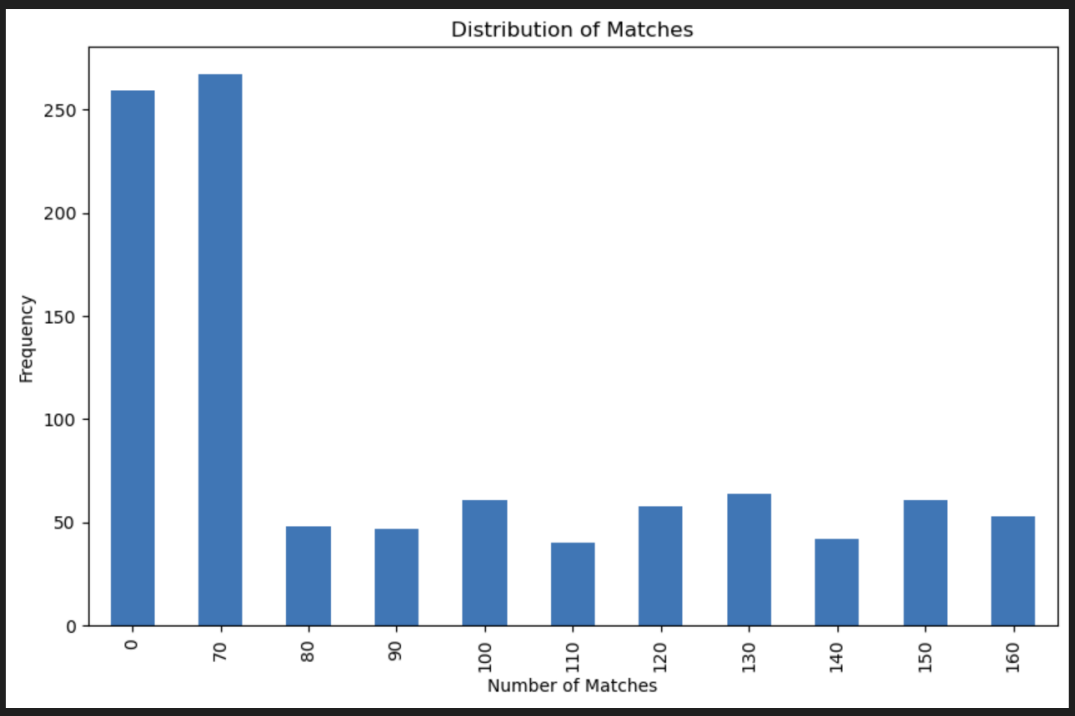
✓ 0.0s

```
0      False
1      False
2      False
3      False
4      False
       ...
995    False
996    False
997    False
998    False
999    False
Length: 1000, dtype: bool
```

```python
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
df['Matches'].value_counts().sort_index().plot(kind='bar')
plt.title('Distribution of Matches')
plt.xlabel('Number of Matches')
plt.ylabel('Frequency')
plt.show()
```

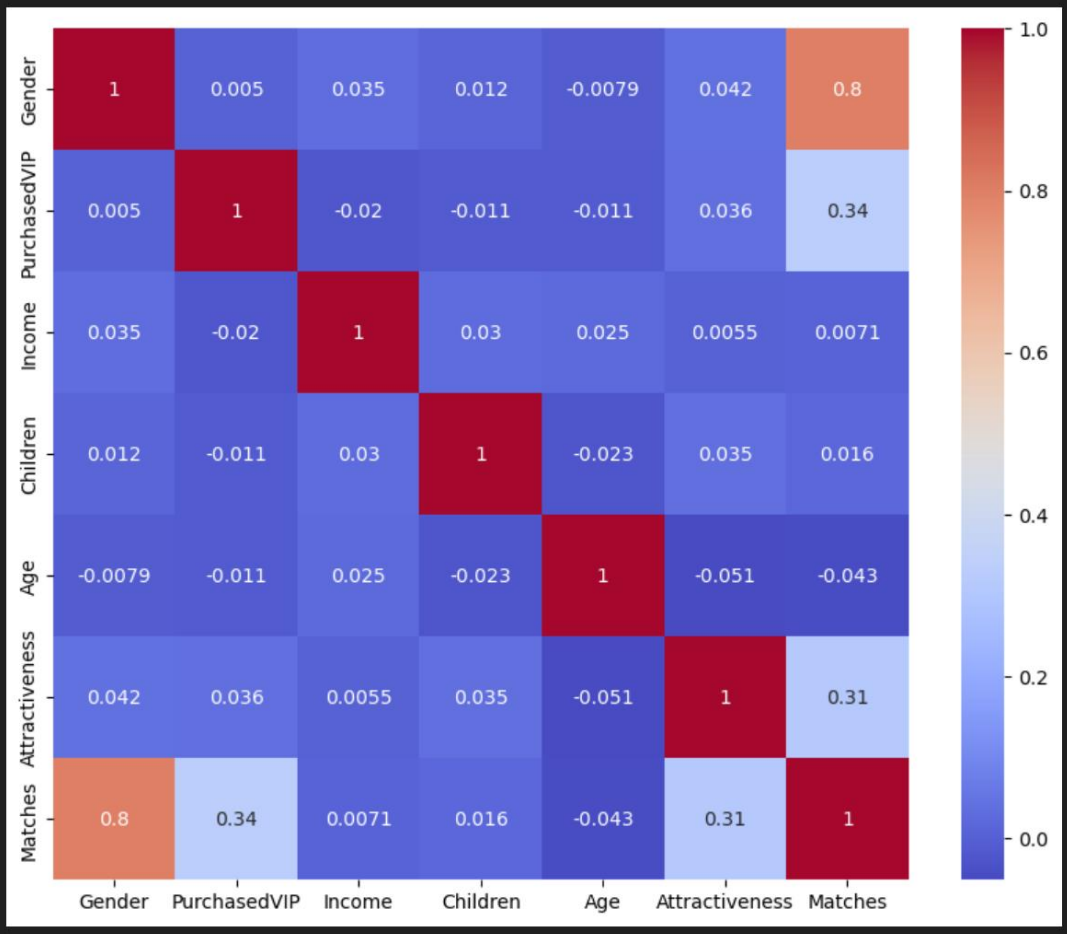
✓ 0.9s



```
import seaborn as sns
corr_matrix = df.corr()

# Plot heatmap of the correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.show()
```

✓ 1.3s



```python
import numpy as np



# Calculate quartiles
Q1 = np.percentile(df['Income'], 25)
Q3 = np.percentile(df['Income'], 75)

# Calculate IQR
IQR = Q3 - Q1

# Define limits
lower_limit = Q1 - 1.5 * IQR
upper_limit = Q3 + 1.5 * IQR

# Remove outliers
filtered_data = df[(df['Income'] >= lower_limit) & (df['Income'] <= upper_limit)]

print("Original data:", df)
print("Filtered data (without outliers):", filtered_data)
Q1
```

✓ 0.0s

```
Original data:      Gender  PurchasedVIP  Income  Children  Age  Attractiveness  Matches
0         0             1   51777         3   47               5       70
1         1             0   36646         0   42               7      130
2         0             0   53801         1   25               5        0
3         0             0   56105         0   35               8        0
4         0             0   55597         1   36               6        0
..      ...           ...     ...       ...  ...             ...      ...
995       0             0   36799         0   28               1        0
996       0             1   43882         2   46               9       70
997       1             0   49629         2   49               6      120
998       1             0   45706         1   22               8      140
999       0             0   43075         0   43               3        0

[1000 rows x 7 columns]
Filtered data (without outliers):      Gender  PurchasedVIP  Income  Children  Age  Attractiveness  Matches
0         0             1   51777         3   47               5       70
1         1             0   36646         0   42               7      130
2         0             0   53801         1   25               5        0
3         0             0   56105         0   35               8        0
```

3. Model building:

```
x=filtered_data.drop(columns=['Matches'])
y=filtered_data['Matches']
```

✓ 0.0s

+ Code + Markdo

```
x
```

✓ 0.0s

	Gender	PurchasedVIP	Income	Children	Age	Attractiveness
0	0	1	51777	3	47	5
1	1	0	36646	0	42	7
2	0	0	53801	1	25	5
3	0	0	56105	0	35	8
4	0	0	55597	1	36	6
...	...	...	...	...	...	...
995	0	0	36799	0	28	1
996	0	1	43882	2	46	9
997	1	0	49629	2	49	6
998	1	0	45706	1	22	8
999	0	0	43075	0	43	3

992 rows × 6 columns

```
import pandas as pd
from sklearn.model_selection import train_test_split


# Perform train-test split
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)

# Check the shapes of the resulting datasets
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

✓ 0.3s

```
X_train shape: (793, 6)
X_test shape: (199, 6)
y_train shape: (793,)
y_test shape: (199,)
```

```python
from sklearn.preprocessing import StandardScaler


scaler = StandardScaler()

# Fit the scaler on the training data and transform the training data
X_train_scaled = scaler.fit_transform(X_train)

# Transform the test data (note: we only transform the test data, not fit)
X_test_scaled = scaler.transform(X_test)

# Check the scaled training data
print("Scaled X_train:\n", X_train_scaled)
print("Scaled X_test:\n", X_test_scaled)
```

✓ 0.0s

```
Scaled X_train:
 [[-1.01653005 -0.97385982 -0.4262749   1.01920173  1.46738229  0.17512899]
 [ 0.98373875  1.02684183 -0.64275935  1.01920173 -1.71224557  1.22457954]
 [ 0.98373875 -0.97385982 -0.42688586 -1.0013657  -0.72546451  0.17512899]
 ...
 [ 0.98373875 -0.97385982  0.86377286  0.00891802 -0.83510685  0.87476269]
 [ 0.98373875 -0.97385982  0.14405879  0.00891802 -0.06761047  1.22457954]
 [ 0.98373875  1.02684183 -0.63838078  1.01920173  0.59024357 -1.22413841]]
Scaled X_test:
 [[-1.01653005  1.02684183 -0.83439799  0.00891802  0.69988591 -0.52450471]
 [-1.01653005  1.02684183  0.36471816 -1.0013657   0.48060123 -1.57395526]
 [ 0.98373875 -0.97385982  0.834243    2.02948545 -1.49296089 -0.87432156]
 ...
 [-1.01653005 -0.97385982 -1.16818728  1.01920173  0.15167421  0.17512899]
 [ 0.98373875  1.02684183 -0.63267847 -1.0013657   1.02881293 -0.17468786]
 [-1.01653005 -0.97385982 -1.33131432  1.01920173  0.26131655 -0.17468786]]
```

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
import joblib
rf_regressor = RandomForestRegressor(n_estimators=100, random_state=42)

# Train the model on the scaled training data
rf_regressor.fit(X_train_scaled, y_train)

# Make predictions on the test set
y_pred = rf_regressor.predict(X_test_scaled)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)
print("Root Mean Squared Error:", mse**0.5)

joblib.dump(rf_regressor, 'random_forest_model.pkl')
joblib.dump(scaler, 'scaler.pkl')
```

14] ✓ 0.4s

```
Mean Squared Error: 0.0
Root Mean Squared Error: 0.0

['scaler.pkl']
```

4. app.py

add.py > predict

```python
from flask import Flask, request, render_template
import joblib
import numpy as np

app = Flask(__name__)

# Load the trained model and scaler
model = joblib.load('random_forest_model.pkl')
scaler = joblib.load('scaler.pkl')

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

@app.route('/predict', methods=['POST'])
def predict():
    gender = int(request.form['gender'])
    vip = int(request.form['vip'])
    income = float(request.form['income'])
    children = int(request.form['children'])
    age = int(request.form['age'])
    attractiveness = int(request.form['attractiveness'])

    # Create the input array for prediction
    input_data = np.array([[gender, vip, income, children, age, attractiveness]])

    # Scale the input data
    input_data_scaled = scaler.transform(input_data)

    # Perform the prediction
    predicted_matches = model.predict(input_data_scaled)[0]

    return f'<h1>Predicted Matches: {predicted_matches}</h1>'

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

Index.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Predict Online Dating Matches</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='styles.css') }}">
</head>
<body>
    <div class="container">
        <h1>Predict Online Dating Matches</h1>
        <form action="/predict" method="post">
            <label for="gender">Gender:</label>
            <select id="gender" name="gender">
                <option value="0">Male</option>
                <option value="1">Female</option>
            </select>

            <label for="vip">Purchased VIP:</label>
            <select id="vip" name="vip">
                <option value="0">No</option>
                <option value="1">Yes</option>
            </select>

            <label for="income">Income (USD):</label>
            <input type="number" id="income" name="income" required>

            <label for="children">Number of Children:</label>
            <input type="number" id="children" name="children" required>

            <label for="age">Age:</label>
            <input type="number" id="age" name="age" required>

            <label for="attractiveness">Attractiveness (1-10):</label>
            <input type="number" id="attractiveness" name="attractiveness" min="1" max="10" required>

            <button type="submit">Predict Matches</button>
        </form>
    </div>
</body>
</html>
```

CSS

atic > # styles.css > button

```css
/* styles.css */
body {
    font-family: Arial, sans-serif;
    background: linear-gradient(to right, #6a11cb, #2575fc); /* Gradient backgrou
    margin: 0;
    padding: 0;
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100vh;
}

.container {
    background-color: rgba(255, 255, 255, 0.9); /* Semi-transparent white */
    padding: 20px 40px;
    border-radius: 10px;
    box-shadow: 0 0 20px rgba(0, 0, 0, 0.1);
    text-align: center;
}

h1 {
    color: #333;
    margin-bottom: 20px;
}

form {
    display: flex;
    flex-direction: column;
    align-items: center;
}

label {
    margin-bottom: 10px;
    font-weight: bold;
    color: #333;
}

input, select, button {
    margin-bottom: 20px;
    padding: 10px;
    font-size: 16px;
    width: 100%;
    max-width: 300px;
    box-sizing: border-box;
}
```

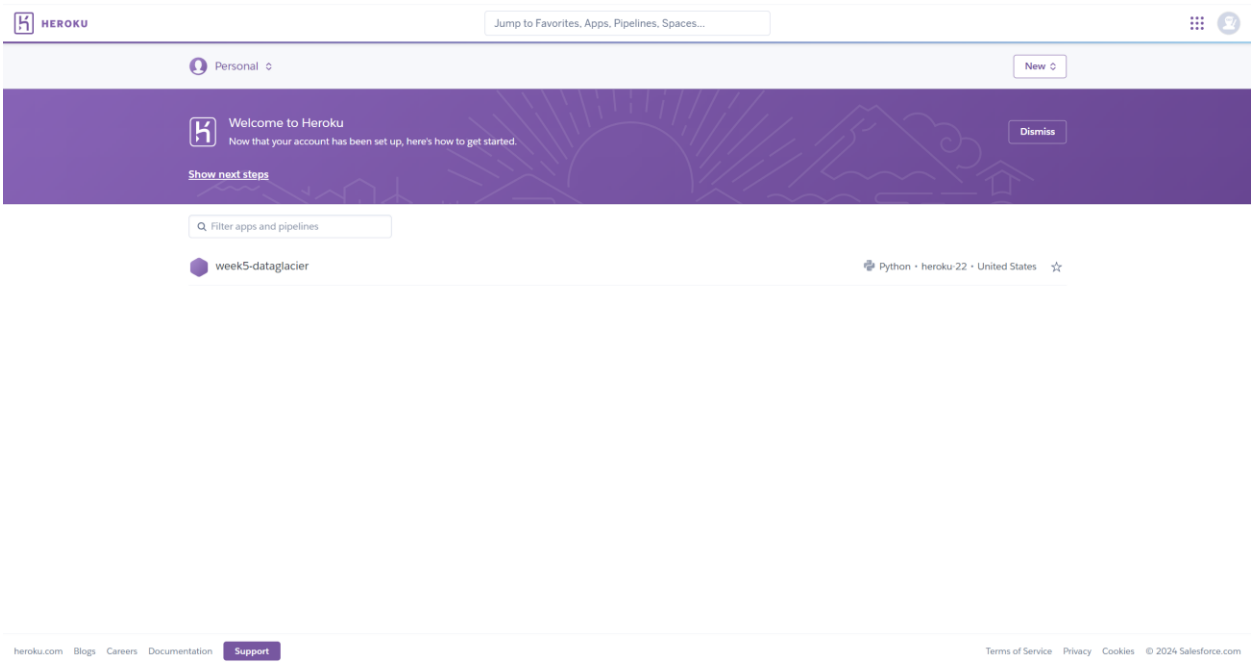
```css
button {
    background-color: #5cb85c;
    color: #fff;
    border: none;
    border-radius: 5px;
    cursor: pointer;
    transition: background-color 0.3s ease;
}

button:hover {
    background-color: #4cae4c;
}
```

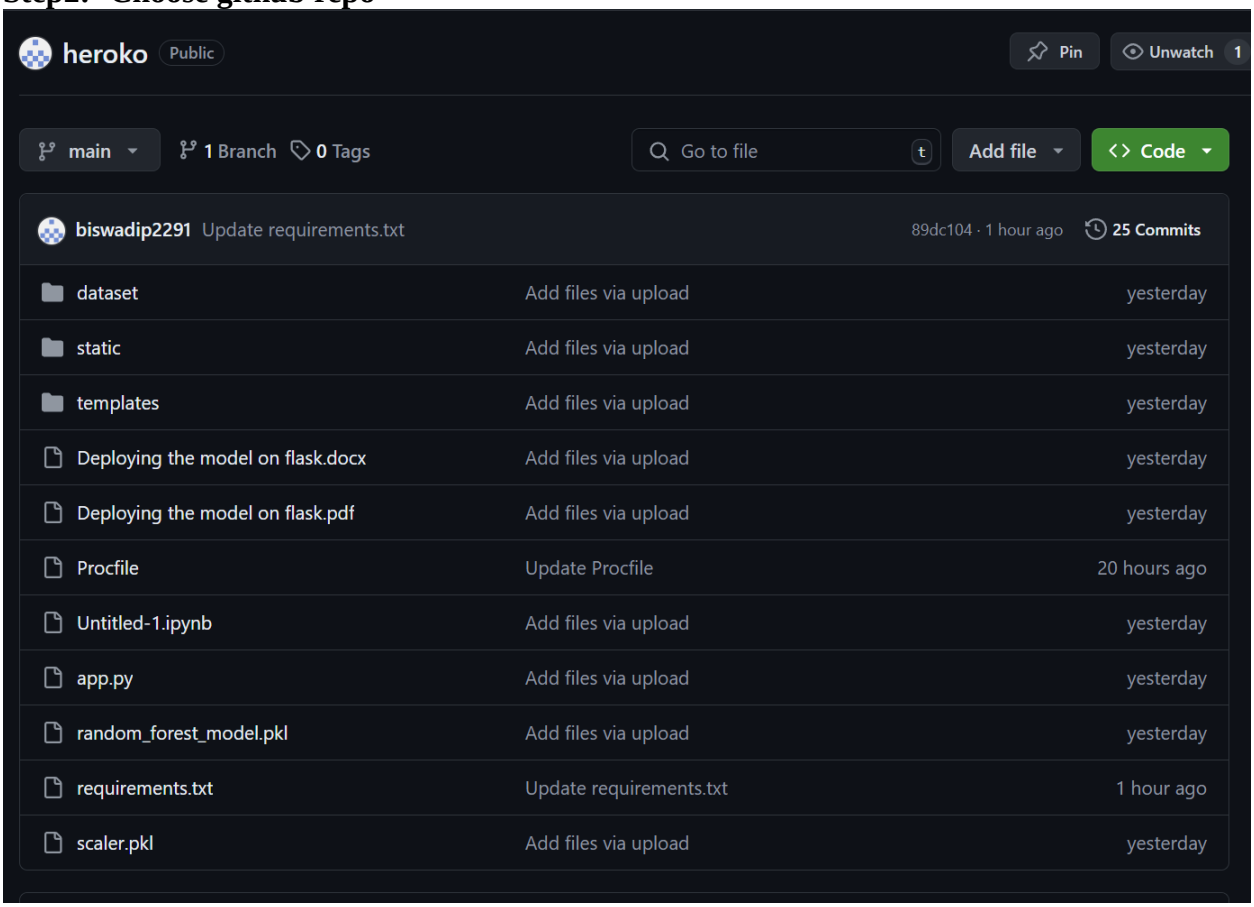
5. Deployment of web app on Heruku:

Step1:-

Create an account

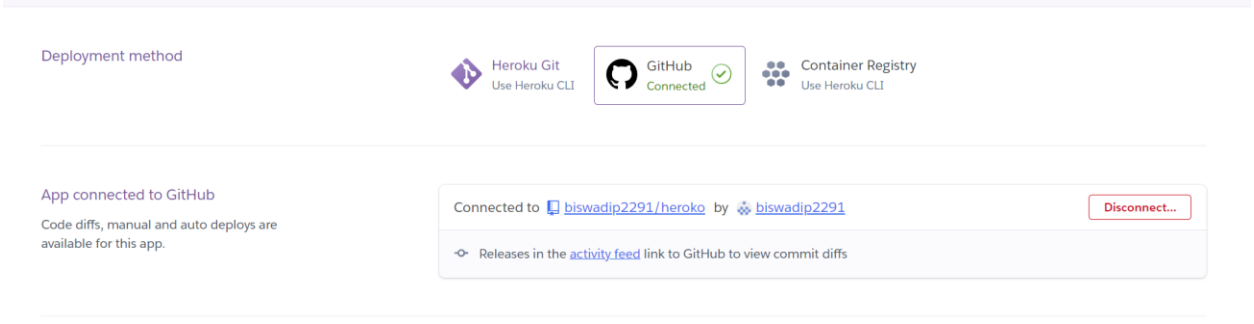


Step2:- Choose github repo



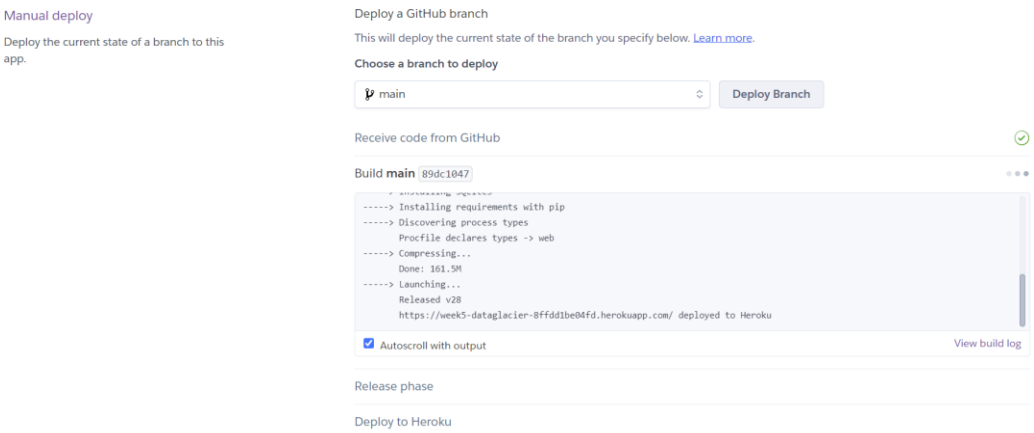
This is my Github Repo

Step 3:-
Connect Heroku to github account

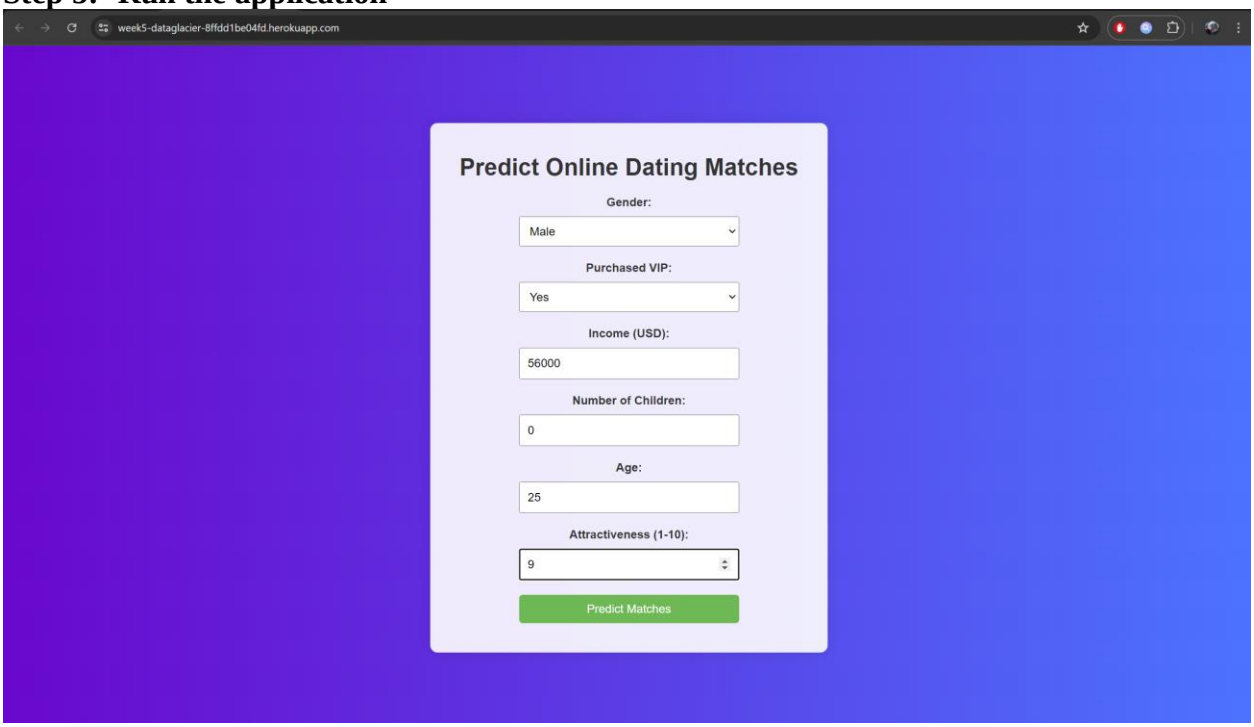


Step4:-

Deploy the application



Step 5:- Run the application



Predicted Matches: 70.0