```
import pandas as pd
import matplotlib.pyplot as plt
```

```
df=pd.read_csv('/content/tv_shows (1).csv')
df.shape
df.head()
```

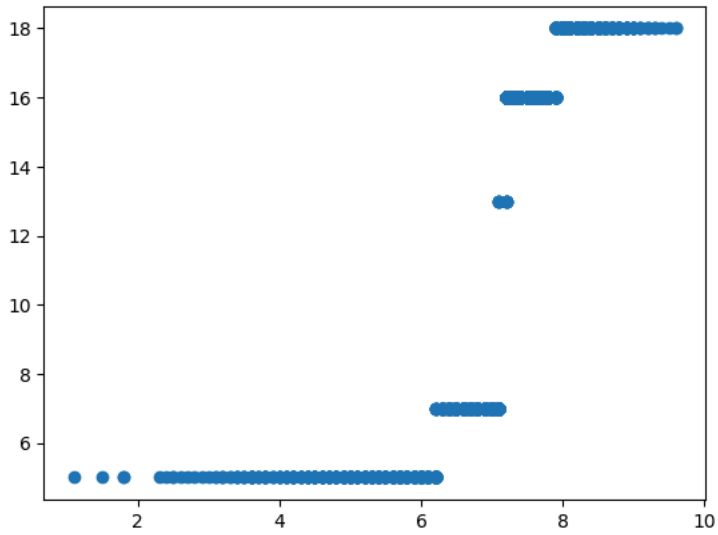
	Unnamed: 0	ID	Title	Year	Age	IMDb	Rotten Tomatoes	Netflix	Hulu	Prime Video	Disney+	Type
0	0	1	Breaking Bad	2008	5	1.1/10	100/100	1	0	0	0	1
1	1	2	Stranger Things	2016	5	1.5/10	96/100	1	0	0	0	1
2	2	3	Attack on Titan	2013	5	1.8/10	95/100	1	1	0	0	1
3	3	4	Better Call Saul	2015	5	1.8/10	94/100	1	0	0	0	1
4	4	5	Dark	2017	5	2.3/10	93/100	1	0	0	0	1

Next steps: View recommended plots

```
df['IMDb']=df['IMDb'].str.replace('/10',' ').astype(float)
```

```
plt.scatter(df.IMDb, df.Age)
```

```
<matplotlib.collections.PathCollection at 0x7aff91124520>
```




```
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest= train_test_split(df[['IMDb']],df.Age,test_size=0.3,random_state=27)
```

```
print(ytrain)
```

```
2676    18
1810    16
1966    16
1660    16
176      5
        ..
3096    18
1317     7
752      7
2591    18
1043     7
Name: Age, Length: 2244, dtype: int64
```

```
from sklearn.linear_model import LogisticRegression
regModel= LogisticRegression()
regModel.fit(xtrain,ytrain)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

▾ LogisticRegression

LogisticRegression()

```
regModel.predict(xtest)
```

```
array([18,  5,  7, 16, 16, 18, 18, 16, 18, 18,  7, 16, 16, 16, 16, 16, 16,
        5, 16, 16, 18,  5,  7, 18, 16, 18, 18, 16, 18,  5,  7, 16, 16, 18,
       16, 16, 18, 16, 16,  5,  7, 18, 18,  7,  5,  5,  5, 16, 16, 18, 18,
       16,  7,  7,  5,  5, 16,  7,  5,  5, 18, 16,  7,  5, 18, 18, 18,  5,
        5, 18,  5, 18, 18,  5, 18, 18, 16, 16, 18, 18, 18, 16, 16, 16, 16,
       18, 18,  5,  7, 16,  7,  5, 16,  7,  7, 16, 18,  7, 18, 16,  5, 16,
        7, 18,  7,  5, 18, 18,  5, 16, 18, 18,  7, 18,  7, 18, 18,  7, 18,
       18,  7, 18, 16, 18, 18,  7, 18, 16, 18, 16, 18, 16, 16, 16, 16, 18,
        7, 18,  7,  7,  5,  5,  5,  7,  7, 16, 18, 18, 16,  5, 16,  7, 16,
       18,  7,  5, 16,  5,  5, 16, 16,  7, 16,  5,  5, 16,  5, 16, 16, 18,
       16,  7, 16, 16, 16,  5, 18, 18,  7, 18,  5,  7,  5,  7,  7, 18,  5,
       16, 18, 18, 16,  7, 16, 16,  7, 16,  7,  7, 16,  7, 16,  7,  7,  7,
       16,  5,  7,  7,  7, 18,  7, 16,  7, 16,  5,  7,  7,  5,  7,  7, 18,
        5, 18, 16,  5,  5, 18, 18, 16, 16, 16,  7, 16,  7,  5, 16, 18, 18,
        7, 16,  5, 18,  7, 16, 16, 16, 18, 18,  7, 18, 16, 16, 16, 16,  5,
       16, 16,  5,  7,  7, 18, 16, 18, 16,  7,  5, 18,  5, 16, 16,  7, 18,
       18, 18,  5,  5,  5,  5,  5, 16,  7, 18,  7, 16, 16,  5,  7, 18,  5,
        7,  5, 16, 16,  5,  7,  7,  5, 18, 18,  7,  7, 16, 18,  5, 18, 16,
       16, 16, 16,  7, 16, 16, 16, 16,  7,  7, 18, 16, 16,  7,  7, 16, 16,
       18,  5,  7,  7,  7, 16,  7,  5,  7, 16,  7,  7,  7, 16, 18, 18,  5,
        5,  7, 16, 18,  7,  7, 16, 18, 18,  7,  7, 16,  5,  5, 18,  5,  5,
       16, 18, 18,  7, 18, 18,  5, 18,  7,  7, 16,  7, 16, 16, 18, 18,  7,
        5,  7,  5, 16, 18,  5, 18, 18, 18,  7, 16, 16, 16,  5,  7, 18,  7,
       16, 18,  5, 18, 18,  7, 18, 18, 16,  7,  7, 16, 18, 16, 16, 18, 18,
       16, 18,  5, 18,  5,  5,  7,  5, 16,  7, 18,  7, 18, 16,  5, 18,  7,
       18,  5,  7, 18,  7,  7, 18,  7, 18, 18,  5,  5, 18,  7, 18,  5, 16,
        7,  5,  7,  7, 16, 16, 18, 18,  7,  5,  5, 16,  5, 18, 16,  7,  7,
        7, 18, 16,  5, 16, 18, 16, 18,  5,  5, 18,  7, 16, 16,  7,  7, 16,
        7, 16, 16, 18, 18,  5,  7, 18, 16, 18,  5, 16,  7,  5, 18, 16,  7,
       18, 18, 16,  7,  7, 16,  5, 16,  5,  5, 18, 16,  5, 18, 18,  7, 18,
       18, 16, 18,  5, 16,  7, 18, 16, 18, 16,  5, 16, 18, 18,  7,  5, 18,
        7,  5, 16, 18, 16,  5, 18, 16, 18,  5, 18,  5, 16,  5,  7,  7, 18,
        7,  7, 18, 16, 16, 16, 18,  7,  5, 16, 18, 18, 18, 18, 18, 16,  7,
       18, 18, 18,  7, 18, 16, 16,  7,  7, 18,  5, 16, 16,  7,  7,  7, 18,
       16, 18,  7, 16,  7, 16,  7,  7,  7, 16,  5, 16,  5,  7,  5,  5,  5,
       18, 16,  7, 16,  7, 16, 16,  5, 16, 18, 16, 16, 16, 16, 18, 16,  7,
        7,  7,  7, 16,  5,  7, 16,  5,  7, 16, 16, 18, 18, 18, 18,  7, 16,
        7,  7, 16,  7,  7,  5, 18, 18, 18,  7,  7,  7,  7,  7, 16,  7, 16,
       16, 16, 18,  5, 16, 16, 18,  7, 18, 18,  5, 16, 18, 18, 16, 18, 16,
       18, 18,  5, 16,  7,  7, 18, 16,  7, 18, 16,  7, 18, 18, 18, 18, 16,
       18,  5,  7, 18,  7, 16,  7,  5,  7, 18, 16, 18, 18, 16,  7, 18, 18,
        5,  5,  7,  7,  5,  5, 16,  7,  5, 16, 18, 16,  7, 18, 16, 18, 16,
        5, 16,  5,  5, 18,  7, 16, 18, 18, 18, 18, 16,  7,  5, 18, 16,  7,
       18,  7, 16,  5,  5,  5, 16, 16, 16,  5, 18, 18, 16, 16,  7,  7,  5,
        5,  5, 16,  5,  5, 16, 16,  7, 18, 16, 18, 16,  5,  7, 18, 16, 16,
       16, 16, 16, 18,  7, 16,  7,  5, 16, 16,  7,  7, 18, 18, 16, 18,  5,
       18, 16,  7,  5,  5,  7, 18,  5,  5,  7, 18, 16, 18,  5,  5, 18, 18,
       18, 18, 16, 16, 16, 16,  7, 18,  7, 18, 16, 16,  7, 16,  5,  7, 18,
       16,  5, 16,  5, 16, 18,  7,  7,  7,  5, 18,  5,  5,  7,  5, 18, 18,
        5, 18,  7,  5, 18,  5, 18,  7, 18,  5, 18, 16, 18,  5,  7, 18,  5,
       16, 16,  5,  7, 16, 16,  5,  5, 16,  7, 18,  5, 18, 16, 16,  7,  7,
        5, 16,  7,  5,  7,  5,  7, 18, 18, 16, 16,  7,  7, 16,  5, 18, 18,
        7, 16,  7, 16,  7,  7, 16,  7, 16,  5, 18, 16,  5,  7,  7,  5, 16,
       16, 18, 18, 16,  5, 18, 18,  5,  7, 16, 16,  5, 18, 16,  5,  5, 18,
       16,  7,  7, 18, 18,  7,  7,  7, 16, 16, 18, 18, 18,  5,  5, 16,  5,
       18, 16, 18, 16, 18, 18, 16, 18,  7,  5, 16, 16, 16,  7, 16,  7, 16,
        7, 16, 18,  7,  5, 18,  7, 18, 16, 18, 18])
```

```
regModel.predict_proba(xtest)
```

```
array([[2.00769054e-09, 5.52878624e-04, 4.38606097e-04, 4.98370093e-01,
        5.00638420e-01],
       [1.00000000e+00, 4.33030594e-13, 5.56636947e-25, 1.88455040e-31,
        2.44239899e-55],
       [5.53617358e-03, 8.83401801e-01, 5.03379605e-03, 1.06023400e-01,
        4.82895681e-06],
       ...,
       [5.12801488e-07, 1.50933961e-02, 2.72345106e-03, 9.35417575e-01,
        4.67650654e-02],
       [1.39736774e-12, 3.60030318e-06, 1.25572979e-05, 4.72024852e-02,
        9.52781357e-01],
       [1.03004111e-13, 5.59218283e-07, 3.19528234e-06, 1.78967852e-02,
        9.82099460e-01]])
```

```
regModel.score(xtest,ytest)
```

```
0.9771547248182763
```

```
# Define a function to filter recommendations for ages 5 to 16
def recommend_for_5_to_16(age_recommendations):
    recommended_ids = []
    for idx, probs in enumerate(age_recommendations):
        # Check if there are enough values to unpack
        if len(probs) >= 2:
            above_18_prob, above_5_prob = probs[0], probs[1]
            # Check if the probability of being above 5 is higher than above 18
            if above_5_prob > above_18_prob:
                # Add the ID to recommended list
                recommended_ids.append(xtest.index[idx])  # Assuming index is the ID
    return recommended_ids



# Calculate class probabilities for each prediction
proba = regModel.predict_proba(xtest)

# Filter recommendations for ages 5 to 16 based on class probabilities
filtered_recommendations = recommend_for_5_to_16(proba)

# Print the recommended IDs for ages 5 to 16
#print("\nRecommended IDs for ages 5 to 16:", filtered_recommendations)
print("\nRecommended IDs for ages 5 to 16:")
for id in filtered_recommendations:
    print(id)
```

```
Recommended IDs for ages 5 to 16:
2383
1124
1784
2202
2536
2751
1573
2521
2720
1108
1576
2078
1676
1585
1836
1781
1565
1738
2968
1235
3158
2002
3118
2400
2221
3074
780
1743
1666
3141
1992
2243
2855
1923
1786
1382
3169
2374
738
2267
1951
2953
2757
1893
821
1327
2340
917
2964
1785
1266
3131
3148
```

```
2839
3121
3062
```