Data Science – Final Project

By: Niv Neuvirth

About the project

- Beer is one of the most popular drinks and is widely consumed all over the world. It is distributed in bottles, cans and commonly available on draught, particularly in pubs and bars.
- The process of making beer is known as brewing and a dedicated building for the making of beer is called a brewhouse. A company that makes beer is called a brewery.
- There are more than 19,000 brewery's worldwide that produce a lot of different beers.
- But in this very big variety of beers, what makes a beer **GREAT**?
  I will try to provide an answer at the end of the project.

Research question:
What defines high rated beers, and is it possible to predict the rating of a specific beer?

- I enjoy drinking beer, and it is one of the reasons I chose this subject, but it is not the main reason. The main reason is that beer has a big influence on our society. People enjoy it in social events, sport events and it binds them together, so it will be very fun and exciting to research such an interesting topic.

- To try and answer the research question, I will collect information about different beers and follow the timeline steps as described in the next slide.

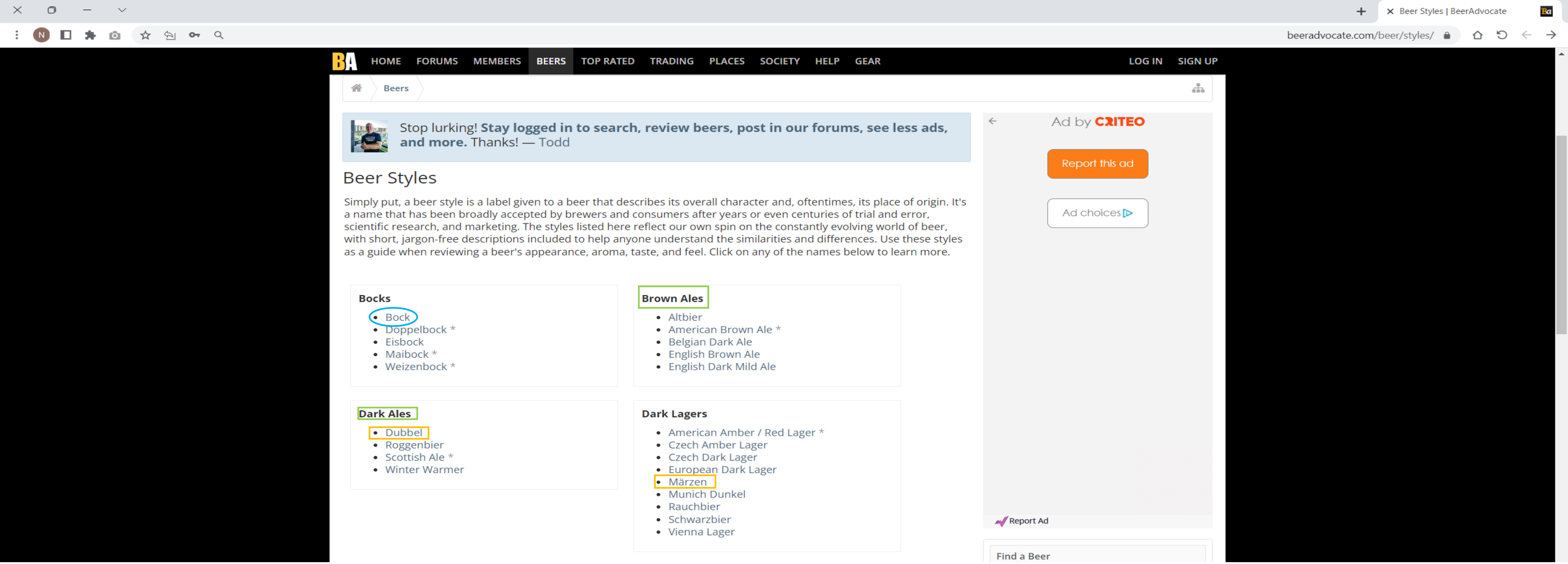
- Before we start it is worth understanding some definitions that will be mentioned along the project:
    - ABV – Alcohol By Volume. A standard measure of how much alcohol is contained in a given volume of alcoholic beverage.
    - IBU – International Bitterness Units. A scale to gauge the level of a beer's bitterness.

Timeline

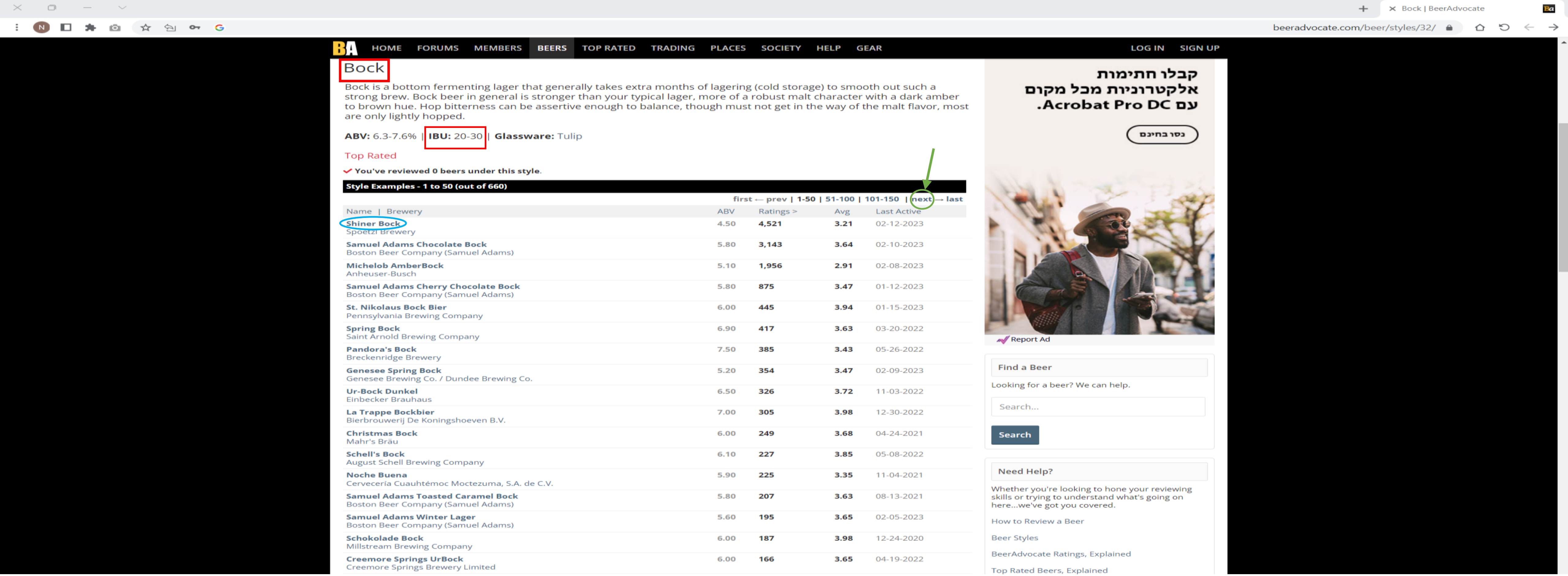
1. Data Scraping
2. Data Handling
3. EDA & Visualization
4. Machine Learning
5. Conclusion

Data Scraping

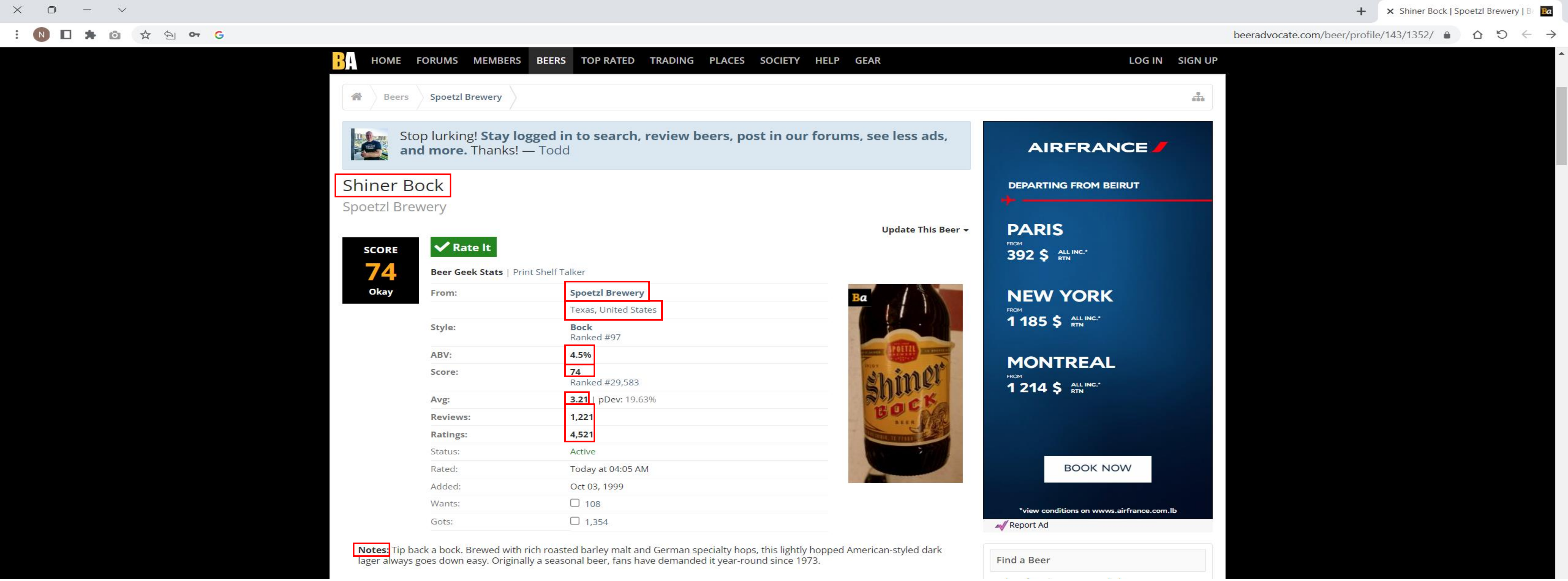
- The data was scraped from one main website: https://www.beeradvocate.com/beer/styles/
- BeerAdvocate is one of the biggest online beer rating sites. It was founded in 1996 by brothers Todd and Jason Alström and is based in Boston, Massachusetts, and Denver, Colorado, USA.
- The web-scraping was done using Selenium and Beautiful Soup python libraries.
- To understand the scraping code and process, the next couple of slides will describe the format of the website.



- This is the main page of the site leading to all the information.
- It contains the 14 main beer styles (Brown Ales, Dark Ales…) and the 120 primary styles(Marzen, Dubbel…).
- Each primary style contains a number of different beers.
- Let’s click on the ‘Bock’ primary style to see an example (on the next slide).



- This page contains all the beers of the 'Bock' style. You have 50 beers maximum a page, and to get to the next page you have to click the 'next' button.
- In this page the scraper gets the data for the following categories: 'primary style', 'minimum IBU', 'maximum IBU' and 'main style'(from the previous page). These data categories are similar to all the beers of this primary style. The scraper scrapes this data once on every first page of a primary style.
- Let's click on the 'Shiner Bock' beer name to see an example (on the next slide).



- This page contains the rest of the data that the scraper will scrape.
- In this page the scraper gets the data for the following categories: ‘name’, ‘brewery’, ‘country’, ‘ABV’, ‘average score’, ‘number of reviews’, ‘number of ratings’ and ‘notes’
- Now that we know the format of the website and the data that we want to scrape, let’s continue to the scraper code (on the next slide)

Scraper code:

- The scraper code is built to run in a for loop and a nested while and 2 for loops.
- The for loop runs in range of 0-119 for the 120 primary beer styles. Each loop scrapes the data of all the beers under the current primary style.
- With the help of the 'Beautiful Soup' python library and the function 'get_beer_data1' we get the data categories described in slide number 7.
- the first for loop creates a list('beer_links') of the 50 or less beer links on the current page as described in slide number 7.
- The next for loop loads the page to each specific beer from the list('beer_links) and with the help of the 'Beautiful Soup' python library and the function 'get_beer_data2' we get the data categories described in slide number 8.
- All the beer data is written to the variable 'curr_beer' and appended to the list 'data'.
- The while loop runs until there is no 'next' button to click on, meaning there are no more beers to scrape under the current primary style. This is done with the help of the 'Selenium' python library. When we exit the 2 for loops, we find the 'next' button element and try to click it. If there is no exception we move to next page of beers, and if there is an exception, we exit the While loop and continue to the next primary style.
- After all the data is stored and we exit the for loop, we create a Data Frame from the collected data and store it in a CSV file. This file will be used for the next sections of the project.

```python
for curr_style in range(0, len(linkToStyles)):
    url = "https://www.beeradvocate.com" + linkToStyles[curr_style]
    time.sleep(0.5)
    try:
        response = requests.get(url, headers=headers)
    except:
        continue
    soup = BeautifulSoup(response.content, "html.parser")
    get_beer_data1(curr_beer, soup, main_styles, curr_style)

    clickable_next_button = True
    while clickable_next_button:
        temp_beer_links = get_links(url, {"class": "mainContent"})
        beer_links = []
        for beer in temp_beer_links:
            if ('/beer/profile/' in beer) and beer.count('/') == 5:
                beer_links.append(beer)

        for beer in range(0, len(beer_links)):
            curr_beer_url = 'https://www.beeradvocate.com' + beer_links[beer]
            time.sleep(0.5)
            try:
                response = requests.get(curr_beer_url, headers=headers)
            except:
                continue
            soup = BeautifulSoup(response.content, "html.parser")
            get_beer_data2(curr_beer, soup)
            data.append(curr_beer.copy())

        driver.get(url)
        try:
            next_button = WebDriverWait(driver, 1).until(
                EC.presence_of_element_located((By.LINK_TEXT, 'next'))
            )
            driver.execute_script('arguments[0].click()', next_button)
            url = driver.current_url
        except:
            clickable_next_button = False

df = pd.DataFrame(data)
df.to_csv('beer_data_final.csv')
```

In [2]:
```python
df = pd.read_csv('crawling_beer_data.csv', header=0, sep=',', thousands=',', encoding='latin-1')
```

In [3]:
```python
df.head()
```

Out[3]:

	Primary_Style	Min_IBU	Max_IBU	Main_Style	Name	Brewery	Country	ABV	Web_Score	Average_Score	Num_of_reviews	Num_of_ratings
0	Bock	20	30	Bocks	Shiner Bock	Spoetzl Brewery	Texas, United States	4.50%	74.0	3.21	1220	3299
1	Bock	20	30	Bocks	Samuel Adams Chocolate Bock	Boston Beer Company (Samuel Adams)	Massachusetts, United States	5.80%	82.0	3.64	1247	1895
2	Bock	20	30	Bocks	Michelob AmberBock	Anheuser-Busch	Missouri, United States	5.20%	68.0	2.91	724	1231
3	Bock	20	30	Bocks	Samuel Adams Cherry Chocolate Bock	Boston Beer Company (Samuel Adams)	Massachusetts, United States	5.80%	79.0	3.47	188	687
4	Bock	20	30	Bocks	St. Nikolaus Bock Bier	Pennsylvania Brewing Company	Pennsylvania, United States	6%	88.0	3.94	225	220

In [4]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 210888 entries, 0 to 210887
Data columns (total 13 columns):
 #   Column          Non-Null Count   Dtype
---  ------          --------------   -----
 0   Primary_Style   210888 non-null  object
 1   Min_IBU         210888 non-null  int64
 2   Max_IBU         210888 non-null  int64
 3   Main_Style      210888 non-null  object
 4   Name            210886 non-null  object
 5   Brewery         210888 non-null  object
 6   Country         210851 non-null  object
 7   ABV             210888 non-null  object
 8   Web_Score       34798 non-null   float64
 9   Average_Score   210888 non-null  float64
 10  Num_of_reviews  210888 non-null  int64
 11  Num_of_ratings  210888 non-null  int64
 12  Notes           210885 non-null  object
dtypes: float64(2), int64(4), object(7)
memory usage: 20.9+ MB
```

In [5]:
```python
df.shape
```

Out[5]:
```
(210888, 13)
```

- This is the Data Frame containing all the data we scraped.
- It has 210,888 rows(each row is a different beer) and 13 columns(each column is a different information about the beer).
- Next section – Data Handling.

Data handling

```
In [6]: df_copy = df.copy()

In [7]: df_copy['ABV']=df_copy['ABV'].str.replace('%','')
        df_copy['ABV']=df_copy['ABV'].replace('Not listed',np.nan)
        df_copy['ABV'] = pd.to_numeric(df_copy['ABV'])

In [8]: df_copy = df_copy.dropna()

In [9]: df_copy = df_copy.drop_duplicates()

In [10]: df_copy.loc[df_copy['Country'].str.contains('United States'), 'Country'] = 'United States'
         df_copy.loc[df_copy['Country'].str.contains('Canada'), 'Country'] = 'Canada'
         df_copy.loc[df_copy['Country'].str.contains('United Kingdom'), 'Country'] = 'United Kingdom'

In [11]: df_copy.shape

Out[11]: (34641, 13)
```

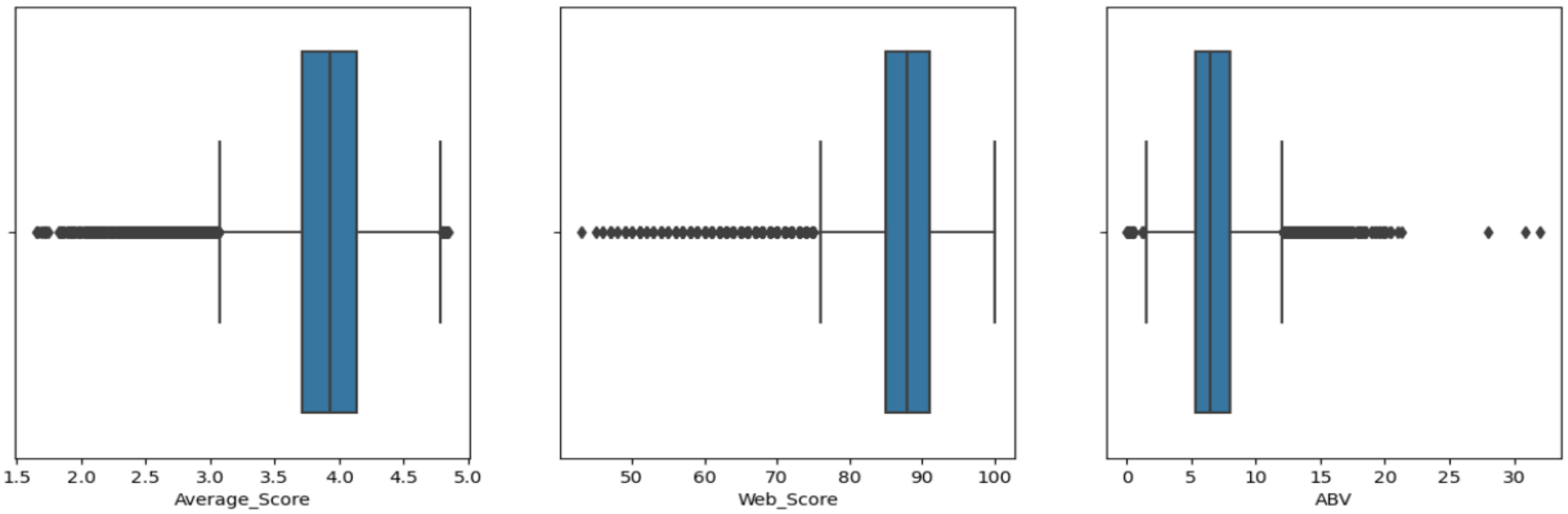
- Created a copy of the data frame to work on.
- Removed the '%' symbol to each beer in the 'ABV' column, replaced values of 'Not listed' to 'NaN' and converted the column to numeric type with the 'to_numeric()' method.
- Removed rows with missing information - 'NaN' values, with the 'dropna()' method.
- Removed duplicative rows with the 'drop_duplicates()' method.
- Some 'Country' data is featured as 'state, country', for example: 'California, United States'. In order to get a more cleaned data I converted it to 'country' only, in the example: 'United States'. This is done for all the beer data is the countries 'United States', 'Canada' and 'United Kingdom'. This is done with the help of the 'loc' method.
- After cleaning the data, the cleaned data frame has 34,641 rows and 13 columns.

Dealing with outliers:

- As we learned during the course, an outlier is not necessarily wrong information.
- In the boxplots below you can see there are outliers, but they are in the normal range.
- ‘Average_Score’ is in range of 0.0 – 5.0, ‘Web_Score’ is in range of 0-100 and ‘ABV’ is in range of 0-40.
- All the outliers are in the normal range. They are ‘True’ outliers.
- We will not delete or change the outliers because they reflect and affect the true outcome.
- All the other outliers in the numeric column are in the normal range as well.

In [14]:
```
f, axes = plt.subplots(1, 3, figsize=(15,5))
sns.boxplot(df_copy.Average_Score, ax=axes[0])
sns.boxplot(df_copy.Web_Score, ax=axes[1])
sns.boxplot(df_copy.ABV, ax=axes[2])
```

Out[14]:
```
<AxesSubplot:xlabel='ABV'>
```



In [18]:
```python
# save cleaned data to csv file
df_copy.to_csv('cleaned_beer_data.csv', index=False)
```

- After we are done with handling the data, we will save the cleaned data to a new csv file, and continue to work on it in the next section.
- Next section – EDA & Visualization.

EDA & visualization

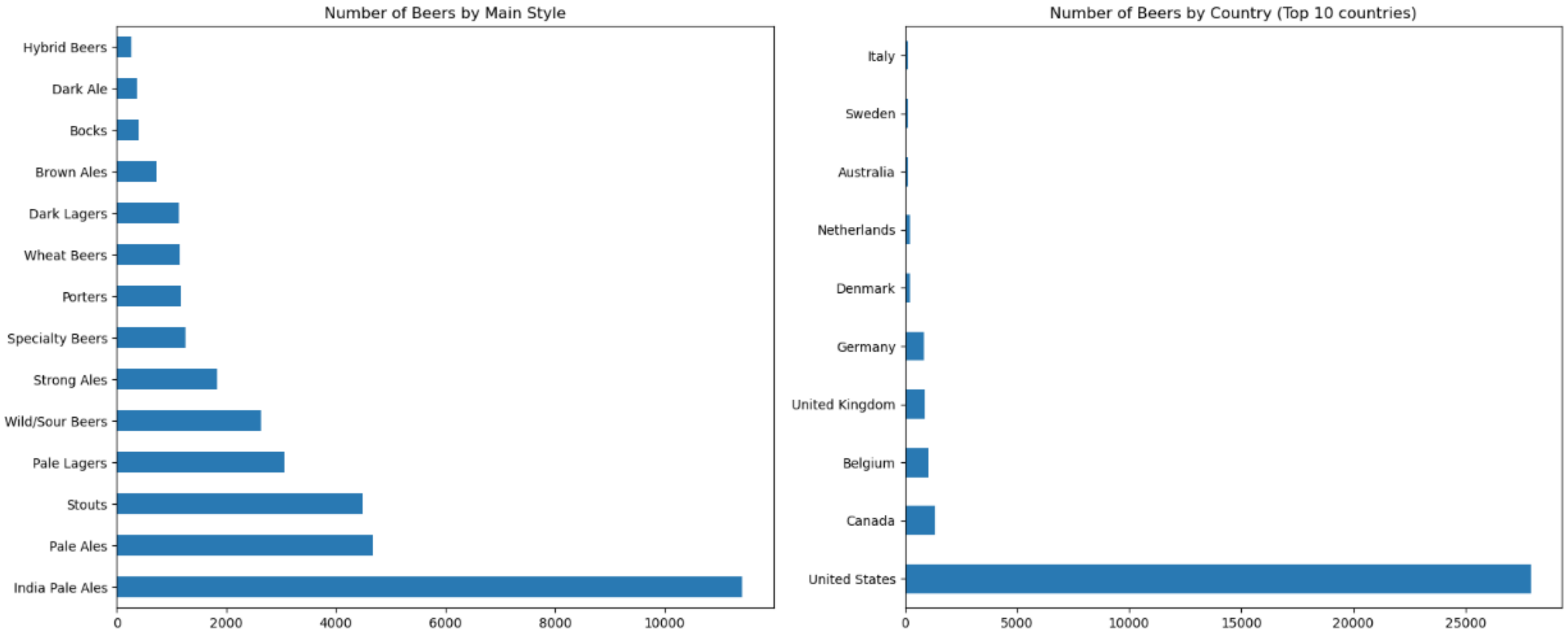
- Create a new data frame from the cleaned data csv file

```
In [19]: cdf = pd.read_csv('cleaned_beer_data.csv', header=0, sep=',', thousands=',', encoding='latin-1')
```

- Next we will visualize the data in different ways and make a conclusion.

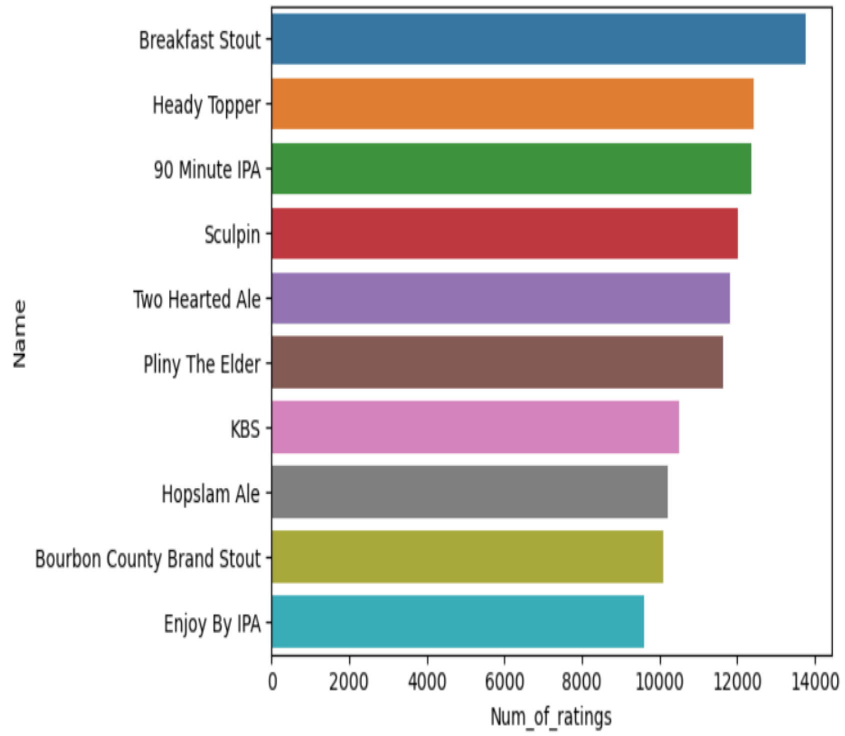
In [21]:
```python
f, axes = plt.subplots(1, 2, figsize=(20,8))
cdf['Main_Style'].value_counts().plot(kind='barh', ax=axes[0], title='Number of Beers by Main Style' )
cdf['Country'].value_counts()[cdf['Country'].value_counts() > 125].plot(kind='barh', ax=axes[1], title='Number of Beers by Countr
```

Out[21]: <AxesSubplot:title={'center':'Number of Beers by Country (Top 10 countries)'}>



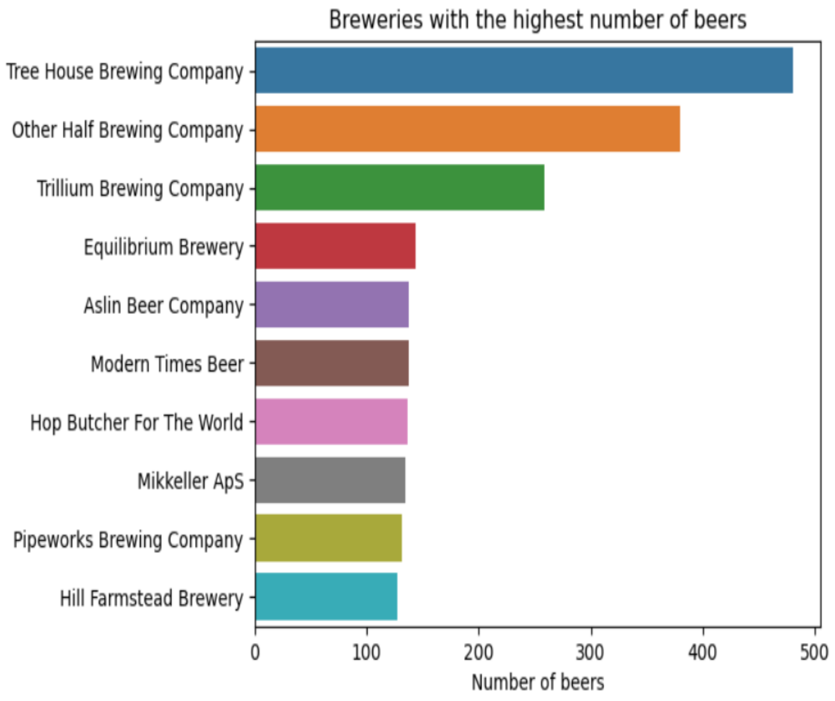
```
In [24]: # highest rated beers
popular_beers = cdf.nlargest(10, ['Num_of_ratings']).set_index('Name')['Num_of_ratings']
sns.barplot(popular_beers, popular_beers.index)

Out[24]: <AxesSubplot:xlabel='Num_of_ratings', ylabel='Name'>
```



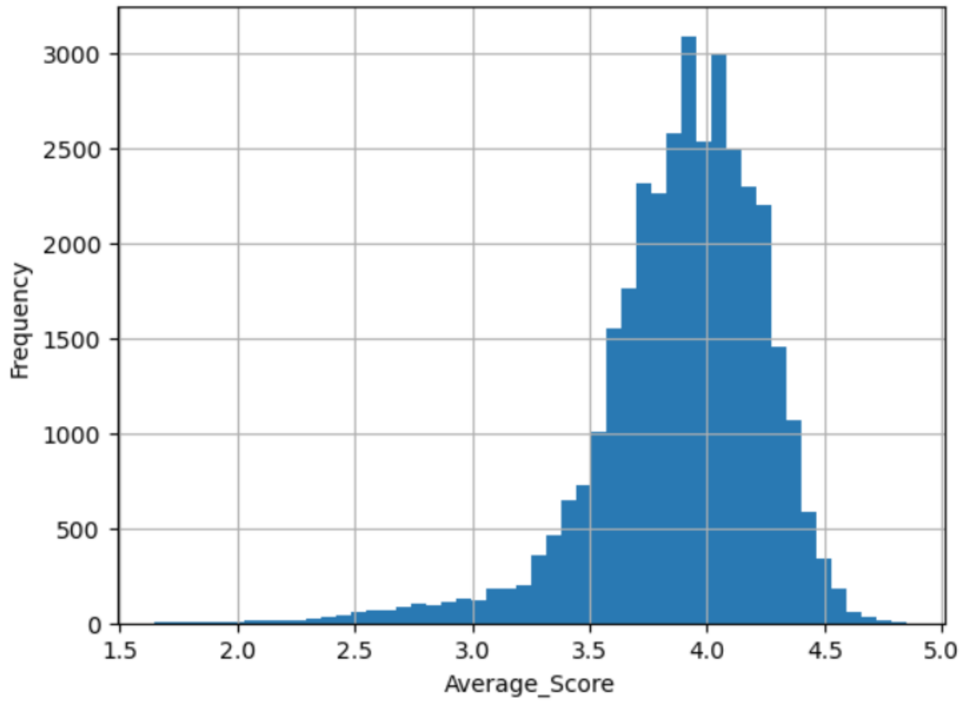

```
In [39]: # Breweries with the highest number of beers
top_breweries = cdf['Brewery'].value_counts().head(10)
sns.barplot(top_breweries, top_breweries.index)
plt.title('Breweries with the highest number of beers')
plt.xlabel('Number of beers')

Out[39]: Text(0.5, 0, 'Number of beers')
```



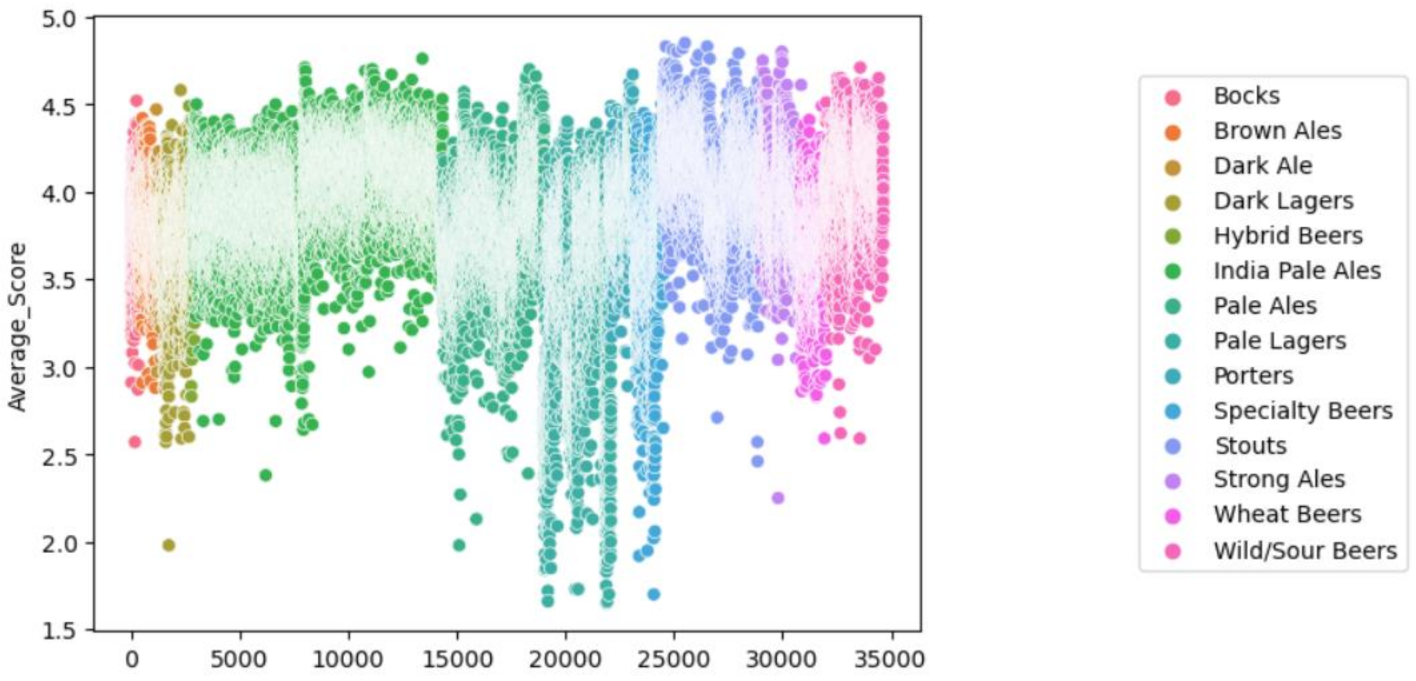
In [22]:
```
cdf['Average_Score'].hist(bins=50)
plt.xlabel('Average_Score')
plt.ylabel('Frequency')
```

Out[22]: Text(0, 0.5, 'Frequency')



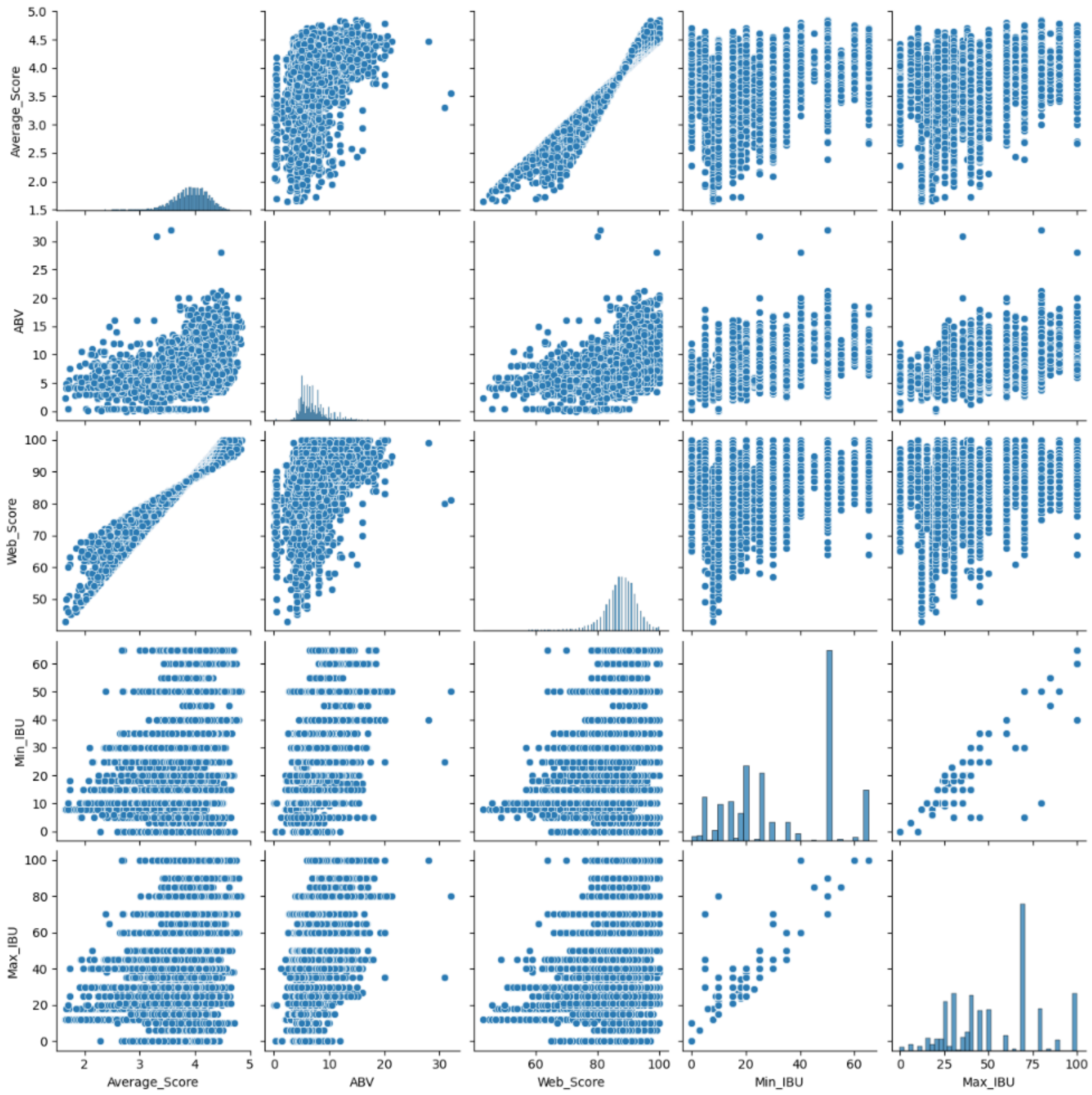
In [72]:
```
sns.scatterplot(data=cdf, x=cdf.index, y='Average_Score', hue = 'Main_Style')
plt.legend(loc='center left', bbox_to_anchor=(1.25, 0.5), ncol=1)
```

Out[72]: <matplotlib.legend.Legend at 0x1d51ba4d310>



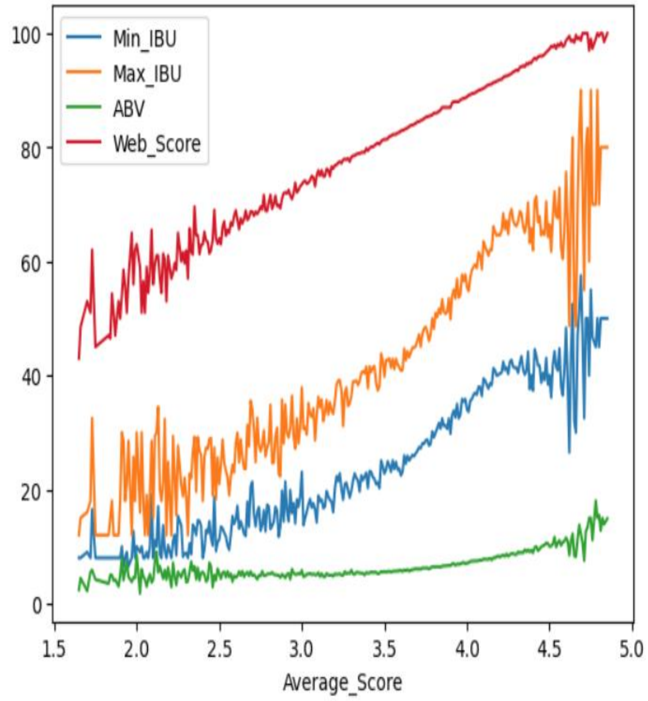
In [40]:
```
sns.pairplot(cdf[['Average_Score', 'ABV','Web_Score', 'Min_IBU', 'Max_IBU']])
```

Out[40]: <seaborn.axisgrid.PairGrid at 0x128b125fa90>



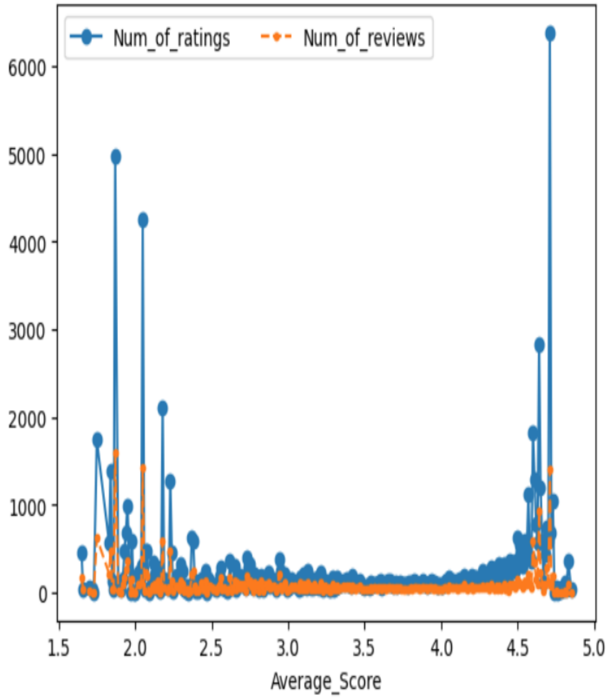
```
In [33]: p =cdf.groupby(['Average_Score']).mean()[['Min_IBU','Max_IBU','ABV','Web_Score']]
         p.plot()
```

Out[33]: <AxesSubplot:xlabel='Average_Score'>




```
In [34]: t =cdf.groupby(['Average_Score']).mean()[['Num_of_ratings', 'Num_of_reviews']]
         t.plot(style=['o-','.--']).legend(loc='upper left', ncol=2)
```

Out[34]: <matplotlib.legend.Legend at 0x128b0f13bb0>



Text Analysis:

most common words from the ‘Notes’ column.

It won’t be used to try to predict the result but its still nice and cool to see.

```
In [52]: wc = WordCloud(background_color="white", max_words=100, mask=transformed_beer_mask,stopwords=stopwords, contour_width=3, contour_
         wc=wc.generate(text)
         plt.figure(figsize=[20,10])
         plt.imshow(wc, interpolation='bilinear')
         plt.axis("off")
         plt.show()
```

EDA & Visualization conclusion:

- More than 75% of the beers are from the ‘United States’.
- The main style with the most beers is ‘India Pale Ales.
- The ‘Tree House Brewing Company’ has the largest number of beers in the dataset(over 450 beers).
- Most beers got the average score between 3.5 – 4.5.
- The ‘Average_Score’ and ‘Web_Score’ columns have a linear relationship.
- Beers with the highest average score and lowest average score got more user rating and reviews, but it seems there is no connection between the number of ratings and reviews to the average score.
- If we look at the mean of the ‘Min_IBU’ and ‘Max_IBU’ columns it seems that higher rated beers (beers with a higher average score) have a higher IBU.
- If we look at the mean of the ‘ABV’ column it seems higher rated beers have a higher ABV, but by a small margin.
- It is hard to say, bus it seems that higher IBU and ABV are more common in higher rated beers.
- Next section – Machine Learning.

Machine learning

- We would like to predict the rating of a beer, that’s why our target column would be the ‘Average_Score’.

- The column contains real values, and that’s why we will use the Regression supervised learning model.

- We will use the Linear Regression algorithm, and by the prediction result decide if a different algorithm is needed.

- Before we start we need to make some adjustments to the data for it to work best for the machine learning model.
- We will copy the cleaned data frame to a new one, the ‘mldf’ (machine learning data frame).
- Data Encoding: with the help of the LabelEncoder() method we convert categorical columns to numeric.
- Drop the ‘Notes’ column. It has no use for the next step.
- Now we can proceed to the next step – splitting the data and training a model.

In [66]:
```
mldf = cdf.copy()
```

In [67]:
```
le = preprocessing.LabelEncoder()
mldf['Brewery'] = le.fit_transform(mldf['Brewery'])
mldf['Name'] = le.fit_transform(mldf['Name'])
mldf['Main_Style'] = le.fit_transform(mldf['Main_Style'])
mldf['Primary_Style'] = le.fit_transform(mldf['Primary_Style'])
mldf['Country'] = le.fit_transform(mldf['Country'])
```

In [68]:
```
mldf=mldf.drop('Notes', axis=1)
```

In [69]:
```
mldf
```

Out[69]:

	Primary_Style	Min_IBU	Max_IBU	Main_Style	Name	Brewery	Country	ABV	Web_Score	Average_Score	Num_of_reviews	Num_of_ratings
0	27	20	30	0	25662	3739	105	4.5	74.0	3.21	1220	3299
1	27	20	30	0	24883	611	105	5.8	82.0	3.64	1247	1895
2	27	20	30	0	18614	179	105	5.2	68.0	2.91	724	1231
3	27	20	30	0	24881	611	105	5.8	79.0	3.47	188	687
4	27	20	30	0	27099	3138	105	6.0	88.0	3.94	225	220
...	...	...	...	...	...	...	...	...	...	...	...	...
34636	117	5	30	13	2231	3571	105	6.6	92.0	4.38	4	6
34637	117	5	30	13	20523	253	105	6.0	89.0	4.02	5	5
34638	117	5	30	13	3739	3128	105	4.6	87.0	3.84	0	10
34639	117	5	30	13	17657	982	80	4.2	85.0	3.70	4	6
34640	117	5	30	13	30879	4499	105	5.0	91.0	4.31	1	9

34641 rows × 12 columns

- We create a new data frame for the feature vectors without the 'Average_Score' column('X'), and a series containing the 'Average_Score' only('y').
- Next, with the help of the 'train_test_split' method we split 80% of the data to the training set and 20% of the data to the test set.
- After we split the data we can train our algorithm along with the training set data and the 'fit()' method.

In [70]:
```
#dataframe for the feature vectors (without the target column) and a series containing the corresponding target values
X_columns = mldf.columns[mldf.columns!='Average_Score']
X = mldf[X_columns]
y = mldf['Average_Score']
```

In [71]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state = 42)
```

In [72]:
```
#train model
trained_model = LinearRegression()
trained_model.fit(X_train,y_train)
```

Out[72]:
```
LinearRegression()
```

- Next step is to use the test set data to check how accurately our algorithm predicts the ‘Average_Score’.
- We use the trained model and the test set data to make a prediction.

```
In [103]: predictions = trained_model.predict(X_test)
```

- Now that we have our model predictions, we evaluate them.

- Evaluation of the predictions:
  - Calculate the R-Squared.
  - Compare the actual output values for ‘X_test’ with the predicted values.

In [104]:
```
r2_score(y_test, predictions)
```

Out[104]:
```
0.9599373687626908
```

In [105]:
```
pred = pd.DataFrame({'Actual': y_test.tolist(), 'Predicted': predictions.tolist()})
pred.head(15)
```

Out[105]:

	Actual	Predicted
0	3.70	3.692114
1	4.09	4.055656
2	3.89	3.869424
3	4.08	4.136890
4	3.84	3.834587
5	3.59	3.670601
6	4.13	4.112080
7	3.94	3.911009
8	4.00	3.960199
9	3.97	3.967338
10	3.59	3.560655
11	4.07	4.094409
12	3.67	3.759140
13	3.70	3.743076
14	4.01	4.012277

- The R-Squared value for the model is 0.9599 which is very good but may be a sign for overfitting.
- I believe the reason for the high R-Squared value is the linear relationship the target column has with the 'Web_Score' column as we saw in the EDA part.
- Lets drop the 'Web_Score' column from the 'X' data frame, train a new model and check out the result.

In [78]:
```
X1 = X.drop('Web_Score', axis=1)
```

In [79]:
```
X1
```

Out[79]:

	Primary_Style	Min_IBU	Max_IBU	Main_Style	Name	Brewery	Country	ABV	Num_of_reviews	Num_of_ratings
0	27	20	30	0	25662	3739	105	4.5	1220	3299
1	27	20	30	0	24883	611	105	5.8	1247	1895
2	27	20	30	0	18614	179	105	5.2	724	1231
3	27	20	30	0	24881	611	105	5.8	188	687
4	27	20	30	0	27099	3138	105	6.0	225	220
...	...	...	...	...	...	...	...	...	...	...
34636	117	5	30	13	2231	3571	105	6.6	4	6
34637	117	5	30	13	20523	253	105	6.0	5	5
34638	117	5	30	13	3739	3128	105	4.6	0	10
34639	117	5	30	13	17657	982	80	4.2	4	6
34640	117	5	30	13	30879	4499	105	5.0	1	9

34641 rows × 10 columns

In [80]:
```
X_train1, X_test1, y_train1, y_test1 = train_test_split(X1, y, test_size=0.2, random_state = 42)
```

In [81]:
```
#train model
trained_model = LinearRegression()
trained_model.fit(X_train1,y_train1)
```

Out[81]:
```
LinearRegression()
```

In [82]:
```
predictions1 = trained_model.predict(X_test1)
```

In [83]:
```
r2_score(y_test1, predictions1)
```

Out[83]:
```
0.3461487700667748
```

- The R-Squared value for the new model is 0.3461. dramatically lower than the first model.
- As I suspected the 'Web_Score' column had to much of an impact on the result of the model.
- This makes sense since the correlation between the 2 column is very high.

```
In [133]: mldf['Average_Score'].corr(mldf['Web_Score'])

Out[133]: 0.9787906867289128
```

- We don't want our model to relay on one column and specifically the 'Web_Score' column. New beers we would like to test won't neccesraly have a web score but will have the other columns data.
- Let's try to improve the current model:

```
In [223]: X2 = X1.copy()

In [224]: X2['Style'] = X2['Primary_Style']+X2['Main_Style']
          X2['Reviews+Ratings'] = X2['Num_of_reviews']-X2['Num_of_ratings']
          X2 = X2.drop(['Main_Style', 'Primary_Style', 'Num_of_reviews', 'Num_of_ratings'], axis=1)

In [225]: X_train2, X_test2, y_train2, y_test2 = train_test_split(X2, y, test_size=0.2, random_state = 42)

In [226]: #train model
          trained_model = LinearRegression()
          trained_model.fit(X_train2,y_train2)

Out[226]: LinearRegression()

In [227]: predictions2 = trained_model.predict(X_test2)

In [228]: r2_score(y_test2, predictions2)

Out[228]: 0.3335188894844754
```

- The R-Squared value we got was lower. Meaning we can not improve the model with the data we have.
- Lets try a different Regression model – the Random Forest Regressor algorithm.

In [119]:
```
X_train3, X_test3, y_train3, y_test3 = train_test_split(X2, y, test_size=0.2, random_state = 42)
```

In [120]:
```
rf = RandomForestRegressor(n_estimators=100, random_state=42)
rf.fit(X_train3,y_train3)
```

Out[120]:
```
RandomForestRegressor(random_state=42)
```

In [121]:
```
predictions3 = rf.predict(X_test3)
```

In [123]:
```
r2_score(y_test3, predictions3)
```

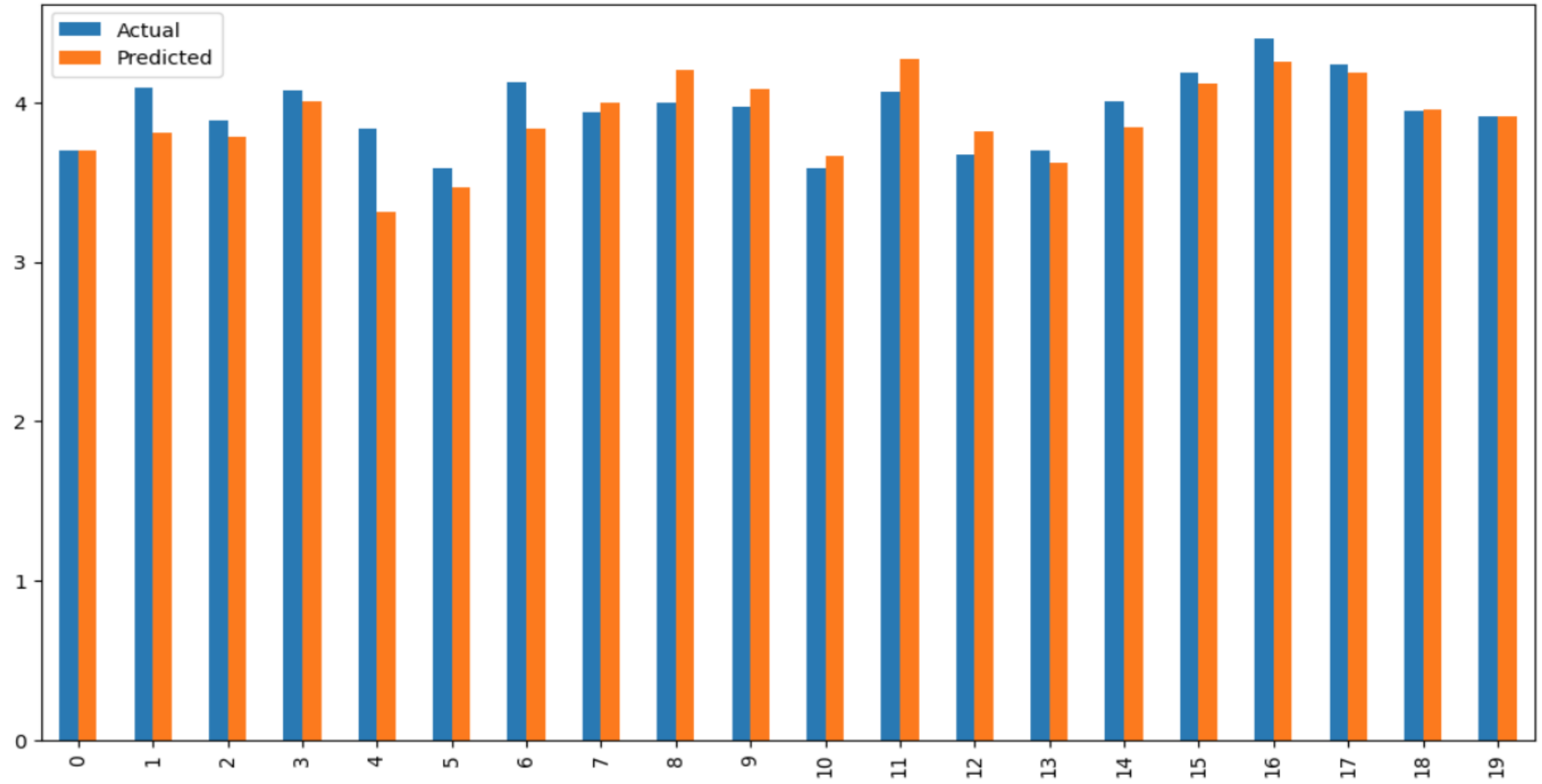
Out[123]:
```
0.6070186437011702
```

In [122]:
```
pred3 = pd.DataFrame({'Actual': y_test3.tolist(), 'Predicted': predictions3.tolist()})
pred3.head(15)
```

Out[122]:

	Actual	Predicted
0	3.70	3.7012
1	4.09	3.8095
2	3.89	3.7859
3	4.08	4.0048
4	3.84	3.3119
5	3.59	3.4643
6	4.13	3.8323
7	3.94	3.9991
8	4.00	4.2070
9	3.97	4.0873
10	3.59	3.6678
11	4.07	4.2701
12	3.67	3.8214
13	3.70	3.6254
14	4.01	3.8421

```
In [229]: pred3.head(20).plot(kind='bar', figsize=(13, 7))
Out[229]: <AxesSubplot:>
```



- The R-Squared value for the Random Forest Regressor model is 0.6070, making it a better model than the Linear Regression one.

Conclusion

- It is unclear what defines a great beer. On average, higher rated beers have a higher ABV percentage and a higher IBU.

- The Style of the beer doesn’t affect its rating, meaning there isn’t a specific style for higher rated beers.

- It is possible to predict the rating of a beer:
  - The best ML model to predict the rating of a beer is Random Forest Regressor.
  - It predicts the rating of a beer correctly by 60%, which is not bad considering the first model was 34%.

Hope you enjoyed, cheers!

Citations

- Campus site – introduction to data science course
- BeerAdvocate website
- https://en.wikipedia.org/wiki/Beer#Brewing
- https://www.carlingpartnership.com/insights/who-owns-who-in-the-world-of-beer/#:~:text=A%20survey%20released%20by%20Alltech,companies%20worldwide%2C%20across%20208%20countries
- https://en.wikipedia.org/wiki/Beer_rating
- https://www.datacamp.com/tutorial/wordcloud-python
- https://www.keboola.com/blog/random-forest-regression
- https://datavizpyr.com/how-to-place-legend-outside-the-plot-with-seaborn-in-python/
- https://www.youtube.com/watch?v=Xjv1sY630Uc&list=PLzMcBGfZo4-n40rB1XaJ0ak1bemvlqumQ