1. Main Research Topic

In this project, I analyze Age-of-Acquisition (AoA) statistics for roughly 30,000 English and 30,000 Dutch words. Starting with some more basic queries, I unveil key characteristics of each dataset. I then group words by their first letter and compare these groupings across both languages. This comparative analysis aims to highlight similarities and differences in vocabulary learning for English and Dutch speakers.

2. Used Data Sources

A general link providing more information on this data can be found here. To directly download each excel file, clink on the links below:

1. (https://web.archive.org/web/20230208135926/http://crr.ugent.be/papers/AoA_ratings_Kuperm
   (Please note that this data has been saved as an excel file and renamed 'Age_of_Acquisition_English.xlsx' in this notebook.)

2. (https://web.archive.org/web/20230208125948/http://crr.ugent.be/papers/Overzicht%20AoA%2
   (Please note that this data has been saved as an excel file and renamed 'Age_of_Acquisition_Dutch.xlsx' in this notebook.)

For the academic articles produced from this research, please see the links below:

1. Kuperman, V., Stadthagen-Gonzalez, H., & Brysbaert, M. (2012). Age-of-Acquisition Ratings for 30,000 English Words. *Behav Res*, 44, 978–990
2. Brysbaert, M., Stevens, M., De Deyne, S., Voorspoels, W., & Storms, G. (2014). Norms of Age of Acquisition and Concreteness for 30,000 Dutch Words. *Acta Psychologica*, 150, 80-84

3. Methodology

Preparing and Organizing the Data: After downloading and saving each excel file to my Jupyter environment, I imported the following three libraries: 1.) pandas 2.) seaborn and 3.) matplotlib. Pandas was then used to open each file for reading. To confirm the information looked correct, I used the .head() and .describe() methods to quickly see the uploaded information.

Analyzing the Data: Once my data was correctly uploaded, I proceeded to analyze the data according to three different sections:

- Section 1: Analysis of df_1
- Section 2: Analysis of df_2
- Section 3: Comparing AoA's through Visualizations

For the first two sections, I zoom in on the individual datasets by performing both tabular data analysis and data visualization. The insights gained from each dataset were then compared against each other in the final section.

Limitations: As someone who doesn't speak Dutch, my ability to analyze the Dutch dataset is particularly limited. While Brysbaert et al. offer translations that aided my comprehension of certain words, this only provided a limited insight into the broader context of Dutch language learning. However, by categorizing words according to their first letters, I purposefully distance myself from scrutinizing individual words too closely. Exploring specific words within the groups I've defined could be a more intriguing avenue for future investigation.

4. Preparing and Organizing the Data

Step 1: Importing libraries

In [1]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

Step 2: Opening and reading data files

In [2]:
```
#Saving English datset under variable "df_1".
df_1 = pd.read_excel('Age_of_Acquisition_English.xlsx')
```

In [3]:
```
#Saving Dutch datset under variable "df_2".
df_2 = pd.read_excel('Age_of_Acquisition_Dutch.xlsx')
```

Step 3: Head and describe data to confirm it is correctly uploaded

DataFrame 1: *Age of Acquisition (English)*

In [4]:
```
df_1.head()
```

Out[4]:

	Word	OccurTotal	OccurNum	Freq_pm	Rating.Mean	Rating.SD	Dunno
0	a	22	22	20415.274510	2.893384	1.21	1.000000
1	aardvark	18	18	0.411765	9.890000	3.66	1.000000
2	abacus	20	13	0.235294	8.690000	3.77	0.650000
3	abalone	18	13	0.509804	12.230000	3.54	0.722222
4	abandon	19	19	8.098039	8.320000	2.75	1.000000

In [5]:
```
df_1.describe()
```

Out[5]:

	OccurTotal	OccurNum	Freq_pm	Rating.Mean	Rating.SD	Dunno
count	31124.000000	31124.000000	30387.000000	31105.000000	31046.000000	31124.000000
mean	22.918519	20.439115	25.544835	11.000033	3.187698	0.873487
std	85.072469	84.271333	499.087837	3.044179	0.904152	0.198555
min	15.000000	0.000000	0.019608	1.580000	0.000000	0.000000
25%	18.000000	16.000000	0.078431	8.940000	2.570000	0.833333
50%	19.000000	18.000000	0.352941	11.170000	3.110000	0.952381
75%	20.000000	19.000000	1.666667	13.200000	3.730000	1.000000
max	1939.000000	1934.000000	41857.117647	25.000000	9.900000	1.000000

Results: The .head() and .describe() queries for df_1 already reveal important information. We can see a number of interesting columns, including Rating.Mean which shows the average AoA for each word. Through .describe(), we also see that the average AoA (or Rating.Mean) for the entire dataset is approximately 11.00. This will be particularly important in my analysis later, after I group words based on their first letter.

DataFrame 2: *Age of Acquisition (Dutch)*

In [6]:
```
df_2.head()
```

Out[6]:

	Word	Translation	Ghyselinck2000	Unnamed: 3	Ghyselinck2003	Unnamed: 5	Moors2013	Unname
0	NaN	NaN	AoA	%Known	AoA	%Known	AoA	%Know
1	mama	NaN	2.25	100	2	100	2.4	1(
2	ja	NaN	NaN	NaN	NaN	NaN	NaN	Na
3	mamma	NaN	1.915094	98.148148	NaN	NaN	NaN	Na
4	papa	NaN	2.138889	100	2.2	100	2.5875	1(

Results: While a single title is sometimes used for two columns in excel, this format doesn't directly translate in Python, making messy column headers for df_2. For instance, "Ghyselinck2003" becomes the header for AoA data and the column directly adjacent to it becomes "Unnamed:5". Moreover, record #1 does not belong to the dataset and should actually be included in the header. Therefore, most of these headers need to be renamed and the first record dropped.

In [7]:
```
#Renaming the columns
df_2.rename(columns={'Ghyselinck2000': 'Ghyselinck_2000_AoA', 'Unnamed: 3': 'Ghyselinc
                     'Ghyselinck2003': 'Ghyselinck_2003_AoA', 'Unnamed: 5': 'Ghyselinck
                     'Moors2013': 'Moors_2013_AoA', 'Unnamed: 7': 'Moors_2013_%_Known',
                     'Brysbaert2014': 'Brysbaert_2014_AoA', 'Unnamed: 9': 'Brysbaert_20
                     'Average': 'Average_AoA', 'Unnamed: 11':'Average_%_Known',
                     }, inplace=True)
```

```
#Dropping Record #1
df_2.drop(0, inplace=True)
df_2.reset_index(drop=True, inplace=True)
```

In [8]:
```
df_2.head()
```

Out[8]:

	Word	Translation	Ghyselinck_2000_AoA	Ghyselinck_2000_%_Known	Ghyselinck_2003_AoA	Ghyse
0	mama	NaN	2.25	100	2	
1	ja	NaN	NaN	NaN	NaN	
2	mamma	NaN	1.915094	98.148148	NaN	
3	papa	NaN	2.138889	100	2.2	
4	nee	NaN	NaN	NaN	NaN	

In [9]:
```
df_2.describe()
```

Out[9]:

	Word	Translation	Ghyselinck_2000_AoA	Ghyselinck_2000_%_Known	Ghyselinck_2003_AoA	G
count	31177	5099	2816	2816	2333.0	
unique	31177	4351	2120	262	136.0	
top	mama	**********	13	100	9.5	
freq	1	68	11	1386	51.0	

Results: df_2 looks much cleaner after renaming these columns. Though most of my research will pertain to Average AoA's, it is still important to start my analysis with clean and structured data.

5. Analyzing the Data

Section 1: Analysis of df_1

To start my analysis of df_1, I perform a few simple queries that help frame the data. This involves identifying the highest and lowest AoAs, as well as calculating the average word length. Following these queries, I start developing unique visualizations. For these visualizations, I focus on grouping words according to their first letters and showing the number of words that fall within each grouping. These groupings are then examined within the context of average AoA data.

Query #1: *Highest AoA*

In [10]:
```
df_1.sort_values('Rating.Mean', ascending = False).head(10)
```

Out[10]:

	Word	OccurTotal	OccurNum	Freq_pm	Rating.Mean	Rating.SD	Dunno
8762	eisteddfod	21	1	NaN	25.00	NaN	0.047619
1344	architrave	19	1	0.039216	21.00	NaN	0.052632
3734	calceolaria	19	2	0.019608	21.00	5.66	0.105263
19898	penury	18	5	0.019608	20.60	1.52	0.277778
19000	oubliette	19	4	0.098039	20.50	4.43	0.210526
15091	kendo	18	2	0.372549	20.50	6.36	0.111111
23974	schottische	18	4	0.039216	20.25	3.86	0.222222
27579	thrombocytopenia	20	6	0.019608	20.17	3.97	0.300000
19803	pederasty	19	2	0.019608	20.00	4.24	0.105263
11170	furfural	18	1	0.117647	20.00	NaN	0.055556

Query #2: *Lowest AoA*

In [11]:
```
df_1.sort_values('Rating.Mean', ascending = True).head(10)
```

Out[11]:

	Word	OccurTotal	OccurNum	Freq_pm	Rating.Mean	Rating.SD	Dunno
17429	momma	19	19	8.078431	1.580000	0.69	1.000000
16407	mama	19	19	103.705882	1.890000	1.29	1.000000
2069	bantling	21	1	NaN	2.000000	NaN	0.047619
17424	mom	18	18	430.392157	2.220000	1.22	1.000000
20924	potty	18	18	1.686275	2.280000	0.83	1.000000
31031	yes	21	21	1996.764706	2.310598	1.68	1.000000
30336	water	19	19	225.058824	2.370000	0.76	1.000000
19001	ouch	21	21	10.960784	2.417168	1.34	1.000000
30483	wet	19	19	39.215686	2.470000	1.35	1.000000
25738	spoon	18	18	7.607843	2.500000	0.79	1.000000

Results: Queries 1 and 2 reveal an AoA range of 1.58-25.00 for df_1. It would be interesting to explore the specific places these AoA statistics were acquired from. A quick google search on the word with the highest AoA - "eisteddfod" - reveals that it has Welsh origins, perhaps suggesting at least some of this data comes from English-speakers in the UK. AoA's might differ between individual English-speaking countries, making future analysis on a per country basis warranted. Additionally, we see how "momma", "mama" , and "mom" are all included as some of the earliest words acquirable for English speakers. In future analyses, we could group synonyms and see how many occur within each lanugage. Or, if we had more specific information on where this data comes from, we could investigate whether instances of "momma", "mama" , and "mom" change based on a given country's regulations surrounding parental paid leave (i.e., how

much time mothers/fathers are given off of work to help the child when they are first born). For now, I will just note it and move on.

Queries #3 and #4 *Average Word Length*

In [12]:
```
df_1.sort_values(by='Word', key=lambda x: x.str.len(), ascending=False).head(10)
```

Out[12]:

	Word	OccurTotal	OccurNum	Freq_pm	Rating.Mean	Rating.SD	Dunno
7058	deinstitutionalization	19	19	0.019608	16.05	2.93	1.000000
8806	electroencephalograph	19	13	0.019608	15.85	2.44	0.684211
13374	hypercholesterolemia	22	7	0.019608	17.00	4.16	0.318182
14386	institutionalization	20	18	0.019608	13.44	2.31	0.900000
5330	compartmentalization	19	19	0.039216	13.11	3.73	1.000000
6108	counterrevolutionary	18	15	0.117647	14.67	2.72	0.833333
28785	undercapitalization	19	10	0.019608	15.30	5.03	0.526316
26221	straightforwardness	18	18	0.019608	11.39	2.25	1.000000
6090	counterintelligence	20	20	0.372549	13.45	2.96	1.000000
17727	multidimensionality	18	17	0.019608	14.76	3.31	0.944444

In [13]:
```
#Augmenting df_1 with a new column titled "WordLength"
#.apply() taken from ChatGPT
df_1['WordLength'] = df_1['Word'].apply(lambda x: len(str(x)))

#Calculating the longest, shortest, and average WordLength

#Longest
max_df_1_word_length = df_1['WordLength'].max()

#Shortest
min_df_1_word_length = df_1['WordLength'].min()

#Average
avg_df_1_word_length = round(df_1['WordLength'].mean(), 2)

#Print
print(f'The longest word in df_1 is {max_df_1_word_length} letters.')
print(f'The shortest word in df_1 is {min_df_1_word_length} letter.')
print(f'The average word length for df_1 is {avg_df_1_word_length} letters.')
```

```
The longest word in df_1 is 22 letters.
The shortest word in df_1 is 1 letter.
The average word length for df_1 is 8.05 letters.
```

Results: Words in df_1 range between lengths of 1 and 22 letters, though the average is 8.05. Perhaps this data shares a high correlation with the AoA data - in other words, maybe longer words only become acquirable as English speakers get older. Let's perform a correlation query to check.

Query #5: *Correlation Between WordLength and AoA*

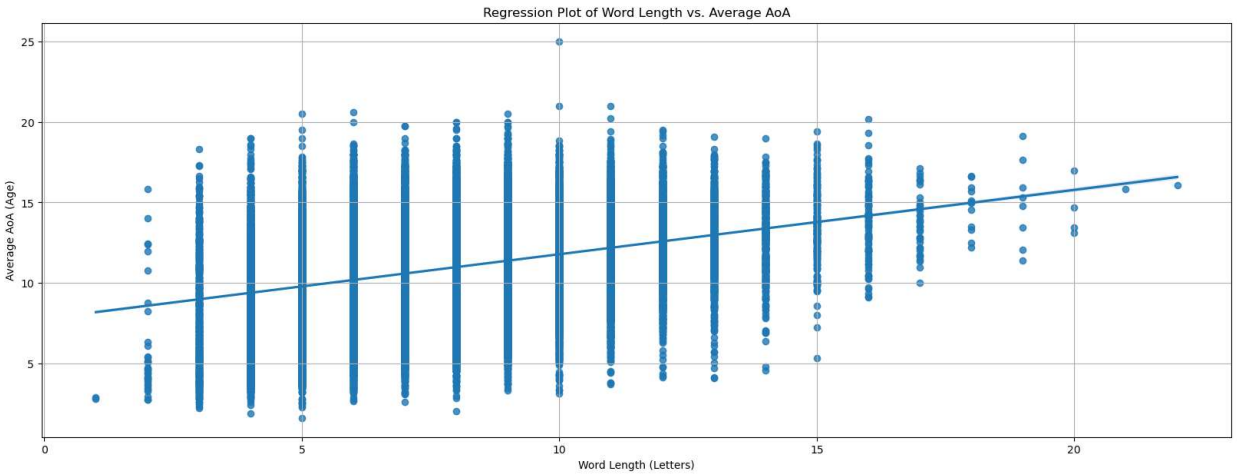
In [14]:
```
#Calculating the correlation between WordLength and AoA
Word_Lenth_to_AoA_correlation_coefficient = df_1['WordLength'].corr(df_1['Rating.Mean'

#Print
print(f'The Correlation Coefficient between word length and average AoA for df_1 is {W
```

```
The Correlation Coefficient between word length and average AoA for df_1 is 0.3395021
79821234.
```

Results: The correlation is rather low (only 34%), suggesting that the two don't directly work hand in hand. Let's visualize this data on a Regression Plot just to see it more clearly.

Query #6: *Regression Plot of WordLength vs. Average AoA*

In [15]:
```
#Setting the regression plot size
plt.figure(figsize=(20, 7))

#Creating regression plot for Average AoA and WordLength
sns.regplot(x=df_1['WordLength'], y=df_1['Rating.Mean'])
plt.title('Regression Plot of Word Length vs. Average AoA')
plt.xlabel('Word Length (Letters)')
plt.ylabel('Average AoA (Age)')
plt.grid(True)

#Plot
plt.show()
```



Results: This helps confirm that words with the highest AoA's aren't necessarily the longest in terms of letters. For instance, the longest word in df_1, "deinstitutionalization" (22 letters), has an AoA of 16.05 whereas shorter words such as "eisteddfod", "architrave", "calceolaria", "penury" and "oubliette" all have AoAs above 20.00. We could have deduced this earlier from the tables produced in Queries #1-#3, though the extra correlation calculation and subsequent visualization help confirm it. Instead of looking at individual words, let's now change focus and start grouping words based on their first letter.

Queries #7 - #9: *Grouping by First Letter*

In [16]:
```
#Augmenting df_1 with new column titled "FirstLetter"
df_1['FirstLetter'] = df_1['Word'].str[0].str.lower() #Lowering uppercase words to avo
```

```
#Creating letter_counts (Not augmented to df_1)
letter_counts = df_1['Word'].groupby(df_1['FirstLetter']).count().reset_index(name='Wo
```

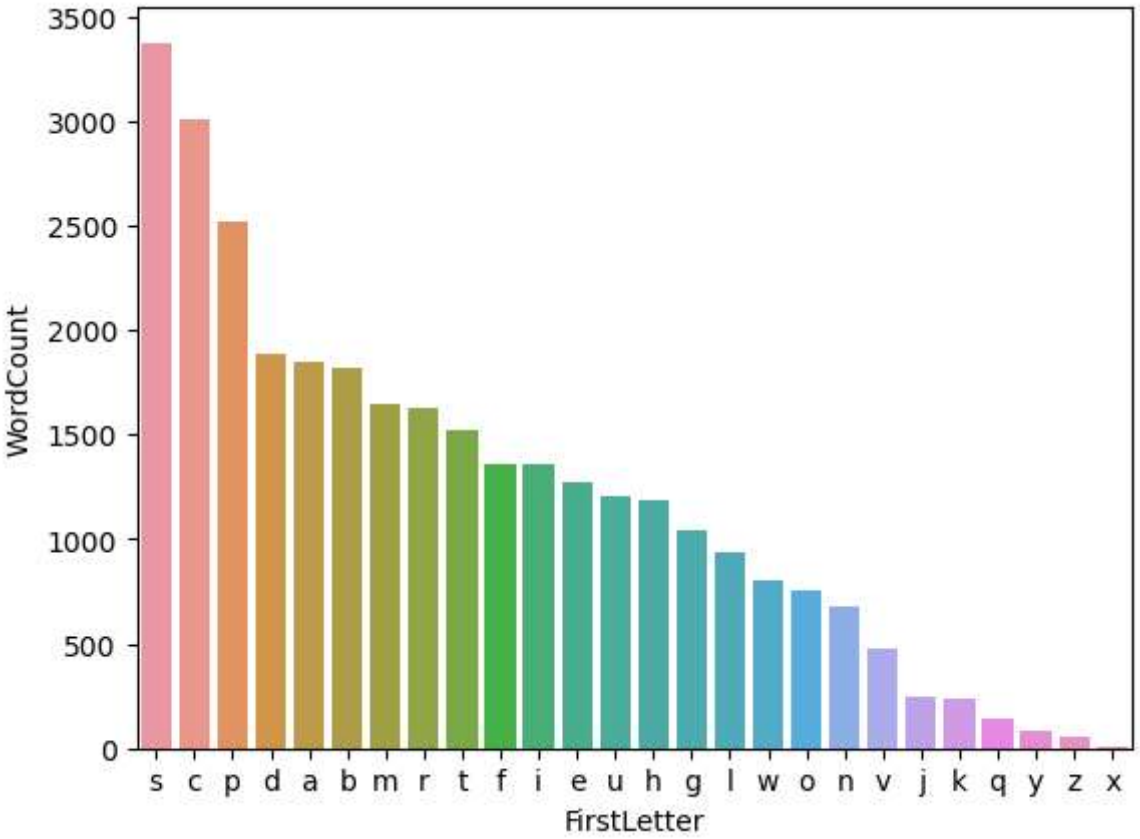
In [17]:
```
#Quickly looking at table for letter_counts (sorted by first letter groupings with the
letter_counts.sort_values('WordCount', ascending = False)
```

Out[17]:

	FirstLetter	WordCount
18	s	3377
2	c	3012
15	p	2519
3	d	1889
0	a	1848
1	b	1821
12	m	1646
17	r	1632
19	t	1518
5	f	1364
8	i	1363
4	e	1277
20	u	1204
7	h	1191
6	g	1040
11	l	938
22	w	800
14	o	757
13	n	674
21	v	479
9	j	251
10	k	239
16	q	139
24	y	85
25	z	53
23	x	7

In [18]:
```
#Creating bar plot for data in letter_counts
sns.barplot(x='FirstLetter', y='WordCount', data=letter_counts,
            order=letter_counts.sort_values('WordCount', ascending=False)['FirstLetter
```

Out[18]: <Axes: xlabel='FirstLetter', ylabel='WordCount'>



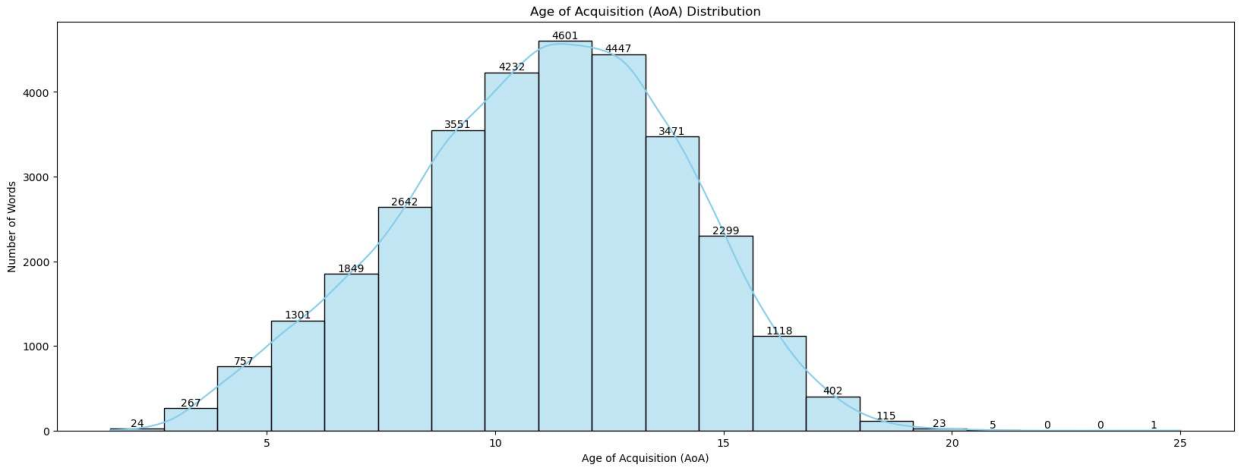
Results: Interestingly, words starting with "s", "c", "p" make up almost 30% of all words in the dataset. Before we look at this more closely, let's create a more general visualization for the distribution of words across AoA ratings. After this, we can look at how "s", "c", and "p" deviate from the average distribution. (NOTE: A few words in df_1 included capital letters. Since there were only a handful, I lowered them to help normalize the data.)

Query #10: *Number of Words Acquired per AoA*

In [19]:
```
#Setting the figure size
plt.figure(figsize=(20, 7))

#Creating all_words and all_words_plot
all_words = df_1
all_words_plot = sns.histplot(data=all_words, x='Rating.Mean', bins=20, kde=True, colo

#Adding counts on top of each bin to see distribution more clearly
#Need to ensure text is positioned in the center of each bar and at the top
#Used ChatGPT for help
for bar in all_words_plot.patches:
    height = bar.get_height()
    all_words_plot.text(bar.get_x() + bar.get_width() / 2, height, f'{int(height)}', h

#Finding the bin with the highest count
#Used ChatGPT for help
max_bar = max(all_words_plot.patches, key=lambda x: x.get_height())
max_x_coordinate_bottom_left = max_bar.get_x()
max_x_coordinate_bottom_right = max_bar.get_x() + max_bar.get_width()
```

```
#Rounding two decimal places
max_x_coordinate_bottom_left = round(max_x_coordinate_bottom_left, 2)
max_x_coordinate_bottom_right = round(max_x_coordinate_bottom_right, 2)

#Labeling histogram
plt.title('Age of Acquisition (AoA) Distribution')
plt.xlabel('Age of Acquisition (AoA)')
plt.ylabel('Number of Words')

#Plot
plt.show()

#Print
print(f' The bar with the highest number of words accounts for AoAs between {max_x_coo
```

```
C:\Users\jacmo\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: u
se_inf_as_na option is deprecated and will be removed in a future version. Convert in
f values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```




```
The bar with the highest number of words accounts for AoAs between 10.95 and 12.12.
```

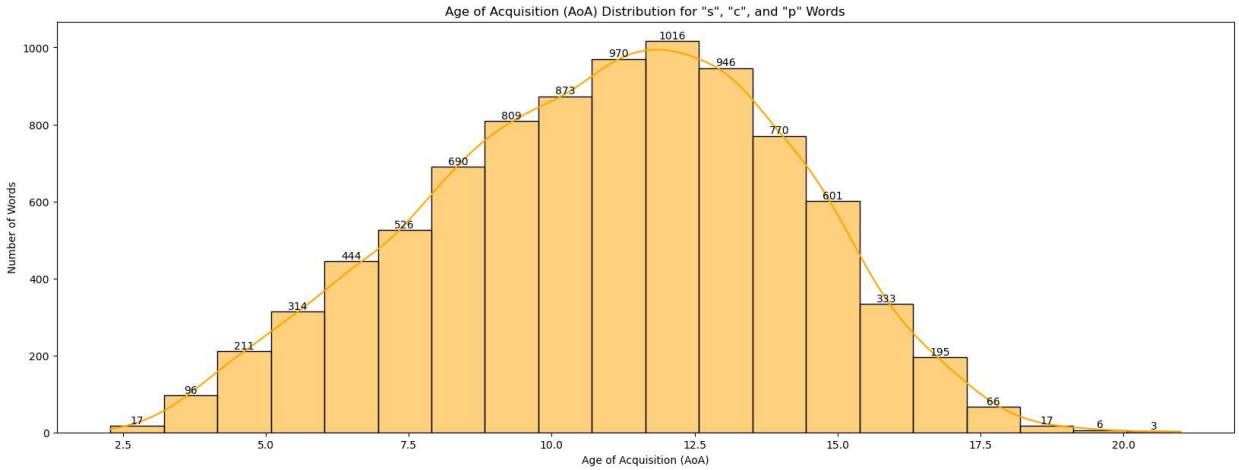
Results: This visualization demonstrates that the largest number of English words in df_1 are learned between ages 10.95 and 12.12. Importantly, this correlates with our mean AoA of 11.00. Let's now look at "s", "c", and "p" words more closely.

Query #11: *How are "s", "c", and "p" Words Distributed?*

In [20]:
```
#Set the figure size
plt.figure(figsize=(20, 7))

#Creating scp_words and scp_words_plot
scp_words = df_1[df_1['FirstLetter'].isin(['s', 'c', 'p'])]
scp_words_plot = sns.histplot(data=scp_words, x='Rating.Mean', bins=20, kde=True, colo

#Adding counts on top of each bin to see distribution more clearly
#Need to ensure text is positioned in the center of each bar and at the top
for bar in scp_words_plot.patches:
    height = bar.get_height()
    scp_words_plot.text(bar.get_x() + bar.get_width() / 2, height, f'{int(height)}', h

#Finding the bin with the highest count
max_bar = max(scp_words_plot.patches, key=lambda x: x.get_height())
max_x_coordinate_bottom_left = max_bar.get_x()
max_x_coordinate_bottom_right = max_bar.get_x() + max_bar.get_width()
```

```
#Rounding two decimal places
max_x_coordinate_bottom_left = round(max_x_coordinate_bottom_left, 2)
max_x_coordinate_bottom_right = round(max_x_coordinate_bottom_right, 2)

#Labeling histogram
plt.title('Age of Acquisition (AoA) Distribution for "s", "c", and "p" Words')
plt.xlabel('Age of Acquisition (AoA)')
plt.ylabel('Number of Words')

#Plot
plt.show()

#Print
print(f' The bar with the highest number of words accounts for AoAs between {max_x_coo
```

```
C:\Users\jacmo\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: u
se_inf_as_na option is deprecated and will be removed in a future version. Convert in
f values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```




```
The bar with the highest number of words accounts for AoAs between 11.64 and 12.58.
```

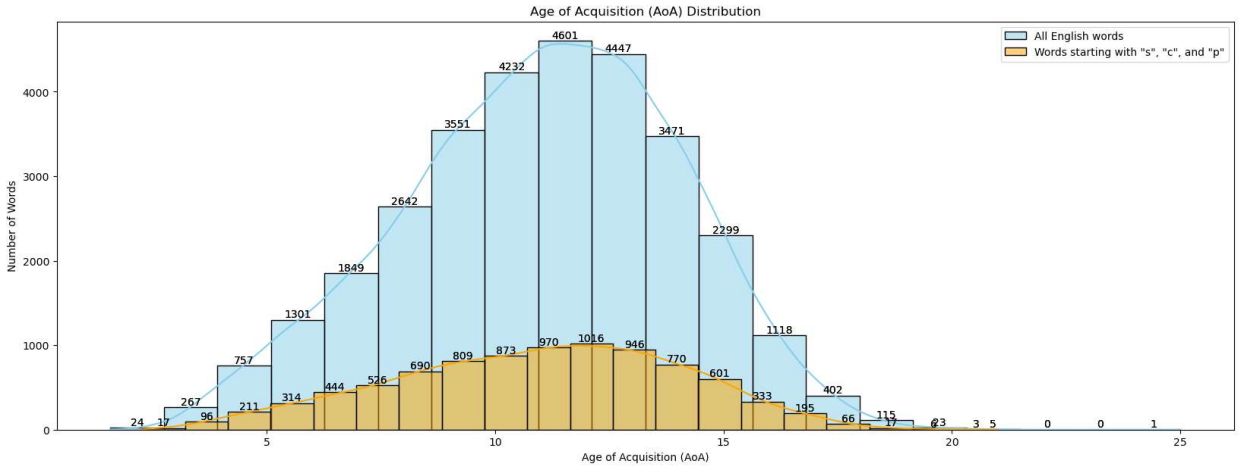
Results: The AoA distribution for "s", "c", and "p" words follows the previous histogram closely, but not perfectly. Here, the largest number of words are learned between the ages 11.64 and 12.58 which is slightly above the Average AoA for the complete dataset. Let's develop one more histogram that shows the two graphs overlapping each other. This will help us more clearly see how the two relate to each another.

Query #12: *How do "s", "c", and "p" Words Deviate From Overall AoA Distribution?*

In [21]:
```
#Setting the figure size
plt.figure(figsize=(20, 7))

#all_words_plot
all_words_plot = sns.histplot(data=all_words, x='Rating.Mean', bins=20, kde=True, colo
                              label= "All English words")

#scp_words_plot
scp_words_plot = sns.histplot(data=scp_words, x='Rating.Mean', bins=20, kde=True, colo
                              label='Words starting with "s", "c", and "p"' )

#Adding counts on top of each bin to see distribution more clearly
#Need to ensure text is positioned in the center of each bar and at the top
```

```
for bar in all_words_plot.patches:
    height = bar.get_height()
    all_words_plot.text(bar.get_x() + bar.get_width() / 2, height, f'{int(height)}', h

for bar in scp_words_plot.patches:
    height = bar.get_height()
    scp_words_plot.text(bar.get_x() + bar.get_width() / 2, height, f'{int(height)}', h

#Adding a legend
plt.legend()

#Labeling histogram
plt.title('Age of Acquisition (AoA) Distribution')
plt.xlabel('Age of Acquisition (AoA)')
plt.ylabel('Number of Words')

#Plot
plt.show()
```

```
C:\Users\jacmo\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: u
se_inf_as_na option is deprecated and will be removed in a future version. Convert in
f values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\jacmo\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: u
se_inf_as_na option is deprecated and will be removed in a future version. Convert in
f values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



Results: Now that we can see the deviation of "s", "c", and "p" from the Average AoA, let's calculate each letter's percent deviation from df_1's Average AoA.

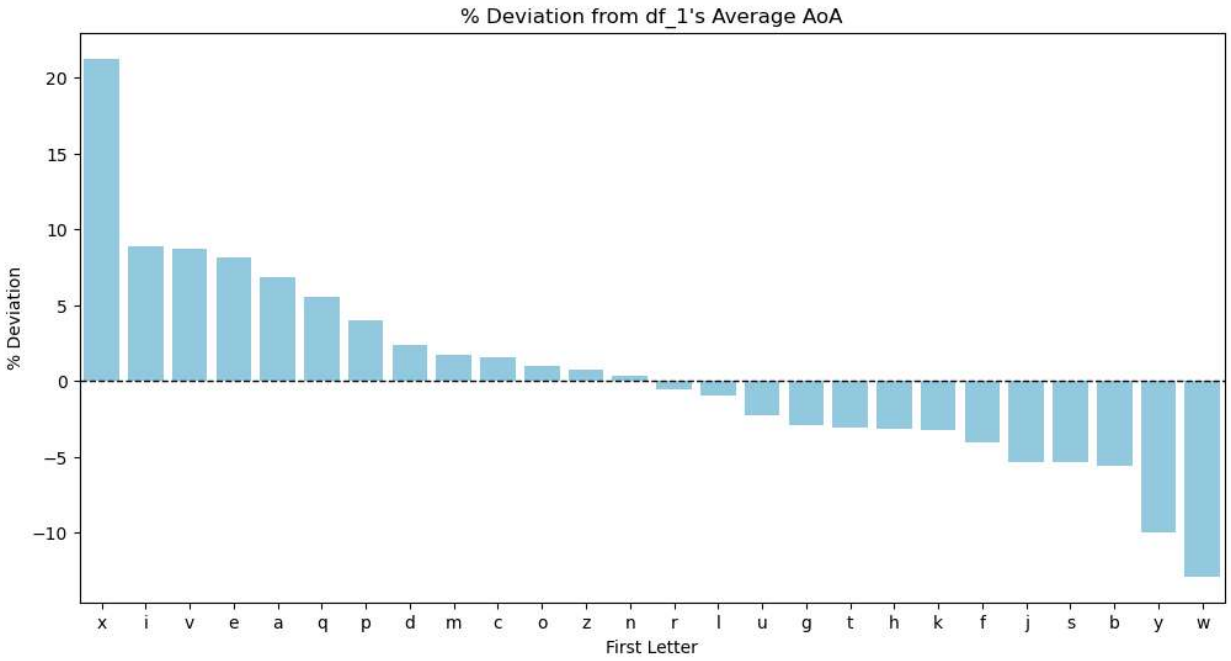
Query #13 and #14: *Plotting Each Letter's Deviation From df_1's Average AoA*

In [22]:
```
#Calculating average AoA in df_1
overall_mean = df_1['Rating.Mean'].mean()
overall_mean_rounded = round(overall_mean, 2)
print(f'The average AoA for df_1 is {overall_mean_rounded}.')

#Calculating the Average AoA for each letter grouping
letter_means = round((df_1.groupby('FirstLetter')['Rating.Mean'].mean().reset_index(na

#Calculating the Deviation % from the overall_mean
letter_means['Deviation %'] = round((((letter_means['LetterMean'] - overall_mean)/over
```

```
#Sorting letter_means
letter_means_sorted = letter_means.sort_values(by='Deviation %', ascending=False)

#Print
print(letter_means_sorted)
```

```
The average AoA for df_1 is 11.0.
   FirstLetter  LetterMean  Deviation %
23           x       13.34        21.27
8            i       11.98         8.91
21           v       11.96         8.73
4            e       11.90         8.18
0            a       11.75         6.82
16           q       11.61         5.55
15           p       11.44         4.00
3            d       11.26         2.36
12           m       11.19         1.73
2            c       11.17         1.55
14           o       11.11         1.00
25           z       11.08         0.73
13           n       11.04         0.36
17           r       10.94        -0.55
11           l       10.89        -1.00
20           u       10.75        -2.27
6            g       10.68        -2.91
19           t       10.66        -3.09
7            h       10.65        -3.18
10           k       10.64        -3.27
5            f       10.55        -4.09
9            j       10.41        -5.36
18           s       10.41        -5.36
1            b       10.38        -5.64
24           y        9.90       -10.00
22           w        9.58       -12.91
```

In [23]:
```
#Setting the barplot size
plt.figure(figsize=(12, 6))

#Creating a bar plot
sns.barplot(data=letter_means_sorted, x='FirstLetter', y='Deviation %', color='skyblue

#Adding a horizontal line at y=0 to indicate df_1's Average AoA
#Used ChatGPT for help
plt.axhline(0, color='black', linestyle='--', linewidth=1)

#Labeling bar plot
plt.xlabel('First Letter')
plt.ylabel('% Deviation')
plt.title("% Deviation from df_1's Average AoA")

#Plot
plt.show()
```



Results: The letters within 1% deviation from df_1's Average AoA are "o", "z", "n", "r", and "l". Let's now perform the same queries for the Dutch dataset and see how close they are to this Section.

Section 2: Analysis of df_2

Query #15: *Highest AoA*

In [24]:
```
#Including only the most pertinent columns
df_2.sort_values('Average_AoA', ascending=False)[['Word', 'Translation','Average_AoA',
```

Out[24]:

	Word	Translation	Average_AoA	Average_%_Known
31160	samovar	NaN	23	11.111111
31159	motet	NaN	20.045455	12.599206
31158	wallingant	NaN	19.75	22.222222
31157	mui	NaN	19.5	11.111111
31156	feut	NaN	19.5	11.111111
31155	assuradeur	NaN	19.5	11.111111
31154	fiducie	NaN	19	11.111111
31153	judicium	NaN	18.8	27.777778
31152	bursaal	NaN	18.666667	33.333333
31151	ostracisme	NaN	18.5	22.222222

Query #16: *Lowest AoA*

In [25]:
```
#Including only the most pertinent columns
df_2.sort_values('Average_AoA', ascending=True)[['Word', 'Translation','Average_AoA',
```

Out[25]:

	Word	Translation	Average_AoA	Average_%_Known
0	mama	NaN	2.044257	100
1	ja	NaN	2.25	100
2	mamma	NaN	2.332547	93.518519
3	papa	NaN	2.336327	100
4	nee	NaN	2.555556	100
5	kaka	NaN	2.611111	100
6	pappa	NaN	2.764825	87.962963
7	ikke	NaN	2.9	100
8	tutje	NaN	2.941176	94.444444
9	tuut	toot	2.957447	94

Results: Queries 15 and 16 reveal an AoA range of 2.04-23.00 for df_2. This is similar to the data we have for df_1, though slightly smaller. Interestingly, while we see instances of "mama" and "mamma" within 10 of the earliest words learned - just like in df_1 - we also see the inclusion of "papa" and "pappa". As mentioned earlier, it would be intesting to compare regulations for parental paid leave in Dutch speaking countries and English speaking countries. For instance, maybe Dutch speaking father's have more time off when the baby is born, allowing the baby to learn "papa" quicker. More tests would have to be conducted to prove such a statement; for now, I simply note it and continue by looking at the Average Word Length.

Query #17 and #18: *Average Word Length*

In [26]:
```
#Including only the most pertinent columns
df_2.sort_values(by='Word', key=lambda x: x.str.len(), ascending=False)[['Word', 'Trar
```

Out[26]:

	Word	Translation	Average_AoA	Average_%_Known
25751	volksvertegenwoordiger	representative	12.5	100
25017	hersenvliesontsteking	meningitis	12.3	100
13313	kruidje-roer-mij-niet	touch-me-not	9.8	87.5
16754	doorzettingsvermogen	NaN	10.5	100
20782	snelheidsovertreding	breach of the speed limit	11.3	100

In [27]:
```
#Augmenting df_2 with a new column titled "Word_Length"
df_2['Word_Length'] = df_2['Word'].apply(lambda x: len(str(x)))

#Calculating the longest, shortest, and average Word_Length

#Longest
```

```
max_df_2_word_length = df_2['Word_Length'].max()

#Shortest
min_df_2_word_length = df_2['Word_Length'].min()

#Average
avg_df_2_word_length = round(df_2['Word_Length'].mean(), 2)

print(f'The longest word in df_2 is {max_df_2_word_length} letters.')
print(f'The shortest word in df_2 is {min_df_2_word_length} letter.')
print(f'The average word length for df_2 is {avg_df_2_word_length} letters.')
```

```
The longest word in df_2 is 22 letters.
The shortest word in df_2 is 1 letter.
The average word length for df_2 is 8.32 letters.
```

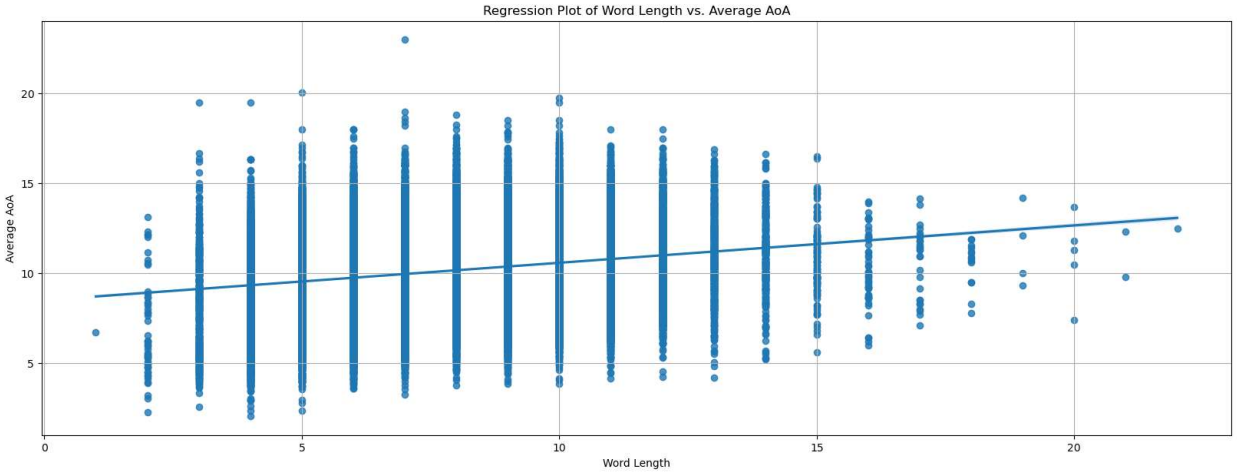
Results: Interestingly, the longest and shortest words in df_1 and df_2 are the same lengths (22 and 1 letter(s) long), and the average word length only differ by .27 letters (8.05 letters in df_1 vs. 8.32 letters in df_2). Let's see what the correlation between Word Length and Average AoA looks like for df_2.

Queries #19 and #20: *Correlation Between WordLength and AoA*

In [28]:
```
#Calculating the correlation between Word_Length and AoA
correlation_coefficient = df_2['Word_Length'].corr(df_2['Average_AoA'])

#Print
print(f'Correlation Coefficient: {correlation_coefficient}')
```

```
Correlation Coefficient: 0.21028656013653999
```

In [29]:
```
#Setting the regression plot size
plt.figure(figsize=(20, 7))

#Converting the columns to numeric types
#Used ChatGPT
df_2['Word_Length'] = pd.to_numeric(df_2['Word_Length'], errors='coerce')
df_2['Average_AoA'] = pd.to_numeric(df_2['Average_AoA'], errors='coerce')


#Creating regression plot for Average AoA and Word_Length
sns.regplot(x=df_2['Word_Length'], y=df_2['Average_AoA'])
plt.title('Regression Plot of Word Length vs. Average AoA')
plt.xlabel('Word Length')
plt.ylabel('Average AoA')
plt.grid(True)

#Plot
plt.show()
```



Results: As we can see, the correlation is actually lower for Dutch words at 21%. These perhaps tells us that longer words are learned at a younger age in Dutch than in English (though not by a significant amount, as the correlation for English words was also fairly low). It might also tell us something more about Dutch grammar and the construction of long words over multi-word, phrasal constructions, as is typical in English (i.e., "snelheidsovertreding" vs. "breach of the speed limit"). More research could be done as it relates to this question. For now, let's move on by grouping words by their first letter.

Query #21 - #23: *Grouping by First Letter*

In [30]:
```
#Augmenting datset with new column titled "FirstLetter"
df_2['FirstLetter'] = df_2['Word'].str[0].str.lower() #Lowering uppercase words to avo

#Creating total_letter_counts (Not augmented to df_2)
total_letter_counts = df_2['Word'].groupby(df_2['FirstLetter']).count().reset_index(na
```

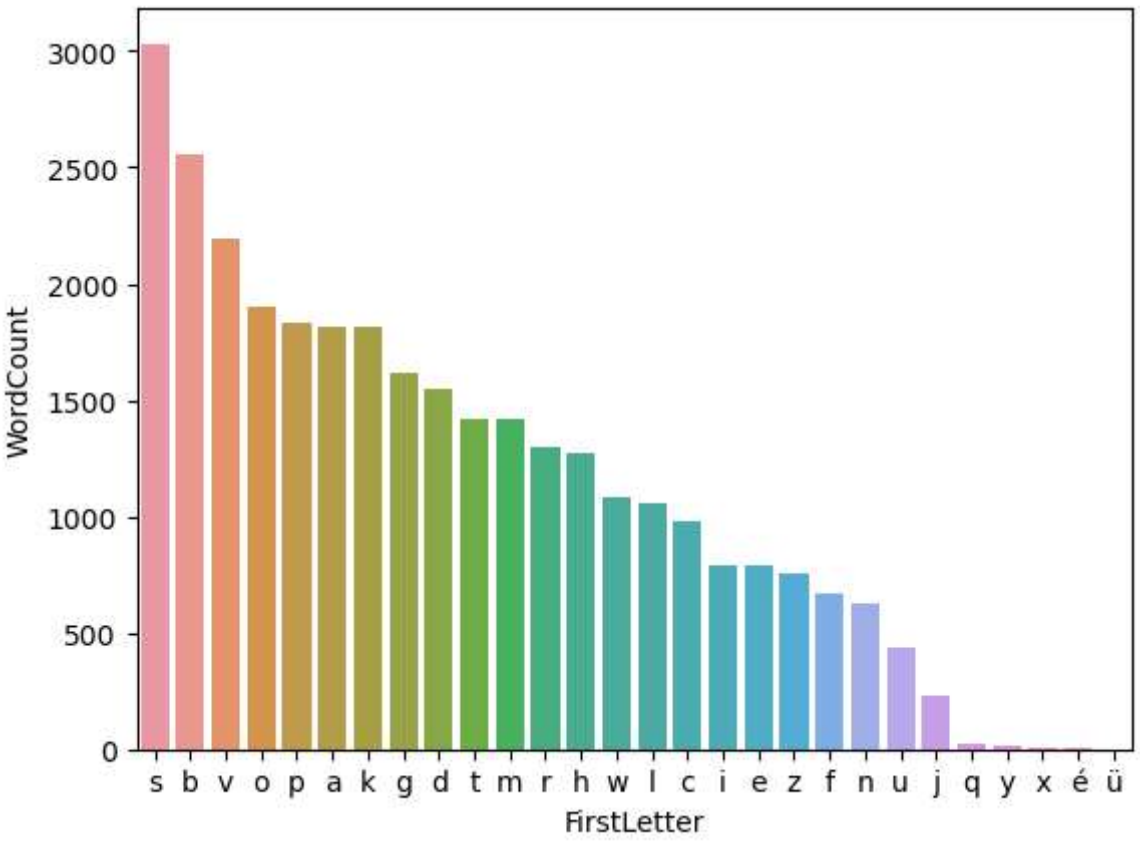
In [31]:
```
#Quickly looking at total_letter_counts table
total_letter_counts.sort_values('WordCount', ascending = False)
```

Out[31]:

	FirstLetter	WordCount
18	s	3032
1	b	2554
21	v	2198
14	o	1897
15	p	1829
0	a	1813
10	k	1812
6	g	1613
3	d	1551
19	t	1422
12	m	1416
17	r	1299
7	h	1276
22	w	1086
11	l	1056
2	c	976
8	i	792
4	e	787
25	z	756
5	f	669
13	n	623
20	u	435
9	j	234
16	q	26
24	y	17
23	x	5
26	é	2
27	ü	1

In [32]:
```
#Creating bar plot for data in total_letter_counts
sns.barplot(x='FirstLetter', y='WordCount', data=total_letter_counts,
            order=total_letter_counts.sort_values('WordCount', ascending=False)['First
```

Out[32]:
```
<Axes: xlabel='FirstLetter', ylabel='WordCount'>
```



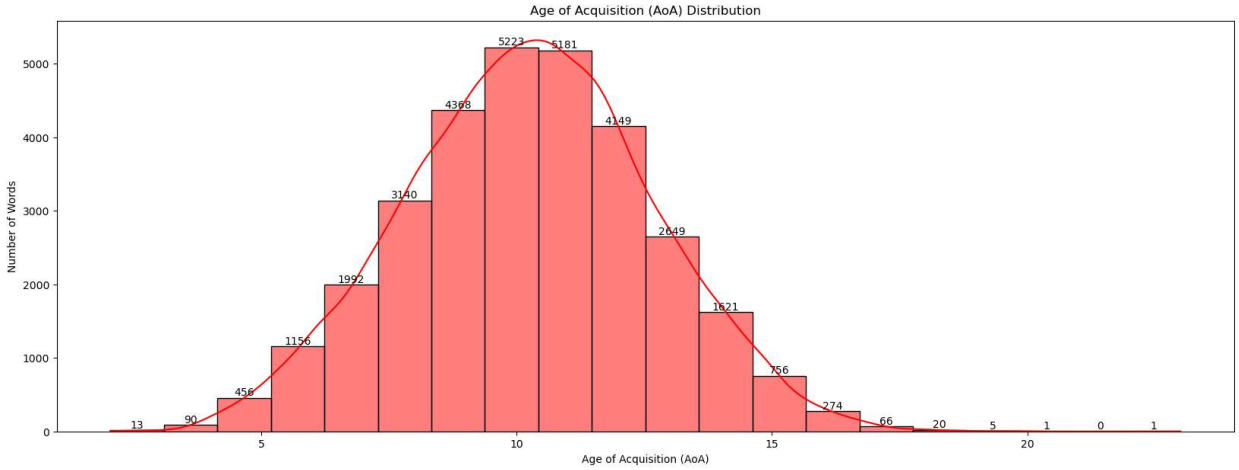
Results: The queries reveal that "s" words occur most frequently in Dutch, just as in English. This could be a coincidence; let's turn our attention to the distribution of words across AoA ratings and continue performing different queries.

Query 24: *Number of Words Acquired per AoA*

In [33]:
```
#Setting the figure size
plt.figure(figsize=(20, 7))

#Creating all_dutch_words and all_dutch_words_plot
all_dutch_words = df_2
all_dutch_words_plot = sns.histplot(data=all_dutch_words, x='Average_AoA', bins=20, kd

#Adding counts on top of each bin for all_words_plot
for bar in all_dutch_words_plot.patches:
    height = bar.get_height()
    all_dutch_words_plot.text(bar.get_x() + bar.get_width() / 2, height, f'{int(height

#Finding the bin with the highest count
max_dutch_bar = max(all_dutch_words_plot.patches, key=lambda x: x.get_height())
max_x_dutch_coordinate_bottom_left = max_dutch_bar.get_x()
max_x_dutch_coordinate_bottom_right = max_dutch_bar.get_x() + max_dutch_bar.get_width(

#Rounding two decimal places
max_x_dutch_coordinate_bottom_left = round(max_x_dutch_coordinate_bottom_left, 2)
max_x_dutch_coordinate_bottom_right = round(max_x_dutch_coordinate_bottom_right, 2)

#Labeling the histogram
plt.title('Age of Acquisition (AoA) Distribution')
plt.xlabel('Age of Acquisition (AoA)')
```

```
plt.ylabel('Number of Words')

#Plot
plt.show()

#Print
print(f' The bar with the highest number of words accounts for AoAs between {max_x_dut
```

```
C:\Users\jacmo\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: u
se_inf_as_na option is deprecated and will be removed in a future version. Convert in
f values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```




```
The bar with the highest number of words accounts for AoAs between 9.38 and 10.43.
```

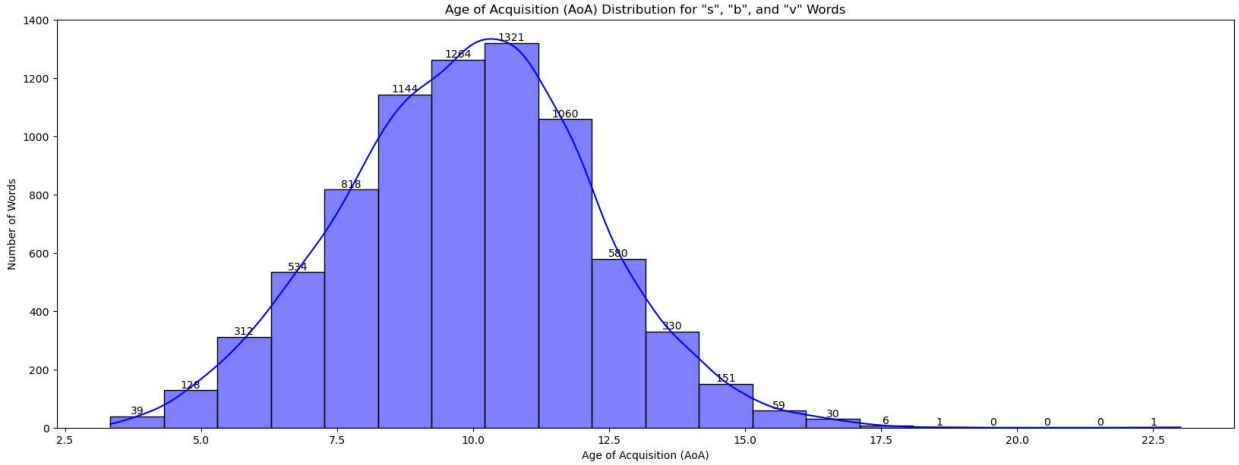
Results: While English speakers tend to learn the most number of words between the ages of 10.95 and 12.12, Dutch speakers learn the most number of words between 9.38 and 10.43. This isn't that far off from the average AoA in df_1, but still worth noting. Let's now look at how groupings "s", "b", and "v" - the three most frequent letters Dutch words start with in df_2 - relate to this distribution.

Query 25: *How are "s", "b", and "v" Words Distributed?*

In [34]:
```
#Setting the figure size
plt.figure(figsize=(20, 7))

#Creating sbv_words and sbv_words_plot
sbv_words = df_2[df_2['FirstLetter'].isin(['s', 'b', 'v'])]
sbv_words_plot = sns.histplot(data=sbv_words, x='Average_AoA', bins=20, kde=True, colo

#Adding counts on top of each bin for all_words_plot
for bar in sbv_words_plot.patches:
    height = bar.get_height()
    sbv_words_plot.text(bar.get_x() + bar.get_width() / 2, height, f'{int(height)}', h

#Finding the bin with the highest count for scp_words_plot
max_bin_sbv_words = max(sbv_words_plot.patches, key=lambda x: x.get_height())
max_x_coordinate_bottom_left_sbv_words = max_bin_sbv_words.get_x()
max_x_coordinate_bottom_right_sbv_words = max_bin_sbv_words.get_x() + max_bin_sbv_word

#Rounding two decimal places
max_x_coordinate_bottom_left_sbv_words = round(max_x_coordinate_bottom_left_sbv_words,
max_x_coordinate_bottom_right_sbv_words = round(max_x_coordinate_bottom_right_sbv_word
```

```
#Labeling the histogram
plt.title('Age of Acquisition (AoA) Distribution for "s", "b", and "v" Words')
plt.xlabel('Age of Acquisition (AoA)')
plt.ylabel('Number of Words')

#Plot
plt.show()

#Print
print(f' The bar with the highest number of words accounts for AoAs between {max_x_coo
```

```
C:\Users\jacmo\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: u
se_inf_as_na option is deprecated and will be removed in a future version. Convert in
f values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```




```
The bar with the highest number of words accounts for AoAs between 10.22 and 11.2.
```

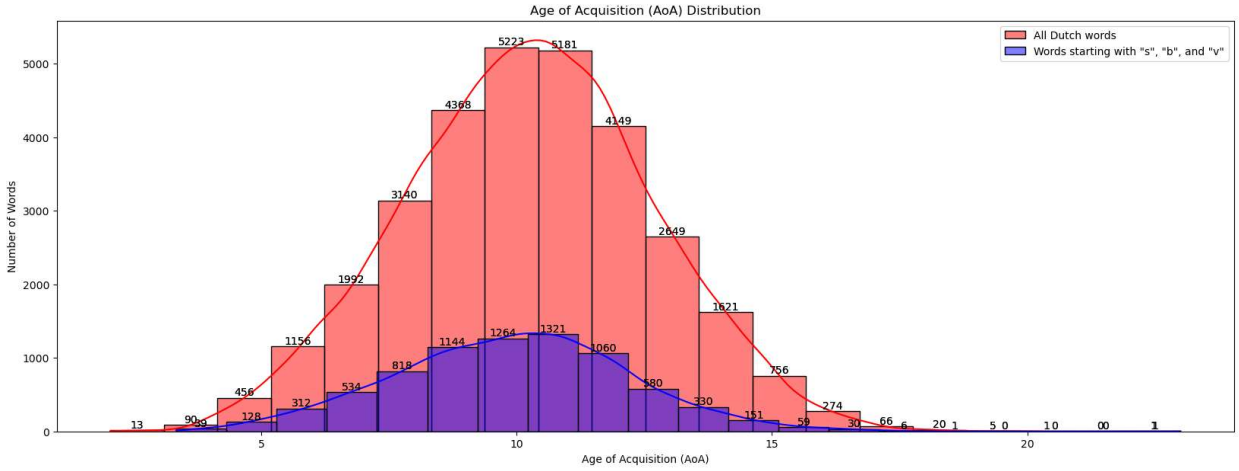
Results: There is a slight deviation from the Average AoA. Just as with df_1, let's show this deviation by overlaying the two histograms.

Query #26: *How do "s", "c", and "p" Words Deviate From Overall AoA Distribution?*

In [35]:
```
#Setting the figure size
plt.figure(figsize=(20, 7))

#all_dutch_words_plot
all_dutch_words_plot = sns.histplot(data=df_2, x='Average_AoA', bins=20, kde=True, col
                                    label = "All Dutch words")

#sbv_words_plot
sbv_words_plot = sns.histplot(data=sbv_words, x='Average_AoA', bins=20, kde=True, colo
                              label='Words starting with "s", "b", and "v"')

# Adding counts on top of each bin for all_words_plot
for bar in all_dutch_words_plot.patches:
    height = bar.get_height()
    all_dutch_words_plot.text(bar.get_x() + bar.get_width() / 2, height, f'{int(height

# Adding counts on top of each bin for sbv_words_plot
for bar in sbv_words_plot.patches:
    height = bar.get_height()
    sbv_words_plot.text(bar.get_x() + bar.get_width() / 2, height, f'{int(height)}', h

#Adding a legend
```

```
plt.legend()

#Labeling histogram
plt.title('Age of Acquisition (AoA) Distribution')
plt.xlabel('Age of Acquisition (AoA)')
plt.ylabel('Number of Words')

#Plot
plt.show()
```

```
C:\Users\jacmo\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: u
se_inf_as_na option is deprecated and will be removed in a future version. Convert in
f values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\jacmo\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: u
se_inf_as_na option is deprecated and will be removed in a future version. Convert in
f values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



Results: The difference between these two histograms is slight. Let's now look at the deviations of each first letter grouping from the Average AoA for df_2.

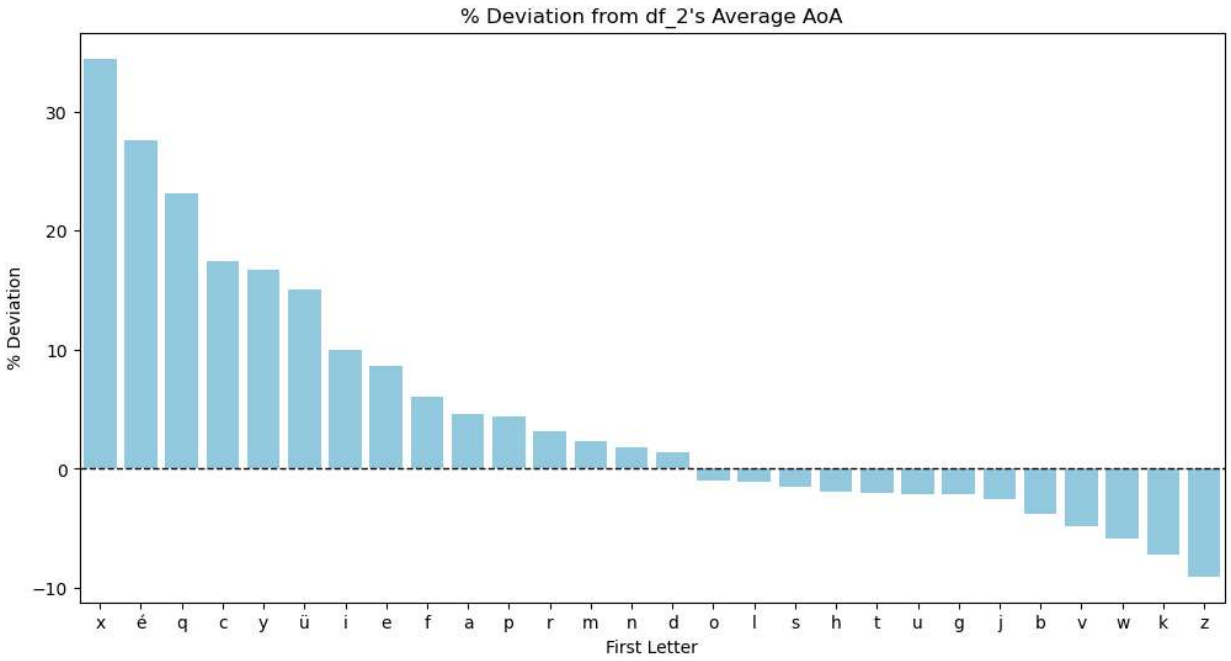
Queries #27 and #28: *Plotting Each Letter's Deviation From df_1's Average AoA*

In [36]:
```
#Calculating Average AoA in df_2
overall_mean_2 = df_2['Average_AoA'].mean()
overall_mean_2_rounded = round(overall_mean_2, 2)
print(f'The average AoA for df_1 is {overall_mean_2_rounded}.')

# Calculate the Average AoA for each letter grouping
letter_means_2 = round((df_2.groupby('FirstLetter')['Average_AoA'].mean().reset_index(

# Calculate each letter groupings deviation from the overall mean
letter_means_2['Deviation %'] = round(((letter_means_2['LetterMean'] - overall_mean_2)

#Sorting letter_means_2
letter_means_sorted_2 = letter_means_2.sort_values(by='Deviation %', ascending=False)

#Print
print(letter_means_sorted_2)
```

```
The average AoA for df_1 is 10.23.
   FirstLetter  LetterMean  Deviation %
23           x       13.76        34.45
26           é       13.06        27.61
16           q       12.60        23.12
2            c       12.02        17.45
24           y       11.94        16.67
27           ü       11.78        15.10
8            i       11.26        10.02
4            e       11.12         8.65
5            f       10.85         6.02
0            a       10.71         4.65
15           p       10.69         4.45
17           r       10.56         3.18
12           m       10.47         2.30
13           n       10.42         1.81
3            d       10.38         1.42
14           o       10.13        -1.02
11           l       10.12        -1.12
18           s       10.08        -1.51
7            h       10.04        -1.90
19           t       10.03        -2.00
20           u       10.02        -2.09
6            g       10.02        -2.09
9            j        9.97        -2.58
1            b        9.85        -3.75
21           v        9.74        -4.83
22           w        9.64        -5.81
10           k        9.50        -7.17
25           z        9.31        -9.03
```

In [37]:

```
# Set the figure size
plt.figure(figsize=(12, 6))

# Create a bar plot
sns.barplot(data=letter_means_sorted_2, x='FirstLetter', y='Deviation %', color='skybl

# Add a horizontal line at y=0 to indicate the overall mean
plt.axhline(0, color='black', linestyle='--', linewidth=1)

# Set labels and title
plt.xlabel('First Letter')
plt.ylabel('% Deviation')
plt.title("% Deviation from df_2's Average AoA")

#Plot
plt.show()
```



Results: While the deviation percentages are slightly higher in df_2 than df_1, o", "n", and "l" also fall within a 2% deviation limit from the Average AoA. This is interesting considering we are dealing with two different languages. In order to more clearly visualize some of the similiarities and differences I have highlighted so far in my investigation, I now turn to my last section, "Comparing AoA's through Visualizations".

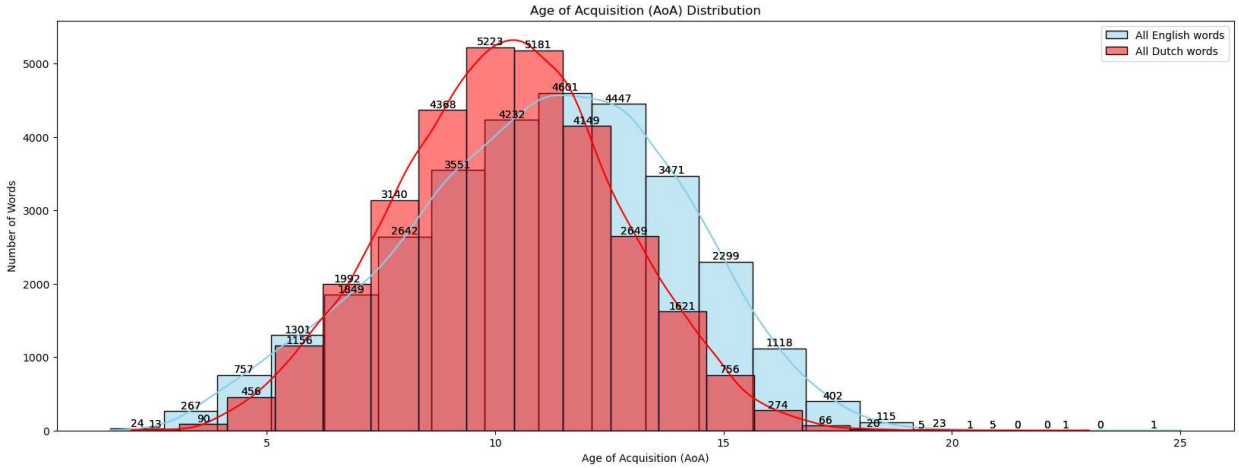
Comparing AoA's through Visualizations

Query #29: *Comparing Distribution of English AoA to Distributions of Dutch AoA*

In [38]:
```
#Setting the figure size
plt.figure(figsize=(20, 7))

#all_english_words_plot
all_english_words_plot = sns.histplot(data=df_1, x='Rating.Mean', bins=20, kde=True, c
                              label= "All English words")

#all_dutch_words_plot
all_dutch_words_plot = sns.histplot(data=all_dutch_words, x='Average_AoA', bins=20, kd
                                    label = "All Dutch words")


#Adding counts on top of each bin to see distribution more clearly
#Need to ensure text is positioned in the center of each bar and at the top
for bar in all_english_words_plot.patches:
    height = bar.get_height()
    all_english_words_plot.text(bar.get_x() + bar.get_width() / 2, height, f'{int(heig

for bar in all_dutch_words_plot.patches:
    height = bar.get_height()
    all_dutch_words_plot.text(bar.get_x() + bar.get_width() / 2, height, f'{int(height

#Adding a legend
plt.legend()
```

```
#Labeling histogram
plt.title('Age of Acquisition (AoA) Distribution')
plt.xlabel('Age of Acquisition (AoA)')
plt.ylabel('Number of Words')

#Plot
plt.show()
```

```
C:\Users\jacmo\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: u
se_inf_as_na option is deprecated and will be removed in a future version. Convert in
f values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\jacmo\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: u
se_inf_as_na option is deprecated and will be removed in a future version. Convert in
f values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

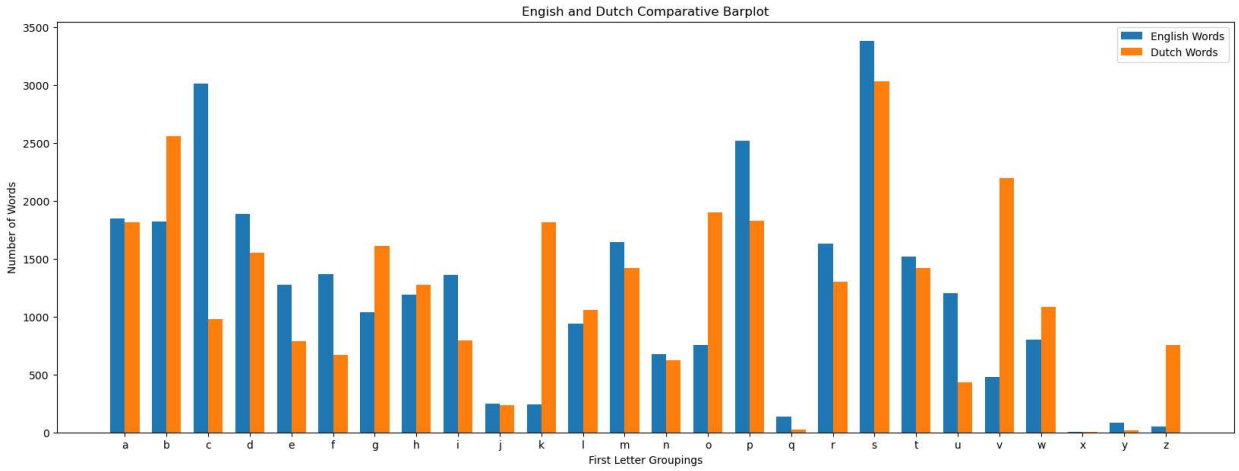


Results: When we overlap the two histograms, we see that Dutch speakers tend to learn a higher number of words earlier than English speakers.

Queries #30 and #31: *Comparing English and Dutch First Letter Groups and Deviation from Average AoA*

In [39]:
```
#ChatGpt used for the majority of this query. Particularly used for concatenating proc
#é and ü in letter_counts and ensuring bars were correctly plotted side-by-side.
#The same structure was then used for Query 31.

#Setting the figure size
plt.figure(figsize=(20, 7))

#Creating Variable "common_categories" to help calculate a new dataframe
common_categories = set(letter_counts['FirstLetter']) & set(total_letter_counts['First

#Creating a new dataframe for the characters missing in df_1 FirstLetter groupings (i.
missing_categories_df = pd.DataFrame({'FirstLetter': list(set(total_letter_counts['Fir

#Concatenating letter_counts with missing_categories_df
letter_counts = pd.concat([letter_counts, missing_categories_df], ignore_index=True)

#Sorting letter_counts to maintain a consistent order
letter_counts = letter_counts.sort_values(by='FirstLetter')

#Filtering dataframes to include only common categories
filtered_1 = letter_counts[letter_counts['FirstLetter'].isin(common_categories)]
```

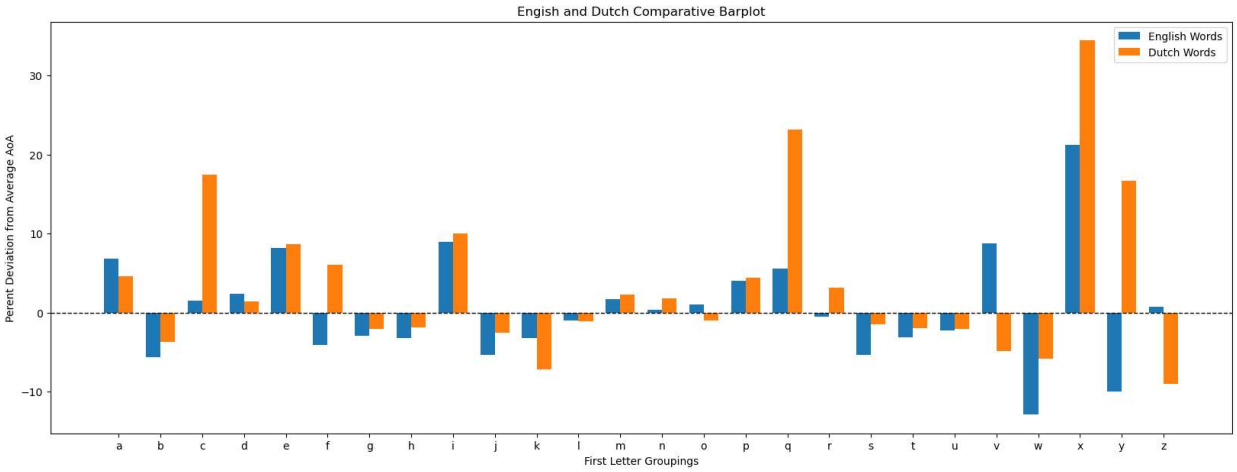
```
filtered_2 = total_letter_counts[total_letter_counts['FirstLetter'].isin(common_categc

#Setting up positions for the bars (want to ensure they are side-by-side not stacked c
bar_width = 0.35
bar_positions_df1 = range(len(filtered_1))
bar_positions_df2 = [pos + bar_width for pos in bar_positions_df1]

#Plotting data from df1
plt.bar(bar_positions_df1, filtered_1['WordCount'], width=bar_width, label='English Wc

#Plotting data from df2
plt.bar(bar_positions_df2, filtered_2['WordCount'], width=bar_width, label='Dutch Word

#Adding a legend
plt.legend()

#Labeling bar graph
plt.xlabel('First Letter Groupings')
plt.ylabel('Number of Words')
plt.title('Engish and Dutch Comparative Barplot')
plt.xticks([pos + bar_width / 2 for pos in bar_positions_df1], filtered_1['FirstLetter


#Plot
plt.show()
```



In [40]:

```
#Setting the figure size
plt.figure(figsize=(20, 7))

#Creating Variable "common_categories" to help calculate a new dataframe
common_categories = set(letter_means['FirstLetter']) & set(letter_means_2['FirstLetter

#Creating a new dataframe for the characters missing in df_1 FirstLetter groupings (i.
missing_categories_df = pd.DataFrame({'FirstLetter': list(set(letter_means_2['FirstLet

#Concatenating letter_means with missing_categories_df
letter_means = pd.concat([letter_means, missing_categories_df], ignore_index=True)

#Sorting letter_means to maintain a consistent order
letter_means = letter_means.sort_values(by='FirstLetter')

#Filtering dataframes to include only common categories
df1_new_filtered = letter_means[letter_means['FirstLetter'].isin(common_categories)]
df2_new_filtered = letter_means_2[letter_means_2['FirstLetter'].isin(common_categories
```

```
#Setting up positions for the bars (want to ensure they are side-by-side not stacked c
bar_width = 0.35
bar_positions_df1 = range(len(df1_new_filtered))
bar_positions_df2 = [pos + bar_width for pos in bar_positions_df1]

#Plotting data from df1
plt.bar(bar_positions_df1, df1_new_filtered['Deviation %'], width=bar_width, label='Er

#Plotting data from df2
plt.bar(bar_positions_df2, df2_new_filtered['Deviation %'], width=bar_width, label='Du

#Adding a horizontal line at y=0 to indicate the overall mean
plt.axhline(0, color='black', linestyle='--', linewidth=1)

#Labeling bar graph
plt.xlabel('First Letter Groupings')
plt.ylabel('Perent Deviation from Average AoA')
plt.title('Engish and Dutch Comparative Barplot')
plt.xticks([pos + bar_width / 2 for pos in bar_positions_df1], df1_new_filtered['First
plt.legend()

#Plot
plt.show()
```



Results: The final two queries show key differences between the first letter groupings for Dutch and English words. Some letter groupings between the two languages do not correlate at all (such as "z" in both queries #30 and #31), though a number present similiarites (such as "e", "i" and "s").

6. Conclusion

In my analysis of AoA statistics for 30,000 English and 30,000 Dutch words, I have uncovered key characteristics of each dataset, grouped words by their first letter, and compared these groupings across both languages. Noteworthy insights included key variations in AoA ranges for specific queries, observable similarities and difference in early learned words between the two languages, and a comparison of word lengths and AoA correlations. As I hinted at throughout this notebook, these insights only scratch the surface of what could be looked at when

comparing the two datasets and more research could be done from the work developed in this notebook.