Importing a database table as pandas dataframe and some EDA

We will now be importing some of the tables from the **stats** database into this notebook as pandas dataframes.

In [1]:

```
from sqlalchemy import create_engine
import pymysql
import pandas as pd
```

In [2]:

```
# 'mysql+pymysql://mysql_user:mysql_password@mysql_host/mysql_db'
db_connection_str = 'mysql+pymysql://root:qaz2wsx@localhost/stats'
db_connection = create_engine(db_connection_str)
```

In [3]:

```
# import the "posts" table from the 'stats' database into pandas dataframe
posts = pd.read_sql('SELECT * FROM posts', con=db_connection)
```

In [4]:

```
posts.head(3)
```

Out[4]:

	Id	PostTypeId	AcceptedAnswerId	ParentId	Score	ViewCount	Body	OwnerUserId	OwnerDisplayName	LastEditorUserId	LastEditDate
0	1	1	15.0	NaN	49	4896.0	<p>How should I elicit prior distributions fro...	8.0	None	NaN	NaT
1	2	1	59.0	NaN	34	33132.0	<p>In many different statistical methods there...	24.0	None	NaN	2010-08-07 17:56:45
2	3	1	5.0	NaN	71	6503.0	<p>What are some valuable Statistical Analysis...	18.0	None	183.0	2011-02-12 05:50:04

In [5]:

```
posts.shape
```

Out[5]:

```
(372715, 18)
```

The *posts* dataframe has 372,715 rows and 18 columns.

In [6]:

```
posts.dtypes
```

Out[6]:

```
Id                            int64
PostTypeId                    int64
AcceptedAnswerId            float64
ParentId                    float64
Score                         int64
ViewCount                   float64
Body                         object
OwnerUserId                 float64
OwnerDisplayName             object
LastEditorUserId            float64
LastEditDate         datetime64[ns]
LastActivityDate     datetime64[ns]
Title                        object
Tags                         object
AnswerCount                 float64
CommentCount                  int64
FavoriteCount               float64
CreationDate         datetime64[ns]
dtype: object
```

We now also know the datatypes for all the columns in the *posts* dataframe. We will convert the datatype for any column(s), if necessary.

As you can see above, the *posts* dataframe has more than 370,000 rows. In the future, we might work with datasets that have millions of rows.

When we have such huge datasets, often the EDA and manipulation and training models takes very long. Let's see if we can work with a smaller subset of this dataset and still get similar (*as close as possible to the ground truth*) results.

We will do a very small experiment for this above task. We will pull random subsamples from the *posts* dataframe and compare the distributions of **score** with the distribution of score of the entire dataset.

Distribution of Score

Score in this dataset is defined as (number of upvotes - number of downvotes) for the posts (*which includes questions, answers, etc*).

First the distribution of **score** for the entire dataset.

In [7]:
```
posts['Score'].describe()
```

Out[7]:
```
count    372715.000000
mean          3.166299
std           9.712236
min         -58.000000
25%           1.000000
50%           1.000000
75%           3.000000
max        1714.000000
Name: Score, dtype: float64
```

As we can see above, the mean score is 3.16, and the median is 1.0 -- this suggests that our distribution is right skewed.

Let's confirm this further.

In [8]:
```
posts['Score'].quantile(0.9)
```

Out[8]:
```
6.0
```

In [9]:
```
posts['Score'].quantile(0.95)
```

Out[9]:
```
11.0
```

As suspected, 95% of the posts have a *score* of less than equal to 11 (*when the maximum goes upto 1714*). Only 5% of the posts have high scores, which are responsible for a high mean, and an even higher standard deviation.

Let's check this hypothesis as well.

Let us create a new dataframe **posts_95percentile** containing only the rows that have a *score* of less than equal to 11.

In [10]:
```
posts_95percentile = posts.loc[posts['Score'] <= 11]
```

Now, let's check the distribution once again.

In [11]:
```
posts_95percentile['Score'].describe()
```

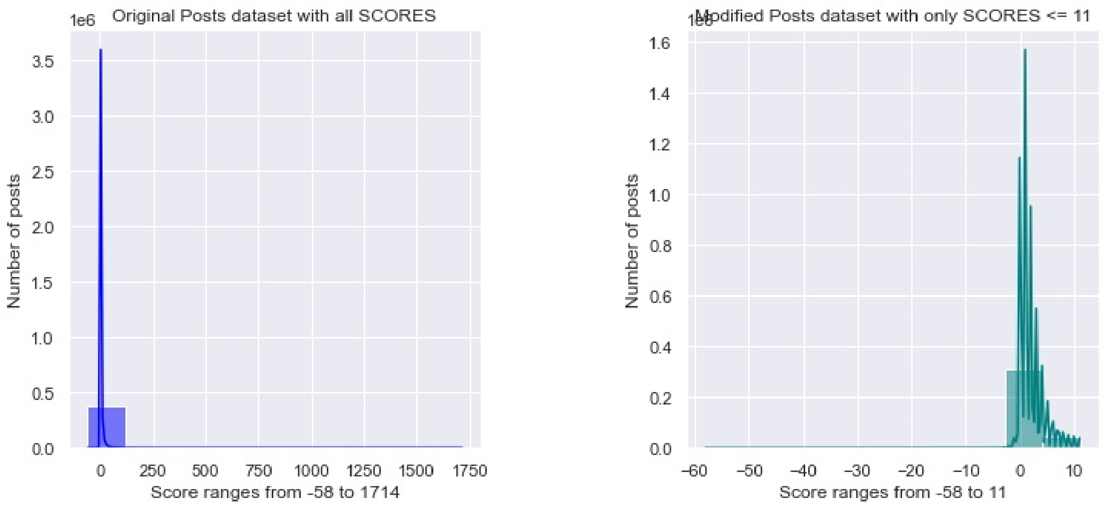
Out[11]:
```
count    356220.000000
mean          2.014609
std           2.256631
min         -58.000000
25%           1.000000
50%           1.000000
75%           3.000000
max          11.000000
Name: Score, dtype: float64
```

The **mean** has dropped from 3.16 to 2.01 and **standard deviation** from 9.71 to 2.25 (*i.e. the score is less spread now*).

Let us also visualise both the distributions.

In [12]:
```
import seaborn as sns
```

```
import seaborn as sns
import matplotlib.pyplot as plt

sns.set(style="darkgrid")

fig, (ax1,ax2) = plt.subplots(1, 2, figsize=(12, 5))

sns.histplot(data=posts, x="Score", kde=True, bins=10, color="blue", ax=ax1)
ax1.set_title('Original Posts dataset with all SCORES')
ax1.set_xlabel('Score ranges from -58 to 1714')
ax1.set_ylabel('Number of posts')

sns.histplot(data=posts_95percentile, x="Score", kde=True, bins=10, color="teal", ax=ax2)
ax2.set_title('Modified Posts dataset with only SCORES <= 11')
ax2.set_xlabel('Score ranges from -58 to 11')
ax2.set_ylabel('Number of posts')

fig.subplots_adjust(wspace=0.5)
plt.show()
```



As you might notice above, the distribution on the right seems a little bit more uniform than the one on the left.

You do not notice much variation in the sub-plots above because most of the posts don't get any upvotes or downvotes (hence, the peak around zero).
Nevertheless, we will now pull random subsamples from the **posts_95percentile** dataframe (*which has 356,220 rows*), and compare the distribution of scores.

Distribution of Score for different random subsamples

Let us pull three different subsamples from the dataframe **posts_95percentile** and compare their distribution of score(s). We will extract 25%, 50%, and 75% respectively of random data/rows from the dataframe.

In [13]:

```
# pull random 25 percent of rows
posts_25percent_random = posts_95percentile.sample(frac=0.25)

# pull random 50 percent of rows
posts_50percent_random = posts_95percentile.sample(frac=0.50)

# pull random 75 percent of rows
posts_75percent_random = posts_95percentile.sample(frac=0.75)
```

In [14]:

```
posts_25percent_random['Score'].describe()
```

Out[14]:

```
count    89055.000000
mean         2.015451
std          2.270530
min        -58.000000
25%          1.000000
50%          1.000000
75%          3.000000
max         11.000000
Name: Score, dtype: float64
```

In [15]:

```
posts_50percent_random['Score'].describe()
```

Out[15]:

```
count    178110.000000
```

```
mean          2.016321
std           2.258812
min         -58.000000
25%           1.000000
50%           1.000000
75%           3.000000
max          11.000000
Name: Score, dtype: float64
```

In [16]:

```
posts_75percent_random['Score'].describe()
```

Out[16]:

```
count    267165.000000
mean          2.013220
std           2.254812
min         -58.000000
25%           1.000000
50%           1.000000
75%           3.000000
max          11.000000
Name: Score, dtype: float64
```

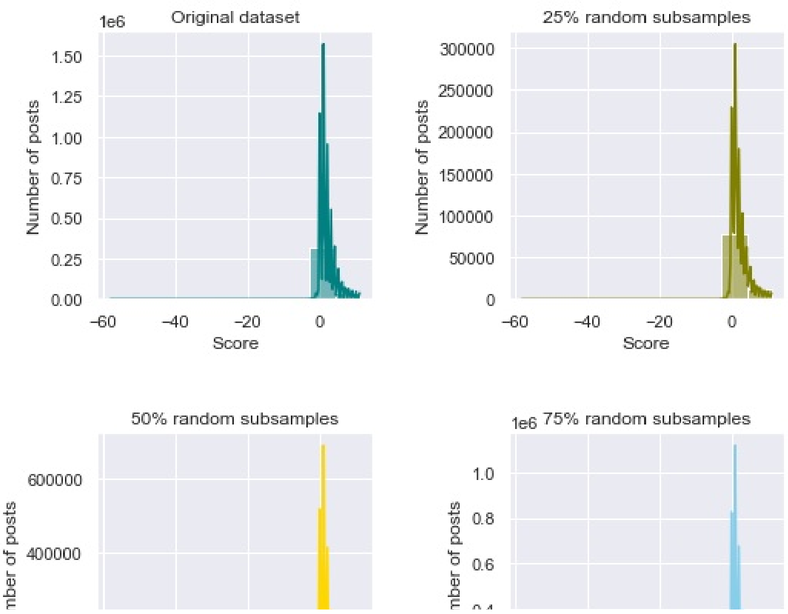
We do notice some subtle differences between the three subsamples. For example, the **posts_25percent_random** has a mean of 2.00 as compared to 2.01 for the original dataset (*very minute difference!*). For the same dataframe, the score at 25 percentile is still 0. The **posts_75percent_random** seems to be closest to the original dataset (*which isn't a surprise!*).

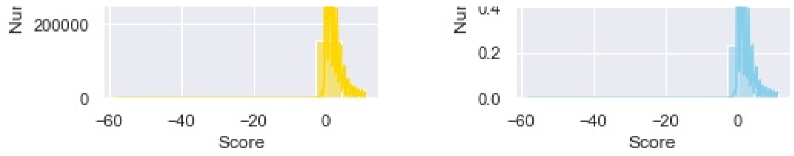
Having said the above, there isn't much difference between the three subsamples. This is primarily because most of the posts have a score of zero (as mentioned earlier). However, this might not be the case for future datasets and therefore, we have to be very careful when selecting a smaller subset especially, when training models (*high variance*).

Let us once visualise these subsamples as well.

In [17]:

```
sns.set(style="darkgrid")

fig, axs = plt.subplots(2, 2, figsize=(8, 8))

sns.histplot(data=posts_95percentile, x="Score", kde=True, bins=10, color="teal", ax=axs[0,0])
axs[0,0].set_title('Original dataset')
axs[0,0].set_ylabel('Number of posts')

sns.histplot(data=posts_25percent_random, x="Score", kde=True, bins=10, color="olive", ax=axs[0,1])
axs[0,1].set_title('25% random subsamples')
axs[0,1].set_ylabel('Number of posts')

sns.histplot(data=posts_50percent_random, x="Score", kde=True, bins=10, color="gold", ax=axs[1,0])
axs[1,0].set_title('50% random subsamples')
axs[1,0].set_ylabel('Number of posts')

sns.histplot(data=posts_75percent_random, x="Score", kde=True, bins=10, color="skyblue", ax=axs[1,1])
axs[1,1].set_title('75% random subsamples')
axs[1,1].set_ylabel('Number of posts')

fig.subplots_adjust(wspace=0.5)
fig.subplots_adjust(hspace=0.5)
plt.show()
```





As mentioned above, not much difference between the three subsamples, and they all look very similar to the original histogram.

Let us explore our original **stats** database further.
This time let us also import the *tags* table into a dataframe, and see which topics get the most questions. We will also further explore how the number of questions have changed (increased/decreased) over the length of time.

Analysis on Tags

We will begin with importing the *tags* table into a pandas dataframe.

In [18]:
```
tags = pd.read_sql('SELECT * FROM tags', con=db_connection)
```

In [19]:
```
tags.head()
```

Out[19]:

	Id	TagName	Count	ExcerptPostId	WikiPostId
0	1	bayesian	6961	20258.0	20257.0
1	2	prior	866	62158.0	62157.0
2	3	elicitation	11	NaN	NaN
3	5	open-source	16	NaN	NaN
4	6	distributions	8428	8046.0	8045.0

TagName is the name of the tag and **Count** is the number of times the particular tag appears in the asked questions.

Let us observe the most used top 15 tags.

In [20]:
```
tags_mostUsed = tags.sort_values(by = 'Count', ascending = False)
tags_top15 = tags_mostUsed.head(15)
```

In [21]:
```
tags_top15
tags_top15.style.set_properties(**{'background-color': 'pink'}, subset=['Count'])
```

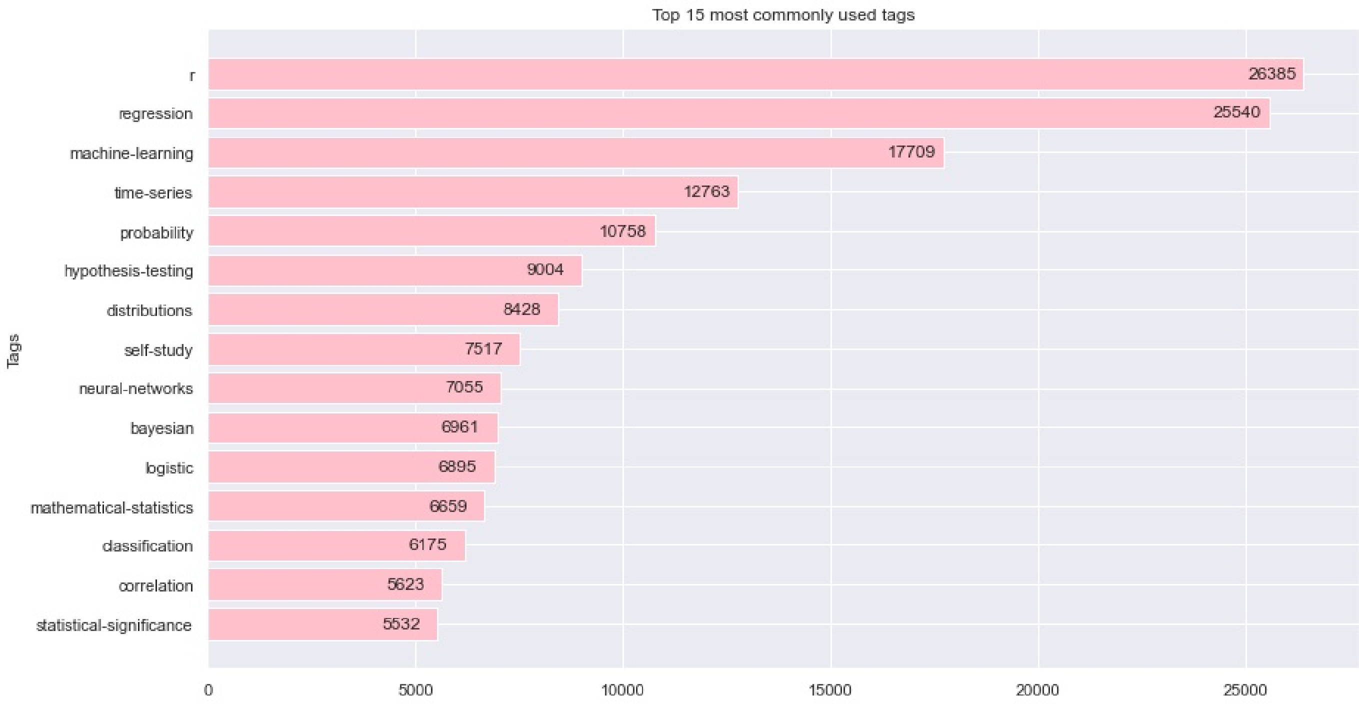
Out[21]:

	Id	TagName	Count	ExcerptPostId	WikiPostId
23	41	r	26385	2331.000000	2254.000000
49	111	regression	25540	8172.000000	8171.000000
5	9	machine-learning	17709	9066.000000	9065.000000
18	30	time-series	12763	3017.000000	3016.000000
86	188	probability	10758	20256.000000	20255.000000
21	38	hypothesis-testing	9004	3019.000000	3018.000000
4	6	distributions	8428	8046.000000	8045.000000
665	1483	self-study	7517	51952.000000	51951.000000
35	62	neural-networks	7055	50760.000000	50759.000000
0	1	bayesian	6961	20258.000000	20257.000000
453	975	logistic	6895	28327.000000	28326.000000
96	213	mathematical-statistics	6659	33251.000000	33250.000000
47	94	classification	6175	20262.000000	20261.000000
26	47	correlation	5623	8556.000000	8555.000000
65	135	statistical-significance	5532	20260.000000	20259.000000

Let's also visualise the above.

In [22]:
```
tags_top15 = tags_top15.sort_values(by=['Count']) # to present the horizontal bars from most to least
```

```
fig, ax = plt.subplots(figsize=(14, 8))

plt.barh(width = tags_top15.Count, y = tags_top15.TagName, color = 'pink')
plt.title('Top 15 most commonly used tags')
plt.ylabel('Tags')

ax.bar_label(ax.containers[0], padding = -38)

plt.show()
```



To be honest, the tag **r** is a surprise for me. Maybe, a lot of the users of stats.stackexchange.com use r (*as the programming language*) for their statistical analysis. This is followed by **regression**, **machine-learning**, **time-series**, and so on... (***self-study*** *is an interesting choice for tag*)

Let us now see how the number of question have changed over time.

Number of questions over time

This time lets visualise how the number of questions posted on stats.stackexchange.com have changed in the past decade (*my money is on they must have increased as more and more people have increasingly gotten interested in data science/statistics*).

We will once again use the **posts** dataframe.

```
posts.head(5)
```

	Id	PostTypeId	AcceptedAnswerId	ParentId	Score	ViewCount	Body	OwnerUserId	OwnerDisplayName	LastEditorUserId	La
0	1	1	15.0	NaN	49	4896.0	<p>How should I elicit prior distributions fro...	8.0	None	NaN	
1	2	1	59.0	NaN	34	33132.0	<p>In many different statistical methods there...	24.0	None	NaN	
2	3	1	5.0	NaN	71	6503.0	<p>What are some valuable Statistical Analysis...	18.0	None	183.0	
3	4	1	135.0	NaN	23	42655.0	<p>I have two groups of data. Each with a dif...	23.0	None	NaN	
4	5	2	NaN	3.0	90	NaN	The R-project

<p><a href="http... | 23.0 | None | 23.0 | |

As we know from here, **PostTypeId = 1** are questions and **PostTypeId = 2** are answers. Since we are only interested in questions, we will simply extract them from the *posts* dataframe.

In [25]:
```
posts_questions = posts.loc[posts['PostTypeId'] == 1]
```

In [26]:
```
posts_questions.shape
```

Out[26]:
```
(187996, 18)
```

We have 187,996 questions.

Next, we have the **CreationDate** column which tells us when the question was posted. Since we are only interested in the year, we will simply extract just the relevant columns from our dataframe.

In [28]:
```
posts_questionsANDyear = pd.DataFrame()
```

In [29]:
```
posts_questionsANDyear['Body'], posts_questionsANDyear['Year'] = posts_questions['Body'], posts_questions['Creati
```

In [30]:
```
posts_questionsANDyear.head()
```

Out[30]:

	Body	Year
0	<p>How should I elicit prior distributions fro...	2010
1	<p>In many different statistical methods there...	2010
2	<p>What are some valuable Statistical Analysis...	2010
3	<p>I have two groups of data. Each with a dif...	2010
5	<p>Last year, I read a blog post from <a href=...	2010

We will now use the **groupby** method and get the *count* of how many questions were posted in a given year. We also want to return the result as a dataframe so that it is easier to visualise later.

In [31]:
```
questions_by_year = posts_questionsANDyear.groupby('Year').size().reset_index(name='Count')
```

In [32]:
```
questions_by_year
```

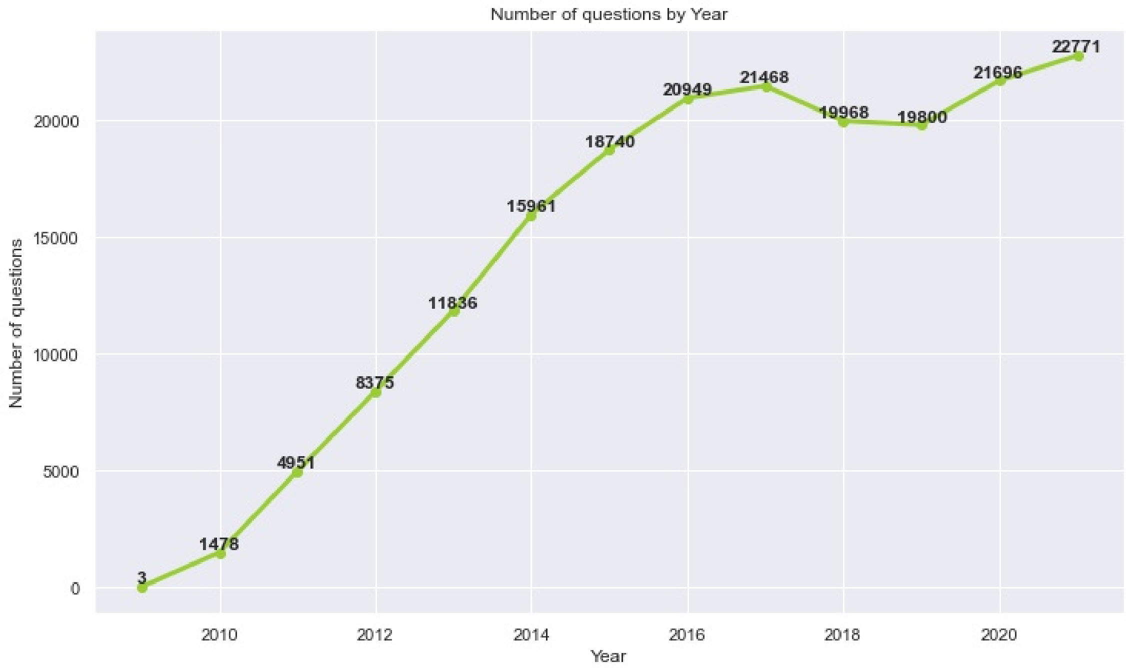
Out[32]:

	Year	Count
0	2009	3
1	2010	1478
2	2011	4951
3	2012	8375
4	2013	11836
5	2014	15961
6	2015	18740
7	2016	20949
8	2017	21468
9	2018	19968
10	2019	19800
11	2020	21696
12	2021	22771

Let us also visualise the above.

In [51]:
```
fig, ax = plt.subplots(figsize=(12, 7))
```

```
plt.plot('Year', 'Count', data=questions_by_year, marker='o', color='yellowgreen', linewidth=3)
plt.title('Number of questions by Year')
plt.ylabel('Number of questions')
plt.xlabel('Year')

for line in range(0, questions_by_year.shape[0]):
     plt.annotate(
          questions_by_year.Count[line],
          (questions_by_year.Year[line], questions_by_year.Count[line]+5),
          fontsize=12,
          weight='bold',
          va='bottom',
          ha='center'
     )

plt.show()
```



As predicted, the number of questions have generally increased with a slight dip in the years 2018 and 2019.

Question Tags over time

Now that we know that the number of questions being posted on stats.stackexchange.com have increased over time, let us also try to find out how the topics (*aka tags*) have evolved in the same time frame.

For this, we will create a new dataframe **posts_tagsANDyear** and extract only the relevant columns.

In [53]:

```
posts_tagsANDyear = pd.DataFrame()
```

In [54]:

```
posts_tagsANDyear['Body'], posts_tagsANDyear['Tags'], posts_tagsANDyear['Year'] = posts_questions['Body'], posts_
```

In [56]:

```
posts_tagsANDyear.head()
```

Out[56]:

	Body	Tags	Year
0	<p>How should I elicit prior distributions fro...	<bayesian><prior><elicitation>	2010
1	<p>In many different statistical methods there...	<distributions><normality-assumption>	2010
2	<p>What are some valuable Statistical Analysis...	<software><open-source>	2010
3	<p>I have two groups of data. Each with a dif...	<distributions><statistical-significance>	2010
5	<p>Last year, I read a blog post from <a href=...	<machine-learning><pac-learning>	2010

We see that there are multiple tags associated with any given question.

You might also remember that *Tags* data type is **object**.

Let's do some further data wrangling so that we have only one tag per row.

In [57]:

```
import re

posts_tagsANDyear_modified = posts_tagsANDyear.assign(Tags=posts_tagsANDyear['Tags'].str.findall(r"(?<=\<)(.*?)(?
```

In [58]:

```
posts_tagsANDyear_modified.head(20)
```

Out[58]:

	Body	Tags	Year
0	<p>How should I elicit prior distributions fro...	bayesian	2010
0	<p>How should I elicit prior distributions fro...	prior	2010
0	<p>How should I elicit prior distributions fro...	elicitation	2010
1	<p>In many different statistical methods there...	distributions	2010
1	<p>In many different statistical methods there...	normality-assumption	2010
2	<p>What are some valuable Statistical Analysis...	software	2010
2	<p>What are some valuable Statistical Analysis...	open-source	2010
3	<p>I have two groups of data. Each with a dif...	distributions	2010
3	<p>I have two groups of data. Each with a dif...	statistical-significance	2010
5	<p>Last year, I read a blog post from <a href=...	machine-learning	2010
5	<p>Last year, I read a blog post from <a href=...	pac-learning	2010
6	<p>I've been working on a new method for analy...	dataset	2010
6	<p>I've been working on a new method for analy...	sample	2010
6	<p>I've been working on a new method for analy...	population	2010
6	<p>I've been working on a new method for analy...	teaching	2010
7	<p>Sorry, but the emptyness was a bit overwhel...	humor	2010
9	<p>Many studies in the social sciences use Lik...	ordinal-data	2010
9	<p>Many studies in the social sciences use Lik...	likert	2010
9	<p>Many studies in the social sciences use Lik...	scales	2010
9	<p>Many studies in the social sciences use Lik...	measurement	2010

This looks better. We will also *reset index*.

In [59]:

```
posts_tagsANDyear_modified.reset_index(drop=True)
```

Out[59]:

	Body	Tags	Year
0	<p>How should I elicit prior distributions fro...	bayesian	2010
1	<p>How should I elicit prior distributions fro...	prior	2010
2	<p>How should I elicit prior distributions fro...	elicitation	2010
3	<p>In many different statistical methods there...	distributions	2010
4	<p>In many different statistical methods there...	normality-assumption	2010
...	...	...	...
575812	<p>I'm working on a project which request to d...	predictive-models	2021
575813	<p>I'm working on a project which request to d...	model	2021
575814	<p>When I read up the example for Bayes Theore...	bayesian	2021
575815	<p>I want to train a model that takes image as...	neural-networks	2021
575816	<p>I want to train a model that takes image as...	conv-neural-network	2021

575817 rows × 3 columns

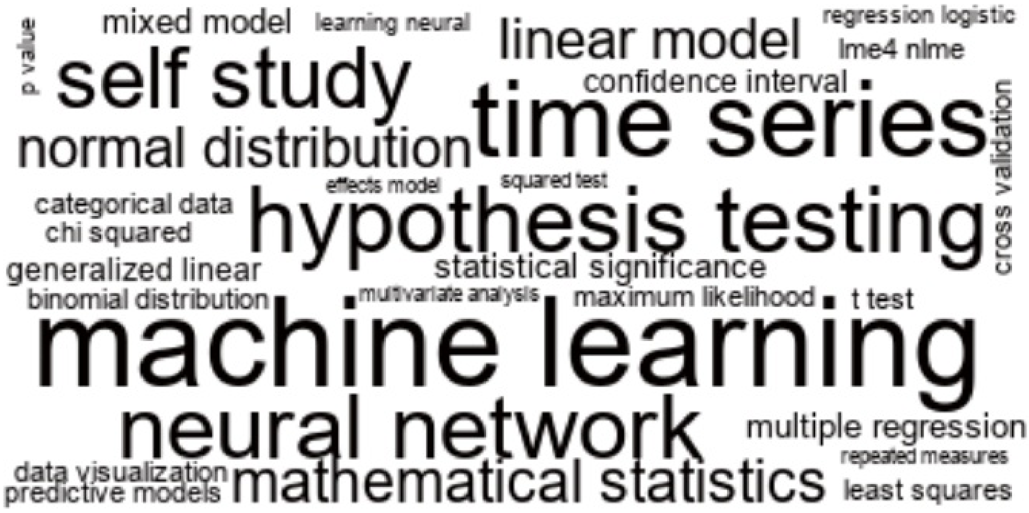
In [86]:

```
from wordcloud import WordCloud

def black_color_func(word, font_size, position, orientation, random_state=None, **kwargs):
    return("hsl(0,100%, 1%)")

tags_wc = WordCloud(font_path = './arial.ttf', max_words=30, background_color='white').generate(' '.join(posts_ta
tags_wc.recolor(color_func = black_color_func)
```

<wordcloud.wordcloud.WordCloud at 0x1a1a2ff3e80>

Out[86]: <wordcloud.wordcloud.WordCloud at 0x1a1a2[illegible]>

In [87]:
```
plt.figure(figsize=[12,10])

plt.imshow(tags_wc, interpolation="bilinear")
plt.axis("off")
plt.show()
```



In [60]:
```
tags_by_year = posts_tagsANDyear_modified.groupby(['Year', 'Tags'])['Year'].size().reset_index(name='Count')

tags_by_year
```

Out[60]:

	Year	Tags	Count
0	2009	cooks-distance	1
1	2009	data-visualization	1
2	2009	graph-theory	1
3	2009	moving-average	1
4	2009	outliers	2
...	...	...	...
12843	2021	z-score	37
12844	2021	z-statistic	3
12845	2021	z-test	34
12846	2021	zero-inflation	49
12847	2021	zipf	3

12848 rows × 3 columns

References

1. https://www.codegrepper.com/code-examples/sql/jupyter+notebook+mysql+connector+to+dataframe
2. https://stackoverflow.com/questions/37730243/importing-data-from-a-mysql-database-into-a-pandas-data-frame-including-column-n/37730334
3. https://python-graph-gallery.com/25-histogram-with-several-variables-seaborn
4. https://stackoverflow.com/questions/62351154/how-to-get-different-titles-for-each-of-the-subplots-in-seaborn
5. https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.sample.html
6. https://stackoverflow.com/questions/30228069/how-to-display-the-value-of-the-bar-on-each-bar-with-pyplot-barh
7. https://www.geeksforgeeks.org/how-to-annotate-matplotlib-scatter-plots/
8. https://www.analyticsvidhya.com/blog/2021/08/creating-customized-word-cloud-in-python/