# Research of the dependence of behavioral indicators on Core

# Web Vitals indicators

Optimizing for quality of user experience is key to the long-term success of any site on the web. Whether you're a business owner, marketer, or developer, Web Vitals can help you quantify the experience of your site and identify opportunities to improve.
The Web Vitals initiative aims to simplify the landscape, and help sites focus on the metrics that matter most, the Core Web Vitals. Core Web Vitals are the subset of Web Vitals that apply to all web pages, should be measured by all site owners, and will be surfaced across all Google tools. Each of the Core Web Vitals represents a distinct facet of the user experience, is measurable in the field, and reflects the real-world experience of a critical user-centric outcome.

For this analysis, we teamed up with WebCEO, a leading SEO platform.

---

In [2]:

```
#Importing necessary Python libraries for data analysis and visualisation
import numpy as np # linear algebra
import pandas as pd # data processing
import matplotlib.pyplot as plt #data visualisation
import seaborn as sns #data visualisation
import missingno as miss #data visualisation

from IPython.display import Image #images import
from IPython.core.display import HTML #images import
from IPython.display import display #images import
from scipy.stats import kstest
from scipy.stats import spearmanr

print('Libraries imported')
```

```
Libraries imported
```

The metrics that make up Core Web Vitals will evolve over time. The current set for 2021 focuses on three aspects of the user experience—loading, interactivity, and visual stability—and includes the following metrics (and their respective thresholds):

---

In [3]:

```
#Importing images
x = Image(url= "https://webdev.imgix.net/vitals/lcp_ux.svg", width=200, height=175)
y = Image(url= "https://webdev.imgix.net/vitals/fid_ux.svg", width=200, height=175)
z = Image(url= "https://webdev.imgix.net/vitals/cls_ux.svg", width=200, height=175)
display(x, y, z)
```

- Largest Contentful Paint (LCP): measures loading performance. To provide a good user experience, LCP should occur within 2.5 seconds of when the page first starts loading.
- First Input Delay (FID): measures interactivity. To provide a good user experience, pages should have a FID of less than 100 milliseconds.
- Cumulative Layout Shift (CLS): measures visual stability. To provide a good user experience, pages should maintain a CLS of less than 0.1.

For each of the above metrics, to ensure you're hitting the recommended target for most of your users, a good threshold to measure is the 75th percentile of page loads, segmented across mobile and desktop devices.

As main behavioral indicators we've choosed Bounce Rate, Time on Site and Pages Per Session(or Page Depth) metrics

- Bounce Rate is single-page sessions divided by all sessions or the percentage of all sessions on your site in which users viewed only a single page and triggered only a single request to the Analytics server. In other words, it collects all sessions where a visitor only visited one page and divides it by all sessions.
- Time on Site is the amount of time (in seconds) visitors have spent on your website.
- The Page Depth metric or Pages per Session (also pages/session or average page depth) indicates the average number of pages visited by a user within a session.

We will research the relation of Core Web Vitals indicators with behavioral metrics on the website. In other words, how loading speed, interactivity, and visual stability of the website impact on user experience.

# Import the data

We analyzed 20K projects because we only needed those which met all our requirements: access to Google Analytics, Core Web Vitals data available, etc.

---

In [4]:

```
#importing the dataset
df = pd.read_csv('t39974_pages.csv', sep=';')
df = df.drop(['page_id','domain_id'], axis=1)

#Add column with average time on page. Time on Page column shows total time
df['duration'] = df.time_on_page / df.sessions

#Drop pages with less than 10 session
df = df[df['sessions'] > 10]
```

---

In [5]:

```
#the number of rows in the dataset
df.shape
```

Out[5]:

(136519, 9)

---

In [6]:

```
missing_values_index = df[df['duration'].isnull()].index.tolist()
df = df.drop(missing_values_index)

missing_values_index = df[df['lcp'].isnull()].index.tolist()
df = df.drop(missing_values_index)
df = df[df['fid'].notna()]
df = df.drop(['fcp'], axis=1)

df = df.round(2)
```

# Data preprocessing

Categorizing data into 3 segments by Google's scales by Core Web Vitals parameteres

---

In [7]:

```
#Creating the dataset with statistics by scale(GOOD, NEED IMPROVEMENT, POOR) by Core Web Vitals parameters
COLORS = {'GOOD': 'green', 'NEEDS IMPROVEMENT': 'orange', 'POOR': 'red'}

def cls(item):
    if item < 0.1:
        return 'GOOD'
    if item > 0.25:
        return 'POOR'
    return 'NEEDS IMPROVEMENT'

def lcp(item):
    if item > 4000:
        return 'POOR'
    if item < 2500:
        return 'GOOD'
    return 'NEEDS IMPROVEMENT'
```

```
def fid(item):
    if item > 300:
        return 'POOR'
    if item < 100:
        return 'GOOD'
    return 'NEEDS IMPROVEMENT'

frame=pd.DataFrame({
    'lcp': df.lcp.apply(lcp).value_counts(normalize=True).apply(lambda v: round(100 * v, 2)),
    'cls': df.cls.apply(cls).value_counts(normalize=True).apply(lambda v: round(100 * v, 2)),
    'fid': df.fid.apply(fid).value_counts(normalize=True).apply(lambda v: round(100 * v, 2))
})

#Creating 3 more columns in the dataset with categorisation(GOOD, NEED IMPROVEMENT, POOR) by parameters
df['cls_category'] = df['cls'].apply(lambda row: cls(row))
df['fid_category'] = df['fid'].apply(lambda row: fid(row))
df['lcp_category'] = df['lcp'].apply(lambda row: lcp(row))
```

---

In [8]:

```
df.to_csv (r'/Users/User/Downloads/dataframe.csv', index = False, header=True)
frame.to_csv (r'/Users/User/Downloads/categories.csv', index = False, header=True)
```

## Data distribution in columns

---

In [9]:

```
sns.pairplot(df, diag_kind="kde")

plt.show()
```

---

In [ ]:

So, the dataset is ready. The main goal of our research it to check relations between Core Web Vitals and behavioral metrics. We want to find out, do they really impact on user experience on the website or not and how exactly Core Web Vitals correlate with users' behaviour

# Cumulative Layout Shift

CLS measures the sum total of all individual layout shift scores for every unexpected layout shift that occurs during the entire lifespan of the page. A layout shift occurs any time a visible element changes its position from one rendered frame to the next.

## Layout shift score

To calculate the layout shift score, the browser looks at the viewport size and the movement of unstable elements in the viewport between two rendered frames. The layout shift score is a product of two measures of that movement: the impact fraction and the distance fraction.
The union of the visible areas of all unstable elements for the previous frame and the current frame—as a fraction of the total area of the viewport—is the impact fraction for the current frame.
The distance fraction is the greatest distance any unstable element has moved in the frame (either horizontally or vertically) divided by the viewport's largest dimension (width or height, whichever is greater)
The layout shift score is a product of two measures of that movement: the impact fraction and the distance fraction (both defined below).

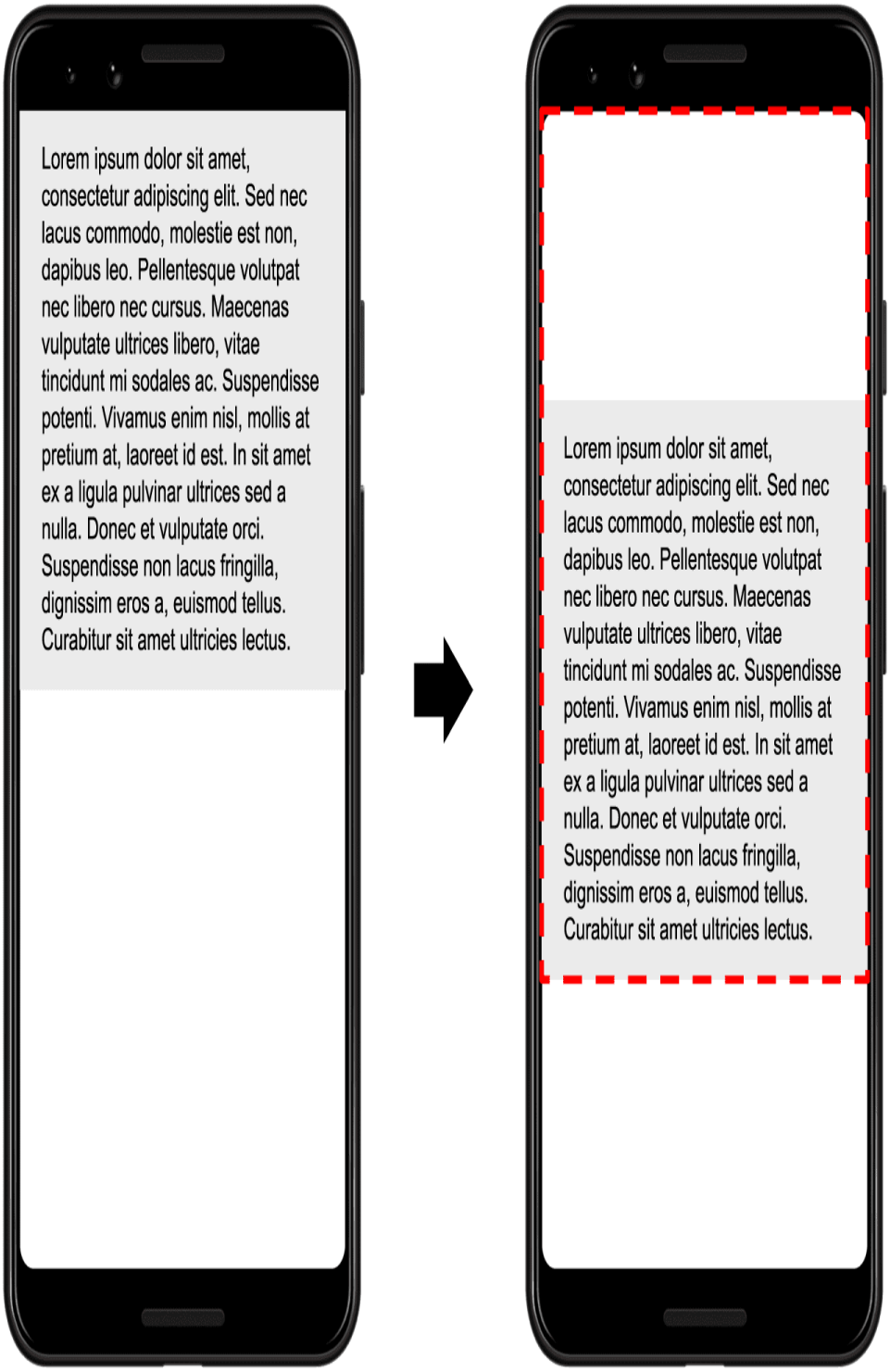
layout shift score = impact fraction * distance fraction

- The impact fraction measures how unstable elements impact the viewport area between two frames.The union of the visible areas of all unstable elements for the previous frame and the current frame—as a fraction of the total area of the viewport—is the impact fraction for the current frame.
- The other part of the layout shift score equation measures the distance that unstable elements have moved, relative to the viewport. The distance fraction is the greatest distance any unstable element has moved in the frame (either horizontally or vertically) divided by the viewport's largest dimension (width or height, whichever is greater).

---

In [10]:

```
display(Image(url= "https://webdev.imgix.net/cls/layout-shift-1.png", height = 800, width=600))
```




**What is a good CLS score?**

To provide a good user experience, sites should strive to have a CLS score of less than 0.1. To ensure you're hitting this target for most of your users, a good threshold to measure is the 75th percentile of page loads, segmented across mobile and desktop devices.

---

In [11]:

```
display(Image(url= "https://web.dev/vitals/cls_8x2.svg"))
```

## See the scores received by the websites we've analyzed:

Let's see how our projects are arranged by this classification.

---

In [12]:

```
frame.cls
```

Out[12]:

```
GOOD                65.13
NEEDS IMPROVEMENT   17.03
POOR                17.84
Name: cls, dtype: float64
```

---

In [13]:

```
ax = frame[['cls']].T.plot.barh(rot=0, stacked=True, figsize=(15, 2), color=COLORS)
ax.set_xlim([0, 100])
ax.set_xlabel("%")
```

Out[13]:

```
Text(0.5, 0, '%')
```

More than 60% of the projects have Good CLS score. Poor and Need Improvement segments have ±17% share from the total amount of projects

## Correlation between CLS and behavioral metrics

We will visualise the distribution of our projects on two axes - one of the Core Web Vitals metric and all three behavioral metrics in course to see is there correlation visually.
We're going to use two types of plots - a kernel density estimate (KDE) plot and scatter plot.
A scatter plot (aka scatter chart, scatter graph) uses dots to represent values for two different numeric variables. The position of each dot on the horizontal and vertical axis indicates values for an individual project. Scatter plots are used to observe relationships between variables.
We will see the distribution of all projects by two metrics, how they are distributed uniformly and whether there is a pattern of distribution or not.
A KDE plot is a method for visualizing the distribution of observations in a dataset, analagous to a histogram. KDE represents the data using a continuous probability density curve. Relative to a histogram, KDE can produce a plot that is less cluttered and more interpretable, especially when drawing multiple distributions.
We divide our dataset into 3 groups by each Core Web Vitals, based on Google interpretation - Poor, Need improvement and Good. Then we visualise this three distributions on KDE to see how these clusters distribute by behavioral metric.

---

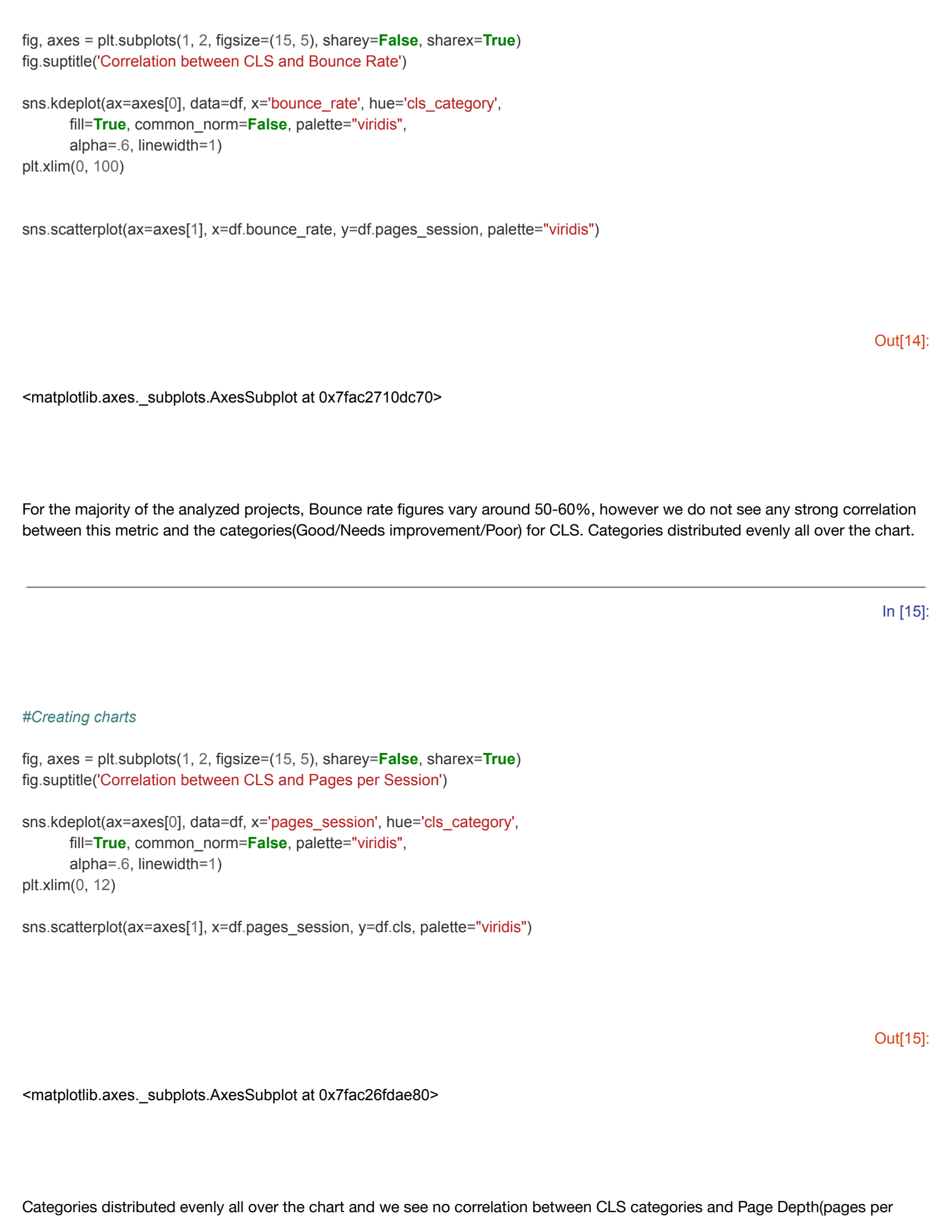
In [14]:

```
#Creating charts
```

```
fig, axes = plt.subplots(1, 2, figsize=(15, 5), sharey=False, sharex=True)
fig.suptitle('Correlation between CLS and Bounce Rate')

sns.kdeplot(ax=axes[0], data=df, x='bounce_rate', hue='cls_category',
        fill=True, common_norm=False, palette="viridis",
        alpha=.6, linewidth=1)
plt.xlim(0, 100)


sns.scatterplot(ax=axes[1], x=df.bounce_rate, y=df.pages_session, palette="viridis")
```

Out[14]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fac2710dc70>
```



For the majority of the analyzed projects, Bounce rate figures vary around 50-60%, however we do not see any strong correlation between this metric and the categories(Good/Needs improvement/Poor) for CLS. Categories distributed evenly all over the chart.

---

In [15]:

```
#Creating charts

fig, axes = plt.subplots(1, 2, figsize=(15, 5), sharey=False, sharex=True)
fig.suptitle('Correlation between CLS and Pages per Session')

sns.kdeplot(ax=axes[0], data=df, x='pages_session', hue='cls_category',
        fill=True, common_norm=False, palette="viridis",
        alpha=.6, linewidth=1)
plt.xlim(0, 12)

sns.scatterplot(ax=axes[1], x=df.pages_session, y=df.cls, palette="viridis")
```

Out[15]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fac26fdae80>
```



Categories distributed evenly all over the chart and we see no correlation between CLS categories and Page Depth(pages per

session) metrics.

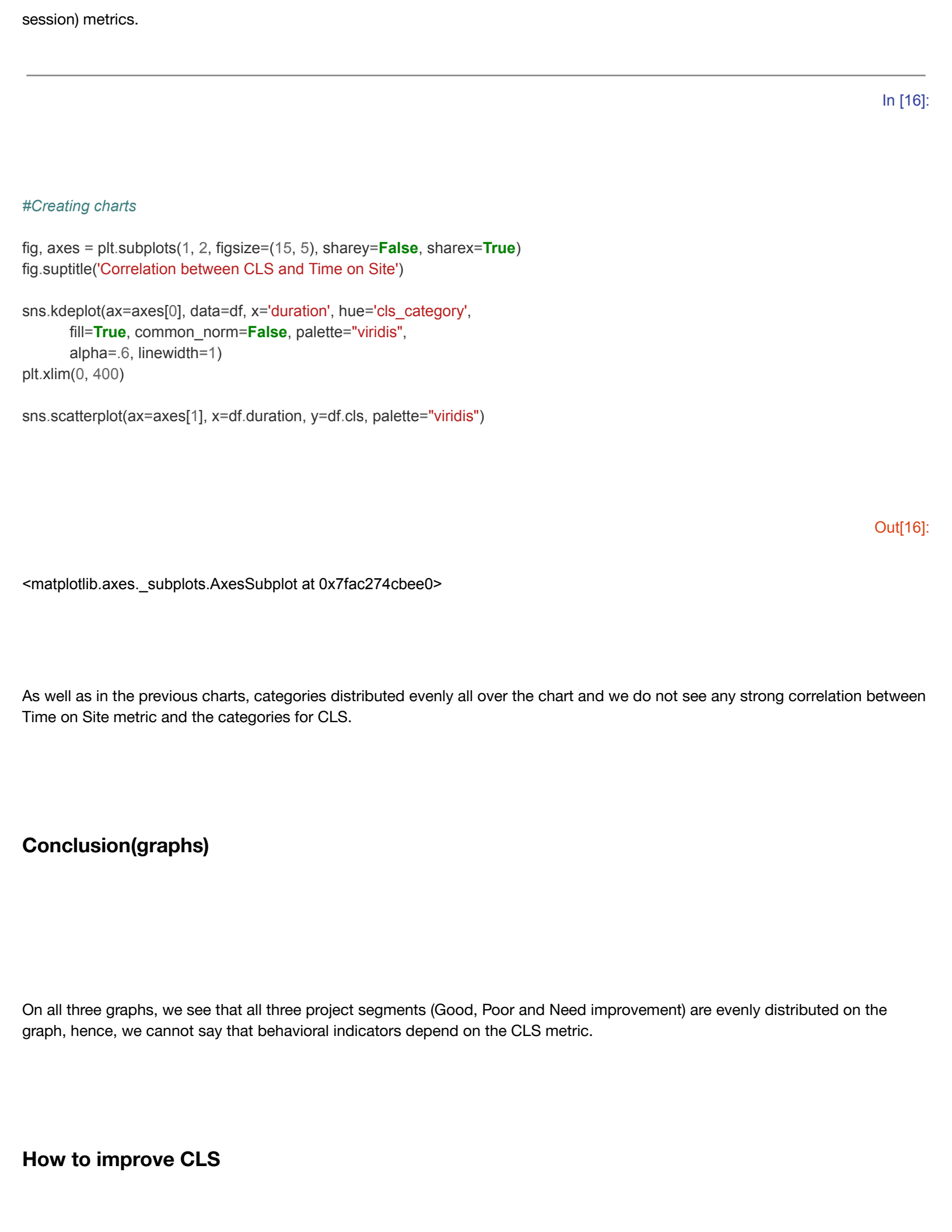
---

In [16]:

```
#Creating charts

fig, axes = plt.subplots(1, 2, figsize=(15, 5), sharey=False, sharex=True)
fig.suptitle('Correlation between CLS and Time on Site')

sns.kdeplot(ax=axes[0], data=df, x='duration', hue='cls_category',
        fill=True, common_norm=False, palette="viridis",
        alpha=.6, linewidth=1)
plt.xlim(0, 400)

sns.scatterplot(ax=axes[1], x=df.duration, y=df.cls, palette="viridis")
```

Out[16]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fac274cbee0>
```



As well as in the previous charts, categories distributed evenly all over the chart and we do not see any strong correlation between Time on Site metric and the categories for CLS.

## Conclusion(graphs)

On all three graphs, we see that all three project segments (Good, Poor and Need improvement) are evenly distributed on the graph, hence, we cannot say that behavioral indicators depend on the CLS metric.

## How to improve CLS

For most websites, you can avoid all unexpected layout shifts by sticking to a few guiding principles:

- Always include size attributes on your images and video elements, or otherwise reserve the required space with something like CSS aspect ratio boxes. This approach ensures that the browser can allocate the correct amount of space in the document while the image is loading. Note that you can also use the unsized-media feature policy to force this behavior in browsers that support feature policies.
- Never insert content above existing content, except in response to a user interaction. This ensures any layout shifts that occur are expected.
- Prefer transform animations to animations of properties that trigger layout changes. Animate transitions in a way that provides context and continuity from state to state.

# First Input Delay (FID)

First Input Delay (FID) is an important, user-centric metric for measuring load responsiveness because it quantifies the experience users feel when trying to interact with unresponsive pages—a low FID helps ensure that the page is usable.

## What is FID?

FID measures the time from when a user first interacts with a page (i.e. when they click a link, tap on a button, or use a custom, JavaScript-powered control) to the time when the browser is actually able to begin processing event handlers in response to that interaction.

## FID in detail

FID only measures the "delay" in event processing. It does not measure the event processing time itself nor the time it takes the browser to update the UI after running event handlers.
Even though this time is important to the user and does affect the experience, it's not included in this metric because doing so could incentivize developers to add workarounds that actually make the experience worse—that is, they could wrap their event handler logic in an asynchronous callback (via setTimeout() or requestAnimationFrame()) in order to separate it from the task associated with the event. The result would be an improvement in the metric score but a slower response as perceived by the user.

---

In [17]:

```
display(Image(url= "https://webdev.imgix.net/fid/fid-full.svg"))
```

The above visualization shows a page that's making a couple of network requests for resources (most likely CSS and JS files), and—after those resources are finished downloading—they're processed on the main thread.
This results in periods where the main thread is momentarily busy, which is indicated by the beige-colored task blocks.
Long first input delays typically occur between First Contentful Paint (FCP) and Time to Interactive (TTI) because the page has rendered some of its content but isn't yet reliably interactive. To illustrate how this can happen, FCP and TTI have been added to the timeline.
Because the input occurs while the browser is in the middle of running a task, it has to wait until the task completes before it can respond to the input. The time it must wait is the FID value for this user on this page.

## What is a good FID score?

To provide a good user experience, sites should strive to have a First Input Delay of less than 100 milliseconds.

---

In [18]:

```
display(Image(url= "https://web.dev/vitals/fid_8x2.svg"))
```

## See the scores received by the websites we've analyzed:

Let's see how our projects are arranged by this classification.

---

In [19]:

```
frame.fid
```

Out[19]:

```
GOOD                53.85
NEEDS IMPROVEMENT    37.58
POOR                8.57
Name: fid, dtype: float64
```

---

In [20]:

```
ax = frame[['fid']].T.plot.barh(rot=0, stacked=True, figsize=(15, 2), color=COLORS)
ax.set_xlim([0, 100])
ax.set_xlabel("%")
```

Out[20]:

```
Text(0.5, 0, '%')
```

More than 50% projects have a Good FID score. The other segments (Poor and Needs improvement) account for less than 0.5%. This means that sites have almost no delay between receiving user input and starting to process the request.

## Correlation between First Input Delay and behavioral metrics

---

In [21]:

```
#Creating charts

fig, axes = plt.subplots(1, 2, figsize=(15, 5), sharey=False, sharex=True)
fig.suptitle('Correlation between FID and Bounce Rate')

sns.kdeplot(ax=axes[0], data=df, x='bounce_rate', hue='fid_category',
        fill=True, common_norm=False, palette="viridis",
        alpha=.6, linewidth=1)
plt.xlim(0, 100)

sns.scatterplot(ax=axes[1], x=df.bounce_rate, y=df.fid, palette="viridis")
plt.ylim(0, 3000)
```

Out[21]:

```
(0.0, 3000.0)
```

---

In [22]:

```
#Creating charts

fig, axes = plt.subplots(1, 2, figsize=(15, 5), sharey=False, sharex=True)
fig.suptitle('Correlation between FID and Pages per Session')

sns.kdeplot(ax=axes[0], data=df, x='pages_session', hue='fid_category',
        fill=True, common_norm=False, palette="viridis",
        alpha=.6, linewidth=1)
plt.xlim(0, 15)

sns.scatterplot(ax=axes[1], x=df.pages_session, y=df.fid, palette="viridis")
plt.ylim(0, 3000)
```

Out[22]:

```
(0.0, 3000.0)
```

---

In [23]:

```
#Creating charts

fig, axes = plt.subplots(1, 2, figsize=(15, 5), sharey=False, sharex=True)
fig.suptitle('Correlation between FID and Time on Site')

sns.kdeplot(ax=axes[0], data=df, x='duration', hue='fid_category',
        fill=True, common_norm=False, palette="viridis",
        alpha=.6, linewidth=1)
plt.xlim(0, 350)

sns.scatterplot(ax=axes[1], x=df.duration, y=df.fid, palette="viridis")
plt.ylim(0, 3000)
```

Out[23]:

```
(0.0, 3000.0)
```

## Conclusion

We must not forget that the graph does not show absolute quantities, but the density relative to this segment. Although we see some pattern of the FID indicator segments, but since the Need improvements and Poor segments have a total share of less than 0.5%, we cannot visually determine whether there is a dependence. It can be determined statistically.

### How to improve FID

- Reduce the impact of third-party code
- Reduce JavaScript execution time
- Minimize main thread work
- Keep request counts low and transfer sizes small

## Largest Contentful Paint (LCP)

Largest Contentful Paint (LCP) is an important, user-centric metric for measuring perceived load speed because it marks the point in the page load timeline when the page's main content has likely loaded—a fast LCP helps reassure the user that the page is useful.

### What is LCP?

The Largest Contentful Paint (LCP) metric reports the render time of the largest image or text block visible within the viewport.

---

In [24]:

```
display(Image(url= "https://web.dev/vitals/lcp_8x2.svg"))
```

## What is a good LCP score?

To provide a good user experience, sites should strive to have Largest Contentful Paint occur within the first 2.5 seconds of the page starting to load. To ensure you're hitting this target for most of your users, a good threshold to measure is the 75th percentile of page loads, segmented across mobile and desktop devices.

## What elements are considered?

The types of elements considered for Largest Contentful Paint are:

- < img > elements
- < image > elements inside an element
- < video > elements (the poster image is used)
- An element with a background image loaded via the url() function (as opposed to a CSS gradient)
- Block-level elements containing text nodes or other inline-level text elements children.

## Examples

Here are some examples of when the Largest Contentful Paint occurs on a few popular websites:

---

In [25]:

```
display(Image(url= "https://webdev.imgix.net/lcp/lcp-instagram-filmstrip.png"))
```

---

In [26]:

```
display(Image(url= "https://webdev.imgix.net/lcp/lcp-google-filmstrip.png"))
```

In the first example, the Instagram logo is loaded relatively early and it remains the largest element even as other content is progressively shown. In the Google search results page example, the largest element is a paragraph of text that is displayed before any of the images or logo finish loading. Since all the individual images are smaller than this paragraph, it remains the largest element throughout the load process.

## See the scores received by the websites we've analyzed:

Let's see how our projects are arranged by this classification.

---

In [27]:

frame.lcp

Out[27]:

```
GOOD                53.77
NEEDS IMPROVEMENT   28.76
POOR                17.47
Name: lcp, dtype: float64
```

---

In [28]:

```
ax = frame[['lcp']].T.plot.barh(rot=0, stacked=True, figsize=(15, 2), color=COLORS)
ax.set_xlim([0, 100])
ax.set_xlabel("%")
```

Out[28]:

Text(0.5, 0, '%')

More than half of the projects have a Good LCP indicator

## Correlation between Largest Contentful Paint and behavioral metrics

---

In [29]:

```
#Creating charts

fig, axes = plt.subplots(1, 2, figsize=(15, 5), sharey=False, sharex=True)
fig.suptitle('Correlation between CLS and Bounce Rate')

sns.kdeplot(ax=axes[0], data=df, x='bounce_rate', hue='lcp_category',
        fill=True, common_norm=False, palette="viridis",
        alpha=.6, linewidth=1)
plt.xlim(0, 100)

sns.scatterplot(ax=axes[1], x=df.bounce_rate, y=df.lcp, palette="viridis")
plt.ylim(0, 30000)
```

Out[29]:

```
(0.0, 30000.0)
```

---

In [30]:

```
#Creating charts

fig, axes = plt.subplots(1, 2, figsize=(15, 5), sharey=False, sharex=True)
fig.suptitle('Correlation between CLS and Pages per Session(Page depth)')

sns.kdeplot(ax=axes[0], data=df, x='pages_session', hue='lcp_category',
        fill=True, common_norm=False, palette="viridis",
        alpha=.6, linewidth=1)
plt.xlim(0, 12)

sns.scatterplot(ax=axes[1], x=df.pages_session, y=df.lcp, palette="viridis")
plt.ylim(0, 30000)
```

Out[30]:

```
(0.0, 30000.0)
```

---

In [31]:

```
#Creating charts

fig, axes = plt.subplots(1, 2, figsize=(15, 5), sharey=False, sharex=True)
fig.suptitle('Correlation between CLS and Time on Site')

sns.kdeplot(ax=axes[0], data=df, x='duration', hue='lcp_category',
        fill=True, common_norm=False, palette="viridis",
        alpha=.6, linewidth=1)
plt.xlim(0, 400)

sns.scatterplot(ax=axes[1], x=df.duration, y=df.lcp, palette="viridis")
plt.ylim(0, 30000)
```

Out[31]:

```
(0.0, 30000.0)
```

## Conclusion

On all three graphs, we see that all three website segments (Good, Poor and Needs improvement) are evenly distributed on the graph, that is, we cannot say that behavioral indicators depend on the LCP metric

## How to improve LCP

- Apply instant loading with the PRPL pattern
- Optimizing the Critical Rendering Path
- Optimize your CSS
- Optimize your Images
- Optimize web Fonts
- Optimize your JavaScript (for client-rendered sites)

In [ ]:

In [ ]:

# Correlation check by statistical methods

Since we haven't seen the relationship between the indicator and behavioral indicators visually, we will check the relationship with statistical methods

As far as the real distribution of the variable does not correspond to the normal (Gaussian), uniform, exponential distribution we use Spearman's correlation - non-parametric test that is used to measure the relation between two variables.

## Heatmap of the Spearman's correlation between metrics

In [34]:

```
df = df.drop(['sessions', 'time_on_page'], axis=1)
df['avg_time_on_page'] = df['duration']
df = df.drop(['duration'], axis=1)
```

---

In [38]:

```
cols = ['bounce_rate', 'pages_session', 'avg_time_on_page', 'fid', 'lcp', 'cls', 'cls_category',
    'fid_category', 'lcp_category']
df = df[cols]
df
```

Out[38]:

| | bounce_rate | pages_session | avg_time_on_page | fid | lcp | cls | cls_category | fid_category | lcp_category |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 0.76 | 2.17 | 163.80 | 16.0 | 915.0 | 0.00 | GOOD | GOOD | GOOD |
| **1** | 26.87 | 2.67 | 116.90 | 16.0 | 1035.0 | 0.01 | GOOD | GOOD | GOOD |
| **2** | 41.03 | 2.13 | 142.74 | 21.0 | 710.0 | 0.03 | GOOD | GOOD | GOOD |
| **3** | 84.06 | 1.17 | 1.33 | 54.0 | 1705.5 | 0.00 | GOOD | GOOD | GOOD |
| **4** | 100.00 | 1.00 | 0.00 | 43.0 | 1722.5 | 0.00 | GOOD | GOOD | GOOD |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **208075** | 29.20 | 2.32 | 76.69 | 221.5 | 4816.0 | 0.65 | POOR | NEEDS IMPROVEMENT | POOR |
| **208077** | 3.51 | 8.92 | 145.60 | 1139.0 | 5124.5 | 0.00 | GOOD | POOR | POOR |
| **208078** | 5.56 | 3.72 | 165.14 | 1166.0 | 3406.0 | 0.03 | GOOD | POOR | NEEDS IMPROVEMENT |
| **208079** | 33.02 | 1.29 | 445.63 | 16.0 | 795.0 | 0.00 | GOOD | GOOD | GOOD |
| **208083** | 96.14 | 1.11 | 12.42 | 137.0 | 1795.5 | 0.00 | GOOD | NEEDS IMPROVEMENT | GOOD |

136347 rows × 9 columns

In [39]:

```
#Creating a correlations' heatmap

f, ax = plt.subplots(figsize=(9, 6))
matrix = np.triu(df.corr(method = 'spearman'))
ax = sns.heatmap(df.corr(method = 'spearman'), center=0,
                 annot=True,
                 linewidths=.5,
                 ax=ax,
                 mask=matrix,
                 cmap=sns.diverging_palette(20, 220, n=200),)
```

When using the coefficient of rank correlation, conditionally assess the tightness of the relations between the metrics, considering the value of the coefficient:

- less than 0.3 - a sign of weak correlation,
- values of more than 0.3, but less than 0.7 - a sign of moderate correlation,
- values of 0.7 and more - a sign of high correlation.

Therefore we can conclude that there is:

- strong correlation between Bounce Rate and Page Depth metrics;
- strong correlation between Time on Site and Page Depth metrics;
- strong correlation between Bounce Rate and Time on Site metrics;
- almost no correlation between other parameters.

# Example

Even if Core Web Vitals are not directly related to behavioral metrics, it will still become a ranking factor in May 2021. Therefore, to have these indicators low - may be tantamount to the loss of ranks
Let us show you with an applied example how to view these data, interpret them correctly, and what to do if they are not within the

recommended norm.
Because you want both metrics and the instances where they happened and the tips on how to fix them - we will analyze a real website in the WebCEO interface

As an example, we've analysed imdb.com. Here are the resuls of the field data over the last 30 days.

---

In [39]:

```
display(Image('/Users/User/Downloads/field_data.png'))
```

## 1. Speed up visual loading and improve Largest Contentful Paint(LCP)

One of the first indicators of page loading for the user is the appearance of the content at the top of the page.
This is where Largest Contentful Paint (LCP) is measured, the first metric in Core Web Vitals. It shows how fast the main element on the page is loading.
To determine what this element is, check the page in Chrome DevTools and it will appear in a waterfall chart under the Performance tab.

---

In [40]:

```
display(Image('/Users/User/Downloads/lcp.png'))
```

Select Screenshots and start profiling the page while it loads by clicking the record button. When the page profile is created, when you hover over the loading chart at the top, you will see a screenshot of the page loading over time. This will help you visualize how quickly the various elements of the page are loading.

To speed up the loading of the LCP element and content on the first screen, consider using techniques such as preloading so that browsers get these resources first.

---

In [41]:

display(Image(url='https://d.searchengines.guru/20/61/image004-5fa19a8cd4d34-1024x254__73055e76.png'))

```
<head>
  <meta charset="utf-8">
  <title>Video preload example</title>

  <link rel="preload" href="sintel-short.mp4" as="video" type
  <link rel="preload" href="sintel-short.webm" as="video" typ
</head>
```

You can find next particular issues that affect LCP metric in the WebCEO report:

- Properly size images. Serve images that are appropriately-sized to save cellular data and improve load time.
- Avoid chaining critical requests. The Critical Request Chains below show you what resources are loaded with a high priority. Consider reducing the length of chains, reducing the download size of resources, or deferring the download of unnecessary resources to improve page load.
- Largest Contentful Paint element. This is the largest contentful element painted within the viewport.

---

In [42]:

display(Image('/Users/User/Downloads/lcp_issue.png'))

## 2. Optimize the activity of the main thread and decrease First Input Delay (FID) metric

There are many hidden issues that cause the user to wait for the browser to respond to a click or click on a page. And that's what the second Core Web Vitals metric, First Input Delay (FID), measures.
While this experience can be frustrating for users, there are some things we can do to fix the problem and reduce the latency between user actions and browser responses.
Long tasks are a common cause of these problems. Basically, these are pieces of JavaScript that block the main thread for a long period of time and cause the page to freeze and stop responding.
In Chrome DevTools, long tasks can be found at the top of the waterfall chart under the Main tab. They will be highlighted with a red triangle.

---

In [43]:

```
display(Image('/Users/User/Downloads/fid.png'))
```

The required fix will vary depending on the actions that block the main thread, but the usual fix for dealing with long tasks is to split code and load scripts in smaller chunks.

You can find next particular issues that affect FID metric in the WebCEO report:

- Remove unused JavaScript. Remove unused JavaScript to reduce bytes consumed by network activity.
- Avoid chaining critical requests. Consider reducing the length of chains, reducing the download size of resources, or deferring the download of unnecessary resources to improve page load.
- Avoid long main-thread tasks. Lists the longest tasks on the main thread, useful for identifying worst contributors to input delay:

---

In [44]:

```
display(Image('/Users/User/Downloads/list.png'))
```

## 3. Reserve space for images and embedded files for good Cumulative Layout Shift (CLS)

The third metric from the Core Web Vitals set, Cumulative Layout Shift (CLS), estimates the offsets of the visual page layout on load. This is in order to gauge the frustrating UX area that everyone is likely to have encountered.
For example, a user is about to click on a certain link, but the page is displaced and ends up accidentally clicking on another area of the page.
One of the most common reasons for a high CLS score and therefore poor UX is a lack of image reservations and built-in downloads.
For example, using the Screenshots feature in the Performance tab in Chrome DevTools, we can see that there are several layout shifts that can be viewed in the tab with loading screens in more detail.

---

In [45]:

```
display(Image('/Users/User/Downloads/cls.png'))
```

You can find next particular issues that affect CLS metric in the WebCEO report:

- Avoid non-composited animations. Animations which are not composited can be janky and increase CLS.
- Avoid large layout shifts. These DOM elements contribute most to the CLS of the page:

---

In [46]:

```
display(Image('/Users/User/Downloads/layout.png'))
```

- Avoid an excessive DOM size. A large DOM will increase memory usage, cause longer style calculations, and produce costly layout reflows.

---

In [47]:

```
display(Image('/Users/User/Downloads/dom.png'))
```

# Summary

We analysed the Core Web Vitals metrics, which in the spring of 2021 will become one of the ranking factors for the Google search engine. Metrics measure loading speed, UX, and interactivity.
We collected data from ± 20,000 websites and checked their behavioral indicators on the site and Core Web Vitals metrics.
Average behavioral indicators:

- Bounce Rate - 55%
- Page Depth - 2.5
- Time on Site - 115 sec

Core Web Vitals metrics:

- CLS - 0.22
- FID - 4.8 ms
- LCP - 3670 ms

Most sites have good performance and are ready for the Google update.

After distributing the Core Web Vitals data by segments by Google values, we visually assessed whether there is a dependence of each of the core behavioral indicators on Core Web Vitals.
Next, we carried out a statistical analysis of the relationships between the indicators. We can summarize that while Core Web Vitals are responsible for important metrics for a positive UX on the site, we cannot claim that they correlate with behavioral metrics.

---

In [ ]: