Data Mining

Crawling and Analysis of Internship & Job Portal

Team

Manideep Choutapelly	533
Akanksha Mhadolkar	613
Anish Tipnis	696
Sagar Parker	634

Table of Contents

Introduction

We have conducted extensive exploration on internships and jobs in the current market. Our dataset is unique to the internet, we have undertaken the challenging task of creating end to end dataset, by crawling leading websites, Internshala and Monster India, as well as merging well known public dataset - stackoverflow developer survey from the years 2015 to 2020. We have performed extensive data exploration, using various data science and data mining techniques. The elaborated procedure is given in brief in later sections.

Objectives

1) Build a high-quality dataset and develop an efficient data preprocessing pipeline
2) Discover trends in current internships and job market
3) Highlight the areas of potential growth and improvement
4) Understand current demands in the software industry, detect anomalies

About Libraries

Our data collection strategy primarily consists of crawling individual pages(using selenium and requests library) at a time, as within the limitations of the respective robot.txt file for the website, and downloading the html page. Strategy is discussed in depth in later sections.

We have used Python as our language of choice, following best practices, guidelines and conventions. Jupyter notebook in a virtual environment for local modelling and exploration and Deep Note[1] for Collaborative workspace.

Frameworks used in this projects are:

1) Pandas

[1] https://deepnote.com/

2) Sklearn
3) MLextension library
4) Numpy
5) Plotly
6) Matplotlib
7) Seaborn

Literature of Concept

Our project extends and overlaps multiple domains ranging from regular expression parsing and web scraping to exploratory data analysis and applied data mining techniques also a bit of Natural language processing carried out extensively in text analysis. Below we summarize some important concepts necessary for understanding the project in depth.

Pattern Matching

This is one of the important concepts used while extraction of information from plain html pages. I.e. We use pattern matching to extract and convert unstructured data to more structured data.

Regular Expression

Regular expressions are patterns used to match character combinations in strings.

Meta Characters

All alphabetic characters and digits match themselves literally in regular expressions. A number of punctuation characters have special meaning:

```
^ $ . * + ? = | \ / ( ) [ ] { }
```

Some of these characters have special meaning only within certain contexts of regular expressions and are treated literally in other contexts. As a general rule, if you want to include any of these punctuation characters literally in a regular expression, you must precede them with a \. The most common mistake is using a period without a backslash to literally match a period character. Period without a backslash matches any possible symbol except a newline.

Repetitions

The characters that specify repetition always follow the pattern to which they are being applied. By using repetitions it possible to match a specific number of the same kind of character or pattern:

+ Matches the previous item one or more times. For example, A+ will match strings such as A, AA, AAA etc.

? Matches the previous item zero or one time. It is used to match an optional part of the pattern.

***** Matches the previous item zero or more time. It is used to match optional parts of the pattern.

{n}Matches the previous item exactly n times. For example, A{2} will match strings such as AA.

{n,m} Matches the previous item at least n times, but no more than m times. For example, A{2,5} will match strings such as AA, AAA, AAAA or AAAAA.

{n,} Matches the previous item at least n or more times. For example, A{2,} will match strings such as AA, AAA, AAAA, AAAAA and so on.

Character Types

\d Matches any decimal digits (0,1,2,3,4,5,6,7,8,9).

\D Matches any character that is not a decimal digit.

\s Matches any whitespace character such as space, tab and newline.

\S Matches any character that is not a whitespace.

\w Matches any "word" character (letter, digit or the underscore).

\W Matches any character that is not a "word" character.

[...] Matches any character that is listed inside square brackets. For example [abc] matches one character that is either a, b or c.

[^...] Matches any character that is not listed inside square brackets. For example [^abc] matches one character that is NOT a, b or c.

[0-9] Matches any character between 0 and 9. It is equivalent to using \d.

[A-D] Matches any character between A and F (A, B, C, D).

Web Scraping

Web scraping a web page involves fetching it and extracting from it. Fetching is the downloading of a page (which a browser does when a user views a page). Therefore, web crawling is a main component of web scraping, to fetch pages for later processing. Once fetched, then extraction can take place. *The content of a page may be parsed, searched, reformatted, its data copied into a spreadsheet or loaded into a database.* Web scrapers typically take something out of a page, to make use of it for another purpose somewhere else. An example would be to find and copy names and telephone numbers, or companies and their URLs, or e-mail addresses to a list (contact scraping).

Why is web scraping essential?

1) The website you want to extract data from does not provide a public API, and there are no comparable third-party APIs which provide the same set of data you need.
2) If there is an API, then the free tier is rate limited, meaning you are capped to calling it only a certain number of times. The paid tier of the API is cost prohibitive for your intended use case, but accessing the website itself is free.
3) The API does not expose all the data you wish to obtain even in their paid tier, whereas the website contains that information.

What are web pages all about?

All web pages are composed of HTML, which basically consists of plain text wrapped around tags that let web browsers know how to render the text. Examples of these tags include the following:

1) Every HTML document starts and ends with <html>...</html> tags.
2) By convention, <!DOCTYPE html> at the start of an HTML document. Note that any text wrapped in "<!" and ">" is considered to be a comment and not really rendered by web browsers.
3) <head>...</head> encloses meta-information about the document.
4) <body>...</body> encloses the body of the document.
5) <title>...</title> element specifies the title of the document.
6) <h1>...</h1> to <h6>...</h6> tags are used for headers.
7) ... to indicate a division in an HTML document, generally used to group a set of elements.
8) ... to enclose a paragraph.
9)
 to set a line break. <table>...</table> to start a table block.
10) <tr>...<tr/> is used for the rows.

11) <td>...</td> is used for individual cells.
12) <img> for images.
13) <a>... for hyperlinks.
14) ..., <ol>... for unordered and ordered lists, respectively; inside of these, <li>... is used for each list item.

HTML tags also contain common attributes enclosed within these tags:

1) href attribute defines a hyperlink and anchor text and is enclosed by <a> tags.
2) Jay M. Patel's homepage
3) Filename and location of images are specified by src attribute of the image tag.

Tools And Techniques

Tools and frameworks vary from language to language but for sake of simplicity, we will discuss python and its environment for scraping.

1) Scraping can be as simple as requesting an API endpoint or website. Then parsing the body of the request which usually contains HTML.
2) However, this can lead to problem ranging from wastage of bandwidth for server to complete (unintentional) DDOS attack caused by some bug in a BOT or program responsible for crawling.
3) Hence, to prevent this, lately every website is hidden behind cloudflare network, which prevents access for bots. Other mechanisms include CAPTCHA, Client Side Rendering, etc.
4) To overcome this, headless browsers are used for scraping and crawling. They work even on client side rendered websites, which are javascript heavy, requiring DOM access for page layout. Which obviously cannot be performed by conventional GET requests.
5) One of the most important things is to respect the robot.txt file present in all websites.

For this project we have used Selenium and Selenium IDE for crawling the website. Elaborated extraction process is present in **Building Dataset** section

Data Preprocessing

Why Do We Need To Preprocess The Data?

Much of the raw data contained in databases is unpreprocessed, incomplete, and noisy. For example

1) fields that are obsolete or redundant;

2) missing values;
3) Outliers.
4) Not in form edible for exploration.
5) data in a form not suitable for the data mining models;
6) values not consistent with policy or common sense.

In order to be useful for data mining purposes, the databases need to undergo preprocessing, in the form of data cleaning and data transformation.

Depending on the data set, data preprocessing alone can account for **10–60%** of all the time and effort for the entire data mining process.

Some problems encountered during cleaning our dataset.

1) Salary part of extraction from HTML pages did not always yield a single number, it was a range sometimes in string, consisting of lower and upper bound of salary. This data cannot be used directly for modelling, hence we had to split the salary and extract minimal, maximum and mean salary

Association Mining

Association Mining is Analysis of the study of attributes and characteristics that go together. Seek to uncover associations among the attributes.

Association rules take the form of "If Antecedent then consequent".

If an item says Z is not frequent then for any item A, Z\cup A and will not be frequent .

Important Terminology

Support = Number of transactions containing both A and B / Total Number of transactions. 2% support indicates that all transaction under analysis A and B are bought together

Confidence = Number of transactions containing both A and B / Number of transactions containing A 60% confidence indicates that 60% of people who bought A also bought B

Itemset Frequency: Number of transactions contain the itemset

Transactions: Set of Items

NLP concepts

NLP (Natural Language Processing) is a field of artificial intelligence that studies the interactions between computers and human languages, in particular how to program computers to process and analyze large amounts of natural language data. NLP is often applied for analysing text data.

We are going to be using NLP for performing text analysis on our dataset.

Text Preprocessing

Lower Casing :

Converting a word to lower case (NLP -> nlp). Words like Book and book mean the same but when not converted to the lower case those two are represented as two different words in the vector space model (resulting in more dimensions)

Stop word removal :

Stop words are very commonly used words (a, an, the, etc.) in the documents. These words do not really signify any importance as they do not help in distinguishing two documents. We are using the NLTK library for performing stop word removal.

Spelling correction :

Sometimes a dataset may contain typo errors, hence first we need to correct that data to reduce multiple copies of the same words, which represent the same meaning. We'll be using TextBlob library in python.

Stemming :

Stemming is the process by which we bring down a word from its different forms to the root word. This helps us establish meaning to different forms of the same words without having to

deal with each form separately. For example, the words 'fishing', 'fished', and 'fisher' all get stemmed to the word 'fish'. We're going to use NLTK library to perform stemming.

Keyword Extraction

Using TF-IDF Vectorizer :

When we need to compare the similarity between two documents, it's helpful to assign a measure of importance for each word in the documents, so we can focus on specific parts. TF-IDF consists in finding this "importance" by a product of two terms: The TF term tells the frequency of the word in the document, while the IDF term is about how rarely are documents with that specific word. The basic idea is: If the word appears a lot in very few documents, it must be important. If it appears a lot in many documents, it must not be important("the", "of", "a", for example). We are going to use Scikit-Learn library for this purpose.

Using YAKE :

Yet Another Keyword Extractor (Yake) library selects the most important keywords using the text statistical features method from the article. With the help of YAKE, you can control the extracted keyword word count and other features. It is a light weight unsupervised keyword extraction approach. It heavily relies on statistical text features which are selected and computed from a single document. This is where it sets itself apart from the TF-IDF approach and does not require any dictionaries or any external corpora.

Using CountVectorizer :

CountVectorizer converts a collection of text documents to a matrix of token counts: the occurrences of tokens in each document. This implementation produces a sparse representation of the counts. For each sentence in the corpus, the position of the tokens (words in our case) is completely ignored.

Word Cloud

Word Cloud or Tag Clouds is a visualization technique for texts that are natively used for visualizing the tags or keywords from the websites. These keywords typically are single words

that depict the context of the webpage the word cloud is being made from. These words are clustered together to form a Word Cloud.

Each word in this cloud has a variable font size and color tone. Thus, this representation helps to determine words of prominence. A bigger font size of a word portrays its prominence more relative to other words in the cluster. Word Cloud can be built in varying shapes and sizes based on the creators' vision. The number of words plays an important role while creating a Word Cloud. More number of words does not always mean a better Word Cloud as it becomes cluttered and difficult to read. A Word Cloud must always be semantically meaningful and must represent what it is meant for. For visualization, matplotlib is a basic library that enables many other libraries to run and plot on its base including seaborn or wordcloud.

Algorithms

Text Analysis

1. **Stop word removal**

A stop word is a commonly used word (such as “the”, “a”, “an”, “in”) that a search engine has been programmed to ignore, both when indexing entries for searching and when retrieving them as the result of a search query. Stop words can safely be removed without sacrificing the meaning of the sentence. While performing text analysis, it is essential to remove stop words in order to keep unwanted words out of the corpus.

Removing stop words using python libraries is pretty easy and can be done in many ways. We have used the Natural Language Toolkit, or more commonly called NLTK for stop word removal and we did so with the following code :

```
from nltk.corpus import stopwords

def remove_stop_words(s):
    if type(s) == str:
        stop = stopwords.words('english')
        return " ".join(x for x in s.split() if x not in stop)
    else:
        return None
```

2. **TF-IDF Vectorizer**

TF-IDF stands for Term Frequency Inverse Document Frequency. TF-IDF helps in finding this “importance” by a product of two terms: The TF term tells the frequency of the word in the document, while the IDF term is about how rarely are documents with that specific word. The basic idea is: If the word appears a lot in very few documents, it must be important. If it appears a lot in many documents, it must not be important(“the”, “of”, “a”, for example).

We can easily implement TF-IDF vectorization in python using the Scikit-Learn library. Here’s how we can perform TF-IDF vectorization :

```
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf_vectorizer = TfidfVectorizer(max_features=200000,
                                   stop_words='english',
                                   use_idf=True,
                                   ngram_range=(1,3))

tfidf_matrix = tfidf_vectorizer.fit_transform(text_analysis_df.about_internship)
terms = tfidf_vectorizer.get_feature_names()
```

3. **Countvectorizer**

Countvectorizer is used to transform a given text into a vector on the basis of the frequency (count) of each word that occurs in the entire text. This is helpful when we have multiple such texts, and we wish to convert each word in each text into vectors (for use in further text analysis). CountVectorizer creates a matrix in which each unique word is represented by a column of the matrix, and each text sample from the document is a row in the matrix.

Here's how we can implement count vectorization in python using the Scikit-Learn library :

```
from sklearn import feature_extraction

count_vec = feature_extraction.text.CountVectorizer()
```

4. **YAKE**

YAKE stands for Yet Another Keyword Extractor. It is a light-weight unsupervised automatic keyword extraction method which relies on text statistical features extracted from single documents to select the most important keywords of a text. It follows an unsupervised Approach for Automatic Keyword Extraction using Text Features.

We are working with KeywordExtractor from Yake library.

```
from yake import KeywordExtractor

def yake_keyword_extractor(input_str):
    keywords = kw_extractor.extract_keywords(text=input_str)
    keywords = [x for x, y in keywords]
    for keyword in keywords:
        all_keywords[keyword] = all_keywords.get(keyword, 0) + 1
    return ','.join(keywords)
```

5. **Word Cloud**

Word Cloud is a data visualization technique used for representing text data in which the size of each word indicates its frequency or importance. Significant textual data points can be highlighted using a word cloud. Word clouds are widely used for analyzing data from social network websites.

For generating a word cloud in Python we need matplotlib and wordcloud. We can get a word cloud by implementing the following code :

```
from wordcloud import WordCloud
import matplotlib.pyplot as plt

def get_word_cloud(input_str):
    wordcloud = WordCloud().generate(input_str)
    plt.figure(figsize=(10,5))
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.axis("off")
```

FP Growth

Fp Growth Algorithm (Frequent pattern growth). FP growth algorithm is an improvement of apriori algorithm. FP growth algorithm used for finding frequent itemset in a transaction database without candidate generation. FP growth represents frequent items in frequent pattern trees or FP-tree. It constructs an FP Tree rather than using the generate and test strategy of Apriori and therefore it is faster than Apriori algorithm. The focus of the FP Growth algorithm is on fragmenting the paths of the items and mining frequent patterns.

We are going to implement FP growth with the Mlxtend (machine learning extensions) library in python.

```
from mlxtend.frequent_patterns import fpgrowth

frequent_itemsets = fpgrowth(freq_skills_df, min_support=0.05, use_colnames=True)
```

Building Dataset

Scraping Strategy

For responsive Website: (Monster India)

1) Use Selenium (Headless Browser) to load the website (using Paginated search URL).
2) Wait for the content to load, using random wait time between 2 seconds and 8 seconds.
3) Wait time is randomised to avoid IP ban, and to respect the maximum request per second allowed by robot.txt file for the website
4) Instead of parsing the content, download the HTML markup and store them in a directory
5) Do this for all the pages for the search results.
6) There can be a second round of scraping to retrieve pages with links in search result in a similar manner.

For non-responsive websites: (Internshala)

1) Perform GET request after randomized time interval for reasons explained above
2) Retrieve the search results for all pages, store them.
3) For reading content inside each post in search result, first parse over all the stored pages to create a list of links for each post.
4) Now, again perform the scraping task for the links.

Data Cleaning

This is one the most time consuming part, as every website has different schema and logic. Requiring different set of tools and techniques for extraction as well as cleaning the extracted data. Some of the problems faced during our data cleaning process are:

1) Extracted Salary as raw string in form of range, sometimes absent.
2) Skills, expressed as string, which are not standardised eg. AWS and Amazon Web Services are the same logically. Also, data modelling nor exploration is possible on a string containing multiple entities.
3) Date of posting, which in some cases is in the form of “last day/week/month”. This has to be converted to respective time relative to date of data extraction.
4) Stipends field, expressed in form of “$xxx/month”.

Dependencies:

```
!pip install scikit-learn numpy seaborn matplotlib Scrapy beautifulsoup4
requests pyprind selenium
```

Monster India:

Scraping

```
ef download_page(page_src,name):
    os.environ['MOZ_HEADLESS'] = '1'
    driver =
webdriver.Firefox(executable_path='/Users/admin/Projects/Msc/Data
Mining/Scrapping/geckodriver')
    driver.get(page_src)
    scroll_down(driver)
    with open(f"../raw/{name}.html", "w") as f:
        f.write(driver.page_source)
    driver.close()

TOTAL_POSTINGS = 6000
TOTAL_PAGES = TOTAL_POSTINGS // 100

bar = pyprind.ProgBar(TOTAL_PAGES)
for i in range(TOTAL_PAGES):
    download_page(get_parameterized_url(i),
f"monster-india-filtered-by-salary/monster-india-search-page-{i}")
    # after every 10 requests wait for 4 seconds
    if(i % 10 == 0):
        time.sleep(random.randint(0,4))
    bar.update()
print(bar)
```

Parsing Search Results

```
def get_job_posting_date(footer_node):
    return {'timestamp' :footer_node.find(class_='posted').text.strip()}

def get_job_title_info(node):
    title_node = node.find(class_='job-tittle')
```

```
    job_title = title_node.find('h3').text.strip()
    location = title_node.find(class_='loc').text.strip()
    company_name = title_node.find(class_='company-name').text.strip()
    package = title_node.find(class_='loc salarySnb').text.strip()
    experience = title_node.find(class_='exp').text.strip()

    return {'location': location, 'job_title': job_title,
            'company_name': company_name, 'package': package,
            'experience': experience}

def get_job_desc(node):
    job_desc_node = node.find(class_='job-descrip').text.strip()
    return {'job_description': job_desc_node}

def get_job_skills(node):
    skills_node = node.find(class_='descrip-skills')
    skills = []
    if skills_node:
        skills_nodes = skills_node.find_all('a')
        for s_node in skills_nodes:
            skills.append(s_node.text)
    remove_special_char = map(lambda x:
x.replace('\n','').replace(',','').strip(), skills)
    filtered_skills = filter(lambda x: len(x) > 1, remove_special_char)
    skills_string = ','.join(filtered_skills)
    return {'skills': skills_string}
```

```
all_job_posts = list()
bar = pyprind.ProgBar(len(all_jobs_listings))
for search_page in all_jobs_listings:
    bar.update()
    try:
        path = '../raw/monster-india-filtered-by-salary' + '/' +
search_page
        soup = BeautifulSoup(open(path), 'html.parser')
        # Notice the strucutre of each jobs card, it has the className
job-apply-card.
        all_job_cards = soup.find_all(class_='job-apply-card')

        for job_card in all_job_cards:
            try:
                # card-body and card-footer are the child elements
containing relavant content
                card_body_node = job_card.find(class_='card-body')
                card_footer = job_card.find(class_='card-footer')

                data = dict()
                posted_on = get_job_posting_date(card_footer)
                title_info = get_job_title_info(card_body_node)
                job_desc = get_job_desc(card_body_node)
                job_skills = get_job_skills(card_body_node)

                data.update(posted_on)
                data.update(title_info)
                data.update(job_desc)
                data.update(job_skills)

                all_job_posts.append(data)
            except Exception as e:
                print('Failed to parse node' + e)

    except Exception as e:
        print(e)
```

Output:

```
0% [##############################] 100% | ETA: 00:00:00
Total time elapsed: 00:13:05

Title:
  Started: 06/01/2021 09:30:22
  Finished: 06/01/2021 09:43:28
  Total time elapsed: 00:13:05
```

Cleaning

timestamp column

Notice that the data comes in forms of:

- `Posted: N days ago`
- `Posted: an hour ago`
- `Posted: N hours ago`
- `Posted: a day ago`
- `Posted: a month ago`

Also, the data was downloaded at 9:00 am Approx. on 1st July 2021.

Treatment

- We care only about the date and not the exact time (hour, seconds) on which the job post was made
- Hence, every timestamp that has hours, will have Date as 01/06/2021.
- For the Edge case of Month 'AGO', we consider that as 01/05/2021.

Algorithm

- Hence, we are left to extact the N from `N days ago`
- Subtract `N` from `31` with month as `05 (may)`.
- The Postings with `N hours ago` falls in `1st July`

For sake of scalability we'll keep month number and days as CONSTANTs so that in future, code can be ran as it is by simply updating the month and days of past month.

```
LAST_MONTH = 5
LAST_MONTH_DAYS = 31
YEAR = 2021
PUBLICATION_DAY = 1

def extract_days_before_N(input_string):
    '''
        return timestamp

        if the input_string is in the form of Posted: N days ago, it
returns timestamp wtr to past month
        for input_string in form of `hours ago`, the date of publication is
used.
    '''
    re_extract_num = re.compile('\d+')
```

```
    finds = re_extract_num.findall(input_string)
    if(len(finds) > 0):
        num = int(finds[0])
        exact_day = LAST_MONTH_DAYS - num
        exact_date = date(YEAR, LAST_MONTH, exact_day)
        return int(exact_date.strftime("%s"))
    else:
        publication_date = date(YEAR, LAST_MONTH + 1, PUBLICATION_DAY)
        return int(publication_date.strftime("%s"))

# Map string to integer(timestamp)
df.timestamp = df.timestamp.apply(extract_days_before_N)
```

Packages

- In the extracted Dataset, `package` is in categorical format.
- There are about `1250` categories of Packages
- This makes it easier for classification based machine learning algorithm to work
- However, It would be helpful to extract `minimum`, `maximum` and `mean` salary from the given package

So we Keep the package (categorical) variable as it as, and create 3 more features:

- minimum_salary
- maximum_salary
- mean_salary

```
def convert_package_to_min_salaries(package):
    hyphened_salary = package.split(' ')[0]
    sal_1, sal_2 = hyphened_salary.split('-')
    integer_sal_1 = int(sal_1.replace(',',''))
    integer_sal_2 = int(sal_2.replace(',',''))
    min_sal = min(integer_sal_1, integer_sal_2)
    return min_sal


def convert_package_to_max_salaries(package):
    hyphened_salary = package.split(' ')[0]
    sal_1, sal_2 = hyphened_salary.split('-')
    integer_sal_1 = int(sal_1.replace(',',''))
    integer_sal_2 = int(sal_2.replace(',',''))
```

```
    max_sam = max(integer_sal_1, integer_sal_2)
    return max_sal


def convert_package_to_mean_salaries(package):
    hyphened_salary = package.split(' ')[0]
    sal_1, sal_2 = hyphened_salary.split('-')
    integer_sal_1 = int(sal_1.replace(',',''))
    integer_sal_2 = int(sal_2.replace(',',''))
    mean_sal = np.average([integer_sal_1, integer_sal_2])
    return mean_sal


min_salary_feature = df.package.apply(convert_package_to_min_salaries)
max_salary_feature = df.package.apply(convert_package_to_max_salaries)
mean_salary_feature = df.package.apply(convert_package_to_mean_salaries)
```

Experience

Values are in the form of `N-M Years` or `Fresher`

Algorithm

Method 1

- Split by space and grab the first element. we get `N-M`
- Split by '-' to get 'N' and 'M'
- Convert to integer and compute, min,max, mean experience

Method 2

- Use Regex to grab the integers
- use `min()` and `max()`

We'll also see which is faster method.

```
def extract_min_experience_split(input_string: str):
    if(input_string.find("Years") != -1):
        exp_range = input_string.split(' ')[0]
        exp_1, exp_2 = exp_range.split('-')
        min_exp = min(int(exp_1), int(exp_2))
        return min_exp
    return 0
```

```
def extract_max_experience(input_string: str):
    if(input_string.find("Years") != -1):
        exp_range = input_string.split(' ')[0]
        exp_1, exp_2 = exp_range.split('-')
        min_exp = max(int(exp_1), int(exp_2))
        return min_exp
    return 0

def extract_mean_experience(input_string: str):
    if(input_string.find("Years") != -1):
        exp_range = input_string.split(' ')[0]
        exp_1, exp_2 = exp_range.split('-')
        min_exp = np.average([int(exp_1), int(exp_2)])
        return min_exp
    return 0
```

Internshala

Procedure

- Download Search Pages
- Parse all individual links (relevant to internship post)
- Download All internship post pages
- Parse and Extract all content of internship pages

```
def scroll_down(driver):
    driver.set_window_size(1440, 875)
    driver.execute_script("window.scrollTo(0,397)")
    time.sleep(1)
    driver.execute_script("window.scrollTo(0,24695)")

def get_parameterized_url(start_after):
    return
f'https://internshala.com/internships/computer%20science-internship/page-{s
tart_after}'


def download_page(page_src,name):
    os.environ['MOZ_HEADLESS'] = '1'
    driver =
```

```
webdriver.Firefox(executable_path='/Users/admin/Projects/Msc/Data
Mining/Scrapping/geckodriver')
    driver.get(page_src)
    scroll_down(driver)
    with open(f"../raw/{name}.html", "w") as f:
        f.write(driver.page_source)
    driver.close()

def requests_download_page(URL, filename):
    res = requests.get(URL, headers=my_headers)
    with open(f"../raw/{filename}.html", "w") as f:
        f.write(res.text)
```

```
TOTAL_PAGES = 50

bar = pyprind.ProgBar(TOTAL_PAGES)
for i in range(1, TOTAL_PAGES + 1):
    requests_download_page(get_parameterized_url(i),
f"internshala-june/search-pages/search-page-{i}")
    time.sleep(1)
    if(i % 10 == 0):
        time.sleep(random.randint(2,4))
    bar.update()
print(bar)
```

```
 Title:
   Started: 07/01/2021 09:27:53
   Finished: 07/01/2021 09:30:17
   Total time elapsed: 00:02:23
```

(Cleaning part is mostly similar to monster india dataset, and has been omitted here)

Stackoverflow Dataset

We have collected stackoverflow survey results of the past 5 years, i.e. from 2015 to 2020, for time series analysis. Here are the procedures followed for merging datasets from different surveys. One of the major problems is constant change in schema i.e. feature names as well the semantics of the feature, is not consistent across years.

The following fields are extracted:

1) Country
2) Gender
3) Developer Type
4) Languages
5) Databases and Platforms
6) Frameworks

Loading all the surveys

```
ROOT_CSV_PATH = 'csvs'
path = ROOT_CSV_PATH + '/' + '2011/2011 Stack Overflow Survey Results.csv'
path_2015 = f"{ROOT_CSV_PATH}/2015/2015 Stack Overflow Developer Survey
Responses.csv"
path_2016 = f"{ROOT_CSV_PATH}/2016/2016 Stack Overflow Survey
Responses.csv"
path_2017 = f"{ROOT_CSV_PATH}/2017/survey_results_public.csv"
path_2018 = f"{ROOT_CSV_PATH}/2018/survey_results_public.csv"
path_2019 = f"{ROOT_CSV_PATH}/2019/survey_results_public.csv"
path_2020 = f"{ROOT_CSV_PATH}/2020/survey_results_public.csv"


df_2011 = pd.read_csv(path)
df_2015 = pd.read_csv(path_2015)
df_2016 = pd.read_csv(path_2016)
df_2017 = pd.read_csv(path_2017)
df_2018 = pd.read_csv(path_2018)
df_2019 = pd.read_csv(path_2019)
df_2020 = pd.read_csv(path_2020)
```

Countries

```
df_country = pd.DataFrame(columns=['country','count', 'year'])


all_countries_series = [df_2015.Country, df_2016.country, df_2017.Country,
                        df_2018.Country, df_2019.Country, df_2020.Country]
all_year = [D'2015', '2016', '2017', '2018', '2019', '2020']

df_country = append_to_df(df_country ,all_countries_series ,all_year,
'country')
```

Gender

```
df_gender = pd.DataFrame(columns=['gender', 'count', 'year'])

all_age_series = [df_2015.Gender, df_2016.gender, df_2017.Gender,
                df_2018.Gender, df_2019.Gender, df_2020.Gender]

df_gender = append_to_df(df_gender, all_age_series, all_year, 'gender')
```

Dev Type

```
def delimiter_sep_series(sep=','):

    def sep_series(series):
        series_map = {}
        for item in series.values:
            if type(item) == str:
                for i in item.split(sep):
                    key = i.strip()
                    series_map[key] = series_map.get(key, 0) + 1

        return pd.Series(data=list(series_map.values()),
index=list(series_map.keys()))

    return sep_series


df_devtypes = pd.DataFrame(columns=['dev_type','count', 'year'])
```

```
all_devtypes = [df_2015.Occupation.value_counts(),
df_2016.occupation.value_counts(),
               df_2017.DeveloperType.agg(delimiter_sep_series(';')),
df_2018.DevType.agg(delimiter_sep_series(';')),
               df_2019.DevType.agg(delimiter_sep_series(';')),
df_2020.DevType.agg(delimiter_sep_series(';'))]

for idx, series in enumerate(all_devtypes):
    dev = series.index.tolist()
    counts = series.values.tolist()

    _df = pd.DataFrame(zip(dev, counts, [all_year[idx] for _ in
range(len(counts))]),
                                    columns=['dev_type', 'count', 'year'])
    df_devtypes = df_devtypes.append(_df)
```

Languages

```
all_languages =
[df_2016.tech_do.agg(delimiter_sep_series(';')),df_2017.HaveWorkedLanguage.
agg(delimiter_sep_series(';')),

df_2018.LanguageWorkedWith.agg(delimiter_sep_series(';')),df_2019.LanguageW
orkedWith.agg(delimiter_sep_series(';')),
                 df_2020.LanguageWorkedWith.agg(delimiter_sep_series(';'))]

df_languages = pd.DataFrame(columns=['languages','count', 'year'])

for idx, series in enumerate(all_languages):
    dev = series.index.tolist()
    counts = series.values.tolist()

    _df = pd.DataFrame(zip(dev, counts, [all_year[idx] for _ in
range(len(counts))]),
                       columns=['languages', 'count', 'year'])

    df_languages = df_languages.append(_df)
```

Databases and platforms

```
def column_to_df(columns, col):
    all_cols = [
        df_2017[columns[0]].agg(delimiter_sep_series(';')),
        df_2018[columns[1]].agg(delimiter_sep_series(';')),
        df_2019[columns[2]].agg(delimiter_sep_series(';')),
        df_2020[columns[3]].agg(delimiter_sep_series(';'))
    ]

    df_cols = pd.DataFrame(columns=[col,'count', 'year'])

    for idx, series in enumerate(all_cols):
        dev = series.index.tolist()
        counts = series.values.tolist()
        all_year=['2017', '2018','2019','2020']
        _df = pd.DataFrame(zip(dev, counts, [all_year[idx] for _ in
range(len(counts))]),
                           columns=[col, 'count', 'year'])

        df_cols = df_cols.append(_df)

    return df_cols

df_databases = column_to_df(['HaveWorkedDatabase', 'DatabaseWorkedWith',
'DatabaseWorkedWith', 'DatabaseWorkedWith'], 'database')
df_platforms = column_to_df(['HaveWorkedPlatform', 'PlatformWorkedWith',
'PlatformWorkedWith', 'PlatformWorkedWith'], 'platform')
```

Frameworks

```
columns = ['HaveWorkedFramework', 'FrameworkWorkedWith',
'WebFrameWorkedWith', 'MiscTechWorkedWith' , 'WebframeWorkedWith',
'MiscTechWorkedWith']
all_cols = [
    df_2017[columns[0]].agg(delimiter_sep_series(';')),
    df_2018[columns[1]].agg(delimiter_sep_series(';')),
    df_2019[columns[2]].agg(delimiter_sep_series(';')),
    df_2019[columns[3]].agg(delimiter_sep_series(';')),
    df_2020[columns[4]].agg(delimiter_sep_series(';')),
    df_2020[columns[5]].agg(delimiter_sep_series(';'))
```

```
]

df_frameworks = pd.DataFrame(columns=['frameworks','count', 'year'])

for idx, series in enumerate(all_cols):
    dev = series.index.tolist()
    counts = series.values.tolist()
    all_year=['2017', '2018','2019','2019','2020','2020']
    _df = pd.DataFrame(zip(dev, counts, [all_year[idx] for _ in
range(len(counts))]),
                       columns=['frameworks', 'count', 'year'])

    df_frameworks = df_frameworks.append(_df)
```

Code And Output

Internshala Qualitative Analysis

EDA

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

from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori, fpmax, fpgrowth

df = pd.read_csv('internshala_clean.csv')
df.head(3)
```

	location object	domain_profile object	company_name object	about_the_work_from_… ob…	skills object	who_can_apply object	othe
0	Work From Home	Embedded Android Development	Augrade Private Limited	Selected intern's day-to-day responsibilities include:…	Java Linux Embedded …	Only those candidates can apply who:…	1. E on d trou
1	Work From Home	Technical Project Management	Zansys Technologies	Selected intern's day-to-day responsibilities include:…	PHP Software Testing…	Only those candidates can apply who:…	nan
2	Work From Home	Full Stack Development (Web+Flutter)	Rising Youth Foundation	Selected intern's day-to-day responsibilities include:…	MongoDB Node.js Flutter…	Only those candidates can apply who:…	nan

3 rows × 259 columns

Skills Feature Analysis

```
# Skills Features
df.skills[:10]
```

```
0    Java\nLinux\nEmbedded Systems\nC Programming\n...
1                        PHP\nSoftware Testing\nReactJS
2    MongoDB\nNode.js\nFlutter\nREST API\nDart\nAnd...
3    HTML\nCSS\nJavaScript\njQuery\nMS-Office\nWord...
4                        HTML\nCSS\nReact Native\nRedux
5    HTML\nCSS\nJavaScript\nBootstrap\nNode.js\nRea...
6              WordPress\nEnglish Proficiency (Written)
7                            JavaScript\nReactJS\nRedux
8                                                   NaN
9                    HTML\nCSS\nJavaScript\nMathematics
Name: skills, dtype: object
```

Creating list of list of individual skills for each row

```
super_skills_list = []
def aggregate_skills(series):
    if(type(series) == str):
        skills = series.split('\n')
        super_skills_list.append(skills)

df.skills.agg(aggregate_skills)
```

```
0       None
1       None
2       None
3       None
4       None
        ...
2369    None
2370    None
2371    None
2372    None
2373    None
Name: skills, Length: 2374, dtype: object
```

Applying FP Growth Algorithm on Skills

```
from mlxtend.preprocessing import TransactionEncoder
```

```
from mlxtend.frequent_patterns import apriori, fpmax, fpgrowth

te = TransactionEncoder()
te_ary = te.fit(super_skills_list).transform(super_skills_list)
freq_skills_df = pd.DataFrame(te_ary, columns=te.columns_)

frequent_itemsets = fpgrowth(freq_skills_df, min_support=0.05, use_colnames=True)
frequent_itemsets.sort_values(by='support', ascending=True)
```

	support float64 0.050645007166746296 – 0.2…	itemsets object frozenset({'Node.j… 2.2% frozenset({'JavaSc… 2.2% 44 others 95.7%
35	0.050645007166746296	frozenset({'Node.js', 'CSS', 'HTML'})
30	0.050645007166746296	frozenset({'JavaScript', 'CSS', 'PHP', 'HTML'})
44	0.050645007166746296	frozenset({'ReactJS', 'React Native'})
22	0.051600573339703776	frozenset({'ReactJS', 'JavaScript', 'HTML'})
29	0.05351170568561873	frozenset({'JavaScript', 'CSS', 'PHP'})
Expand rows 5 - 40		
2	0.19875776397515527	frozenset({'ReactJS'})
38	0.22981366459627328	frozenset({'CSS', 'HTML'})
11	0.2508361204013378	frozenset({'CSS'})
10	0.2814142379359771	frozenset({'HTML'})
9	0.2904921165790731	frozenset({'JavaScript'})
46 rows × 2 columns		

Association Rules Mining

```
from mlxtend.frequent_patterns import association_rules
```

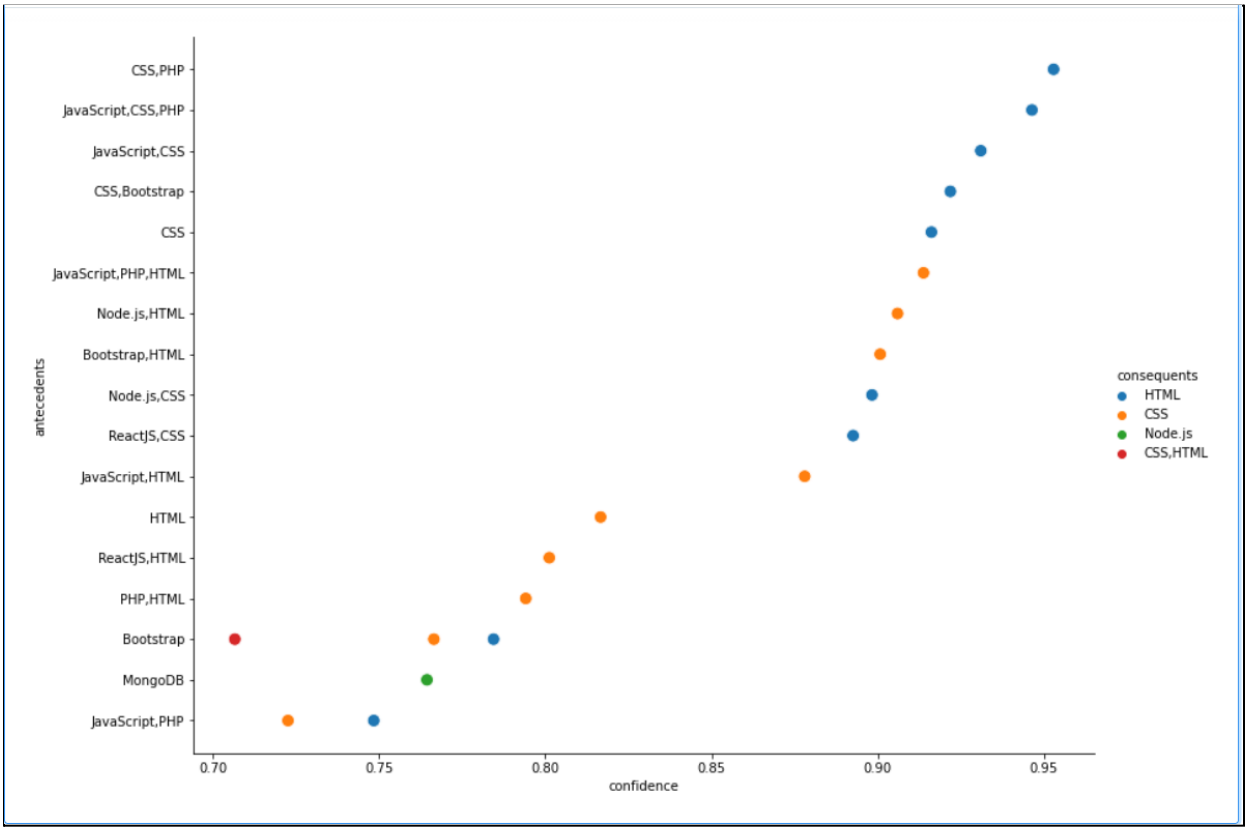
```
rules_df = association_rules(frequent_itemsets, metric="confidence",
min_threshold=0.01)
rules_df.sort_values(by='confidence', ascending=False, inplace=True)
rules_df.head(20)
```

	antecedents object frozenset({'Bootst… 15% frozenset({'JavaSc… 10% 15 others 75%	consequents object frozenset({'HTM… 45% frozenset({'CSS… 45% 2 others 10%	antecedent support float64 0.05351170568561873 – 0.28…	consequent support float64 0.16961299569995222 – 0.28…	support float64 0.050645007166746296
24	frozenset({'CSS', 'PHP'})	frozenset({'HTML'})	0.08122312470138557	0.2814142379359771	0.07740086000955566
42	frozenset({'JavaScript', 'CSS', 'PHP'})	frozenset({'HTML'})	0.05351170568561873	0.2814142379359771	0.05064500716674629
78	frozenset({'JavaScript', 'CSS'})	frozenset({'HTML'})	0.16626851409460106	0.2814142379359771	0.15480172001911133
88	frozenset({'CSS', 'Bootstrap'})	frozenset({'HTML'})	0.061156235069278544	0.2814142379359771	0.05637840420449116
74	frozenset({'CSS'})	frozenset({'HTML'})	0.2508361204013378	0.2814142379359771	0.22981366459627328
Expand rows 5 - 14					
87	frozenset({'Bootstrap'})	frozenset({'CSS'})	0.07978977544194936	0.2508361204013378	0.06115623506927854
71	frozenset({'MongoDB'})	frozenset({'Node.js'})	0.09125656951743909	0.16961299569995222	0.0697563306258958
30	frozenset({'JavaScript', 'PHP'})	frozenset({'HTML'})	0.0740563784042045	0.2814142379359771	0.05542283803153368
37	frozenset({'JavaScript', 'PHP'})	frozenset({'CSS'})	0.0740563784042045	0.2508361204013378	0.05351170568561873
92	frozenset({'Bootstrap'})	frozenset({'CSS', 'HTML'})	0.07978977544194936	0.22981366459627328	0.05637840420449116
20 rows × 9 columns					

Graphical Representation of top rules

```
rules_df.antecedents = rules_df.antecedents.apply(lambda s: ','.join([x for x in
```

```
s]))
rules_df.consequents = rules_df.consequents.apply(lambda s: ','.join([x for x in
s]))
plt.figure(figsize=(18,1))
sns.relplot(data=rules_df[:20], y='antecedents' , x='confidence',
hue='consequents', s=100,  height=8.27, aspect=11.7/8.27)
```



Internshala EDA

```
import pandas as pd
import re
from datetime import date
import numpy as np
import math
import pygal
import seaborn as sns
#from plotly.offline import init_notebook_mode,iplot
import plotly.graph_objects as go
import matplotlib.pyplot as plt
import plotly.express as px
from IPython.display import SVG, display
from plotly.subplots import make_subplots

import plotly.offline as pyo
import plotly.graph_objs as go

from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori, fpmax, fpgrowth
# Set notebook mode to work in offline
pyo.init_notebook_mode()
```

```
# Global Functions
sns.set_palette("pastel")
def plot_series_pie_chart(series, title, max=10,
color_seq=px.colors.qualitative.Pastel1):
    series.sort_values(ascending=False, inplace=True)
    series = series[:max]
    fig = px.pie(
        names=series.index,
        values=series.values,
        title=title,
        color_discrete_sequence=color_seq
        )

    fig['data'][0].update({'textinfo' : 'label+text+value+percent'})
    fig.show(renderer="colab")

def plot_bar_chart(series, title, text_position='outside', xaxis_label=None,
yaxis_label=None):
    fig = go.Figure(
        data=[go.Bar(x = series.index, y = series.values, text= series.values)],
        layout_title_text=title,
```

```
    )
    fig.update_traces(texttemplate='%{text:.2s}', textposition=text_position)
    fig.update_layout(xaxis_tickangle=-45)
    if xaxis_label != None:
        fig.update_layout(xaxis_title=xaxis_label,)
    if yaxis_label != None:
        fig.update_layout(yaxis_title=yaxis_label,)

    fig.show(renderer="colab")
```

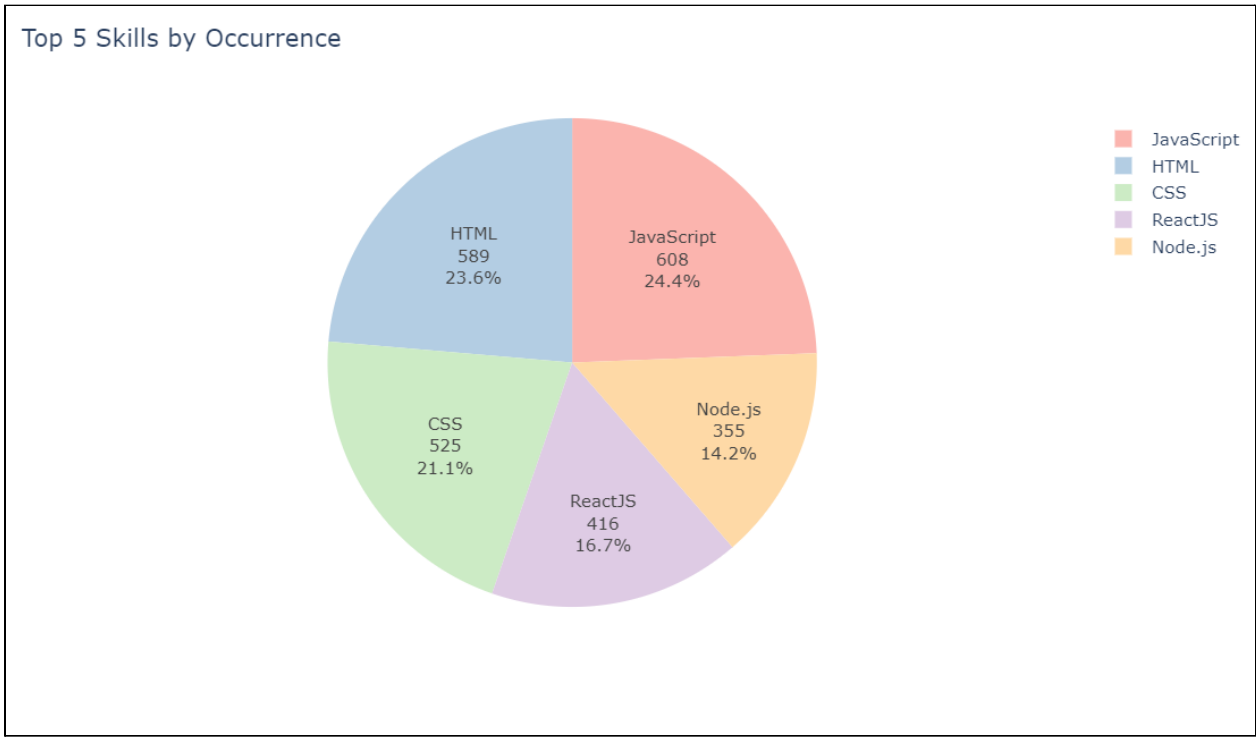
```
df = pd.read_csv('internshala_clean.csv')
df.head(3)
```

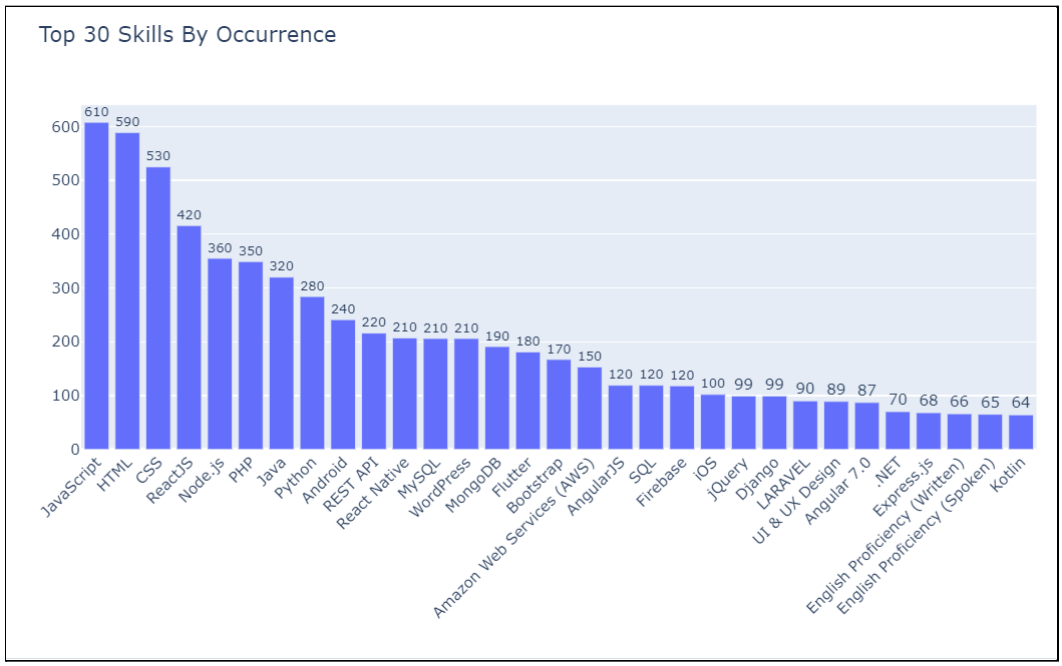
	location object	domain_profile object	company_name object	about_the_work_from_… ob…	skills object	who_can_apply object
0	Work From Home	Embedded Android Development	Augrade Private Limited	Selected intern's day-to-day responsibilities include:…	Java Linux Embedded …	Only those candidates can apply who:…
1	Work From Home	Technical Project Management	Zansys Technologies	Selected intern's day-to-day responsibilities include:…	PHP Software Testing…	Only those candidates can apply who:…
2	Work From Home	Full Stack Development (Web+Flutter)	Rising Youth Foundation	Selected intern's day-to-day responsibilities include:…	MongoDB Node.js Flutter…	Only those candidates can apply who:…

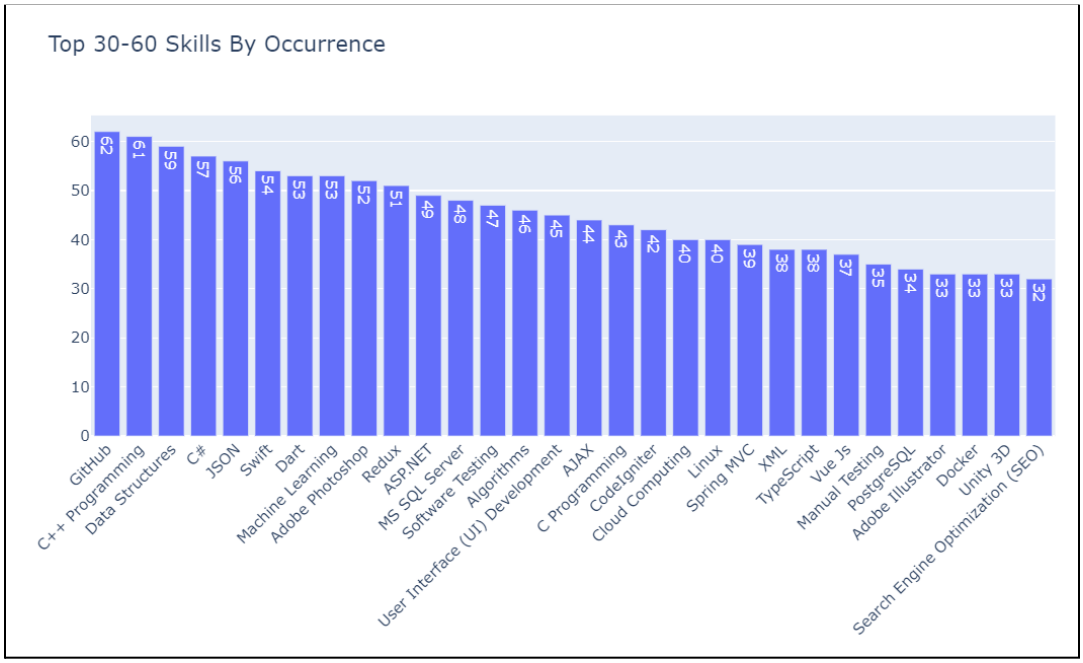
3 rows × 259 columns

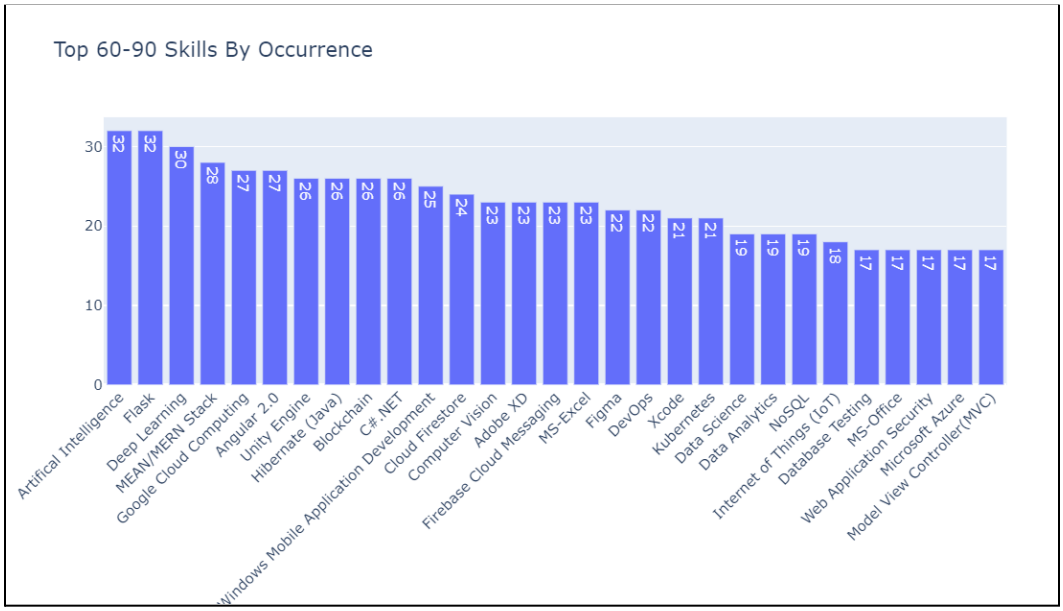
Skills Analysis

```
df.skills = df.skills.apply(lambda s: s.replace('Amazon Web Server (AWS)', 'Amazon
Web Services (AWS)') if type(s) == str else None)
skills_super_set = dict()
def agg_skillset(input_string):
    skillset = {}
    for skill in input_string:
        if(type(skill) == str):
            for i in skill.split('\n'):
                skillset[i] = skillset.get(i, 0) + 1
    return skillset

skills_super_set = df.skills.agg(agg_skillset)
skills_count_series = pd.Series(skills_super_set).sort_values(ascending=False)
plot_series_pie_chart(skills_count_series[:5], "Top 5 Skills by Occurrence")
plot_bar_chart(skills_count_series[:31], "Top 30 Skills By Occurrence")
plot_bar_chart(skills_count_series[31:61], "Top 30-60 Skills By Occurrence",
'inside')
plot_bar_chart(skills_count_series[61:90], "Top 60-90 Skills By Occurrence",
'inside')
```

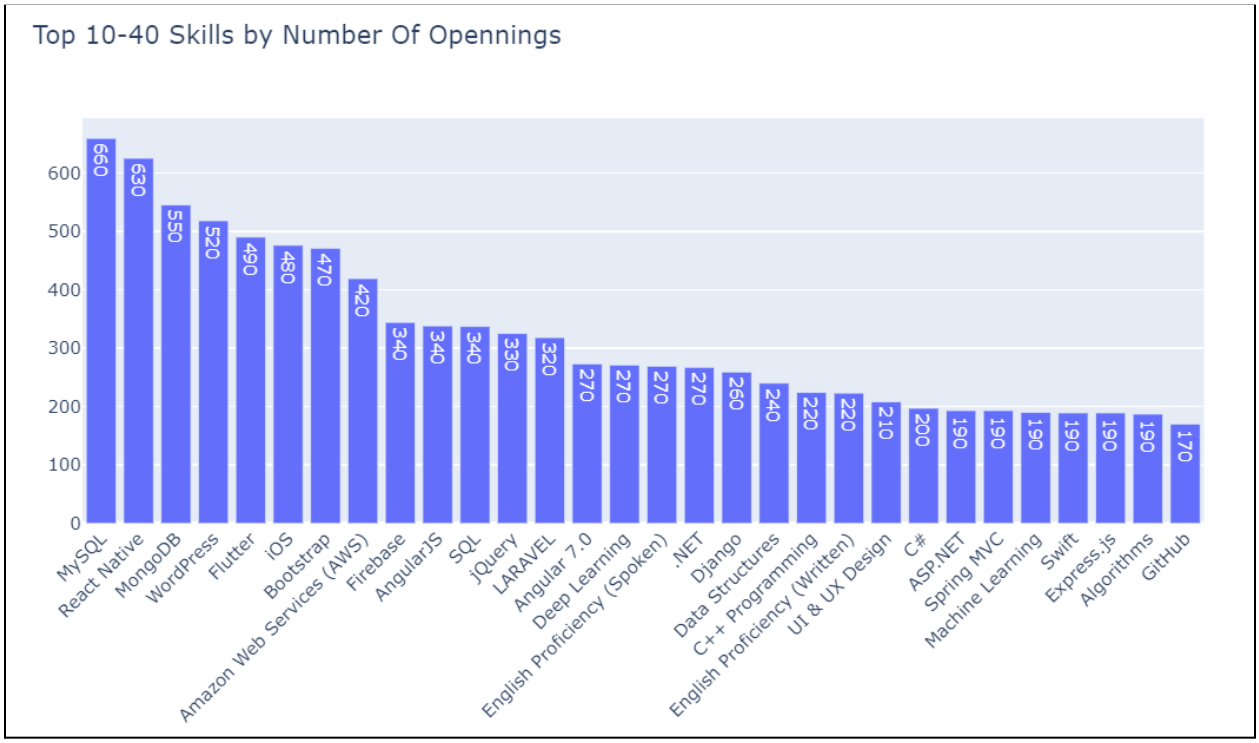



Top 30 Skills By Occurrence
600
500
400
300
200
100
0
610
590
530
420
360
350
320
280
240
220
210
210
210
190
180
170
150
120
120
120
100
99
99
90
89
87
70
68
66
65
64
JavaScript
HTML
CSS
ReactJS
Node.js
PHP
Java
Python
Android
REST API
React Native
MySQL
WordPress
MongoDB
Flutter
Bootstrap
Amazon Web Services (AWS)
AngularJS
SQL
Firebase
iOS
jQuery
Django
LARAVEL
UI & UX Design
Angular 7.0
.NET
Express.js
English Proficiency (Written)
English Proficiency (Spoken)
Kotlin

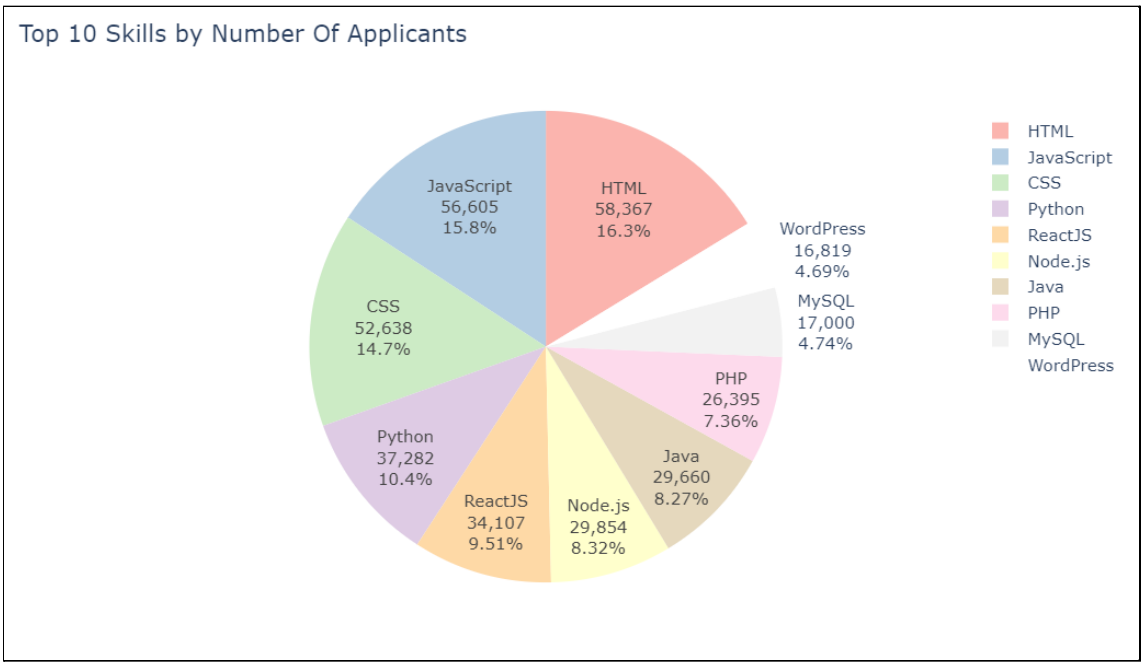

Top 30-60 Skills By Occurrence
60
50
40
30
20
10
0
62
61
59
57
56
54
53
53
52
51
49
48
47
46
45
44
43
42
40
40
39
38
38
37
35
34
33
33
33
32
GitHub
C++ Programming
Data Structures
C#
JSON
Swift
Dart
Machine Learning
Adobe Photoshop
Redux
ASP.NET
MS SQL Server
Software Testing
Algorithms
User Interface (UI) Development
AJAX
C Programming
CodeIgniter
Cloud Computing
Linux
Spring MVC
XML
TypeScript
Vue Js
Manual Testing
PostgreSQL
Adobe Illustrator
Docker
Unity 3D
Search Engine Optimization (SEO)

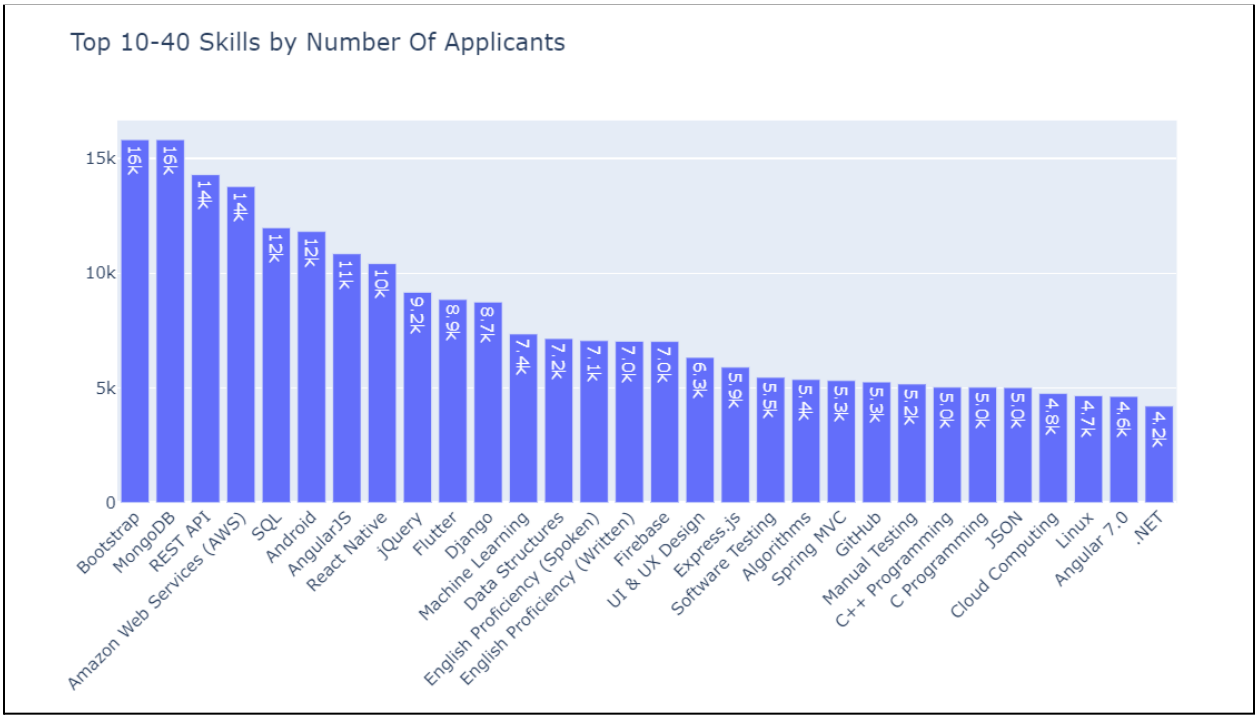



```
skills_openings_dict = {}
def compute_skills_openings(row):
    openings = row['number_of_openings']
    skills_str = row['skills']
    if(skills_str != None):
        split = skills_str.split('\n')
        for s in split:
            skills_openings_dict[s] = skills_openings_dict.get(s, 0) + openings


df.agg(compute_skills_openings,axis=1)
skills_openings_series =
pd.Series(skills_openings_dict).sort_values(ascending=False)

plot_series_pie_chart(skills_openings_series[:10], "Top 10 Skills by Number Of
Opennings")
plot_bar_chart(skills_openings_series[10:40], "Top 10-40 Skills by Number Of
Opennings", 'inside')
```

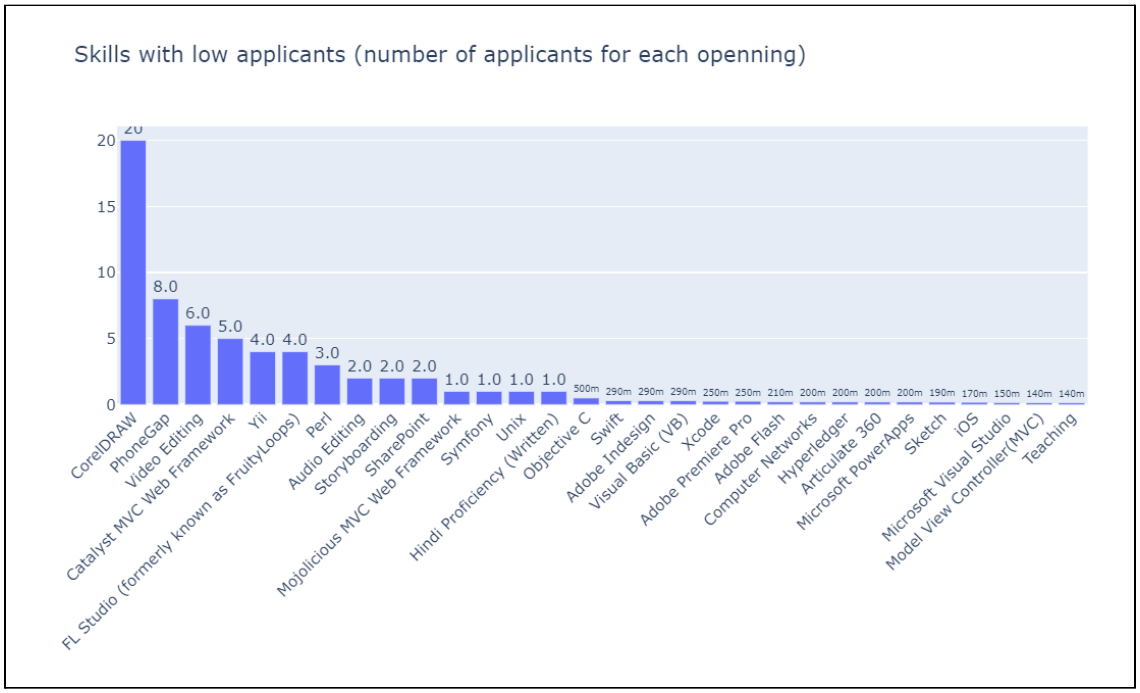


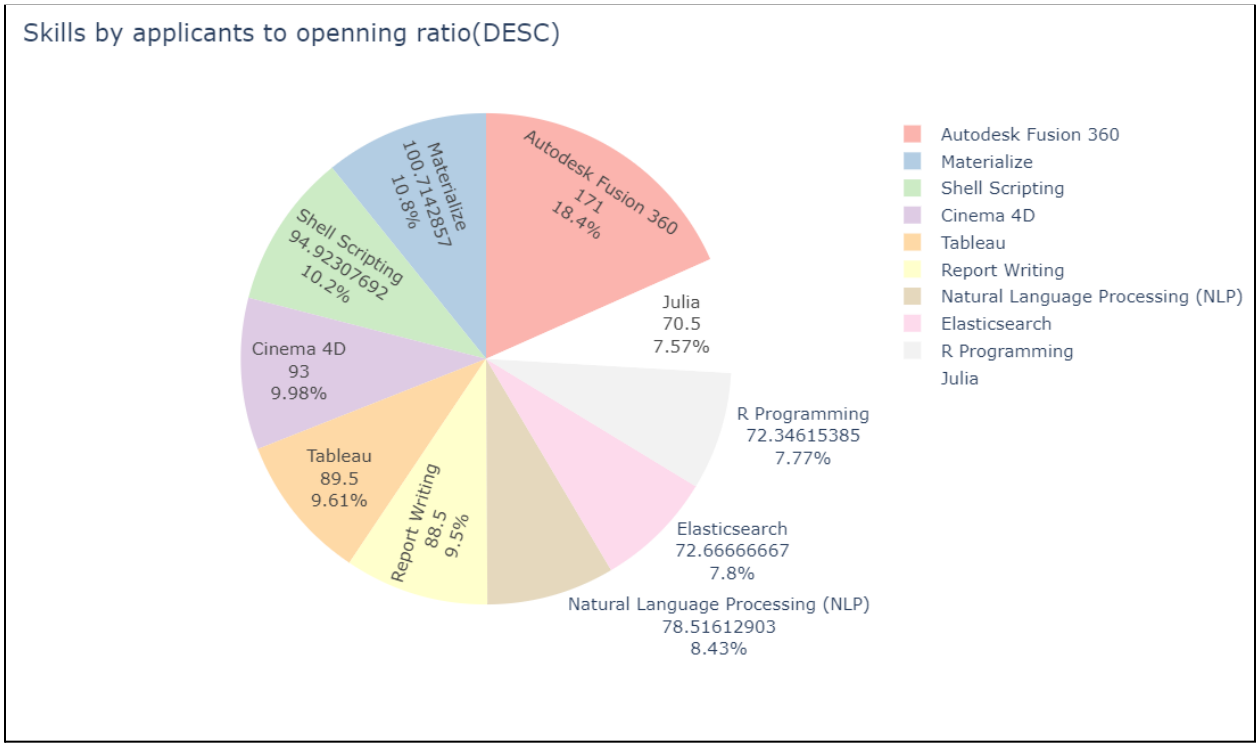

```
skills_applicants_dict = {}
def compute_skills_openings(row):
    openings = row['applicants']
    skills_str = row['skills']
    if(skills_str != None):
        split = skills_str.split('\n')
        for s in split:
            skills_applicants_dict[s] = skills_applicants_dict.get(s, 0) + openings


df.agg(compute_skills_openings,axis=1)
skills_applicants_series =
pd.Series(skills_applicants_dict).sort_values(ascending=False)
plot_series_pie_chart(skills_applicants_series[:10], "Top 10 Skills by Number Of
Applicants")
plot_bar_chart(skills_applicants_series[10:40], "Top 10-40 Skills by Number Of
Applicants", 'inside')
```

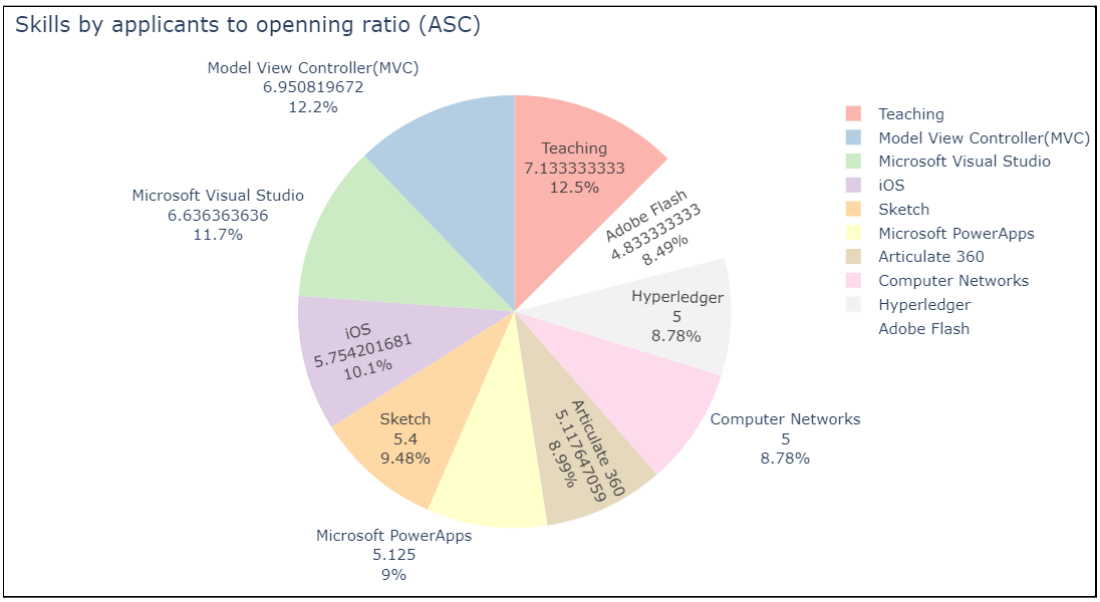

Top 10 Skills by Number Of Applicants
JavaScript
56,605
15.8%
HTML
58,367
16.3%
WordPress
16,819
4.69%
MySQL
17,000
4.74%
CSS
52,638
14.7%
PHP
26,395
7.36%
Python
37,282
10.4%
Java
29,660
8.27%
ReactJS
34,107
9.51%
Node.js
29,854
8.32%
HTML
JavaScript
CSS
Python
ReactJS
Node.js
Java
PHP
MySQL
WordPress


Top 10-40 Skills by Number Of Applicants
15k
10k
5k
0
15k
16k
14k
14k
12k
12k
11k
10k
9.2k
8.9k
8.7k
7.4k
7.2k
7.1k
7.0k
7.0k
6.3k
5.9k
5.5k
5.4k
5.3k
5.3k
5.2k
5.0k
5.0k
5.0k
4.8k
4.7k
4.6k
4.2k
Bootstrap
MongoDB
REST API
Amazon Web Services (AWS)
SQL
Android
AngularJS
React Native
jQuery
Flutter
Django
Machine Learning
Data Structures
English Proficiency (Spoken)
English Proficiency (Written)
Firebase
UI & UX Design
Express.js
Software Testing
Algorithms
Spring MVC
GitHub
Manual Testing
C++ Programming
C Programming
JSON
Cloud Computing
Linux
Angular 7.0
.NET

```
# Number of openning available per applicant
skills_o2a_dict = {}

# number of applicants for unit openning
skills_a2o_dict = {}

for skill in skills_count_series.index:
    a = skills_applicants_dict[skill]
    o = skills_openings_dict[skill]
    skills_a2o_dict[skill] = a/o
    if(a == 0):
        skills_o2a_dict[skill] = o
    else:
        skills_o2a_dict[skill] = o/a

skills_o2a_series = pd.Series(skills_o2a_dict)
skills_a2o_series = pd.Series(skills_a2o_dict)


plot_bar_chart(skills_o2a_series.sort_values(ascending=False)[:30], "Skills with
low applicants (number of applicants for each openning)")
plot_series_pie_chart(skills_a2o_series.sort_values(ascending=False)[:30], "Skills
by applicants to openning ratio(DESC)")
plot_series_pie_chart(skills_a2o_series.sort_values(ascending=True)[:30], "Skills
by applicants to openning ratio (ASC)")
```


Skills with low applicants (number of applicants for each openning)
20
15
10
5
0
8.0
6.0
5.0
4.0 4.0
3.0
2.0 2.0 2.0
1.0 1.0 1.0 1.0
500m 290m 290m 290m 250m 250m 210m 200m 200m 200m 200m 190m 170m 150m 140m 140m
CorelDRAW
PhoneGap
Video Editing
Catalyst MVC Web Framework
Yii
FL Studio (formerly known as FruityLoops)
Perl
Audio Editing
Storyboarding
SharePoint
Mojolicious MVC Web Framework
Symfony
Unix
Hindi Proficiency (Written)
Objective C
Swift
Adobe Indesign
Visual Basic (VB)
Xcode
Adobe Premiere Pro
Adobe Flash
Computer Networks
Hyperledger
Articulate 360
Microsoft PowerApps
Sketch
iOS
Microsoft Visual Studio
Model View Controller(MVC)
Teaching

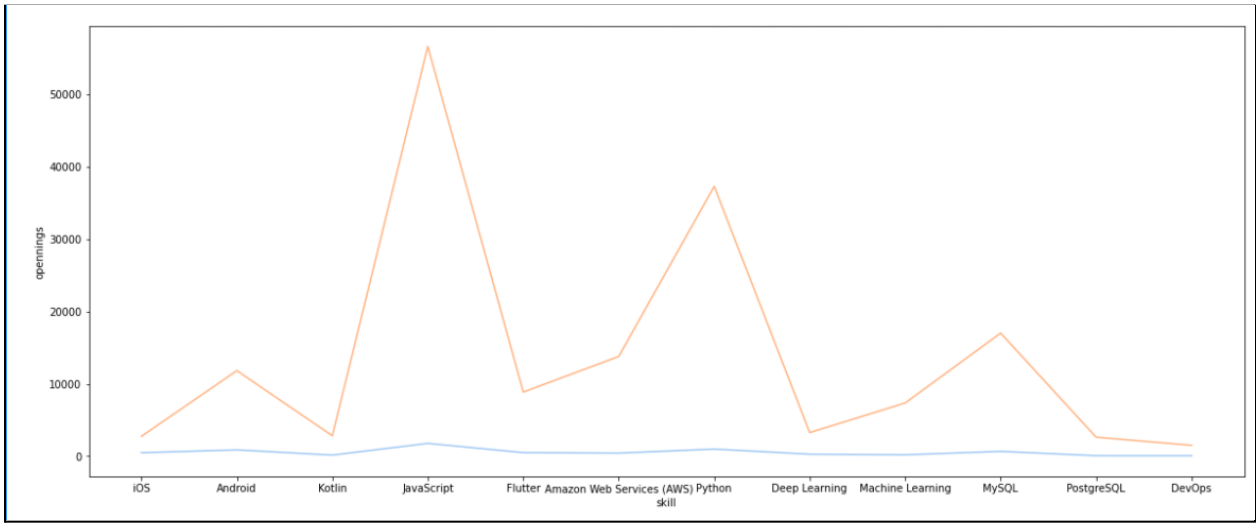

Skills by applicants to openning ratio(DESC)
Autodesk Fusion 360
171
18.4%
Materialize
100.7142857
10.8%
Shell Scripting
94.92307692
10.2%
Cinema 4D
93
9.98%
Tableau
89.5
9.61%
Report Writing
88.5
9.5%
Julia
70.5
7.57%
R Programming
72.34615385
7.77%
Elasticsearch
72.66666667
7.8%
Natural Language Processing (NLP)
78.51612903
8.43%
Autodesk Fusion 360
Materialize
Shell Scripting
Cinema 4D
Tableau
Report Writing
Natural Language Processing (NLP)
Elasticsearch
R Programming
Julia



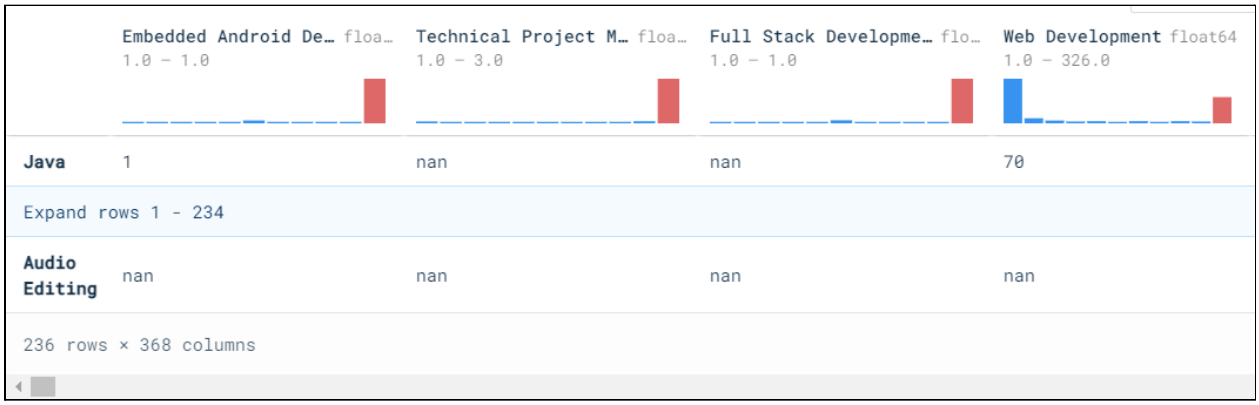

```
SELECT_SKILLS = ['iOS', 'Android', "Kotlin","JavaScript", 'Flutter', 'Amazon Web
Services (AWS)',
'Python','Deep Learning', 'Machine Learning',
'MySQL', 'PostgreSQL', 'DevOps']

skills_oa_dict = []

for skill in SELECT_SKILLS:
    d = {}
    d['skill'] = skill
    d['openings'] = skills_openings_dict[skill]
    d['applicants'] = skills_applicants_dict[skill]
    d['oa_ratio'] = skills_o2a_dict[skill]
    d['ao_ratio'] = skills_a2o_dict[skill]
    skills_oa_dict.append(d)
selected_skills_oa_df = pd.DataFrame(skills_oa_dict)
plt.figure(figsize=(20,8))
ax = sns.lineplot(x=selected_skills_oa_df.skill, y=selected_skills_oa_df.openings)
sns.lineplot(x=selected_skills_oa_df.skill, y=selected_skills_oa_df.applicants,
ax=ax)
selected_skills_oa_df
```

	skill object iOS ..... 8.3% Android ..... 8.3% 10 others 83.3%	openings int64 61 – 1768	applicants int64 1501 – 56605	oa_ratio float64 0.025846823561420214 – 0.1…	ao_ratio float64 5.754201680672269 – 38.689…
0	iOS	476	2739	0.17378605330412558	5.754201680672269
1	Android	869	11819	0.07352567899145444	13.600690448791715
2	Kotlin	148	2818	0.0525195173882186	19.04054054054054
3	JavaScript	1768	56605	0.031233989930218177	32.01640271493213
4	Flutter	490	8853	0.055348469445385746	18.06734693877551
Expand rows 5 - 6					
7	Deep Learning	271	3257	0.08320540374577833	12.018450184501845
8	Machine Learning	190	7351	0.025846823561420214	38.689473684210526
9	MySQL	659	17000	0.038764705882352944	25.796661608497725
10	PostgreSQL	72	2625	0.027428571428571427	36.458333333333336
11	DevOps	61	1501	0.04063957361758828	24.60655737704918

12 rows × 5 columns



```
domain_with_skills = {}
def compute_skills_in_domains(series):
    domain = series['domain_profile']
    skills = series['skills']
    if(skills != None):
        skills = skills.split('\n')

        domain_skills = domain_with_skills.get(domain, {})
        for skill in skills:
            domain_skills[skill] = domain_skills.get(skill, 0) + 1

        domain_with_skills[domain] = domain_skills


_ = df.agg(compute_skills_in_domains, axis=1)
domain_skills_df = pd.DataFrame(domain_with_skills)
domain_skills_df
```

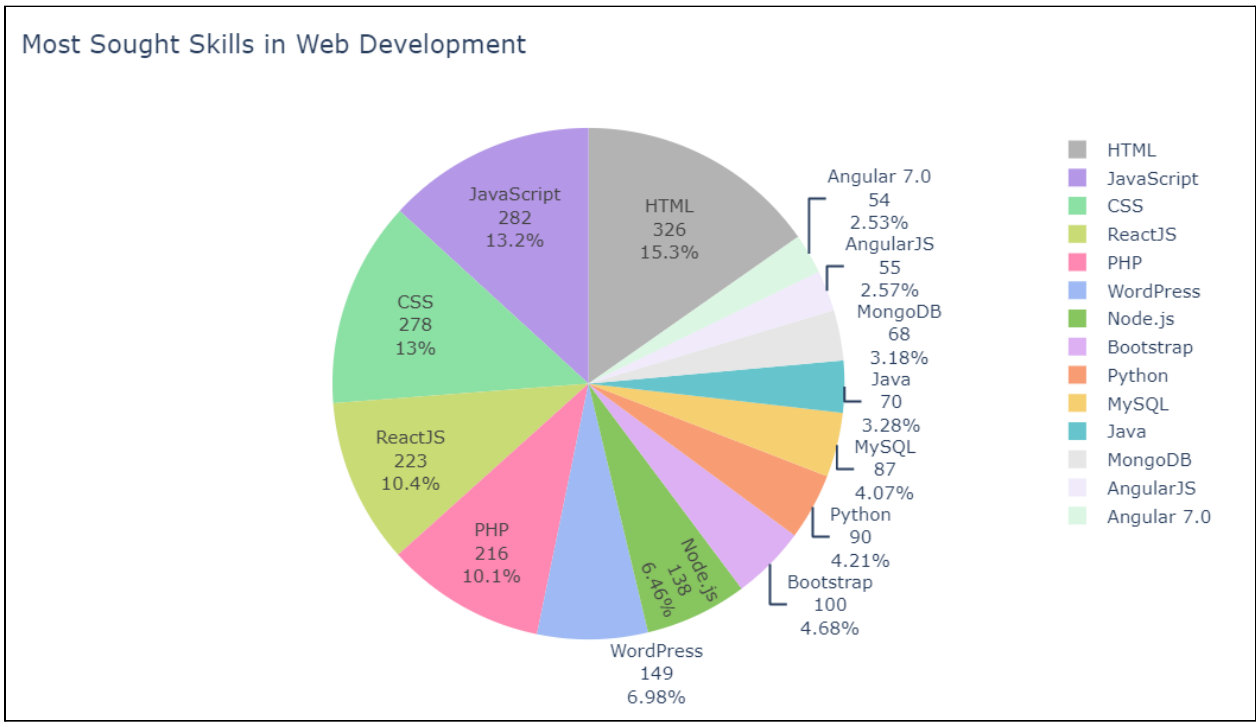


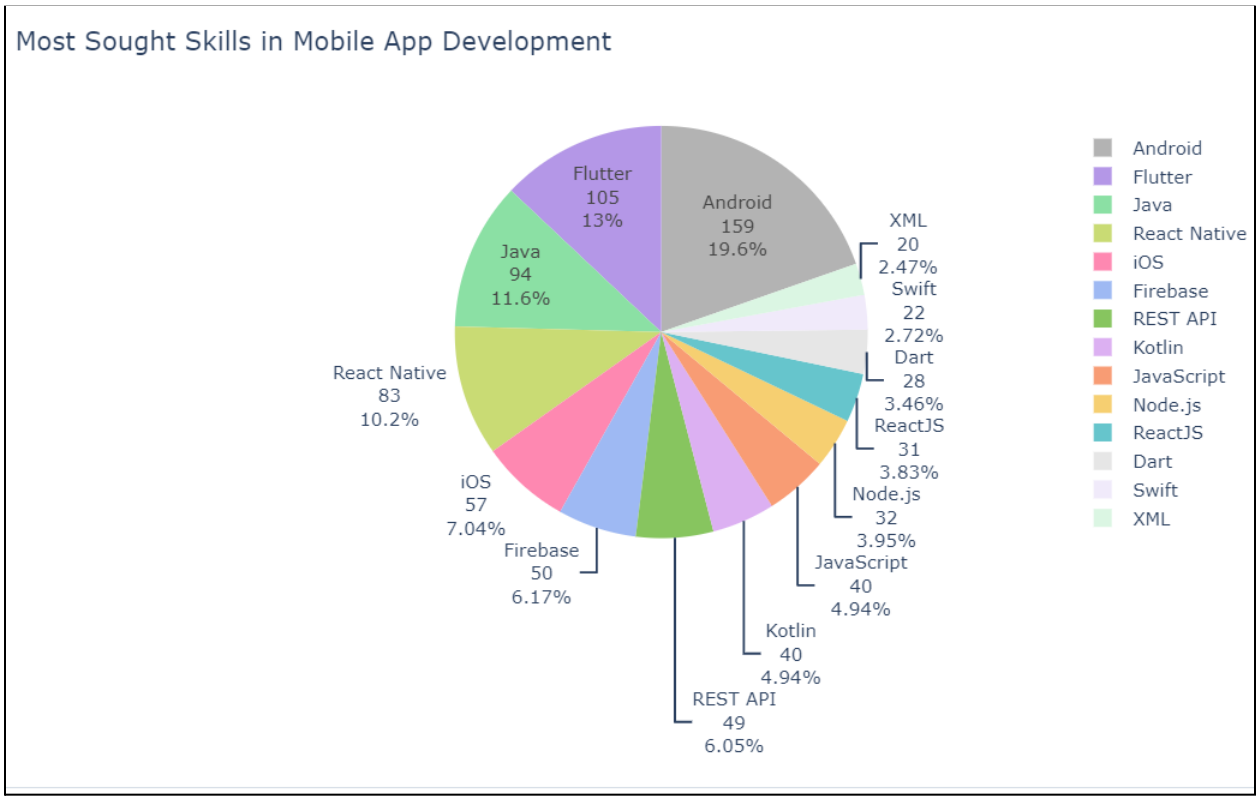
	Embedded Android De… floa… 1.0 – 1.0	Technical Project M… floa… 1.0 – 3.0	Full Stack Developme… flo… 1.0 – 1.0	Web Development float64 1.0 – 326.0
Java	1	nan	nan	70
Expand rows 1 - 234				
Audio Editing	nan	nan	nan	nan
236 rows × 368 columns				

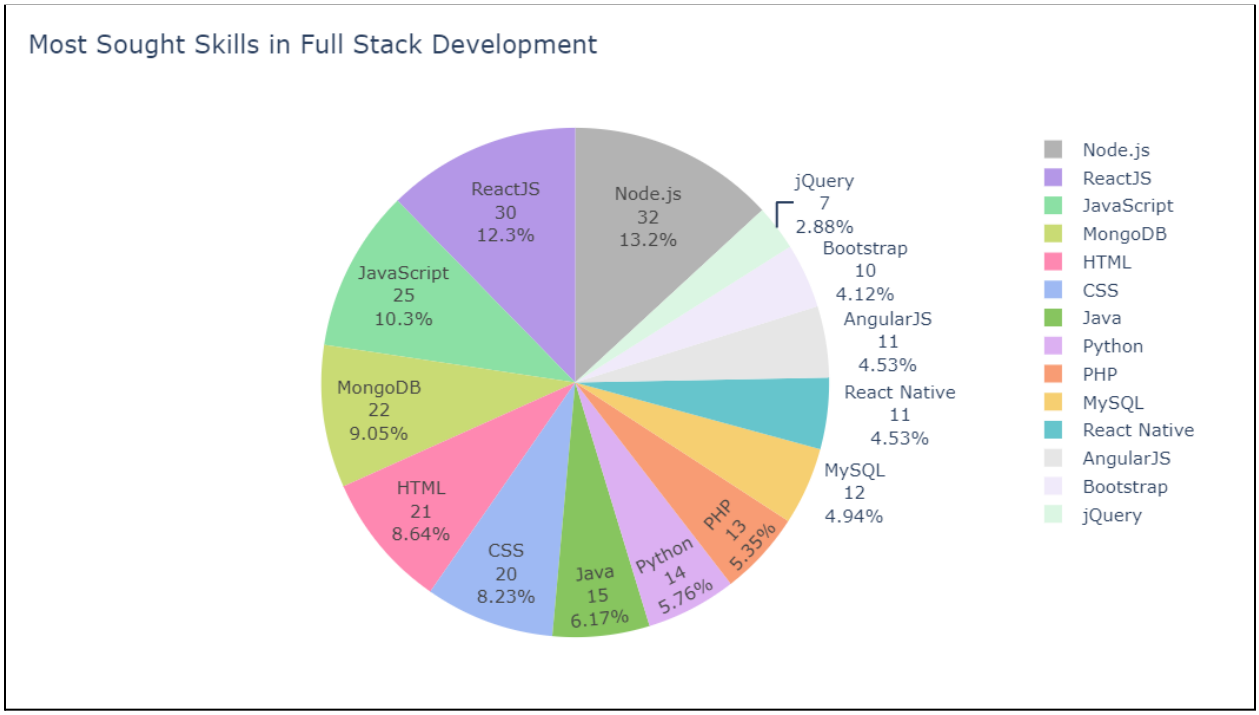
```
df.domain_profile.value_counts()[:20]
```

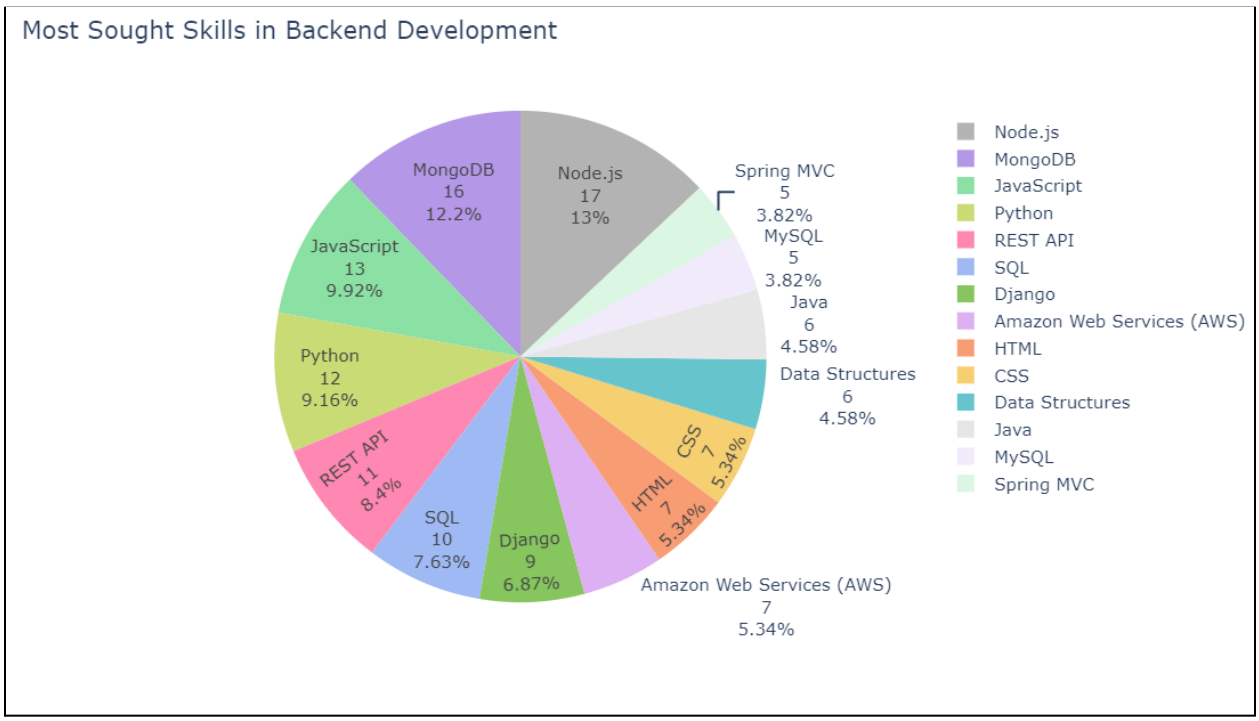
```
Web Development              799
Mobile App Development       344
Full Stack Development        53
Front End Development         49
WordPress Development         45
Android App Development       42
Backend Development           39
PHP Development               39
Flutter Development           37
Software Development          36
Software Testing              34
iOS App Development           32
Python Development            32
Product Management            30
Node.js Development           26
React Native Development      22
ReactJS Development           21
Game Development              16
Java Development              16
Blockchain Development        16
Name: domain_profile, dtype: int64
```

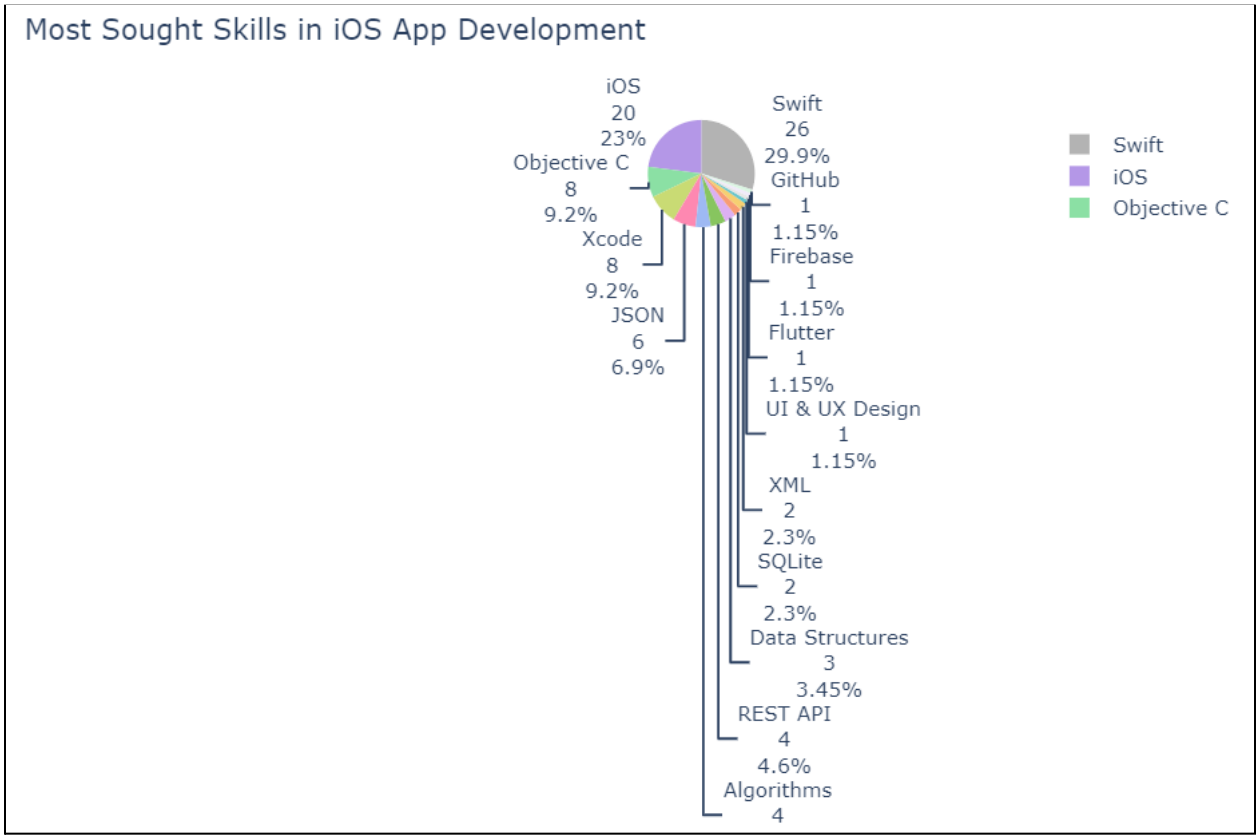
```
selected_domains = ['Web Development', 'Mobile App Development', 'Full Stack
Development',
'Backend Development', 'iOS App Development', 'Game Development']

for domain in selected_domains:
    stack =
domain_skills_df[domain][pd.notna(domain_skills_df[domain])].sort_values(ascending=
False)
    plot_series_pie_chart(stack, f"Most Sought Skills in {domain}", max=14,
color_seq=px.colors.qualitative.Pastel_r)
```

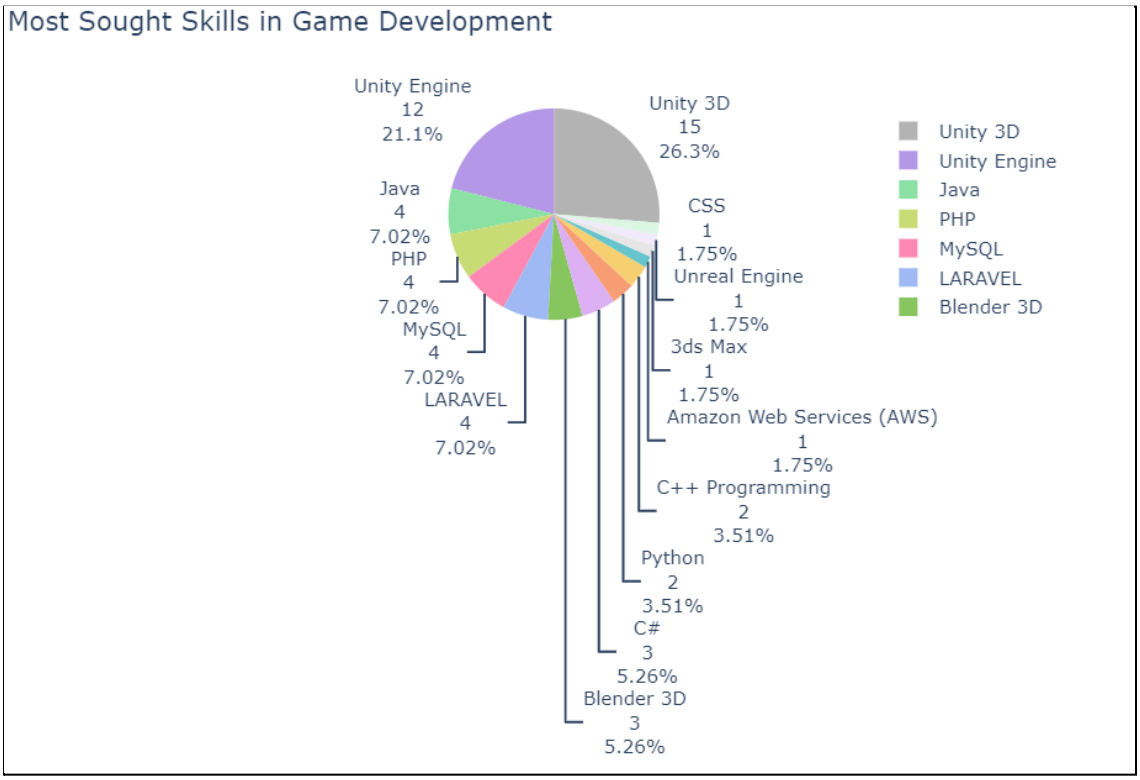

Most Sought Skills in Web Development
HTML 326 15.3%
JavaScript 282 13.2%
CSS 278 13%
ReactJS 223 10.4%
PHP 216 10.1%
WordPress 149 6.98%
Node.js 138 6.46%
Bootstrap 100 4.68%
Python 90 4.21%
MySQL 87 4.07%
Java 70 3.28%
MongoDB 68 3.18%
AngularJS 55 2.57%
Angular 7.0 54 2.53%
HTML
JavaScript
CSS
ReactJS
PHP
WordPress
Node.js
Bootstrap
Python
MySQL
Java
MongoDB
AngularJS
Angular 7.0


Most Sought Skills in Mobile App Development
Android 159 19.6%
Flutter 105 13%
Java 94 11.6%
React Native 83 10.2%
iOS 57 7.04%
Firebase 50 6.17%
REST API 49 6.05%
Kotlin 40 4.94%
JavaScript 40 4.94%
Node.js 32 3.95%
ReactJS 31 3.83%
Dart 28 3.46%
Swift 22 2.72%
XML 20 2.47%
Android
Flutter
Java
React Native
iOS
Firebase
REST API
Kotlin
JavaScript
Node.js
ReactJS
Dart
Swift
XML


Most Sought Skills in Full Stack Development
Node.js 32 13.2%
ReactJS 30 12.3%
JavaScript 25 10.3%
MongoDB 22 9.05%
HTML 21 8.64%
CSS 20 8.23%
Java 15 6.17%
Python 14 5.76%
PHP 13 5.35%
MySQL 12 4.94%
React Native 11 4.53%
AngularJS 11 4.53%
Bootstrap 10 4.12%
jQuery 7 2.88%
Node.js
ReactJS
JavaScript
MongoDB
HTML
CSS
Java
Python
PHP
MySQL
React Native
AngularJS
Bootstrap
jQuery


Most Sought Skills in Backend Development
Node.js 17 13%
MongoDB 16 12.2%
JavaScript 13 9.92%
Python 12 9.16%
REST API 11 8.4%
SQL 10 7.63%
Django 9 6.87%
Amazon Web Services (AWS) 7 5.34%
HTML 7 5.34%
CSS 7 5.34%
Data Structures 6 4.58%
Java 6 4.58%
MySQL 5 3.82%
Spring MVC 5 3.82%
Node.js
MongoDB
JavaScript
Python
REST API
SQL
Django
Amazon Web Services (AWS)
HTML
CSS
Data Structures
Java
MySQL
Spring MVC


Most Sought Skills in iOS App Development
iOS
20
23%
Objective C
8
9.2%
Xcode
8
9.2%
JSON
6
6.9%
Swift
26
29.9%
GitHub
1
1.15%
Firebase
1
1.15%
Flutter
1
1.15%
UI & UX Design
1
1.15%
XML
2
2.3%
SQLite
2
2.3%
Data Structures
3
3.45%
REST API
4
4.6%
Algorithms
4
Swift
iOS
Objective C


Most Sought Skills in Game Development
Unity Engine
12
21.1%
Java
4
7.02%
PHP
4
7.02%
MySQL
4
7.02%
LARAVEL
4
7.02%
Unity 3D
15
26.3%
CSS
1
1.75%
Unreal Engine
1
1.75%
3ds Max
1
1.75%
Amazon Web Services (AWS)
1
1.75%
C++ Programming
2
3.51%
Python
2
3.51%
C#
3
5.26%
Blender 3D
3
5.26%
Unity 3D
Unity Engine
Java
PHP
MySQL
LARAVEL
Blender 3D

```
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori, fpmax, fpgrowth

super_skills_list = []
def aggregate_skills(series):
    if(type(series) == str):
        skills = series.split('\n')
        super_skills_list.append(skills)
```

```
df.skills.agg(aggregate_skills)
# super_skills_list

te = TransactionEncoder()
te_ary = te.fit(super_skills_list).transform(super_skills_list)
freq_skills_df = pd.DataFrame(te_ary, columns=te.columns_)

frequent_itemsets = fpgrowth(freq_skills_df, min_support=0.05, use_colnames=True)

# ### alternatively:
# #frequent_itemsets = apriori(df, min_support=0.6, use_colnames=True)
# #frequent_itemsets = fpmax(df, min_support=0.6, use_colnames=True)
```

```
frequent_itemsets.sort_values(by='support', ascending=True)
```

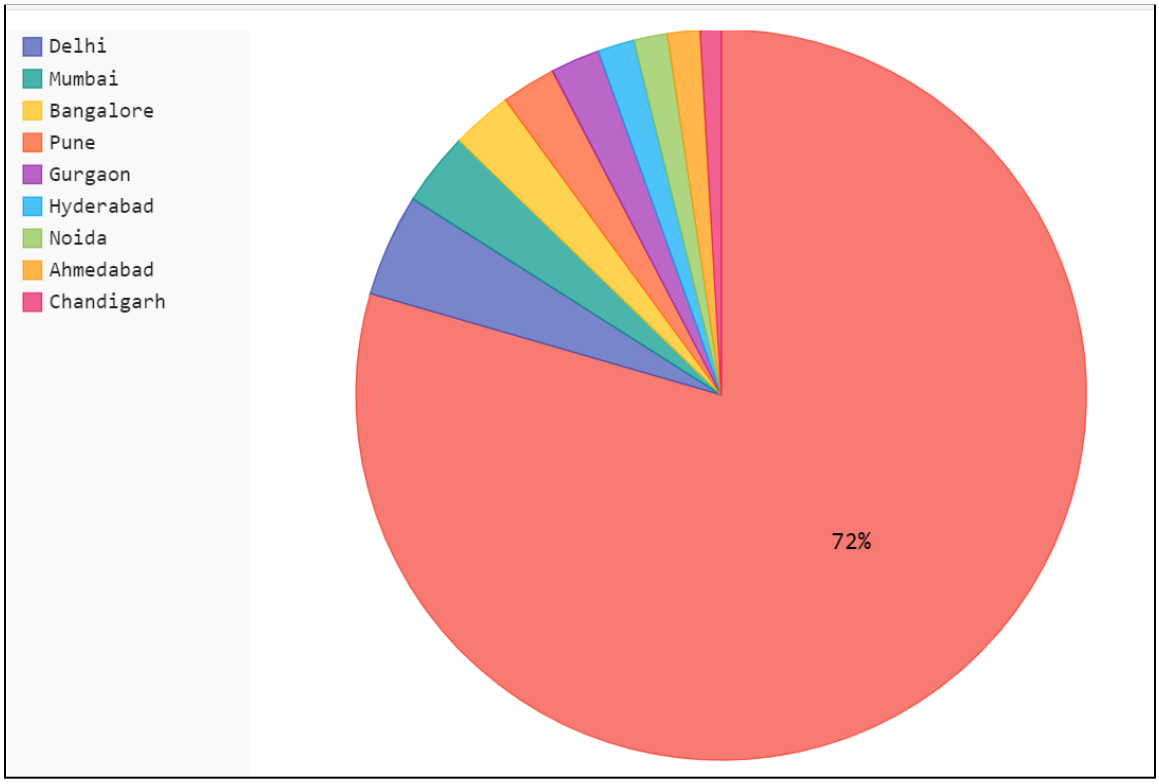
	support float64 0.050645007166746296 – 0.2…	itemsets object frozenset({'CSS', … 2.1% frozenset({'CSS', … 2.1% 45 others 95.7%
36	0.050645007166746296	frozenset({'CSS', 'Node.js', 'HTML'})
31	0.050645007166746296	frozenset({'CSS', 'HTML', 'PHP', 'JavaScript'})
45	0.050645007166746296	frozenset({'ReactJS', 'React Native'})
23	0.051600573339703776	frozenset({'ReactJS', 'HTML', 'JavaScript'})
30	0.05351170568561873	frozenset({'CSS', 'PHP', 'JavaScript'})
Expand rows 5 - 41		
2	0.19875776397515527	frozenset({'ReactJS'})
39	0.22981366459627328	frozenset({'CSS', 'HTML'})
11	0.2508361204013378	frozenset({'CSS'})
10	0.2814142379359771	frozenset({'HTML'})
9	0.2904921165790731	frozenset({'JavaScript'})
47 rows × 2 columns		

Location Analysis

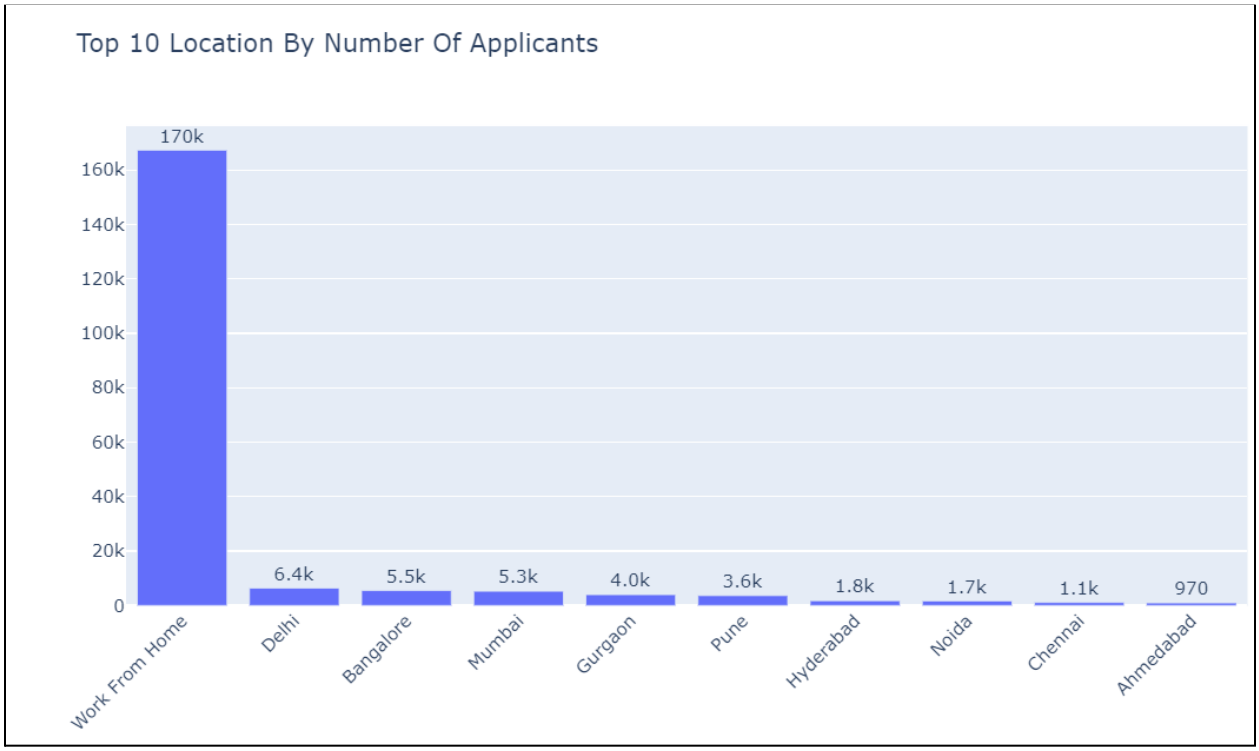
```
all_location_series =
df.location.value_counts()/df.location.value_counts().values.sum()
all_locations = all_location_series.sort_values(ascending=False).to_dict()
display("Top 5 Locations")
display(df.location.value_counts()[:5])
```

```
'Top 5 Locations'

Work From Home    1715
Delhi               98
Mumbai              70
Bangalore           58
Pune                52
Name: location, dtype: int64
```

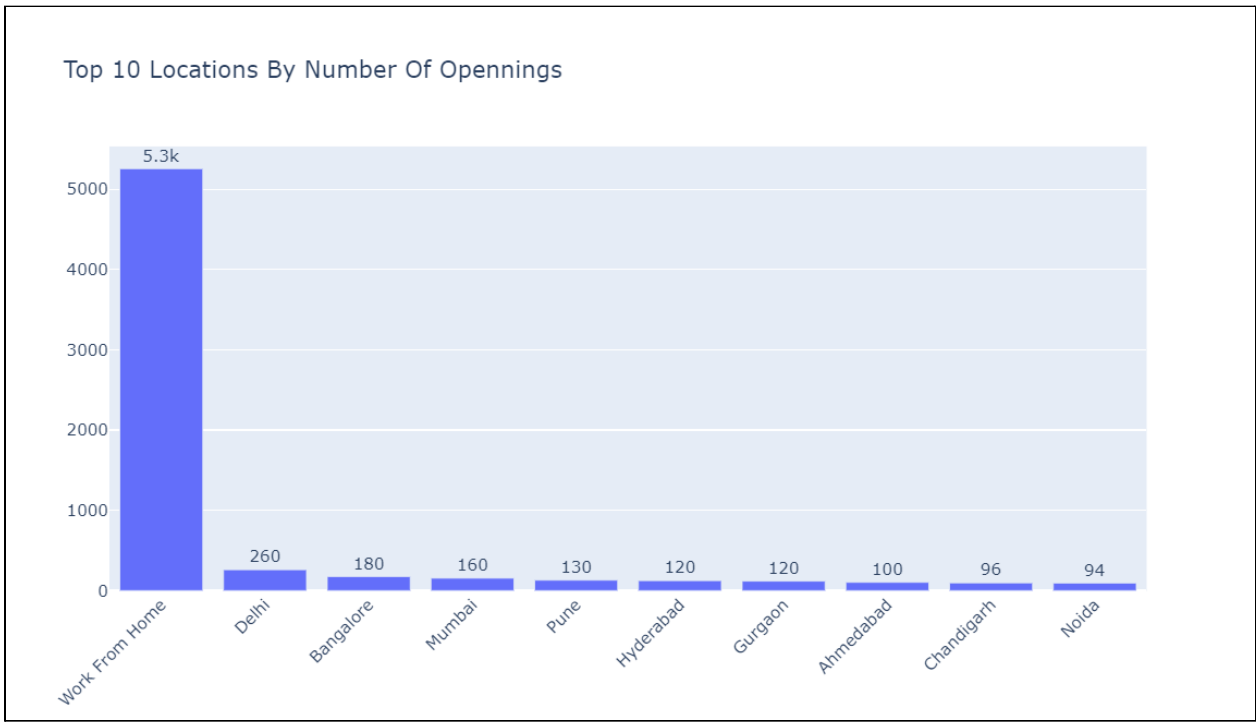
```
top_10_locations = list(all_locations.keys())[:10]
pie_chart = pygal.Pie(print_values=True)
pie_chart.value_formatter = lambda x: f"{math.floor(x*100)}%"
pie_chart.title = 'Top 10 Locations'
for location in top_10_locations:
    pie_chart.add(location, all_locations[location])

# pie_chart.render()
# pie_chart
display(SVG(pie_chart.render(disable_xml_declaration=True)))
```



```
location_group_by = df.groupby(by='location')
top_10_applicant_by_location =
location_group_by.applicants.sum().sort_values(ascending=False)[:10]

label = top_10_applicant_by_location.index.tolist()
value = top_10_applicant_by_location.values.tolist()

fig = go.Figure(
    data=[go.Bar(
        x = label,
        y = value,
        text= value)],
    layout_title_text="Top 10 Location By Number Of Applicants",
)
fig.update_traces(texttemplate='%{text:.2s}', textposition='outside')
fig.update_layout(xaxis_tickangle=-45)
fig.show(renderer="colab")
```

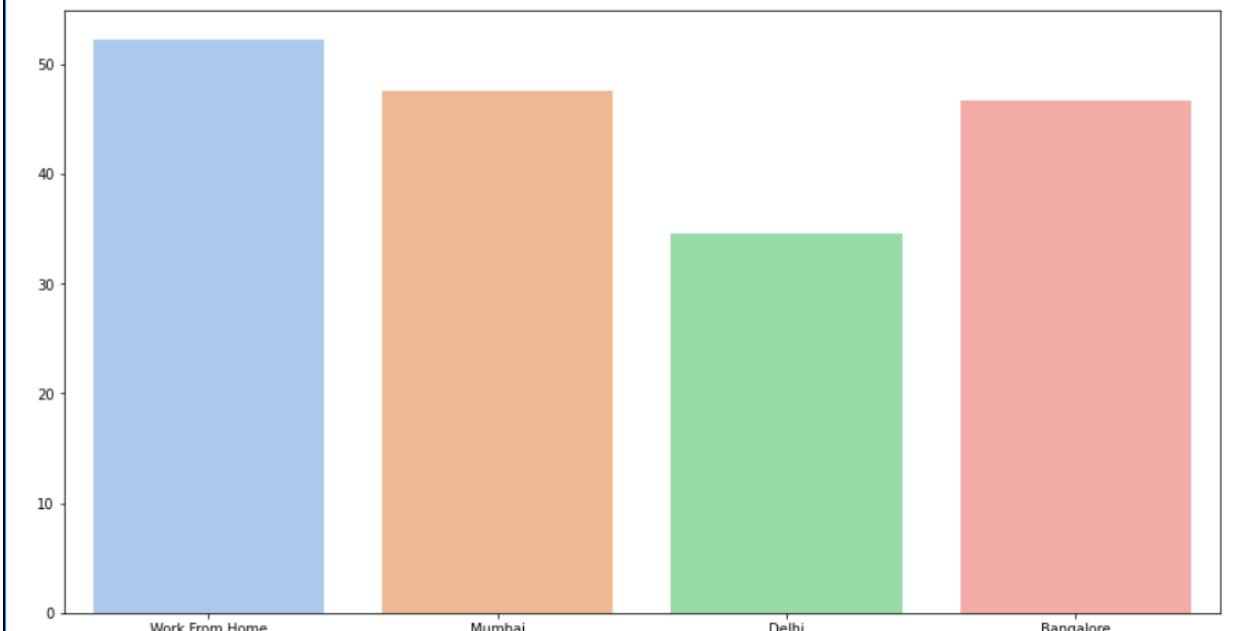


```
 location_with_openings =
location_group_by.number_of_openings.sum().sort_values(ascending=False)[:10]
# sns.barplot(y=location_with_openings.index[:20],
x=location_with_openings.values[:20])

fig = go.Figure(
    data=[go.Bar(
        x = location_with_openings.index.tolist(),
        y = location_with_openings.values.tolist(),
        text= location_with_openings.values)],
    layout_title_text="Top 10 Locations By Number Of Openings",
)
fig.update_traces(texttemplate='%{text:.2s}', textposition='outside')
fig.update_layout(xaxis_tickangle=-45)
fig.show(renderer="colab")
```

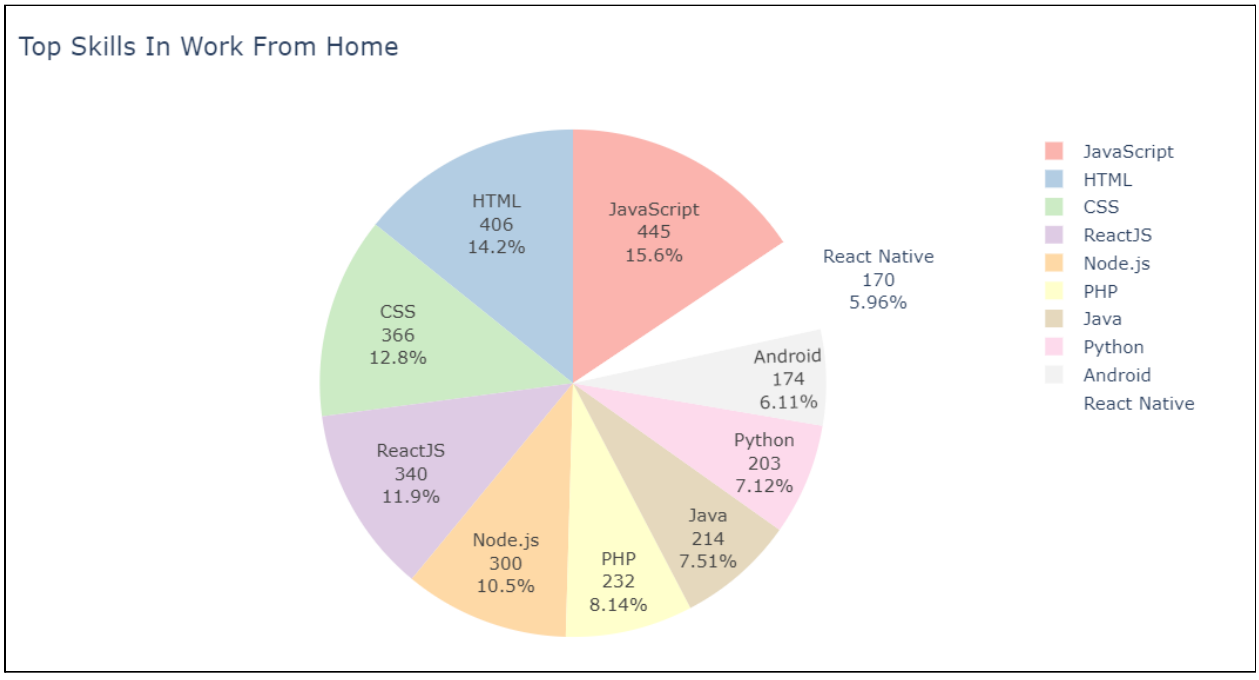


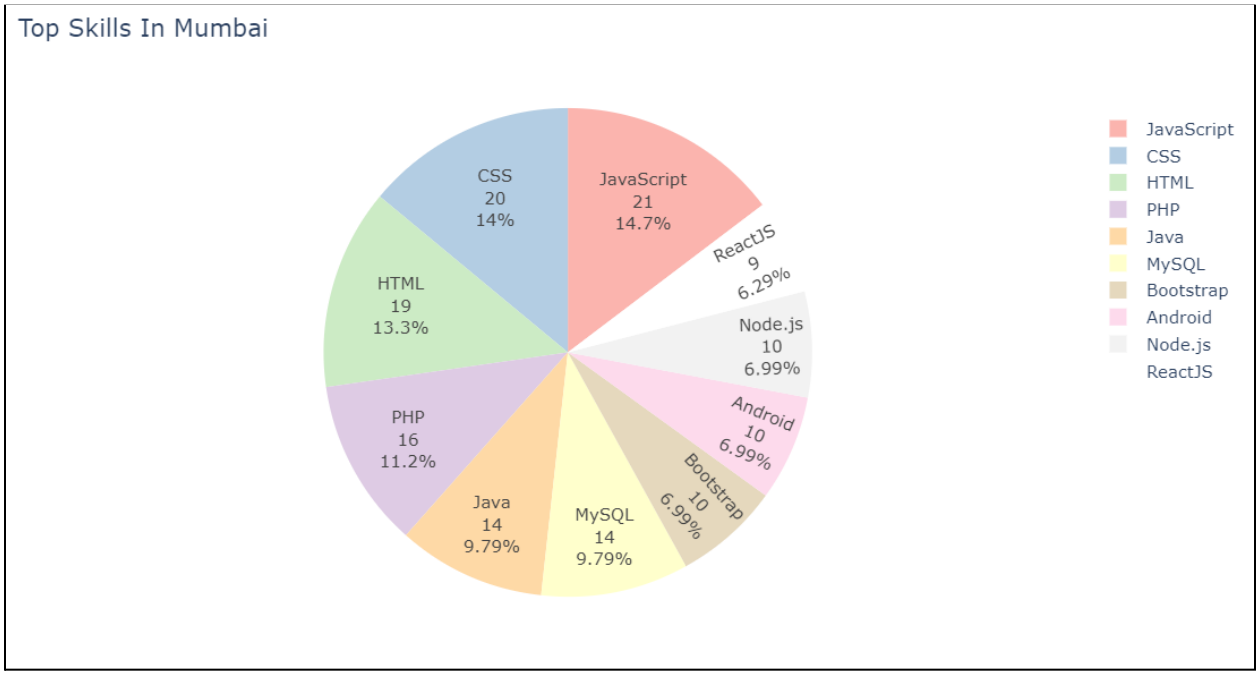
```
def compute_openning_to_applicants_ratio(series):
    return series.applicants/series.number_of_openings

# location_group_by.agg(compute_openning_to_applicants_ratio)
# df.applicants/df.number_of_openings
# df.apply(lambda row: print(row))

cols = {"Work From Home": 0, "Mumbai":0, "Delhi":0, 'Bangalore':0}

for col in cols:
    cols[col] =
location_group_by.apply(compute_openning_to_applicants_ratio)[col].mean()

title = "Mean ratio of Number of applicants vs openning"
plt.figure(figsize=(15,8))
sns.barplot(x=list(cols.keys()), y=list(cols.values()))
display(pd.Series(cols))
```

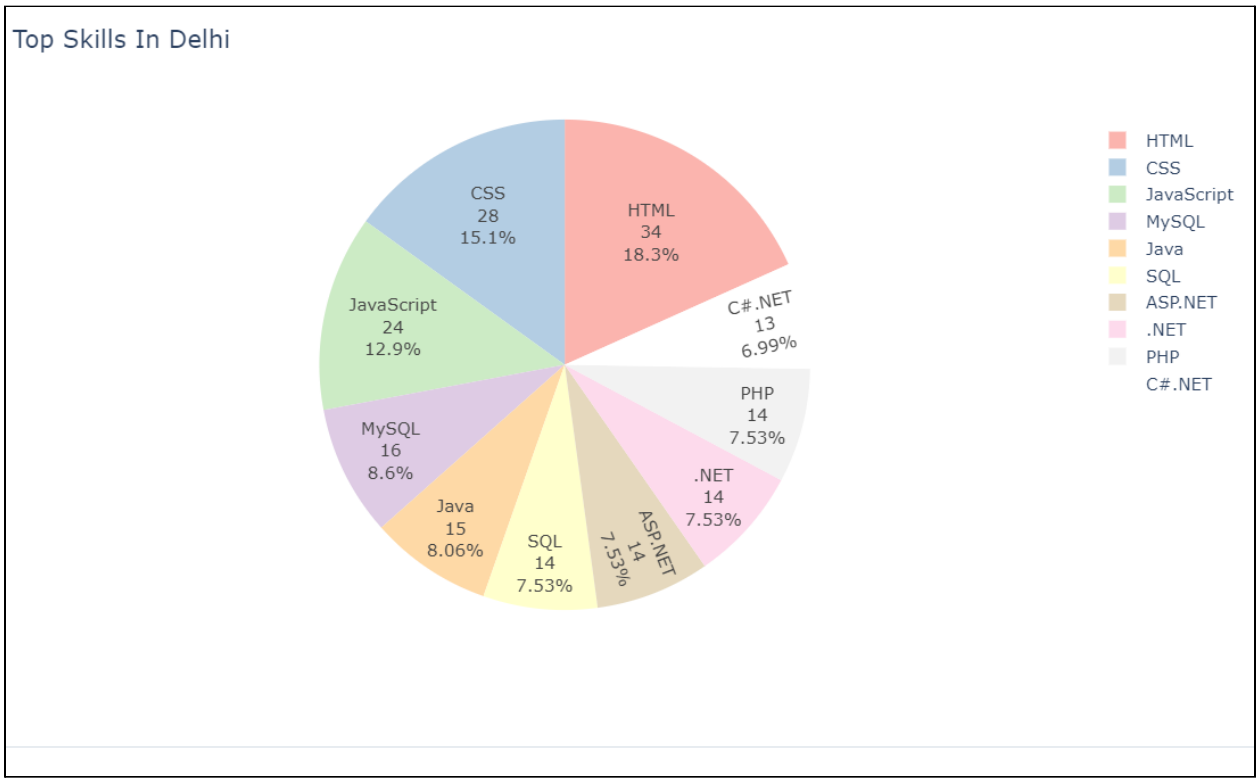
```
Work From Home    52.317927
Mumbai            47.648095
Delhi             34.532483
Bangalore         46.670977
dtype: float64
```

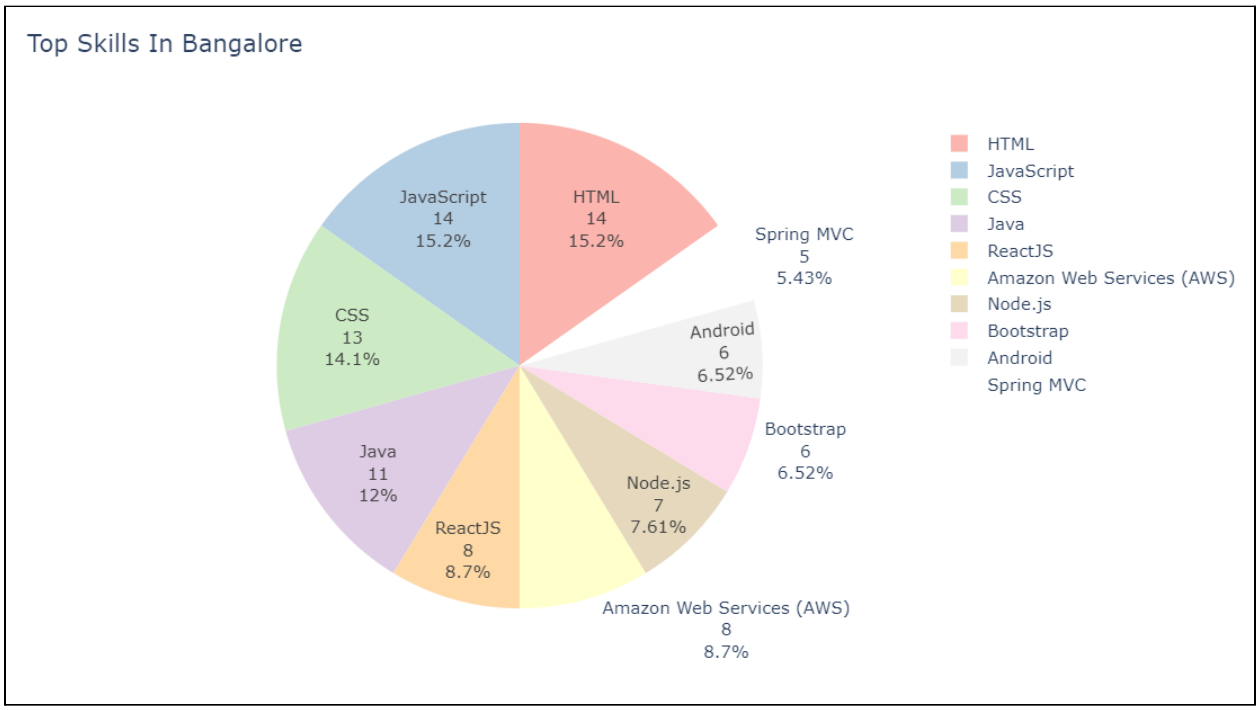


```
# location(top 5) vs most popular skill (top 5 skills)
def skills_counter(series):
    skills_super_set = {}
    def counter(input_string):
        if(type(input_string) == str):
            for i in input_string.split('\n'):
                skills_super_set[i] = skills_super_set.get(i, 0) + 1
        return None
    series.apply(counter)
    return skills_super_set

location_gb_ag_skills = location_group_by.skills.agg(skills_counter)

wfh_series = pd.Series(location_gb_ag_skills['Work From Home'])
mumbai_series = pd.Series(location_gb_ag_skills['Mumbai'])
delhi_series = pd.Series(location_gb_ag_skills['Delhi'])
bangalore_series = pd.Series(location_gb_ag_skills['Bangalore'])

plot_series_pie_chart(wfh_series, "Top Skills In Work From Home")
plot_series_pie_chart(mumbai_series, "Top Skills In Mumbai")
plot_series_pie_chart(delhi_series, "Top Skills In Delhi")
plot_series_pie_chart(bangalore_series, "Top Skills In Bangalore")
```



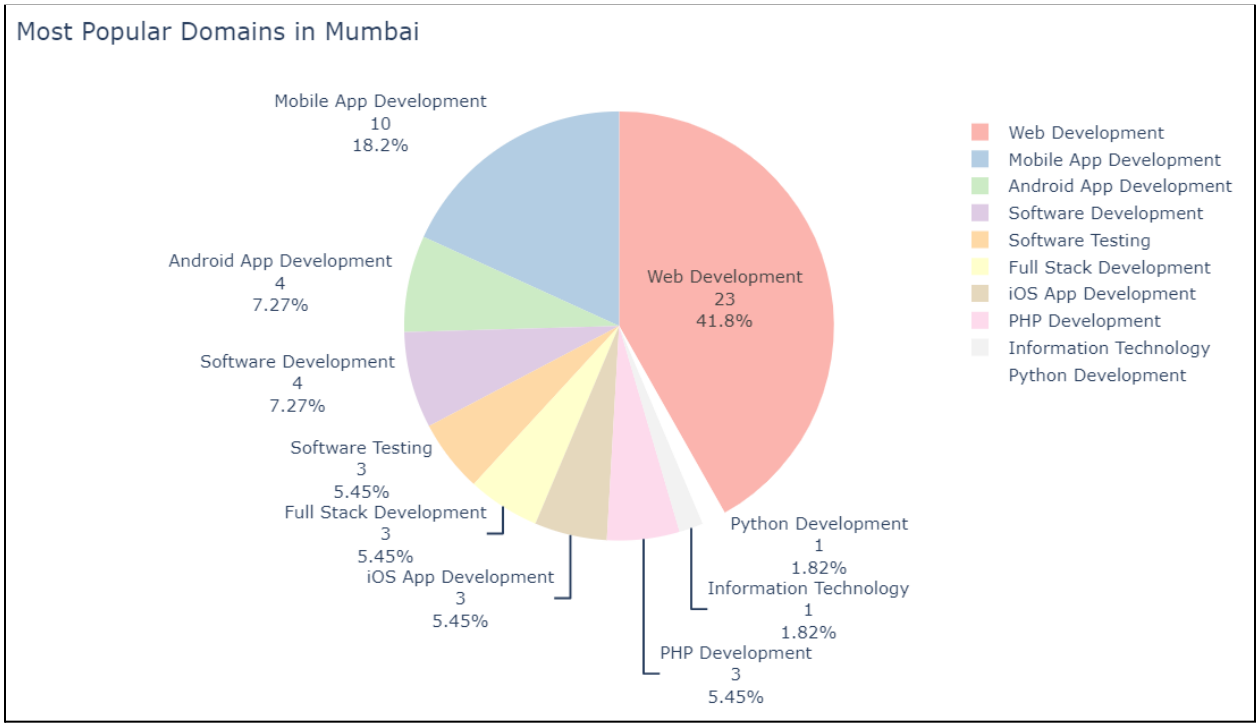

Top Skills In Mumbai
JavaScript 21 14.7%
ReactJS 9 6.29%
Node.js 10 6.99%
Android 10 6.99%
Bootstrap 10 6.99%
MySQL 14 9.79%
Java 14 9.79%
PHP 16 11.2%
HTML 19 13.3%
CSS 20 14%
JavaScript
CSS
HTML
PHP
Java
MySQL
Bootstrap
Android
Node.js
ReactJS


Top Skills In Delhi
HTML 34 18.3%
C#.NET 13 6.99%
PHP 14 7.53%
.NET 14 7.53%
ASP.NET 14 7.53%
SQL 14 7.53%
Java 15 8.06%
MySQL 16 8.6%
JavaScript 24 12.9%
CSS 28 15.1%
HTML
CSS
JavaScript
MySQL
Java
SQL
ASP.NET
.NET
PHP
C#.NET

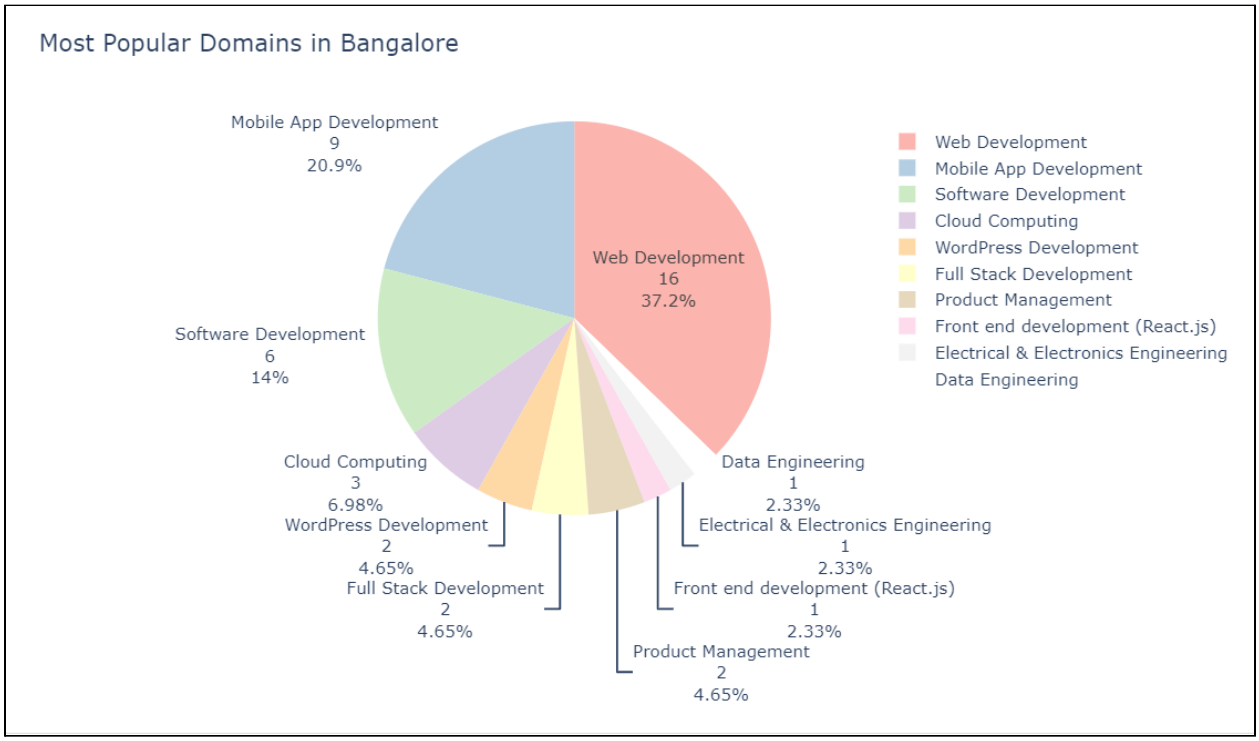


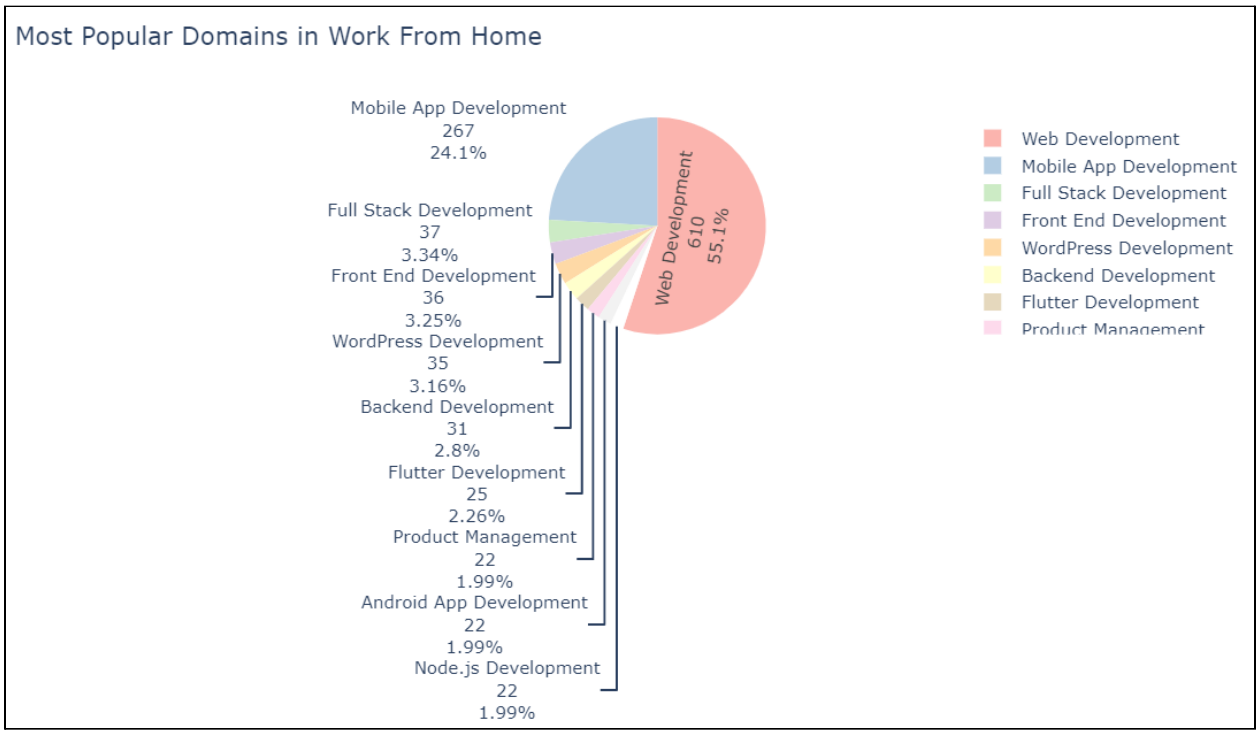

```
# location vs domain_profile

def location_per_domain(series):
    return series.value_counts().to_dict()

location_domain_groupby = location_group_by.domain_profile.agg(location_per_domain)

cols = ['Mumbai', 'Bangalore', 'Work From Home']

for col in cols:
    series = pd.Series(location_domain_groupby[col])
    plot_series_pie_chart(series, f"Most Popular Domains in {col}")
```

Most Popular Domains in Mumbai



Most Popular Domains in Bangalore

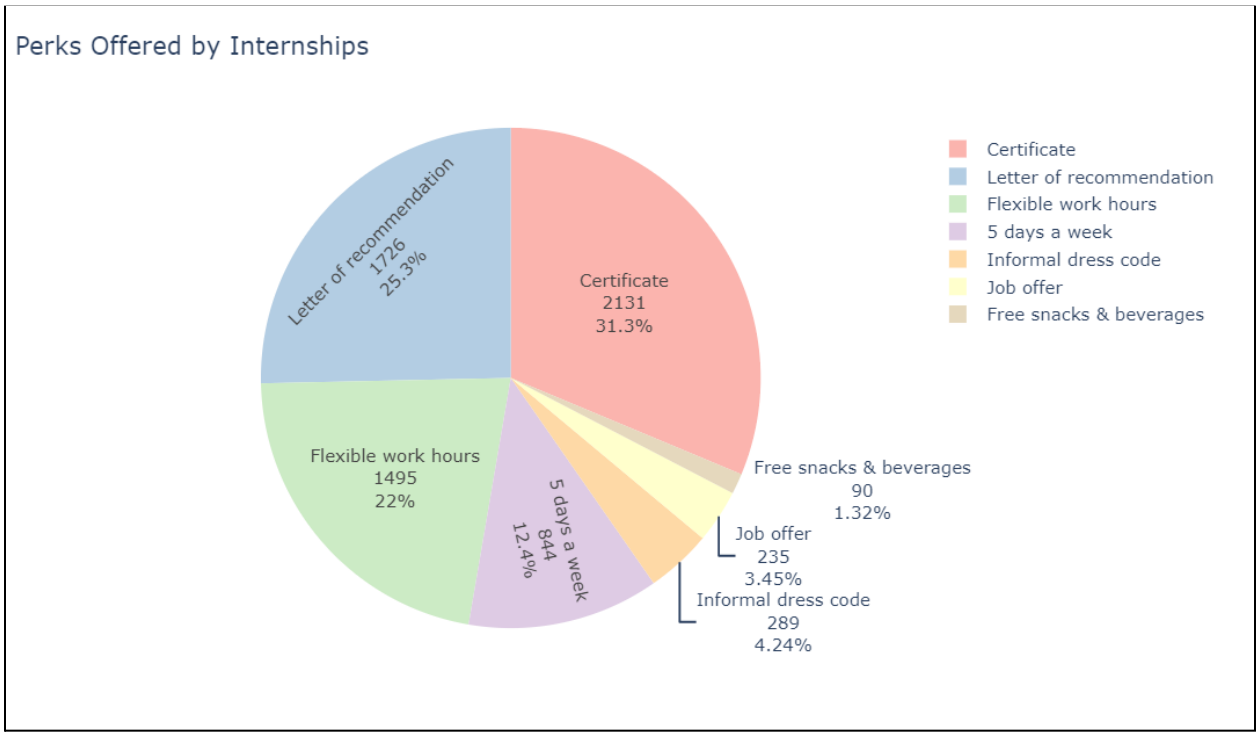



Most Popular Domains in Work From Home
Mobile App Development
267
24.1%
Full Stack Development
37
3.34%
Front End Development
36
3.25%
WordPress Development
35
3.16%
Backend Development
31
2.8%
Flutter Development
25
2.26%
Product Management
22
1.99%
Android App Development
22
1.99%
Node.js Development
22
1.99%
Web Development
610
55.1%
Web Development
Mobile App Development
Full Stack Development
Front End Development
WordPress Development
Backend Development
Flutter Development
Product Management

Perks

```
all_perks = {}

def extract_perks(series):
    if(type(series) == str):
        perks = series.split("\n")
        for perk in perks:
            perk = perk.strip()
            all_perks[perk] = all_perks.get(perk, 0) + 1

_ = df.perks.apply(extract_perks)
```

```
perks_series = pd.Series(data=list(all_perks.values()),
index=list(all_perks.keys()))

plot_series_pie_chart(perks_series, 'Perks Offered by Internships')
```



Who Can Apply

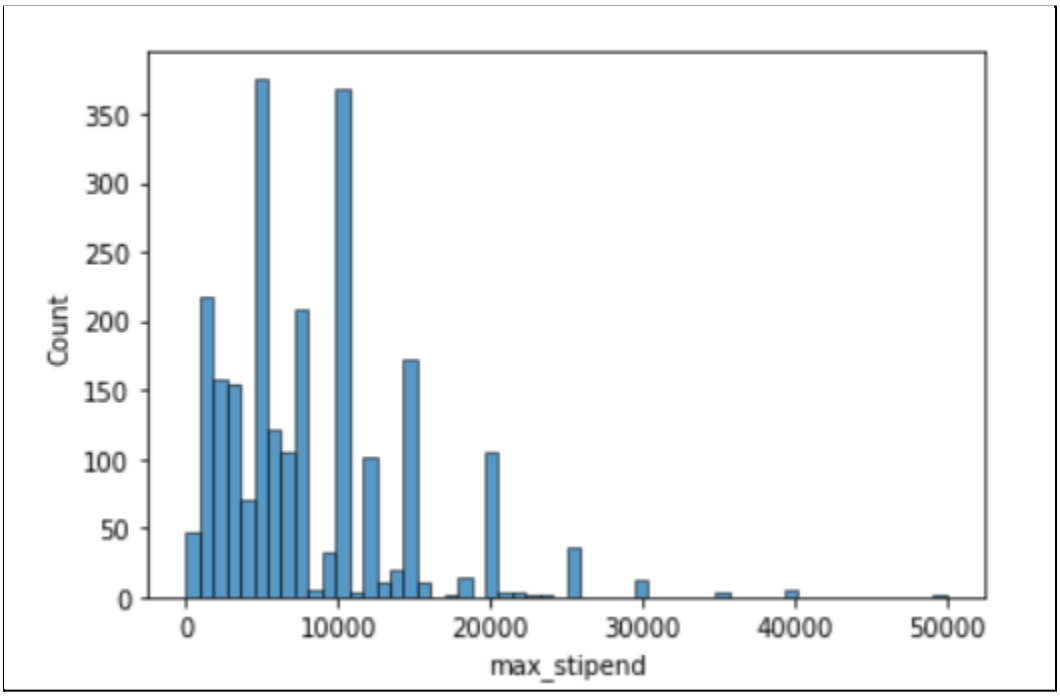
```
all_application_features = {}

def extract_perks(series):
    if(type(series) == str):
        application_data = series.split("\n")
        for application in application_data:
            application = application.strip()
            all_application_features[application] =
all_application_features.get(application, 0) + 1

_ = df.who_can_apply.apply(extract_perks)
all_application_features
```

```
{'Only those candidates can apply who:': 2374,
 '1. are available for the work from home job/internship': 1715,
 "2. can start the work from home job/internship between 19th May'21 and 23rd Jun'21": 144,
 '3. are available for duration of 5 months': 24,
 '4. have relevant skills and interests': 2123,
 '* Women wanting to start/restart their career can also apply.': 1324,
 "2. can start the work from home job/internship between 26th May'21 and 30th Jun'21": 174,
 '3. are available for duration of 4 months': 126,
 "2. can start the work from home job/internship between 23rd May'21 and 27th Jun'21": 54,
 '3. are available for duration of 1 month': 179,
 '1. are available for full time (in-office) internship': 599,
 "2. can start the internship between 23rd May'21 and 27th Jun'21": 11,
 '3. are available for duration of 6 months': 826,
 "2. can start the work from home job/internship between 25th May'21 and 29th Jun'21": 97,
 "2. can start the work from home job/internship between 28th May'21 and 2nd Jul'21": 119,
 '3. are available for duration of 3 months': 792,
 "2. can start the work from home job/internship between 18th May'21 and 22nd Jun'21": 89,
 '3. are available for duration of 3 weeks': 3,
 "2. can start the work from home job/internship between 29th May'21 and 3rd Jul'21": 103,
 '3. are available for duration of 2 months': 406,
 "2. can start the internship between 19th May'21 and 23rd Jun'21": 38,
 '4.  are from Delhi, Gurgaon, Noida and neighboring cities': 4,
 '5. have relevant skills and interests': 251,
 "2. can start the internship between 27th May'21 and 1st Jul'21": 78,
 "2. can start the work from home job/internship between 20th May'21 and 24th Jun'21": 110,
 "2. can start the work from home job/internship between 10th May'21 and 16th Jun'21": 1,
 "2. can start the internship between 28th May'21 and 2nd Jul'21": 38,
 '1. are available for the part time job/internship': 60,
 "2. can start the part time job/internship between 25th May'21 and 29th Jun'21": 3,
 '4.  are from Jaipur': 8,
 "2. can start the work from home job/internship between 31st May'21 and 5th Jul'21": 107,
```

Stipend

```
plot_bar_chart(df.stipend.value_counts()[:20], "Most common stipend Packages",
xaxis_label='Package in INR', yaxis_label='Count')
```

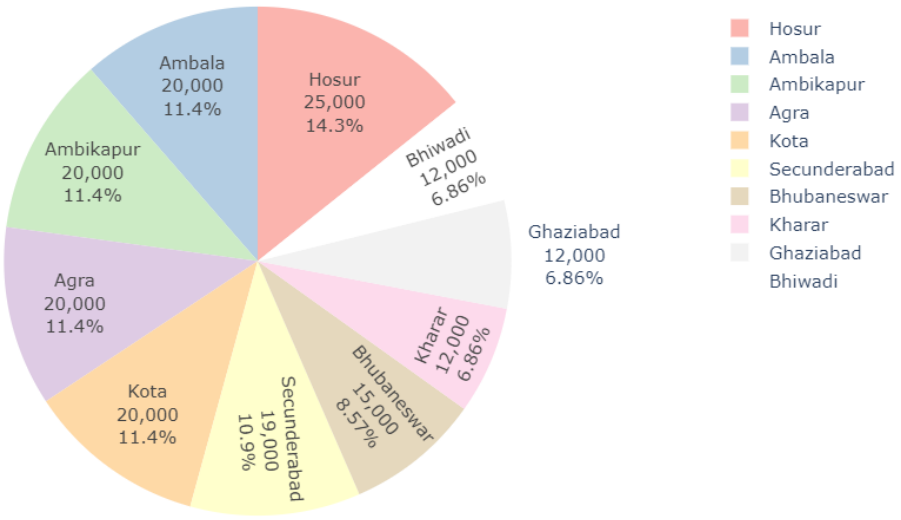
```
display("Number of Unpaid Internships = " + str(len(df[df.min_stipend == 0])))
stipend_sorted_df = df.sort_values(by='max_stipend', ascending=False)
sns.histplot(df.max_stipend)
```



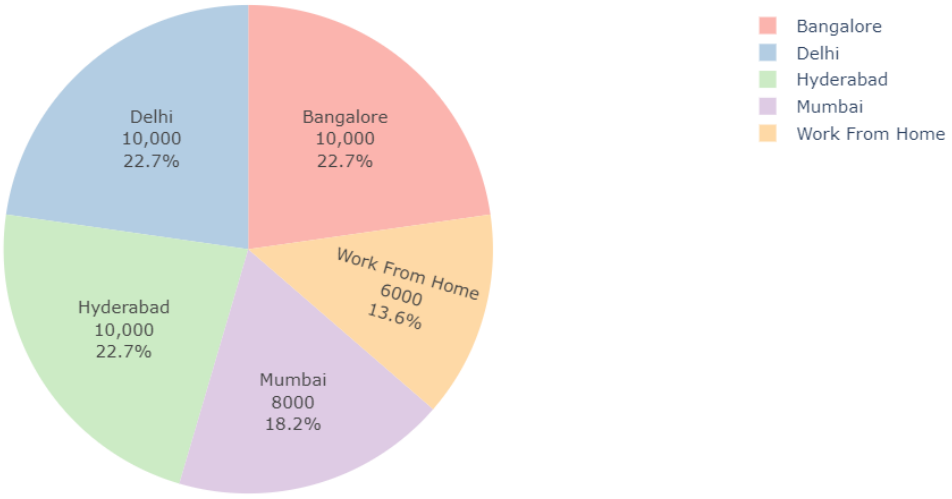
Location

```
# Location
top_paying_locations =
df.groupby('location').max_stipend.median().sort_values(ascending=False)
selected_locations = ['Work From Home', 'Mumbai', 'Bangalore', 'Delhi',
'Hyderabad']
median_stipend = [top_paying_locations[selected_locations[i]] for i in
range(len(selected_locations))]

plot_series_pie_chart(top_paying_locations, "Top Paying Location")
plot_series_pie_chart(pd.Series(median_stipend, index=selected_locations), "Median
Stipend in Popular locations")
```

Top Paying Location

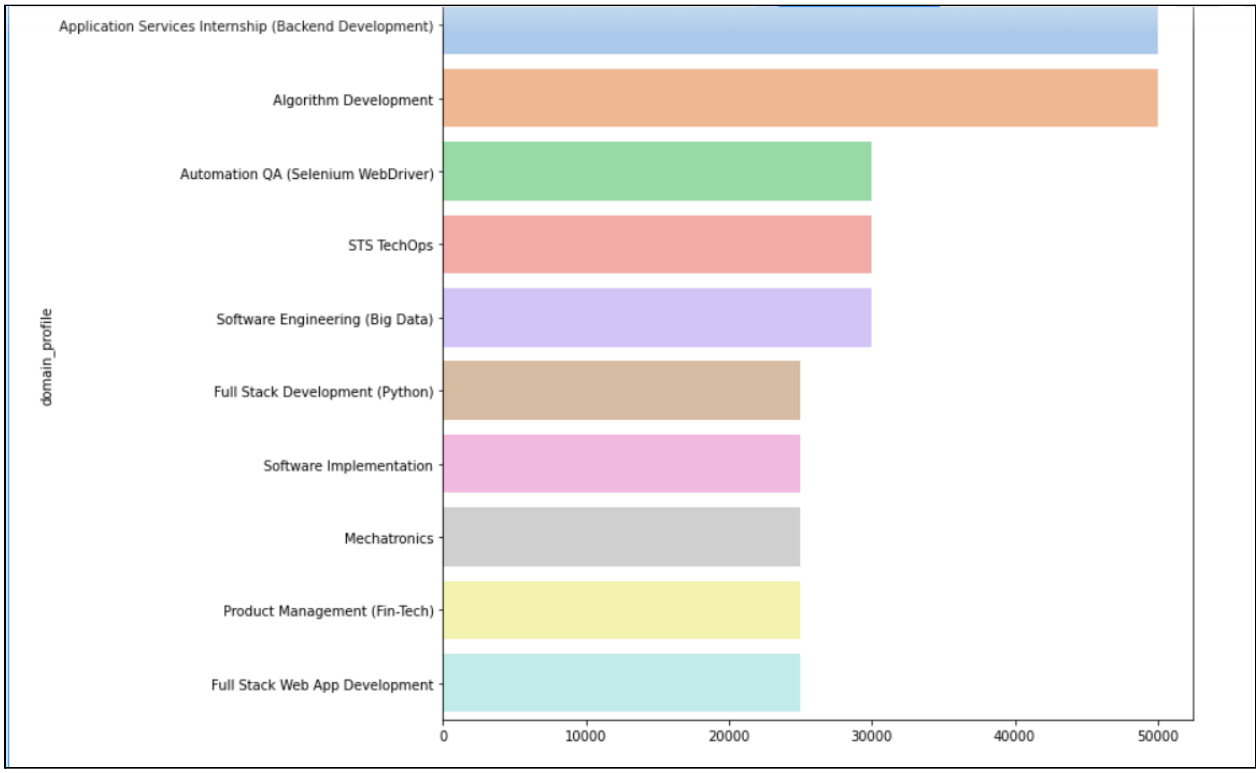


Median Stipend in Popular locations



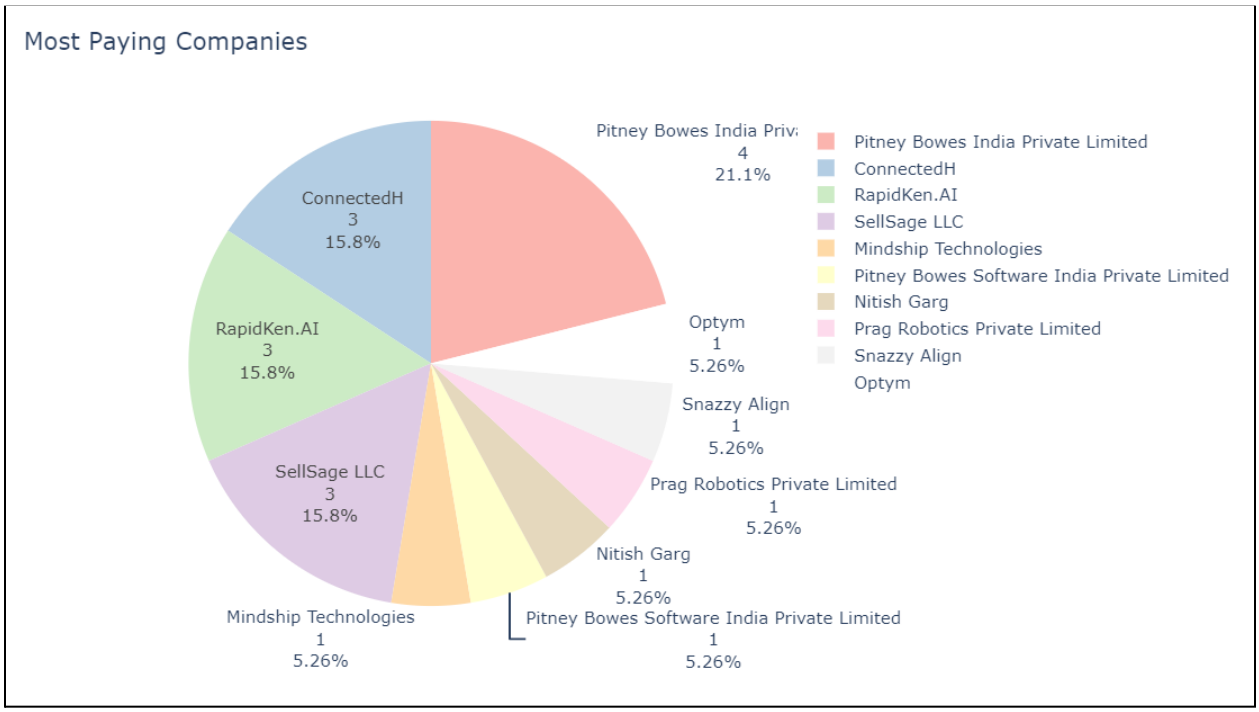
Domain

```
domain_stipend =
df.groupby('domain_profile').max_stipend.median().sort_values(ascending=False)
number_of_internships = df.domain_profile.value_counts().loc[domain_stipend.index]

plt.figure(figsize=(10,10))
ax = sns.barplot(y=domain_stipend[:10].index, x=domain_stipend[:10].values)
ax.set_title("Top Paying Domains")
```



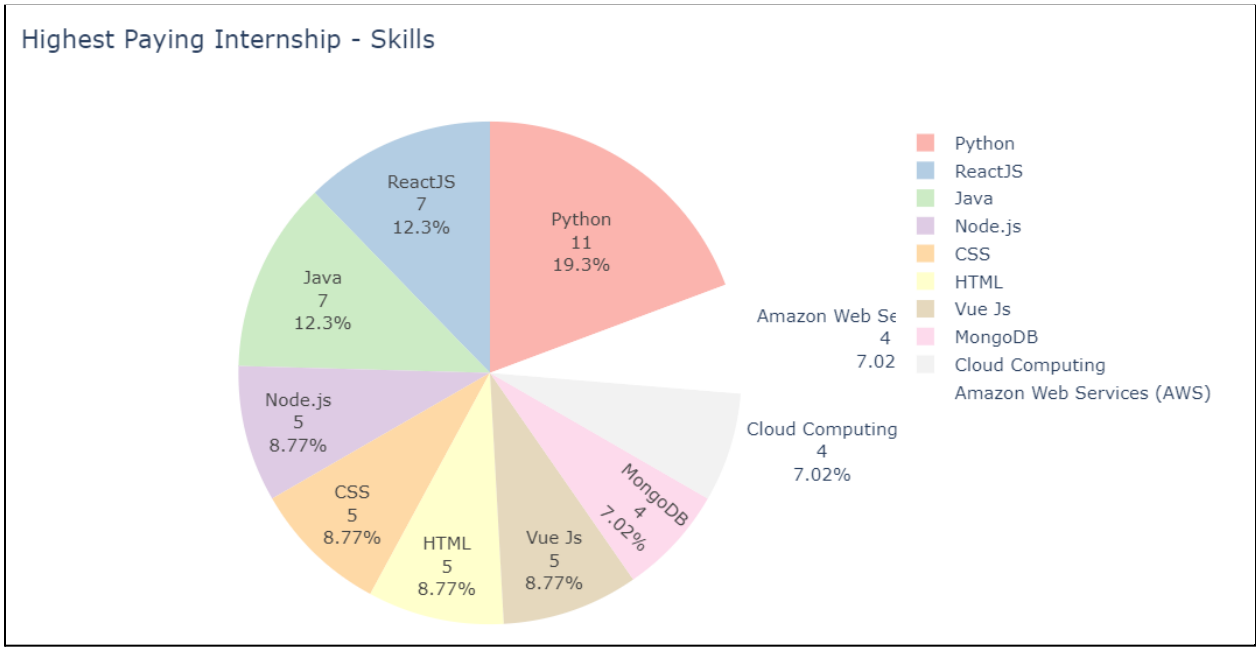
Company Name

```
company_names = stipend_sorted_df[:30].company_name.value_counts()
plot_series_pie_chart(company_names, "Most Paying Companies")
```

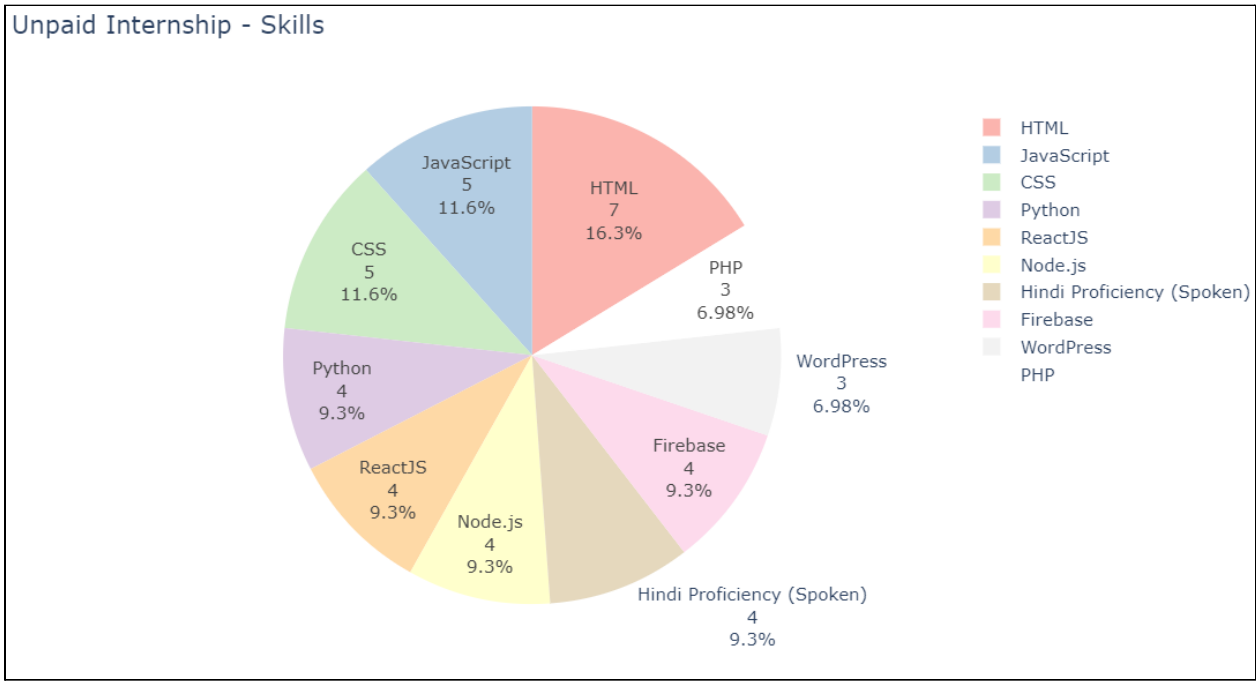


Skills

```
def get_series_from_dict(d):
    return pd.Series(data=list(d.values()), index=list(d.keys()))

high_paying_skillset =
get_series_from_dict(stipend_sorted_df[:30].skills.agg(agg_skillset))
unpaid_skillset =
get_series_from_dict(stipend_sorted_df[-30:].skills.agg(agg_skillset))

plot_series_pie_chart(high_paying_skillset, 'Highest Paying Internship - Skills')
plot_series_pie_chart(unpaid_skillset, 'Unpaid Internship - Skills')
```

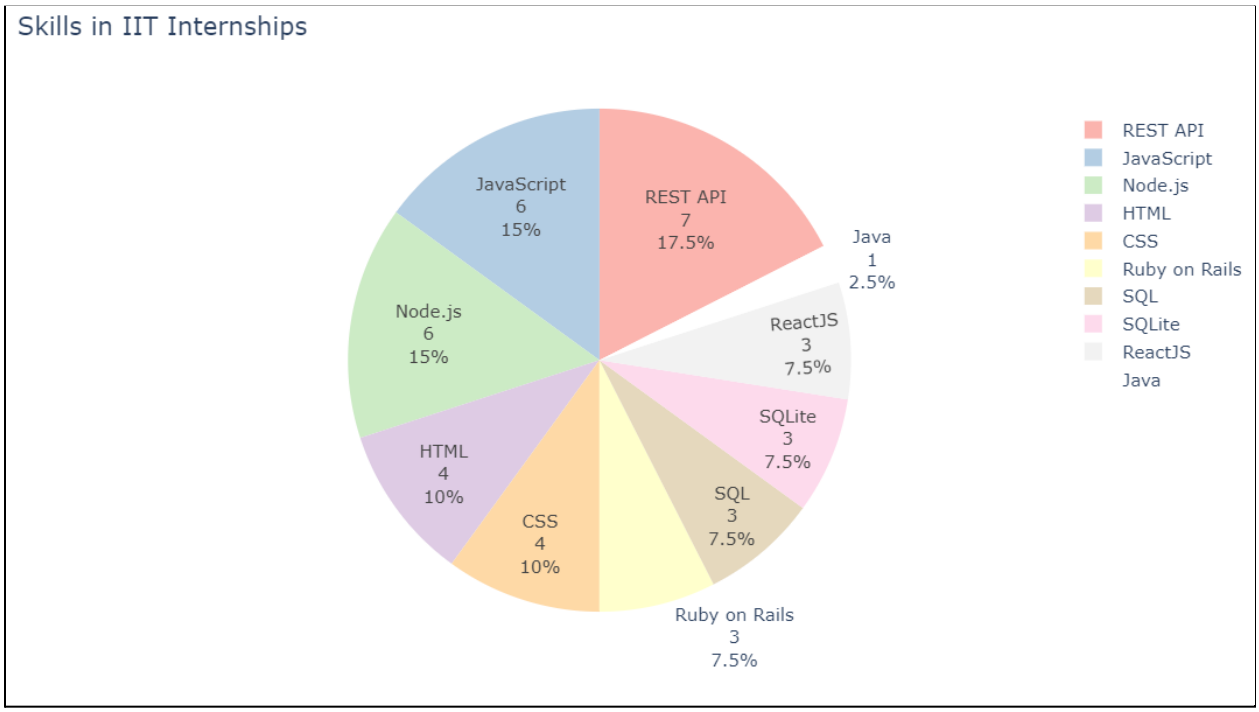


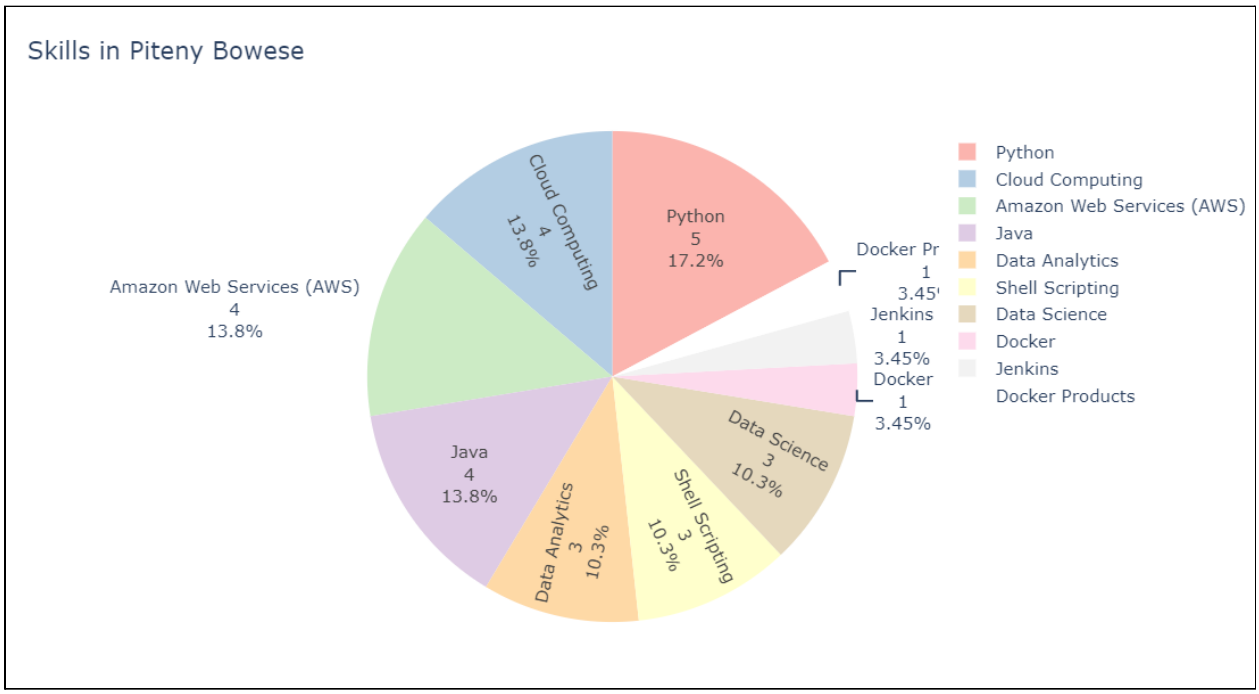
Unpaid Internship Skills



Analysing Skills From Popular Companies

```
def get_internships_by_company(name):
    return df[df.company_name.str.contains(name)]

iit_df = get_internships_by_company('IIT')
piteny_bowese_df = get_internships_by_company('Pitney Bowes')

iit_skill_set = agg_skillset(iit_df.skills)
piteny_bowese_skill_set = agg_skillset(piteny_bowese_df.skills)

plot_series_pie_chart(get_series_from_dict(iit_skill_set), "Skills in IIT
Internships")
plot_series_pie_chart(get_series_from_dict(piteny_bowese_skill_set), "Skills in
Piteny Bowese")
```

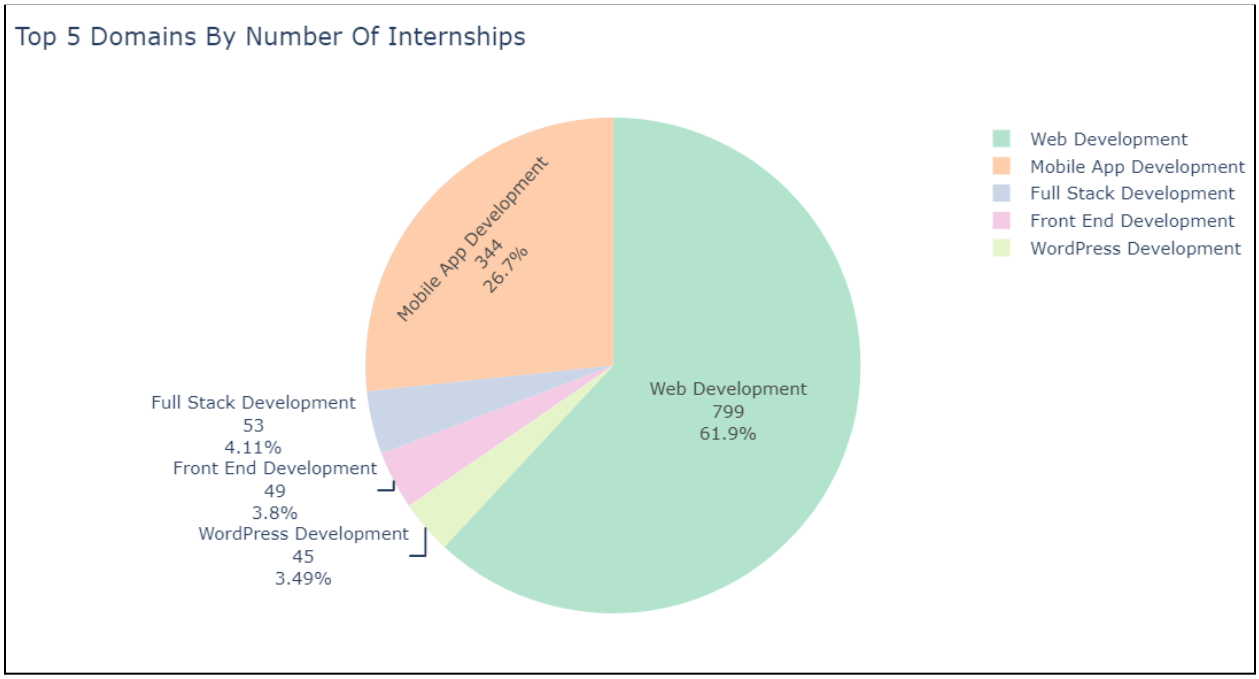



Skills in Piteny Bowese
Python
5
17.2%
Cloud Computing
4
13.8%
Amazon Web Services (AWS)
4
13.8%
Java
4
13.8%
Data Analytics
3
10.3%
Shell Scripting
3
10.3%
Data Science
3
10.3%
Docker
1
3.45%
Jenkins
1
3.45%
Docker Pr
1
3.45
Python
Cloud Computing
Amazon Web Services (AWS)
Java
Data Analytics
Shell Scripting
Data Science
Docker
Jenkins
Docker Products

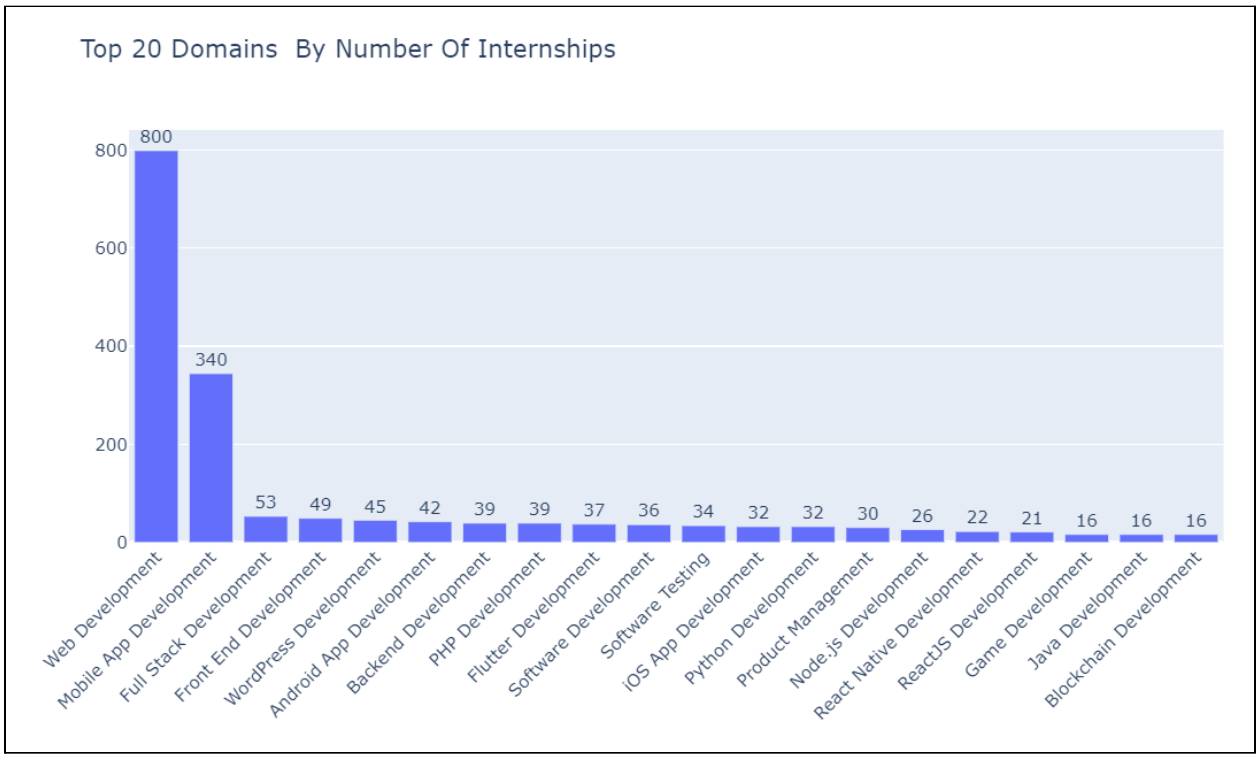
```
print(f'Median Stipend in IITs {iit_df.max_stipend.median()}')
print(f'Number of applicants for one openning =
{iit_df.applicants.sum()/iit_df.number_of_openings.sum()}')
```

Domain Profile

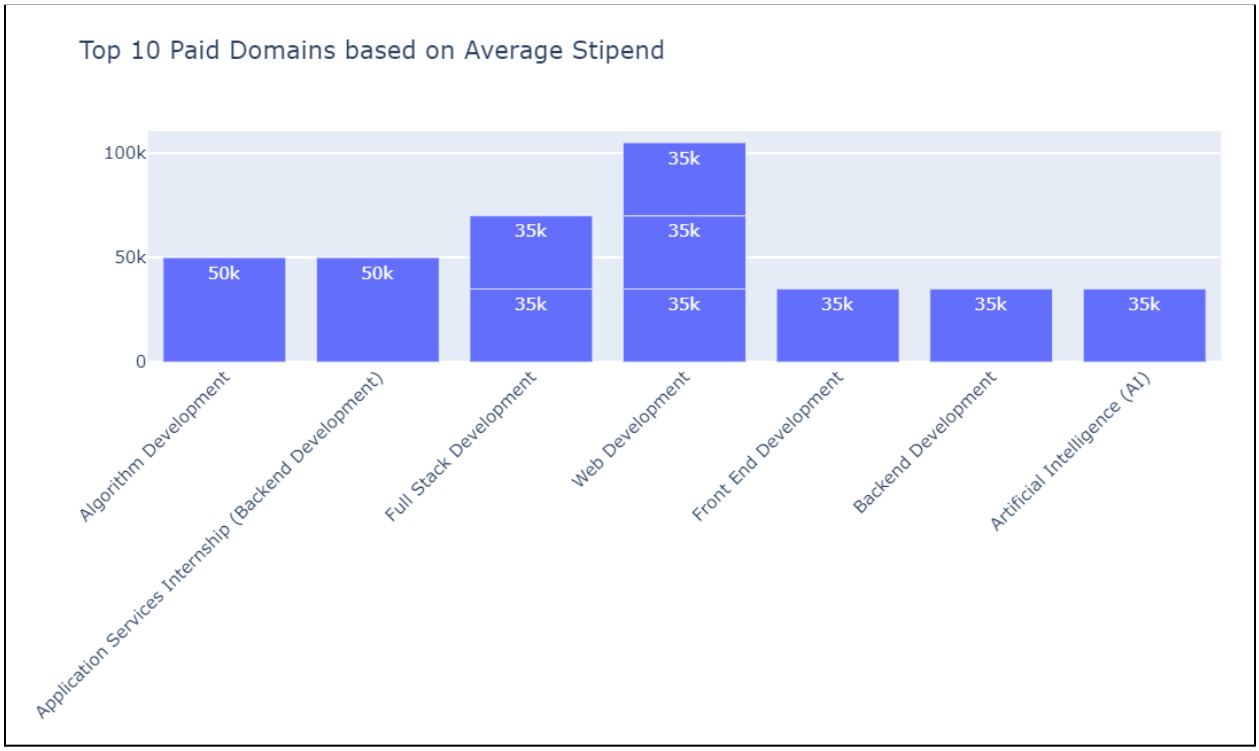
```
########### DOMAIN #############
# top 5 domains pie chart
label = df['domain_profile'].value_counts()[:5].index.tolist()
value = df['domain_profile'].value_counts()[:5].values.tolist()
'''trace = go.Pie(labels=label,
                values=value,
                marker=dict(colors=['red']),
                # Seting values to
                hoverinfo="value"
               )
data = [trace]
layout = go.Layout(title="Top 5 Domains")
fig = go.Figure(data = data,layout = layout)
'''

fig = px.pie(
    names = label,
    values = value,
    title="Top 5 Domains By Number Of Internships",
    color_discrete_sequence=px.colors.qualitative.Pastel2
    )
fig['data'][0].update({'textinfo' : 'label+text+value+percent'})
fig.show(renderer="colab")
```



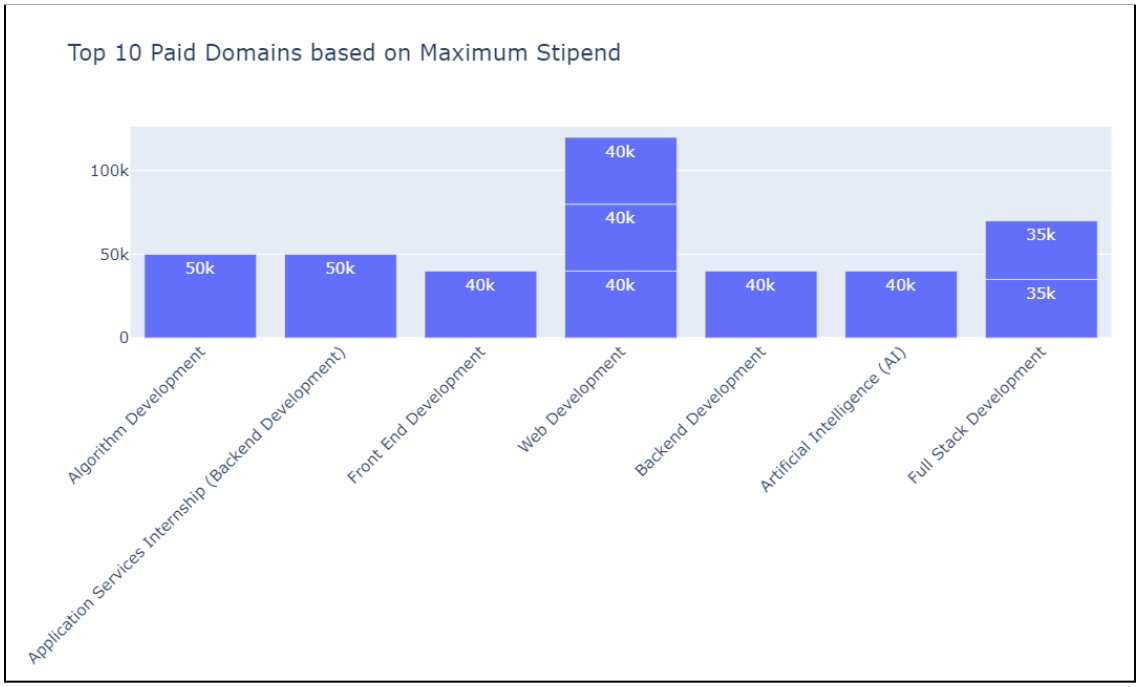

```
# top 20 domains bar graph
label = df['domain_profile'].value_counts()[:20].index.tolist()
value = df['domain_profile'].value_counts()[:20].values.tolist()

fig = go.Figure(
    data=[go.Bar(y = value, x = label, text= value)],
    layout_title_text="Top 20 Domains  By Number Of Internships",
)
fig.update_traces(texttemplate='%{text:.2s}', textposition='outside')
fig.update_layout(xaxis_tickangle=-45)
fig.show(renderer="colab")
```




```
# top 10 paid avg by domain
#df[['company_name','domain_profile','mean_stipend']].sort_values(by=['mean_stipend'], ascending=False)[:10]
davg = df[['domain_profile','mean_stipend']].sort_values(by=['mean_stipend'], ascending=False)[:10].reset_index()
fig = go.Figure(
    data=[go.Bar(y = davg.mean_stipend, x = davg.domain_profile, text=davg.mean_stipend)],
    layout_title_text="Top 10 Paid Domains based on Average Stipend",
)
fig.update_traces(texttemplate='%{text:.2s}', textposition='inside')
fig.update_layout(xaxis_tickangle=-45)
fig.show(renderer="colab")
```




```
#top 10 paid max by domain
dmax = df[['domain_profile','max_stipend']].sort_values(by=['max_stipend'],
ascending=False)[:10].reset_index()
fig = go.Figure(
    data=[go.Bar(y = dmax.max_stipend, x = dmax.domain_profile, text=
dmax.max_stipend)],
    layout_title_text="Top 10 Paid Domains based on Maximum Stipend",
)
fig.update_traces(texttemplate='%{text:.2s}', textposition='inside')
fig.update_layout(xaxis_tickangle=-45)
fig.show(renderer="colab")
```

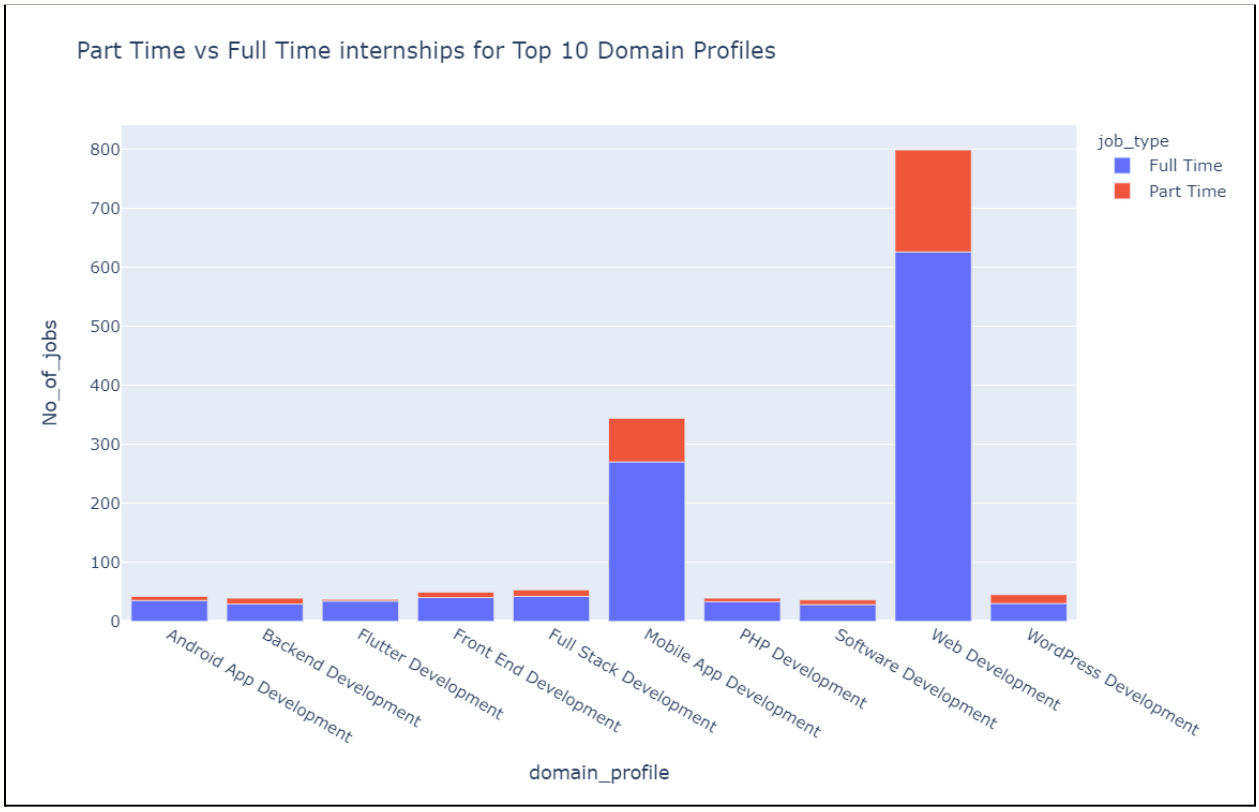

Top 10 Paid Domains based on Maximum Stipend
100k
50k
0
50k
50k
40k
40k
40k
40k
40k
40k
35k
35k
Algorithm Development
Application Services Internship (Backend Development)
Front End Development
Web Development
Backend Development
Artificial Intelligence (AI)
Full Stack Development

```
# top 10 domains vs duration
df[['domain_profile','duration']].value_counts()[:5]
```

```
domain_profile          duration
Web Development         3 Months    257
                        6 Months    255
                        2 Months    169
Mobile App Development  3 Months    127
                        2 Months     91
dtype: int64
```

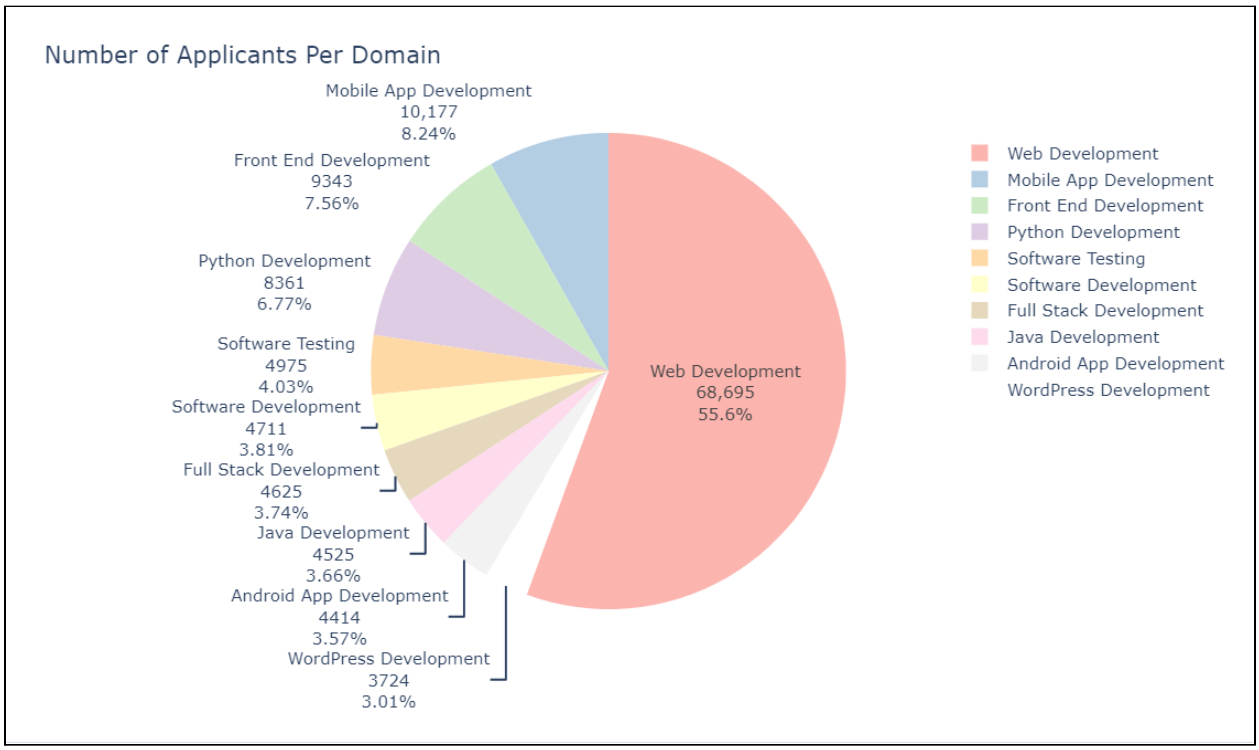
Domain Vs Part time

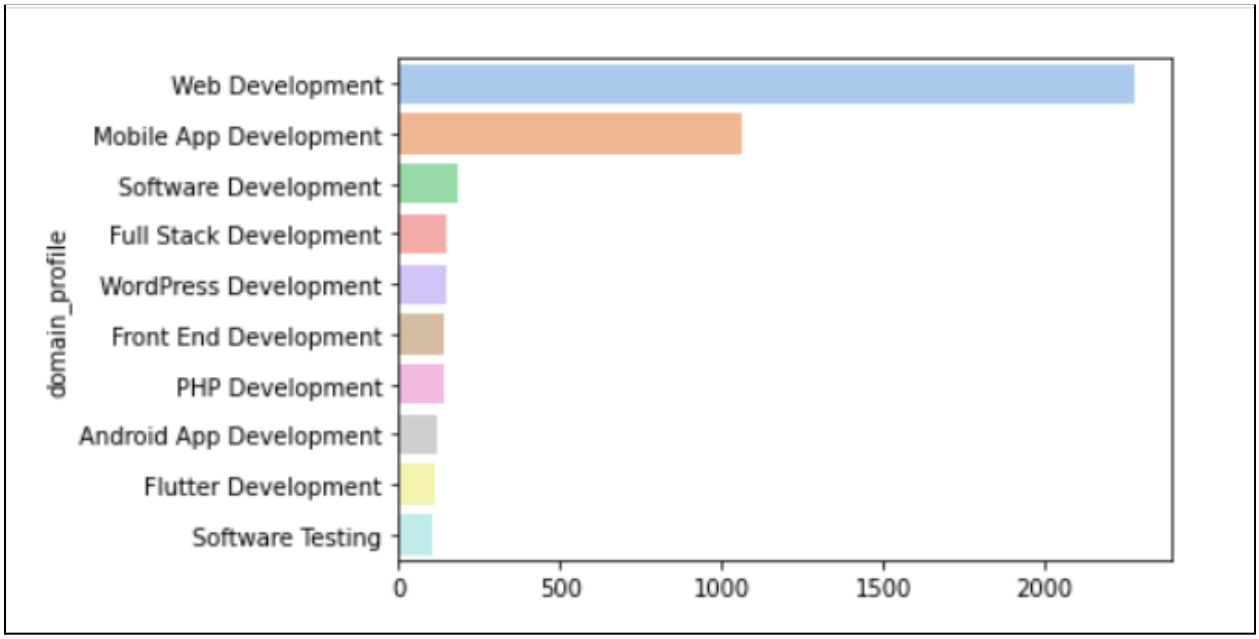
```
dom = df['domain_profile'].value_counts()[:10]

pt =
df[df['domain_profile'].isin(dom.index)].groupby('domain_profile')['is_part_time'].
value_counts()
pt2 = pt.reset_index(level=['domain_profile'])
pt2['job_type'] = pt2.index
d = {0: 'Full Time', 1: 'Part Time'}
pt2 = pt2.replace(d)
pt2.rename(columns={'is_part_time': 'No_of_jobs'}, inplace=True)

fig = px.bar(pt2, x=pt2.domain_profile, y=pt2.No_of_jobs, color = 'job_type',
height=600, title="Part Time vs Full Time internships for Top 10 Domain Profiles")
fig.show(renderer="colab")
```




```
"""
#Anish Tipnis, delete this code
Applicants and Number of openings

Applicants vs domain_profile (what's the distribution for application count wtr to
domain?)
ratio of applicants vs openning in domain profile
top 5 domains wrt to number of applicants
top 5 domains wrt to number of opennings
top 5 domains wrt to ratio of applicants and opennings
Demand and supply analysis, what is the ratio of appliants vs number of opennings?
Number of part time opennings and their ratio.



#top 5 domains wrt to number_of_openings
all_domain =
df.number_of_openings.value_counts()/df.number_of_openings.value_counts().values.su
m()
all_domain_series = all_domain.sort_values(ascending=False).to_dict()
display("Top 5 openings")
```

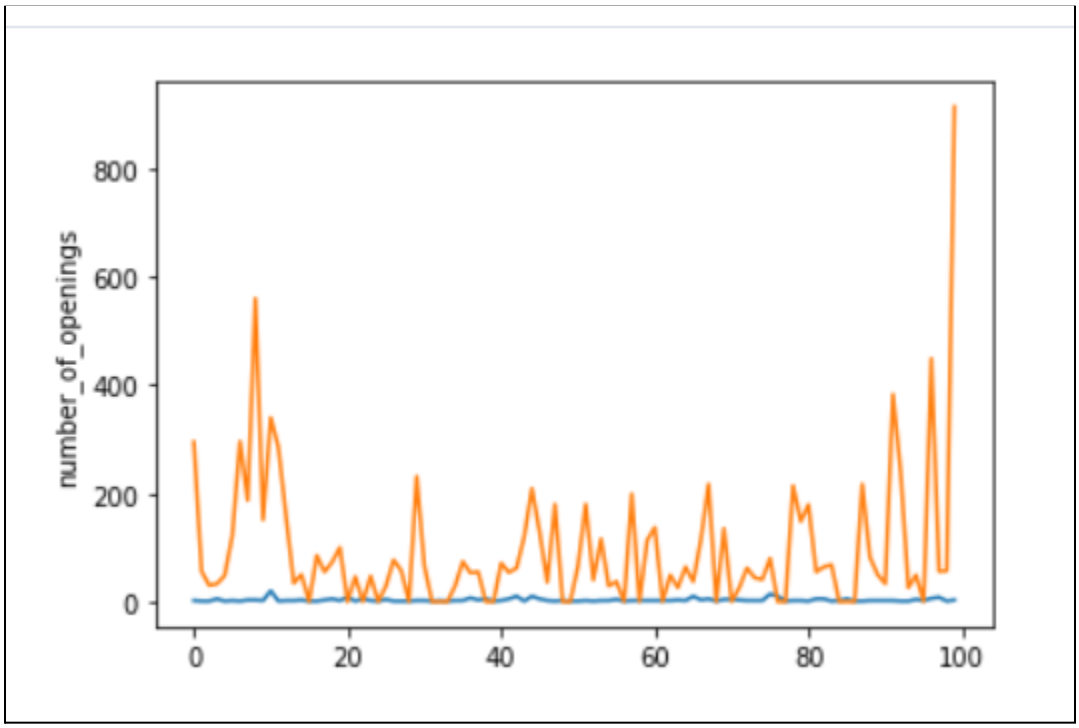
```
display(df.number_of_openings.value_counts()[:5])
"""
#####################################################
def number_of_applicants_per_domain(series):
    return series.sum()

group_by_domain_profile = df.groupby(by='domain_profile')
domain_by_applicants =
group_by_domain_profile.applicants.agg(number_of_applicants_per_domain).sort_values
(ascending=False)

plot_series_pie_chart(domain_by_applicants, "Number of Applicants Per Domain")
#####################################################

#top 5 domains wrt opening
def no_of_app_opening(series):
    return series.sum()

domain_profile_openings = df.groupby(by='domain_profile')
profile_applicants =
domain_profile_openings.number_of_openings.agg(number_of_applicants_per_domain).sor
t_values(ascending=False)
#print(profile_applicants)
#piechart
#plot_series_pie_chart(profile_applicants,"Openings wrt Domains")
x=sns.barplot(y=profile_applicants.index[:10], x=profile_applicants.values[:10])
print(x)
```






```
#no of applicants to no of openings
applicants_list = df['applicants'].tolist()[:5]
no_of_app_openings_list = df['number_of_openings'].tolist()[:5]
#print(applicants_list, no_of_app_openings_list)
```

```
#sns.barplot(x=applicants_list, y=no_of_app_openings_list)
#samplecode
#sns.barplot(x="day", y="total_bill", hue="sex", data=tips)

#ax = sns.barplot(x=applicants_list, y=no_of_app_openings_list, hue="applied vs
open")

ratio_applicant_to_num_opennings =  df.applicants/df.number_of_openings
# ratio_applicant_to_num_opennings.mean()

ax = sns.lineplot(x=df.index[:100], y=df.number_of_openings[:100])
sns.lineplot(x=df.index[:100], y=df.applicants[:100], ax=ax)

ratio_applicant_to_num_opennings
```


number_of_openings
800
600
400
200
0
0
20
40
60
80
100

```
dfnew = pd.concat([df, ratio_applicant_to_num_opennings], axis=0)
dfnew.columns
```

```
Index([                      0,                  '.NET',              '3ds Max',
                          'AJAX',                'ANSYS',  'ARM Microcontroller',
                       'ASP.NET',  'Adobe After Effects', 'Adobe Creative Suite',
            'Adobe Dreamweaver',
       ...
                  'max_stipend',         'mean_stipend',          'min_stipend',
           'number_of_openings',   'other_requirements',                'perks',
                        'skills',          'start_date',              'stipend',
                'who_can_apply'],
      dtype='object', length=260)
```

Monster India Dataset

EDA

```
import pandas as pd
import re
from datetime import date
import numpy as np
import math
import pygal
import seaborn as sns
#from plotly.offline import init_notebook_mode,iplot
import plotly.graph_objects as go
import matplotlib.pyplot as plt
import plotly.express as px
from IPython.display import SVG, display
from plotly.subplots import make_subplots

import plotly.offline as pyo
import plotly.graph_objs as go

from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori, fpmax, fpgrowth
# Set notebook mode to work in offline
pyo.init_notebook_mode()
```

```
# Global Functions
sns.set_palette("pastel")
def plot_series_pie_chart(series, title, max=10,
color_seq=px.colors.qualitative.Pastel1):
    series.sort_values(ascending=False, inplace=True)
    series = series[:max]
    fig = px.pie(
        names=series.index,
        values=series.values,
        title=title,
        color_discrete_sequence=color_seq
        )

    fig['data'][0].update({'textinfo' : 'label+text+value+percent'})
    fig.show(renderer="colab")

def plot_bar_chart(series, title, text_position='outside', xaxis_label=None,
yaxis_label=None):
```

```
    fig = go.Figure(
        data=[go.Bar(x = series.index, y = series.values, text= series.values)],
        layout_title_text=title,

    )
    fig.update_traces(texttemplate='%{text:.2s}', textposition=text_position)
    fig.update_layout(xaxis_tickangle=-45)
    if xaxis_label != None:
        fig.update_layout(xaxis_title=xaxis_label,)
    if yaxis_label != None:
        fig.update_layout(yaxis_title=yaxis_label,)

    fig.show(renderer="colab")
```

```
df = pd.read_csv('monster-india/monster-india-clean.csv')
df = df.drop('Unnamed: 0', axis=1)
```

```
df.head(3)
```

Preview Visualize

	timestamp int64	location object	job_title object	company_name object	package object	experience object	job_description object
0	1620930600	Bengaluru / Bangalore	Veeva Vault	2COMS Consulting Private Limited	60,000-3,00,000 INR Per Annum	4-9 Years	Urgent opening for experienced professionals in a …
Expand rows 1 - 1							
2	1620930600	Bengaluru / Bangalore	IT Recruiter C2H	2COMS Consulting Private Limited	50,000-3,00,000 INR Per Annum	0-5 Years	We are hiring for Immediate Requirement for Position : IT …

3 rows × 468 columns

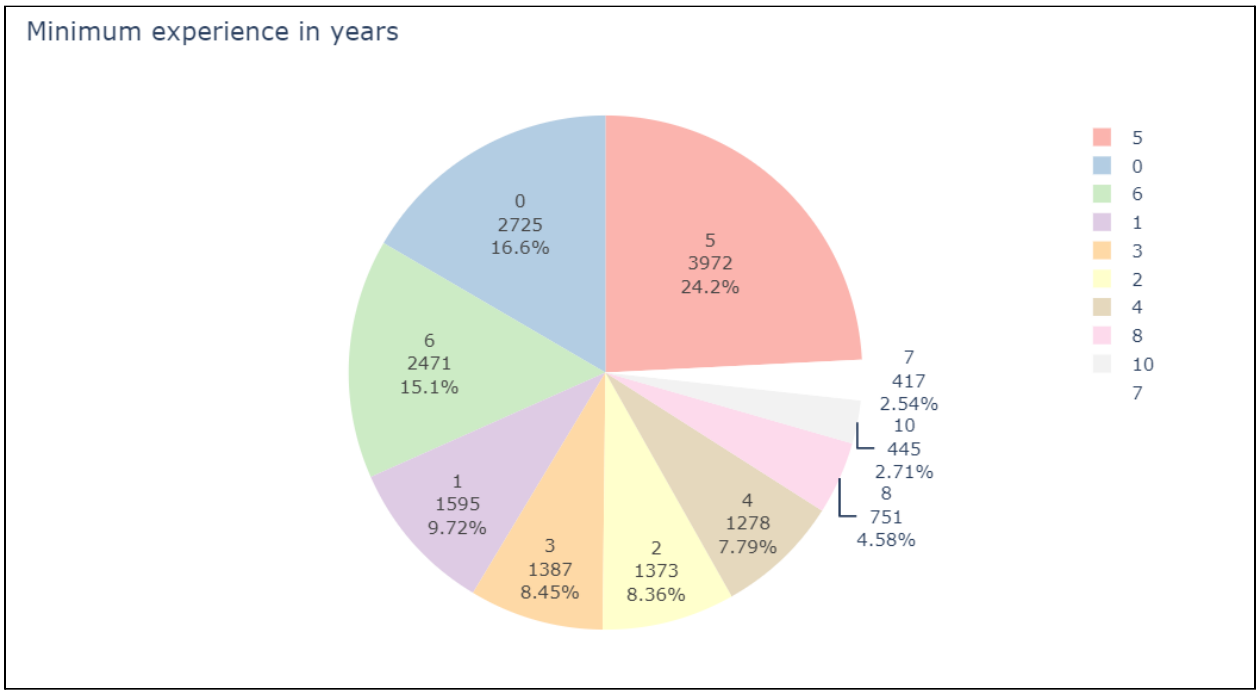
```
df.describe()
```

	timestamp float64	min_salary float64	mean_salary float64	min_experience float64	max_experience float64	mean_experience
count	16800	16800	16800	16800	16800	16800
mean	1620304328.5714285	634840.5982738095	919696.9447023809	3.9467857142857143	8.158988095238096	6.05288690476190
std	777142.8277741116	566506.632266377	728248.4018587257	2.8566191587174776	3.922217899283065	3.28247995913623
min	1619807400	0	0	0	0	0
25%	1619807400	210000	350000	1	5	3.5
50%	1619807400	600000	850000	5	8	6.5
75%	1620757800	900000	1300000	6	10	7.5
max	1622485800	9000000	12800000	25	31	28

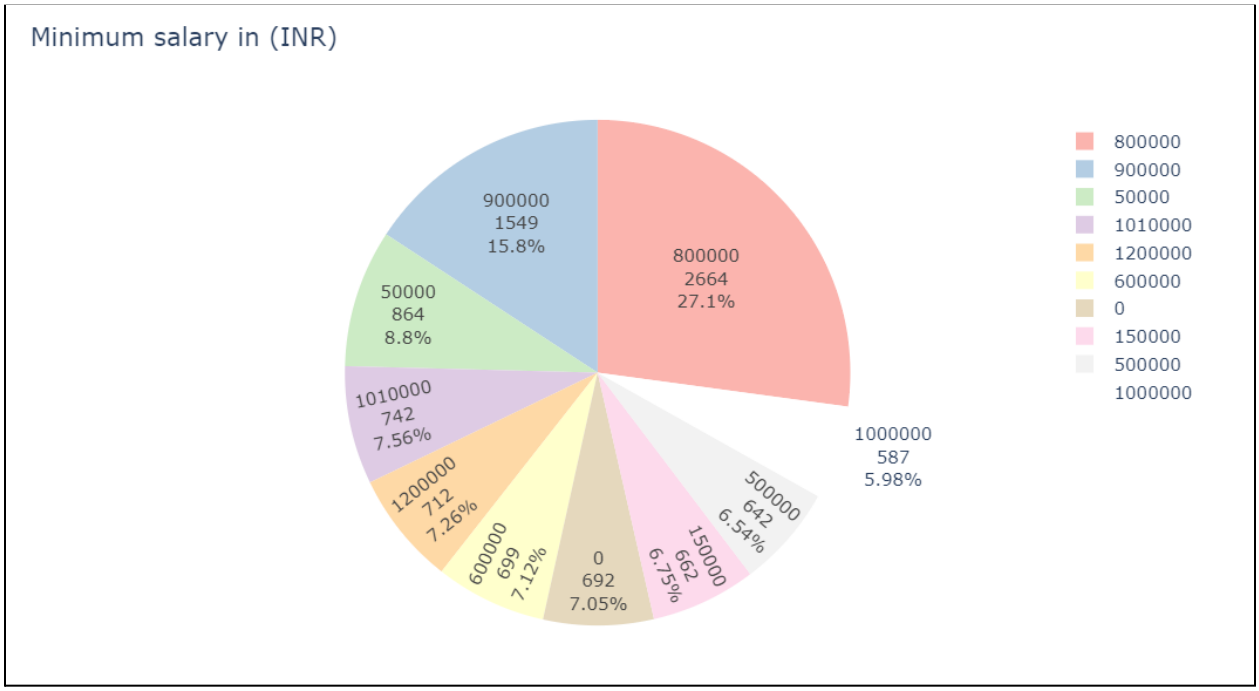
8 rows × 6 columns

Relationship between salary and experience

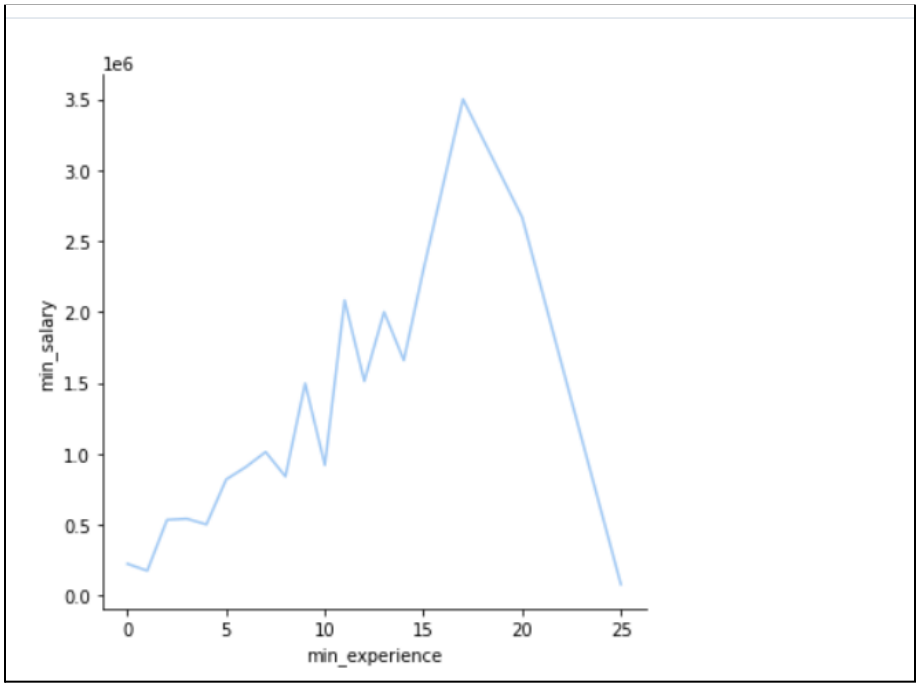
```
plot_series_pie_chart(df.min_experience.value_counts(), "Minimum experience in years")
```




```
plot_series_pie_chart(df.min_salary.value_counts(), "Minimum salary in (INR)")
```



```
sns.relplot(x='min_experience',y='min_salary',kind = 'line', data=df, ci = None)
```

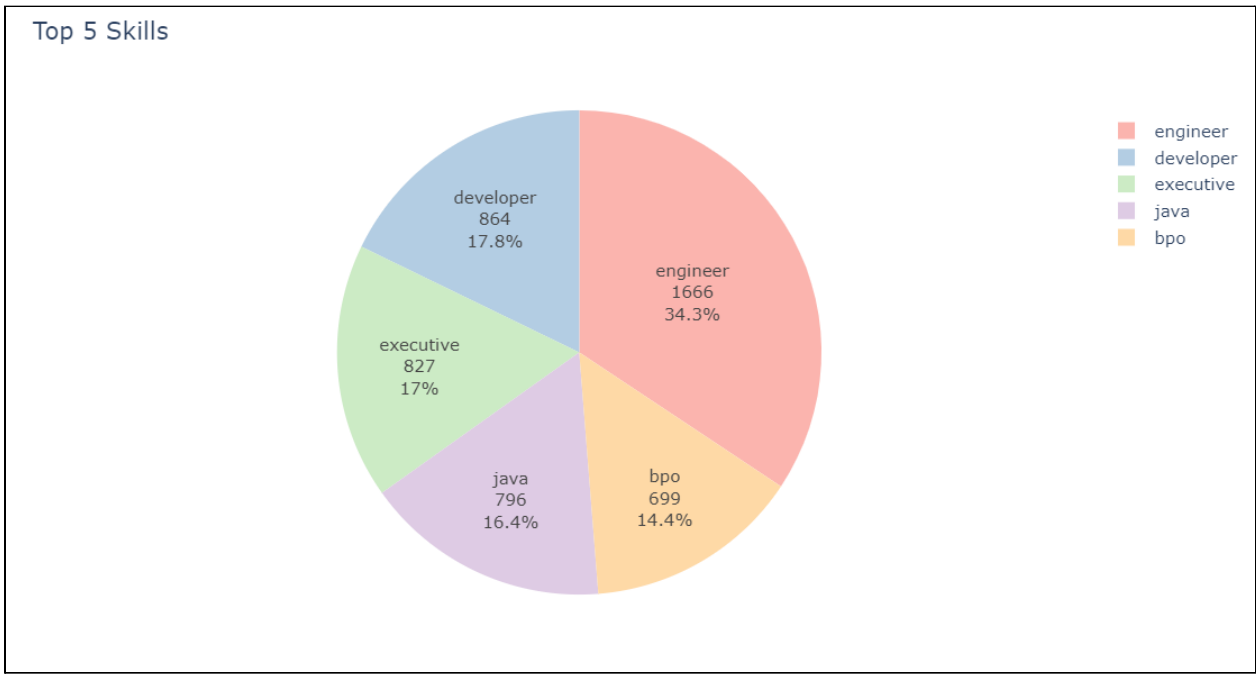


Skills Analysis

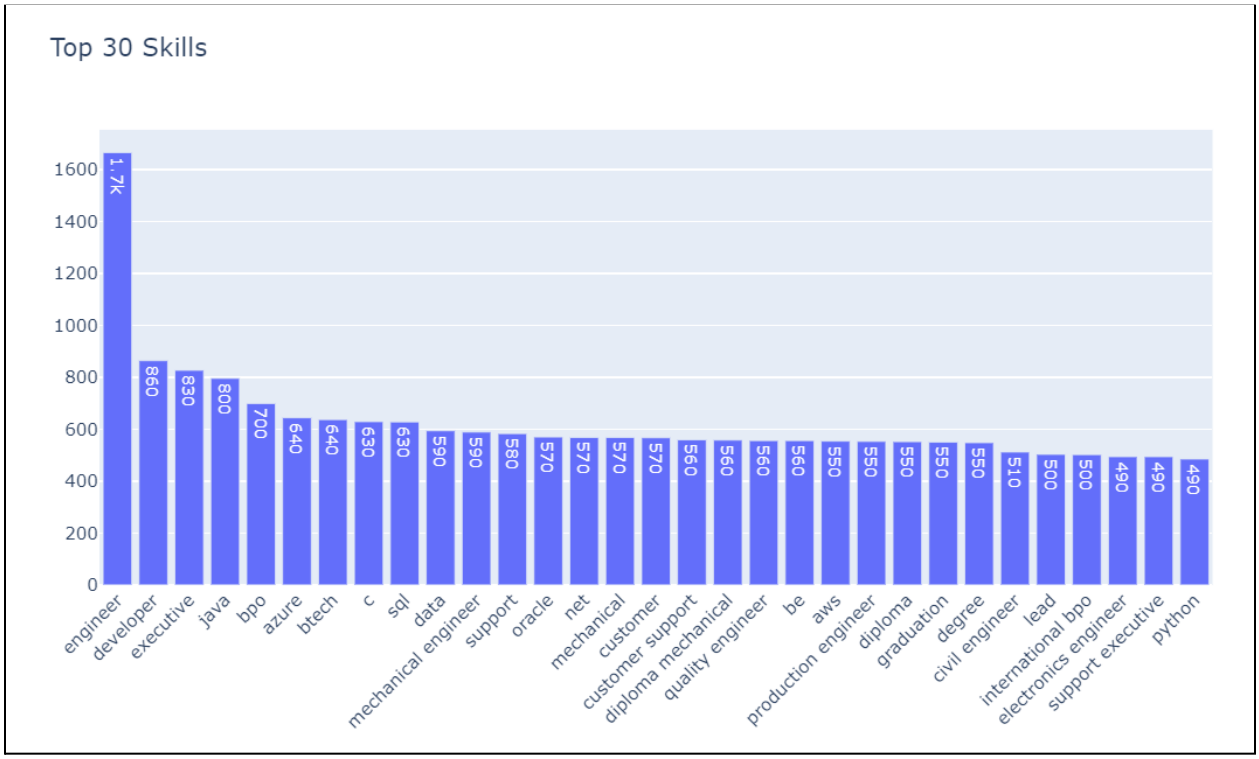
```
skills_super_set = dict()
def agg_skillset(input_string):
    skillset = {}
    for skill in input_string:
        if(type(skill) == str):
            for i in skill.split(','):
                if(i!=''):
                    skillset[i] = skillset.get(i, 0) + 1
    return skillset

skills_super_set = df.skills.agg(agg_skillset)
skills_count_series = pd.Series(skills_super_set).sort_values(ascending=False)
skills_count_series
```

```
engineer                1666
developer                864
executive                827
java                     796
bpo                      699
                        ...
lead ba                    1
dblead assocaite           1
dblead                     1
assocaite architect        1
sec principal              1
Length: 15218, dtype: int64
```

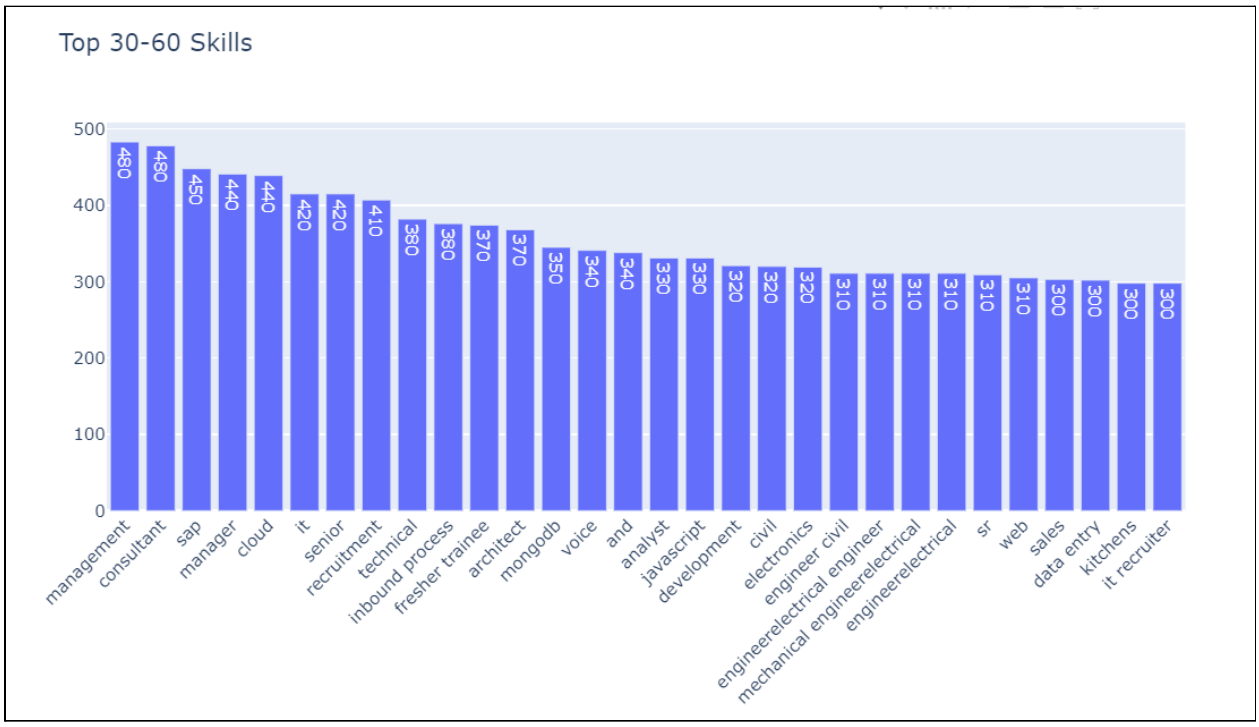
```
plot_series_pie_chart(skills_count_series[:5], "Top 5 Skills")
```


Top 5 Skills
developer
864
17.8%
engineer
1666
34.3%
executive
827
17%
java
796
16.4%
bpo
699
14.4%
engineer
developer
executive
java
bpo

```
plot_bar_chart(skills_count_series[:31], "Top 30 Skills",'inside')
```



```
plot_bar_chart(skills_count_series[31:61], "Top 30-60 Skills", 'inside')
```

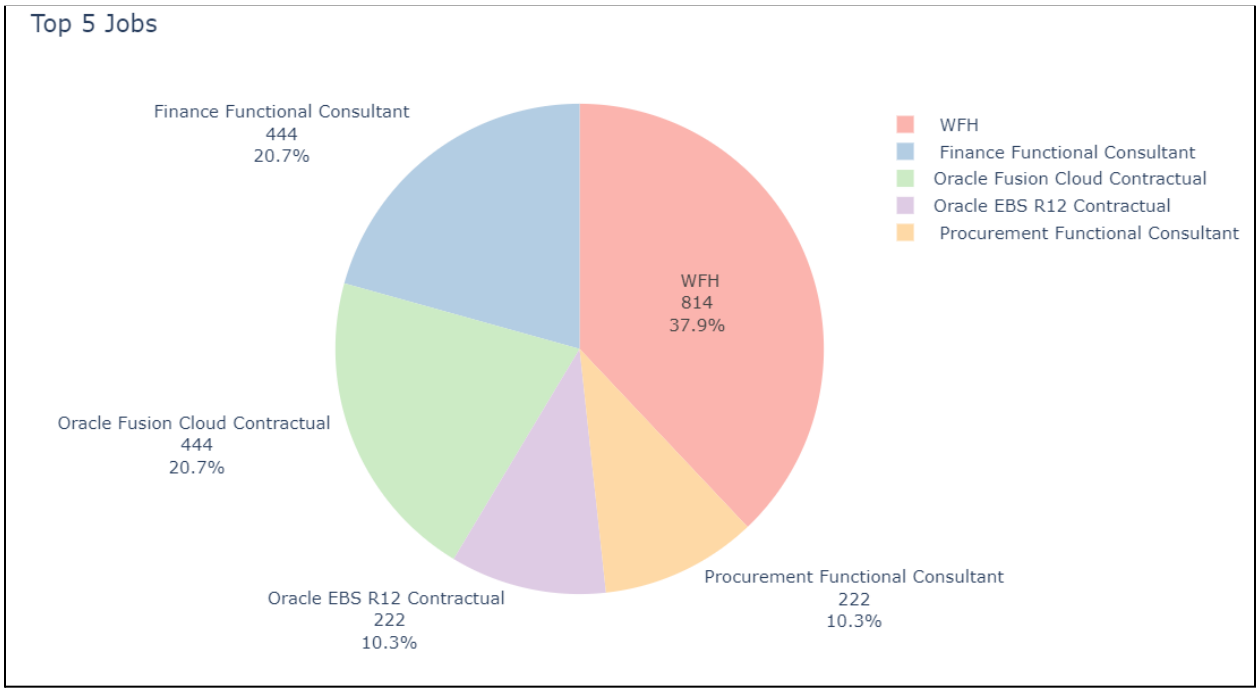


Jobs analysis

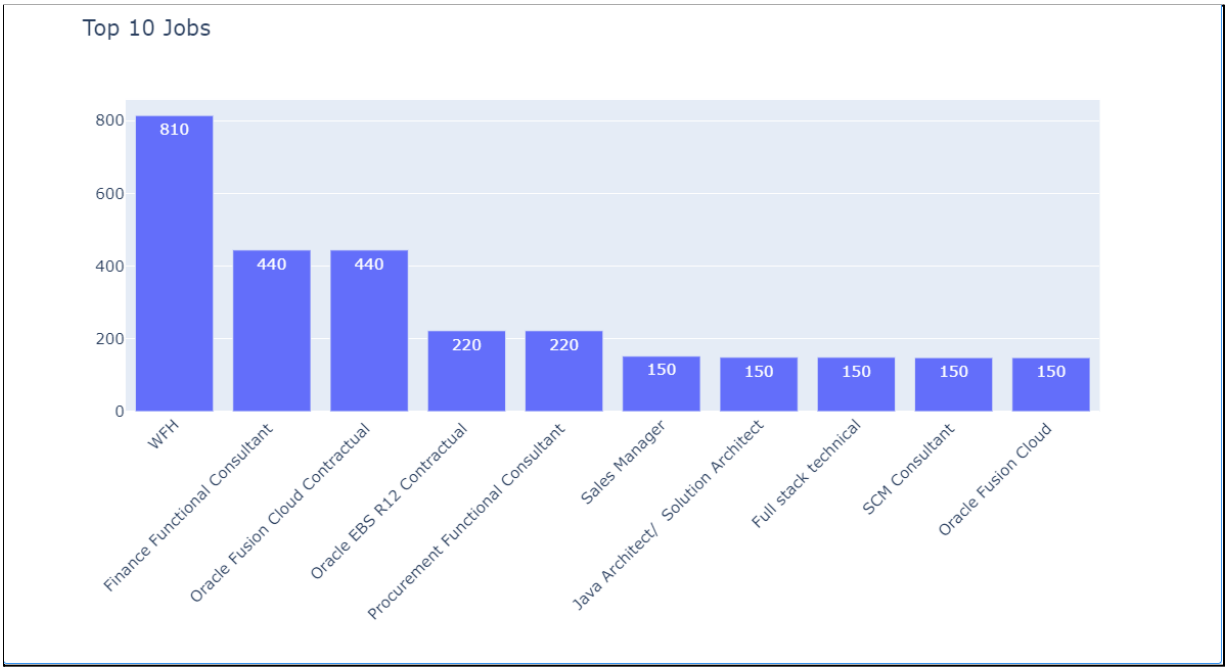
```
jobs_super_set = dict()
def agg_jobset(input_string):
    for job in input_string.split('||'):
        if(type(job) == str):
            jobs_super_set[job] = jobs_super_set.get(job,0) + 1
    return jobs_super_set

jobs_title = df.job_title.agg(agg_jobset)[0]

job_title_count_series = pd.Series(jobs_title).sort_values(ascending=False)
job_title_count_series[:15]
```

```
 WFH                                                              814
 Finance Functional Consultant                                    444
Oracle Fusion Cloud Contractual                                   444
Oracle EBS R12 Contractual                                        222
 Procurement Functional Consultant                                222
Sales Manager                                                     152
Java Architect/  Solution Architect                               149
Full stack technical                                              149
 SCM Consultant                                                   148
Oracle Fusion Cloud                                               148
IT Recruiter                                                      105
Full Stack Developer                                               86
Hiring For-MNC-Multiple positions-For PUNE/Mumbai/Delhi/Bangalore  84
.Net Developer                                                     82
Business Analyst                                                   80
dtype: int64
```

```
plot_series_pie_chart(job_title_count_series[:5], "Top 5 Jobs ")
```

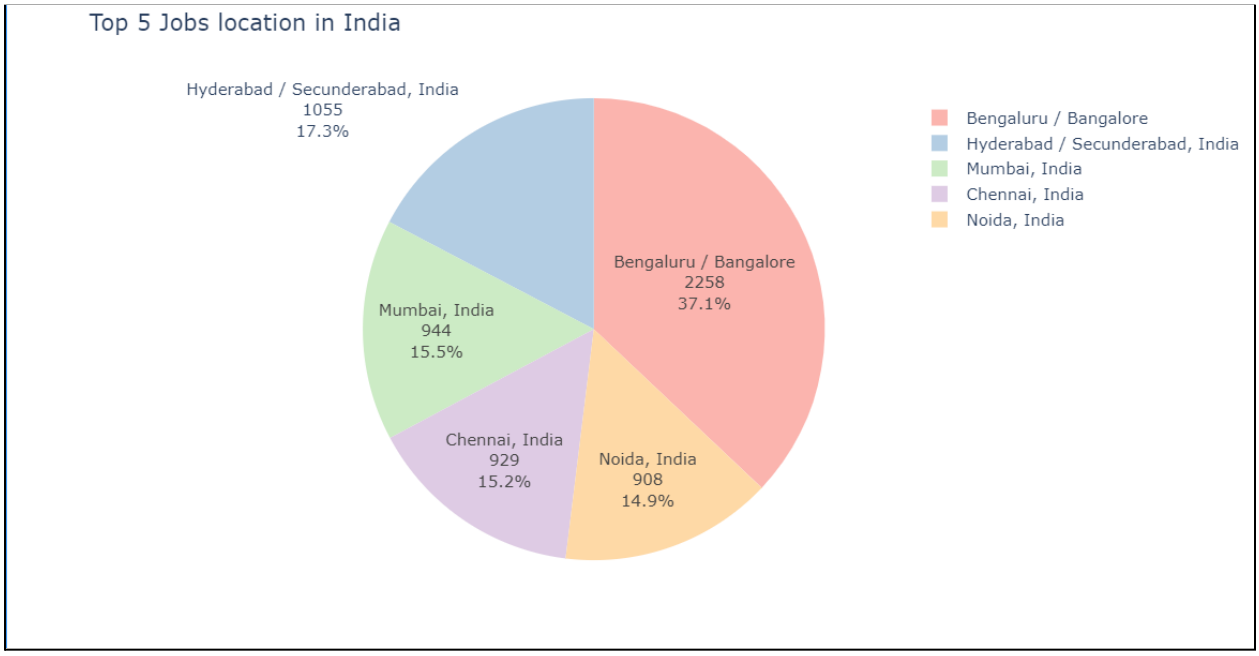



```
plot_bar_chart(job_title_count_series[0:10], "Top 10 Jobs", 'inside')
```

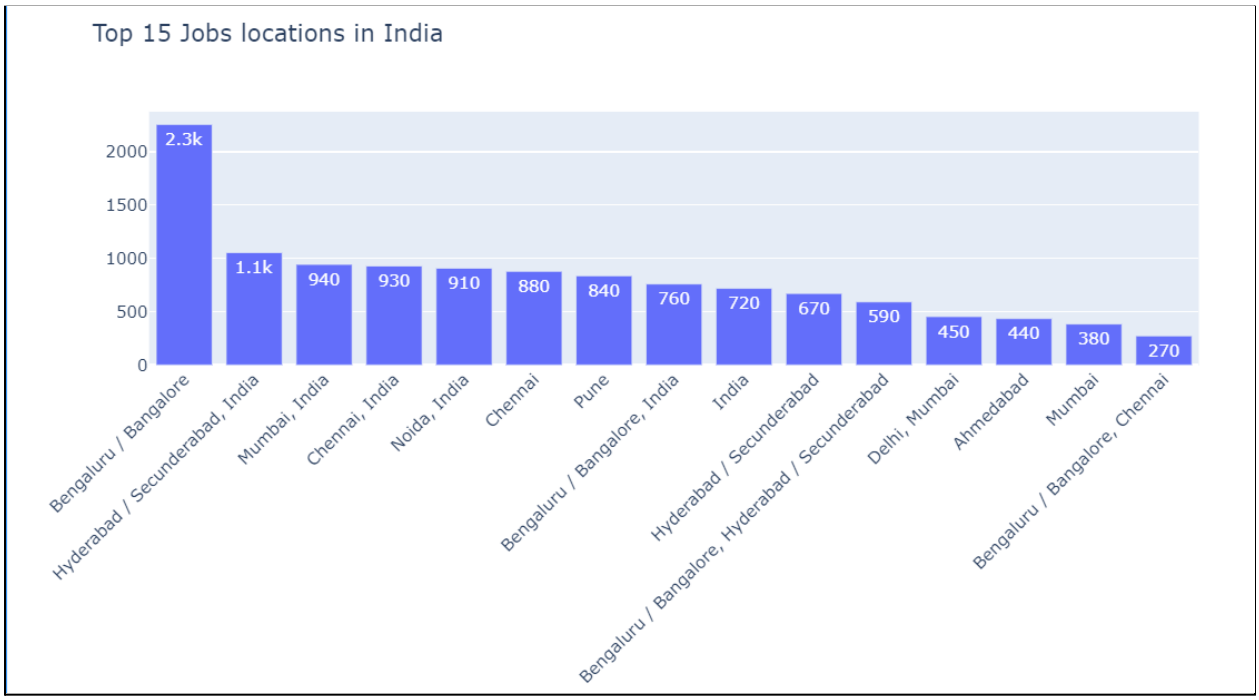


Location based analysis

```
plot_series_pie_chart(df.location.value_counts()[:5], "Top 5 Jobs location in India")
```



```
plot_bar_chart(df.location.value_counts()[0:15], "Top 15 Jobs locations in India",
'inside')
```



```
df.groupby(by='location').package.value_counts().sort_values(ascending=False)
```

```
location                                  package
Ahmedabad                                 1,50,000-3,00,000 INR Per Annum      385
Bengaluru / Bangalore                     10,00,000-20,00,000 INR Per Annum    312
Bengaluru / Bangalore, Hyderabad / Secunderabad  10,10,000-25,50,000 INR Per Annum    296
Delhi, Mumbai                             10,10,000-25,50,000 INR Per Annum    296
Mumbai, India                             8,00,000-14,00,000 INR Per Annum     234
                                                                               ...
Chennai, Mumbai                           6,00,000-12,50,000 INR Per Annum       1
                                          6,20,000-14,20,000 INR Per Annum       1
                                          6,50,000-10,60,000 INR Per Annum       1
                                          7,50,000-10,50,000 INR Per Annum       1
Gurgaon / Gurugram                        5,00,000-15,00,000 INR Per Annum       1
Name: package, Length: 3027, dtype: int64
```

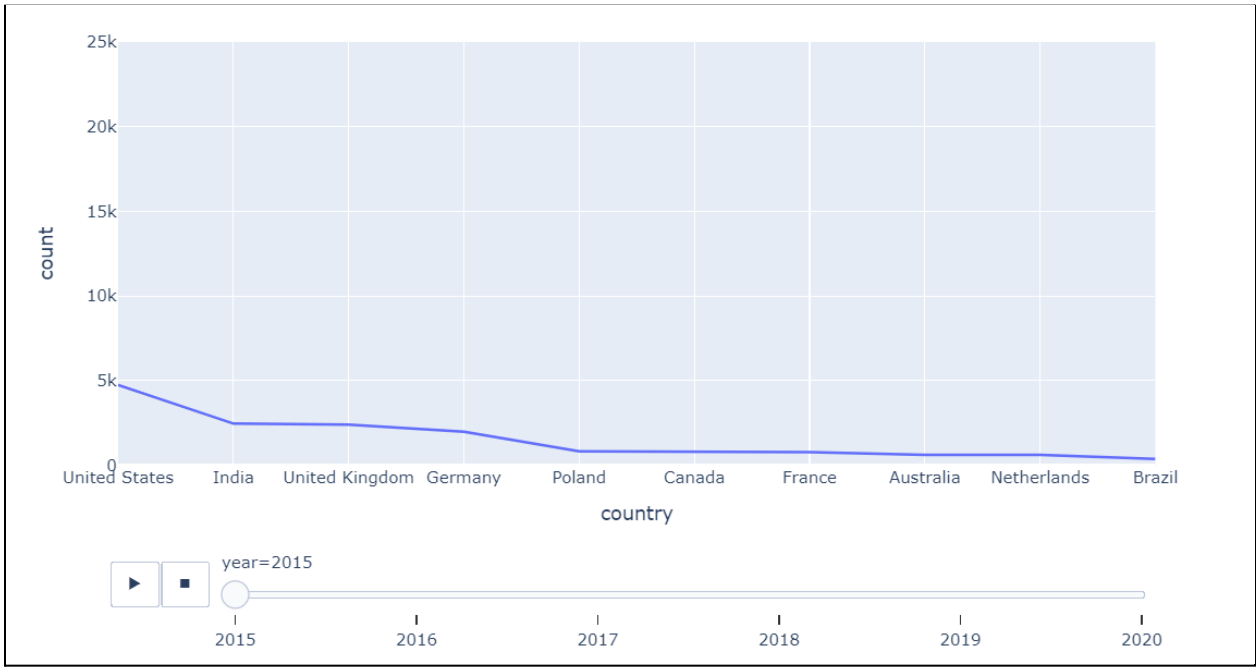
```
df.groupby(by='location').experience.value_counts().sort_values(ascending=False)
```

```
location                                          experience
Bengaluru / Bangalore                             5-10 Years     551
Ahmedabad                                         0-3 Years      374
Delhi, Mumbai                                     6-15 Years     296
Bengaluru / Bangalore, Hyderabad / Secunderabad   6-15 Years     296
Bengaluru / Bangalore                             5-8 Years      264
                                                                 ...
Delhi, India                                      3-4 Years        1
Delhi, Hyderabad / Secunderabad                   8-10 Years       1
                                                  2-12 Years       1
                                                  10-15 Years      1
Gurgaon / Gurugram, Pune                          1-10 Years       1
Name: experience, Length: 1952, dtype: int64
```

StackOverflow Dataset

EDA

```
import pandas as pd
import numpy as np
import plotly.express as px
df_country = pd.read_csv('so_yearly/so_countries_2015-20.csv')
countries = df_country.query("year == 2020").sort_values(by='count',
ascending=False).country[:10].values.tolist()

graph_df = df_country[df_country.country.isin(countries)]
px.line(graph_df,
           animation_frame="year",
           x="country",
           y="count",
           hover_name="country",
            range_y=[0,25000]
           )
```




```
df_country.head()
```

	country object	count int64	year int64
0	United States	4745	2015
1	India	2461	2015
2	United Kingdom	2403	2015
3	Germany	1976	2015
4	Poland	833	2015

5 rows × 3 columns

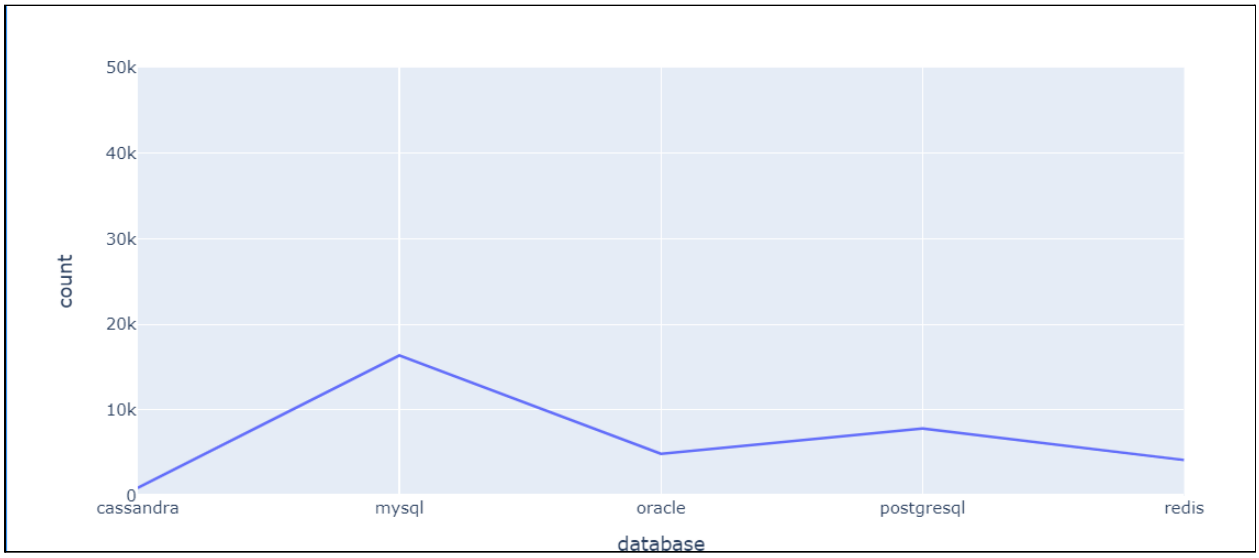
```
df_database = pd.read_csv('so_yearly/so_databases_2017-20.csv')
df_database.head()
```

	database object	count int64	year int64
0	MySQL	16375	2017
1	SQLite	7838	2017
2	MongoDB	6192	2017
3	Redis	4143	2017
4	SQL Server	11358	2017

5 rows × 3 columns

```
selected_databases = ['oracle', 'mysql', 'redis', 'cassandra', 'postgresql']

filtered_list = []

def databases_validator(row):
    for db in selected_databases:
        if(db in row.database.lower()):
            filtered_list.append({'database': db, 'count': row['count'],
'year':row['year']})

_ = df_database.agg(databases_validator, axis=1)
print(len(filtered_list))
df_filtered_databases = pd.DataFrame(filtered_list).groupby(['database',
'year']).agg({'count': 'sum'}).reset_index()

px.line(df_filtered_databases,
           animation_frame="year",
           x="database",
```

```
        y="count",
        hover_name="database",
         range_y=[0,50000]
        )
```



Dev-Types

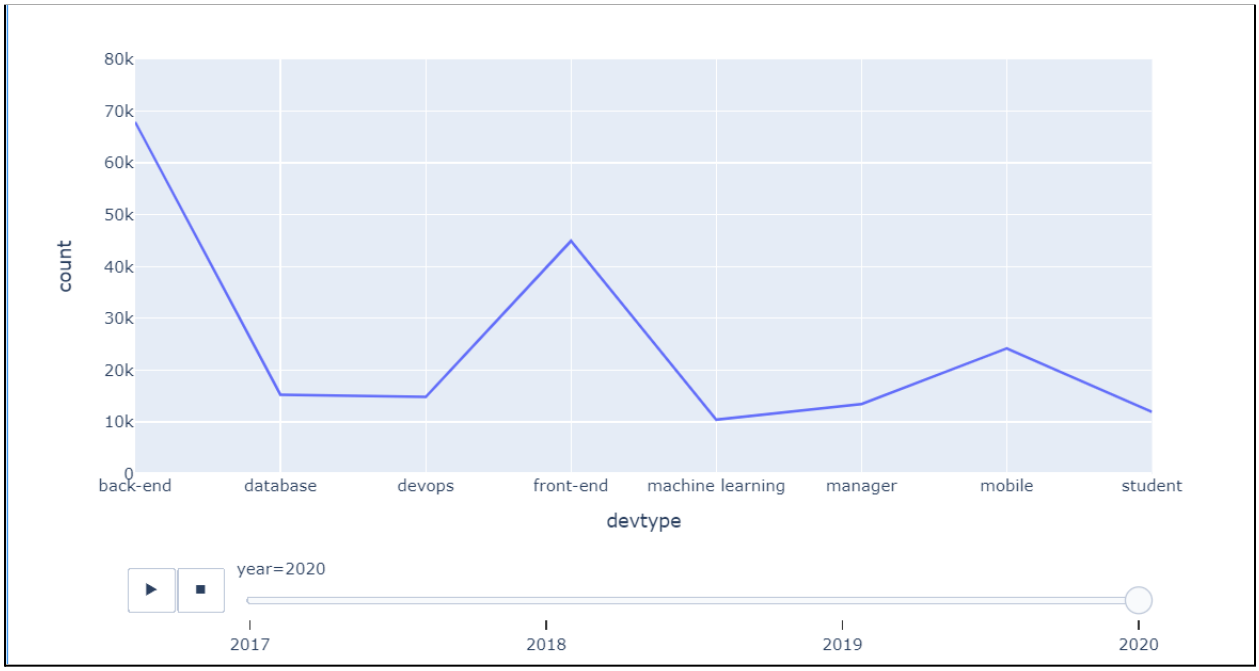
```
df_devtype = pd.read_csv('so_yearly/so_devtypes_2015-20.csv')
df_devtype.head()
```

	dev_type object	count int64	year int64
0	Full-stack web developer	6765	2017
1	Student	2845	2017
2	Back-end web developer	2104	2017
3	Desktop developer	1735	2017
4	Front-end web developer	1242	2017

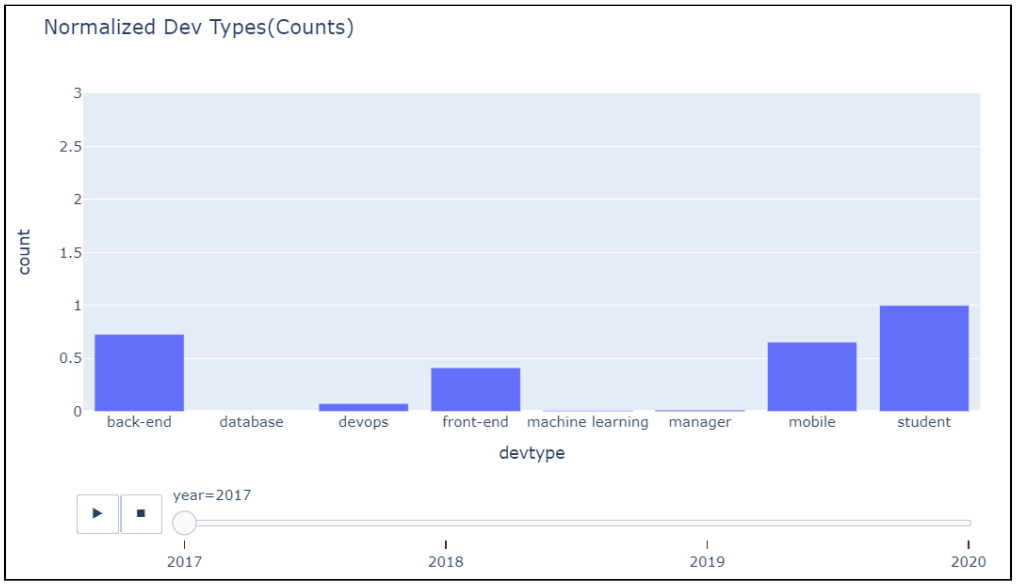
5 rows × 3 columns

```
common_skills = ['back-end', 'front-end', 'machine learning', 'manager', 'student',
'devops', 'mobile', 'database']
filtered_list = []

def skills_validator(skill):
    for filter_skill in common_skills:
        if(filter_skill in skill.dev_type.lower()):
            # filtered_list_2020[filter_skill] =
filtered_list_2020.get(filter_skill, 0) + skill['count']
            filtered_list.append({'devtype': filter_skill, 'count': skill['count'],
'year':skill['year']})

_ = df_devtype.agg(skills_validator, axis=1)
print(len(filtered_list))
from sklearn.preprocessing import StandardScaler

# Normalize the count with respect to each year group.

def normalize(feature_name):
```

```
    def feature(df):
        result = df.copy()
        max_value = df[feature_name].max()
        min_value = df[feature_name].min()
        result[feature_name] = (df[feature_name] - min_value) / (max_value -
min_value)
        return result

    return feature

df_filtered_dev_types = pd.DataFrame(filtered_list).groupby(['devtype',
'year']).agg({'count': 'sum'}).reset_index()
df_normalized_devtype =
df_filtered_dev_types.groupby("year").apply(normalize('count')).reset_index(drop=Tr
ue)
px.line(df_filtered_dev_types,
           animation_frame="year",
           x="devtype",
           y="count",
           hover_name="devtype",
            range_y=[0,80000]
           )
```



```
px.bar(df_normalized_devtype,
           animation_frame="year",
           x="devtype",
           y="count",
           hover_name="devtype",
           range_y=[0,3],
           title='Normalized Dev Types(Counts)')
```



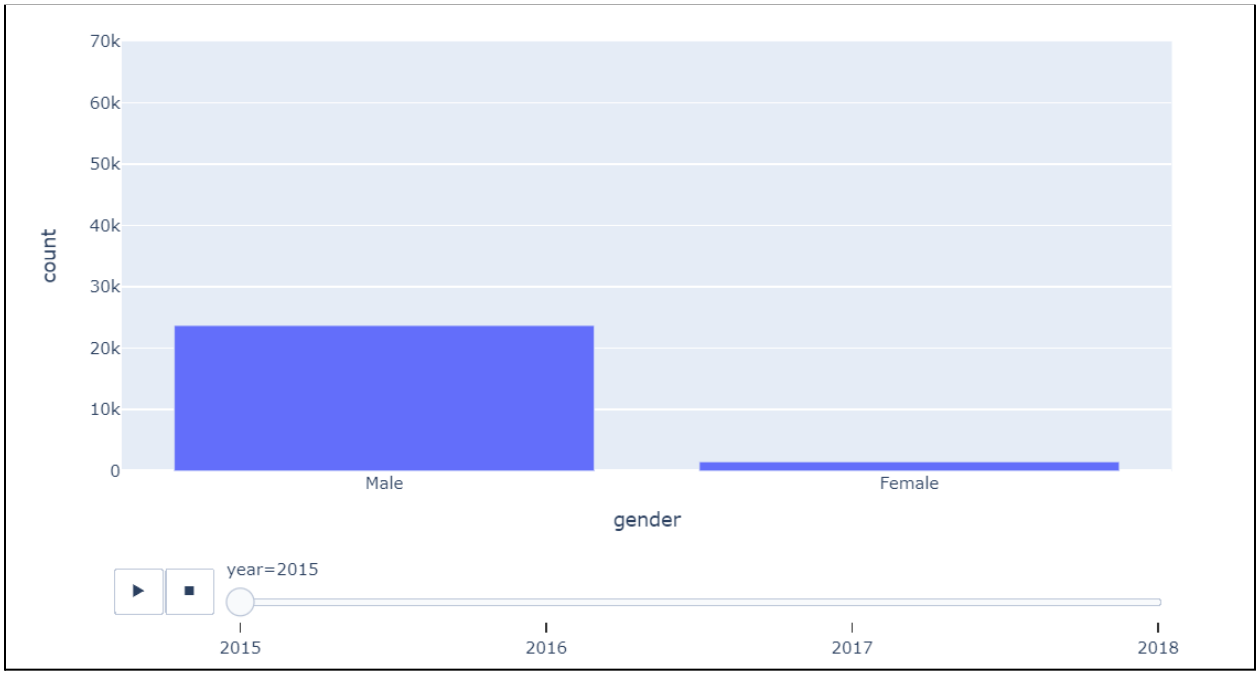
Gender

```
df_gender = pd.read_csv('so_yearly/so_gender_2015-20.csv')
df_gender.head()
```

	gender object	count int64	year int64
0	Male	23699	2015
1	Female	1480	2015
2	Prefer not to disclose	437	2015
3	Other	128	2015
4	Male	51388	2016

5 rows × 3 columns

```
gendy = df_gender[df_gender.gender.isin(['Male', 'Female'])]
px.bar(gendy,
           animation_frame="year",
           x="gender",
           y="count",
           hover_name="gender",
            range_y=[0,70000]
           )

# px.bar(gendy, x='year', y='count')
```



Languages

```
df_language = pd.read_csv('so_yearly/so_languages-2016-20.csv')
df_language.head()
```

	languages object	count int64	year int64
0	iOS	4498	2017
1	Objective-C	3202	2017
2	Android	8601	2017
3	Arduino / Raspberry Pi	3797	2017
4	AngularJS	8823	2017

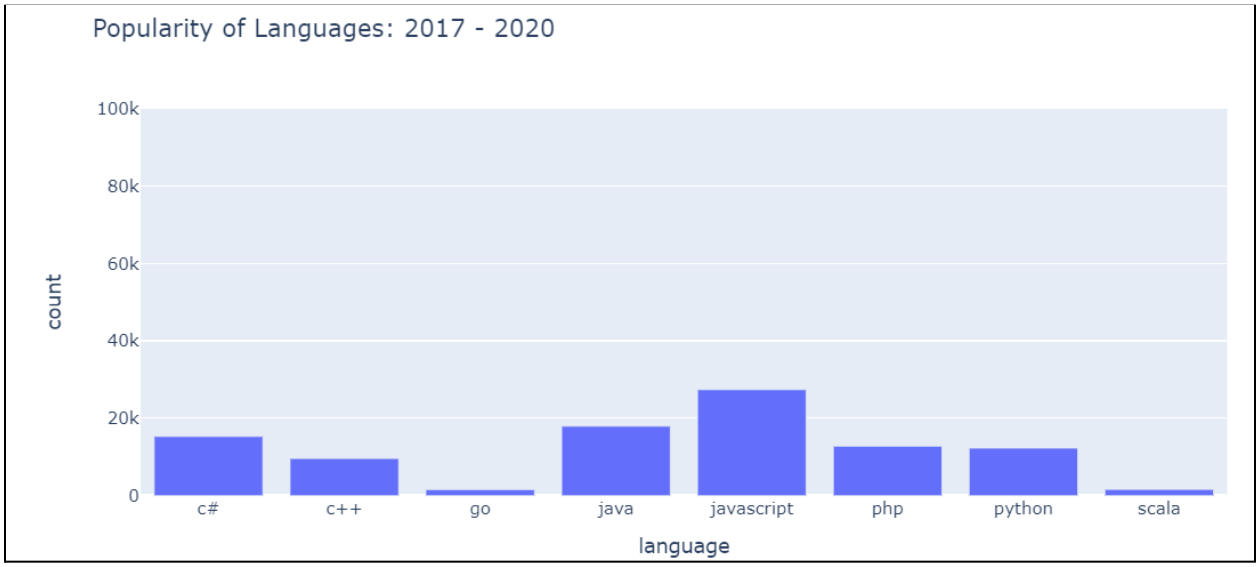
5 rows × 3 columns

```
languages = ['python', 'javascript','java', 'scala', 'php', 'go', 'c++', 'c#']

filtered_list = []

def languages_validator(row):
    for lang in languages:
        if(lang == row.languages.lower()):
            filtered_list.append({'language': lang, 'count': row['count'], 'year':
row['year']})

_ = df_language.agg(languages_validator, axis=1)
print(len(filtered_list))
from sklearn.preprocessing import StandardScaler

# Normalize the count with respect to each year group.

def normalize(feature_name):

    def feature(df):
        result = df.copy()
        max_value = df[feature_name].max()
```

```
        min_value = df[feature_name].min()
        result[feature_name] = (df[feature_name] - min_value) / (max_value -
min_value)  + 1
        return result

    return feature

df_filtered_languages = pd.DataFrame(filtered_list).groupby(['language',
'year']).agg({'count': 'sum'}).reset_index()
df_normalized_languages =
df_filtered_languages.groupby("year").apply(normalize('count')).reset_index(drop=Tr
ue)
df_filtered_languages.language.unique()
```

```
array(['c', 'c#', 'c++', 'go', 'java', 'javascript', 'php', 'python',
       'scala'], dtype=object)
```

```
language = df_language.query("year == 2020").sort_values(by='count',
ascending=False).languages[:10].values.tolist()

lang_df = df_language[df_language.languages.isin(language)]
px.bar(df_filtered_languages,
           animation_frame="year",
           x="language",
           y="count",
           hover_name="language",
            range_y=[0,100000],
            title='Popularity of Languages: 2017 - 2020')
```



Platforms

```
df_platforms = pd.read_csv('so_yearly/so_platforms_2017-20.csv')
df_platforms.head()
```

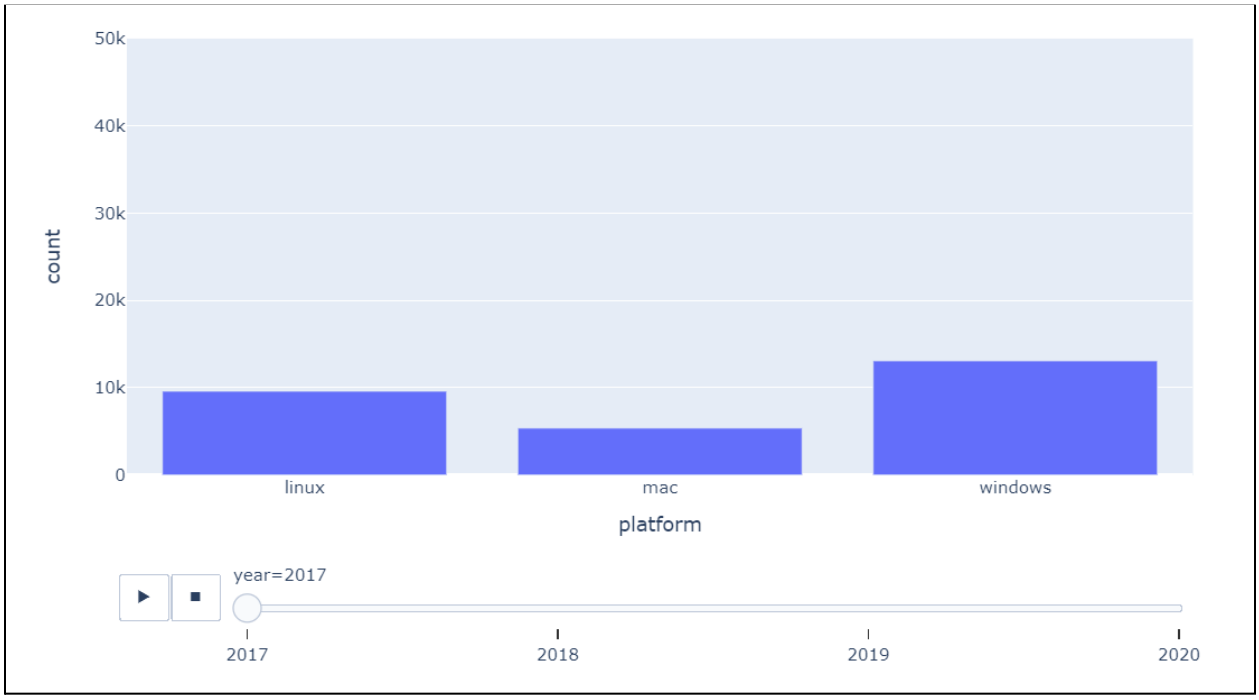
	platform object	count int64	year int64
0	iOS	4782	2017
1	Amazon Web Services (AWS)	8183	2017
2	Windows Desktop	11949	2017
3	Linux Desktop	9593	2017
4	Mac OS	5363	2017

5 rows × 3 columns

```
platforms = df_platforms.query("year == 2020").sort_values(by='count',
ascending=False).platform[:10].values.tolist()
selected_platforms = ['windows', 'mac', 'linux', ]

filtered_list = []

def platforms_validator(row):
    for platform in selected_platforms:
        if(platform in row.platform.strip().lower() and row.platform.lower() !=
'Windows Phone'):
            filtered_list.append({'platform': platform, 'count': row['count'],
'year': row['year']})

_ = df_platforms.agg(platforms_validator, axis=1)
print(len(filtered_list))
```

```
df_selected_platforms = pd.DataFrame(filtered_list).groupby(['platform',
'year']).agg({'count': 'sum'}).reset_index()
```

```
px.bar(df_selected_platforms,
           animation_frame="year",
           x="platform",
           y="count",
           hover_name="platform",
            range_y=[0,50000]
           )
```



Text Analysis

```
import pandas as pd
import re
from datetime import date
import numpy as np
import math
from nltk import everygrams
import nltk
from nltk.corpus import stopwords
from textblob import TextBlob
from nltk.stem.snowball import SnowballStemmer

from sklearn import feature_extraction
from sklearn.feature_extraction.text import TfidfVectorizer
import nltk
import ssl

try:
    _create_unverified_https_context = ssl._create_unverified_context
except AttributeError:
    pass
else:
    ssl._create_default_https_context = _create_unverified_https_context

nltk.download('wordnet')
nltk.download('stopwords')
nltk.download('punkt')
```

```
df = pd.read_csv('internshala_clean.csv')
```

```
new_df =
df.rename(columns={'about_the_work_from_home_job/internship':'about_internship'})
text_analysis_df = new_df[['about_internship', 'other_requirements']]
text_analysis_df.fillna('', inplace=True)
```

```
text_analysis_df
```

	about_internship	other_requirements
0	Selected intern's day-to-day responsibilities ...	1. Experience in hands-on development and trou...
1	Selected intern's day-to-day responsibilities ...	
2	Selected intern's day-to-day responsibilities ...	
3		
4	Selected intern's day-to-day responsibilities ...	
...	...	...
2369	Selected intern's day-to-day responsibilities ...	
2370	Selected intern's day-to-day responsibilities ...	
2371	1. Working on direct client projects and handl...	
2372		
2373	Selected intern's day-to-day responsibilities ...	

2374 rows × 2 columns

Text Preprocessing

```
def preprocess(dataframe, feature, function):
    dataframe.loc[:, feature] = dataframe[feature].apply(function)
def convert_to_lowercase(s):
    if(type(s) == str):
        return s.lower()
    else:
        return None

def remove_special_char(s):
    return s.replace('[^\w\s]','') if type(s) == str else None

def remove_stop_words(s):
    if type(s) == str:
        stop = stopwords.words('english')
        return " ".join(x for x in s.split() if x not in stop)
    else:
        return None

def correct_spellings(s):
    if(type(s) == str):
        return str(TextBlob(s).correct())
```

```
    else:
        return None

stemmer = SnowballStemmer("english")
def tokenize_and_stem(text):
    '''
        Functions for sentence tokenizer, to remove numeric tokens and raw
#punctuation
    '''
    if type(text) != str: return None
    tokens = [word for sent in nltk.sent_tokenize(text) for  word in
nltk.word_tokenize(sent)]
    filtered_tokens = []
    for token in tokens:
        if re.search('[a-zA-Z]', token):
            filtered_tokens.append(token)
    stems = [stemmer.stem(t) for t in filtered_tokens]
    return stems

preprocessing_functions = [convert_to_lowercase, remove_special_char,
remove_stop_words]

for func in preprocessing_functions:
    preprocess(text_analysis_df, 'about_internship', func)
```

Keyword Extraction using TF-IDF

```
tfidf_vectorizer = TfidfVectorizer(max_features=200000,stop_words='english',
use_idf=True, ngram_range=(1,3))

tfidf_matrix = tfidf_vectorizer.fit_transform(text_analysis_df.about_internship)

terms = tfidf_vectorizer.get_feature_names()
feature_array = np.array(tfidf_vectorizer.get_feature_names())
tfidf_sorting = np.argsort(tfidf_matrix.toarray()).flatten()[::-1]

n = 3
top_n = feature_array[tfidf_sorting][:n]
data = np.squeeze(np.sum(tfidf_matrix.todense(), axis=0).reshape(-1,1)).tolist()[0]
keyword_series = pd.Series(data=data, index=tfidf_vectorizer.get_feature_names())
keyword_series = keyword_series.sort_values(ascending=False)
keyword_series[30: 50]
```

```
new                                19.093998
develop                            17.120192
software                           16.124382
user                               15.409647
react                              15.270170
include work                       15.121716
mobile                             15.089814
responsibilities include work      15.075362
features                           14.734603
android                            14.611351
data                               14.470755
designing                          14.384072
creating                           14.305479
performance                        14.225763
create                             14.196863
backend                            14.098060
js                                 13.928468
testing                            13.710084
build                              13.408841
apis                               13.213017
dtype: float64
```

```
feature_array[tfidf_sorting][:200]
```

```
array(['web', 'web pages', 'pages', 'designs', 'languages turn photoshop',
       'files animated responsive', 'web technologies work',
       'management prototype', 'management prototype ideas',
       'wireframes visual designs', 'wireframes visual',
       'prototype ideas', 'prototype ideas order',
       'multiple browsers mobile', 'files animated',
       'web based animation', 'css web pages', 'elevate concepts level',
       'javascript scripting languages', 'css web', 'elevate concepts',
       'fellow team members', 'commerce using',
       'commerce using javascript', 'elevate', 'web pages based',
       'effective communication', 'effective communication fellow',
       'tools ensure technical', 'css3 web based', 'javascript scripting',
       'fellow team', 'members management',
       'devices thoroughly demonstrate', 'engage development',
       'engage development websites', 'concepts level',
       'development websites includes', 'scripting languages turn',
       'include engage development', 'animation tools ensure',
       'animation tools', 'dynamic engaging', 'dynamic engaging web',
       'demonstrate effective communication', 'development html css',
       'css3 web', 'engaging web', 'languages turn',
       'members management prototype', 'demonstrate effective',
       'work development dynamic', 'functionality commerce',
       'animated responsive web', 'work multiple browsers',
       'animated responsive', 'animated', 'functionality commerce using',
       'visual designs implementation', 'using javascript scripting',
       'tools ensure', 'pages based wireframes', 'photoshop files',
       'pages based', 'works elevate', 'ux designs test',
       'includes development', 'html css web', 'browsers mobile devices',
       'browsers mobile', 'open source web', 'photoshop files animated',
       'includes development html', 'thoroughly demonstrate effective',
       'thoroughly demonstrate', 'team members management',
       'communication fellow team', 'ideas order learn', 'source web',
```

Keyword Extraction using YAKE

```
from yake import KeywordExtractor

all_keywords = {}
kw_extractor = KeywordExtractor(lan="en")

def yake_keyword_extractor(input_str):
    keywords = kw_extractor.extract_keywords(text=input_str)
    keywords = [x for x, y in keywords]
    for keyword in keywords:
        all_keywords[keyword] = all_keywords.get(keyword, 0) + 1
    return ','.join(keywords)

text_analysis_df.about_internship.apply(yake_keyword_extractor)
```

```
0       selected intern,responsibilities include,selec...
1       selected intern,responsibilities include,selec...
2       selected intern,responsibilities include,selec...
3
4       selected intern,responsibilities include,selec...
                              ...
2369    selected intern,responsibilities include,selec...
2370    selected intern,responsibilities include,artif...
2371    working,experience,development,gain,working di...
2372
2373    selected intern,responsibilities include,web,i...
Name: about_internship, Length: 2374, dtype: object
```

```
keywords
```

```
['selected intern',
 'responsibilities include',
 'selected',
 'intern',
 'responsibilities',
 'include',
 'css practices',
 'interface',
 'creating website layout',
 'creating website',
 'modern html',
 'application',
 'codes',
 'team',
 'website layout',
 'layout using modern',
 'creating',
 'html',
 'css',
 'practices']
```

Keyword word extraction using Count Vectorizer

```
count_vec = feature_extraction.text.CountVectorizer()
```

```
response = count_vec.fit_transform(text_analysis_df.about_internship)

feature_array = np.array(count_vec.get_feature_names())
count_sorting = np.argsort(response.toarray()).flatten()[::-1]

feature_array[count_sorting][:100]
```

```
array(['web', 'development', 'pages', 'based', 'designs', 'day', 'using',
       'work', 'engaging', 'websites', 'animation', 'level', 'commercial',
       'effective', 'to', 'intern', 'communication', 'tools',
       'responsibilities', 'feasibility', 'ideas', 'elevate', 'scripting',
       'mobile', 'fellow', 'commerce', 'animated', 'functionality',
       'visual', 'photoshop', 'test', 'prototype', 'multiple', 'dynamic',
       'includes', 'include', 'technologies', 'thoroughly', 'technical',
       'devices', 'ux', 'team', 'source', 'next', 'css', 'css3',
       'management', 'implementation', 'responsive', 'wireframes',
       'learn', 'languages', 'html', 'engage', 'order', 'demonstrate',
       'concepts', 'selected', 'javascript', 'turn', 'works', 'browsers',
       'members', 'open', 'ensure', 'ui', 'files', 'html5', 'es5',
       'everything', 'evse', 'evolving', 'enhancement', 'enhancements',
       'enhancing', 'error', 'evolve', 'evolution', 'everyday', 'erp',
       'enormous', 'every', 'events', 'event', 'envisaged', 'enriching',
       'enrichment', 'environments', 'evueme', 'example', 'ex',
       'evaluations', 'executing', 'engineering', 'executes', 'executed',
       'execute', 'exciting', 'eod', 'equivalent'], dtype='<U19')
```

```
count_vect_df = pd.DataFrame(response.todense(),
columns=count_vec.get_feature_names())
keywords_series = pd.Series(all_keywords)

keywords_series.sort_values(ascending=False)[:160]
```

```
selected intern             1654
responsibilities include    1646
selected                    1632
intern                      1621
responsibilities            1611
                            ...
design build                  18
designing building            18
bugs                          18
products                      18
technical                     18
Length: 160, dtype: int64
```

```
data = np.squeeze(np.sum(response.todense(), axis=0).reshape(-1,1)).tolist()[0]
```

```
count_vect_df['11'].sum()
#32
```

```
keyword_series = pd.Series(data=data, index=count_vec.get_feature_names())
keyword_series = keyword_series.sort_values(ascending=False)
```

```
keyword_series[:40]
```

```
day                                 87.135030
working                             44.943190
day day                             43.392380
intern                              43.262979
include                             43.068278
selected                            42.988183
selected intern                     42.896717
responsibilities                    42.871539
intern day                          42.859560
intern day day                      42.859560
selected intern day                 42.859288
day responsibilities                42.840408
day day responsibilities            42.840408
responsibilities include            42.819319
day responsibilities include        42.801609
work                                41.842245
development                         31.706766
website                             28.948596
web                                 26.869604
end                                 25.845426
using                               24.893811
design                              24.319796
applications                        23.688710
developing                          22.060565
code                                21.878531
include working                     21.613985
responsibilities include working    21.613985
team                                21.058146
application                         20.762955
app                                 20.473102
new                                 19.093998
develop                             17.120192
```

Word Cloud

```
from wordcloud import WordCloud
import matplotlib.pyplot as plt

def get_word_cloud(input_str):
    wordcloud = WordCloud().generate(input_str)
    plt.figure(figsize=(10,5))
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.axis("off")
get_word_cloud(str(keyword_series[10:].index))
```

Conclusion

Growing number of challenges in web crawling

- The web is a dynamic space that doesn't have a set standard for data formats and structures. Collecting data in a format that can be understood by machines can be a challenge due to the lack of uniformity. For instance, a webpage can be created using HTML, CSS, Java, PHP, or XML. The process of data extraction becomes challenging when web crawlers need structured data on a massive scale. The problem gets amplified when the web crawlers have to extract data from thousands of web sources pertaining to a specific schema.
- A majority of websites update their content on a daily or hourly basis. The crawler has to download all these pages to provide updated information to the user. The problem arises when the crawler starts downloading all such pages as it puts unnecessary pressure on internet traffic.

Data cleaning is a crucial yet very complicated task

- There is no consistent data format. Some of the data is in a structured, readily understandable format. This kind of data is usually quite easy to clean, parse and analyze. However, some of the data is really messy, and cannot be used as is for analysis. This includes missing data, irregularly formatted data, and irrelevant data which is not worth analyzing at all.
- The volume of data that businesses deal with on a day to day basis is in the scale of terabytes or even petabytes. Making sense of all this data, coming from a variety of sources and in different formats is, undoubtedly, a huge task. There are a whole host of tools designed to ease this process today, but it remains an incredibly tricky challenge to sift through the large volumes of data and prepare it for analysis.
- Data cleansing can be quite an exhaustive and time-consuming task, especially for data scientists. Cleaning the data requires removal of duplications, removing or replacing missing entries, correcting misfielded values, ensuring consistent formatting and a host of other tasks which take a considerable amount of time.

- It is very difficult to take advantage of the intrinsic value offered by the dataset if it does not adhere to the quality standards set by the business, making data cleaning a crucial component of the data analysis process. Therefore data cleaning cannot be ignored.

Growing Technologies

- Web Development is the most popular and common stack, found in internships as well as in jobs. It's stack is dominated by HTML, Javascript and CSS, Bootstrap is the only CSS framework that made it to the list of skills. And Node.JS is a popular choice for backend in most of the internship requests.
- Javascript continues to dominate the language stack across all forms, in Internships as well as stackoverflow surveys.
- Surprising take away from most occurring skills in Internshala is React.JS leading the chart at 4th position and Node.JS after that.
- PHP comes in the top 10 most occurring stacks.
- MySQL is the most used database engine in Internshala, Monster India and Stackoverflow, followed by PostgreSQL.
- In Internshala, the skills in most applied internships are Bootstrap with 16k applicants and MongoDB at second spot with almost 16k applicants.
- In mobile development, Native Android is seen to be leading technology, surprisingly flutter has more popularity then IOS or other options in Internships.
- Node.JS and MongoDB is the most popular choice in backend.

References

Articles

1) Regular Expressions Overview (evermap.com)
2) Women in Tech
3) FP Growth Tutorial
4) Introduction to Plotly: An interactive exploration | Deepnote

Documentations

1) Plotly Python Open Source Graphing Library
2) Seaborn: Statistical Data Visualization
3) Pandas: open source data analysis and manipulation tool
4) scikit-learn: machine learning in Python — scikit-learn 0.16.1 documentation

Books

1) Data Mining And Predictive Analytics by Daniel T. Larose, Chantal D. Larose
2) Data Mining Concepts and Techniques by Jiawei Han and Micheline Kamber
3) Getting Structured Data from the Internet by Jay M. Patel