# MergedAssignment

May 26, 2024

A report by Molly Murray (molly.murray@griffithuni.edu.au), Kobi Chambers (kobi.chambers@griffithuni.edu.au) and Joseph Fernando (joseph.fernando@griffithuni.edu.au)

[19]:
```
# Imports
import string
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from IPython import display
from IPython.display import HTML
%matplotlib inline
```

## 1 0.0 Preprocessing

### 1.1 0.1 Load and visualise the dataset

[66]:
```
df = pd.read_csv("data.csv", on_bad_lines='skip', engine='python')
```

[67]:
```
df.head()
```

[67]:
```
         Id                                             Title  \
0  37404348                           Casual Stock Replenisher
1  37404337                           Casual Stock Replenisher
2  37404356  RETAIL SALES SUPERSTARS and STYLISTS Wanted - …
3  37404330                              Team member - Belrose
4  37404308  Business Banking Contact Centre Specialist, Ni…

                                         Company                      Date  \
0                                    Aldi Stores  2018-10-07T00:00:00.000Z
1                                    Aldi Stores  2018-10-07T00:00:00.000Z
2                           LB Creative Pty Ltd  2018-10-07T00:00:00.000Z
3                        Anaconda Group Pty Ltd  2018-10-07T00:00:00.000Z
4  Commonwealth Bank - Business & Private Banking  2018-10-07T00:00:00.000Z

              Location                         Area  \
0               Sydney  North West & Hills District
1    Richmond & Hawkesbury                       NaN
```

```
2                Brisbane         CBD & Inner Suburbs
3  Gosford & Central Coast                        NaN
4                  Sydney       Ryde & Macquarie Park

                   Classification  SubClassification  \
0      Retail & Consumer Products  Retail Assistants
1      Retail & Consumer Products  Retail Assistants
2      Retail & Consumer Products  Retail Assistants
3      Retail & Consumer Products  Retail Assistants
4  Call Centre & Customer Service    Sales - Inbound

                                         Requirement FullDescription  \
0  Our Casual Stock Replenishers pride themselves…             NaN
1  Our Casual Stock Replenishers pride themselves…             NaN
2  BRAND NEW FLAGSHIP STORE OPENING - SUNSHINE PLAZA             NaN
3  Bring it on - do you love the great outdoors a…             NaN
4  We are seeking highly articulate, enthusiastic…             NaN

   LowestSalary  HighestSalary JobType
0             0             30     NaN
1             0             30     NaN
2             0             30     NaN
3             0             30     NaN
4             0             30     NaN
```

[68]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21673 entries, 0 to 21672
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Id                 21673 non-null  int64
 1   Title              21673 non-null  object
 2   Company            20781 non-null  object
 3   Date               21673 non-null  object
 4   Location           21673 non-null  object
 5   Area               13501 non-null  object
 6   Classification     21673 non-null  object
 7   SubClassification  21673 non-null  object
 8   Requirement        21673 non-null  object
 9   FullDescription    21187 non-null  object
 10  LowestSalary       21673 non-null  int64
 11  HighestSalary      21673 non-null  int64
 12  JobType            21200 non-null  object
dtypes: int64(3), object(10)
memory usage: 2.1+ MB
```

```
[70]: df.describe()
```

```
[70]:                  Id  LowestSalary  HighestSalary
      count  2.167300e+04  21673.000000   21673.000000
      mean   3.738861e+07     59.860195      78.924007
      std    2.370948e+04     40.541324      43.979397
      min    3.679829e+07      0.000000      30.000000
      25%    3.738467e+07     30.000000      40.000000
      50%    3.739185e+07     60.000000      70.000000
      75%    3.739844e+07     80.000000     100.000000
      max    3.740440e+07    150.000000     200.000000
```

```
[73]: df.isna().sum()
```

```
[73]: Id                     0
      Title                  0
      Company              892
      Date                   0
      Location               0
      Area                8172
      Classification         0
      SubClassification      0
      Requirement            0
      FullDescription      486
      LowestSalary           0
      HighestSalary          0
      JobType              473
      dtype: int64
```

The dataset contains a total of 318,477 job postings from the website seek from the collected between the 1st of October 2018 and the 13th of March 2019. The dataset has 13 categories, the variations of which can be seen below.

The columns consist of a mix of data types 2 integer columns and 12 string columns. Some columns have missing values, which will be addressed during data preparation.

```
[6]: import pandas as pd
     from tabulate import tabulate

     df = pd.read_csv('data.csv')

     all_columns = ['Id', 'Title', 'Company', 'Date', 'Location', 'Area',
                    'Classification', 'SubClassification', 'Requirement',
                    'FullDescription', 'LowestSalary', 'HighestSalary', 'JobType']

     results = []

     for column in all_columns:
```

```python
    num_variations = df[column].nunique()
    results.append([column, num_variations])

print(tabulate(results, headers=['Category', 'Number of Variations'],
 ↪tablefmt='pretty'))

num_variations_id = next(item[1] for item in results if item[0] == 'Id')
num_variations_date = next(item[1] for item in results if item[0] == 'Date')

id_to_date_ratio = num_variations_id / num_variations_date

print("Ratio of unique IDs to unique dates:", id_to_date_ratio)
```

```
/var/folders/4b/x1qmm8g167d86vjglntrzwdh0000gn/T/ipykernel_15687/447026757.py:4:
DtypeWarning: Columns (0,4,5,6,7) have mixed types. Specify dtype option on
import or set low_memory=False.
  df = pd.read_csv('data.csv')
```

```
+-------------------+----------------------+
|     Category      | Number of Variations |
+-------------------+----------------------+
|        Id         |        318477        |
|       Title       |        168065        |
|      Company      |        40628         |
|       Date        |         163          |
|     Location      |          65          |
|       Area        |          19          |
|  Classification   |          30          |
| SubClassification |         338          |
|    Requirement    |        234287        |
|  FullDescription  |        250901        |
|    LowestSalary   |          11          |
|   HighestSalary   |          11          |
|      JobType      |          4           |
+-------------------+----------------------+
Ratio of unique IDs to unique dates: 1953.846625766871
```

From this data we can learn that each new job has a unique identifier. There are fewer unique titles compared to the total number of records, implying that there might be multiple job listings with the same title. However there are still far more variations for job title than other categories so it is safe to assume there is no uniformed way to name the jobs. Similarly there are significantly less company variations suggesting that some companies must have multiple joblistings.

There are 163 dates from the time period between 1st of October 2018 and the 13th of March 2019 suggesting there were around 1953 new job listings per day.

There are 65 unique locations but only 19 unique areas suggesting some entries with locations may not have had a specified area within that location however the low number of the Location variable means it will be useful in our analysis. LowestSalary and HighestSalary are numerical variables, making them valuable in our analysis as their 11 ranges can be easily compared with catagorical data such as job type or classification.

There are a large number of unique requirements and full descriptions, indicating this data is diverse across the dataset and also shows a lack of uniformity in the writing styles. Conversely there are 4 unique job types, indicating that the categorical variable may have had 4 set options to choose from. As this variable has so little categories it could be useful when determining the relationship between salary range and the job type.

## 1.2 0.2 Clean the dataset

### 1.2.1 0.2.1 Clean Id column

[78]:
```
def verifyUniques(total, sample, message):
    print(message, sample)

    if (total == sample):
        print("\tCorrectly matches length of dataset.")
    else:
        print("\tDoes not match length of dataset. Difference: ", total -␣
 ↪sample)
```

[80]:
```
verifyUniques(total=len(df), sample=len(df['Id'].unique()), message="Unique Id␣
 ↪count:")
```

```
Unique Id count: 21673
        Correctly matches length of dataset.
```

[82]:
```
def findNonNumericCells(column_name):
    numeric_list = []
    non_numeric_list = []
    for cell in df[column_name]:
        if str(cell).isdigit():
            numeric_list.append(cell)
        else:
            non_numeric_list.append(cell)

    print("Non-numeric count:", len(non_numeric_list))

    print(f"\nNon-numeric sampling:\n{non_numeric_list[:
 ↪3]}\n{non_numeric_list[-3:]}")

    print(f"\nNumeric sampling:\n{numeric_list[:3]}\n{numeric_list[-3:]}")
```

[84]:
```
findNonNumericCells(column_name="Id")
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
```

```
Cell In[84], line 1
----> 1 findNonNumericCells(column_name="Id")

Cell In[82], line 5, in findNonNumericCells(column_name)
      3 non_numeric_list = []
      4 for cell in df[column_name]:
----> 5     if cell.isdigit():
      6         numeric_list.append(cell)
      7     else:

AttributeError: 'int' object has no attribute 'isdigit'
```

It seems we should use regex to remove characters after the first digits. i.e. everything from the '&' symbol onwards.

[87]:
```python
df['Id'] = df['Id'].replace(to_replace=r'&.*', value='', regex=True)
```

[89]:
```python
findNonNumericCells(column_name="Id")
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[89], line 1
----> 1 findNonNumericCells(column_name="Id")

Cell In[82], line 5, in findNonNumericCells(column_name)
      3 non_numeric_list = []
      4 for cell in df[column_name]:
----> 5     if cell.isdigit():
      6         numeric_list.append(cell)
      7     else:

AttributeError: 'int' object has no attribute 'isdigit'
```

### 1.2.2 0.2.2 Clean date column

Date seems to be in format: 2018-10-07T00:00:00.000Z

[93]:
```python
df['Date'] = df['Date'].replace(to_replace=r'T.*', value='', regex=True)
```

[95]:
```python
df['Date'] = pd.to_datetime(df['Date']).dt.normalize()
print(df['Date'].dtype)
```

```
datetime64[ns]
```

### 1.2.3 0.2.3 Remove NA requirement rows, since there are only 7 instances

[98]:
```python
df = df[df['Requirement'].notna()]
```

### 1.2.4 0.2.4 Identify unique counts for categorical columns

[101]:
```python
relevant_columns = ['Location', 'Area', 'Classification', 'SubClassification',␣
↪'JobType']

for column in relevant_columns:
    num_unique = df[column].nunique()
    print(f"Number of unique values in column '{column}': {num_unique}")
```

```
Number of unique values in column 'Location': 65
Number of unique values in column 'Area': 19
Number of unique values in column 'Classification': 30
Number of unique values in column 'SubClassification': 333
Number of unique values in column 'JobType': 4
```

[103]:
```python
# Convert relevant columns to categorical dtype
for column in relevant_columns:
    df[column] = df[column].astype('category')
```

### 1.2.5 0.2.6 Drop Duplicates

[106]:
```python
print(len(df[df.duplicated()]))
```

```
0
```

[108]:
```python
df.drop_duplicates(inplace=True)
print(len(df[df.duplicated()]))
```

```
0
```

We may also have duplicate Id cells, so we'll verify they are in fact duplicate rows, then handle accordingly.

[111]:
```python
# We should assess if these really are unique rows or not.
duplicate_id_rows = df[df.duplicated(subset='Id', keep=False)]
duplicate_id_rows.sort_values(by='Id').head()
```

[111]:
```
Empty DataFrame
Columns: [Id, Title, Company, Date, Location, Area, Classification,
SubClassification, Requirement, FullDescription, LowestSalary, HighestSalary,
JobType]
Index: []
```

### 1.2.6 Handling Duplicate IDs

It appears that duplicate IDs occur when a job is updated with a new salary range. To ensure that we maintain only the most up-to-date data, we will follow these steps:

- **Action**: Keep only the last posted instance for each job ID.
- **Reasoning**: The last posted instance is likely to be more indicative of up-to-date data.
- **Approach**: Maintain the original index for dropping rows, as updates can contain the same date.

By implementing this approach, we ensure that the dataset contains the most relevant and current information while preserving the integrity of the original data.

[114]:
```
df = df.drop_duplicates(subset='Id', keep='last')
df.reset_index(drop=True, inplace=True)
```

[116]:
```
# Once again lets look at the Id uniques
verifyUniques(total=len(df), sample=len(df.index.unique()), message="Unique Id␣
↪count:")
```

```
Unique Id count: 21673
        Correctly matches length of dataset.
```

### 1.2.7 0.2.5 Create an avarage salary column

[119]:
```
df = df.assign(AverageSalary=(df['LowestSalary'] + df['HighestSalary']) / 2)
```

## 1.3 0.3 Export preprocessed dataset

[122]:
```
#Save the cleaned and preprocessed dataset
df.to_csv("preprocessed_data.csv")
```

# 2 1 Describe the dataset

## 2.1 1.1 Categories/domains of the dataset

## 2.2 1.2 Dataset size

## 2.3 1.3 Dataset structure/format

[29]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 304307 entries, 0 to 304306
Data columns (total 14 columns):
 #   Column             Non-Null Count   Dtype
---  ------             --------------   -----
 0   Id                 304307 non-null  object
 1   Title              304307 non-null  object
```

```
 2   Company            292853 non-null  object
 3   Date               304307 non-null  datetime64[ns]
 4   Location           183062 non-null  category
 5   Area               113749 non-null  category
 6   Classification     183062 non-null  category
 7   SubClassification  183062 non-null  category
 8   Requirement        304307 non-null  object
 9   FullDescription    290996 non-null  object
 10  LowestSalary       304307 non-null  int64
 11  HighestSalary      304307 non-null  int64
 12  JobType            291071 non-null  category
 13  AverageSalary      304307 non-null  float64
dtypes: category(5), datetime64[ns](1), float64(1), int64(2), object(5)
memory usage: 22.7+ MB
```

### 2.4 1.4 Attributes/Features

- **Title**: The title or position of the job posting.
- **Company**: The company offering the job position.
- **Location**: The location of the job (e.g., city, region).
- **Salary**: The salary range associated with the job.
- **Classification**: The broad classification/category of the job.
- **SubClassification**: The subcategory or specific field of the job.
- **Description**: The full description or details of the job responsibilities and requirements.

#### Date-Related Attributes related to temporal analysis

- **Date**: The date when the job posting was published.

### 2.5 1.5 Which parts of the dataset we will use

- We may use all or a subset of attributes/features depending on the analysis goals for different tasks.
- Relevant parts of the dataset may include attributes related to job titles, companies, locations, and job descriptions.

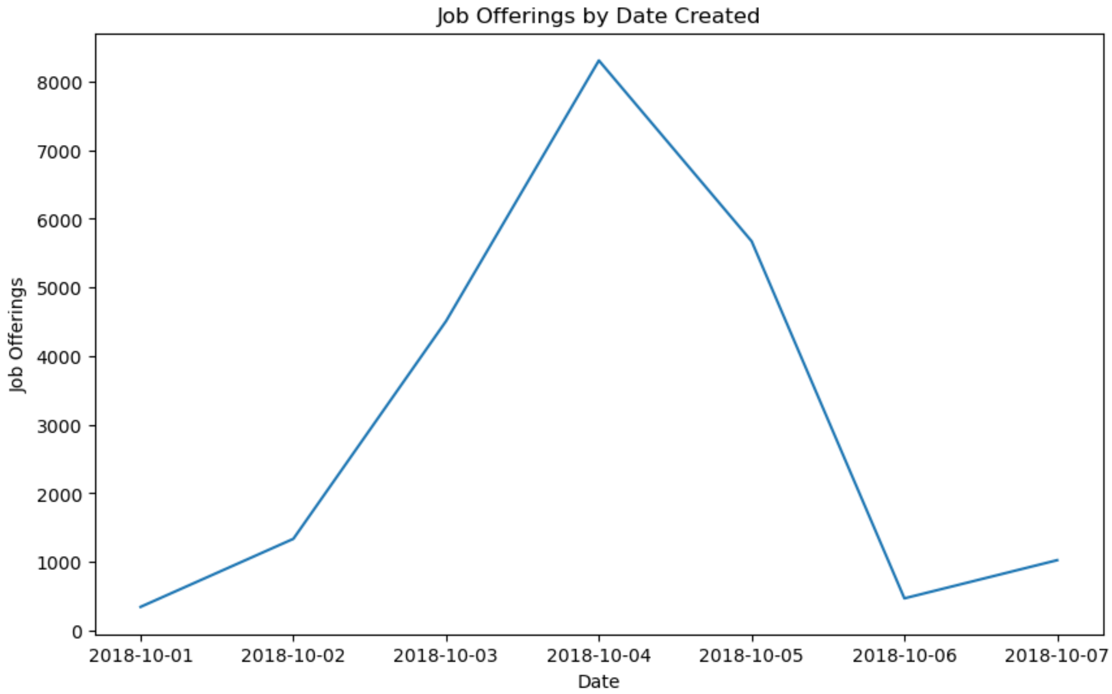
## 3 3 Hypothesis about analysis outcome

Australia is a service based economy, so it should be expected that a majority of job postings are for service related industries such as IT, retail, tourism, etc. As Sydney, Melbourne and Brisbane are the largest Australian cities, it is expected that they will have the most job postings. The top jobs by average salary should be those in management, such as CEO positions, or medicine, such as doctor or surgeon positions.

## 4 4 Study the job metadata

Range of dates the data was acquired

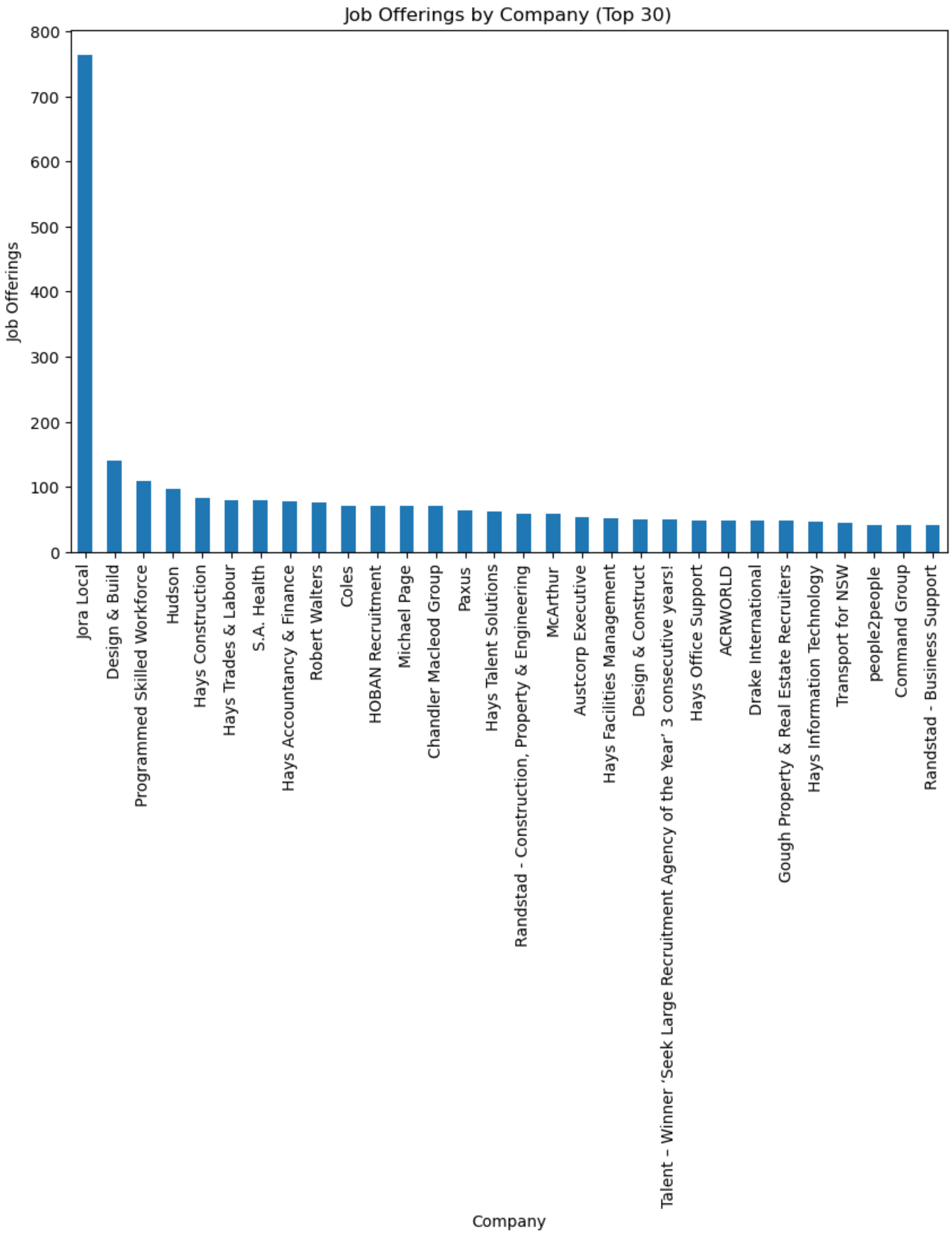
[24]:
```
metadataDf = pd.read_csv('preprocessed_data.csv')
date_counts = metadataDf['Date'].value_counts().sort_index()
date_counts.plot(kind='line', figsize=(10, 6))
plt.title('Job Offerings by Date Created')
plt.xlabel('Date')
plt.ylabel('Job Offerings')
plt.show()
```




Overall a majority of job postings were made Thursday, with Monday and Saturday seeing the fewest. This is unexpected, as it would be thought that most postings are made during the working week, yet Sunday saw more postings than Monday.

Visualisation of the number of job offerings created by each company (top 30 only)

[50]:
```
date_counts = metadataDf['Company'].value_counts().nlargest(30).
↪sort_values(ascending=False)
date_counts.plot(kind='bar', figsize=(10, 6))
plt.title('Job Offerings by Company (Top 30)')
plt.xlabel('Company')
plt.ylabel('Job Offerings')
plt.show()
```




The top 6 companies by job postings are all recruitment agencies. There is a large difference between the number of job postings made by Jora Local and the second largest job poster, one potential explaination is that Jora Local is app based and provides labour for hospitality industries, lowering the barrier to entry compared to the types of labour the other traditional recruitment agencies attempt to acquire

Distribution of job types within classifications

[240]:
```
import plotly.express as px
import ipywidgets as widgets
from IPython.display import display

classification_dropdown = widgets.Dropdown(
    options=metadataDf['Classification'].unique(),
    description='Classification:'
)

def update_pie_chart(classification):
    job_type_count = metadataDf[metadataDf['Classification'] ==␣
 ↪classification]['JobType'].value_counts()
    fig = px.pie(names=job_type_count.index, values=job_type_count.values,␣
 ↪title='Distribution of Job Types for Jobs in ' + classification)
    fig.show()

widgets.interactive(update_pie_chart, classification=classification_dropdown)
```

[240]:
```
interactive(children=(Dropdown(description='Classification:', options=('Retail &
Consumer Products', 'Call Cen…
```

# 5 5 Study the market by locations

## 5.1 5.1 Market size in each city

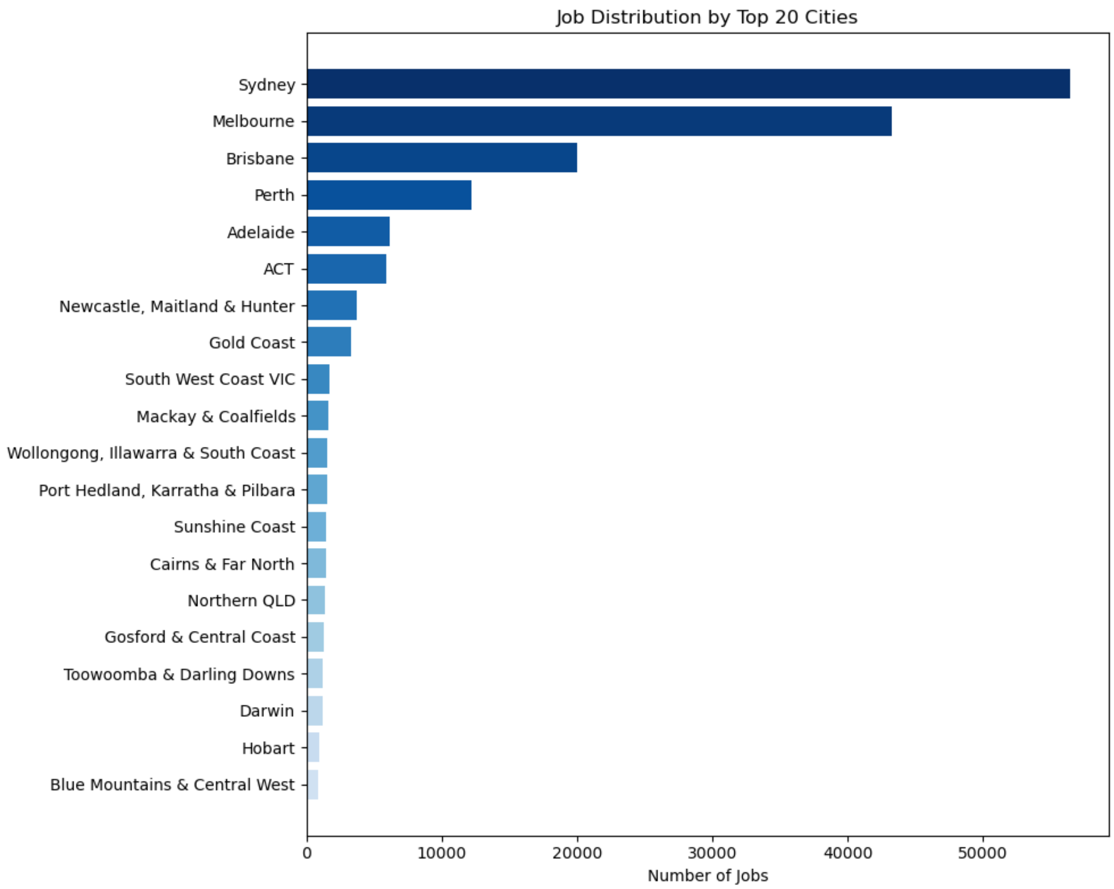
[32]:
```
locationDf = df[df['Location'].notna()]
locationDf = locationDf.groupby('Location', observed=True).count()
locationDf['Location'] = locationDf.index
locationDf['Count'] = locationDf['Id']
```

### 5.1.1 5.1.1 Get top 20 big cities and use bar chart to display market size in each city

[33]:
```
top_20_cities = locationDf.sort_values(by='Count', ascending=False).head(20)
top_20_cities = top_20_cities[::-1]
```

[39]:
```
plt.figure(figsize=(10, 8))
plt.barh(
    top_20_cities['Location'],
    top_20_cities['Count'],
    color=plt.cm.Blues(np.linspace(0.2, 1, len(top_20_cities))),
)
plt.xlabel('Number of Jobs')
plt.title('Job Distribution by Top 20 Cities')
```

```
plt.tight_layout()
plt.show()
```

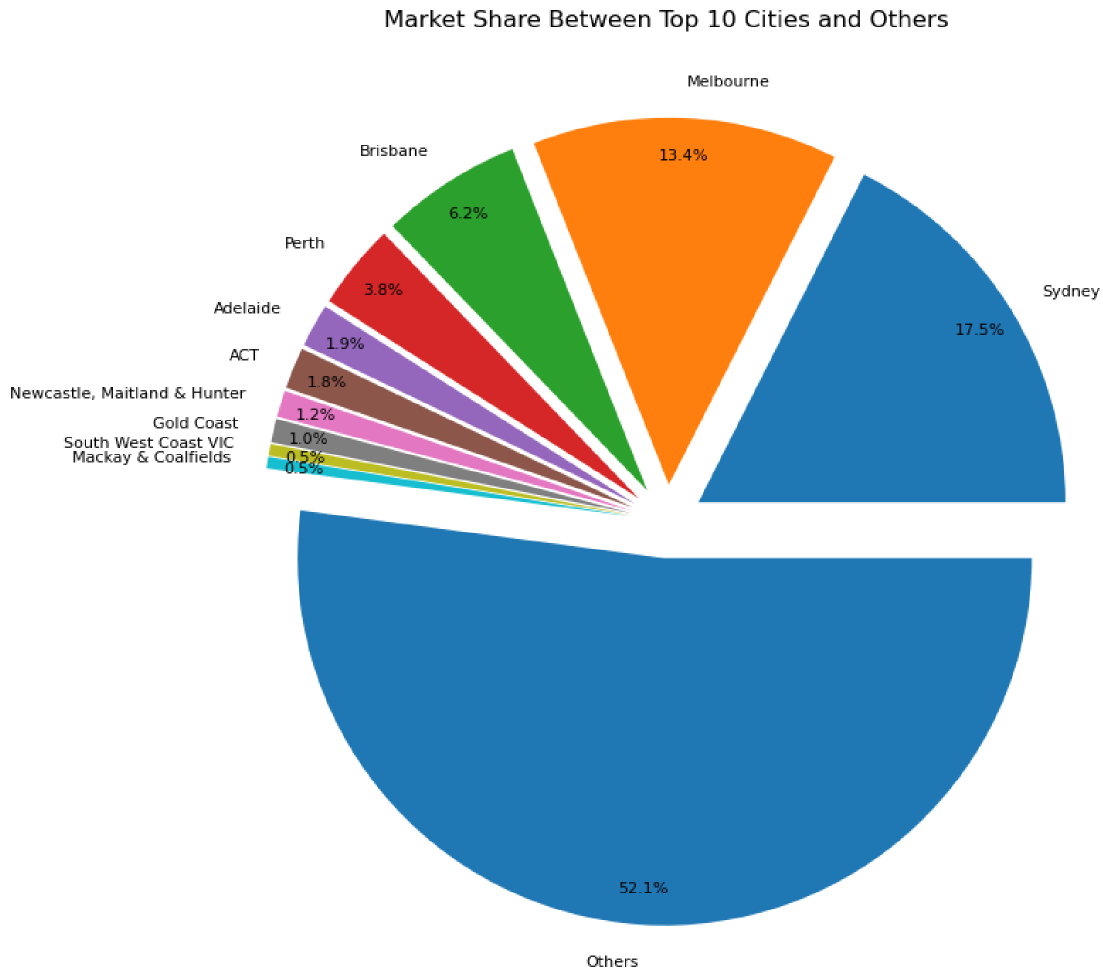



### 5.1.2 5.1.2 Visualise the market share between the cities

[40]:
```
#Get top 10 big cities
locationTop10Df = locationDf.nlargest(10, 'Count').copy()
others_count = locationDf['Count'][10:].sum()

new_row = pd.DataFrame(data={
    'Location': ['Others'],
    'Count': [others_count]
})
locationTop10Df = pd.concat([locationTop10Df, new_row], ignore_index=True)
locationTop10Df.index = locationTop10Df['Location']
```

[41]:
```
explode = (0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1)
textprops = {'fontsize': 8, 'rotation': 0}
```

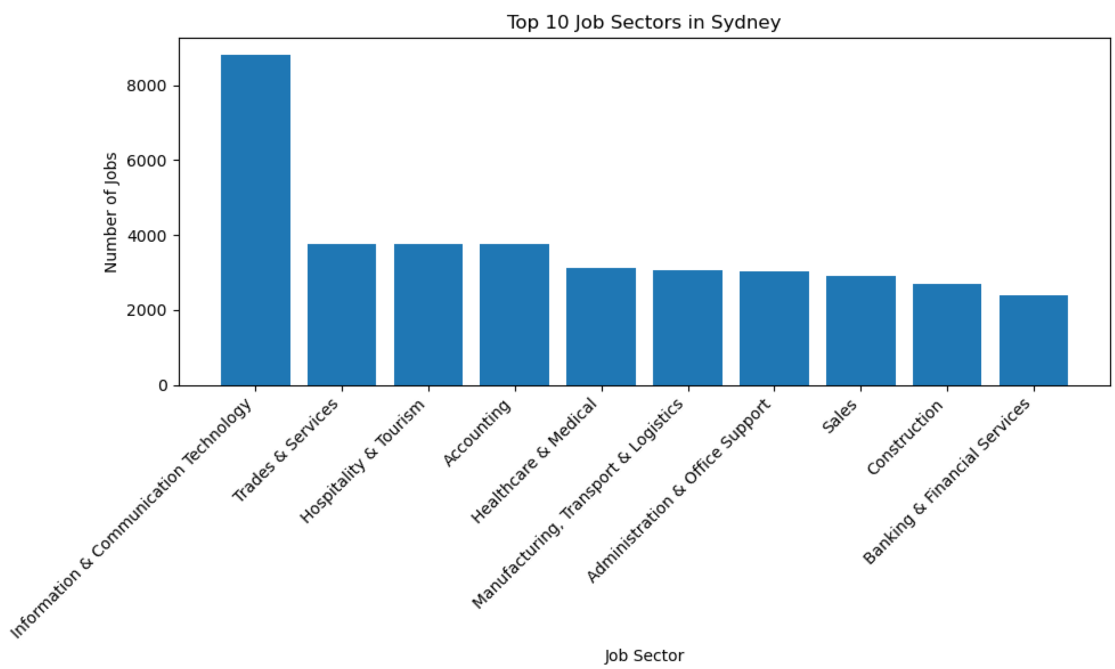
[42]:
```
plt.figure(figsize=(8, 8))
plt.pie(
    locationTop10Df['Count'],
    labels=locationTop10Df['Location'],
    autopct='%1.1f%%',
    explode=explode,
    textprops=textprops,
    pctdistance=0.9,
)
plt.axis('equal')
plt.title('Market Share Between Top 10 Cities and Others', pad=30)
plt.show()
```

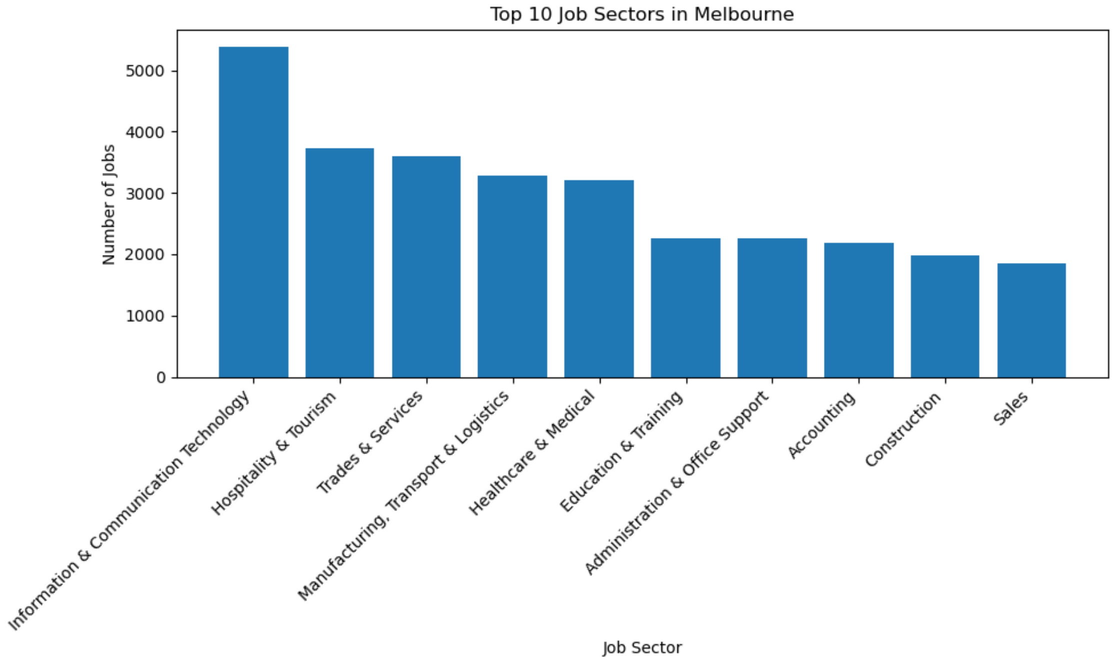


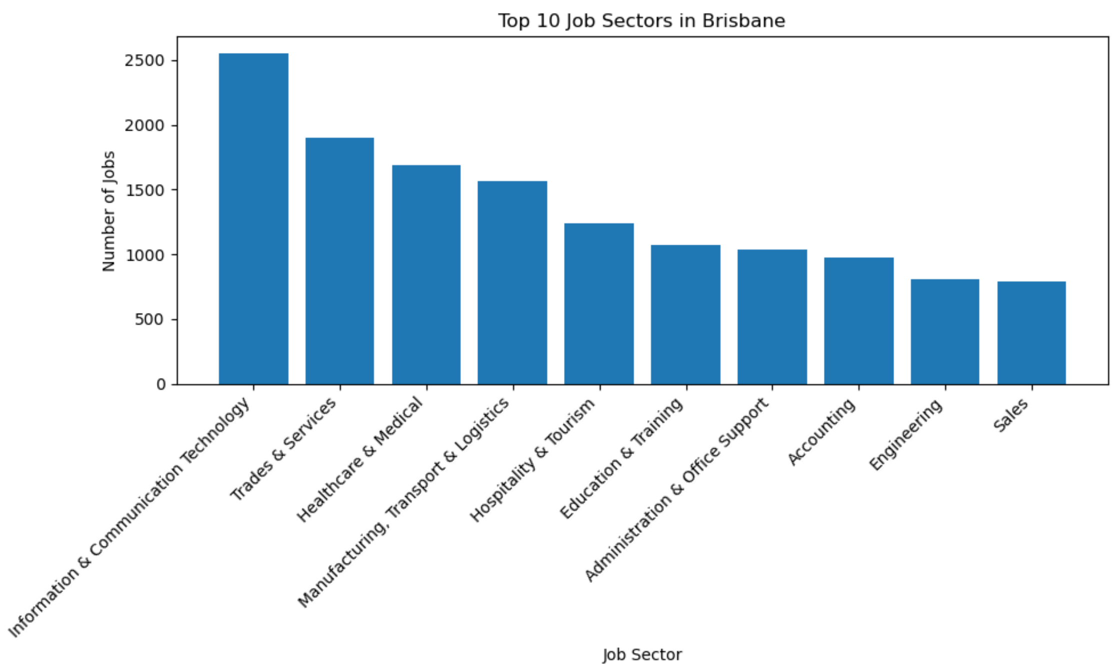
## 5.2 5.2 Top job sectors and sub-sectors in each city.

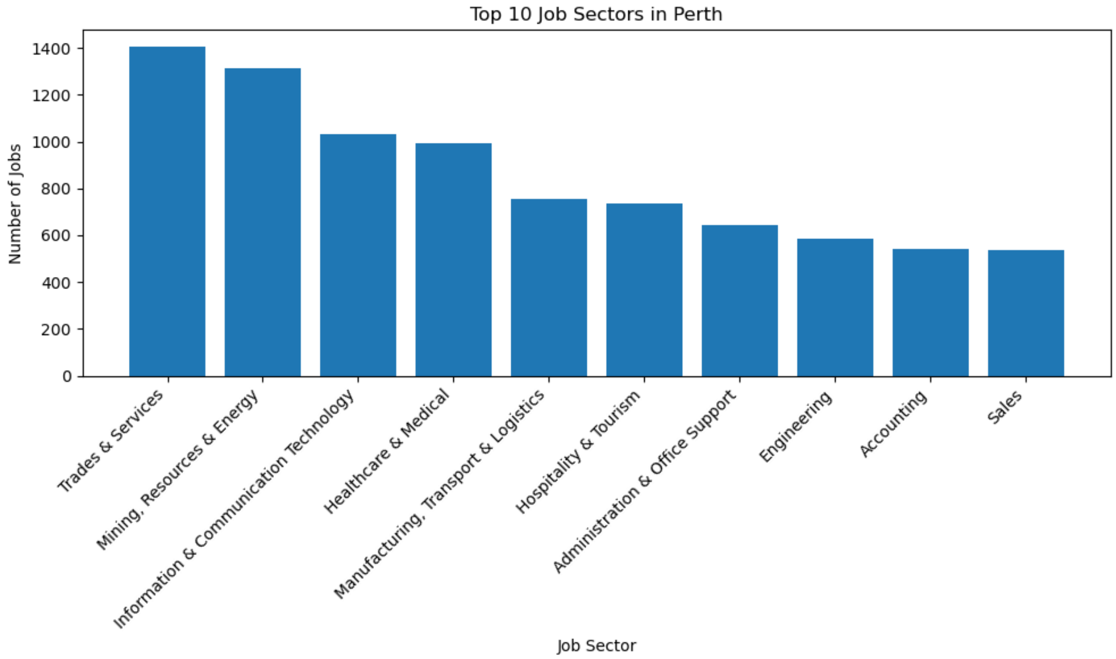
[44]:
```
city_counts = df['Location'].value_counts().nlargest(5)

top_cities = city_counts.index.tolist()

df_top_cities = df[df['Location'].isin(top_cities)]

top_cities_sector_counts = df_top_cities.groupby(['Location',␣
↪'Classification'], observed=True).size().reset_index(name='Count')
```

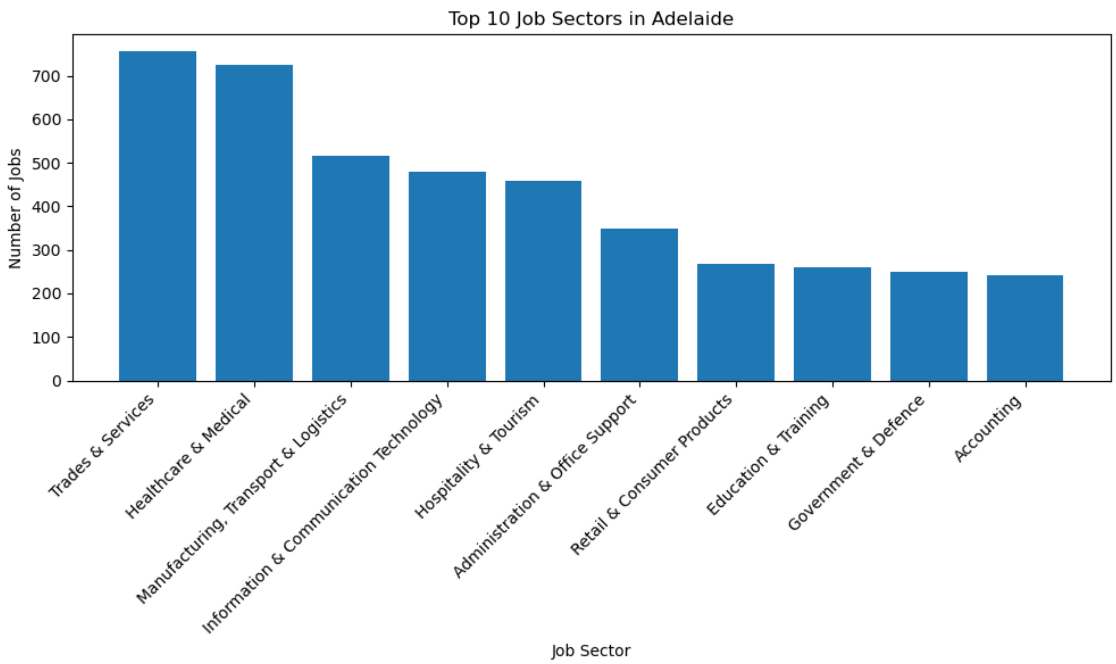
[45]:
```
for city in top_cities:
    city_data = top_cities_sector_counts[top_cities_sector_counts['Location']␣
↪== city].nlargest(10, 'Count')
    plt.figure(figsize=(10, 6))
    plt.bar(city_data['Classification'], city_data['Count'])
    plt.xlabel('Job Sector')
    plt.ylabel('Number of Jobs')
    plt.title('Top 10 Job Sectors in ' + city)
    plt.xticks(rotation=45, ha='right')
    plt.tight_layout()
    plt.show()
```




Top 10 Job Sectors in Melbourne
Number of Jobs
0
1000
2000
3000
4000
5000
Information & Communication Technology
Hospitality & Tourism
Trades & Services
Manufacturing, Transport & Logistics
Healthcare & Medical
Education & Training
Administration & Office Support
Accounting
Construction
Sales
Job Sector

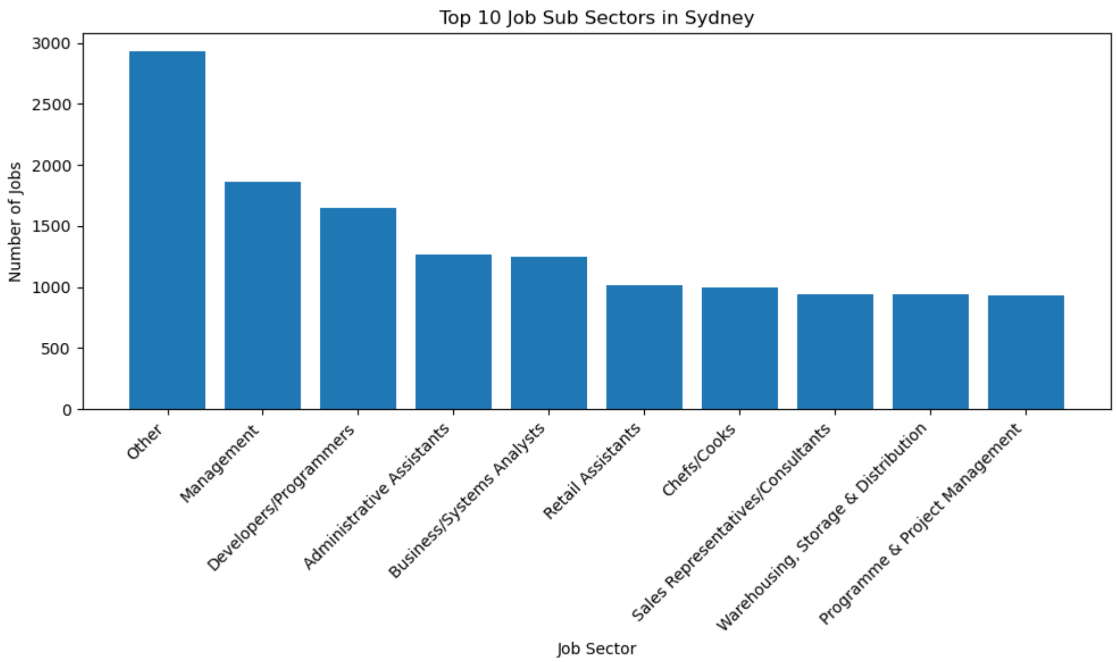

Top 10 Job Sectors in Brisbane
Number of Jobs
0
500
1000
1500
2000
2500
Information & Communication Technology
Trades & Services
Healthcare & Medical
Manufacturing, Transport & Logistics
Hospitality & Tourism
Education & Training
Administration & Office Support
Accounting
Engineering
Sales
Job Sector

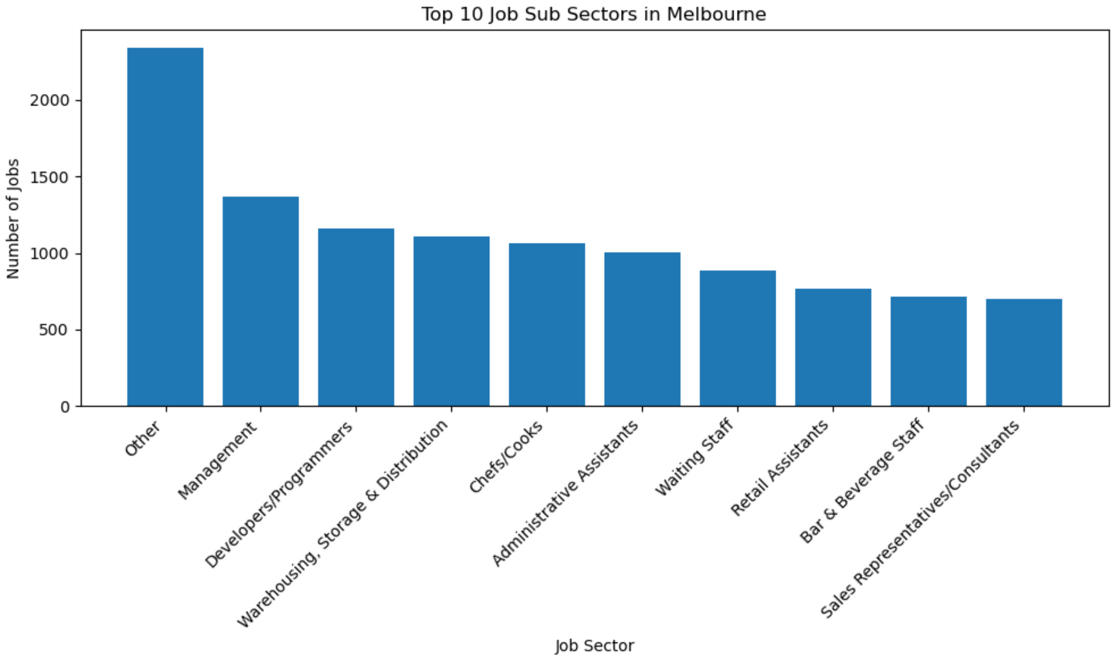


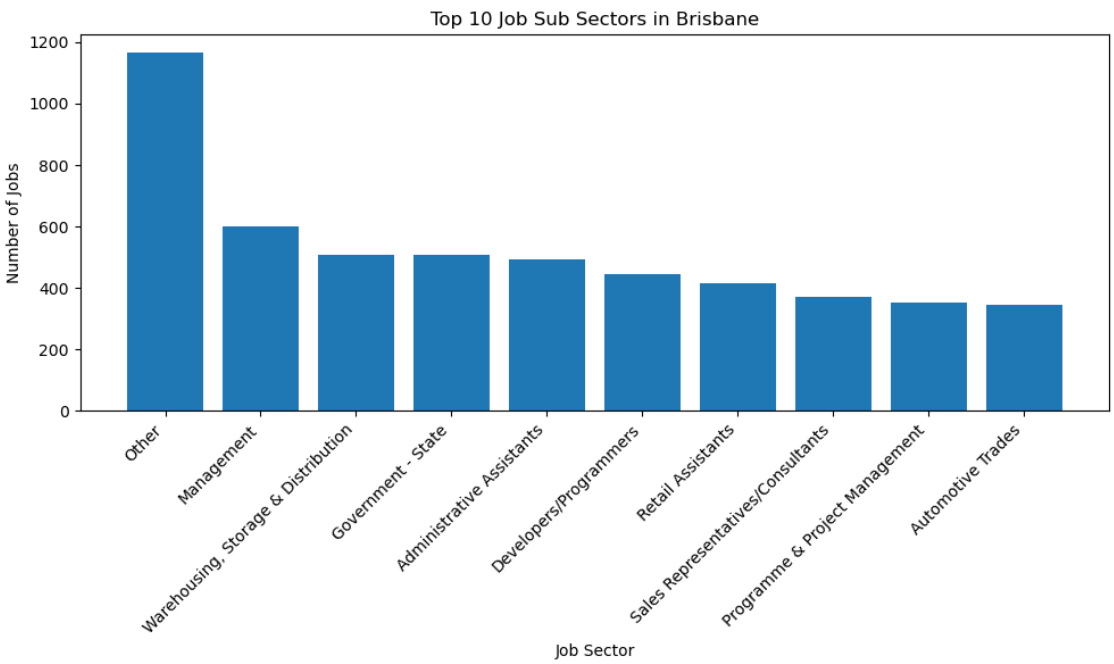



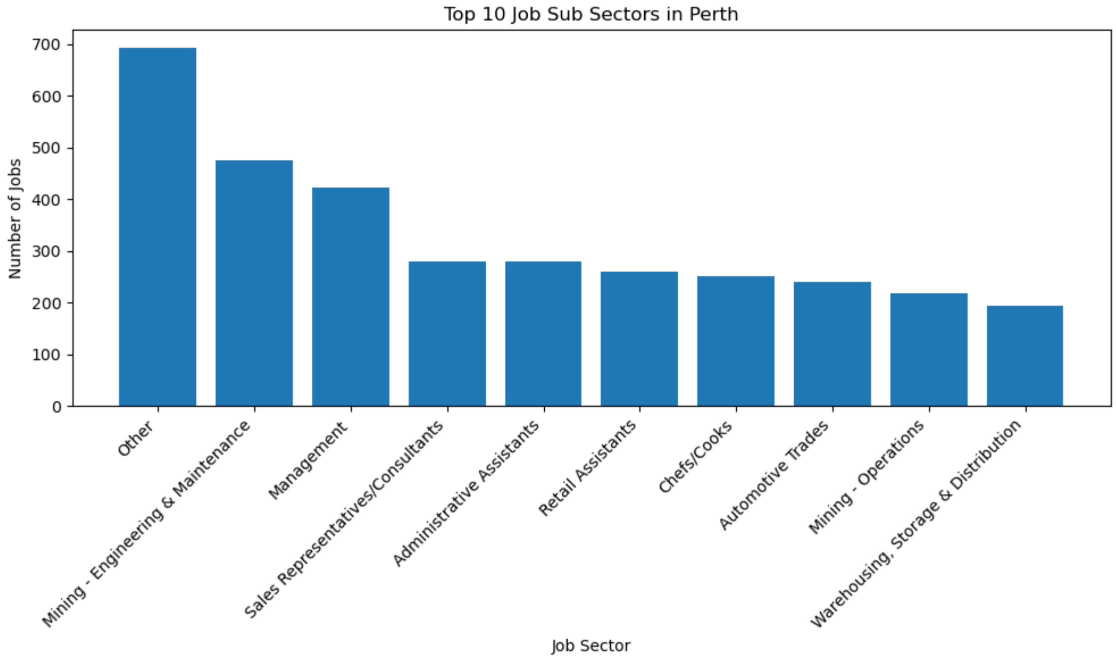

[47]:
```
top_cities_sub_sector_counts = df_top_cities.groupby(['Location',␣
↪'SubClassification'], observed=True).size().reset_index(name='Count')
```

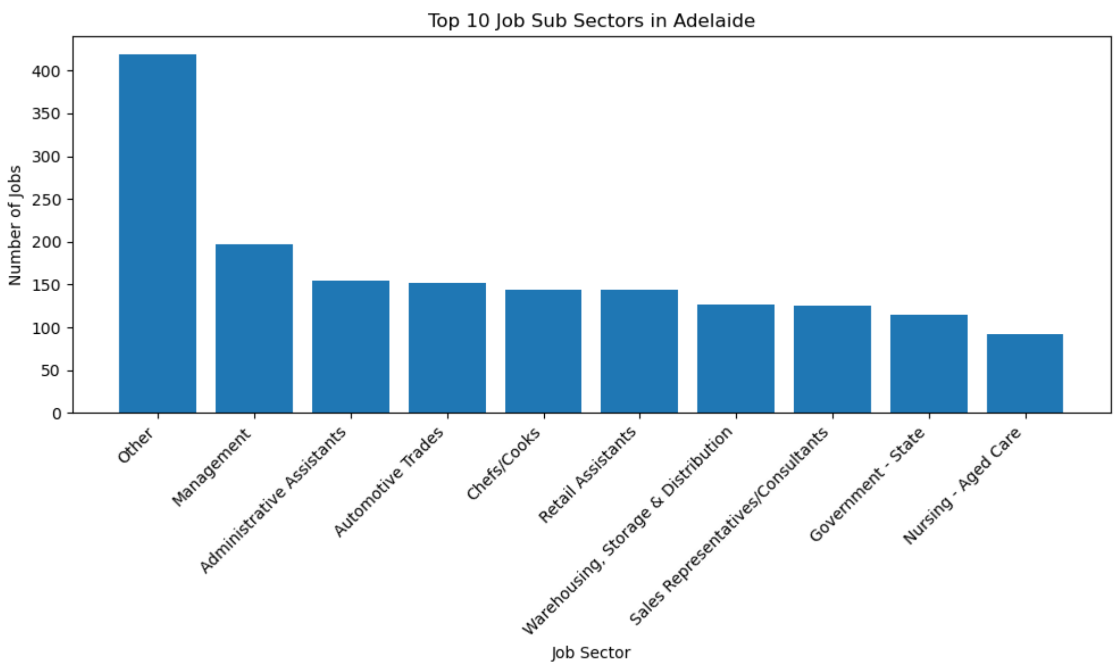
```
[48]: for city in top_cities:
          city_data =␣
      ↪top_cities_sub_sector_counts[top_cities_sub_sector_counts['Location'] ==␣
      ↪city].nlargest(10, 'Count')
          plt.figure(figsize=(10, 6))
          plt.bar(city_data['SubClassification'], city_data['Count'])
          plt.xlabel('Job Sector')
          plt.ylabel('Number of Jobs')
          plt.title('Top 10 Job Sub Sectors in ' + city)
          plt.xticks(rotation=45, ha='right')
          plt.tight_layout()
          plt.show()
```




Top 10 Job Sub Sectors in Melbourne
Number of Jobs
0
500
1000
1500
2000
Other
Management
Developers/Programmers
Warehousing, Storage & Distribution
Chefs/Cooks
Administrative Assistants
Waiting Staff
Retail Assistants
Bar & Beverage Staff
Sales Representatives/Consultants
Job Sector


Top 10 Job Sub Sectors in Brisbane
Number of Jobs
0
200
400
600
800
1000
1200
Other
Management
Warehousing, Storage & Distribution
Government - State
Administrative Assistants
Developers/Programmers
Retail Assistants
Sales Representatives/Consultants
Programme & Project Management
Automotive Trades
Job Sector

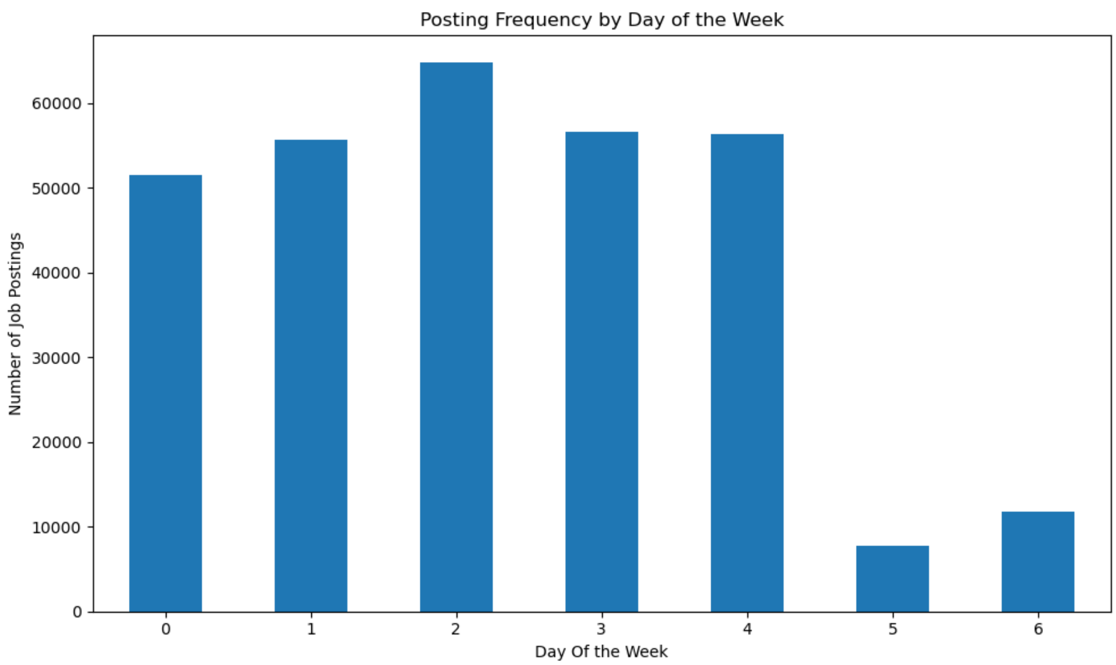

Top 10 Job Sub Sectors in Perth
Number of Jobs
0
100
200
300
400
500
600
700
Other
Mining - Engineering & Maintenance
Management
Sales Representatives/Consultants
Administrative Assistants
Retail Assistants
Chefs/Cooks
Automotive Trades
Mining - Operations
Warehousing, Storage & Distribution
Job Sector


Top 10 Job Sub Sectors in Adelaide
Number of Jobs
0
50
100
150
200
250
300
350
400
Other
Management
Administrative Assistants
Automotive Trades
Chefs/Cooks
Retail Assistants
Warehousing, Storage & Distribution
Sales Representatives/Consultants
Government - State
Nursing - Aged Care
Job Sector

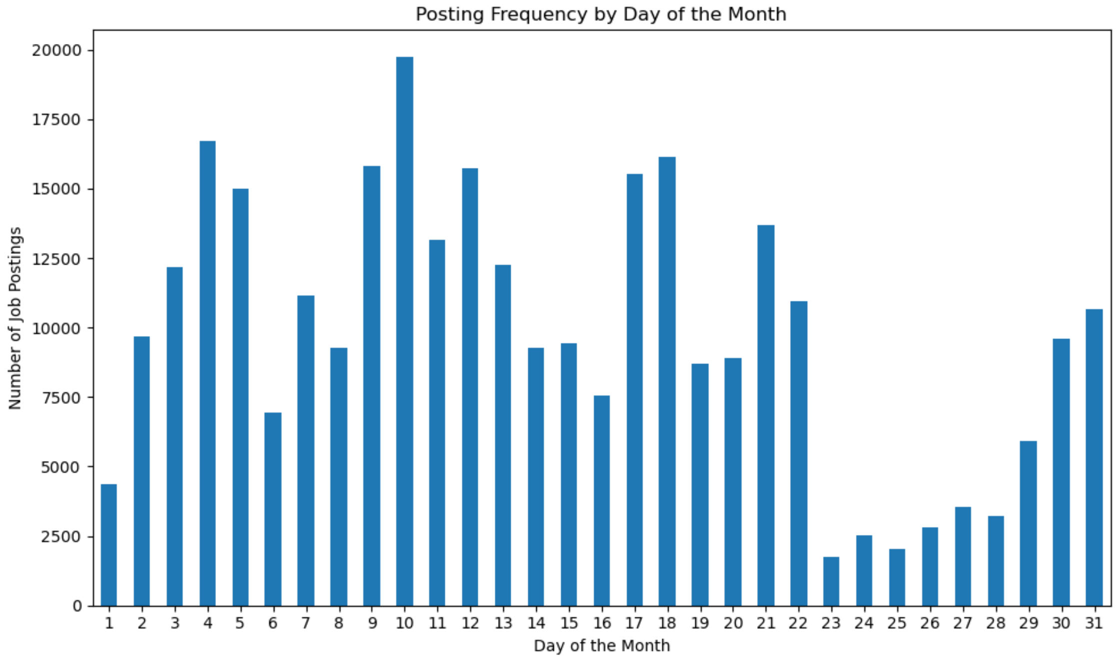
### 5.3 5.3 Job post patterns

[49]:
```
# Extract relevant time information
df['DayOfWeek'] = df['Date'].dt.dayofweek  # Monday=0, Sunday=6
df['DayOfMonth'] = df['Date'].dt.day
df['Month'] = df['Date'].dt.month
```

[50]:
```
# Plot posting frequency by day of the week
plt.figure(figsize=(10, 6))
df.groupby('DayOfWeek').size().plot(kind='bar')
plt.xlabel('Day Of the Week')
plt.ylabel('Number of Job Postings')
plt.title('Posting Frequency by Day of the Week')
plt.xticks(rotation=0)
plt.tight_layout()
plt.show()
```

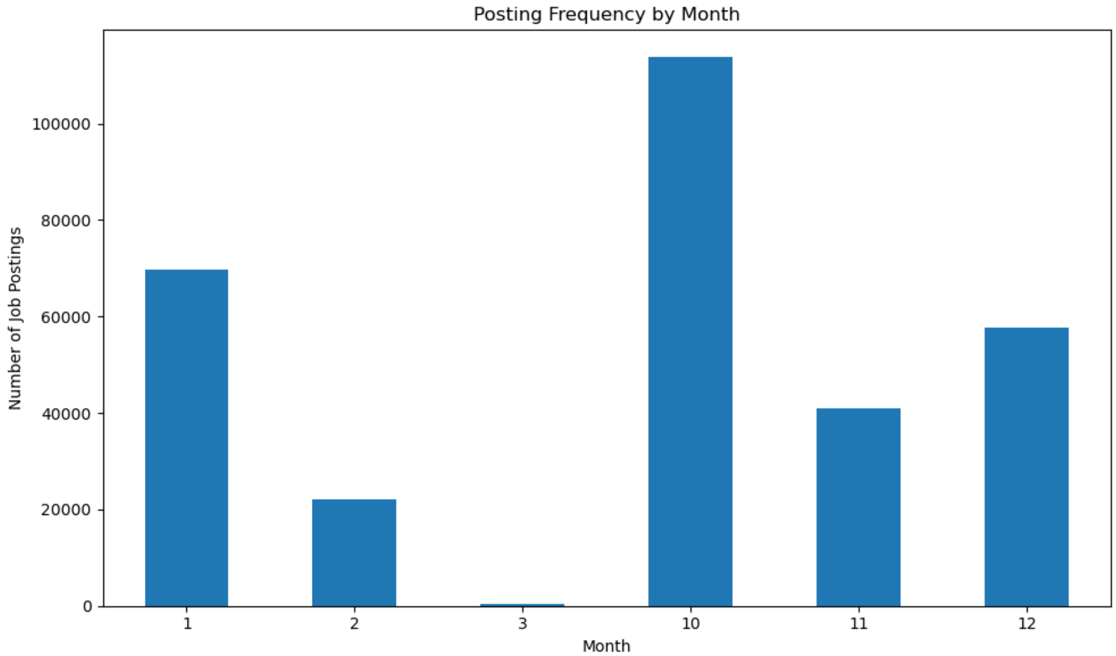



[51]:
```
# Plot posting frequency by day of the month
plt.figure(figsize=(10, 6))
df.groupby('DayOfMonth').size().plot(kind='bar')
plt.xlabel('Day of the Month')
plt.ylabel('Number of Job Postings')
plt.title('Posting Frequency by Day of the Month')
plt.xticks(rotation=0)
plt.tight_layout()
plt.show()
```




[52]:

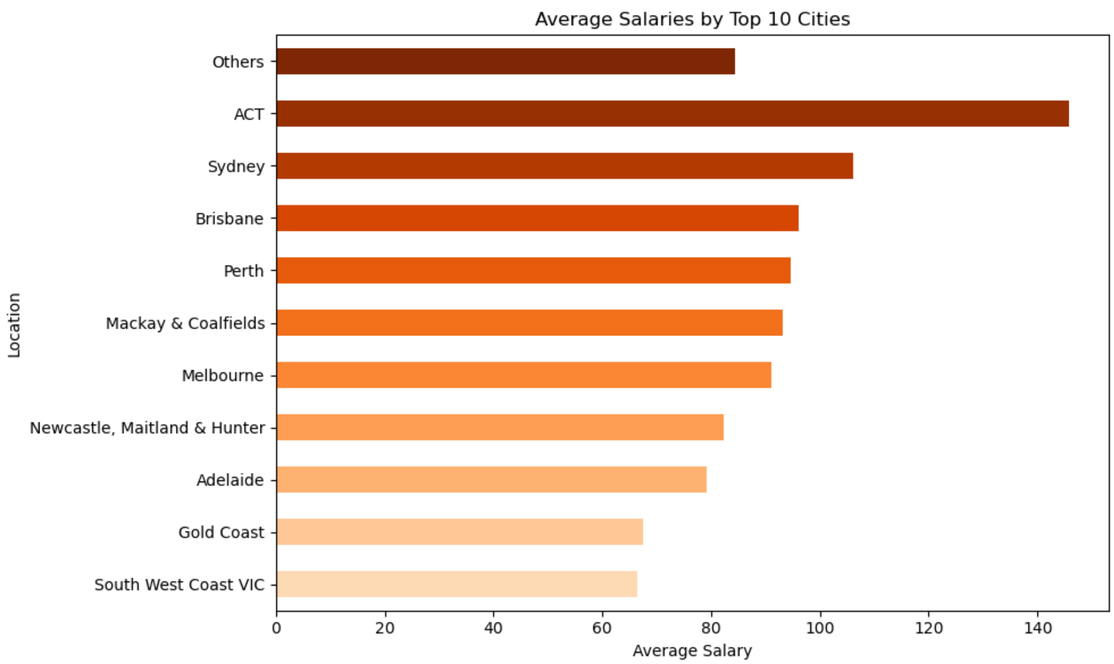
```
# Plot posting frequency by month
plt.figure(figsize=(10, 6))
df.groupby('Month').size().plot(kind='bar')
plt.xlabel('Month')
plt.ylabel('Number of Job Postings')
plt.title('Posting Frequency by Month')
plt.xticks(rotation=0)
plt.tight_layout()
plt.show()
```




## 5.4 5.4 Visualise the salary distribution in top biggest cities

[53]:
```
top_10_cities = locationDf.nlargest(10, 'Count')['Location']

top_10_cities_df = df[df['Location'].isin(top_10_cities)]
average_salary_by_city = top_10_cities_df.groupby('Location',␣
↪observed=True)['AverageSalary'].mean().sort_values()

other_cities_df = df[~df['Location'].isin(top_10_cities)]
average_salary_other_cities = other_cities_df['AverageSalary'].mean()
average_salary_by_city['Others'] = average_salary_other_cities
```

[54]:
```
plt.figure(figsize=(10, 6))
average_salary_by_city.plot(
    kind='barh',
    color=plt.cm.Oranges(np.linspace(0.2, 1, len(average_salary_by_city))),
)
plt.title('Average Salaries by Top 10 Cities')
plt.xlabel('Average Salary')
plt.tight_layout()
plt.show()
```




# 6 6 Study by sectors

Which sectors keep the highest market share?

[190]:

```
from matplotlib import cm
```

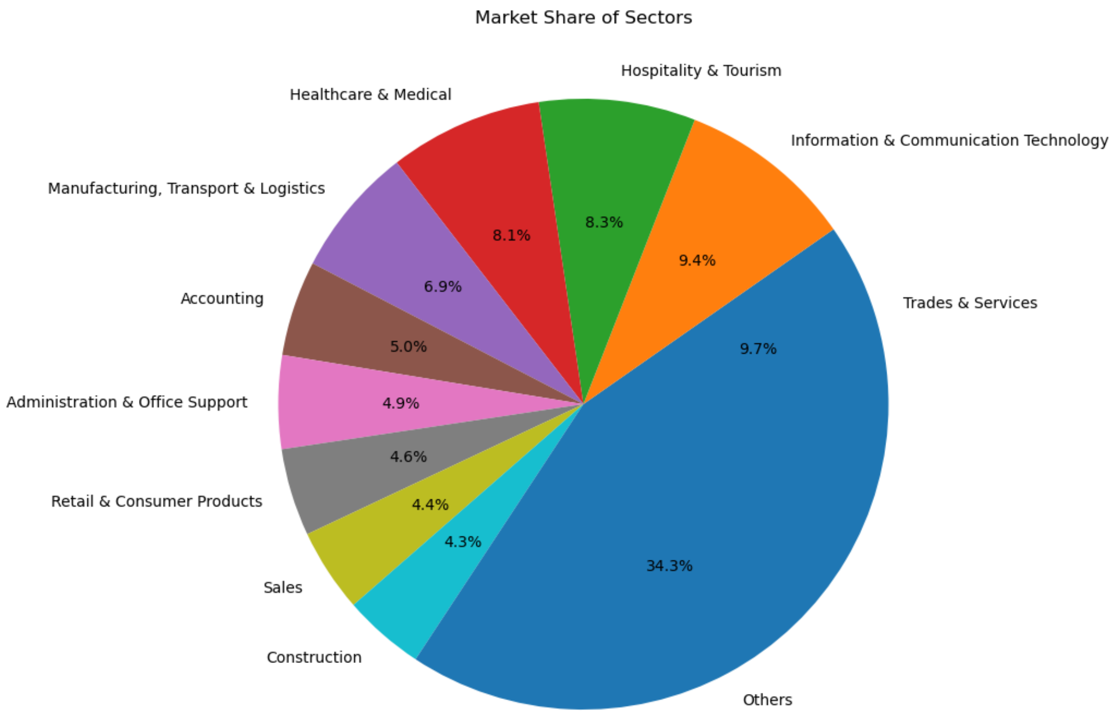
[7]:

```
df=pd.read_csv('preprocessed_data.csv')
df_pie = df['Classification'].groupby(df['Classification'])

df_pie = df_pie.count().sort_values(ascending=False).to_frame()
df_pie['Count'] = df_pie['Classification']
df_pie['Classification'] = df_pie.index
df2 = df_pie[:10].copy()
new_row = pd.DataFrame(data = {
    'Classification' : ['Others'],
    'Count' : [df_pie['Count'][10:].sum()]
})

df2 = pd.concat([df2, new_row])
df2.index = df2['Classification']
```

[9]:

```
plt.figure(figsize=(10, 8))
plt.pie(
    df2['Count'],
    labels=df2['Classification'],
```

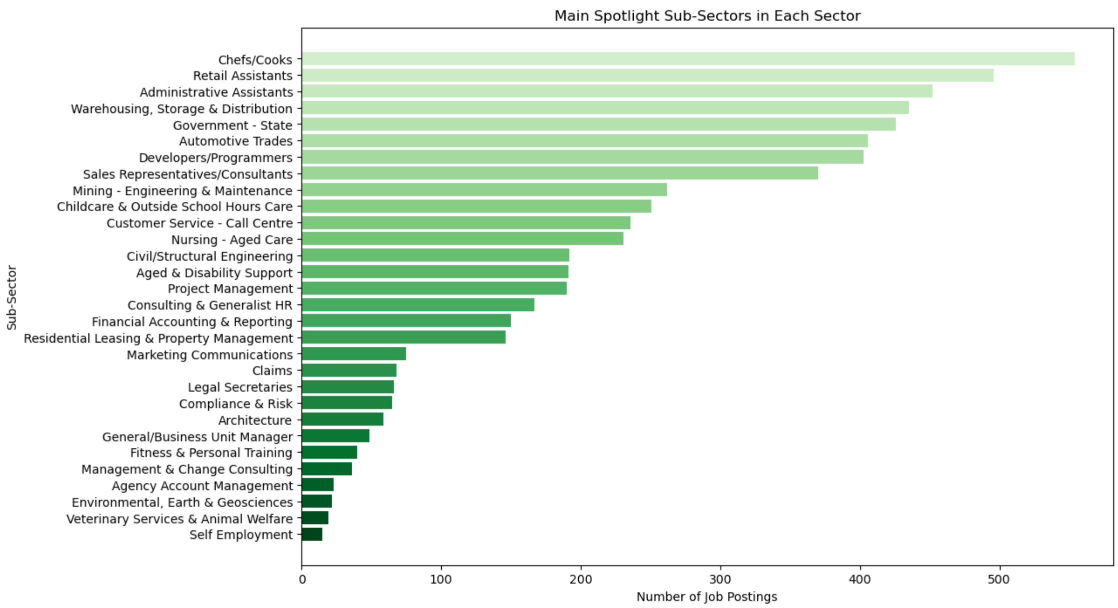
```
    autopct='%1.1f%%',
)
plt.title('Market Share of Sectors', pad=30)
plt.axis('equal')
plt.show()
```




In each sector, which sub-sectors are the main spotlights?

[11]:
```
sector_subsector_counts = df.groupby(['Classification', 'SubClassification']).
    ↪size().reset_index(name='Count')

main_spotlight_subsectors = sector_subsector_counts.groupby('Classification').
    ↪apply(lambda x: x.loc[x['Count'].idxmax()])

main_spotlight_subsectors = main_spotlight_subsectors.sort_values(by='Count',␣
    ↪ascending=False)

plt.figure(figsize=(12, 8))
plt.barh(
    main_spotlight_subsectors['SubClassification'],
    main_spotlight_subsectors['Count'],
    color=plt.cm.Greens(np.linspace(0.2, 1, len(main_spotlight_subsectors))),
)
```

```
plt.title('Main Spotlight Sub-Sectors in Each Sector')
plt.xlabel('Number of Job Postings')
plt.ylabel('Sub-Sector')
plt.gca().invert_yaxis()
plt.show()
```

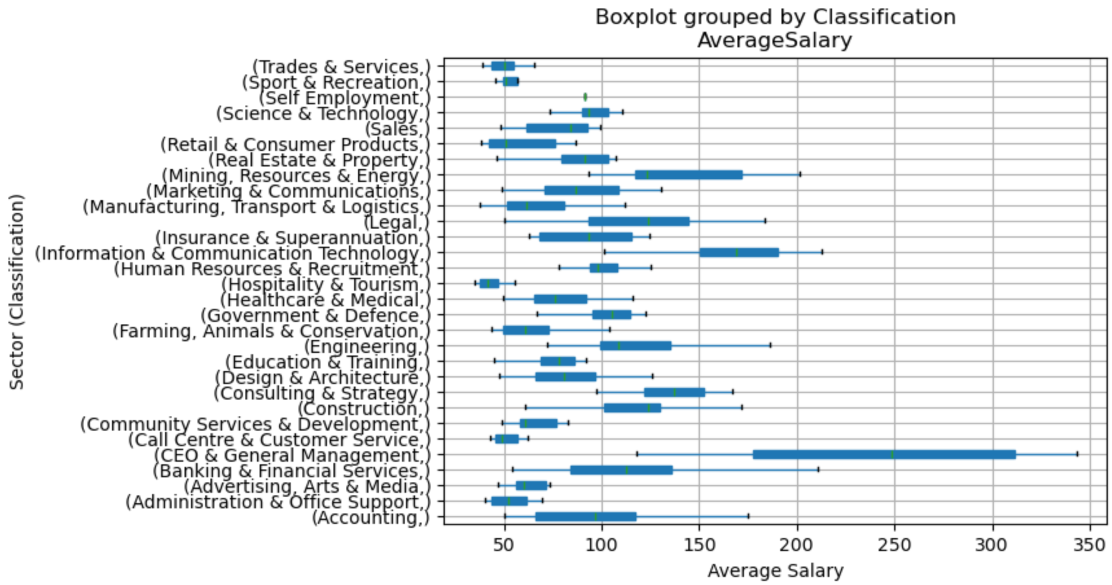



What is the salary range for each sector/sub-sector? Can you compare the salary range between sectors/subsectors

[264]:
```
sector_subsector_average_salary = df.groupby(['Classification',␣
↪'SubClassification'])['AverageSalary'].mean().reset_index()
```

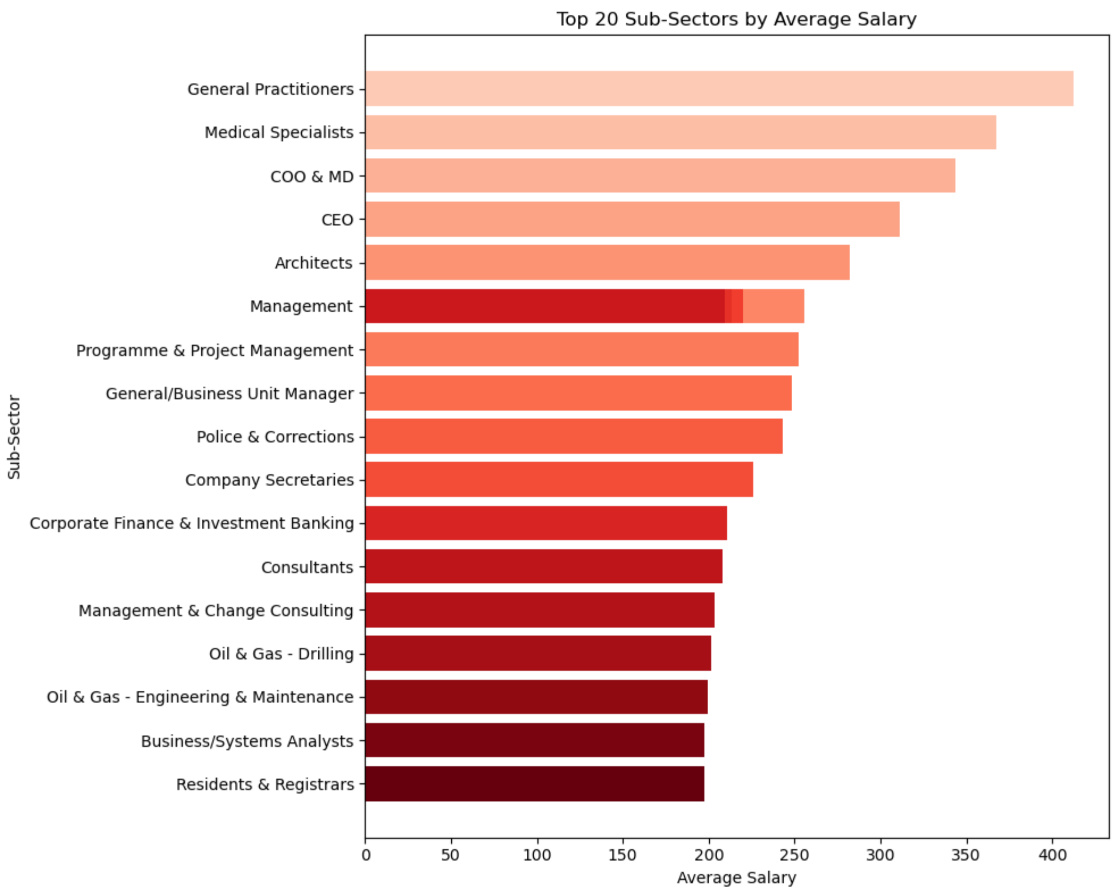
[265]:
```
plt.figure(figsize=(12, 8))
sector_subsector_average_salary.boxplot(
    by=['Classification'],
    column='AverageSalary',
    vert=False,
    patch_artist=True,
    showfliers=False,
)
plt.xlabel('Average Salary')
plt.ylabel('Sector (Classification)')
plt.show()
```

<Figure size 1200x800 with 0 Axes>



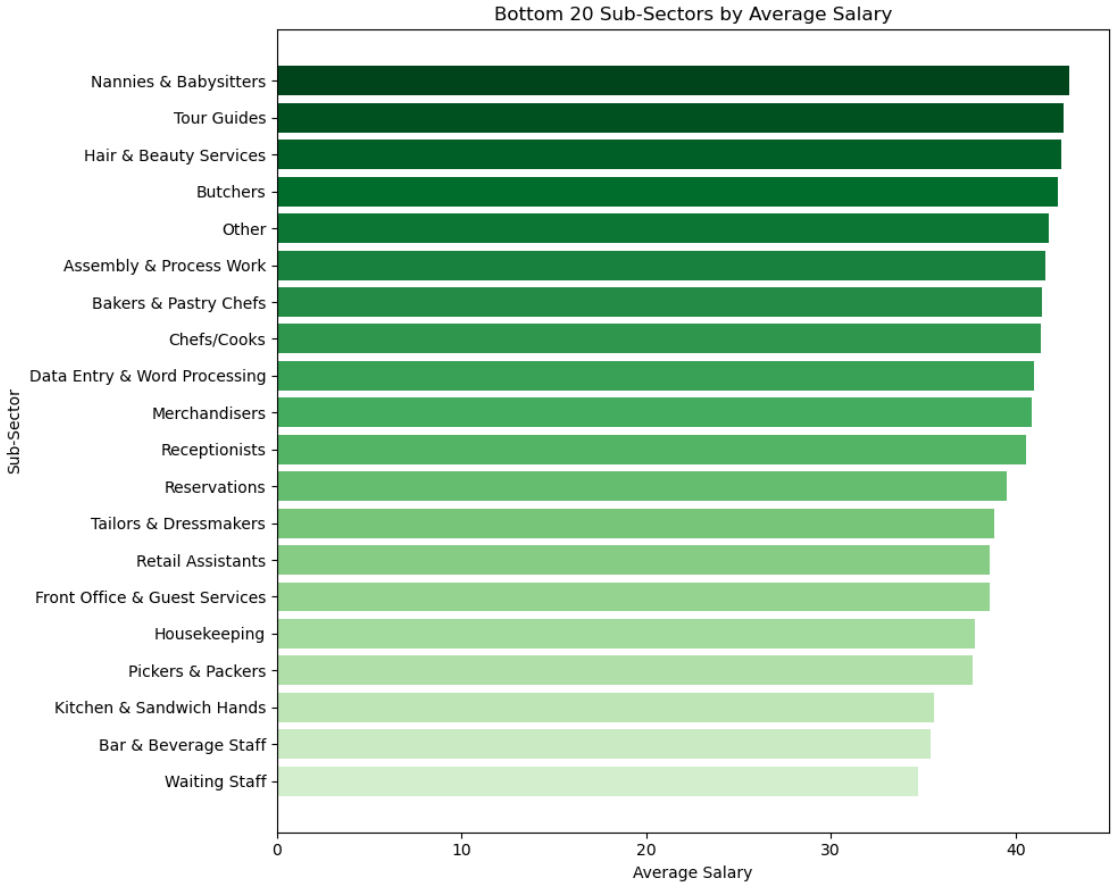

[268]:
```
top_20_subsectors = sector_subsector_average_salary.nlargest(20,␣
↪'AverageSalary')
```

[287]:
```
plt.figure(figsize=(10, 8))
plt.barh(
    top_20_subsectors['SubClassification'],
    top_20_subsectors['AverageSalary'],
    color=plt.cm.Reds(np.linspace(0.2, 1, len(top_20_subsectors))),
)
plt.xlabel('Average Salary')
plt.ylabel('Sub-Sector')
plt.title('Top 20 Sub-Sectors by Average Salary')
plt.gca().invert_yaxis()
plt.tight_layout()
plt.show()
```




[281]:
```
bottom_20_subsectors = sector_subsector_average_salary.nsmallest(20,␣
    ↪'AverageSalary')
```

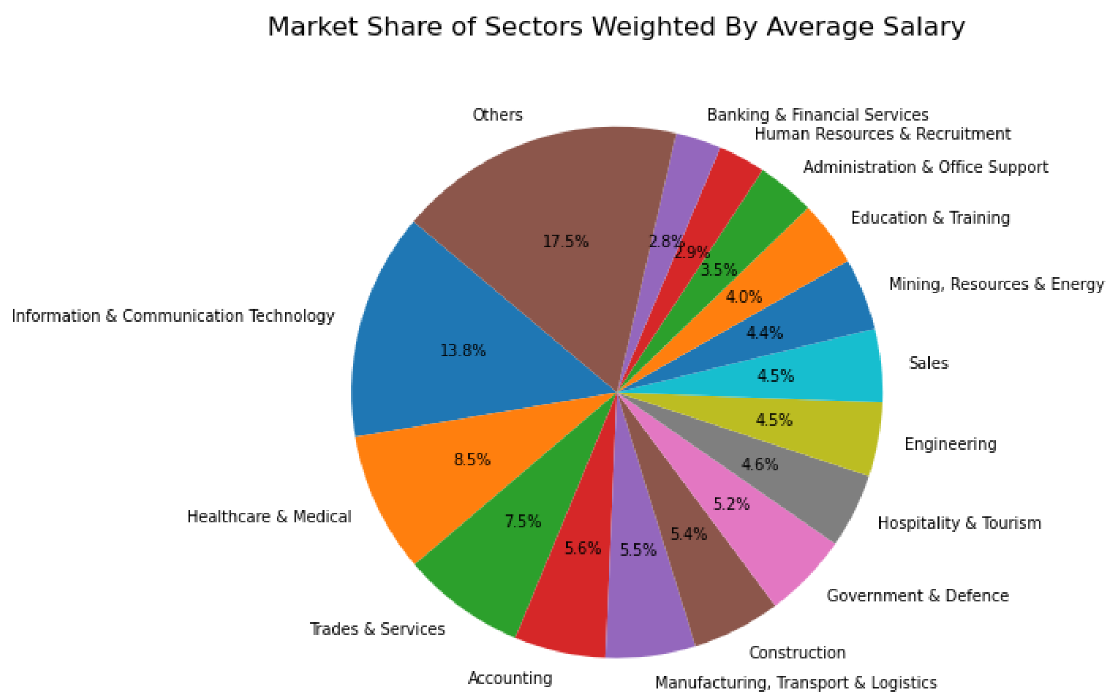
[288]:
```
plt.figure(figsize=(10, 8))
plt.barh(
    bottom_20_subsectors['SubClassification'],
    bottom_20_subsectors['AverageSalary'],
    color=plt.cm.Greens(np.linspace(0.2, 1, len(bottom_20_subsectors))),
)
plt.xlabel('Average Salary')
plt.ylabel('Sub-Sector')
plt.title('Bottom 20 Sub-Sectors by Average Salary')
plt.tight_layout()
plt.show()
```




Market share of jobs weighted by their average salary (a sector with double the salaries has double the value)

[229]:
```
df=pd.read_csv('preprocessed_data.csv')
sector_subsector_average_salary = df.
↪groupby(['Classification'])['AverageSalary'].mean().reset_index()
df_pie = df['Classification'].groupby(df['Classification'])
df_pie = df_pie.count().to_frame()
df_pie = df_pie.rename(columns={'Classification': 'Count'})
merged_df=pd.merge(df_pie, sector_subsector_average_salary, on='Classification')
result = merged_df['AverageSalary'].multiply(merged_df['Count'])
merged_df['WeightedMarketShare'] = result
merged_df=merged_df.sort_values(by='WeightedMarketShare', ascending=False)
merged_df2 = merged_df[:15].copy()
new_row = pd.DataFrame(data = {
    'Classification' : ['Others'],
    'WeightedMarketShare' : [merged_df['WeightedMarketShare'][15:].sum()]
})
```

```
merged_df2 = pd.concat([merged_df2, new_row])
plt.pie(
    merged_df2['WeightedMarketShare'],
    labels=merged_df2['Classification'],
    autopct='%1.1f%%',
    textprops={'fontsize': 7},
    startangle=140
)
plt.title('Market Share of Sectors Weighted By Average Salary', pad=30)
plt.axis('equal')
plt.show()
```




What is the trending of the market, i.e. if a high school student asks you which subject should he/she learn in the university (to guarantee a job in the future) ,what is your advice?

Information & Communication Technology is currently trending in the market, with a high average salary and a large market share of jobs, and overall is the prime recommendation for a student deciding which subject they should study in university. When considering salary and market share, healthcare and medical related jobs are also very good.

Can you detect which skills are required in each sector?

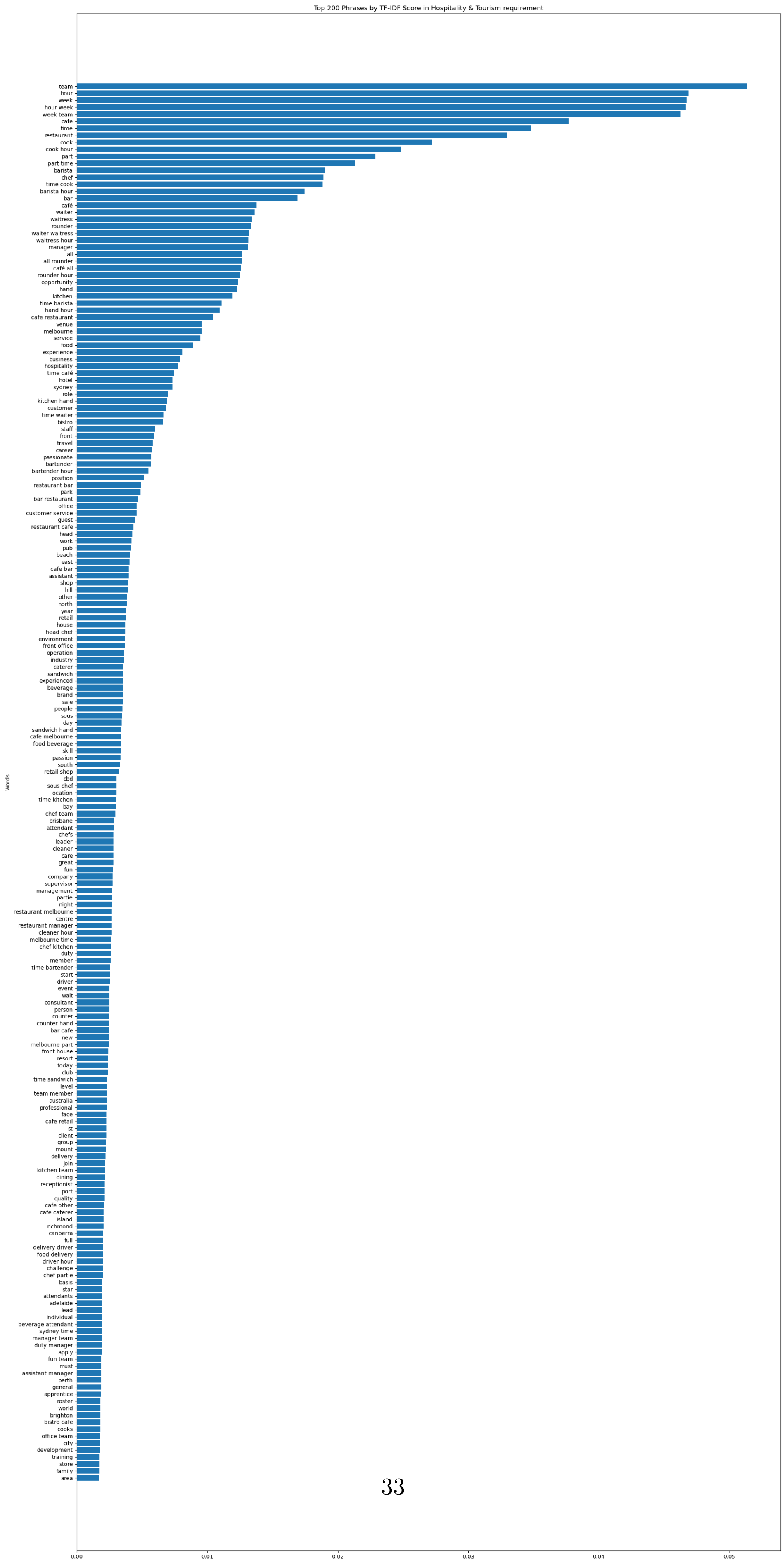
[107]:
```
import nltk
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
from nltk import pos_tag
```

```
[122]: #Skills are nouns
def tokenize_and_filter_nouns(text):
    tokens = nltk.word_tokenize(text)
    tagged_tokens = pos_tag(tokens)
    noun_tokens = [token for token, pos in tagged_tokens if pos.startswith('N')]
    return noun_tokens

def preprocessing(classification, csv=None):
    if csv!=None:
        df = pd.read_csv(csv)
    else:
        df=pd.read_csv("preprocessed_data.csv")
        df.loc[df['Classification']==classification, 'Tokenized'] =␣
↪df['Requirement'].apply(tokenize_and_filter_nouns)
        df.dropna(subset=['Tokenized'], inplace=True)
        # Remove punctuation
        df['Tokenized']=df['Tokenized'].apply(lambda tokens: [token for token␣
↪in tokens if token not in string.punctuation])
        # Remove stop words
        df['Tokenized']=df['Tokenized'].apply(lambda tokens: [token for token␣
↪in tokens if token.lower() not in nltk.corpus.stopwords.words('english')])
        # Lemmatization
        wnl=WordNetLemmatizer()
        df['Tokenized']=df['Tokenized'].apply(lambda tokens: [wnl.
↪lemmatize(token) for token in tokens])
        df['Tokenized']=df['Tokenized'].apply(lambda tokens: ' '.join(tokens))
        df.to_csv(classification+"_tokenized_data.csv")
    return df

def tfidf_transform_and_display(df, classification, phrase_number):
    tfidf_vectorizer = TfidfVectorizer(ngram_range=(1,2))
    tfidf_matrix = tfidf_vectorizer.fit_transform(df['Tokenized'])
    avg_tfidf_scores = tfidf_matrix.mean(axis=0)
    scores = avg_tfidf_scores.A1
    terms = tfidf_vectorizer.get_feature_names_out()
    tfidf_df = pd.DataFrame({'term': terms, 'tfidf': scores})
    tfidf_df = tfidf_df.sort_values(by='tfidf')
    top_words = tfidf_df.nlargest(phrase_number, 'tfidf').iloc[::-1]
    # Create a bar graph of the top words and their scores
    plt.figure(figsize=(20, phrase_number/5))
    plt.barh(top_words['term'], top_words['tfidf'])
    plt.xlabel('TF-IDF Score')
    plt.ylabel('Words')
    plt.title('Top '+str(phrase_number)+' Phrases by TF-IDF Score in␣
↪'+classification+' requirement')
```

```
    plt.tight_layout()
    plt.show()

def skill_identifier(classification, phrase_number, csv=None):
    df=preprocessing(classification, csv)
    tfidf_transform_and_display(df, classification, phrase_number)
```

[124]:
```
#Can use to identify skills for all classifications, only demonstrating one␣
↪example for sake of computational time
skill_identifier("Hospitality & Tourism", 200)
```


Top 200 Phrases by TF-IDF Score in Hospitality & Tourism requirement
team
hour
week
hour week
week team
cafe
time
restaurant
cook
cook hour
part
part time
barista
chef
time cook
barista hour
bar
café
waiter
waitress
rounder
waiter waitress
waitress hour
manager
all
all rounder
café all
rounder hour
opportunity
hand
kitchen
time barista
hand hour
cafe restaurant
venue
melbourne
service
food
experience
business
hospitality
time café
hotel
sydney
role
kitchen hand
customer
time waiter
bistro
staff
front
travel
career
passionate
bartender
bartender hour
position
restaurant bar
park
bar restaurant
office
customer service
guest
restaurant cafe
head
work
pub
beach
east
cafe bar
assistant
shop
hill
other
north
year
retail
house
head chef
environment
front office
operation
industry
caterer
sandwich
experienced
beverage
brand
sale
people
sous
day
sandwich hand
cafe melbourne
food beverage
skill
passion
south
retail shop
cbd
sous chef
location
time kitchen
bay
chef team
brisbane
attendant
chefs
leader
cleaner
care
great
fun
company
supervisor
management
partie
night
restaurant melbourne
centre
restaurant manager
cleaner hour
melbourne time
chef kitchen
duty
member
time bartender
start
driver
event
wait
consultant
person
counter
counter hand
bar cafe
new
melbourne part
front house
resort
today
club
time sandwich
level
team member
australia
professional
face
cafe retail
st
client
group
mount
delivery
join
kitchen team
dining
receptionist
port
quality
cafe other
cafe caterer
island
richmond
canberra
full
delivery driver
food delivery
driver hour
challenge
chef partie
basis
star
attendants
adelaide
lead
individual
beverage attendant
sydney time
manager team
duty manager
apply
fun team
must
assistant manager
perth
general
apprentice
roster
world
brighton
bistro cafe
cooks
office team
city
development
training
store
family
area
Words
0.00
0.01
0.02
0.03
0.04
0.05
TF-IDF Score

# 7 7 Interactive Visualisation

[1]:
```
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.metrics import mean_absolute_error
from sklearn.ensemble import RandomForestRegressor

from lightgbm import LGBMRegressor

import plotly.express as px
import plotly.graph_objects as go
```

[2]:
```
# Load dataset
df = pd.read_csv('preprocessed_data.csv', low_memory=False)
```

[3]:
```
df.describe()
```

[3]:
```
         Unnamed: 0            Id   LowestSalary  HighestSalary  \
count  304312.000000  3.043120e+05  304312.000000  304312.000000
mean   152155.500000  3.779338e+07      66.563691     116.192651
std     87847.451896  2.988405e+05      50.994459     180.602380
min         0.000000  3.167109e+07       0.000000      30.000000
25%     76077.750000  3.750835e+07      30.000000      40.000000
50%    152155.500000  3.770907e+07      60.000000      70.000000
75%    228233.250000  3.801669e+07     100.000000     120.000000
max    304311.000000  3.856613e+07     200.000000     999.000000

       AverageSalary           Year          Month            Day  \
count  304312.000000  304312.000000  304312.000000  304312.000000
mean       91.378171    2018.302587       7.864652      13.961927
std       110.171184       0.459379       4.423799       8.150803
min        15.000000    2018.000000       1.000000       1.000000
25%        35.000000    2018.000000       2.000000       8.000000
50%        65.000000    2018.000000      10.000000      13.000000
75%       110.000000    2019.000000      11.000000      19.000000
```

```
max       599.500000    2019.000000      12.000000      31.000000

           DayOfWeek
count  304312.000000
mean        2.266828
std         1.601637
min         0.000000
25%         1.000000
50%         2.000000
75%         3.000000
max         6.000000
```

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 304312 entries, 0 to 304311
Data columns (total 19 columns):
 #   Column             Non-Null Count   Dtype
---  ------             --------------   -----
 0   Unnamed: 0         304312 non-null  int64
 1   Id                 304312 non-null  int64
 2   Title              304312 non-null  object
 3   Company            292858 non-null  object
 4   Date               304312 non-null  object
 5   Location           183064 non-null  object
 6   Area               113751 non-null  object
 7   Classification     183064 non-null  object
 8   SubClassification  183064 non-null  object
 9   Requirement        304307 non-null  object
 10  FullDescription    291001 non-null  object
 11  LowestSalary       304312 non-null  int64
 12  HighestSalary      304312 non-null  int64
 13  JobType            291076 non-null  object
 14  AverageSalary      304312 non-null  float64
 15  Year               304312 non-null  int64
 16  Month              304312 non-null  int64
 17  Day                304312 non-null  int64
 18  DayOfWeek          304312 non-null  int64
dtypes: float64(1), int64(8), object(10)
memory usage: 44.1+ MB
```

[5]:
```
def Regressor(X, y, categorical_features, n_estimators):
    # Preprocessing pipeline for categorical features
    categorical_transformer = Pipeline(steps=[
        ('imputer', SimpleImputer(strategy='constant', fill_value='missing')),
        ('onehot', OneHotEncoder(handle_unknown='ignore'))
    ])
```

```
    # Combining preprocessing
    preprocessor = ColumnTransformer(
        transformers=[
            ('cat', categorical_transformer, categorical_features)
        ])

    # Define the model
    model = Pipeline(steps=[
        ('preprocessor', preprocessor),
        ('regressor', LGBMRegressor(n_estimators=n_estimators, random_state=42,␣
↪n_jobs=-1))
    ])

    # Split the data
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

    # Train the model
    model.fit(X_train, y_train)

    # Predict and evaluate
    y_pred = model.predict(X_test)
    mae = mean_absolute_error(y_test, y_pred)

    mean_salary = y_test.mean()
    relative_mae = mae / mean_salary

    print(f'\nTrained on feature set: {categorical_features}')

    print(f'\nMean Salary: {mean_salary}')
    print(f'Mean Absolute Error: {mae}')
    print(f'Relative MAE: {relative_mae * 100:.2f}%')

    return model, X_test, y_test, y_pred
```

[6]:
```
unique_jobType = df['JobType'].unique()
print(unique_jobType)
```

```
[nan 'Full Time' 'Contract/Temp' 'Part Time' 'Casual/Vacation']
```

[7]:
```
#filtered_df = df[(df['JobType'] == 'Full Time') | (df['JobType'] == 'Contract/
↪Temp')]
filtered_df = df[(df['JobType'] == 'Full Time')]
```

### 7.0.1 Initial Findings from model

- Lower mean absolute error when only keeping 'Full Time' JobType, although, may be less accurate when trying to predict other JobTypes.
- We also tested a Random Forest Regressor, which took ~40 times longer to fit the model than the LGBMRegressor, with 10% of the data, and 10% of the n_estimators. It did not provide noticable accuracy improvements.

**We also tested a Random Forest Regressor, which took ~40 times longer to fit the model than the LGBMRegressor, with 10% of the data, and 10% of the n_estimators. It did not provide noticable accuracy improvements.**

[8]:
```
categorical_features = ['Title', 'Company', 'Date', 'Location', 'Area',␣
↪'Classification', 'SubClassification', 'JobType', 'Year', 'Month', 'Day',␣
↪'DayOfWeek']
# Splitting features and target
X = df[categorical_features]
y = df['AverageSalary']
n_estimators = 100

model_allfeat, X_test_allfeat, y_test_allfeat, y_pred_allfeat = Regressor(X, y,␣
↪categorical_features, n_estimators)
```

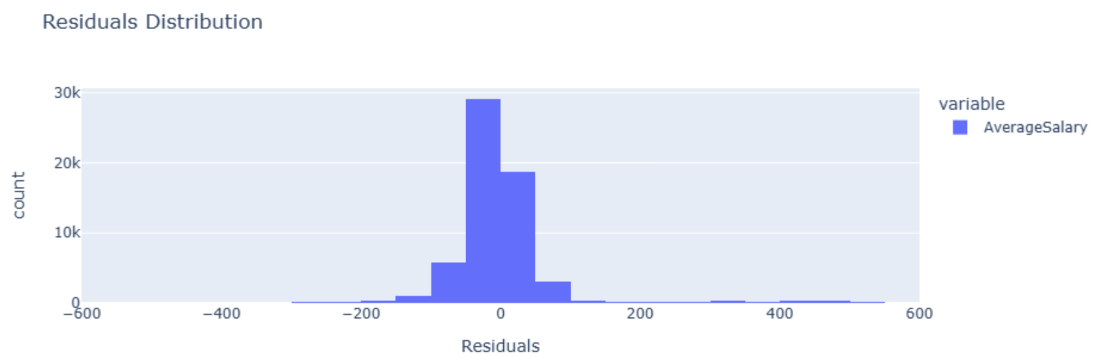
```
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of
testing was 0.314478 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 6696
[LightGBM] [Info] Number of data points in the train set: 243449, number of used
features: 3348
[LightGBM] [Info] Start training from score 91.478047

Trained on feature set: ['Title', 'Company', 'Date', 'Location', 'Area',
'Classification', 'SubClassification', 'JobType', 'Year', 'Month', 'Day',
'DayOfWeek']

Mean Salary: 90.97867341406109
Mean Absolute Error: 42.17853681822634
Relative MAE: 46.36%
```

## 7.1 Interactive Visualisations

[9]:
```
# Interactive plot of actual vs. predicted salaries
fig = px.scatter(x=y_test_allfeat, y=y_pred_allfeat, labels={'x': 'Actual␣
↪Salary', 'y': 'Predicted Salary'},
                 title='Actual vs. Predicted Salaries')
fig.add_trace(go.Scatter(x=[y_test_allfeat.min(), y_test_allfeat.max()],␣
↪y=[y_test_allfeat.min(), y_test_allfeat.max()],
```

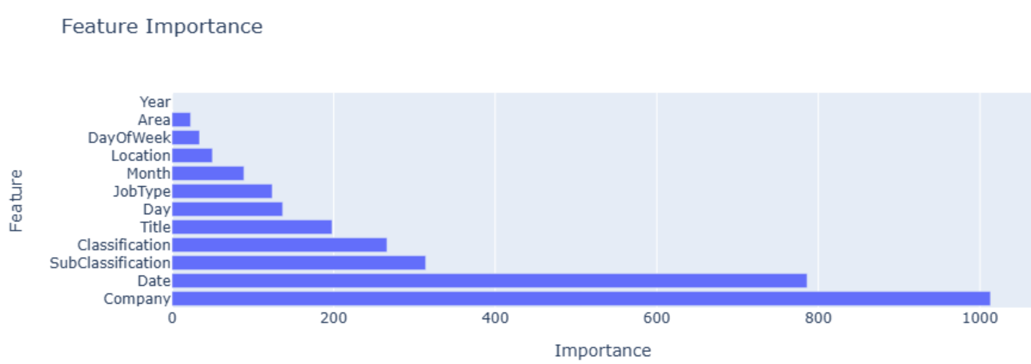
```
                        mode='lines', line=dict(color='red', dash='dash'),␣
↪name='Ideal Fit'))

fig.show()
```


[10]:
```
# Interactive plot of residuals
residuals = y_test_allfeat - y_pred_allfeat
fig_residuals = px.histogram(residuals, nbins=50, labels={'value':␣
↪'Residuals'}, title='Residuals Distribution')
fig_residuals.show()
```




[11]:
```
# Extract feature importances
importances = model_allfeat.named_steps['regressor'].feature_importances_
onehot_feature_names = model_allfeat.named_steps['preprocessor'].
↪named_transformers_['cat']['onehot'].
↪get_feature_names_out(categorical_features)
```

```
# Create a DataFrame for feature importances
importance_df = pd.DataFrame({'Feature': onehot_feature_names, 'Importance':␣
↪importances})

# Aggregate importances by original categorical feature
aggregated_importances = []
for feature in categorical_features:
    feature_mask = importance_df['Feature'].str.startswith(feature)
    total_importance = importance_df.loc[feature_mask, 'Importance'].sum()
    aggregated_importances.append({'Feature': feature, 'Importance':␣
↪total_importance})

# Add numerical features to the aggregated importances list
numerical_importances = importances[len(onehot_feature_names):]
for feature, importance in zip(X.columns[len(categorical_features):],␣
↪numerical_importances):
    aggregated_importances.append({'Feature': feature, 'Importance':␣
↪importance})

# Convert aggregated importances list to DataFrame
importance_df = pd.DataFrame(aggregated_importances)

# Sort by importance
importance_df = importance_df.sort_values(by='Importance', ascending=False)

# Plot feature importances
fig_feature_importance = px.bar(importance_df, x='Importance', y='Feature',␣
↪orientation='h', title='Feature Importance')
fig_feature_importance.show()
```




[12]:
```
importance_df
```

[12]:

| | Feature | Importance |
|---|---|---|
| 1 | Company | 1013 |
| 2 | Date | 786 |
| 6 | SubClassification | 314 |
| 5 | Classification | 266 |
| 0 | Title | 198 |
| 10 | Day | 137 |
| 7 | JobType | 124 |
| 9 | Month | 89 |
| 3 | Location | 50 |
| 11 | DayOfWeek | 34 |
| 4 | Area | 23 |
| 8 | Year | 0 |

### 7.1.1 Based on the feature importance, we will train a new model

[13]:

```
threshold = 100

selected_features = list(importance_df[importance_df['Importance'] >␣
 ↪threshold]['Feature'])
# Splitting features and target
X_selected = X[selected_features]
y = df['AverageSalary']
n_estimators = 100

model_subfeat, X_test_subfeat, y_test_subfeat, y_pred_subfeat =␣
 ↪Regressor(X_selected, y, selected_features, n_estimators)
```

```
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of
testing was 0.001188 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 6494
[LightGBM] [Info] Number of data points in the train set: 243449, number of used
features: 3247
[LightGBM] [Info] Start training from score 91.478047

Trained on feature set: ['Company', 'Date', 'SubClassification',
'Classification', 'Title', 'Day', 'JobType']

Mean Salary: 90.97867341406109
Mean Absolute Error: 42.488301883699734
Relative MAE: 46.70%
```

**By selecting only those features above the threshold of 100, we receive practically the same accuracy, with a slightly quicker training time.**

## 7.2 We will try some methods for improving accuracy

[14]:
```
from sklearn.model_selection import GridSearchCV
```

[15]:
```
def TuneModel(X, y, model, selected_features, param_grid):
    grid_search = GridSearchCV(model, param_grid, cv=3,␣
 ↪scoring='neg_mean_absolute_error', n_jobs=-1)

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

    grid_search.fit(X_train, y_train)

    # Get the best hyperparameters
    best_params = grid_search.best_params_
    print("Best Hyperparameters:", best_params)

    # Use the best estimator for prediction
    best_model = grid_search.best_estimator_
    y_pred_best = best_model.predict(X_test)

    # Evaluate performance
    mean_salary = y_test.mean()
    mae_best = mean_absolute_error(y_test, y_pred_best)
    relative_mae = mae_best / mean_salary

    print(f'\nTrained on feature set: {selected_features}')

    print(f'\nMean Salary: {mean_salary}')
    print(f'Mean Absolute Error (Best Model): {mae_best}')
    print(f'Relative MAE: {relative_mae * 100:.2f}%')

    return grid_search, X_test, y_test, y_pred_best
```

[16]:
```
# Splitting features and target
X = df[categorical_features]
y = df['AverageSalary']

# Define hyperparameter grid for LightGBM
param_grid = {
    'regressor__learning_rate': [0.05, 0.1, 0.2],
    'regressor__num_leaves': [31, 50, 100],
    'regressor__max_depth': [3, 5, -1]
}
```

```
gs_tunedmodel, X_test_gs_tunedmodel, y_test_gs_tunedmodel,␣
↪y_pred_best_gs_tunedmodel = TuneModel(X, y, model_allfeat,␣
↪categorical_features, param_grid)
```

```
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of
testing was 0.412025 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 6696
[LightGBM] [Info] Number of data points in the train set: 243449, number of used
features: 3348
[LightGBM] [Info] Start training from score 91.478047
Best Hyperparameters: {'regressor__learning_rate': 0.2, 'regressor__max_depth':
-1, 'regressor__num_leaves': 100}

Trained on feature set: ['Title', 'Company', 'Date', 'Location', 'Area',
'Classification', 'SubClassification', 'JobType', 'Year', 'Month', 'Day',
'DayOfWeek']

Mean Salary: 90.97867341406109
Mean Absolute Error (Best Model): 39.20304991216834
Relative MAE: 43.09%
```

**This tuning took nearly 3 minutes, but we have identified the following hyperparameters to be the best for this model:** - regressor___learning_rate: 0.2 - regressor___max_depth: -1 - regressor___num_leaves: 100

**Lets visualise the results**

[17]:
```
# Interactive plot of actual vs. predicted salaries
fig = px.scatter(x=y_test_gs_tunedmodel, y=y_pred_best_gs_tunedmodel,␣
↪labels={'x': 'Actual Salary', 'y': 'Predicted Salary'},
                 title='Actual vs. Predicted Salaries')
fig.add_trace(go.Scatter(x=[y_test_gs_tunedmodel.min(), y_test_gs_tunedmodel.
↪max()], y=[y_test_gs_tunedmodel.min(), y_test_gs_tunedmodel.max()],
                         mode='lines', line=dict(color='red', dash='dash'),␣
↪name='Ideal Fit'))

fig.show()
```

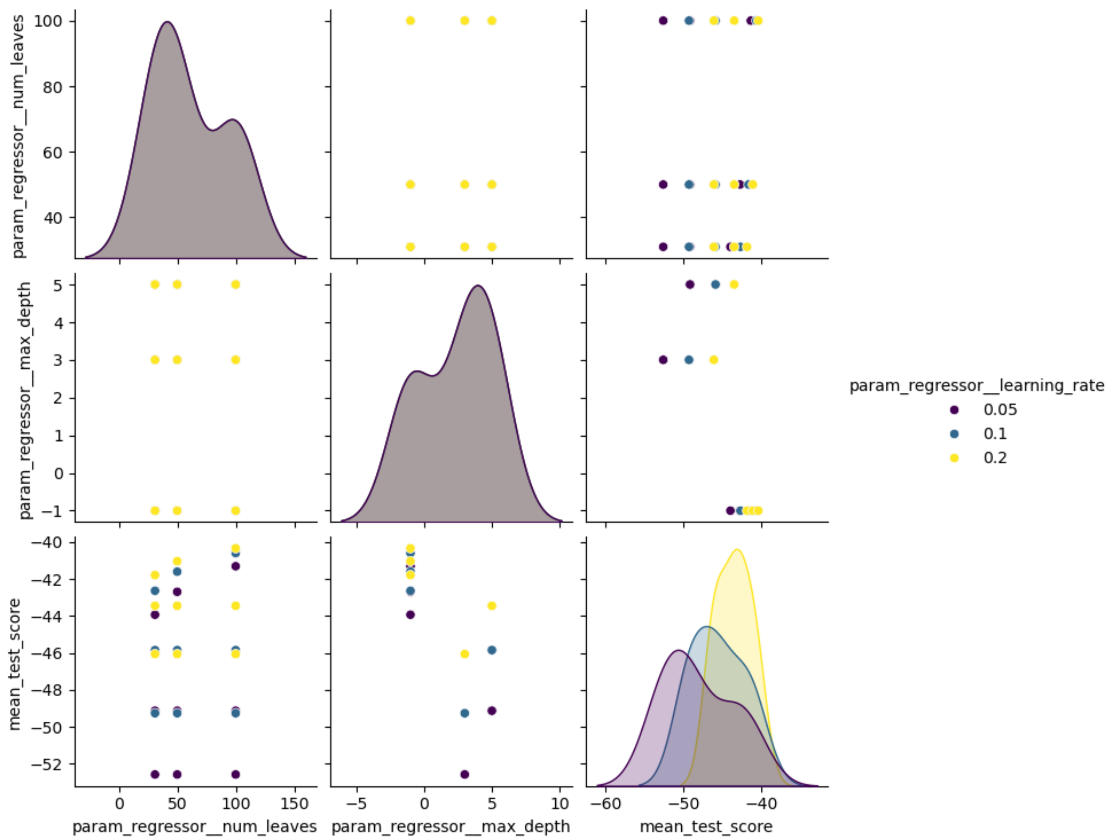

[18]:
```
# Interactive plot of residuals
residuals = y_test_gs_tunedmodel - y_pred_best_gs_tunedmodel
fig_residuals = px.histogram(residuals, nbins=50, labels={'value':␣
↪'Residuals'}, title='Residuals Distribution')
fig_residuals.show()
```

[23]:
```
# Extract grid search results
results = pd.DataFrame(gs_tunedmodel.cv_results_)

# Pivot the results to create a grid for the heatmap
pivot_table = results.pivot_table(index='param_regressor__num_leaves',
                                  columns=['param_regressor__max_depth',␣
↪'param_regressor__learning_rate'],
                                  values='mean_test_score')

# Create a series of subplots for each learning rate
fig = go.Figure()

for lr in param_grid['regressor__learning_rate']:
    sub_pivot = pivot_table.xs(lr, level='param_regressor__learning_rate',␣
↪axis=1)
    fig.add_trace(go.Surface(z=sub_pivot.values,
                             x=sub_pivot.columns.get_level_values(0),
                             y=sub_pivot.index,
                             name=f'Learning Rate: {lr}'))

# Customize the layout
fig.update_layout(title='Grid Search Results (MAE)',
                  scene=dict(xaxis_title='Max Depth',
                             yaxis_title='Num Leaves',
                             zaxis_title='MAE'),
                  height=800, width=1200)
```

```
# Show the interactive plot
fig.show()
```

### 7.2.1 Scatter Plot Matrix (SPLOM)

[32]:
```
# Select relevant columns for SPLOM
relevant_cols = ['param_regressor__learning_rate',␣
↪'param_regressor__num_leaves', 'param_regressor__max_depth',␣
↪'mean_test_score']
sploam_data = results[relevant_cols]

# Plot SPLOM
sns.pairplot(sploam_data, hue='param_regressor__learning_rate',␣
↪palette='viridis')
plt.show()
```




[33]:
```
# Create 3D scatter plot
fig = go.Figure(data=[go.Scatter3d(
    x=results['param_regressor__num_leaves'],
```

```
    y=results['param_regressor__max_depth'],
    z=results['mean_test_score'],
    mode='markers',
    marker=dict(
        size=8,
        color=results['param_regressor__learning_rate'],  # Color by learning␣
 ↪rate
        colorscale='Viridis',  # Set color scale
        opacity=0.8
    )
)])

# Update layout
fig.update_layout(
    scene=dict(
        xaxis_title='Num Leaves',
        yaxis_title='Max Depth',
        zaxis_title='MAE'
    ),
    title='3D Scatter Plot of Hyperparameters and MAE'
)

# Show plot
fig.show()
```

## 8 8 Data analytics findings

Overall, team is frequently mentioned in requirements amongst all job postings. This suggests that teamwork and working within a team is one of the most important skills for any job. When accounting for salaries, ICT, healthcare and trades are the top 3 sectors in Australia. ACT has the highest average salary within Australia, and a number of job postings equal to Adelaide, whereas Sydney has the highest number of job postings, and the second highest average salary, within Australia. The top sub sector among all cities was management, which was to be expected given all jobs require managers after a certain company size.

## 9 9 Suggested actions for balancing the markets

To balance the market share of job sectors, actions that can be taken include tax breaks for companies within sectors that are underrepresented and more accessible and affordable education for important skills used in underrepresented sectors.

## 10 10 Refining data analytics

Different data sources could have been used to provide a more representative sample, for example using different recruitment websites such as LinkedIn or Jora, or even paper postings such as those in the newspaper.

Different parameters that could have been used include the experience required for a position, which would help seperate skills required in more hands on, lower level positions compared to management positions. The size of the company who submitted the job position could also have been used, as positions in smaller companies may demand a greater number of responsibilities and therefore skills required compared to more specialised roles in larger companies.

Other techniques that could have been used include topic modelling to better classify similar skills, e.g. groomer and animal are both associated with the topic of animal caretaking and management.

More relevant data could have been obtained by using more up to date job postings, and by only considering job postings that are less than a certain number of months old. This is because, according to one survey, many job postings are created without the intention to hire, and are therefore not representative of potential positions a candidate could fill (https://clarifycapital.com/job-seekers-beware-of-ghost-jobs-survey).

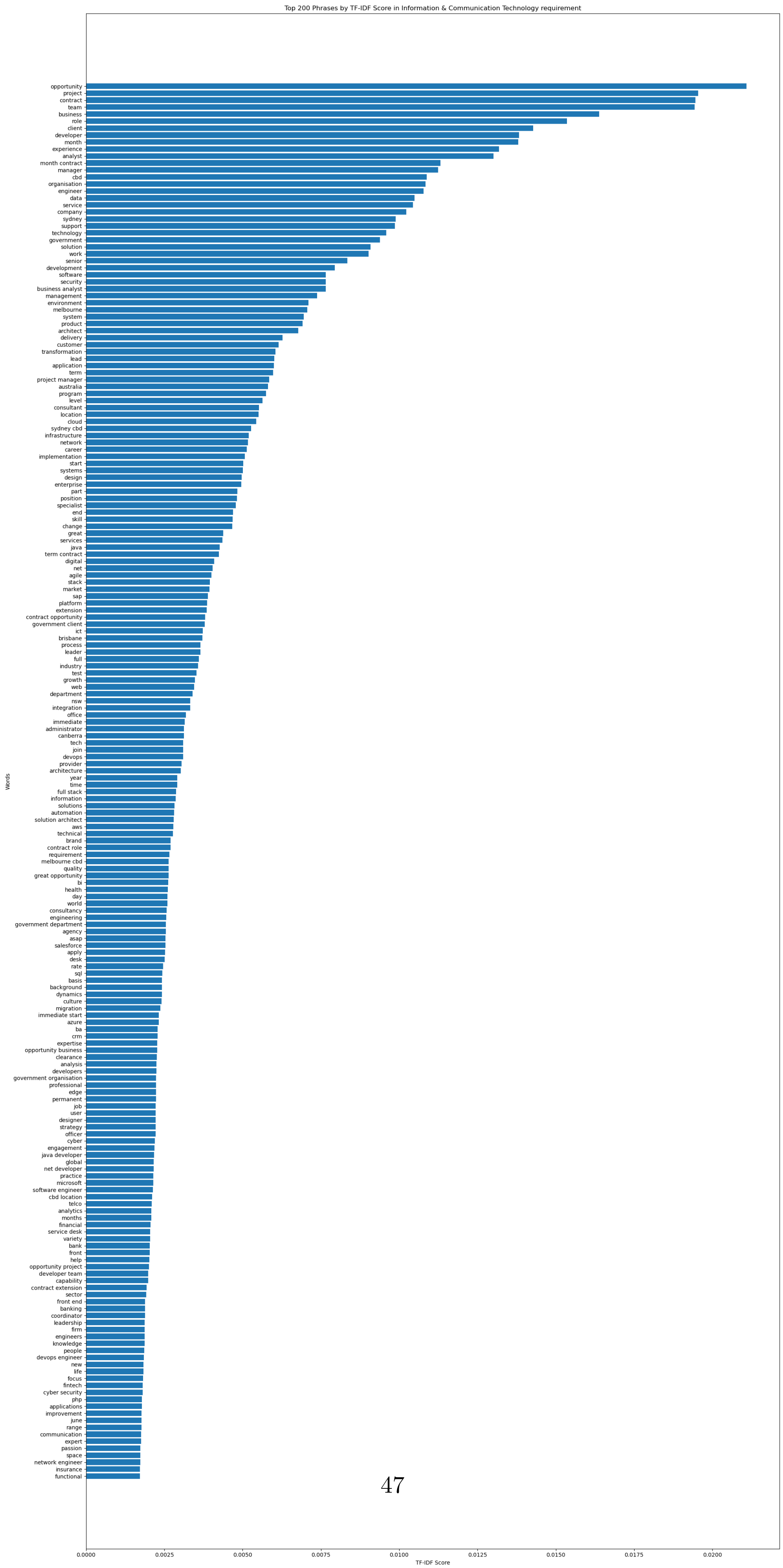
## 11 11 Implications for employers and employees

Implications for exmployees include that ICT, healthcare and trades are the best sector to pursue jobs in, given their balance of a high number of jobs and good salaries. Implications for employers include that Adelaide, despite having a population triple that of the ACT, has about the same number of job openings, indiciating that these labour markets may be in favour of employers given a relative lack of competition.

## 12 12 Case Study 1

[158]:
```
import nltk
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import TfidfVectorizer
from nltk import pos_tag
```

[260]:
```
skill_identifier("Information & Communication Technology", 200, 'tokenized_data.
↪csv')
```


Top 200 Phrases by TF-IDF Score in Information & Communication Technology requirement
opportunity
project
contract
team
business
role
client
developer
month
experience
analyst
month contract
manager
cbd
organisation
engineer
data
service
company
sydney
support
technology
government
solution
work
senior
development
software
security
business analyst
management
environment
melbourne
system
product
architect
delivery
customer
transformation
lead
application
term
project manager
australia
program
level
consultant
location
cloud
sydney cbd
infrastructure
network
career
implementation
start
systems
design
enterprise
part
position
specialist
end
skill
change
great
services
java
term contract
digital
net
agile
stack
market
sap
platform
extension
contract opportunity
government client
ict
brisbane
process
leader
full
industry
test
growth
web
department
nsw
integration
office
immediate
administrator
canberra
tech
join
devops
provider
architecture
year
time
full stack
information
solutions
automation
solution architect
aws
technical
brand
contract role
requirement
melbourne cbd
quality
great opportunity
bi
health
day
world
consultancy
engineering
government department
agency
asap
salesforce
apply
desk
rate
sql
basis
background
dynamics
culture
migration
immediate start
azure
ba
crm
expertise
opportunity business
clearance
analysis
developers
government organisation
professional
edge
permanent
job
user
designer
strategy
officer
cyber
engagement
java developer
global
net developer
practice
microsoft
software engineer
cbd location
telco
analytics
months
financial
service desk
variety
bank
front
help
opportunity project
developer team
capability
contract extension
sector
front end
banking
coordinator
leadership
firm
engineers
knowledge
people
devops engineer
new
life
focus
fintech
cyber security
php
applications
improvement
june
range
communication
expert
passion
space
network engineer
insurance
functional
Words
0.0000
0.0025
0.0050
0.0075
0.0100
0.0125
0.0150
0.0175
0.0200
TF-IDF Score

Based on the current job market dataset, Matthew should take subjects in: - Devops - Sql - Cloud - Amazon Web Services (aws) - Enterprise resource planning software (sap) - Business Intelligence software (bi) - Front end web development (front end) - Java - Agile software development - Full stack development

This is because the terms associated with these skills have a high tf-idf score, indicating that they are most relevant.

## 13 13 Case Study 2

[7]:
```
#Will most likely use second version as it reads from external txt but leaving␣
↪this in incase we change our minds
import numpy as np

df = pd.read_csv('data.csv')
df.fillna('', inplace=True)
df['combined_text'] = df['Title'] + ' ' + df['Company'] + ' ' + df['Location']␣
↪+ ' ' + df['Area'] + ' ' + df['Classification'] + ' ' +␣
↪df['SubClassification'] + ' ' + df['Requirement'] + ' ' +␣
↪df['FullDescription']
df[['Id', 'combined_text']].head()

from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(stop_words='english', max_features=5000)

tfidf_matrix = vectorizer.fit_transform(df['combined_text'])
tfidf_matrix.shape

from sklearn.metrics.pairwise import cosine_similarity

def recommend_jobs(candidate_profile, tfidf_matrix, df, top_n=10):
    candidate_tfidf = vectorizer.transform([candidate_profile])
    cosine_similarities = cosine_similarity(candidate_tfidf, tfidf_matrix).
↪flatten()
    top_indices = cosine_similarities.argsort()[-top_n:][::-1]

    return df.iloc[top_indices]

candidate_profile = "Data scientist with experience in machine learning,␣
↪Python, and data analysis"

top_jobs = recommend_jobs(candidate_profile, tfidf_matrix, df)
top_jobs
```

```
top_jobs[['Id', 'Title', 'Company', 'Location', 'LowestSalary',␣
↪'HighestSalary', 'JobType']]
```

```
/var/folders/4b/x1qmm8g167d86vjglntrzwdh0000gn/T/ipykernel_15687/3505365634.py:4
: DtypeWarning: Columns (0,4,5,6,7) have mixed types. Specify dtype option on
import or set low_memory=False.
  df = pd.read_csv('data.csv')
```

[7]:
```
              Id                                             Title  \
22766   37359901                 Data Scientist - Machine Learning
217101  37960667         Data Scientist - Machine Learning, Python
65365   37438144  Data Scientist - Machine learning - Federal Go…
212820  37981317                                    Data Scientist
108685  37578892                             Senior Data Scientist
131481  37634094                                    Data Scientist
247541  38087116  Data Scientist l Machine Learning l $150,000 -…
200477  37985761  Data Scientist, Analytics, Machine Learning - …
217815  37963250                                    Data Scientist
246736  38001628                                   Data Scientist

                         Company   Location  LowestSalary  HighestSalary  \
22766   FourQuarters Recruitment  Melbourne           150            200
217101          HiTech Personnel                      150            200
65365           HiTech Personnel        ACT           150            200
212820            SustainDigital                      100            120
108685           Morgan McKinley     Sydney           120            150
131481            SustainDigital     Sydney           100            120
247541                                                  0             30
200477              Infinity Pro                        0             30
217815            Compas Pty Ltd                      150            200
246736                Exclaim IT                      150            200

              JobType
22766       Full Time
217101  Contract/Temp
65365   Contract/Temp
212820      Full Time
108685      Full Time
131481      Full Time
247541      Full Time
200477      Full Time
217815  Contract/Temp
246736  Contract/Temp
```

[4]:
```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
```

```python
df = pd.read_csv('data.csv')
df.fillna('', inplace=True)

df['combined_text'] = df['Title'] + ' ' + df['Company'] + ' ' + df['Location']␣
↪+ ' ' + df['Area'] + ' ' + df['Classification'] + ' ' +␣
↪df['SubClassification'] + ' ' + df['Requirement'] + ' ' +␣
↪df['FullDescription']
vectorizer = TfidfVectorizer(stop_words='english', max_features=5000)
tfidf_matrix = vectorizer.fit_transform(df['combined_text'])

with open('resume.txt', 'r') as file:
    candidate_profile = file.read()

def recommend_jobs(candidate_profile, tfidf_matrix, df, top_n=10):
    candidate_tfidf = vectorizer.transform([candidate_profile])

    cosine_similarities = cosine_similarity(candidate_tfidf, tfidf_matrix).
↪flatten()

    top_indices = cosine_similarities.argsort()[-top_n:][::-1]

    return df.iloc[top_indices]

top_jobs = recommend_jobs(candidate_profile, tfidf_matrix, df)

top_jobs[['Id', 'Title', 'Company', 'Location', 'LowestSalary',␣
↪'HighestSalary', 'JobType']]
```

```
/var/folders/4b/x1qmm8g167d86vjglntrzwdh0000gn/T/ipykernel_15687/2064012591.py:5
: DtypeWarning: Columns (0,4,5,6,7) have mixed types. Specify dtype option on
import or set low_memory=False.
  df = pd.read_csv('data.csv')
```

[4]:

```
                                                        Id  \
297032                                            38370021
233800                                            38010359
301026                                            38317273
277283                                            38149897
4396                                              37388208
93618                                             37537775
163173  37899346&searchrequesttoken=fd828ae7-86fe-44e0…
117843                                            37643003
46705                                             37482628
224020                                            37993474

                                                   Title  \
```

```
297032                  Customer Service Representative
233800                  Customer Service Representative
301026                  Customer Service Representative
277283  Casual Food Delivery Driver in Collaroy Plateau
4396           Part time Food Delivery Driver in Doreen
93618          Part time Food Delivery Driver in Doreen
163173    Part time Food Delivery Driver in Craigieburn
117843         Part time Food Delivery Driver in Altona
46705          Part time Food Delivery Driver in Altona
224020           Casual Food Delivery Driver in Mentone

                           Company   Location  LowestSalary  HighestSalary  \
297032                                                    0             30
233800  PFD Food Services Pty Ltd                        40             50
301026  PFD Food Services Pty Ltd                        30             40
277283                 Jora Local                        30             40
4396                   Jora Local  Melbourne             30             40
93618                  Jora Local  Melbourne             30             40
163173                 Jora Local  Melbourne             30             40
117843                 Jora Local  Melbourne             30             40
46705                  Jora Local  Melbourne             30             40
224020                 Jora Local                        30             40

                JobType
297032
233800
301026
277283  Casual/Vacation
4396          Part Time
93618         Part Time
163173        Part Time
117843        Part Time
46705         Part Time
224020  Casual/Vacation
```

TalentFinders is trying to match employees CVs with suitable job opportunities based on their job sector, skills, experience ect. This case study will explore how is it possible to leverage job market data to accurately match a resume to the top ten most suitable jobs for the candidate, using the dataset. This dataset contains job advertisements with various details such as job titles, companies, locations and descriptions.

The dataset is first loaded into a pandas dataframe to make manipulating and changing the data easier. Any missing values are also replaced by empty strings to make the data compatible and consistent.

Then all the text fields that could possibly relate to the candidates relevancy to a particular position were combined into a single column. These included the job titles, company names, locations, areas, classifications, sub-classifications, requirements and full descriptions.

The TF-IDF (Term Frequency-Inverse Document Frequency) technique was then used to convert the text in the combined data into numerical variables to make it easier to capture the importance of the words relative to the dataset. Term frequency measures how frequently the term appears and INverse Document Frequency measures how rare the term is across the document, TF-IDF is the combination of these two values.

The I read the candidates resume from a resume.txt file. To compare the individual profile against the dataset to find the most relevant jobs. To recommend the jobs a function was defined which caluclulates the cosine similarity between this profile and all job descriptions in the dataset. This function returns the top N job descriptions that are the most similar. These can then be generated and displayed with the relevant details.

This way the job recommender uses natural language processing techniques and machine learning to match potential candidates with job opportunities based on the textual similarity between their resume and the job description.