



Programming for Data Analysis (PDAS)
ST1510

CA1 PROJECT

What factors make job postings more visible and increase application rates,
and how can job seekers use these insights to improve their job search?



Name: Goh Kun Ming
Class: DAAA/FT/1B/02

SUMMARY OF STEPS

- **Merging Dataframes**
- **Imputing Missing Data**
- **Feature Engineering**
- **Removing Outliers**
- **Data Analysis & Visualisation**



Merging Dataframes

All 5 datasets were imported as Dataframes and merged together using Job ID. Subsequently, some Data Columns were dropped.



Imputing Missing Data

Data Columns with Missing Data were imputed using various techniques such as median, interpolate, etc.



Feature Engineering

New Features were created to summarised certain data columns which aided in Data Analysis.



Removing Outliers

Outliers from normalised salary were removed using the IQR Method to prevent inaccurate Data Analysis.



Data Analysis & Visualisation

Data Analysis and Visualisation were conducted on the cleaned data which showed proportions and relationships

DATA WRANGLING & ANALYSIS USING PANDAS

IMPUTING MISSING DATA

Missing Values	
company_name	1719
title	0
max_salary	94056
pay_period	87776
location	0
views	1689
min_salary	94056
formatted_work_type	0
applies	100529
remote_allowed	108603
application_type	0
expiry	0
formatted_experience_level	29409
listed_time	0

Missing Values	
company_name	0
title	0
max_salary	0
pay_period	0
location	0
views	0
min_salary	0
formatted_work_type	0
applies	0
remote_allowed	0
application_type	0
expiry	0
formatted_experience_level	0
listed_time	0

REMOVING DUPLICATED DATA

```
# ----- Calculate Total Duplicated Rows ----- #
duplicatesCount = merged_df.duplicated().sum()

# ----- Create DataFrame to Store Result ----- #
duplicateSummary = pd.DataFrame({'Count': [duplicatesCount]}, index = ['Duplicated Rows'])

# ----- Function for Colour Coding ----- #
def colour_code(val):
    if val > 0:
        return 'background-color: red; color: white'
    else:
        return 'background-color: green; color: white'

# ----- Apply Colour Coding to the 'Count' Column ----- #
duplicateSummary = duplicateSummary.style.map(colour_code, subset=['Count'])

# ----- Display the Result ----- #
duplicateSummary
```

Count	
Duplicated Rows	77

```
# ----- Removing Duplicated Data ----- #
merged_df = merged_df.drop_duplicates()

# ----- Calculate Total Duplicated Rows ----- #
duplicatesCount = merged_df.duplicated().sum()

# ----- Create DataFrame to Store Result ----- #
duplicateSummary = pd.DataFrame({'Count': [duplicatesCount]}, index = ['Duplicated Rows'])

# ----- Function for Colour Coding ----- #
def colour_code(val):
    if val > 0:
        return 'background-color: red; color: white'
    else:
        return 'background-color: green; color: white'

# ----- Apply Colour Coding to the 'Count' Column ----- #
duplicateSummary = duplicateSummary.style.map(colour_code, subset=['Count'])

# ----- Display the Result ----- #
duplicateSummary
```

Count	
Duplicated Rows	0

CORRECTING DATA TYPES

```
company_name      object
title             object
max_salary        float64
pay_period        object
location          object
views            float64
min_salary        float64
formatted_work_type object
applies           float64
remote_allowed    float64
application_type  object
expiry           float64
formatted_experience_level object
listed_time       float64
dtype: object
```

```
company_name      object
title             object
max_salary        int64
pay_period        object
location          object
views            int64
min_salary        int64
formatted_work_type object
applies           int64
remote_allowed    int64
application_type  object
expiry           datetime64[ns]
formatted_experience_level object
listed_time       datetime64[ns]
dtype: object
```



DATA WRANGLING & ANALYSIS USING PANDAS

STANDARDISING FORMAT

```
# ----- Finding The Various Quantity ----- #
pay_period_counts = merged_df['pay_period'].value_counts().reset_index()

# ----- Renaming of Data Columns ----- #
pay_period_counts.columns = ['pay_period', 'count']

# ----- Setting New Index ----- #
pay_period_counts = pay_period_counts.set_index('pay_period')

# ----- Displaying of Results ----- #
pay_period_counts
```

	count
pay_period	
YEARLY	108373
HOURLY	14703
MONTHLY	516
WEEKLY	171
BIWEEKLY	9

```
# ----- Standardising Text Format ----- #
merged_df['pay_period'] = merged_df['pay_period'].replace({'YEARLY': 'Yearly', 'MONTHLY': 'Monthly', 'BIWEEKLY': 'Biweekly',
                                                         'WEEKLY': 'Weekly', 'HOURLY': 'Hourly'})

# ----- Verification of Changes ----- #
pay_period_counts = merged_df['pay_period'].value_counts().reset_index()

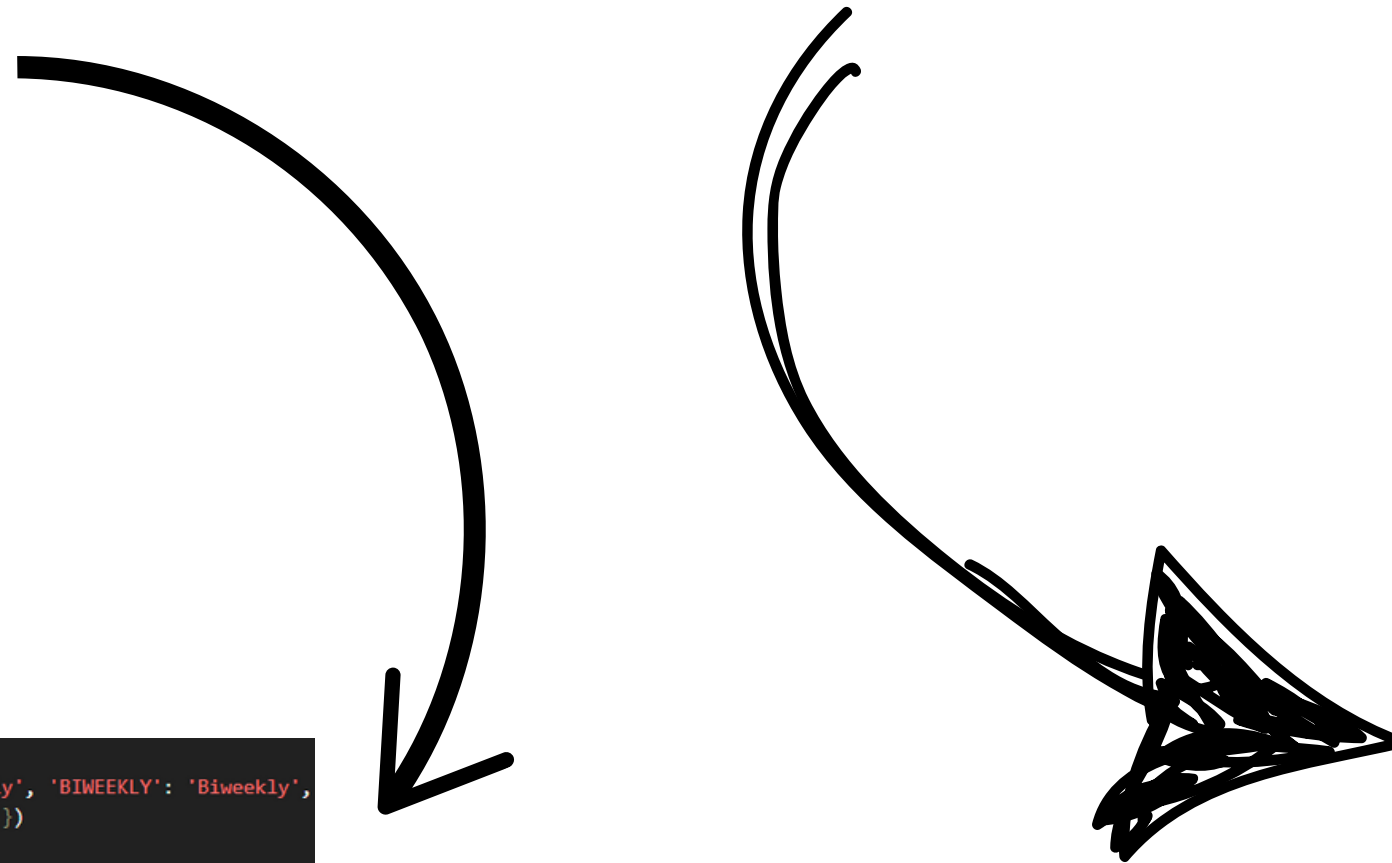
# ----- Renaming of Data Columns ----- #
pay_period_counts.columns = ['pay_period', 'count']

# ----- Setting New Index ----- #
pay_period_counts = pay_period_counts.set_index('pay_period')

# ----- Displaying of Results ----- #
pay_period_counts
```

	count
pay_period	
Yearly	97690
Hourly	14348
Monthly	485
Weekly	170
Biweekly	3

FEATURE ENGINEERING



$$\text{max_salary_normalised} = \text{max_salary (Monthly Pay)} \times 12$$

$$\text{max_salary_normalised} = \text{max_salary (Biweekly Pay)} \times 26$$

$$\text{max_salary_normalised} = \text{max_salary (Weekly Pay)} \times 52$$

$$\text{max_salary_normalised} = \text{max_salary (Hourly Pay)} \times 40 \times 52$$

$$\text{min_salary_normalised} = \text{min_salary (Monthly Pay)} \times 12$$

$$\text{min_salary_normalised} = \text{min_salary (Biweekly Pay)} \times 26$$

$$\text{min_salary_normalised} = \text{min_salary (Weekly Pay)} \times 52$$

$$\text{min_salary_normalised} = \text{min_salary (Hourly Pay)} \times 40 \times 52$$

$$\text{postingDuration} = \text{expiry} - \text{listed_time}$$

$$\text{salaryRange} = \text{max_salary_normalised} - \text{min_salary_normalised}$$

$$\text{applyViewRatio} = \frac{\text{applies}}{\text{views}}$$

DATA WRANGLING & ANALYSIS USING PANDAS

OUTLIERS REMOVAL (IQR)

```
# ----- Describe Method of IQRdf ----- #
IQRdf[['min_salary_normalised', 'max_salary_normalised', 'salary_range']].describe()
```

	min_salary_normalised	max_salary_normalised	salary_range
count	112696.000000	112696.000000	112696.000000
mean	82684.301280	110940.556133	28256.254854
std	33044.907334	44304.099748	19226.453740
min	1.000000	1.000000	0.000000
25%	58846.750000	78333.000000	13334.000000
50%	78750.000000	106250.000000	23334.000000
75%	103000.000000	140000.000000	39785.000000
max	181881.000000	252333.000000	85000.000000

OUTLIERS REMOVAL (Z-SCORE)

```
# ----- Describe Method of z_filtered_df ----- #
zScoreDf[['min_salary_normalised', 'max_salary_normalised', 'salary_range']].describe()
```

	min_salary_normalised	max_salary_normalised	salary_range
count	120527.000000	120527.000000	120527.000000
mean	86961.540252	118751.919338	31790.379085
std	38903.629858	53433.127356	24136.359517
min	1.000000	1.000000	0.000000
25%	60000.000000	80000.000000	14286.000000
50%	80521.000000	110296.000000	25000.000000
75%	107263.500000	149800.000000	43200.000000
max	365533.000000	383333.000000	123534.000000

WINNER

```
# ----- Identifying Invalid Values ----- #
invalidValues = merged_df[['max_salary_normalised', 'min_salary_normalised', 'views', 'applies']].lt(0).sum()

# ----- Storing in DataFrame ----- #
invalidValuesDf = pd.DataFrame({'Negative Values': invalidValues.values}, index = invalidValues.index)

# ----- Function for Colour Coding ----- #
def colour_code(val):
    if val > 0:
        return 'background-color: red; color: white'
    else:
        return 'background-color: green; color: white'

# ----- Apply Colour Coding ----- #
invalidValuesDf = invalidValuesDf.style.map(colour_code, subset=['Negative Values'])

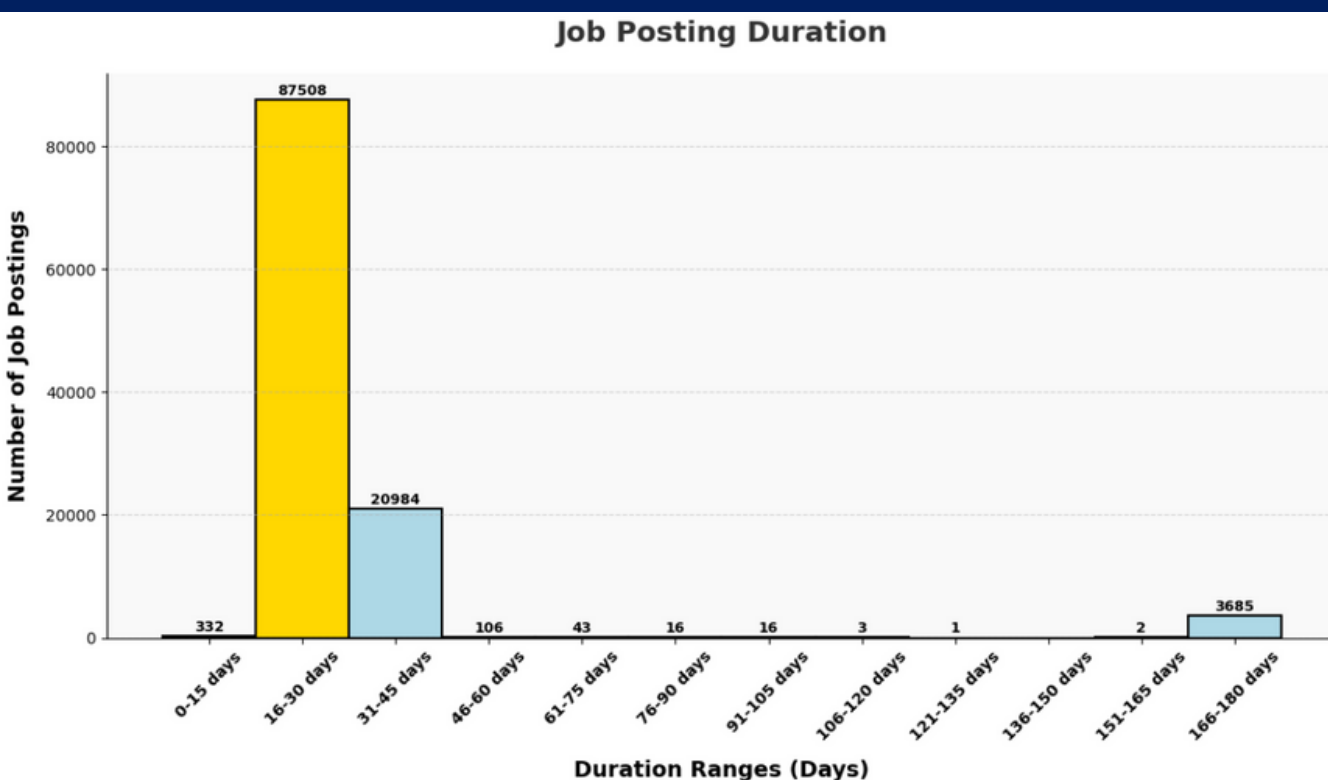
# ----- Display the Styled DataFrame ----- #
invalidValuesDf
```

Negative Values	
max_salary_normalised	0
min_salary_normalised	0
views	0
applies	0

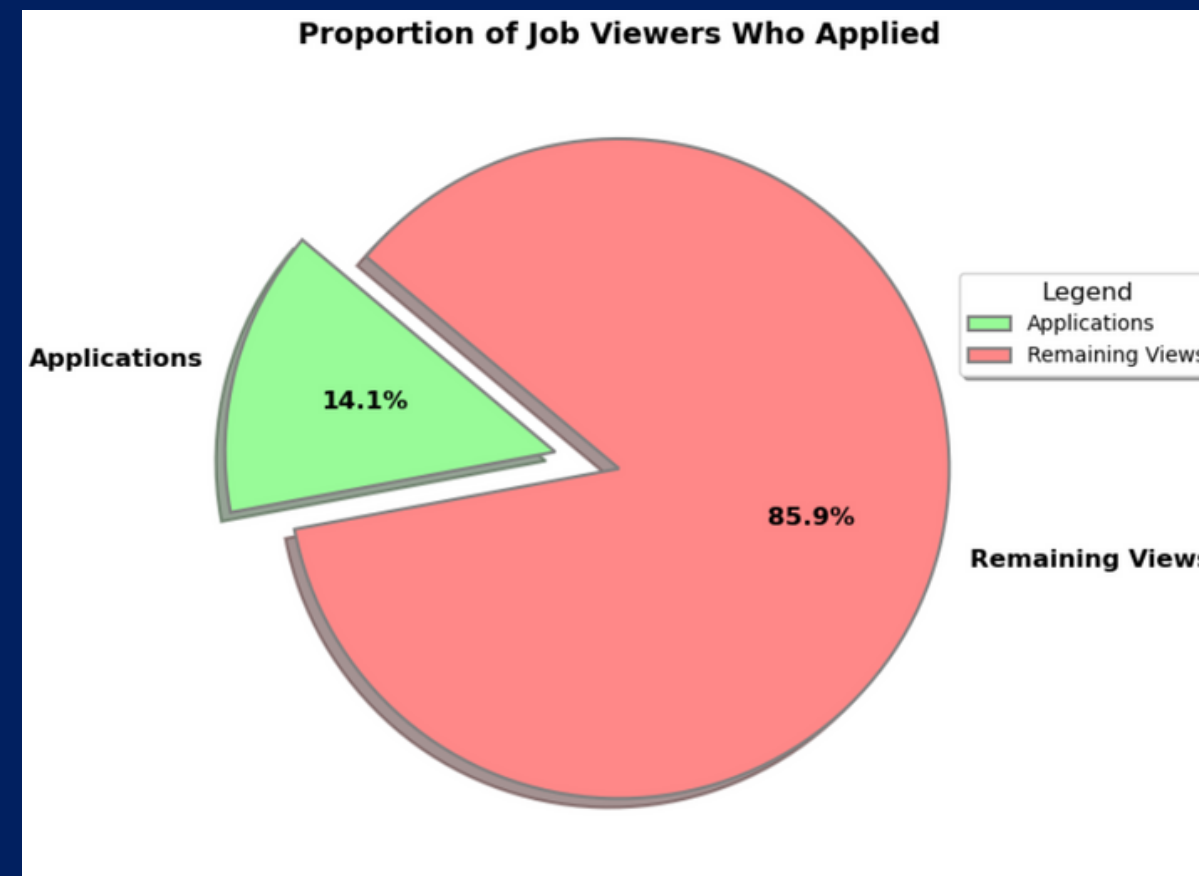
**VALIDATING
NUMERICAL DATA**

DATA VISUALISATION USING MATPLOTLIB

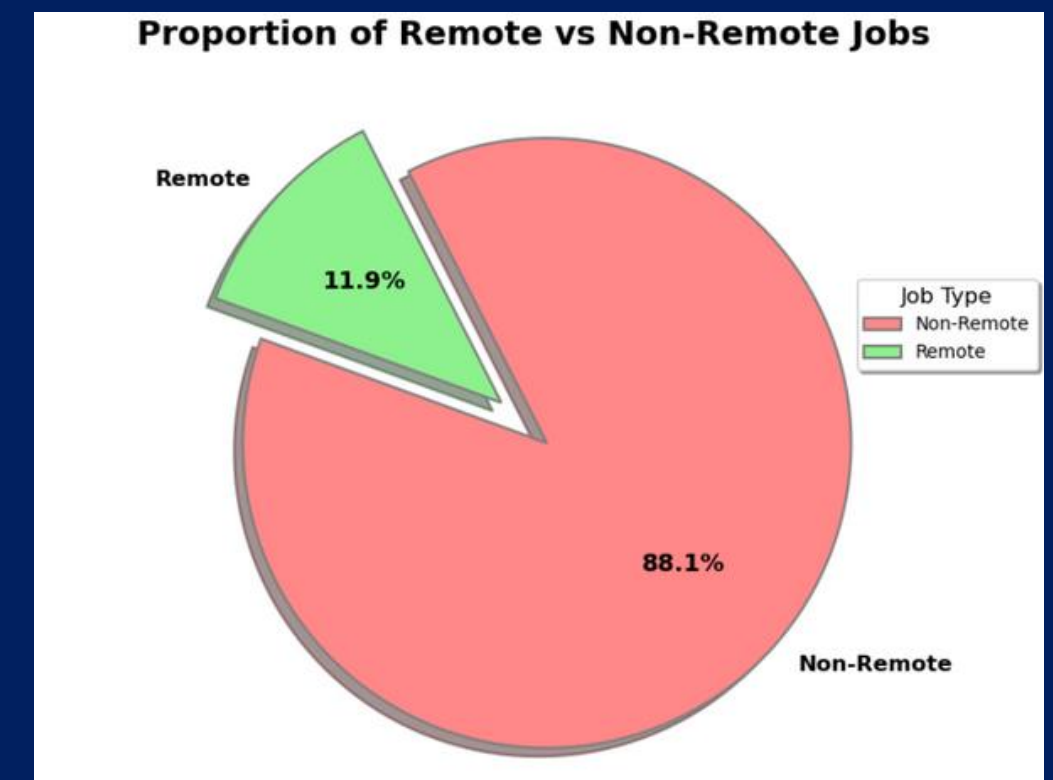
JOB POSTING DURATION



VIEWERS TO APPLICANTS RATIO



JOB TYPE PROPORTION



DATA VISUALISATION USING MATPLOTLIB

APPLY-TO-VIEW RATIO ACROSS SALARY RANGES

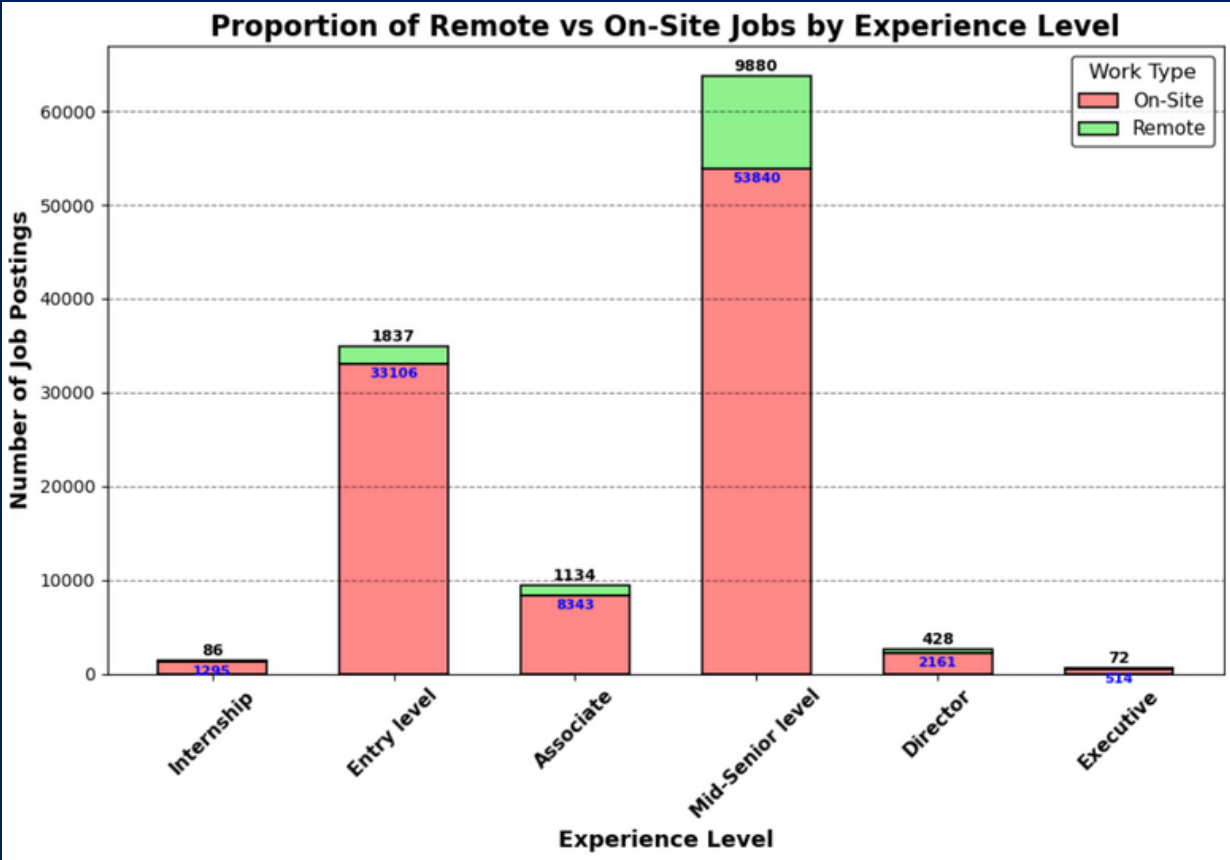


SALARY INSIGHTS ACROSS EXPERIENCE LEVEL AND APPLY-TO-VIEW RATIO

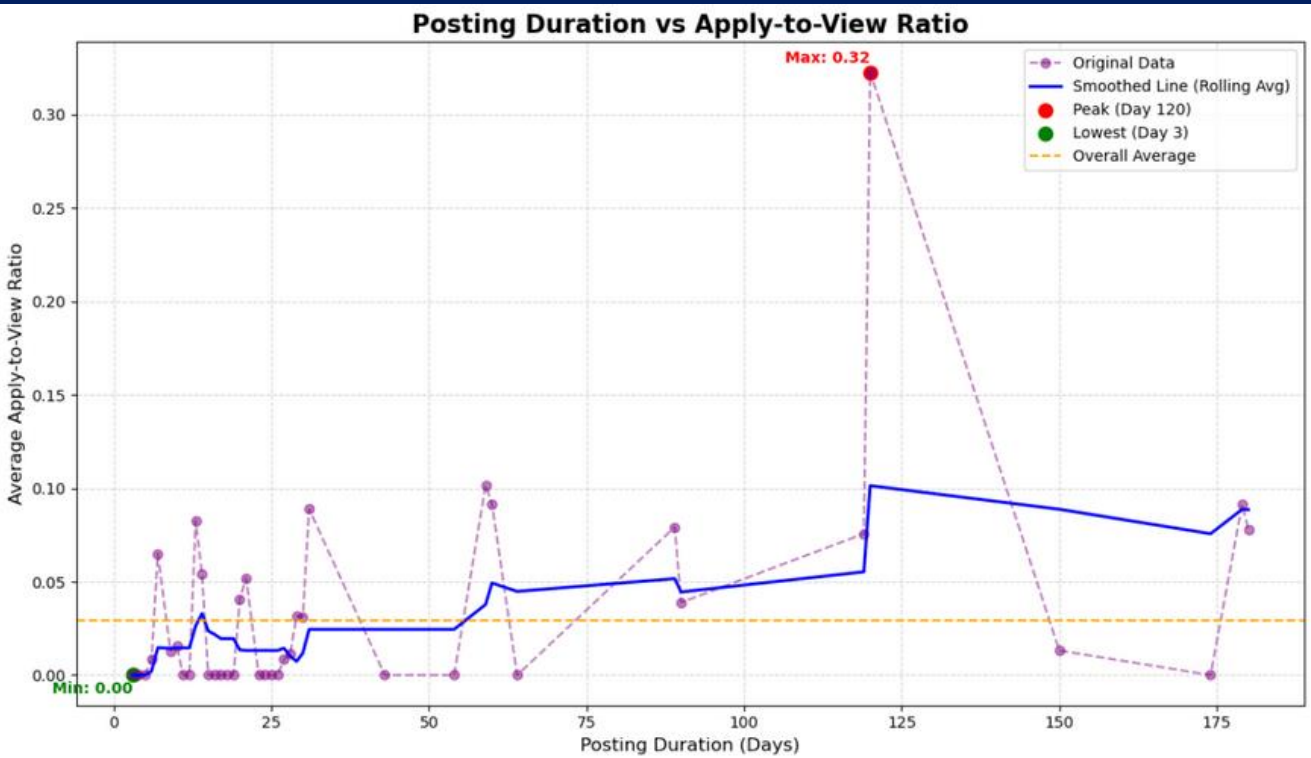


DATA VISUALISATION USING MATPLOTLIB

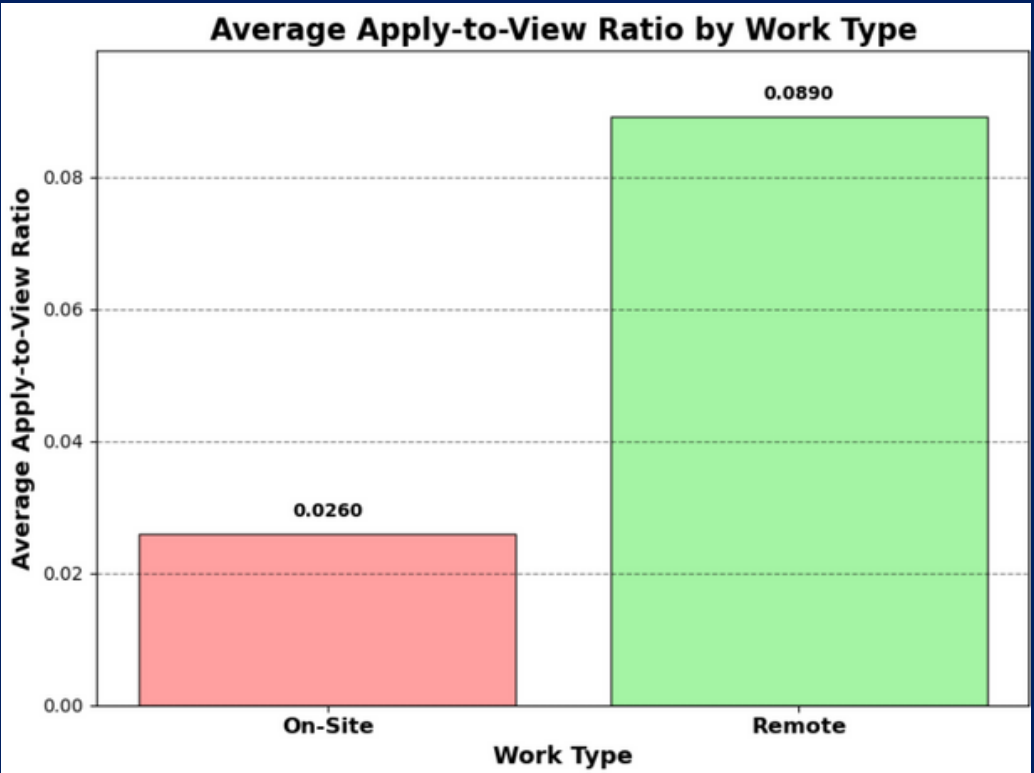
WORK TYPE PROPORTION ACROSS EXPERIENCE LEVEL



POSTING DURATION RELATION TO APPLY- TO-VIEW RATIO



APPLY-TO-VIEW RATIO BY WORK TYPE



KEY INSIGHTS GATHERED

- **JOB POSTINGS TYPICALY LAST FOR 1 MONTH**
- **NOT ALL VIEWERS ARE APPLICANTS**
- **VERY FEW COMPANIES OFFER REMOTE WORKING ARRANGEMENTS**
- **LOWER AND HIGHER SALARY RANGE HAS MORE APPLY-TO-VIEW RATIO**
- **HIGHER EXPERIENCE LEVEL POSITIONS PAY MORE BUT HAVE LOWER APPLY TO VIEW RATIO AS COMPARED TO LOWER EXPERIENCE LEVEL POSITION**
- **REMOTE WORK ARE EXTREMELY RARE FOR INTERNSHIP AND HIGHER EXPERIENCE LEVEL POSITIONS DESPITE HIGH DEMAND.**
- **THERE ARE MORE APPLICANTS FOR REMOTE WORK ARRANGEMENT THAN ON-SITE WORK ARRANGEMENT**
- **JOB POSTINGS HAS HIGHEST APPLY-TO-VIEW RATIO ON 120 DAYS OF POSTING DURATION.**

A background image showing a business meeting. Two hands are visible: one holding a pen and pointing at a line graph on a document, and another hand gesturing. The document contains various charts, including a bar chart with categories like 'Pharmacy', 'Barbers', 'Dietician', 'Lazada', and 'Beauty Clinic', and a pie chart. A yellow sticky note is placed on the document. The entire image has a blue tint.

THANK YOU