Data Analysis Report

Global AI Job Market and Salary Trends 2025

Minh Han

July 7th, 2025

I. Introduction

To build the data-analysis foundation I needed for machine learning, I completed a hands-on project using Python's NumPy, Pandas, Matplotlib and Seaborn on a Kaggle AI-job salary dataset (link: Kaggle Dataset). I cleaned and explored the data, then applied basic statistical tests to uncover patterns in salary distributions. All code is available on GitHub: https://github.com/ImaMinh/AI-Trends-Analysis

Data Analysis is a process of gathering, cleaning, exploring, and interpreting data, which enables us to explore the relationship and correlation between different data groups.
For this report, I want to document my process of analyzing data using a basic dataset taken from Kaggle (the largest site for open-source datasets).
The objectives are:

1. Understanding the relationship between AI Job Salaries and the following factors:
    a. Experience Level
    b. Employment Type (Full-Time, Part-Time, Contract, Freelance)
    c. Company's Remote Ratio
    d. Companies Size
    e. Company's Location
2. Ranking the top required skills of the AI job market in 2025.

Over this report, I hope I can guide some new beginners on how to get started with basic data analysis skills using Python. Additionally, you can also analyze data using tools like Excel, R, and online sites such as yData or SweetViz.

II. Outline:

Data Analysis consist of 6 Steps:

1. Define the objective
2. Obtaining the data
3. Data cleaning and Preprocessing
4. Exploratory Data Analysis
5. Statistical Testing
6. Interpretation of Results

For advanced analysis, you can optionally choose to:

1. Feature Engineering
2. Modelling

In this project, we will focus on using Python to clean data and prepare for EDA. Additionally applying basic statistics to find the relationship between discrete data.

III. Define the objective

The first step of any analysis project is to define the objective. That means knowing what you want to understand from the provided data. Usually, the objectives are given by your manager or team, they may also be driven by your own research questions.

In this project, I set out to investigate how salary is influenced by different factors such as employment type, company's location, and years of experience. The goal is to uncover meaningful insights into one of today's most competitive and rapidly evolving job markets.

With the objectives clearly defined and the tools selected, this report will walk through the analysis process step-by-step. This report will follow a structured approach, starting with stating the problem and acquiring the data, moving through cleaning and exploratory analysis, and finally arriving at statistical testing and insights. Each section will highlight not only the technical steps but also the reasoning behind them, offering a practical guide for anyone beginning their journey into data analysis with Python.

IV. Prepare the necessary tools

Make sure that you have downloaded the necessary tools for the analyzing process. For Python, the basic tools required are Numpy, Pandas, and Matplotlib. Additionally, you can download SciPy for statistical functions and Seaborn for prettier plots.

Ensure that before you start the project, you are comfortable with Pandas data structures like DataFrame (Pandas 2-D array), Series (Pandas 1-D) array, and the methods associated with these structures. Additionally, be sure to understand basic Numpy methods such as np.arange, np.array, np.linspace.

Finally, import the Python's libraries into your code file before you start. This will include: Pandas, Matplotlib.pyplot, seaborn, numpy, and scipy.stats.

V. Data acquisition

You can gather your data from many sources. If you are working in a company, this can be given to you by your company's team. You can also crawl data from the internet, gather raw data from databases and APIs. However, we are generally provided with a csv file.
Fortunately, Kaggle provides us with thousands of datasets downloadable in a csv form, so after downloading the data from the site, we can simply read the data using one line:

```
df = pd.read_csv("./ai_job_dataset.csv")
```

This will automatically convert the csv comma separated file into a Pandas DataFrame. Now, you can manipulate the data using Pandas provided methods. Additionally, we will refer to DataFrame as (df) from now on.

VI. Initial schema check

Right after you load your data into a DataFrame, you should do an initial check on how the dataset looks. This includes using pandas ***df.shape()*** to check the dimensions of the data provided. ***df.columns()*** to check the columns of the DataFrame, ***df.head(n)*** and ***df.tail(n)*** to inspect the n first rows and n last rows of the dataset, and ***df.sample(m)*** to inspect m random rows from the provided data.
The reason why we use methods like ***df.shape(), df.tail(), df.head()***, instead of inspecting the entire dataset directly is because real-world datasets often contain thousands, or even millions, of rows and columns. Loading and reviewing the entire dataset at once would be overwhelming and error prone, so these functions allow us to quickly get a sense of the datasets structure, sample values, and potential issues, making it easier to plan the next steps in our analysis.

VII. Running the inspection:

We can inspect the dataset as follows:

```
shape = df.shape
columns = df.columns
dtypes = df.dtypes

# casting general object dtypes into generic python dtype
object_columns = df.select_dtypes(include=object) # select columns within df that contains
dtype of objects
for column in object_columns.columns:
    dtypes[column] = type(df[column][1])

print(
    ">>> Initial Structural Checks on Dataset: \n",
    "1. Shape: ", shape, "\n",
    "2. Columns: ", columns, "\n"
    "3. Data types: \n", dtypes, "\n\n"
)
```

This code performs an initial structural check on the dataset to understand the overall layout and data types:

- df.shape() returns the number of (rows, columns) in the dataset.
- df.columns() list out all the column names.

- df.dtypes() shows the data type of each columns (e.g. int64, float64, object)

Since columns with object data types can store mixed types (like strings, dates, or even lists), we extract these using select_dtypes(include=object) then loop through them to determine their actual Python types. Then, the final print statement will produce a summary as follows (see Appendix 1 for the full structural-check output)

From the output, we can observe that the dataset has a shape of (15000, 19), meaning it contains 15000 rows and 19 columns. These columns include key information like job titles, salaries, and experience levels, which will be crucial in addressing our research questions later.
The data types include strings, integers, and floats. Inspecting the head, tail, and random samples of the dataset, we can also see that the dataset appears clean: Job titles are grammatically correct, salaries are properly formatted as integers, and company names are consistently structured.
With this initial check complete and the dataset appearing clean, we can move on to the next step: checking for missing values.

VIII. Checking Missing Values:

Checking for missing values is a crucial part of data cleaning and preprocessing. Missing values can create inaccurate analysis, biased results, and faulty conclusions if left untreated. For example, missing values can distort summary statistics, disrupt correlations between variables, or cause errors in model training.
Pandas provides a simple way to detect missing values using the **.isna()** method. With just one line of code:

```
df.isna().sum()
```

We can get the missing values count for each column.
However, as a practice exercise and for the sake of clarity, I implemented a custom function using a for loop and list comprehension.

```
def check_for_missing_values(df: pd.DataFrame)->None:
    missing_counts = [(df[col].isna()).sum() for col in df]
    missing_counts = pd.DataFrame([missing_counts], columns = df.columns, index=(['Missing Values']))
    print(missing_counts.T, "\n\n")

check_for_missing_values(df)
```

Within this function:

```
missing_counts = [(df[col].isna()).sum() for col in df]
```

This list comprehension iterates through each column in the DataFrame and calculates the number of missing values in that column using `df[col].isna().sum()`. The results are then stored in a list, which is converted into a DataFrame for better readability. The output is as follows:

```
                        Missing Values
job_id                               0
job_title                            0
salary_usd                           0
salary_currency                      0
experience_level                     0
employment_type                      0
company_location                     0
company_size                         0
employee_residence                   0
remote_ratio                         0
required_skills                      0
education_required                   0
years_experience                     0
industry                             0
posting_date                         0
application_deadline                 0
job_description_length               0
benefits_score                       0
company_name                         0
```

Our initial data check has shown that our data has no missing value.

Since our dataset has no missing values and well-structured columns, we can move on into the next phase: **checking for duplicates.**

IX. Checking for duplicates

Another crucial step in data cleaning is checking for duplicates. Duplicate records within a data set can cause skewness in our distribution, affect the data integrity and therefore create statistical errors when analyzed. To create accurate and reliable analysis, it's a good idea to detect and remove any duplicates from the dataset before proceeding.

We can check for duplicates using Pandas ***df.duplicated()*** method. Below is a custom function I wrote that performs this check:

```python
def check_general_duplicates(df: pd.DataFrame) -> None:
    duplicates = df[df.duplicated(keep=False)]
    if duplicates.empty:
        print(">>> No duplicated rows\n\n")
    else:
        print(">>> Duplicated rows: \n", duplicates, "\n\n")
```

Here, df.duplicated(keep=False) returns a Boolean series where all instances of duplicated rows are marked as True. You can read about Numpy Boolean indexing or Numpy's Boolean-mask array to understand this Pandas functionality. Then, df[df.duplicated(keep=False)] filters and returns all such duplicated rows.
When executed, our program prints:

```
>>> No duplicated rows
```

This means that our dataset contains no duplicate records - a good sign of initial data integrity. Additionally, as a learning exercise, I also wrote two custom functions that check for duplicated job titles or job ids. Both printed **None**, which is great news. Knowing that our dataset is free of both missing values and duplicate records, we can move on to the next key step of data cleaning: **removing outliers**.

X. Exploratory data analysis

Exploratory Data Analysis (EDA) is a crucial but exciting step in a data analysis project. In this step, we will plot out our distributions, visualize patterns, and identify correlations, which allows us to detect any outliers or anomalies. This step is especially important when we work with variables such as age or income, since these distributions are often skewed.

X.1. Skewness in Data Analysis:

Skewness refers to the asymmetry of a data distribution.
The standard distribution for a dataset is called a normal distribution, where the distribution, when plotted as a histogram, creates a bell-curved shape. Here, the mean, mode, and median of the distribution are equal. This type of distribution is considered to have a skewness of zero.
When our distribution has a tail that is longer on the right-side, this distribution is positively-skewed, or right-skewed. In this case, ***mean > median > mode.***
When our distribution has a tail that is longer on the left-side, this is called negatively-skewed or left-skewed. Here, ***mean < median < mode.***
Skewness can affect our analysis because it can distort the *mean*, pulling it away from the center of the distribution. As a result, relying on the *mean* in a skewed dataset can lead to inaccurate analysis and flawed predictions.
To handle skewness in data analysis, we can use statistical techniques such as log transformation, square-root transformation, box-cox transformation, etc.
However, since our project is aimed at an introductory approach, we can simply use the ***median*** instead of the ***mean*** when analyzing skewed data. The median is a more robust parameter to measure the central tendency when the set has extreme values or abnormal distributions.

X.2. Univariate Exploration, Bivariate, and Multivariate Exploration

Univariate exploration is the process of exploring one variable at a time to find the anomalies or the structure of one variable. It involves plotting the distribution of the variable, describing its statistics, and highlighting the potential outliers using tests called the z-score test or IQR rules.
Additionally, we can also choose to explore the relationship between two variables, this is called bivariate exploration. To explore the relationship and anomalies within 3 or more variables, we can do a multivariate exploration.

Since our project mainly deals with analyzing salary relationships with other variables, it is reasonable to analyze the distribution of salary and remove any anomalies detected. A powerful tool for this is the boxplot.

X.3. Boxplot

A Box Plot is a powerful tool to describe the distribution of a variable and identify its potential outliers. A box plot displays:

- The median (the central line within the box).
- The first quartile (Q1) and the third quartile (Q3)
- The interquartile range (IQR): the distance between Q1 and Q3.
- Whiskers: which are lines that extend to the distribution largest and smallest values. In Matplotlib, whiskers extend to the distribution's Tukey's fences (additional information below)
- Outliers are points that are displayed outside the whiskers.

X.4. Tukey's Fence: Identifying Outliers:

The concept of Tukey's Fence - developed by American mathematician John Tukey - is a method to detect outliers in a dataset using the interquartile range (IQR). IQR is basically the range between Q3 and Q1. The formula is simple: IQR = Q3 - Q1
Any data point that lies outside the range [Q1−1.5×IQR, Q3+1.5×IQR] are considered potential outliers.
By plotting a box plot of salary, we can quickly spot these outliers and decide whether to keep or remove them depending on the goals of the analysis.

X.5. Matplotlib: Plotting in Python

Matplotlib is a widely used Python library for data visualization. It allows us to create plots that display our data distribution. In this project, we use Matplotlib to draw boxplots and other visualizations that help us explore and interpret our dataset.
In our data cleaning example, we used Matplotlib's boxplot to visualize the distribution of salary values. Here's how we do it:

X.6. Plotting the distribution using a box plot

```
def plot_box(salary: pd.Series)->None:
    Q1 = np.percentile(salary, 25)
    Q2 = np.percentile(salary, 50)
    Q3 = np.percentile(salary, 75)
    IQR = Q3 - Q1
```

We first compute the quartiles (Q1, Q2, Q3). Next, we compute the interquartile range (IQR) by taking Q3 - Q1.

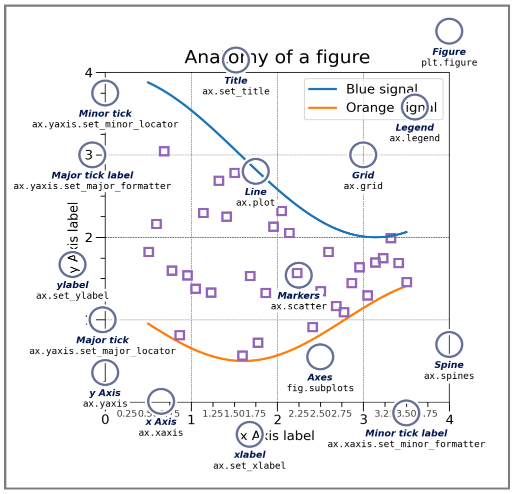
```
     # Boxplot
    fig, ax = plt.subplots()
    fig.set_figheight(5)
    bplot = ax.boxplot(
        salary,
        vert=False,
        patch_artist=True,
        boxprops=dict(linewidth=2.5, color='#555555'),
        whiskerprops=dict(color='#555555', linewidth=1.5),
        capprops=dict(color='#555555', linewidth=1.5)
    )

    for patch in (bplot['boxes']):
        patch.set_facecolor('#4C72B0')

    for median in bplot['medians']:
        median.set(color ='#DD8452',
                linewidth = 3)

    # changing style of fliers
    for flier in bplot['fliers']:
        flier.set(marker ='D',
                markerfacecolor ="#4C72B0",
                markeredgecolor = "#2F4569",
                alpha = 0.5)
```

Before jumping in. Make sure you have already imported Matplotlib using:

```
import matplotlib.pyplot as plt
```

Firstly, it's important to know how figures and axes work in Matplotlib.

Figure 1. Matplotlib's Figure Components



SEQ Source * ARABIC 1. Matplotlib's Figure Components. (n.d.). https://matplotlib.org/stable/users/explain/quick_start.html.

A figure is like a canvas; it's the entire window or page where everything is drawn.
It's also important to distinguish between Axes and Axis. Axes refers to the plot area - the space where your graph (boxplot, histogram, etc.) appears. It includes everything: the actual data, the gridlines, the labels, the title, and the x- and y-axis.
Axis refers to either the x-axis (horizontal) or y-axis(vertical) within one Axes.
If you are plotting only one plot without any subplots, the machine will only produce one axes for you to work on. However, when you create multiple subplots, you get multiple axes returned. A good explanation of the difference between Matplotlib's axes and axis can be found through this video: Subplot Tutorial.

When we create a plot using `fig, ax = plt.subplots()`, we are getting:

❖ fig: the overall figure or canvas
❖ ax: a single axes object - the area where you will draw your plot.

If we create multiple subplots, for example: fig, ax = plt.subplots(2,2), ax becomes a 2x2 array of axes objects, this means that we get 4 axes (sets of x- and y-axis) of 4 different subplots.
In our case, even though we only need one plot. Using plt.subplots() is still better. It gives us more flexibility since it gives us direct access to the axes object, which makes it easier to customize things like axis labels, gridlines, and plot elements later.
Next, we run: `fig.set_figheight(5)`, This sets the height of the subplot figure to 5 inches (Matplotlib's uses inches for sizing by default).

```
bplot = ax.boxplot(
    salary,
    vert=False,
    patch_artist=True,
    boxprops=dict(...),
    whiskerprops=dict(...),
    capprops=dict(...)
)
```

Here, salary is the distribution we want to plot. vert = False makes the boxplot horizontal. patch artist, fliers, whiskerprops, capprops, median are all just additional parameters to change the colors of the box plot's components like outliers and whiskers:

- ❖ patch_artist=True: Allows us to fill the box with color using a for loop as below.
- ❖ boxprops: Customizes the outline of the box.
- ❖ whiskerprops: Styles the whiskers, which extend from the box to the highest and lowest non-outlier values.
- ❖ capprops: Customizes the caps, which are the short horizontal lines at the ends of the whiskers.

```
#changing face color of the box
for patch in (bplot['boxes']):
    patch.set_facecolor('#4C72B0')

# changing style of median line
for median in bplot['medians']:
    median.set(color ='#DD8452',
            linewidth = 3)

# changing style of fliers
for flier in bplot['fliers']:
    flier.set(marker ='D',
            markerfacecolor ="#4C72B0",
            markeredgecolor = "#2F4569",
            alpha = 0.5)
```

After plotting the basic boxplot, we continue to enhance its readability and interpretability by adding gridlines, labeling the axis, customizing the ticks, and visually marking the Tukey fences (boundaries for outliers).
ax.grid(True) turns on grid lines in the background of the plot, this makes it easier for us to align our eyes with the values on the x-axis. Next, we set the label for the x-axis, weight = 'bold' makes the label bolded for reading:

```
ax.set_xlabel("Salary Distribution (USD)", weight="bold")
```

Next, we set and label the values for the x-ticks (markings on the x-axis):

```
ax.set_xticks([Q1, Q2, Q3])
ax.set_xticklabels([...], rotation=45, ha='right', weight='bold')
```

Since the y-axis ticks are not needed, we can remove them by setting:

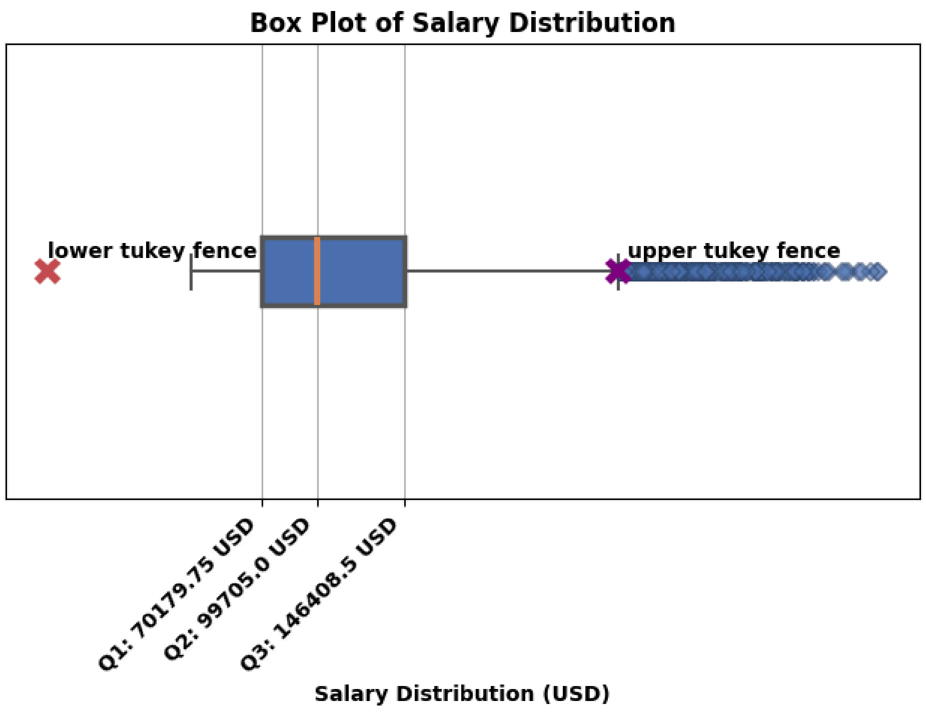
```
ax.set_yticks([])
```

Finally, we can calculate and plot the Tukey's Fences:

```
    ax.plot(lower_fence, 1, marker = 'x', mfc = "#C44E52", ms = 10, mew=4, mec="#C44E52")

   ax.text(float(lower_fence), 1.03, 'lower tukey fence', weight='bold')
    ax.plot(upper_fence, 1, marker = 'x', mfc = 'purple', ms = 10, mew=4, mec='purple')
    ax.text(float(upper_fence) + 5000, 1.03, 'upper tukey fence', weight='bold')
```

And using plt.tight_layout() to enable padding and prevent overlapping, we get:

Figure 2. Box plot of Dataset Salary Distribution



By plotting out the boxplot, we can see that the median salary for AI-related jobs in this dataset is approximately **$99,705 USD**, and most of the AI salaries fall between **$70k** and **$146k USD**. We can also see multiple potential outliers on the right side of the Tukey fences, indicating that the distribution is right-skewed.

While the box plot gives us a great summary of the distribution and outliers, it doesn't show the full shape of the data. To better understand how salaries are spread across the dataset, we can use a histogram. A histogram is particularly useful for visualizing the frequency of salary values. Histograms show us where most salary values are concentrated and whether the data is skewed, uniform, or bimodal (M shaped curve distribution). Here's the code that I used to plot the histogram of the salary distribution:

```python
def plot_hist(salary: pd.Series) -> None:
    fig, axes = plt.subplots(figsize=(12,8))

    # Histogram
    counts, bins, _ = axes.hist(
        salary, 100, color="purple", edgecolor="black", alpha=0.1, label="salary
distribution"
    )

    axes.set_xlabel("Salary", weight="bold")
    axes.set_ylabel("Occurrences", weight="bold")
    axes.legend()
    axes.set_title("Salary Distribution", weight="bold")

    # Plotting the frequency polygon
    bin_midpoints = [(bins[i] + bins[i + 1]) / 2 for i in range(len(bins) - 1)]
    axes.plot(
        bin_midpoints,
        counts,
        marker="o",
        linestyle="-",
        mfc="blue",
        color="black",
        ms=5,
    )

    axes.set_xticks(bin_midpoints[::2])
    axes.set_xticklabels(
        [f"{m}" for m in bin_midpoints[::2]],  # or just `bin_midpoints`
        rotation=45,
        ha="right",# align right so they don't overlap
    )
    axes.tick_params(axis = 'x', labelsize=8)

    # Calculate and print skewness and kurtosis
    skew = salary.skew()
    kurtosis = salary.kurtosis()
    print(
        ">>> Skewness of Distribution: ", skew, "\n",
        ">>> Kurtosis of Distribution: ", kurtosis
    )

    if(kurtosis > 3.0):
        print('--> Leptokurtic')
    elif (kurtosis == 3.0):
        print('--> Mesokurtic')
    else:
        print('--> Platykurtic')

    # Adjust layout to prevent overlap
    plt.tight_layout()
```

In this code, I again use plt.subplots() to plot the histogram. Since we don't specify the layout, it defaults to a single subplot, giving us a 1x1 Axes object.

```
counts, bins, _ = axes.hist(
        salary, 100, color="purple", edgecolor="black", alpha=0.1, label="salary distribution"
)
```

Here, axes.hist() returns 3 things:

- ❖ **Counts** are the number of data points that fall into each bin (height of each bar).
- ❖ **Bins** are the edges of the bins (the range of salary values of each bar).
- ❖ _: the third item is the list of rendered bars (called rectangle objects by Matplotlib). Which isn't needed here, so we ignore it by using _.

These three variables are used to plot our frequency polygon. Frequency polygons, like histograms, are used to show how frequently values occur in different ranges. However, frequency polygons can be placed overlapping one another, and are used to compare the frequency distribution across many variables, allowing us to identify different trends, correlation, etc.

After plotting the frequency polygon, I position the x- and y-ticks of the plot and set their labels. Next, I printed out the distribution skewness and kurtosis.

Kurtosis is a measure that describes the shape and the 'plateau' of a distribution curve. It tells us about the tailed-ness of the distribution. Normal distribution has a kurtosis of 3 and is referred to as mesokurtic. A kurtosis greater than 3 means the distribution is leptokurtic, it has sharper peaks and fatter tails. A kurtosis less than 3 means the distribution is platykurtic, it has flatter peaks and thinner tails, with fewer outliers. Even though kurtosis is not needed in our case, it's still a good thing to know what our distribution kurtosis is.

Running the block of code, we get the following output:

```
>>> General Numeric Stats:
           salary_usd  remote_ratio  years_experience  job_description_length  benefits_score
count   15000.000000  15000.000000      15000.000000            15000.000000    15000.000000
mean   115348.965133     49.483333          6.253200             1503.314733        7.504273
std     60260.940438     40.812712          5.545768              576.127083        1.450870
min     32519.000000      0.000000          0.000000              500.000000        5.000000
25%     70179.750000      0.000000          2.000000             1003.750000        6.200000
50%     99705.000000     50.000000          5.000000             1512.000000        7.500000
75%    146408.500000    100.000000         10.000000             2000.000000        8.800000
max    399095.000000    100.000000         19.000000             2499.000000       10.000000

>>> Standard salary deviation:  60260.94043824809
>>> Skewness of Distribution:  1.2527592133197596
>>> Kurtosis of Distribution:  1.56922226950801
```

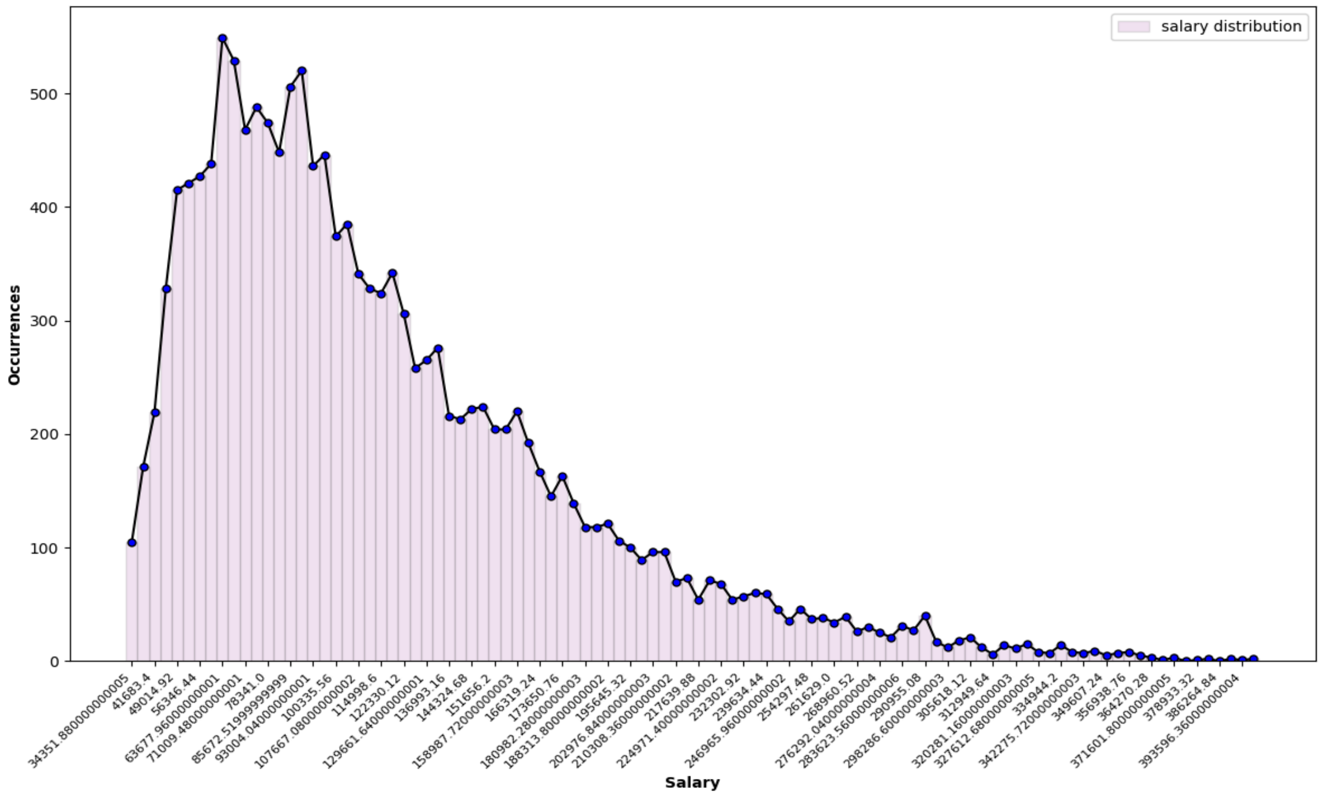
```
--> Platykurtic
Q1: 70179.75
Q3: 146408.5
IQR: 76228.75
```

Here, a skewness of 1.25 indicates that the distribution is right-skewed (positively skewed). This means that **most people earn under the mean, but a small number of high earners pulls the average up.**

A kurtosis of 1.57 suggests that this distribution is platykurtic. This means that the distribution is flatter than a normal curve (less peaked) and has a lighter tail. Next, running plt.show(), we get the following distribution:

Salary Distribution:

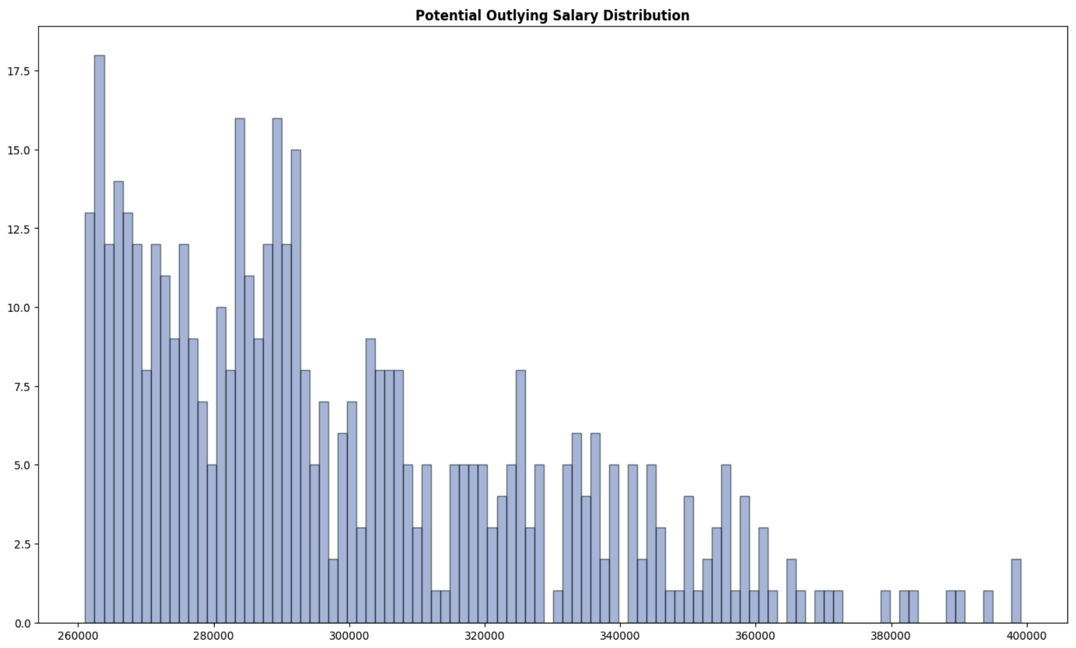
Figure 3 . Salary Distribution of AI Job Dataset



From the histogram, we see that most of the salary lies in the IQR range [70k, 146k], above this salary range are potential outliers. The most extreme value we could see lies around the [386k, 393k] range. It's also better to visualize these potential outliers' distribution.

We should also plot out the distribution of the outlying salary using a histogram:

Figure 4. Potential Outlying Salary Distribution of AI Job Dataset



From this potential outlying distribution, we can see that there are few counts that are around 400k, most of the counts are around **$270k to $280k** USD.

Since outliers can either be real data or false data, they can represent valid but rare observations. Therefore, before deciding to transform or remove them from the dataset, it's important to examine them more closely.

After examining the job experience levels and roles associated with the highest salaries. We could see that all these salary records belong to **Executive-level** job roles, suggesting that these are **valid data points** rather than **anomalies or input errors.** Therefore, it would be reasonable to retain these data points.

X.7. Dropping Redundant Columns

Since our overall review of the dataset is good, we can drop the redundant columns that don't necessarily provide us with the information we need for our objectives.

Since our objectives are concerned with salary relationship with:

- ❖ Experience level
- ❖ Employment type
- ❖ Remote Ratio
- ❖ Company Size
- ❖ Company Location
- ❖ Top required skills,

We can drop the other columns that aren't related like job posting date, education required, etc.

We can do this as follows:

```
to_drop = [
  'job_id',                # just an identifier
  'salary_currency',       # already have salary_usd
  'employee_residence',    # not used in any of the 4 analyses
  'posting_date',          # only needed if we do time-series or "time_to_deadline"
  'application_deadline',  # ditto
  'job_description_length',# unrelated
  'benefits_score',        # not in our objectives
  'company_name',          # too granular--you're grouping by location/size
  'education_required',    # outside our scope
  'industry'               # unrelated
]
df_clean = df.drop(columns=to_drop)
```

Finally, we can export the newly cleaned data into a csv file for later analysis:

```
df_clean.to_csv('ai_job_dataset_cleaned.csv', index=False, sep=',')
```

With our dataset now cleaned, visualized, and trimmed to include only the most relevant columns, we're ready to begin analyzing the relationships between salary and other key factors.
In this next section, we will perform targeted visual and statistical explorations to answer the main questions posed in our objectives:

- ❖ How does experience level impact salary?
- ❖ What is the salary distribution across different employment types?
- ❖ Does remote work flexibility influence compensation?
- ❖ Are company size and location correlated with pay?
- ❖ What are the top required skills in the AI job market in 2025?

To do this, we'll use group comparisons, box plots, count plots, and simple summary statistics to guide our insights.

XI. Employment type vs. Salary

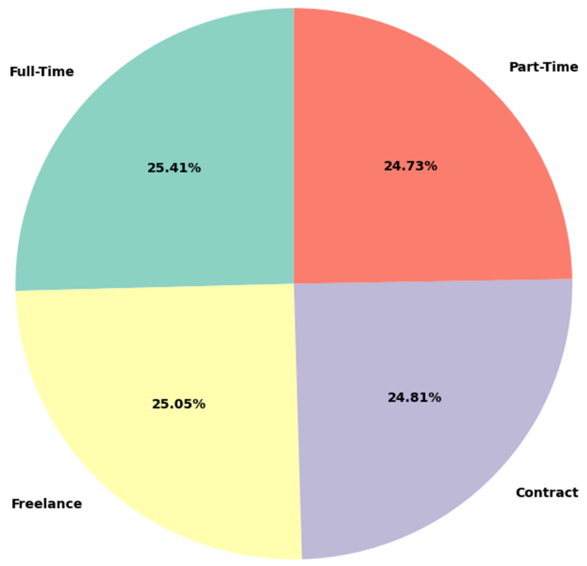
First question we want to know the answer to is whether employment has a significant impact on job income. In this analysis, we will be using two columns: salary_usd and employment_type
We will inspect the dataset and missing values as usual. Since we already checked for duplicates when cleaning the data, checking for duplicates within these two columns is unnecessary as we already know that each and one of the rows are unique records.

Next, we would want to replace the employment types for easier reading:

```
# Renaming the Employment Types:
df['employment_type'] = df['employment_type'].replace({'CT': 'Contract', 'FL': 'Freelance',
'PT': 'Part-Time', 'FT': 'Full-Time'})
```

First, it's a good idea to understand the percentages of each employment type. We can visualize this using a pie chart (`employment_type_vs_salary.py` lines 40-49):

Figure 5. Percentages of Employment Types within the AI Job Dataset



Our distributions are fairly equal across each employment type. Next, we calculate the mean and medians of the groups and plot out their distributions (`emp_ty…` lines 69-152):

Figure 6. Histogram Distribution of Salary by Employment Type

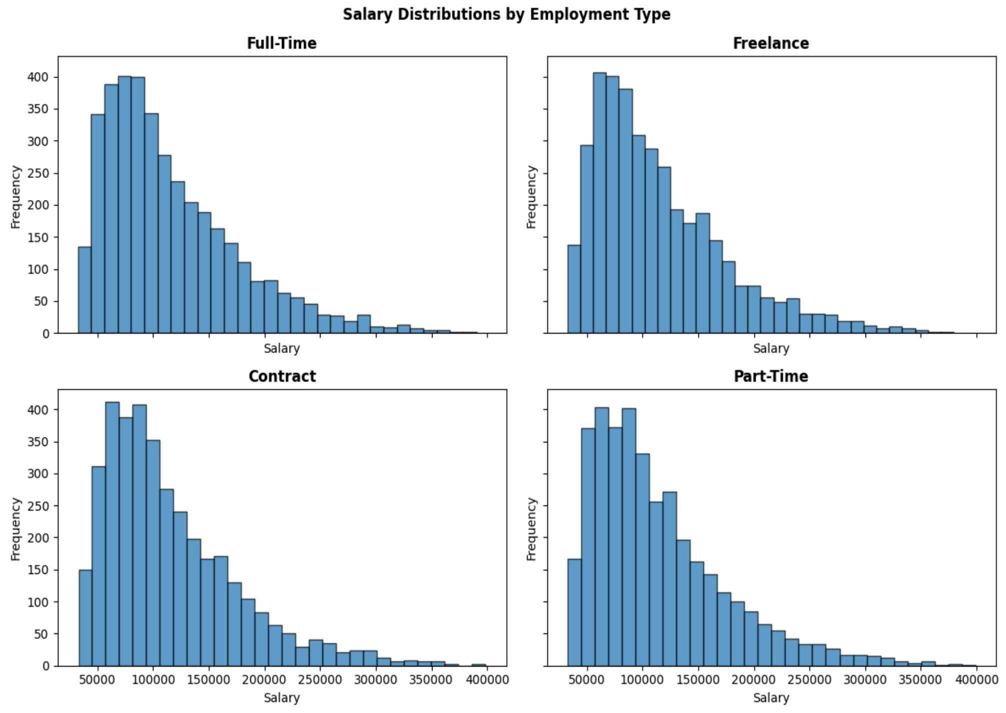


Figure 7. Box plot Distribution of Salary By Employment Type

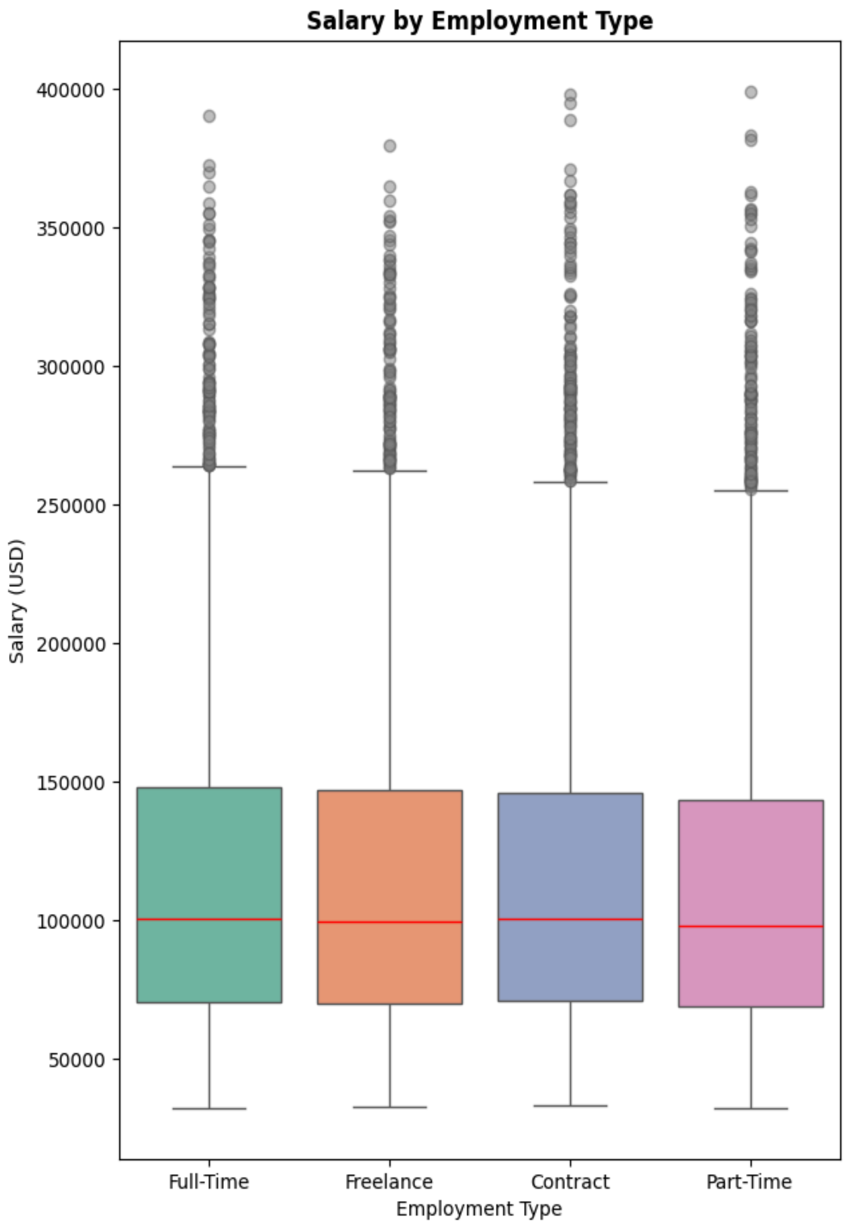
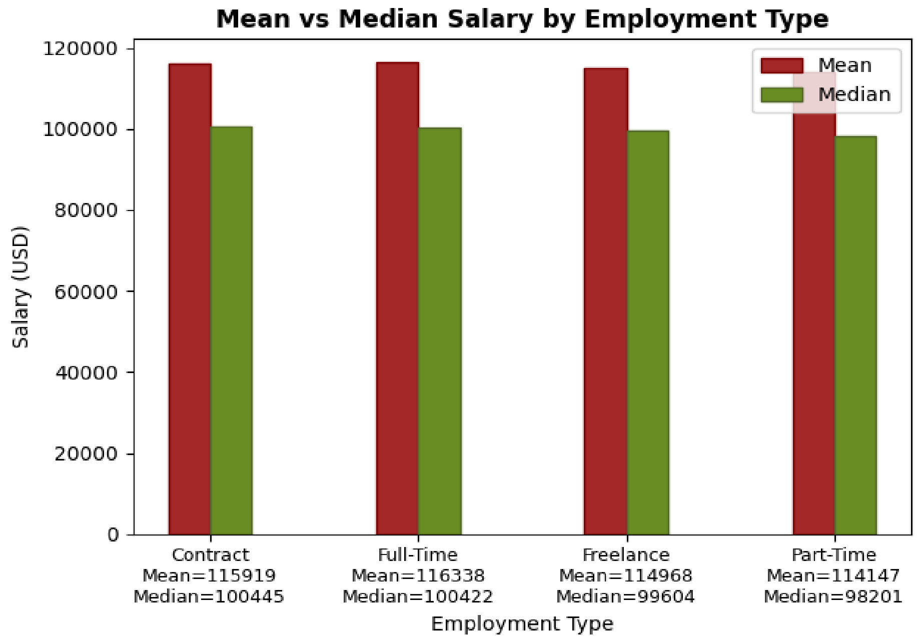


Figure 8 . Bar chart Distribution of Mean and Median Salary by Employment Type



From the plots, we could see that the median salaries across employment types appear similar. **This indicates that employment type might not have a significant impact on the salary paid.** However, it is still important to support our observation with statistical testing.

XI.1. One way ANOVA and why we can't use it

One way **ANOVA (Analysis of Variance)** is a statistical method used to determine whether there are significant differences between the **means (μ)** of three or more independent groups. If there is any significance, it will return a p-value that indicates the result. However, one-way ANOVA assumes that:

- ❖ The data within each group is normally distributed.
- ❖ The variances across groups are equal (homogeneity of variance).
- ❖ Observations are independent.

From our plots, we could see that:

- ❖ Salary distribution is not normal across each group (The distributions are right-skewed within each employment type)
- ❖ Homogeneity of variance is violated: box plots across each group have different whisker lengths and different outlier distributions, this means that one group has a larger variance compared to the other.

So, by intuition, since the whiskers and outliers are different between groups, this proves that variance between each group is different, which violates the homogeneity of variance, therefore we cannot run the one-way ANOVA.

Because of these violations, we use the Kruskal-Wallis H-test, a non-parametric alternative to ANOVA that does not require normality, equal variances, and works by comparing medians (as ranks) instead of means.

The H-test works by:

1. Group every group element into one list.
1. Sort that list.
2. Apply a rank order on each element based on ascending order.
3. For each group, sum the rank of the elements that belong to each group, and we will get the 'rank' of the group.
4. Finally, compare and see if there's any significance between each group's median via computed rank.

XI.2. Interpreting Hypotheses and p-values

A hypothesis is any assumption we have before we run the test. In statistics, there are 2 main types of hypothesis:

- **H0 (Null Hypothesis)**: the hypothesis of minimal variance. We assume that the variance, or the effect of one group on the other is minimal.
- **H1 (Alternative Hypothesis)**: the hypothesis that contradicts the Null Hypothesis. This assumption says that the variance between each group is large, and the effect of one group on the other is considerable.
- **p-value** is a test result that we can use to determine whether H0 or H1 is the correct assumption. If a p-value is > 0.05, this means that H0 is likely the correct assumption. Else if p-value is ≤ 0.05, this indicates that H1 is likely the correct assumption, and there's a significant difference between the median of one or two more groups.

XI.3. Running the test

Using the Kruskal Wallis test provided on Python's SciPy library, we can run the test on the groups we want to compare with (see 'emp…` lines 167-193):

```
stat, p_value = stats.kruskal(*data) # * is ungrouping operator in Python
```

Here, stat is the H value produced by running the test. It measures the degree of difference between the rank distributions of each group.
A small H-value suggests that the groups are similar and supports H0.
A larger H-value suggests that there is a greater difference between groups and supports H1.
Still, we cannot use only H to determine the statistical difference. H only gives us an intuitive sense of the result. We need to calculate the p-value using H to wind up our conclusion.

XI.4. Measuring the effect size

Effect size quantifies the magnitude of the difference between groups. Even if a p-value is small (statistically significant), the effect might still be very small in real-world terms. Conversely, a large effect size can indicate a meaningful difference, even if the p-value is just above the threshold.

In the case of the Kruskal-Wallis test, a common measure of effect size is **epsilon-squared (ϵ^2)**. ϵ^2 ranges from 0 to 1:

- 0.01 to < 0.06: Small effect,
- 0.06 to < 0.14: Moderate effect,
- ≥ 0.14: Large effect.

XI.5. Printing the results

After printing our statistics, we get:

```
>>> H:  4.38866186021197
>>> p-value:  0.22243900160091176
>>> epsilon-squared:  9.260215125446585e-05
>>> H0 can be true <=> No significant variance
>>> Effect size is small
```

The p-value of 0.22 indicates that we fail to reject the null hypothesis, and the extremely small epsilon-squared value (~0.00009) suggests that any practical difference groups are negligible.

XI.6. Final conclusion

From the statistical results and the distribution of salary we have plotted, we can conclude that **employment type does not significantly impact the salary levels for AI-related jobs.**

While the result may not reveal a striking difference between employment types, the process of reaching this conclusion is valuable. Through visualizations and statistical testing, we've not only examined a real-world question but also developed the ability to critically evaluate group differences. By combining **exploratory plots** with **non-parametric testing**, we learned to offer a clear, data-driven way to interpret salary dynamics.

XII. Location vs. Salary Analysis

To assess whether pay differs between countries, we use two columns: 'salary_usd' and 'company_location'. We loaded the data and did an initial inspection for missing values and duplicates. Using df['company_location'].unique(), we identified 20 countries with at least one AI job listing.

To analyze income by country, we need to group salaries by company's location. Pandas provides us with a categorical grouping tool that is very robust. We can group data based on the categories they belong to. We can implement the df.groupby() method as follows:

```
df_stats = data.groupby(['company_location'])['salary_usd'].agg(['mean', 'median', max])
print(">>> Initial Statistics: \n", df_stats.sort_values(by='median', ascending=False))
```

- df.groupby([‘company_location]) splits our dataframe into one subgroup per unique country in company_location.
- Select the [‘salary_usd’] column from each of those country-based subgroups.
- .agg([‘mean’, ‘median’, max]): pass a list of functions we want to compute on each subgroup's salaries.

Running the code gives us the following table:

```
company_location          mean        median     max
Switzerland       170639.085791  152901.0  399095
Denmark           165652.249357  147504.5  379418
Norway            159490.533981  142386.0  362503
…                       …             …          …
```

Next, we examine the salary distribution. We plot out the data using histograms and we get the following distributions (see Appendix 2). We could see that the salaries distributions across countries are heavily right-skewed. Therefore, we cannot use mean as a parameter for comparison. Running the Kruskal-Wallis H-test on the dataset, we get the following results:

```
================
>>>> H Statistics = 3296.9062925927196
>>>> p-value = 0.000e+00
================
```

This indicates that there's a highly significant difference in salary distributions across countries. To rank countries by median pay under heavy right skewness, we employ a nonparametric bootstrap to quantify each median's uncertainty.

XII.1. The bootstrap procedure

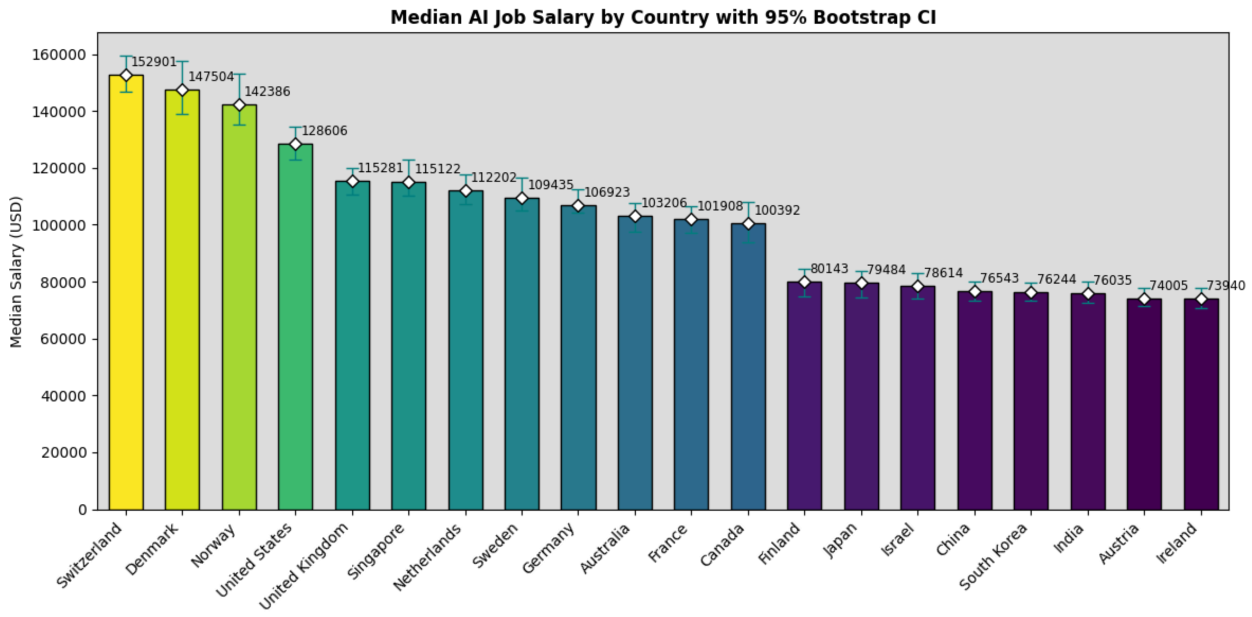
The Bootstrapping method (Efron, 1979) is a data-driven resampling method that approximates the sampling distribution of any statistic: mean, median, percentiles, etc. without assuming normality. For each country, we draw 2,000 bootstrap samples (sampling with replacement from the observed salaries), compute the median of each sample, then take the 2.5th and 97.5th percentiles of those 2,000 medians to form an empirical 95% confidence interval (see 'location_vs_salary.py' lines 102-214):

```
def bootstrap_ci(x, stat=np.median, number_of_repeats=2000, alpha=0.05):
    boot_stats = np.array([
        stat(np.random.choice(x, size=len(x), replace=True)) for _ in range(number_of_repeats)
#bootstrap sampling B number of times.
    ])
    lower, upper = np.percentile(boot_stats, [100*alpha/2, 100*(1-alpha/2)]) # calculate the
lower/upper CI intervals.
    return lower, upper
```

These bootstrap confidence intervals transparently display how much each median might fluctuate if we repeated the survey under identical conditions, making our country-by-country comparisons more informative.

After computing the CI intervals and plotting the salary rankings based on the computed medians, we get the following distribution:

Figure 9. Median AI Job Salary Ranking with 95% Confidence Interval



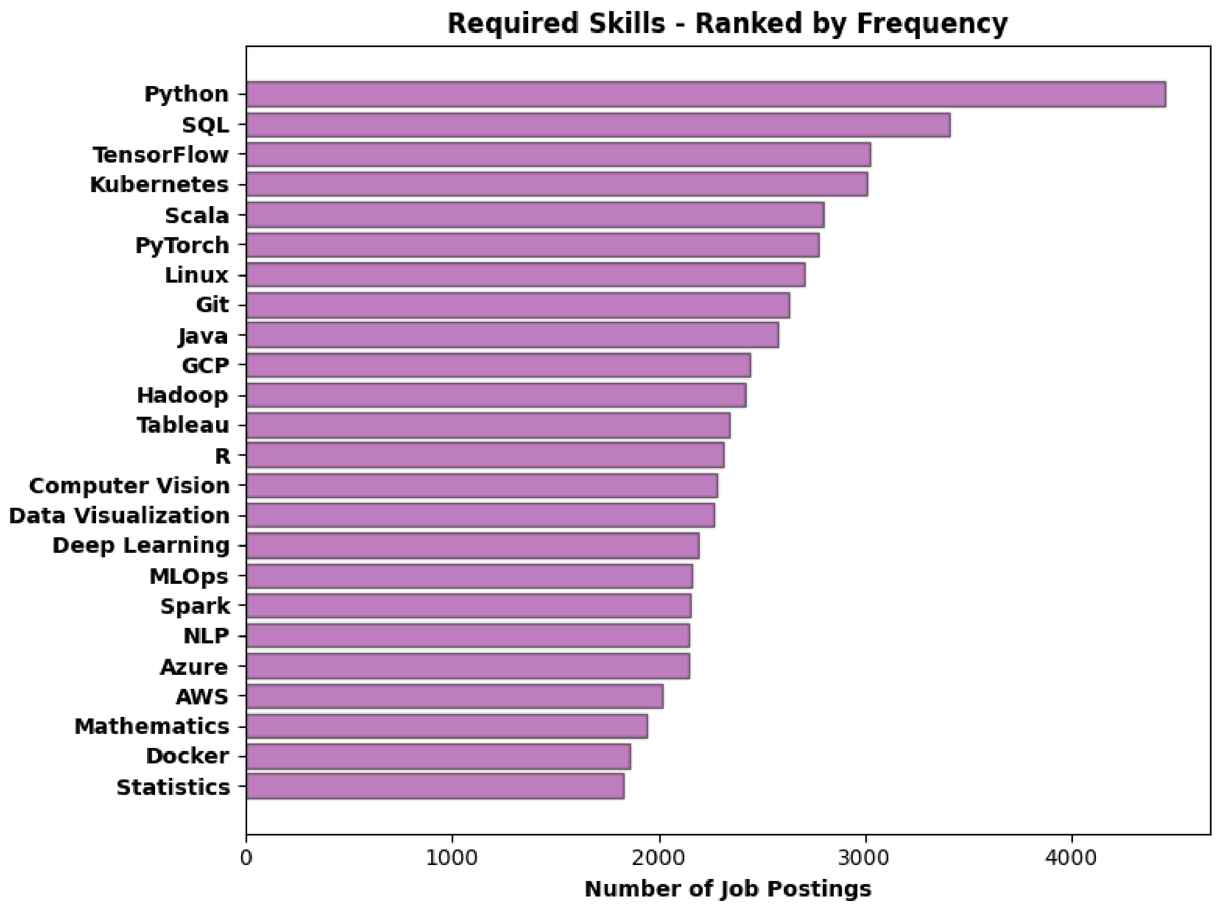
This approximates the rankings in the median AI job salaries by countries. Each bar shows the median salary for that country with a 95% bootstrap confidence interval. However, this is only intended to give us an intuitive look at how salary differs across each country and which country approximately pays more than the other. As we can see Switzerland has the highest pay since its upper bound and lower bound are approximately higher than Denmark. The variability between these groups is not definitive, since we can see that many countries' CIs overlap one another. To formally compare and rank one group to the other, we need to further run complex pairwise comparisons, for example Dunn's test with Bonferroni correction.

Nevertheless, this bootstrap approach establishes a more transparent view of how salaries are distributed across different countries.

XIII. Skills Analysis

To analyze the required skills in the data set, we will use the 'required_skills' column in the data frame. Each row in this column contains a string that lists the required skills. We will start by splitting these strings, organizing all the skills into a list, and then counting the number of occurrences for each unique skill.
Once we have the total counts of different skills, we can use a box plot to visualize the distribution of the skills (for code, see 'skills_analysis.py').

Figure 10. AI Job Market Top Required Skills Ranked by Frequency



From the distribution we can see that **Python** is the most frequently required skill, followed by **SQL**, **TensorFlow**, and so on.

XIV. Experience Level

After calculating the initial salary statistics grouped by experience level, we can see a clear difference across different experience levels. To confirm this, we can plot out a box plot containing the means and medians of the salary distributions (see Appendix 3). Additionally, I created a scatter plot to see how salaries are distributed within each level (see code 'experience_level_vs_salary.py' lines 41-52 and 62-93).

From the plot, we conclude that there is a significant difference in pay:

- ❖ Executive-level roles are the ones that get paid the highest.
- ❖ Followed by Senior-level roles.
- ❖ Followed by Mid-levels.
- ❖ Finally, Entry-level roles are ones that get paid the least.

XV. Company's Remote Ratio

Plotting out the salary distributions across each group we see there are no significant differences between the median incomes. Again, we ran an H-test over the groups to confirm this.
We get an H-value of 4.1 and a p-value of 0.127. This p-value is significantly larger than 0.05, thus, we can conclude that the H0 is true and therefore there is no significant difference between the salary groups. Therefore, remote-ratio does not impact income in this dataset.

XVI. Companies Size:

Similar experience level, we plot out the distributions across each company size (see Appendix 4). We can see that there is a significant difference between company sizes and income and therefore we can conclude that there is a difference between pay across company sizes.

XVII. Years Experienced:

To explore how years of experience affect salary, we used a Seaborn scatterplot with a regression line along with a heatmap (code reference).

Figure 11. Seaborn Scatter Plot for Salary Distribution by Years of Experience with Regression line (0.0 years, 2.5 years, 5.0 years, ...)

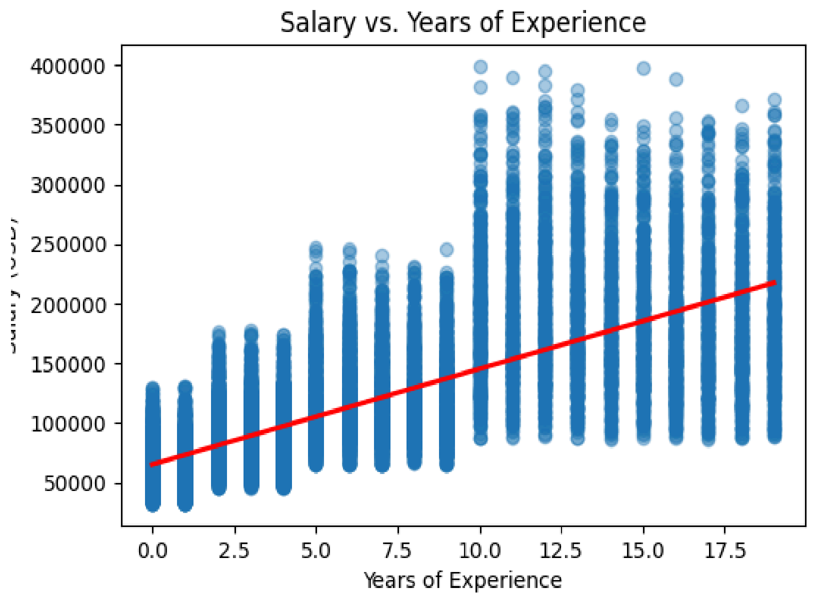
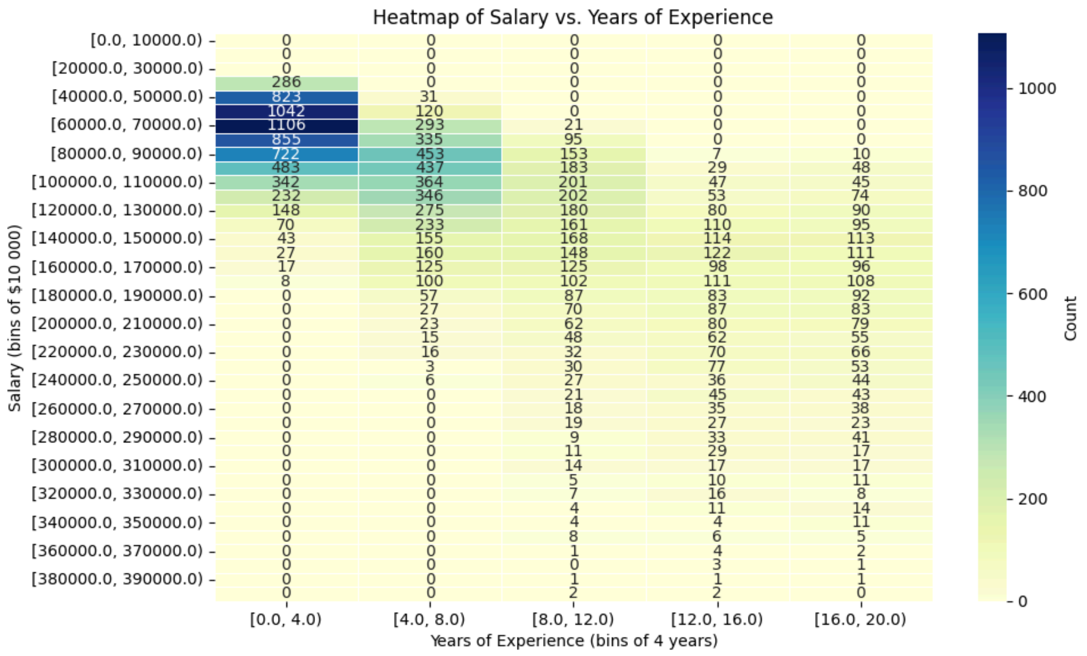


Figure 12. Seaborn Heatmap for Salary by Years of Experience



From both plots and using the Spearman's Rho correlation coefficient, we confirm that there is a strong positive correlation between years of experience and income.

XVIII. Conclusion

XVIII.1. Key takeaways

For this project, we implemented various techniques to analyze data via a robust pipeline:

- ❖ **Cleaning and Exploration:** Systematic checks (shape, dtypes, missing/duplicate values) and EDA via boxplots and histograms to ensure data integrity before analysis.
- ❖ **Nonparametric Testing:** Using Kruskal-Wallis H-test to analyze data correlation when normality or homogeneity of variance is violated.
- ❖ **Bootstrapping:** Applied the bootstrap resampling technique to calculate 95% confidence intervals for salary medians, enabling more transparent comparisons.

XVIII.2. Key Findings

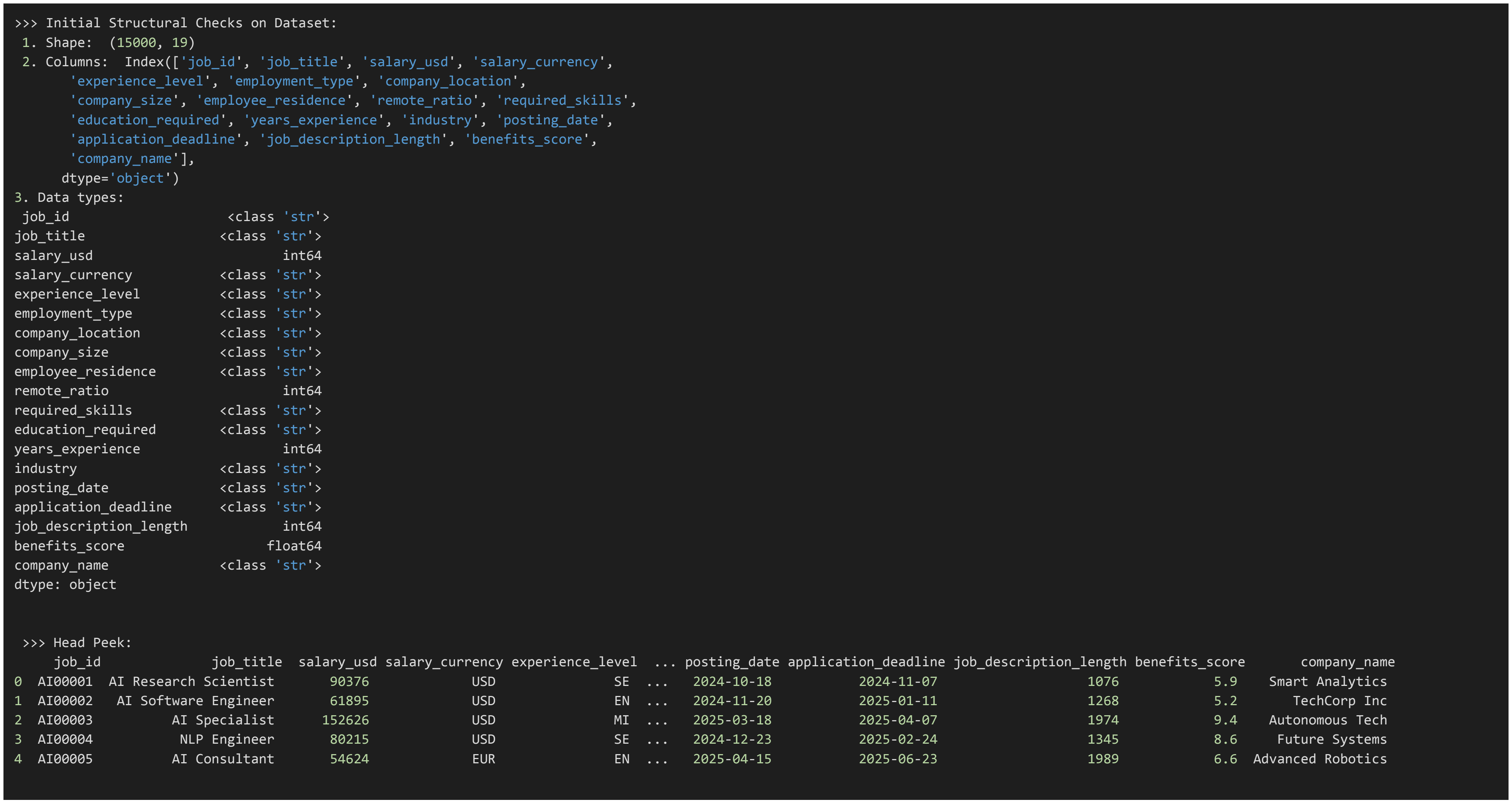
- Employment type shows no significant pay difference (p-value ~ 0.22).
- Plotted a transparent ranking of pay over each country using pay medians and bootstrap CIs. Switzerland is likely the highest paying country, followed by Denmark , Norway, the United States and others.
- Python is the most required skill in the market. SQL is the second highest in demand, followed by Tensorflow, Scala, Kubernetes, and so on.
- Executive-level job roles earn the highest, followed by Senior, Mid-levels, and Entry-levels job roles.
- Large companies pay more on average compared to Medium and Small-sized companies.
- Using Seaborn's scatter plot with a regression line and applying spearman's Rho to analyze the correlation effect. We found a strong positive correlation between years of experience and income.

XVIII.3. Reflection and Next Steps

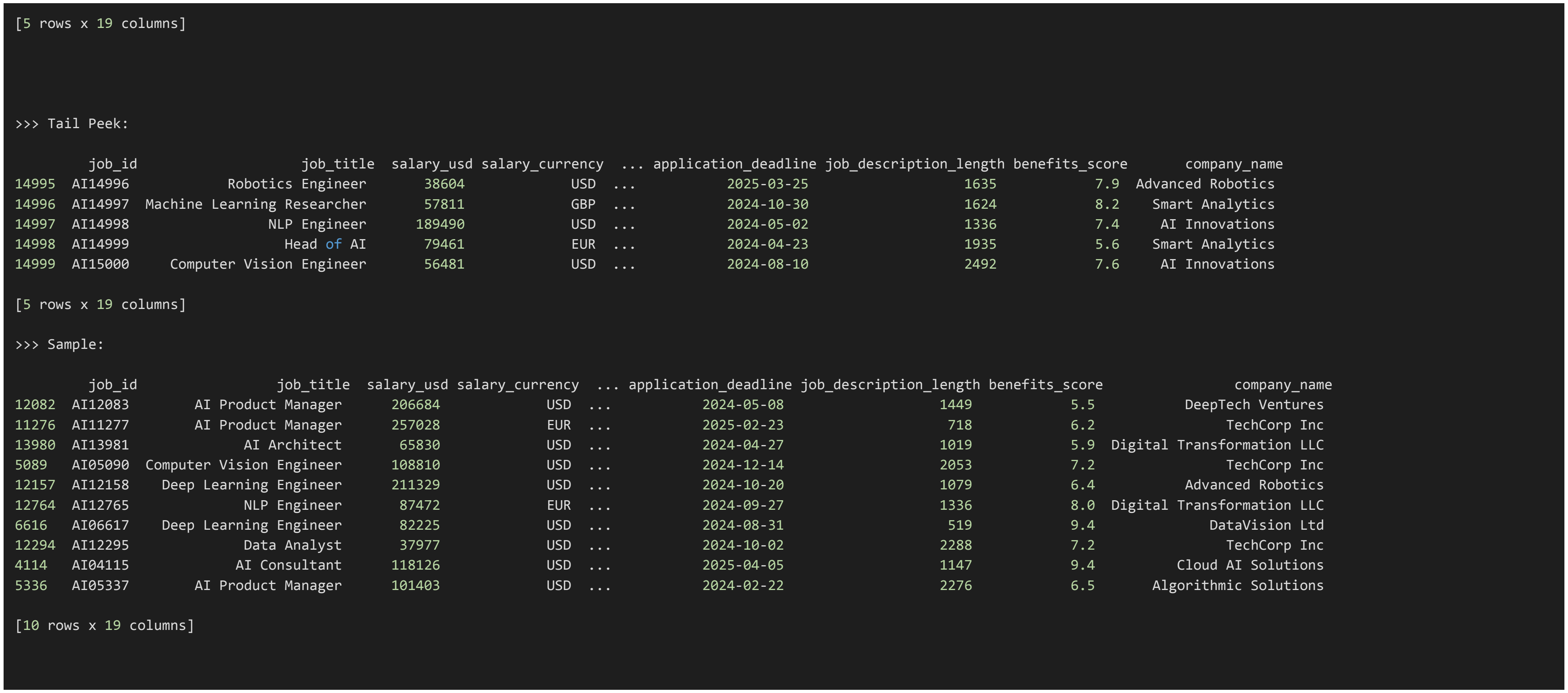
This project has solidified my fluency with Python's data science tools and strengthened my grasp of applied statistics. Going forward, I plan to incorporate better statistical pipelines, transform data and feed clean, featured-engineered datasets into ML systems.

Appendix 1

The following output of the initial structural check looks as follows:



```
>>> Initial Structural Checks on Dataset:
 1. Shape:  (15000, 19)
 2. Columns:  Index(['job_id', 'job_title', 'salary_usd', 'salary_currency',
       'experience_level', 'employment_type', 'company_location',
       'company_size', 'employee_residence', 'remote_ratio', 'required_skills',
       'education_required', 'years_experience', 'industry', 'posting_date',
       'application_deadline', 'job_description_length', 'benefits_score',
       'company_name'],
      dtype='object')
3. Data types:
 job_id                    <class 'str'>
job_title                 <class 'str'>
salary_usd                        int64
salary_currency           <class 'str'>
experience_level          <class 'str'>
employment_type           <class 'str'>
company_location          <class 'str'>
company_size              <class 'str'>
employee_residence        <class 'str'>
remote_ratio                      int64
required_skills           <class 'str'>
education_required        <class 'str'>
years_experience                  int64
industry                  <class 'str'>
posting_date              <class 'str'>
application_deadline      <class 'str'>
job_description_length            int64
benefits_score                  float64
company_name              <class 'str'>
dtype: object


 >>> Head Peek:
    job_id              job_title  salary_usd salary_currency experience_level  ... posting_date application_deadline job_description_length benefits_score       company_name
0  AI00001  AI Research Scientist       90376             USD               SE  ...   2024-10-18           2024-11-07                   1076            5.9    Smart Analytics
1  AI00002   AI Software Engineer       61895             USD               EN  ...   2024-11-20           2025-01-11                   1268            5.2       TechCorp Inc
2  AI00003          AI Specialist      152626             USD               MI  ...   2025-03-18           2025-04-07                   1974            9.4    Autonomous Tech
3  AI00004           NLP Engineer       80215             USD               SE  ...   2024-12-23           2025-02-24                   1345            8.6     Future Systems
4  AI00005          AI Consultant       54624             EUR               EN  ...   2025-04-15           2025-06-23                   1989            6.6  Advanced Robotics
```



```
[5 rows x 19 columns]



>>> Tail Peek:

        job_id                    job_title  salary_usd salary_currency  ... application_deadline job_description_length benefits_score       company_name
14995  AI14996            Robotics Engineer       38604             USD  ...           2025-03-25                   1635            7.9  Advanced Robotics
14996  AI14997  Machine Learning Researcher       57811             GBP  ...           2024-10-30                   1624            8.2    Smart Analytics
14997  AI14998                 NLP Engineer      189490             USD  ...           2024-05-02                   1336            7.4     AI Innovations
14998  AI14999                   Head of AI       79461             EUR  ...           2024-04-23                   1935            5.6    Smart Analytics
14999  AI15000     Computer Vision Engineer       56481             USD  ...           2024-08-10                   2492            7.6     AI Innovations

[5 rows x 19 columns]

>>> Sample:

        job_id                 job_title  salary_usd salary_currency  ... application_deadline job_description_length benefits_score                company_name
12082  AI12083        AI Product Manager      206684             USD  ...           2024-05-08                   1449            5.5            DeepTech Ventures
11276  AI11277        AI Product Manager      257028             EUR  ...           2025-02-23                    718            6.2                 TechCorp Inc
13980  AI13981              AI Architect       65830             USD  ...           2024-04-27                   1019            5.9  Digital Transformation LLC
5089   AI05090  Computer Vision Engineer      108810             USD  ...           2024-12-14                   2053            7.2                 TechCorp Inc
12157  AI12158    Deep Learning Engineer      211329             USD  ...           2024-10-20                   1079            6.4            Advanced Robotics
12764  AI12765              NLP Engineer       87472             EUR  ...           2024-09-27                   1336            8.0  Digital Transformation LLC
6616   AI06617    Deep Learning Engineer       82225             USD  ...           2024-08-31                    519            9.4               DataVision Ltd
12294  AI12295              Data Analyst       37977             USD  ...           2024-10-02                   2288            7.2                 TechCorp Inc
4114   AI04115             AI Consultant      118126             USD  ...           2025-04-05                   1147            9.4            Cloud AI Solutions
5336   AI05337        AI Product Manager      101403             USD  ...           2024-02-22                   2276            6.5        Algorithmic Solutions

[10 rows x 19 columns]
```

Output 1. Initial Dataset Structural Check

Appendix 2

The salary distribution across each country:

Figure 1:

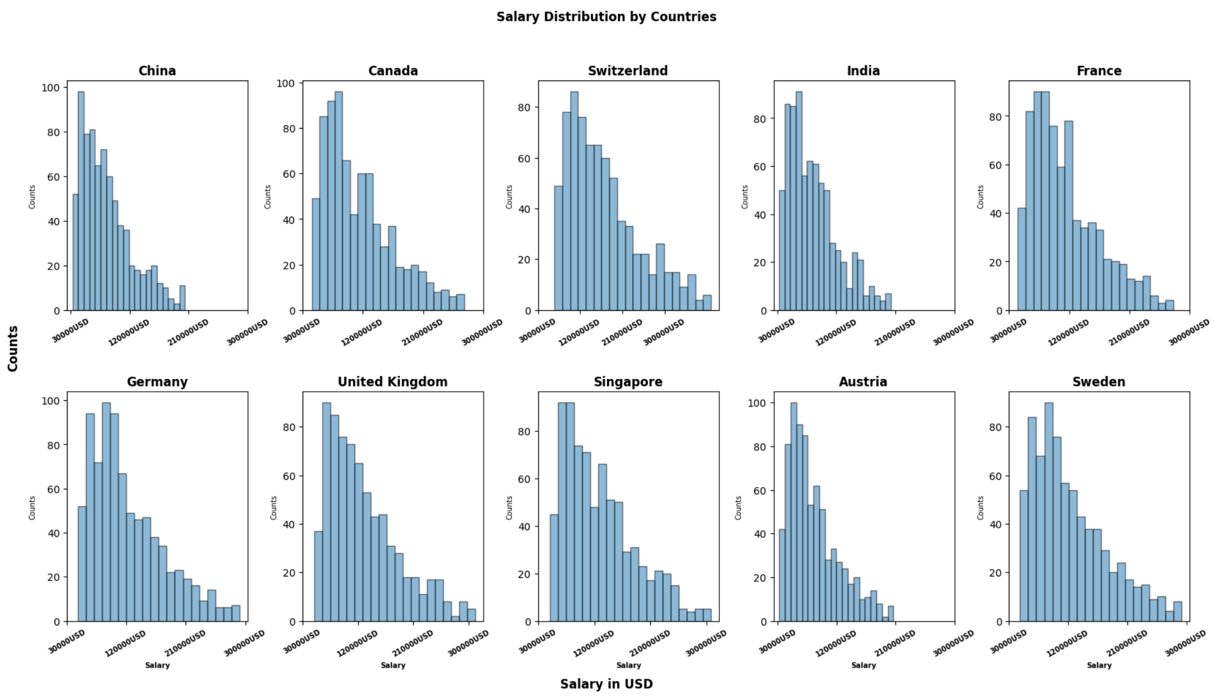
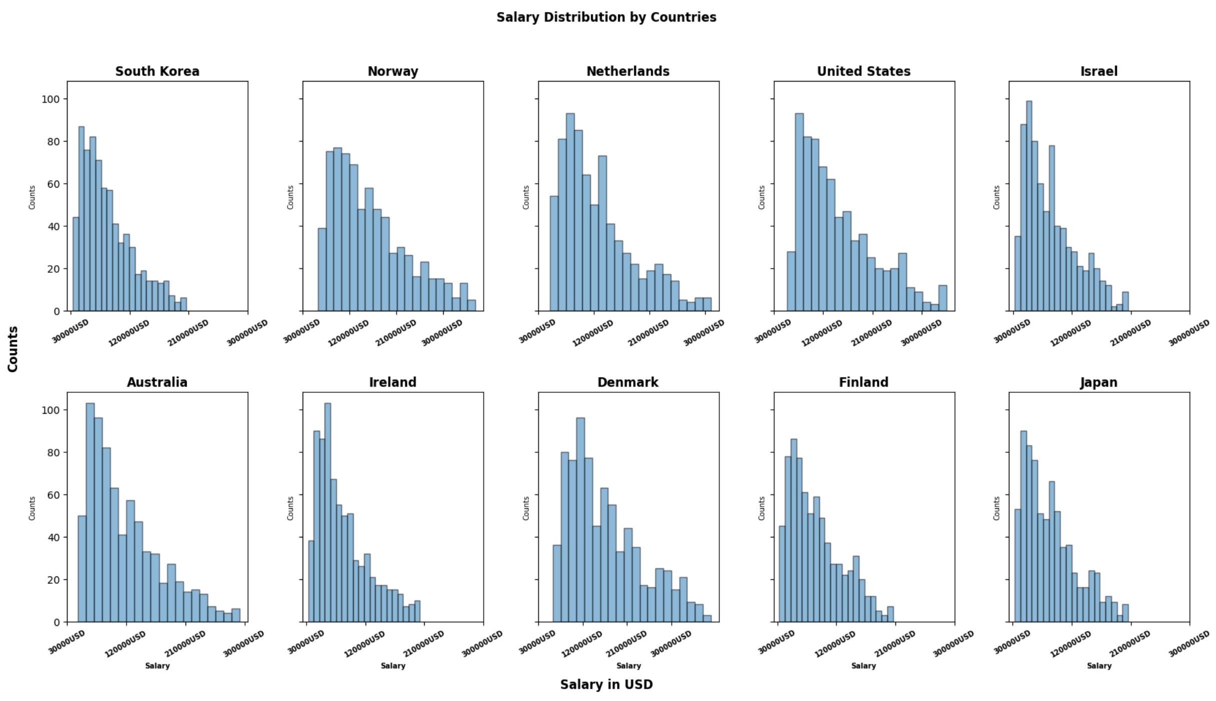
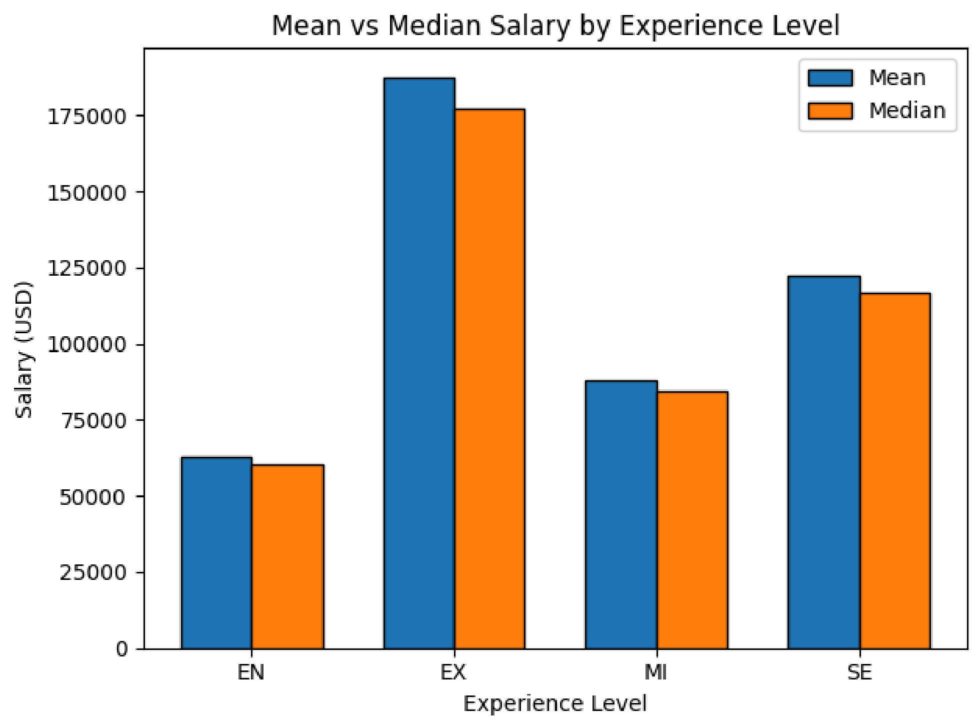


Figure 2:

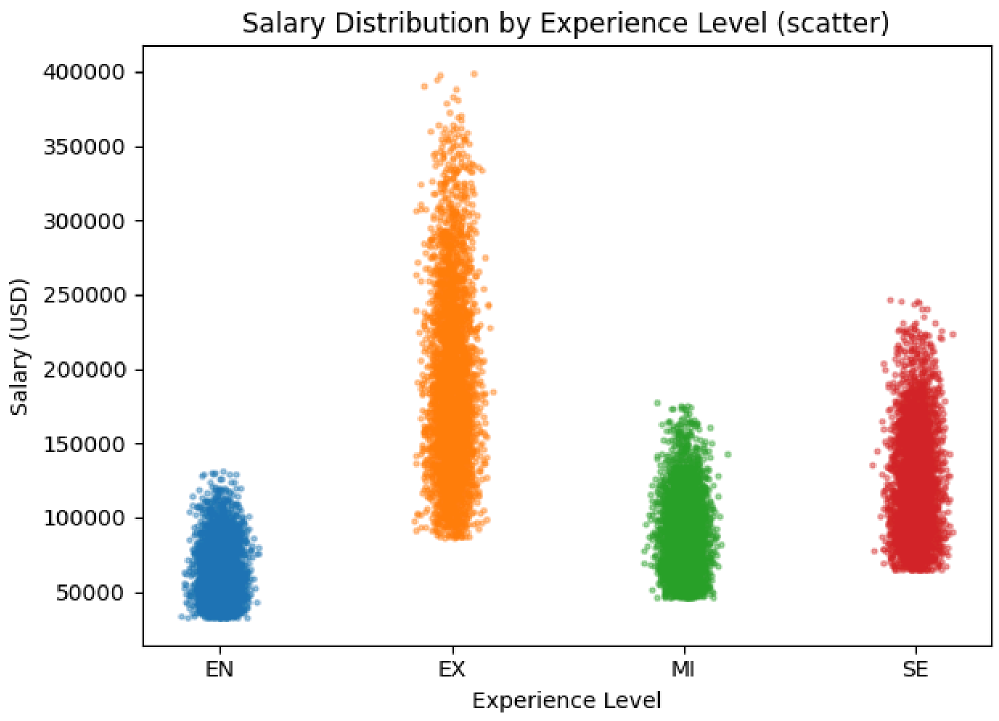


Appendix 3

Mean and Median distribution by Experience level (EX: executive-level, EN: entry-level, MI: mid-level, SE: senior-level):

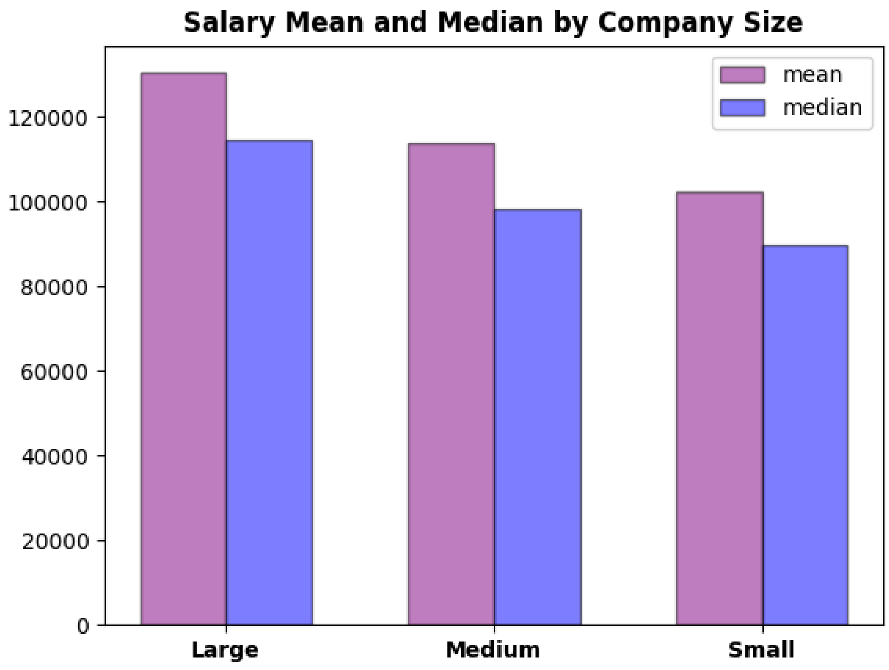


Scatter plot of Salary Distribution by Experience level:

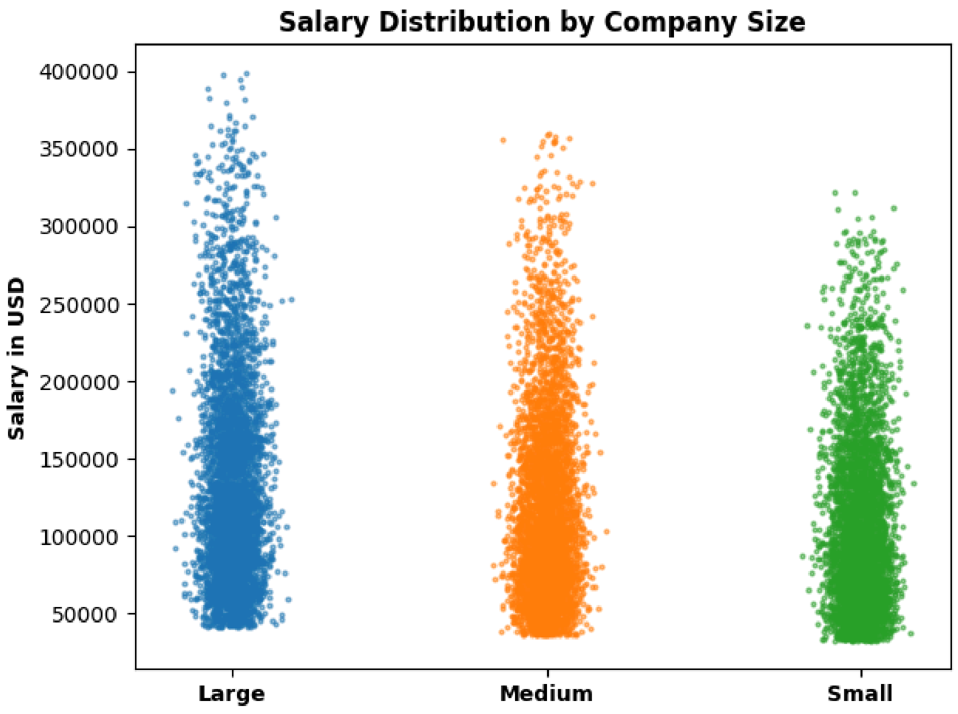


Appendix 4

Mean and Median Salary distribution by Company Size:



Scatter plot of Salary Distribution by Company Size:


Salary Distribution by Company Size
Salary in USD
Large
Medium
Small