

```
In [4]: import pandas as pd

# Load the dataset
df = pd.read_csv('Dataset salary 2024.csv')

print("DataFrame Head:")
print(df.head())
```

DataFrame Head:

|   | work_year | experience_level | employment_type | job_title                 | \ |
|---|-----------|------------------|-----------------|---------------------------|---|
| 0 | 2024      | SE               | FT              | AI Engineer               |   |
| 1 | 2024      | SE               | FT              | AI Engineer               |   |
| 2 | 2024      | SE               | FT              | Data Engineer             |   |
| 3 | 2024      | SE               | FT              | Data Engineer             |   |
| 4 | 2024      | SE               | FT              | Machine Learning Engineer |   |

|   | salary | salary_currency | salary_in_usd | employee_residence | remote_ratio | \ |
|---|--------|-----------------|---------------|--------------------|--------------|---|
| 0 | 202730 | USD             | 202730        | US                 | 0            |   |
| 1 | 92118  | USD             | 92118         | US                 | 0            |   |
| 2 | 130500 | USD             | 130500        | US                 | 0            |   |
| 3 | 96000  | USD             | 96000         | US                 | 0            |   |
| 4 | 190000 | USD             | 190000        | US                 | 0            |   |

|   | company_location | company_size |
|---|------------------|--------------|
| 0 | US               | M            |
| 1 | US               | M            |
| 2 | US               | M            |
| 3 | US               | M            |
| 4 | US               | M            |

```
In [3]: print("Missing values before imputation:")
print(df.isnull().sum())
```

Missing values before imputation:

```
work_year      0
experience_level 0
employment_type 0
job_title      0
salary         0
salary_currency 0
salary_in_usd   0
employee_residence 0
remote_ratio    0
company_location 0
company_size    0
dtype: int64
```

```
In [5]: df_mean_imputed = df.copy()

# Artificially introduce a missing value for demonstration
# For example, set a value in 'salary_in_usd' to NaN
df_mean_imputed.loc[0, 'salary_in_usd'] = None

# Demonstrate mean imputation
mean_salary = df_mean_imputed['salary_in_usd'].mean()
```

```
df_mean_imputed['salary_in_usd'].fillna(mean_salary, inplace=True)

print("DataFrame after mean imputation (demonstration, first row's salary was set to NaN then imputed):")
print(df_mean_imputed.head())
print(f"Mean salary used for imputation: {mean_salary}")
```

DataFrame after mean imputation (demonstration, first row's salary was set to NaN then imputed):

|   | work_year | experience_level | employment_type | job_title \               |
|---|-----------|------------------|-----------------|---------------------------|
| 0 | 2024      | SE               | FT              | AI Engineer               |
| 1 | 2024      | SE               | FT              | AI Engineer               |
| 2 | 2024      | SE               | FT              | Data Engineer             |
| 3 | 2024      | SE               | FT              | Data Engineer             |
| 4 | 2024      | SE               | FT              | Machine Learning Engineer |

|   | salary | salary_currency | salary_in_usd | employee_residence | remote_ratio \ |
|---|--------|-----------------|---------------|--------------------|----------------|
| 0 | 202730 | USD             | 149683.569649 | US                 | 0              |
| 1 | 92118  | USD             | 92118.000000  | US                 | 0              |
| 2 | 130500 | USD             | 130500.000000 | US                 | 0              |
| 3 | 96000  | USD             | 96000.000000  | US                 | 0              |
| 4 | 190000 | USD             | 190000.000000 | US                 | 0              |

|   | company_location | company_size |
|---|------------------|--------------|
| 0 | US               | M            |
| 1 | US               | M            |
| 2 | US               | M            |
| 3 | US               | M            |
| 4 | US               | M            |

Mean salary used for imputation: 149683.56964858162

C:\Users\hj\AppData\Local\Temp\ipykernel\_11256\1052085141.py:9: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.

The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.

```
df_mean_imputed['salary_in_usd'].fillna(mean_salary, inplace=True)
```

```
In [5]: df_median_imputed = df.copy()

# Artificially introduce a missing value for demonstration
df_median_imputed.loc[0, 'salary_in_usd'] = None

# Demonstrate median imputation
median_salary = df_median_imputed['salary_in_usd'].median()
df_median_imputed['salary_in_usd'] = df_median_imputed['salary_in_usd'].fillna(median_salary)

print("DataFrame after median imputation (demonstration, first row's salary was set to NaN then imputed):")
print(df_median_imputed.head())
print(f"Median salary used for imputation: {median_salary}")
```

DataFrame after median imputation (demonstration, first row's salary was set to NaN then imputed):

|   | work_year | experience_level | employment_type | job_title \               |
|---|-----------|------------------|-----------------|---------------------------|
| 0 | 2024      | SE               | FT              | AI Engineer               |
| 1 | 2024      | SE               | FT              | AI Engineer               |
| 2 | 2024      | SE               | FT              | Data Engineer             |
| 3 | 2024      | SE               | FT              | Data Engineer             |
| 4 | 2024      | SE               | FT              | Machine Learning Engineer |

|   | salary | salary_currency | salary_in_usd | employee_residence | remote_ratio \ |
|---|--------|-----------------|---------------|--------------------|----------------|
| 0 | 202730 | USD             | 141300.0      | US                 | 0              |
| 1 | 92118  | USD             | 92118.0       | US                 | 0              |
| 2 | 130500 | USD             | 130500.0      | US                 | 0              |
| 3 | 96000  | USD             | 96000.0       | US                 | 0              |
| 4 | 190000 | USD             | 190000.0      | US                 | 0              |

|   | company_location | company_size |
|---|------------------|--------------|
| 0 | US               | M            |
| 1 | US               | M            |
| 2 | US               | M            |
| 3 | US               | M            |
| 4 | US               | M            |

Median salary used for imputation: 141300.0

```
In [6]: from sklearn.impute import KNNImputer
import numpy as np

df_knn_imputed = df.copy()

# Select numerical columns for KNN imputation
numerical_cols = df_knn_imputed.select_dtypes(include=np.number).columns.tolist()

# Artificially introduce some missing values in numerical columns for demonstration
# For example, in 'salary_in_usd' and 'salary'
df_knn_imputed.loc[0:4, 'salary_in_usd'] = np.nan
df_knn_imputed.loc[5:9, 'salary'] = np.nan

print("Missing values before KNN imputation (artificially introduced for demo):")
print(df_knn_imputed[numerical_cols].isnull().sum())

# Initialize KNN Imputer
# n_neighbors is the number of neighboring samples to use for imputation
imputer = KNNImputer(n_neighbors=5)

# Apply KNN imputation to the numerical columns
df_knn_imputed[numerical_cols] = imputer.fit_transform(df_knn_imputed[numerical_cols])

print("\nDataFrame after KNN imputation (demonstration):")
print(df_knn_imputed.head(10))
print("\nMissing values after KNN imputation (should be zero for numerical columns)")
print(df_knn_imputed[numerical_cols].isnull().sum())
```

Missing values before KNN imputation (artificially introduced for demo):

```
work_year      0
salary         5
salary_in_usd  5
remote_ratio   0
dtype: int64
```

DataFrame after KNN imputation (demonstration):

|   | work_year | experience_level | employment_type | job_title \               |
|---|-----------|------------------|-----------------|---------------------------|
| 0 | 2024.0    | SE               | FT              | AI Engineer               |
| 1 | 2024.0    | SE               | FT              | AI Engineer               |
| 2 | 2024.0    | SE               | FT              | Data Engineer             |
| 3 | 2024.0    | SE               | FT              | Data Engineer             |
| 4 | 2024.0    | SE               | FT              | Machine Learning Engineer |
| 5 | 2024.0    | SE               | FT              | Machine Learning Engineer |
| 6 | 2024.0    | MI               | FT              | ML Engineer               |
| 7 | 2024.0    | MI               | FT              | ML Engineer               |
| 8 | 2024.0    | EN               | FT              | Data Analyst              |
| 9 | 2024.0    | EN               | FT              | Data Analyst              |

|   | salary   | salary_currency | salary_in_usd | employee_residence | remote_ratio \ |
|---|----------|-----------------|---------------|--------------------|----------------|
| 0 | 202730.0 | USD             | 154476.8      | US                 | 0.0            |
| 1 | 92118.0  | USD             | 154476.8      | US                 | 0.0            |
| 2 | 130500.0 | USD             | 160272.8      | US                 | 0.0            |
| 3 | 96000.0  | USD             | 108800.0      | US                 | 0.0            |
| 4 | 190000.0 | USD             | 160476.8      | US                 | 0.0            |
| 5 | 155846.0 | USD             | 160000.0      | US                 | 0.0            |
| 6 | 203846.0 | USD             | 400000.0      | US                 | 0.0            |
| 7 | 123746.0 | USD             | 65000.0       | US                 | 0.0            |
| 8 | 142269.6 | USD             | 101520.0      | US                 | 0.0            |
| 9 | 142269.6 | USD             | 45864.0       | US                 | 0.0            |

|   | company_location | company_size |
|---|------------------|--------------|
| 0 | US               | M            |
| 1 | US               | M            |
| 2 | US               | M            |
| 3 | US               | M            |
| 4 | US               | M            |
| 5 | US               | M            |
| 6 | US               | M            |
| 7 | US               | M            |
| 8 | US               | M            |
| 9 | US               | M            |

Missing values after KNN imputation (should be zero for numerical columns):

```
work_year      0
salary         0
salary_in_usd  0
remote_ratio   0
dtype: int64
```

In [7]: `import numpy as np`

```
# 1. Create a copy of the original DataFrame for IQR method
df_iqr_handled = df.copy()
```

```
# 2. For the 'salary_in_usd' column, calculate Q1 and Q3
Q1 = df_iqr_handled['salary_in_usd'].quantile(0.25)
Q3 = df_iqr_handled['salary_in_usd'].quantile(0.75)

# 3. Calculate the Interquartile Range (IQR)
IQR = Q3 - Q1

# 4. Define the upper and lower bounds for outlier detection
lower_bound_iqr = Q1 - 1.5 * IQR
upper_bound_iqr = Q3 + 1.5 * IQR

print(f"IQR Lower Bound: {lower_bound_iqr}")
print(f"IQR Upper Bound: {upper_bound_iqr}")

# 5. Identify the outliers
outliers_iqr = df_iqr_handled[
    (df_iqr_handled['salary_in_usd'] < lower_bound_iqr) |
    (df_iqr_handled['salary_in_usd'] > upper_bound_iqr)
]
print(f"\nNumber of outliers identified by IQR method: {len(outliers_iqr)}")

# 6. Demonstrate handling these outliers by capping them
df_iqr_handled['salary_in_usd'] = np.where(
    df_iqr_handled['salary_in_usd'] < lower_bound_iqr,
    lower_bound_iqr,
    df_iqr_handled['salary_in_usd']
)
df_iqr_handled['salary_in_usd'] = np.where(
    df_iqr_handled['salary_in_usd'] > upper_bound_iqr,
    upper_bound_iqr,
    df_iqr_handled['salary_in_usd']
)

print("\nDataFrame head after IQR outlier capping:")
print(df_iqr_handled.head())

# Verify if outliers are handled (check min/max values)
print(f"\nMin salary_in_usd after IQR capping: {df_iqr_handled['salary_in_usd'].min}")
print(f"Max salary_in_usd after IQR capping: {df_iqr_handled['salary_in_usd'].max()}"
```

IQR Lower Bound: -26037.5

IQR Upper Bound: 313062.5

Number of outliers identified by IQR method: 283

DataFrame head after IQR outlier capping:

|   | work_year | experience_level | employment_type | job_title                 | \ |
|---|-----------|------------------|-----------------|---------------------------|---|
| 0 | 2024      | SE               | FT              | AI Engineer               |   |
| 1 | 2024      | SE               | FT              | AI Engineer               |   |
| 2 | 2024      | SE               | FT              | Data Engineer             |   |
| 3 | 2024      | SE               | FT              | Data Engineer             |   |
| 4 | 2024      | SE               | FT              | Machine Learning Engineer |   |

|   | salary | salary_currency | salary_in_usd | employee_residence | remote_ratio | \ |
|---|--------|-----------------|---------------|--------------------|--------------|---|
| 0 | 202730 | USD             | 202730.0      | US                 | 0            |   |
| 1 | 92118  | USD             | 92118.0       | US                 | 0            |   |
| 2 | 130500 | USD             | 130500.0      | US                 | 0            |   |
| 3 | 96000  | USD             | 96000.0       | US                 | 0            |   |
| 4 | 190000 | USD             | 190000.0      | US                 | 0            |   |

|   | company_location | company_size |
|---|------------------|--------------|
| 0 | US               | M            |
| 1 | US               | M            |
| 2 | US               | M            |
| 3 | US               | M            |
| 4 | US               | M            |

Min salary\_in\_usd after IQR capping: 15000.0

Max salary\_in\_usd after IQR capping: 313062.5

```
In [8]: from scipy.stats import zscore

# 1. Create a copy of the original DataFrame for Z-score method
df_zscore_handled = df.copy()

# 2. For the 'salary_in_usd' column, calculate the Z-score for each value
df_zscore_handled['salary_in_usd_zscore'] = zscore(df_zscore_handled['salary_in_usd'])

# 3. Define a Z-score threshold (e.g., 3)
zscore_threshold = 3

print(f"Z-score Threshold: {zscore_threshold}")

# 4. Identify the outliers
outliers_zscore = df_zscore_handled[
    (df_zscore_handled['salary_in_usd_zscore'].abs() > zscore_threshold)
]
print(f"\nNumber of outliers identified by Z-score method: {len(outliers_zscore)}")

# 5. Demonstrate handling these outliers by capping them
# Calculate mean and standard deviation for capping values
mean_salary = df_zscore_handled['salary_in_usd'].mean()
std_salary = df_zscore_handled['salary_in_usd'].std()

# Define capping values
upper_cap_zscore = mean_salary + zscore_threshold * std_salary
```

```

lower_cap_zscore = mean_salary - zscore_threshold * std_salary

print(f"\nZ-score Lower Cap: {lower_cap_zscore}")
print(f"Z-score Upper Cap: {upper_cap_zscore}")

df_zscore_handled['salary_in_usd'] = np.where(
    df_zscore_handled['salary_in_usd'] > upper_cap_zscore,
    upper_cap_zscore,
    df_zscore_handled['salary_in_usd']
)
df_zscore_handled['salary_in_usd'] = np.where(
    df_zscore_handled['salary_in_usd'] < lower_cap_zscore,
    lower_cap_zscore,
    df_zscore_handled['salary_in_usd']
)

print("\nDataFrame head after Z-score outlier capping:")
print(df_zscore_handled.head())

# Verify if outliers are handled (check min/max values)
print(f"\nMin salary_in_usd after Z-score capping: {df_zscore_handled['salary_in_usd'].min()}")
print(f"Max salary_in_usd after Z-score capping: {df_zscore_handled['salary_in_usd'].max()}")

```

Z-score Threshold: 3

Number of outliers identified by Z-score method: 138

Z-score Lower Cap: -55829.101496369025

Z-score Upper Cap: 355202.6574416938

DataFrame head after Z-score outlier capping:

|   | work_year | experience_level | employment_type | job_title                 |  |
|---|-----------|------------------|-----------------|---------------------------|--|
| 0 | 2024      | SE               | FT              | AI Engineer               |  |
| 1 | 2024      | SE               | FT              | AI Engineer               |  |
| 2 | 2024      | SE               | FT              | Data Engineer             |  |
| 3 | 2024      | SE               | FT              | Data Engineer             |  |
| 4 | 2024      | SE               | FT              | Machine Learning Engineer |  |

|   | salary | salary_currency | salary_in_usd | employee_residence | remote_ratio |  |
|---|--------|-----------------|---------------|--------------------|--------------|--|
| 0 | 202730 | USD             | 202730.0      | US                 | 0            |  |
| 1 | 92118  | USD             | 92118.0       | US                 | 0            |  |
| 2 | 130500 | USD             | 130500.0      | US                 | 0            |  |
| 3 | 96000  | USD             | 96000.0       | US                 | 0            |  |
| 4 | 190000 | USD             | 190000.0      | US                 | 0            |  |

|   | company_location | company_size | salary_in_usd_zscore |
|---|------------------|--------------|----------------------|
| 0 | US               | M            | 0.774317             |
| 1 | US               | M            | -0.840381            |
| 2 | US               | M            | -0.280086            |
| 3 | US               | M            | -0.783712            |
| 4 | US               | M            | 0.588487             |

Min salary\_in\_usd after Z-score capping: 15000.0

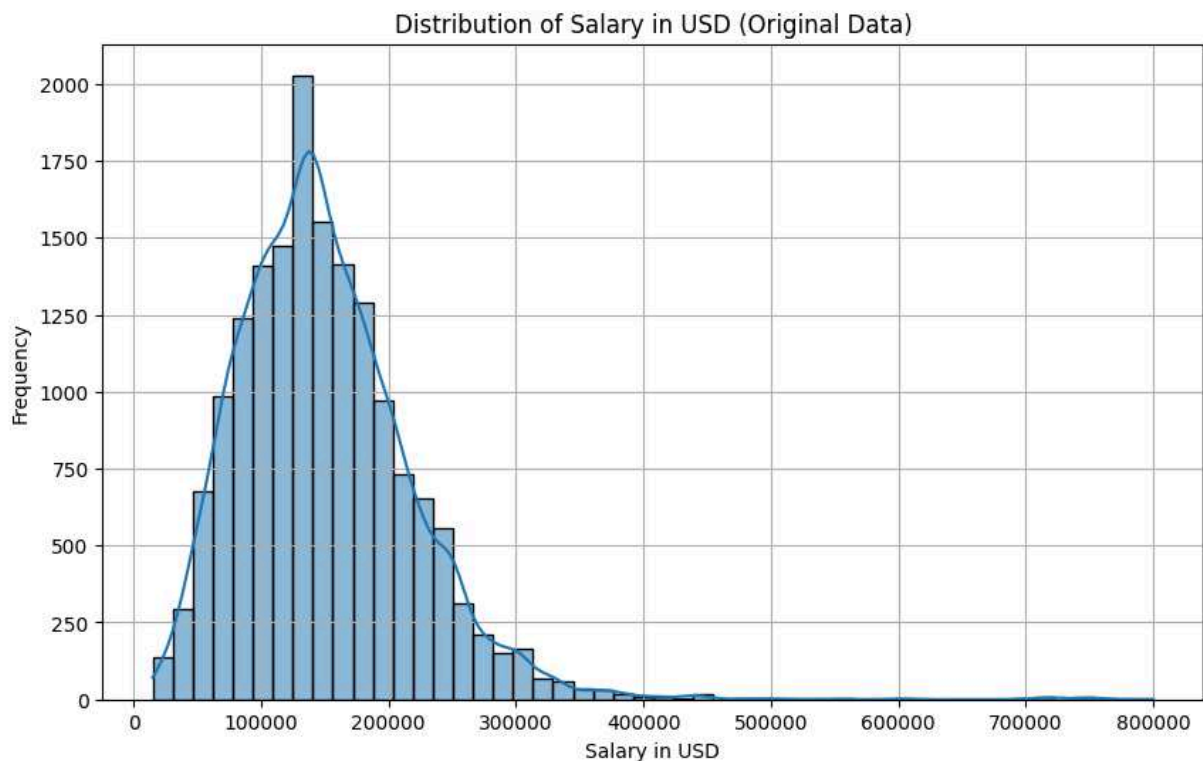
Max salary\_in\_usd after Z-score capping: 355202.6574416938

```
In [9]: import matplotlib.pyplot as plt
import seaborn as sns

print("Matplotlib and Seaborn imported successfully.")
```

Matplotlib and Seaborn imported successfully.

```
In [10]: plt.figure(figsize=(10, 6))
sns.histplot(df['salary_in_usd'], kde=True, bins=50)
plt.title('Distribution of Salary in USD (Original Data)')
plt.xlabel('Salary in USD')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()
```

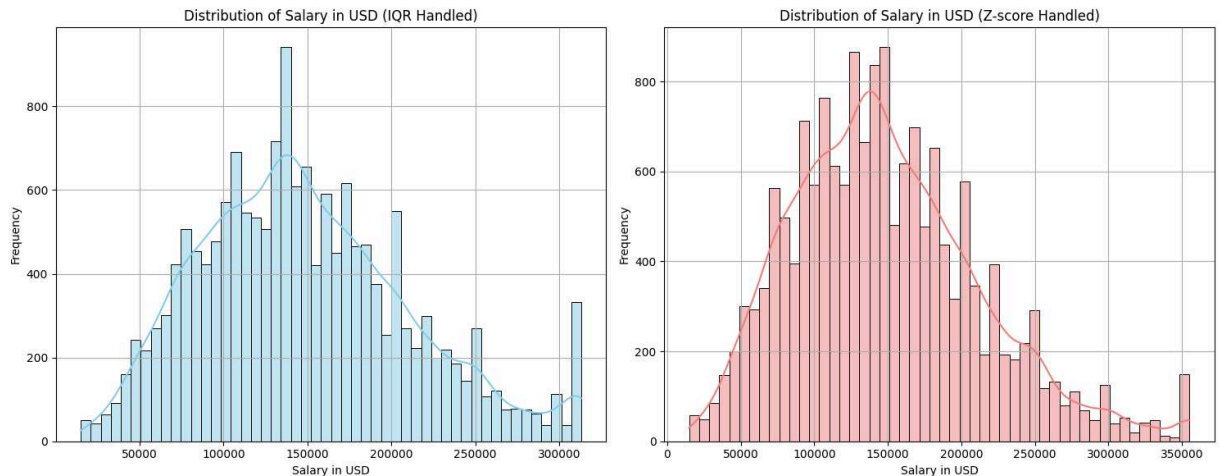


```
In [11]: plt.figure(figsize=(15, 6))

plt.subplot(1, 2, 1)
sns.histplot(df_iqr_handled['salary_in_usd'], kde=True, bins=50, color='skyblue')
plt.title('Distribution of Salary in USD (IQR Handled)')
plt.xlabel('Salary in USD')
plt.ylabel('Frequency')
plt.grid(True)

plt.subplot(1, 2, 2)
sns.histplot(df_zscore_handled['salary_in_usd'], kde=True, bins=50, color='lightcoral')
plt.title('Distribution of Salary in USD (Z-score Handled)')
plt.xlabel('Salary in USD')
plt.ylabel('Frequency')
plt.grid(True)
```

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



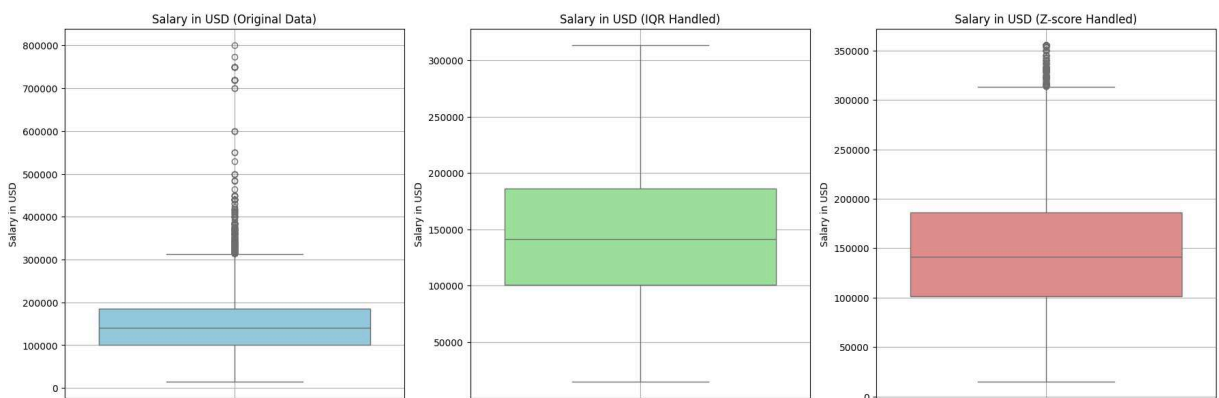
```
In [12]: plt.figure(figsize=(18, 6))

plt.subplot(1, 3, 1)
sns.boxplot(y=df['salary_in_usd'], color='skyblue')
plt.title('Salary in USD (Original Data)')
plt.ylabel('Salary in USD')
plt.grid(True)

plt.subplot(1, 3, 2)
sns.boxplot(y=df_iqr_handled['salary_in_usd'], color='lightgreen')
plt.title('Salary in USD (IQR Handled)')
plt.ylabel('Salary in USD')
plt.grid(True)

plt.subplot(1, 3, 3)
sns.boxplot(y=df_zscore_handled['salary_in_usd'], color='lightcoral')
plt.title('Salary in USD (Z-score Handled)')
plt.ylabel('Salary in USD')
plt.grid(True)

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



**Reasoning:** I will create a box plot to visualize the relationship between 'experience\_level' and 'salary\_in\_usd' using the `df_iqr_handled` DataFrame, as specified in the instructions,

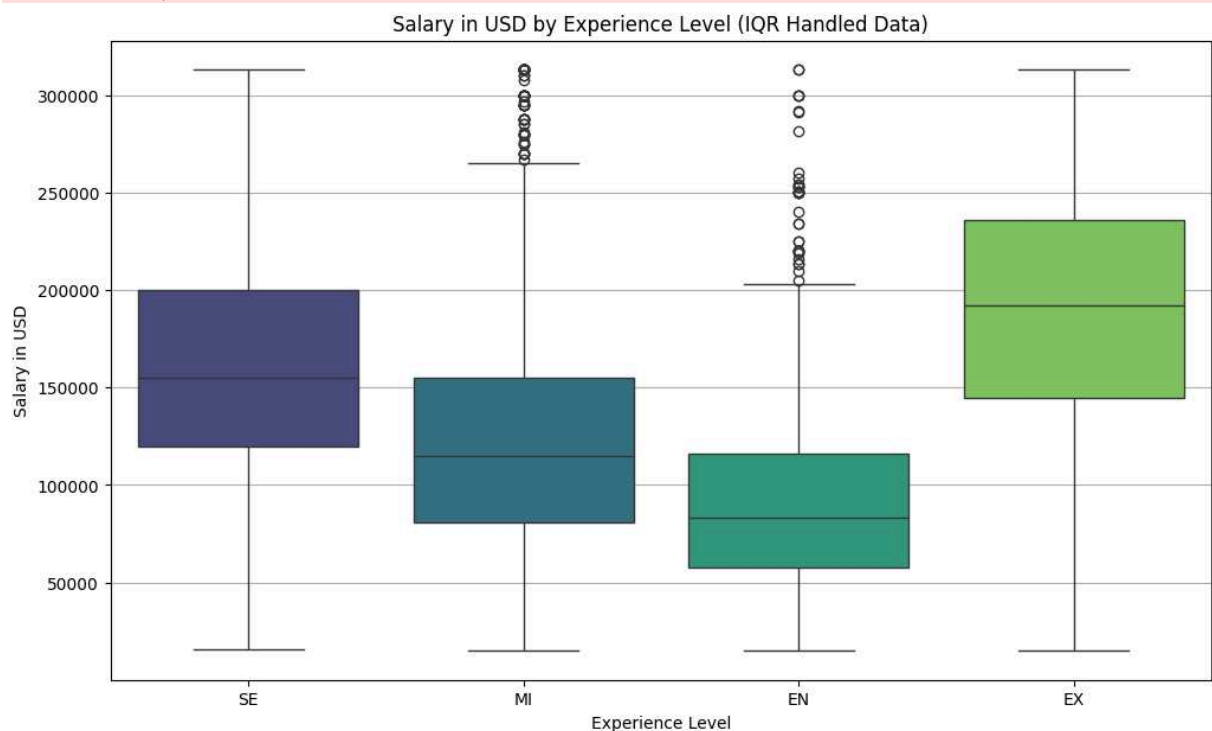
to understand how salary varies across different experience levels after outlier handling.

```
In [13]: plt.figure(figsize=(12, 7))
sns.boxplot(x='experience_level', y='salary_in_usd', data=df_iqr_handled, palette='
plt.title('Salary in USD by Experience Level (IQR Handled Data)')
plt.xlabel('Experience Level')
plt.ylabel('Salary in USD')
plt.grid(axis='y')
plt.show()
```

/tmp/ipython-input-3021644509.py:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

```
sns.boxplot(x='experience_level', y='salary_in_usd', data=df_iqr_handled, palette='viridis')
```



```
In [14]: plt.figure(figsize=(12, 7))
sns.boxplot(x='experience_level', y='salary_in_usd', data=df_iqr_handled, palette='
plt.title('Salary in USD by Experience Level (IQR Handled Data)')
plt.xlabel('Experience Level')
plt.ylabel('Salary in USD')
plt.grid(axis='y')
plt.show()
```

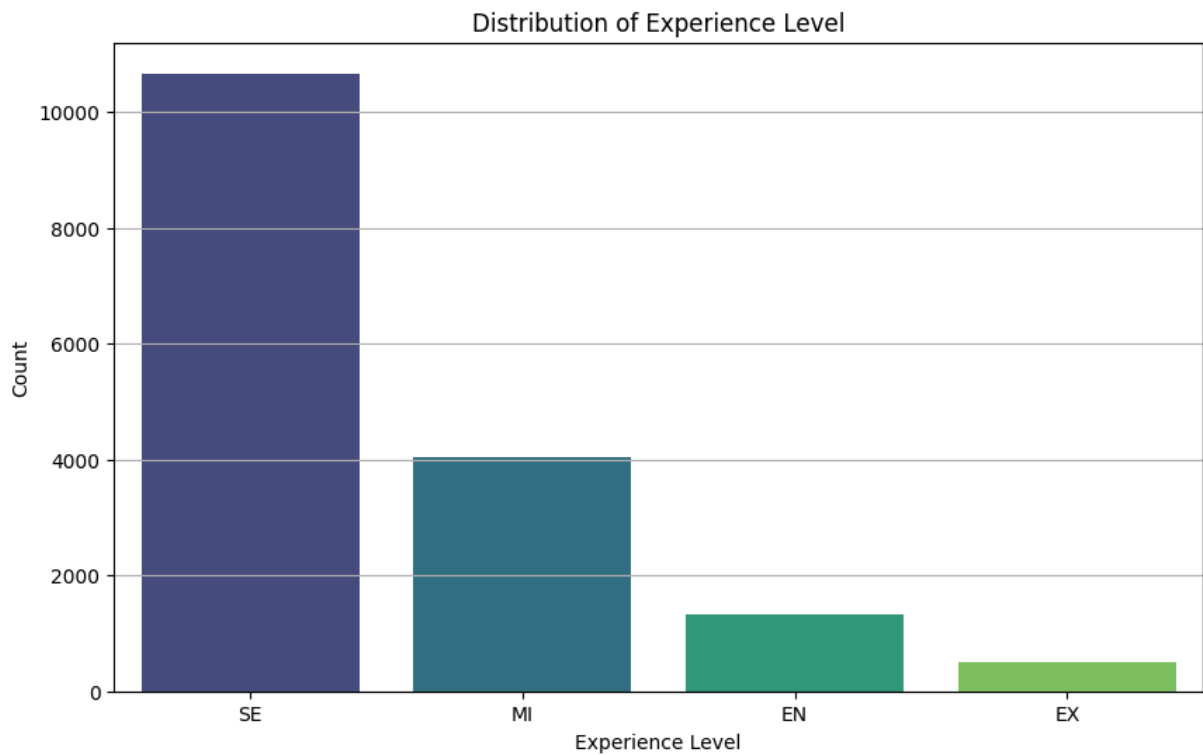


```
In [15]: plt.figure(figsize=(10, 6))
sns.countplot(x='experience_level', data=df_iqr_handled, palette='viridis', order=d
plt.title('Distribution of Experience Level')
plt.xlabel('Experience Level')
plt.ylabel('Count')
plt.grid(axis='y')
plt.show()
```

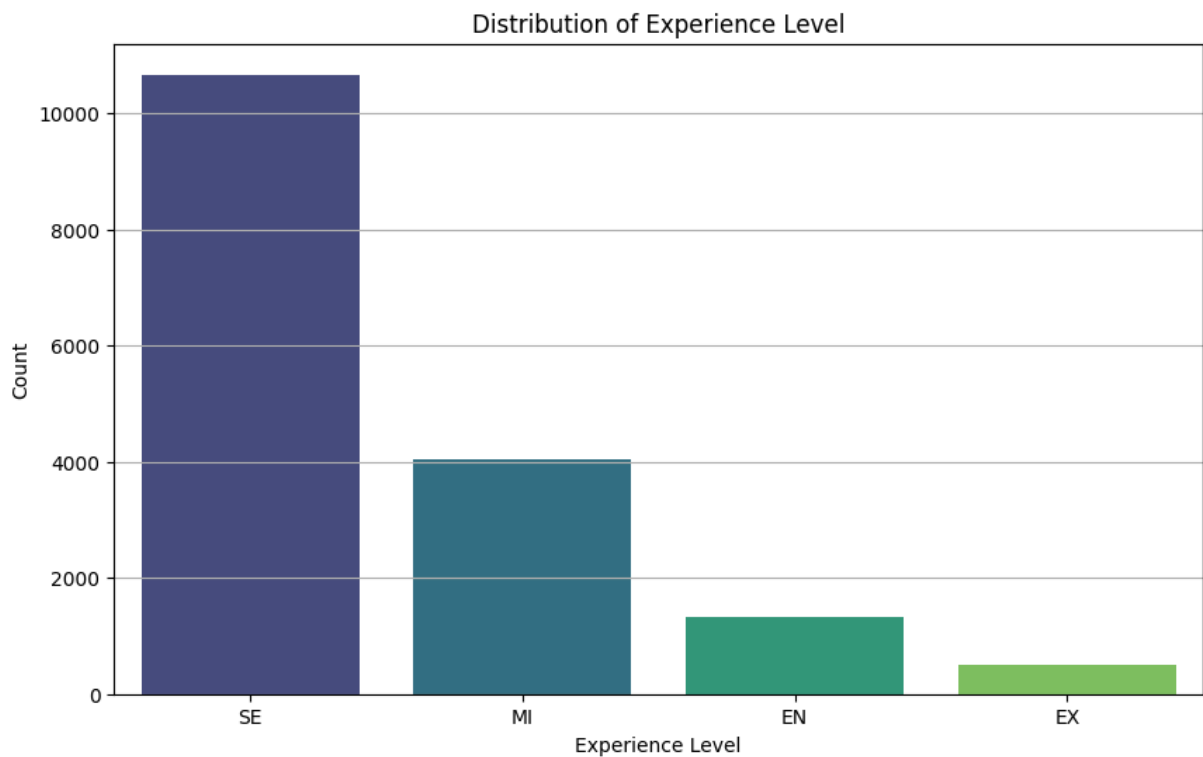
/tmp/ipython-input-3479374369.py:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

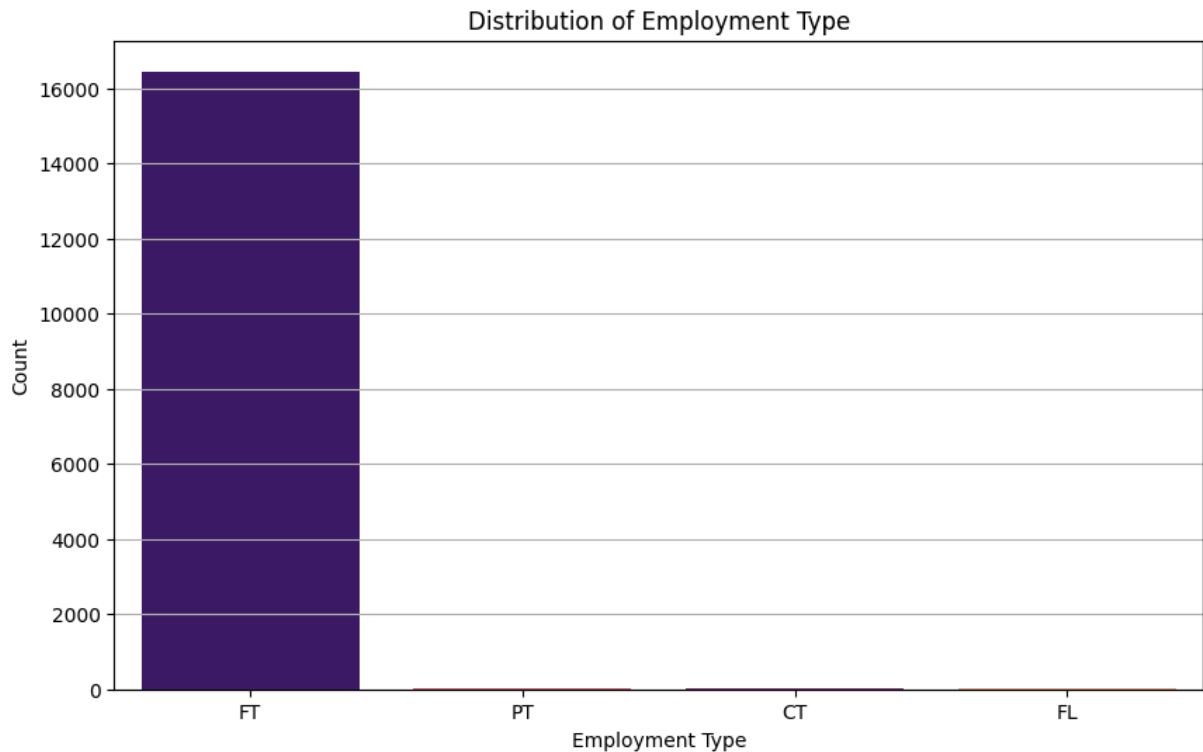
```
sns.countplot(x='experience_level', data=df_iqr_handled, palette='viridis', order=
df_iqr_handled['experience_level'].value_counts().index)
```



```
In [16]: plt.figure(figsize=(10, 6))
sns.countplot(x='experience_level', data=df_iqr_handled, palette='viridis', order=d
plt.title('Distribution of Experience Level')
plt.xlabel('Experience Level')
plt.ylabel('Count')
plt.grid(axis='y')
plt.show()
```



```
In [17]: plt.figure(figsize=(10, 6))
sns.countplot(x='employment_type', data=df_iqr_handled, palette='magma', order=df_i
plt.title('Distribution of Employment Type')
plt.xlabel('Employment Type')
plt.ylabel('Count')
plt.grid(axis='y')
plt.show()
```



```
In [18]: plt.figure(figsize=(10, 6))
sns.countplot(x='company_size', data=df_iqr_handled, palette='plasma', order=df_iqr
plt.title('Distribution of Company Size')
plt.xlabel('Company Size')
plt.ylabel('Count')
plt.grid(axis='y')
plt.show()
```

