Data Science Salary Prediction

1 | Overview

This dataset shows us information about the salaries of data professionals. Our goal is to understand and predict the salary trends among data industries worldwide.

Importing Libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import plotly.graph_objects as go
plt.style.use('ggplot')
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor,AdaBoostRegressor,GradientBoostingRegressor
from sklearn.svm import SVR
from sklearn.linear_model import LinearRegression, Ridge,Lasso
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
from sklearn.model_selection import RandomizedSearchCV
from catboost import CatBoostRegressor
from xgboost import XGBRegressor
from lightgbm import LGBMRegressor
from sklearn.tree import ExtraTreeRegressor, DecisionTreeRegressor
from sklearn.model_selection import train_test_split, GridSearchCV
from scipy.stats import skew
from sklearn.preprocessing import MinMaxScaler
from mlxtend.regressor import StackingCVRegressor
from wordcloud import WordCloud, STOPWORDS
import warnings
warnings.filterwarnings('ignore')
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
pd.set_option('display.width', None)
pd.set_option('display.float_format', lambda x: '%.3f' % x)
```

In [2]:
```
custom_style = {
    "axes.facecolor": "#F3EFE7",      # Background color for the plot
    "axes.edgecolor": "0.6",          # Color of the axes edges
    "axes.grid": True,                # Show grid lines
    "grid.color": "0.99",             # Color of the grid lines
    "axes.spines.left": False,        # Hide the left spine
    "axes.spines.bottom": False,      # Hide the bottom spine
    "axes.spines.top": False,         # Hide the top spine
    "axes.spines.right": False,       # Hide the right spine
    "font.family": "sans-serif",      # Font family
    "font.sans-serif": ["Noto Sans", "Helvetica"]  # Specify the font

}
sns.set_style("white", rc=custom_style)
plt.style.use(custom_style)
```

Loading Data

In [3]:
```
df = pd.read_csv(r'C:\Users\SachinR\Downloads\PersonalPy\Sia\ds_salaries.csv')
```

2 | Data Understanding

- Dataframe `shape`
- `head` and `tail`
- `dtypes`
- `describe`

In [4]:
```
df.shape
```

Out[4]:
```
(3755, 11)
```

In [5]:
```python
df.head(2)
```

Out[5]:

	work_year	experience_level	employment_type	job_title	salary	salary_currency	salary_in_usd	employee_residence	remote_ratio	company_location	company_siz
0	2023	SE	FT	Principal Data Scientist	80000	EUR	85847	ES	100	ES	
1	2023	MI	CT	ML Engineer	30000	USD	30000	US	100	US	

In [6]:
```python
df.columns
```

Out[6]:
```
Index(['work_year', 'experience_level', 'employment_type', 'job_title',
       'salary', 'salary_currency', 'salary_in_usd', 'employee_residence',
       'remote_ratio', 'company_location', 'company_size'],
      dtype='object')
```

In [7]:
```python
df.dtypes
```

Out[7]:
```
work_year              int64
experience_level      object
employment_type       object
job_title             object
salary                 int64
salary_currency       object
salary_in_usd          int64
employee_residence    object
remote_ratio           int64
company_location      object
company_size          object
dtype: object
```

In [8]:
```python
df.describe(include='all').round().T
```

Out[8]:

	count	unique	top	freq	mean	std	min	25%	50%	75%	max
work_year	3755.000	NaN	NaN	NaN	2022.000	1.000	2020.000	2022.000	2022.000	2023.000	2023.000
experience_level	3755	4	SE	2516	NaN	NaN	NaN	NaN	NaN	NaN	NaN
employment_type	3755	4	FT	3718	NaN	NaN	NaN	NaN	NaN	NaN	NaN
job_title	3755	93	Data Engineer	1040	NaN	NaN	NaN	NaN	NaN	NaN	NaN
salary	3755.000	NaN	NaN	NaN	190696.000	671677.000	6000.000	100000.000	138000.000	180000.000	30400000.000
salary_currency	3755	20	USD	3224	NaN	NaN	NaN	NaN	NaN	NaN	NaN
salary_in_usd	3755.000	NaN	NaN	NaN	137570.000	63056.000	5132.000	95000.000	135000.000	175000.000	450000.000
employee_residence	3755	78	US	3004	NaN	NaN	NaN	NaN	NaN	NaN	NaN
remote_ratio	3755.000	NaN	NaN	NaN	46.000	49.000	0.000	0.000	0.000	100.000	100.000
company_location	3755	72	US	3040	NaN	NaN	NaN	NaN	NaN	NaN	NaN
company_size	3755	3	M	3153	NaN	NaN	NaN	NaN	NaN	NaN	NaN

We see there is a significant difference in the 75% value and max value value of `salary_in_usd`. It would be preferable to eliminate the outliers.

Checking for Null values

In [9]:
```python
df.isnull().sum().sum()
```

Out[9]:
```
0
```

Checking for Duplicate Rows

In [10]:
```python
df[df.duplicated()].shape[0]
```

Out[10]:
```
1171
```

Drop `Duplicates` and Reset `Index`

In [11]:
```python
df.drop_duplicates(keep='first', inplace=True)
df.reset_index(drop=True, inplace=True)
df.shape
```

Out[11]:
```
(2584, 11)
```

Dropping `salary`, `salary_currency` as we already have a standard feature i.e `salary_in_usd` which will be our target feature

In [12]:
```python
df.drop(['salary','salary_currency'],axis=1, inplace=True)
```

Updating rows so that it's easier to understand during EDA

In [13]:
```python
df['experience_level'] = df['experience_level'].replace({
    'SE': 'Senior',
```

```
    'EN': 'Entry level',
    'EX': 'Executive level',
    'MI': 'Mid/Intermediate level',
})

df['employment_type'] = df['employment_type'].replace({
    'FL': 'Freelancer',
    'CT': 'Contractor',
    'FT' : 'Full-time',
    'PT' : 'Part-time'
})

df['remote_ratio'] = df['remote_ratio'].astype(str)
df['remote_ratio'] = df['remote_ratio'].replace({
    '0': 'On-Site',
    '50': 'Half-Remote',
    '100' : 'Full-Remote',
})
```

In [14]:

```
df.head(2)
```

Out[14]:

	work_year	experience_level	employment_type	job_title	salary_in_usd	employee_residence	remote_ratio	company_location	company_size
0	2023	Senior	Full-time	Principal Data Scientist	85847	ES	Full-Remote	ES	L
1	2023	Mid/Intermediate level	Contractor	ML Engineer	30000	US	Full-Remote	US	S

3 | Exploratory Data Analysis

Separate Categorical Columns

In [15]:

```
cat_cols = df.select_dtypes('object').columns
cat_cols
```

Out[15]:

```
Index(['experience_level', 'employment_type', 'job_title',
       'employee_residence', 'remote_ratio', 'company_location',
       'company_size'],
      dtype='object')
```

Separate Numeric Columns

In [16]:

```
num_cols = df.select_dtypes('int64').columns
num_cols
```

Out[16]:

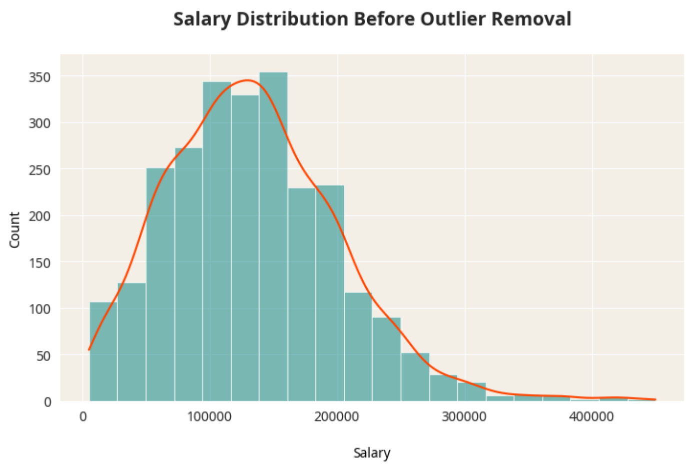
```
Index(['work_year', 'salary_in_usd'], dtype='object')
```

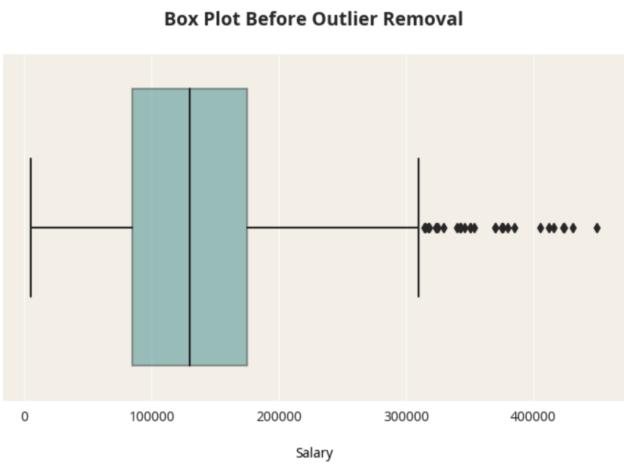
3.1 | Univariate Analysis

Target Distribution

In [17]:

```
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(14, 5))

# Histogram on the left
ax1 = sns.histplot(df['salary_in_usd'], alpha=0.5, color="teal", bins=20, kde=True, ax=axes[0])
axes[0].set_title("Salary Distribution Before Outlier Removal\n", fontsize=14, fontweight="bold")
axes[0].set_xlabel('\nSalary', color="black", fontsize=10)
axes[0].set_ylabel('Count', color="black", fontsize=10)
ax1.lines[0].set_color('orangered')

# Box plot on the right
ax2 = sns.boxplot(data=df, x='salary_in_usd', ax=axes[1], color="teal", boxprops=dict(alpha=0.4))
ax2.set_title("Box Plot Before Outlier Removal\n", fontsize=14, fontweight="bold")
ax2.set_xlabel('\nSalary', color="black", fontsize=10)

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



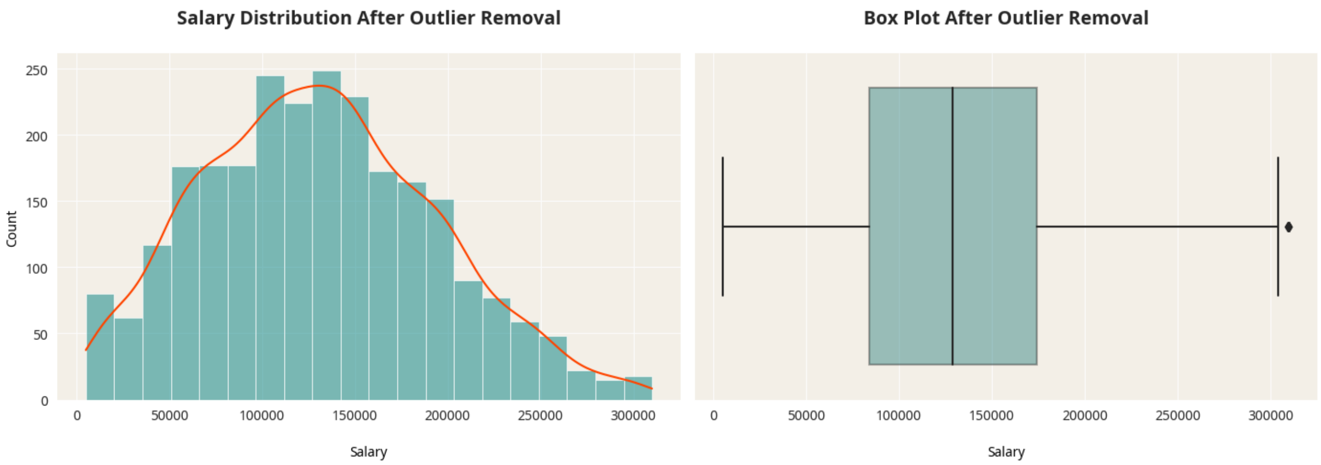


Removing Outliers

In [18]:
```
Q1 = df['salary_in_usd'].quantile(0.25)
Q3 = df['salary_in_usd'].quantile(0.75)
IQR = Q3 - Q1

lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

df = df[(df['salary_in_usd'] >= lower_bound) & (df['salary_in_usd'] <= upper_bound)]
```

Plotting the distribution again

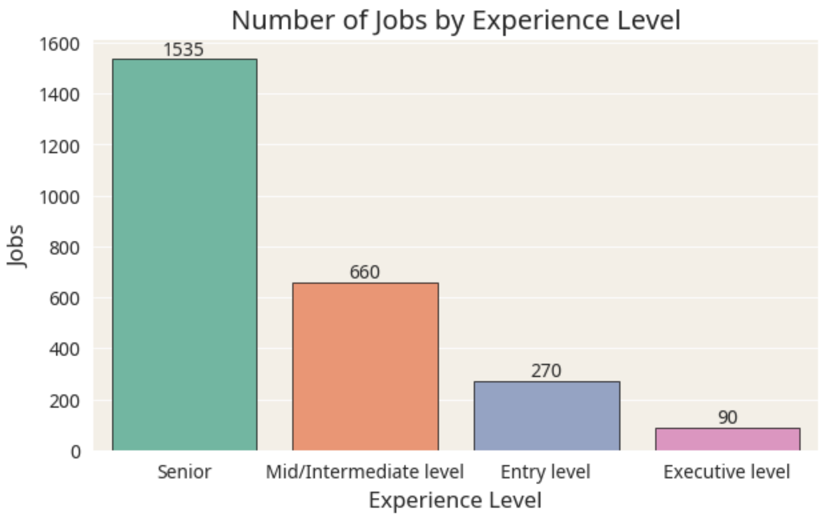
In [19]:
```
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(14, 5))

# Histogram on the left
ax1 = sns.histplot(df['salary_in_usd'], alpha=0.5, color="teal", bins=20, kde=True, ax=axes[0])
axes[0].set_title("Salary Distribution After Outlier Removal\n", fontsize=14, fontweight="bold")
axes[0].set_xlabel('\nSalary', color="black", fontsize=10)
axes[0].set_ylabel('Count', color="black", fontsize=10)
ax1.lines[0].set_color('orangered')

# Box plot on the right
ax2 = sns.boxplot(data=df, x='salary_in_usd', ax=axes[1], color="teal", boxprops=dict(alpha=0.4))
ax2.set_title("Box Plot After Outlier Removal\n", fontsize=14, fontweight="bold")
ax2.set_xlabel('\nSalary', color="black", fontsize=10)

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



Removal of outliers has `improved` the distribution, making it more suitable `modelling`

Experience Level Counts

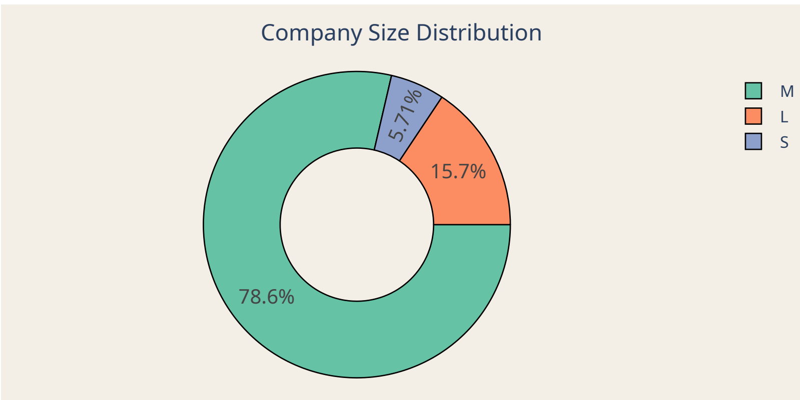
In [20]:
```
custom_palette = sns.color_palette("Set2", n_colors=len(df['experience_level'].unique()))
plt.figure(figsize=(7, 4))
ax = sns.countplot(data=df, x='experience_level', edgecolor='black', palette=custom_palette)
plt.xlabel('Experience Level')
plt.ylabel('Jobs')
plt.title('Number of Jobs by Experience Level')

for p in ax.patches:
    ax.annotate(format(p.get_height(), '.0f'), (p.get_x() + p.get_width() / 2., p.get_height()),
                ha='center', va='center', xytext=(0, 5), textcoords='offset points')

ax.set_facecolor('#F3EFE7')

plt.show()
```



Most of the job positions are held by `Seniors`

Company Size Distribution

In [21]:
```
company_size_counts = df['company_size'].value_counts()
values = company_size_counts.values
names = company_size_counts.index

fig = px.pie(values=values, names=names, title='Company Size Distribution',
             color_discrete_sequence=px.colors.qualitative.Set2,
             hole=0.5)
fig.update_layout(
    font=dict(size=12, family="Noto Sans"),
    paper_bgcolor="#F3EFE7",
    width=600,
    height=300,
    title_x=0.5,
    title_y=0.95,
    margin=dict(l=0, r=10, t=50, b=20)
    ,uniformtext_minsize=15, uniformtext_mode='hide'
)
fig.update_traces(rotation=90, marker=dict(line=dict(color='black', width=1)))
fig.show()
```

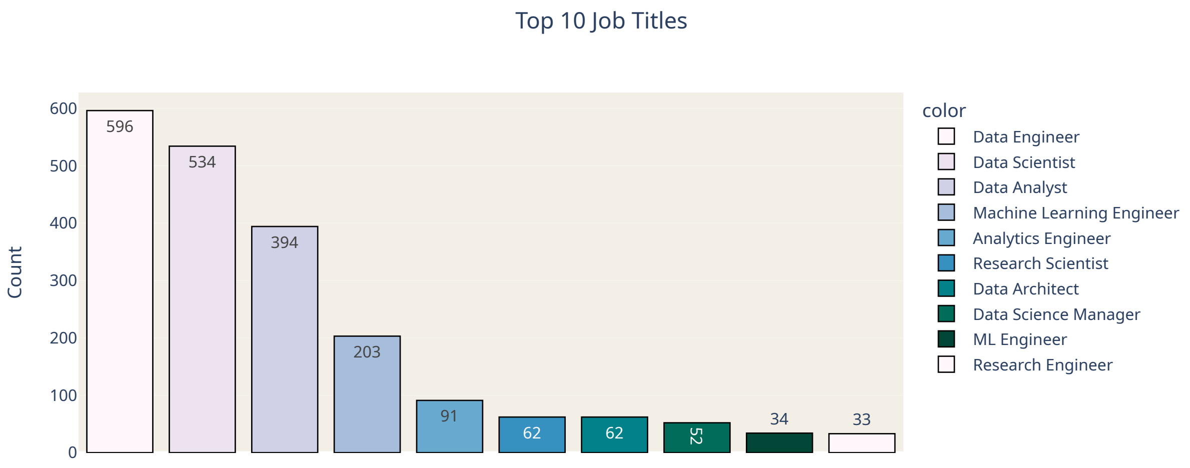


Our data is dominated by `medium` sized companies.

Top 10 Job Titles by Count

In [22]:
```
# Calculate the top 10 job titles
top10_job_title = df['job_title'].value_counts()[:10]

# Create a bar chart
fig = px.bar(
    y = top10_job_title.values,
    x = top10_job_title.index,
    color = top10_job_title.index,
    color_discrete_sequence = px.colors.sequential.PuBuGn,
    text = top10_job_title.values,
    title = 'Top 10 Job Titles',
)

# Update layout for better appearance
fig.update_layout(
```

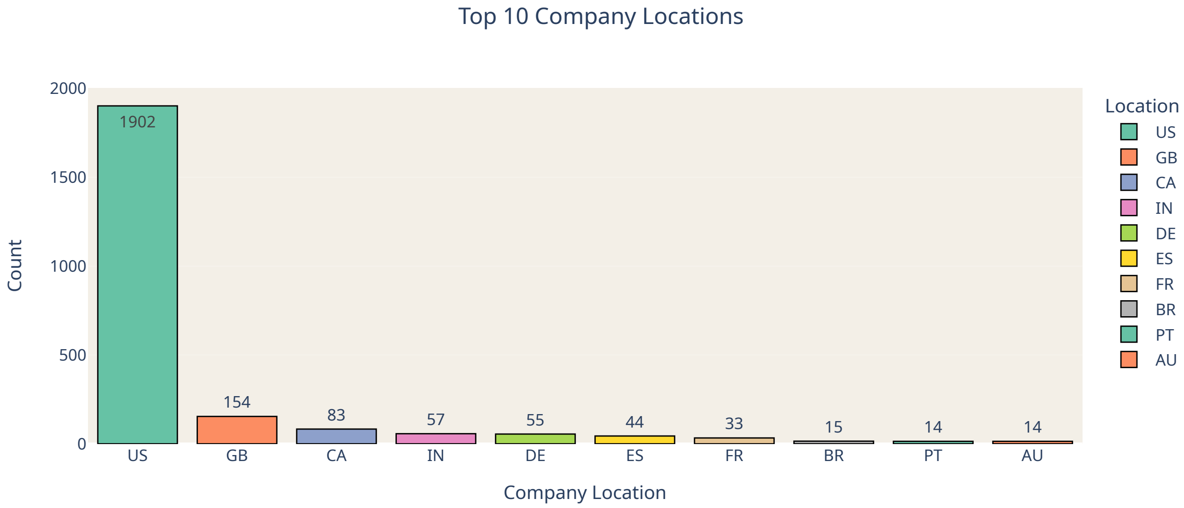
```
    xaxis_title="Job Titles",
    yaxis_title="Count",
    font=dict(size=12, family="Noto Sans"),
    margin=dict(l=50, r=20, t=80, b=50),
    plot_bgcolor="#F3EFE7"
)
fig.update_traces(marker=dict(line=dict(color='black', width=1)))
fig.update_layout(title_x=0.5, title_y=0.95)
fig.update_layout(width=900, height=400)
fig.update_xaxes(visible=False, showticklabels=False)
fig.show()
```



`Data Engineer`, `Data Scientist` and `Data Analyst` are the most common job titles

Top 10 Company Locations

In [23]:
```
# Calculate the top 10 job titles
top10_company_loc = df['company_location'].value_counts()[:10]

# Create a bar chart
fig = px.bar(
    y = top10_company_loc.values,
    x = top10_company_loc.index,
    color = top10_company_loc.index,
    color_discrete_sequence=px.colors.qualitative.Set2,
    labels={"color": 'Location'},
    text = top10_company_loc.values,
    title = 'Top 10 Company Locations',
)

# Update layout for better appearance
fig.update_layout(
    xaxis_title="Company Location",
    yaxis_title="Count",
    font=dict(size=12, family="Noto Sans"),
    margin=dict(l=50, r=20, t=80, b=50),
    plot_bgcolor="#F3EFE7"
)

# Add a black border to the bars
fig.update_traces(marker=dict(line=dict(color='black', width=1)))
fig.update_layout(title_x=0.5, title_y=0.95)
# Adjust the figure size for better visibility
fig.update_layout(width=900, height=400)
fig.show()
```

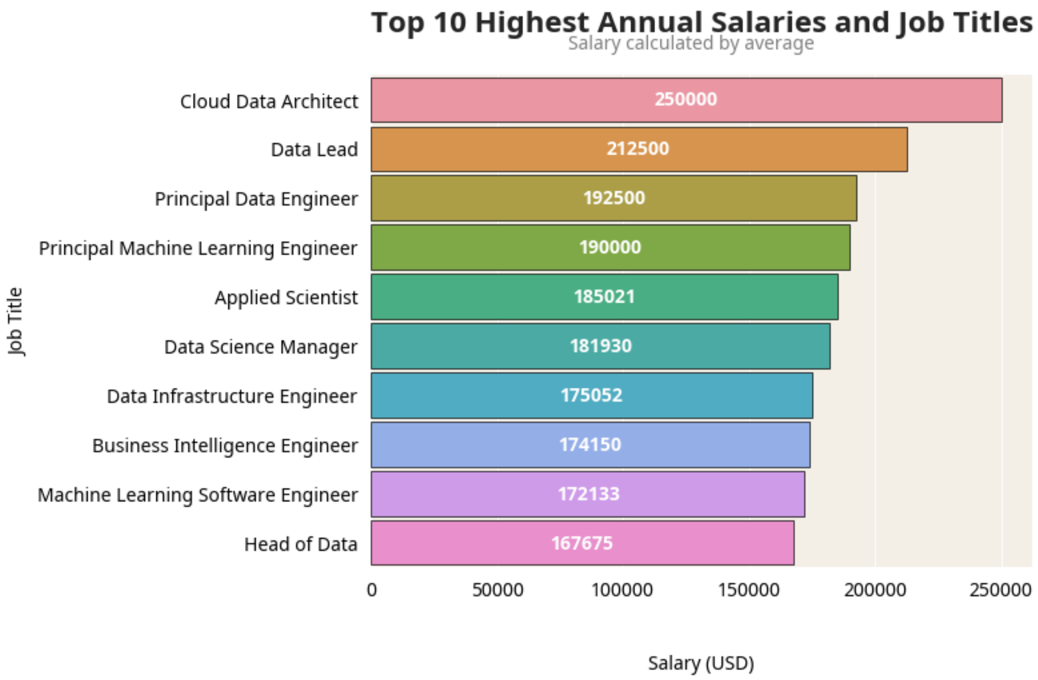


Almost all of the companies in our dataset are located in `US`

3.2 | **Bivariate Analysis**

Top 10 Highest Salaries with Title

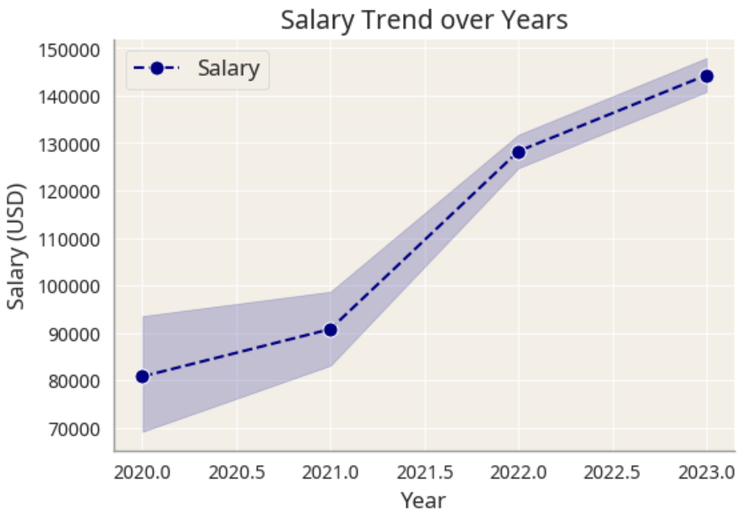
In [24]:
```
top_salary = df.groupby('job_title').agg({'salary_in_usd':'mean'}).round(2)\
                                    .sort_values('salary_in_usd', ascending=False).head(10)

ax= sns.barplot(y = top_salary.index, x ='salary_in_usd',
            data = top_salary, edgecolor='black',
            width=0.9)

plt.title('Top 10 Highest Annual Salaries and Job Titles\n', fontsize=16, fontweight="bold", loc="center")
plt.suptitle("\nSalary calculated by average\n", fontsize = 10, color="gray")
plt.xlabel('\n\nSalary (USD)', color="black", fontsize=10)
plt.ylabel('Job Title', color="black", fontsize=10)
plt.xticks(fontsize=10, color="black")
plt.yticks(fontsize=10, color="black")

for i in ax.containers:
    ax.bar_label(i, size=10, label_type = "center", color="white", fontweight="bold")

plt.show()
```



Salary Trend over Years

In [25]:
```
plt.figure(figsize=(6, 4))
```

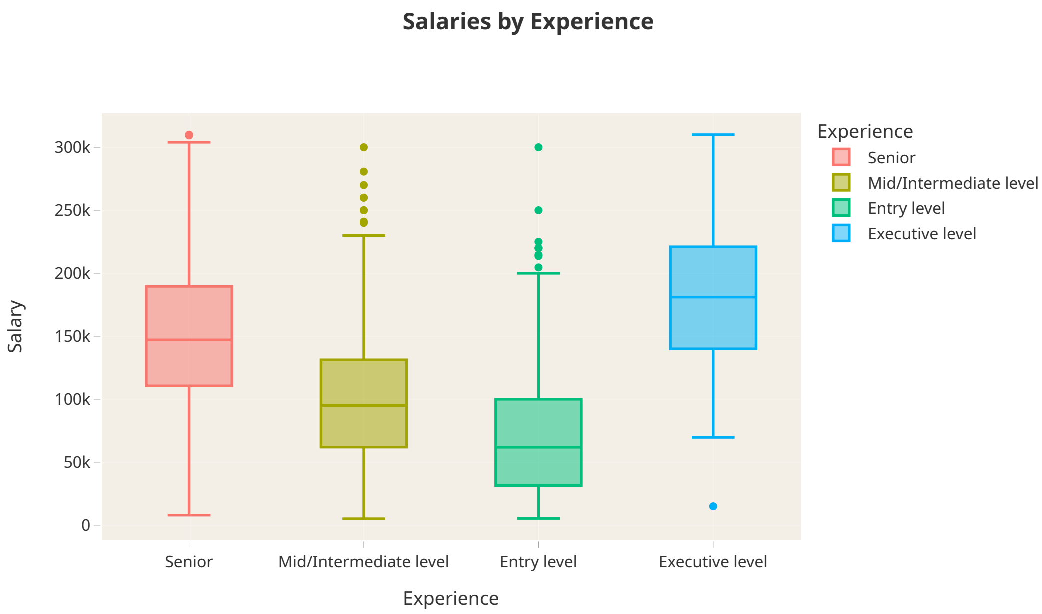
```
salary_trend = df[['salary_in_usd', 'work_year']].sort_values(by='work_year')
p = sns.lineplot(data=salary_trend, x='work_year', y='salary_in_usd',
                 marker='o', linestyle='--', color='navy', markersize=8)

plt.title('Salary Trend over Years', fontsize=14)
plt.xlabel('Year', fontsize=12)
plt.ylabel('Salary (USD)', fontsize=12)

p.legend(['Salary'], loc='upper left', fontsize=12)
sns.despine()
plt.show()
```



We can see that the `salaries` have been `increasing` consistently over time

Salary by Experience

In [26]:
```
fig = px.box(df,x='experience_level',y='salary_in_usd',
       color='experience_level',
       template='ggplot2',
       labels={'experience_level':'Experience','salary_in_usd':'Salary'},
       title='<b>Salaries by Experience')
# Update layout for better appearance
fig.update_layout(
    font=dict(size=12, family="Noto Sans"),
    plot_bgcolor="#F3EFE7"  # Set the background color
)
fig.update_layout(title_x=0.5, title_y=0.95)
fig.update_layout(width=800, height=500)
```

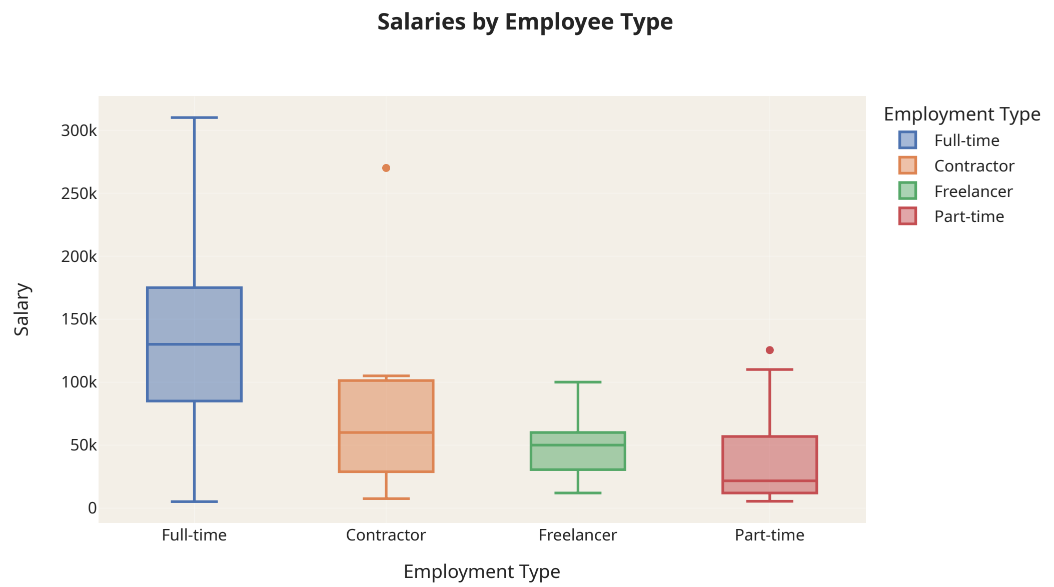


We can see that the median `salary goes up` as you progress in experience

Salary by Employee Type

In [27]:

```
fig = px.box(df,x='employment_type',y='salary_in_usd',color='employment_type',
       template='seaborn',
       labels={'employment_type':'Employment Type','salary_in_usd':'Salary'},
       title='<b>Salaries by Employee Type')
fig.update_layout(
    font=dict(size=12, family="Noto Sans"),
    plot_bgcolor="#F3EFE7"  # Set the background color
)
fig.update_layout(width=800, height=500)
```

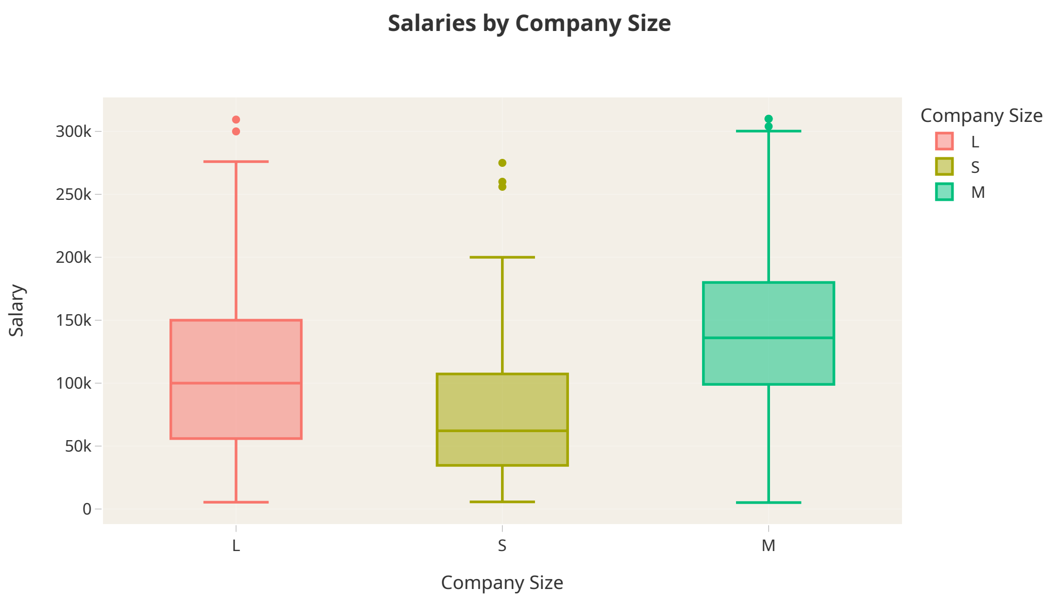


We can see that the `Full-time` employees are paid the `most` and `Part-time` employees are paid the `least`.

Salary by Company Size

In [28]:

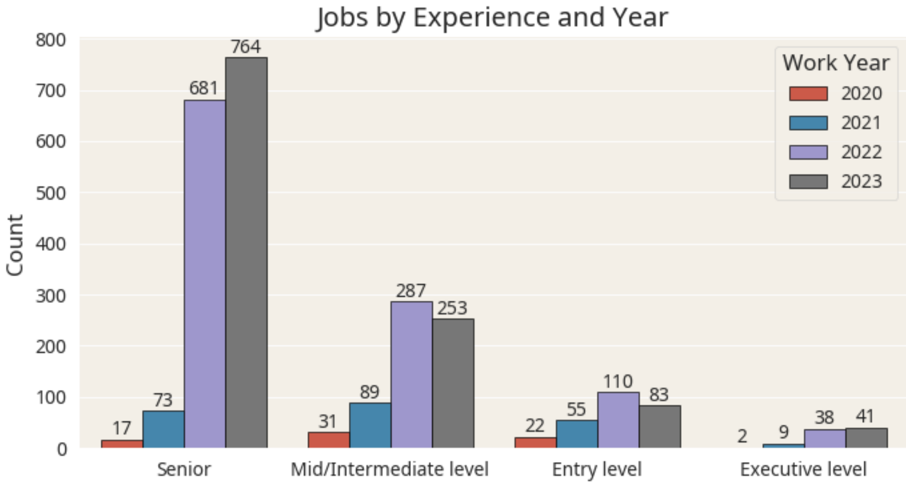
```
fig = px.box(df,x='company_size',y='salary_in_usd',
       color='company_size',template='ggplot2',
       labels={'company_size':'Company Size','salary_in_usd':'Salary'},
       title='<b>Salaries by Company Size')
fig.update_layout(
    font=dict(size=12, family="Noto Sans"),
    plot_bgcolor="#F3EFE7"  # Set the background color
)
fig.update_layout(width=800, height=500)
```



`Medium` sized company pay the `most`. `Small` sized companies pay the `least`.

Jobs by Experience and Work Year

In [29]:
```
plt.figure(figsize=(8, 4))
ax = sns.countplot(data=df, x='experience_level', hue='work_year', edgecolor='black')
plt.ylabel('Count')
plt.xlabel('\n')
plt.title('Jobs by Experience and Year')

for spine in ax.spines.values():
    spine.set_linewidth(2)

for p in ax.patches:
    ax.annotate(format(p.get_height(), '.0f'), (p.get_x() + p.get_width() / 2., p.get_height()),
                ha='center', va='center', xytext=(0, 5), textcoords='offset points')

plt.legend(title='Work Year', title_fontsize=12, loc='upper right')
plt.show()
```

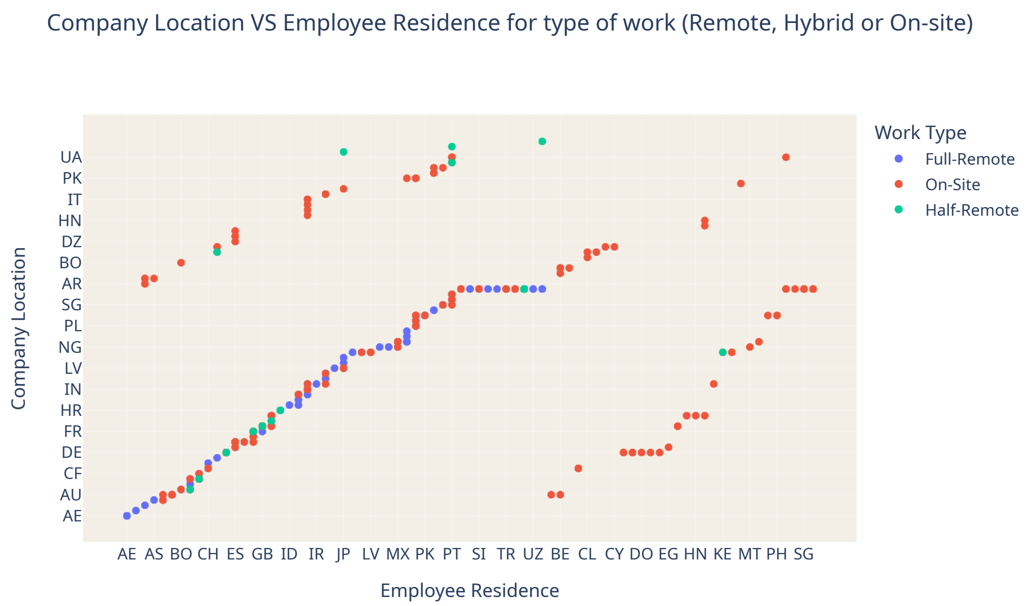


`Jobs` for `Senior` experience levels have dramatically `increased` per year.

3.3 | Multivariate Analysis

Company Location vs Employee Residence by Work Type

In [30]:
```
fig = px.scatter(df, x=df.employee_residence.sort_values(), y=df.company_location.sort_values(), color='remote_ratio',
              labels={"x": 'Employee Residence', "y": 'Company Location', "remote_ratio": 'Work Type'},
              title='Company Location VS Employee Residence for type of work (Remote, Hybrid or On-site)')
fig.update_layout(
    font=dict(size=12, family="Noto Sans"),
    plot_bgcolor="#F3EFE7",
    width=800,
    height=500
)
fig.update_layout(title_x=0.5, title_y=0.95)
fig.show()
```



Some of the employees are working from a different country i.e `working abroad`. We can try to make this insight into a feature.

Top Titles by Salary, Experience and Year

In [31]:
```
top4_job_titles = df['job_title'].value_counts()[:4].index
filtered_df = df[df['job_title'].isin(top4_job_titles)]

fig = px.scatter(filtered_df, x='salary_in_usd',
          y='experience_level',
          size='salary_in_usd',
          hover_name='job_title',
          color='job_title',
          labels={"salary_in_usd": 'Salary', "experience_level": 'Experience', "job_title": 'Job Title'},
          color_discrete_sequence=px.colors.qualitative.Safe,
          animation_frame='work_year',
          title='Top 4 Titles by Experience, Salary and Year')

fig.update_layout(
    font=dict(size=12, family="Noto Sans"),
    plot_bgcolor="#F3EFE7",
    width=800,
    height=500
)
fig.update_layout(title_x=0.5, title_y=0.95)
fig.show()
```

We can clearly see the number of `jobs` as well as their respective `salaries` `increasing` every year

4 | Data Preprocessing

- `Feature Engineering`
- Data `Transformations`
- `Encoding`
- Scaling
- Splitting `Features` and `Target`

4.1 | Feature Engineering

Creating a new feature `Working Abroad` which will be: 0 if the employee's workplace matches their country of residence and 1 if they are employed in a different country.

In [32]:
```
df['working_abroad'] = [0 if i == j else 1 for i,j in zip(df['employee_residence'], df['company_location']) ]
```

Adding new features: `employee_group_mean` , `employee_group_min` , and `employee_group_max` .
These features help us understand salary statistics for different groups of employees based on their work year, experience level, job title, location, and other factors.

In [33]:
```
group_employees = ['work_year','experience_level','employment_type','job_title','employee_residence',
                   'remote_ratio','company_location','company_size','working_abroad']

df['employee_group_mean'] = df.groupby(group_employees)['salary_in_usd'].transform('mean')
df['employee_group_min'] = df.groupby(group_employees)['salary_in_usd'].transform('min')
df['employee_group_max'] = df.groupby(group_employees)['salary_in_usd'].transform('max')
```

4.2 | Data Transformation

Combining similar job titles into more generalized categories for easier analysis and interpretation

In [34]:
```
df['job_title'] = df['job_title'].apply(lambda x: 'Data Scientist' if 'Data Scien' in x or x =='AI Scientist'
                                        else x)

df['job_title'] = df['job_title'].apply(lambda x: 'Data Analyst' if 'Data Anal' in x
                                        or x in ['Data Specialist', 'ETL Developer', 'Analytics Engineer', 'Head of Data']
                                        else x)

df['job_title'] = df['job_title'].apply(lambda x: 'Machine Learning Engineer' if 'Machine Learning' in x
                                        or x in ['ML Engineer', 'NLP Engineer']
                                        else x)

df['job_title'] = df['job_title'].apply(lambda x: 'Data Engineer'
                                        if 'Data Engineer' in x in ['Data Architect','Big Data Architect']
                                        else x)

df['job_title'] = df['job_title'].apply(lambda x: 'Computer Vision Engineer' if 'Computer Vision' in x
                                        else x)
```

Simplifying our categorical features even more by grouping less common categories under `other` .
This makes our data more concise and easier for the models to work with, as we are focusing on the most important categories while combining the less frequent ones.

In [35]:
```
threshold = 0.01  # 1%
update_cat_cols = ['job_title', 'company_location', 'employee_residence']

for feature in update_cat_cols:
    category_counts = df[feature].value_counts(normalize=True)
    rare_categories = category_counts[category_counts < threshold].index
    df[feature] = df[feature].apply(lambda x: 'other' if x in rare_categories else x)
```

4.3 | Encoding

Encoding `company_size` , `remote_ratio` , `experience_level` , `employment_type`

In [36]:
```
df['company_size'] = df['company_size'].replace({'S':1 ,'M': 2, 'L': 3})

df['remote_ratio'] = df['remote_ratio'].replace({'Full-Remote': 3, 'Half-Remote': 2, 'On-Site': 1})

df['experience_level'] = df['experience_level'].replace({'Entry level': 1, 'Mid/Intermediate level': 2,
                                                         'Senior': 3, 'Executive level': 4})
```

```
df['employment_type'] = df['employment_type'].replace({'Freelancer':1 ,'Part-time': 2, 'Full-time': 3, 'Contractor': 4,})
```

`One Hot` Encoding `job_title`, `company_location`, `employee_residence`

In [37]:
```
update_cat_cols = ['job_title', 'company_location', 'employee_residence']
df = pd.get_dummies(df, columns = update_cat_cols, drop_first=True)
```

4.4 | Scaling

Using `MinMaxScaler` to scale all features except target column

In [38]:
```
scaler = MinMaxScaler()
x = scaler.fit_transform(df.drop(['salary_in_usd'],axis=1))
```

In [39]:
```
pd.DataFrame(x).head(2)
```

Out[39]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
0	1.000	0.667	0.667	1.000	1.000	0.000	0.274	0.274	0.265	0.000	0.000	0.000	0.000	1.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000	0.000	0.000	0.000
1	1.000	0.333	1.000	1.000	0.000	0.000	0.077	0.069	0.082	0.000	0.000	0.000	0.000	0.000	1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000

4.5 | Splitting Features and Target

Handling Target Variance and Skewness

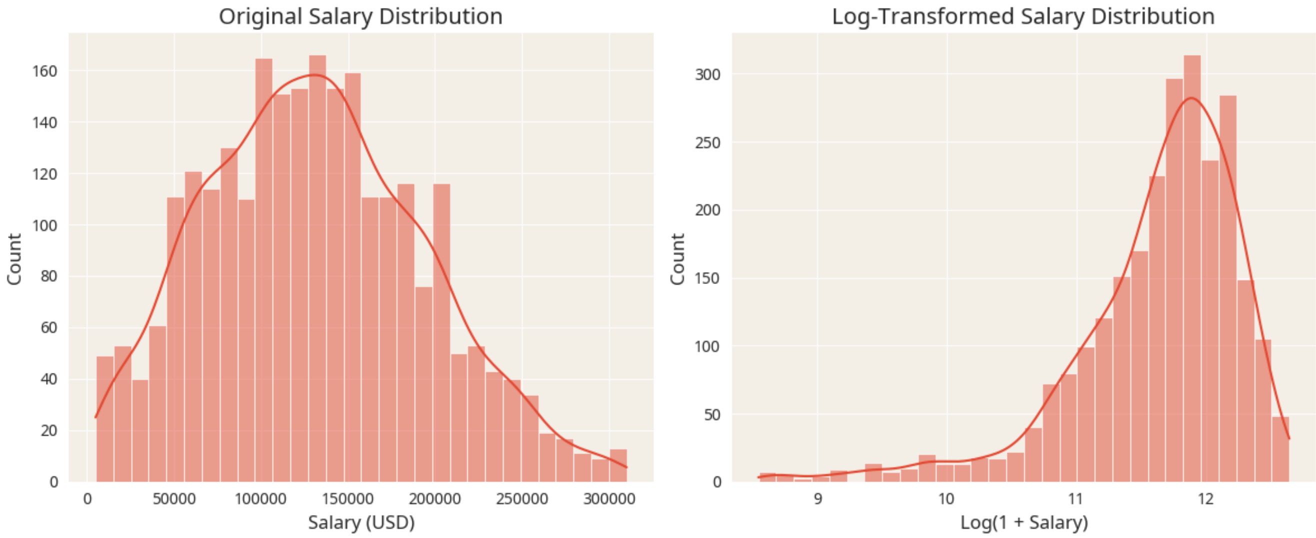
In [40]:
```
salary_variance = df['salary_in_usd'].var()
print("Variance:", round(salary_variance, 2))
salary_skewness = skew(df['salary_in_usd'])
print("Skewness:", round(salary_skewness, 2))
```

```
Variance: 3925829872.3
Skewness: 0.27
```

Since the target has a `high variance` and is slightly right `skewed`, taking the `logarithm` can make the distribution more `suitable` for our models. We can `transform` the target by using natural logarithm - `np.log1p()`

In [41]:
```
y = np.log1p(df['salary_in_usd']).values
```

In [42]:
```
# Original salary
original_salary = df['salary_in_usd']

# Log-transformed salary (add 1 before applying the log)
log_salary = np.log1p(df['salary_in_usd'])

plt.figure(figsize=(12, 5))

# Plot the original salary distribution
plt.subplot(1, 2, 1)
sns.histplot(original_salary, bins=30, kde=True)
plt.title('Original Salary Distribution')
plt.xlabel('Salary (USD)')

# Plot the log-transformed salary distribution
plt.subplot(1, 2, 2)
sns.histplot(log_salary, bins=30, kde=True)
plt.title('Log-Transformed Salary Distribution')
plt.xlabel('Log(1 + Salary)')
plt.tight_layout()
plt.show()
```



5 | Modelling

Train Test Split

In [43]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=.2, random_state=42)
```

Training Models and Storing Predictions

In [44]:
```
ml_algs = {
    'RandomForest' :RandomForestRegressor(),
    'LightGBM' : LGBMRegressor(verbose=-1),
    'Knn' : KNeighborsRegressor(),
    'ExtraTrees' : ExtraTreeRegressor(),
    'GradientBoosting' : GradientBoostingRegressor(),
    'DecisionTree' : DecisionTreeRegressor(),
    'XGB' : XGBRegressor()
}

ml_algs_result = {}

for name, model in ml_algs.items():
    pred = model.fit(x_train, y_train).predict(x_test)
    ml_algs_result[name] = r2_score(y_test, pred)
```

Creating a Model Performance DataFrame

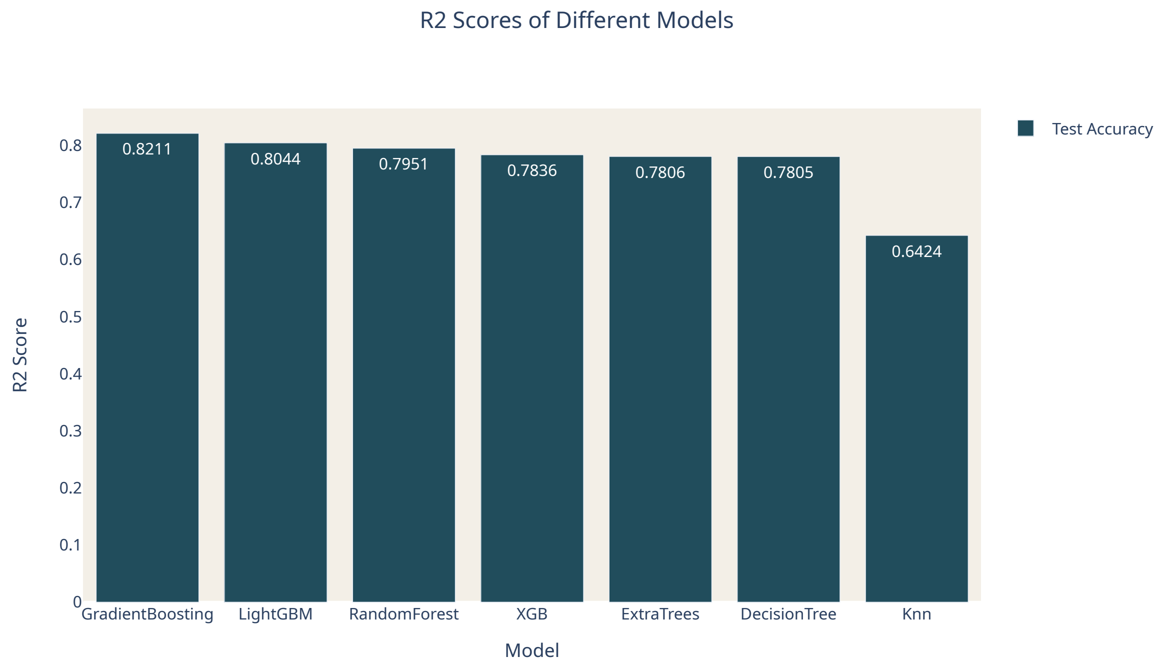
In [45]:
```
sorted_results = dict(sorted(ml_algs_result.items(), key=lambda item: item[1], reverse=True))
model_metrics = pd.DataFrame({'Name': sorted_results.keys(), 'R-squared': sorted_results.values()})
model_metrics
```

Out[45]:

	Name	R-squared
0	GradientBoosting	0.821
1	LightGBM	0.804
2	RandomForest	0.795
3	XGB	0.784
4	ExtraTrees	0.781
5	DecisionTree	0.781
6	Knn	0.642

In [46]:
```
fig = go.Figure()
fig.add_trace(go.Bar(
    x=model_metrics['Name'],
    y=model_metrics['R-squared'],
    name='Test Accuracy',
    marker_color='#214D5C',
    hovertemplate='Model: %{x}<br>Test Accuracy: %{y:.4f}',
    text=model_metrics['R-squared'].apply(lambda x: f'{x:.4f}'),  # Format score to 4 decimal places
    textposition='auto',
    showlegend=True,
))
fig.update_layout(
    font=dict(size=12, family="Noto Sans"),
    plot_bgcolor="#F3EFE7"  # Set the background color
)
fig.update_layout(
    xaxis_title='Model',
    yaxis_title='R2 Score',
    title='R2 Scores of Different Models',
```

```
    width=900,
    height=550,
    xaxis=dict(showgrid=False),
    yaxis=dict(showgrid=False)
)
fig.update_layout(title_x=0.5, title_y=0.95)
```



Hyperparameter Tuning Top 3 Models

Random Forest

In [47]:
```
param_grid = {
    'n_estimators': [55,100,150],
    'criterion': ['mse', 'mae', 'poisson'],
    'max_depth': [4, 5, 6],
    'random_state': [1147]
}
grid_search = GridSearchCV(estimator=RandomForestRegressor(), param_grid=param_grid, cv=5, scoring='r2')
grid_search.fit(x_train, y_train)
best_rf = grid_search.best_estimator_
best_pred = best_rf.predict(x_test)
print("Best RandomForest Hyperparameters:", grid_search.best_params_, '\n')
print('Best R-squared Score:', r2_score(y_test, best_pred))
```

```
Best RandomForest Hyperparameters: {'criterion': 'poisson', 'max_depth': 4, 'n_estimators': 55, 'random_state': 1147}

Best R-squared Score: 0.8304874828485052
```

Gradient Boosting

In [48]:
```
param_grid = {
    'n_estimators': [600, 800],
    'learning_rate': [0.01, 0.05],
    'subsample': [0.9, 0.8],
    'max_depth': [2, 3],
    'random_state': [195, 200]
}
grid_search = GridSearchCV(estimator=GradientBoostingRegressor(), param_grid=param_grid, cv=5, scoring='r2')
grid_search.fit(x_train, y_train)
best_gb = grid_search.best_estimator_
best_pred = best_gb.predict(x_test)
print("Best Hyperparameters:", grid_search.best_params_, '\n')
print("Best R-squared Score:", r2_score(y_test, best_pred))
```

```
Best Hyperparameters: {'learning_rate': 0.01, 'max_depth': 2, 'n_estimators': 600, 'random_state': 200, 'subsample': 0.8}

Best R-squared Score: 0.8299692937961031
```

Extreme Gradient Boosting

In [49]:
```
param_grid = {
    'booster': ['gbtree'],
```

```
    'eta': [0.098, 0.1],
    'min_child_weight': [2, 3],
    'max_depth': [2, 3],
    'gamma': [0.01, 0.02],
    'max_delta_step': [5, 6],
    'reg_alpha': [0.0098, 0.01],
    'objective': ['reg:squarederror']
}
grid_search = GridSearchCV(estimator=XGBRegressor(), param_grid=param_grid, cv=5, scoring='r2')
grid_search.fit(x_train, y_train)
best_xgb = grid_search.best_estimator_
best_pred = best_xgb.predict(x_test)
print("Best Hyperparameters:", grid_search.best_params_, '\n')
print("Best R-squared Score:", r2_score(y_test, best_pred))
```

```
Best Hyperparameters: {'booster': 'gbtree', 'eta': 0.1, 'gamma': 0.02, 'max_delta_step': 5, 'max_depth': 2, 'min_child_weight': 3, 'object
ive': 'reg:squarederror', 'reg_alpha': 0.01}

Best R-squared Score: 0.8288995354986279
```

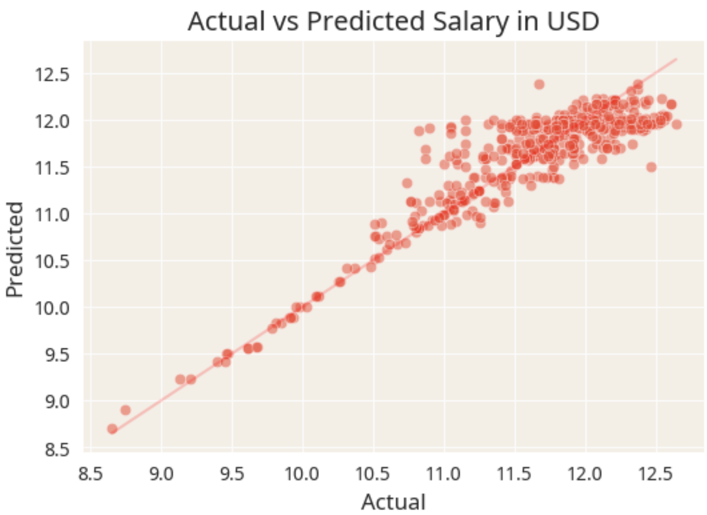
Stacking Models

In [50]:
```
base_regressors = [best_gb, best_xgb, best_rf]
stack_gen = StackingCVRegressor(regressors=base_regressors,
                                meta_regressor=best_xgb, # Model that learns from the predictions of the base models
                                use_features_in_secondary=True, # Features used by the base models will be used for the meta-regressor.
                                random_state=183
)
pred = stack_gen.fit(x_train, y_train).predict(x_test)
print(f'Final R2 Score of Stacked Models : {round(r2_score(y_test, pred) * 100, 2)}%')
```

```
Final R2 Score of Stacked Models : 82.94%
```

Actual Values vs Predicted Values

In [51]:
```
plt.figure(figsize=(6, 4))
sns.scatterplot(x=y_test, y=pred, alpha=0.5)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', alpha = 0.2)
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.title('Actual vs Predicted Salary in USD')
plt.show()
```



Project by Priyanka Singh