In [1]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
sns.set_theme(color_codes=True)
```

In [2]:
```
df = pd.read_csv('ds_salaries.csv')
df.head()
```

Out[2]:

	work_year	experience_level	employment_type	job_title	salary	salary_currency	salary_in_usd	empl
0	2023	SE	FT	Principal Data Scientist	80000	EUR	85847	
1	2023	MI	CT	ML Engineer	30000	USD	30000	
2	2023	MI	CT	ML Engineer	25500	USD	25500	
3	2023	SE	FT	Data Scientist	175000	USD	175000	
4	2023	SE	FT	Data Scientist	120000	USD	120000	

Data Preprocessing Part 1

In [3]:
```
#drop salary column because there's salary in usd column
#drop salary_currency column to make it universal by using only usd
df.drop(columns=['salary', 'salary_currency'], inplace=True)
df.head()
```

Out[3]:

	work_year	experience_level	employment_type	job_title	salary_in_usd	employee_residence	remote_ra
0	2023	SE	FT	Principal Data Scientist	85847	ES	1
1	2023	MI	CT	ML Engineer	30000	US	1
2	2023	MI	CT	ML Engineer	25500	US	1
3	2023	SE	FT	Data Scientist	175000	CA	1
4	2023	SE	FT	Data Scientist	120000	CA	1

In [4]:
```
#Check the missing value
check_missing = df.isnull().sum() * 100 / df.shape[0]
check_missing[check_missing > 0].sort_values(ascending=False)
```

Out[4]:
```
Series([], dtype: float64)
```

In [5]:
```
#Check the number of unique value on object datatype
df.select_dtypes(include='object').nunique()
```

Out[5]:
```
experience_level       4
employment_type        4
job_title             93
employee_residence    78
company_location      72
company_size           3
dtype: int64
```

Categorize the Job Title

In [6]:
```
df.job_title.unique()
```

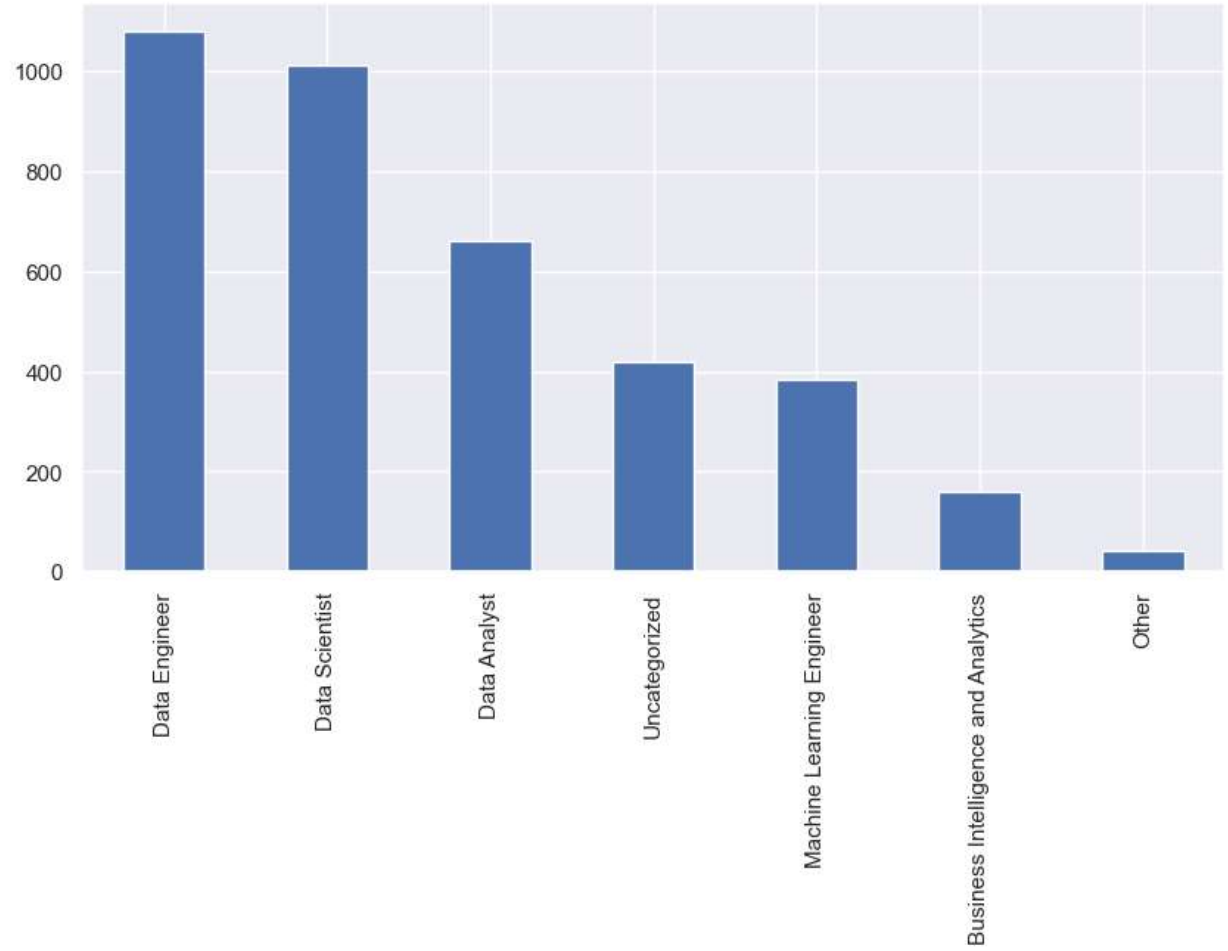
Out[6]:
```
array(['Principal Data Scientist', 'ML Engineer', 'Data Scientist',
       'Applied Scientist', 'Data Analyst', 'Data Modeler',
       'Research Engineer', 'Analytics Engineer',
       'Business Intelligence Engineer', 'Machine Learning Engineer',
       'Data Strategist', 'Data Engineer', 'Computer Vision Engineer',
       'Data Quality Analyst', 'Compliance Data Analyst',
       'Data Architect', 'Applied Machine Learning Engineer',
       'AI Developer', 'Research Scientist', 'Data Analytics Manager',
       'Business Data Analyst', 'Applied Data Scientist',
       'Staff Data Analyst', 'ETL Engineer', 'Data DevOps Engineer',
       'Head of Data', 'Data Science Manager', 'Data Manager',
       'Machine Learning Researcher', 'Big Data Engineer',
       'Data Specialist', 'Lead Data Analyst', 'BI Data Engineer',
       'Director of Data Science', 'Machine Learning Scientist',
       'MLOps Engineer', 'AI Scientist', 'Autonomous Vehicle Technician',
       'Applied Machine Learning Scientist', 'Lead Data Scientist',
       'Cloud Database Engineer', 'Financial Data Analyst',
       'Data Infrastructure Engineer', 'Software Data Engineer',
       'AI Programmer', 'Data Operations Engineer', 'BI Developer',
       'Data Science Lead', 'Deep Learning Researcher', 'BI Analyst',
       'Data Science Consultant', 'Data Analytics Specialist',
       'Machine Learning Infrastructure Engineer', 'BI Data Analyst',
       'Head of Data Science', 'Insight Analyst',
       'Deep Learning Engineer', 'Machine Learning Software Engineer',
       'Big Data Architect', 'Product Data Analyst',
       'Computer Vision Software Engineer', 'Azure Data Engineer',
       'Marketing Data Engineer', 'Data Analytics Lead', 'Data Lead',
       'Data Science Engineer', 'Machine Learning Research Engineer',
       'NLP Engineer', 'Manager Data Management',
       'Machine Learning Developer', '3D Computer Vision Researcher',
       'Principal Machine Learning Engineer', 'Data Analytics Engineer',
       'Data Analytics Consultant', 'Data Management Specialist',
       'Data Science Tech Lead', 'Data Scientist Lead',
       'Cloud Data Engineer', 'Data Operations Analyst',
       'Marketing Data Analyst', 'Power BI Developer',
       'Product Data Scientist', 'Principal Data Architect',
       'Machine Learning Manager', 'Lead Machine Learning Engineer',
       'ETL Developer', 'Cloud Data Architect', 'Lead Data Engineer',
       'Head of Machine Learning', 'Principal Data Analyst',
       'Principal Data Engineer', 'Staff Data Scientist',
       'Finance Data Analyst'], dtype=object)
```

In [7]:
```
def segment_job_title(job_title):
    data_scientist_titles = ['Principal Data Scientist', 'Data Scientist', 'Applied S
    machine_learning_titles = ['ML Engineer', 'Machine Learning Engineer', 'Applied M
    data_analyst_titles = ['Data Analyst', 'Data Quality Analyst', 'Compliance Data A
    data_engineer_titles = ['Data Modeler', 'Data Engineer', 'ETL Engineer', 'Data De
    bi_analytics_titles = ['Data Analytics Manager', 'Business Intelligence Engineer'
    other_titles = ['Data Strategist', 'Computer Vision Engineer', 'AI Developer', 'H

    if job_title in data_scientist_titles:
        return 'Data Scientist'
    elif job_title in machine_learning_titles:
        return 'Machine Learning Engineer'
    elif job_title in data_analyst_titles:
        return 'Data Analyst'
    elif job_title in data_engineer_titles:
        return 'Data Engineer'
    elif job_title in bi_analytics_titles:
        return 'Business Intelligence and Analytics'
    elif job_title in other_titles:
        return 'Other'
    else:
        return 'Uncategorized'
```

In [8]:
```
df['job_title'] = df['job_title'].apply(segment_job_title)
```

In [9]:
```
plt.figure(figsize=(10,5))
df['job_title'].value_counts().plot(kind='bar')
```

Out[9]:
```
<AxesSubplot:>
```



Categorize the Employee Residence

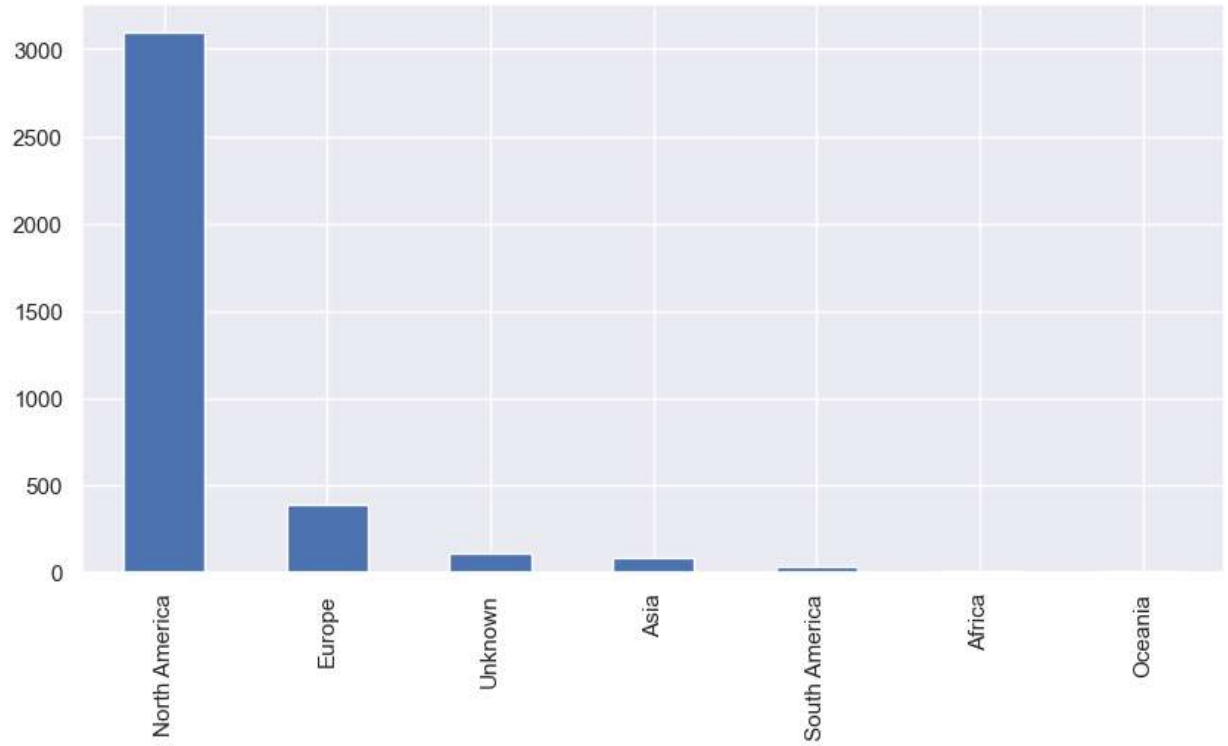
In [10]:
```
df.employee_residence.unique()
```

Out[10]:
```
array(['ES', 'US', 'CA', 'DE', 'GB', 'NG', 'IN', 'HK', 'PT', 'NL', 'CH',
       'CF', 'FR', 'AU', 'FI', 'UA', 'IE', 'IL', 'GH', 'AT', 'CO', 'SG',
       'SE', 'SI', 'MX', 'UZ', 'BR', 'TH', 'HR', 'PL', 'KW', 'VN', 'CY',
       'AR', 'AM', 'BA', 'KE', 'GR', 'MK', 'LV', 'RO', 'PK', 'IT', 'MA',
       'LT', 'BE', 'AS', 'IR', 'HU', 'SK', 'CN', 'CZ', 'CR', 'TR', 'CL',
       'PR', 'DK', 'BO', 'PH', 'DO', 'EG', 'ID', 'AE', 'MY', 'JP', 'EE',
       'HN', 'TN', 'RU', 'DZ', 'IQ', 'BG', 'JE', 'RS', 'NZ', 'MD', 'LU',
       'MT'], dtype=object)
```

In [11]:
```
# Define a function to categorize the unique values
def categorize_region(country):
    if country in ['DE', 'GB', 'PT', 'NL', 'CH', 'CF', 'FR', 'FI', 'UA', 'IE', 'AT',
        return 'Europe'
    elif country in ['US', 'CA', 'MX']:
        return 'North America'
    elif country in ['BR', 'AR', 'CL', 'BO', 'CR', 'DO', 'PR', 'HN', 'UY']:
        return 'South America'
    elif country in ['NG', 'GH', 'KE', 'TN', 'DZ']:
        return 'Africa'
    elif country in ['HK', 'IN', 'CN', 'JP', 'KR', 'BD', 'VN', 'PH', 'MY', 'ID', 'AE'
        return 'Asia'
    elif country in ['AU', 'NZ']:
        return 'Oceania'
    else:
        return 'Unknown'

# Apply the function to the "employee residence" column to create a new column with th
df['employee_residence'] = df['employee_residence'].apply(categorize_region)
```

In [12]:
```
plt.figure(figsize=(10,5))
df['employee_residence'].value_counts().plot(kind='bar')
```

Out[12]: <AxesSubplot:>



Categorize the Company Location

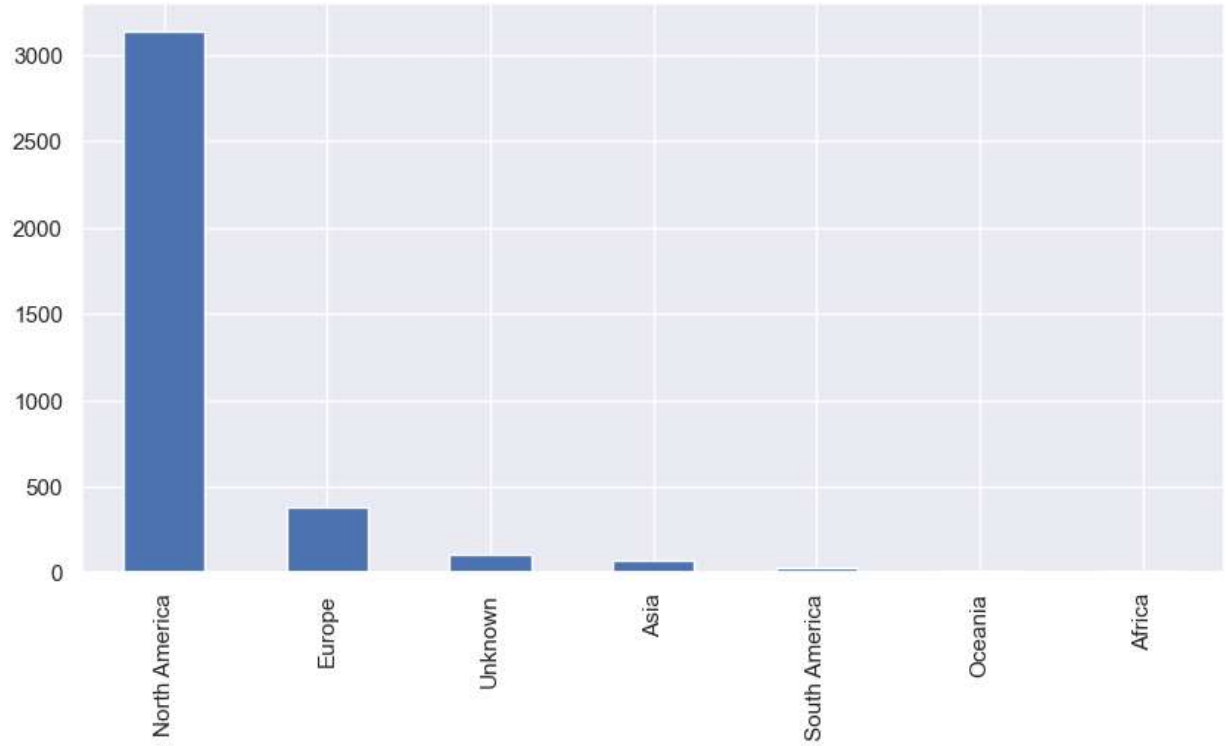
In [13]:
```
df.company_location.unique()
```

Out[13]:
```
array(['ES', 'US', 'CA', 'DE', 'GB', 'NG', 'IN', 'HK', 'NL', 'CH', 'CF',
       'FR', 'FI', 'UA', 'IE', 'IL', 'GH', 'CO', 'SG', 'AU', 'SE', 'SI',
       'MX', 'BR', 'PT', 'RU', 'TH', 'HR', 'VN', 'EE', 'AM', 'BA', 'KE',
       'GR', 'MK', 'LV', 'RO', 'PK', 'IT', 'MA', 'PL', 'AL', 'AR', 'LT',
       'AS', 'CR', 'IR', 'BS', 'HU', 'AT', 'SK', 'CZ', 'TR', 'PR', 'DK',
       'BO', 'PH', 'BE', 'ID', 'EG', 'AE', 'LU', 'MY', 'HN', 'JP', 'DZ',
       'IQ', 'CN', 'NZ', 'CL', 'MD', 'MT'], dtype=object)
```

In [14]:
```
# Define a function to categorize the unique values
def categorize_region(country):
    if country in ['DE', 'GB', 'PT', 'NL', 'CH', 'CF', 'FR', 'FI', 'UA', 'IE', 'AT',
        return 'Europe'
    elif country in ['US', 'CA', 'MX']:
        return 'North America'
    elif country in ['BR', 'AR', 'CL', 'BO', 'CR', 'DO', 'PR', 'HN', 'UY']:
        return 'South America'
    elif country in ['NG', 'GH', 'KE', 'TN', 'DZ']:
        return 'Africa'
    elif country in ['HK', 'IN', 'CN', 'JP', 'KR', 'BD', 'VN', 'PH', 'MY', 'ID', 'AE'
        return 'Asia'
    elif country in ['AU', 'NZ']:
        return 'Oceania'
    else:
        return 'Unknown'

# Apply the function to the "company_location" column to create a new column with the
df['company_location'] = df['company_location'].apply(categorize_region)
```

In [15]:
```
plt.figure(figsize=(10,5))
df['company_location'].value_counts().plot(kind='bar')
```

Out[15]: <AxesSubplot:>



In [16]:
```
#Check the number of unique value on object datatype
df.select_dtypes(include='object').nunique()
```

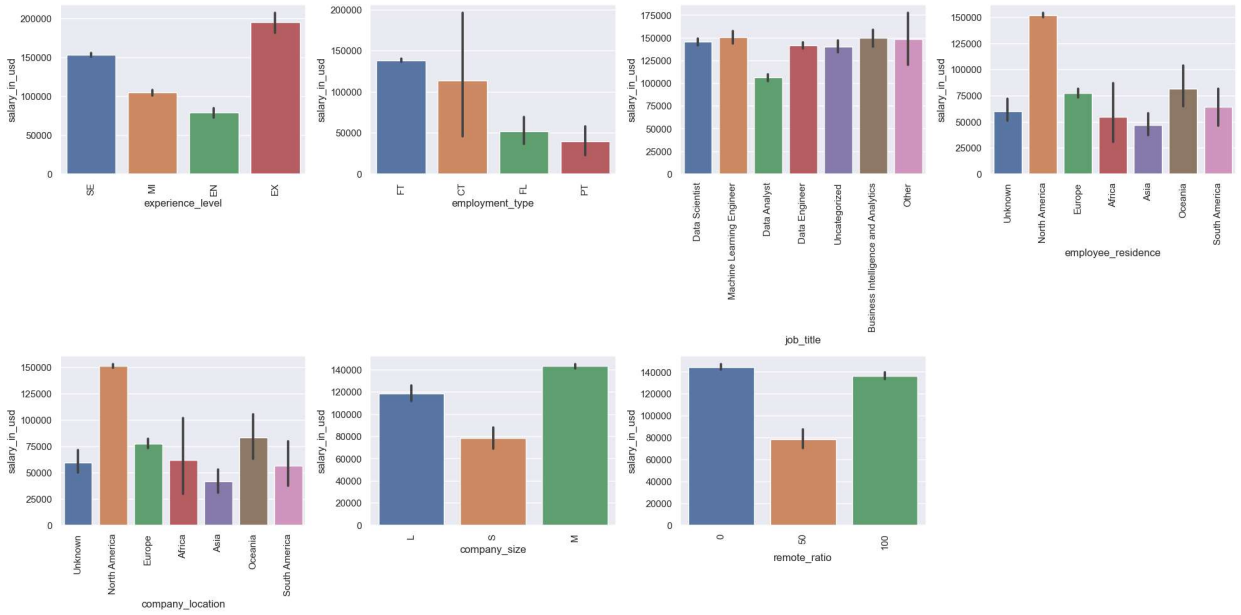
Out[16]:
```
experience_level      4
employment_type       4
job_title             7
employee_residence    7
company_location      7
company_size          3
dtype: int64
```

Exploratory Data Analysis

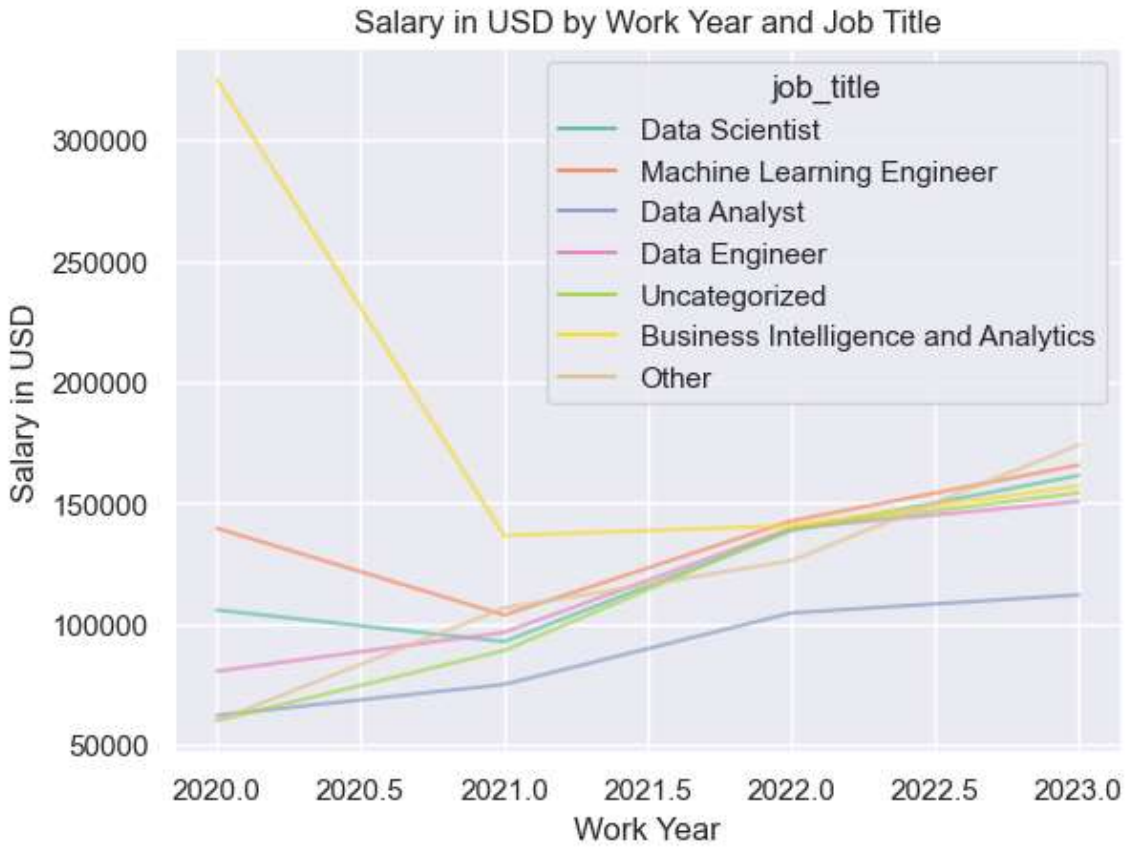
In [19]:
```
df.remote_ratio.unique()
```

Out[19]: array([100, 0, 50], dtype=int64)

In [20]:
```
# list of categorical variables to plot
cat_vars = ['experience_level', 'employment_type', 'job_title', 'employee_residence',
            'company_location', 'company_size', 'remote_ratio']

# create figure with subplots
fig, axs = plt.subplots(nrows=2, ncols=4, figsize=(20, 10))
axs = axs.flatten()

# create barplot for each categorical variable
for i, var in enumerate(cat_vars):
    sns.barplot(x=var, y='salary_in_usd', data=df, ax=axs[i], estimator=np.mean)
    axs[i].set_xticklabels(axs[i].get_xticklabels(), rotation=90)

# remove the eighth subplot
fig.delaxes(axs[7])

# adjust spacing between subplots
fig.tight_layout()

# show plot
plt.show()
```



In [21]:
```
sns.set_style("darkgrid")
sns.set_palette("Set2")

sns.lineplot(x='work_year', y='salary_in_usd', hue='job_title', data=df, ci=None, est

plt.title("Salary in USD by Work Year and Job Title")
plt.xlabel("Work Year")
plt.ylabel("Salary in USD")

plt.show()
```



Data Preprocessing Part 2

Label Encoding for Object datatype

In [22]:
```
# Loop over each column in the DataFrame where dtype is 'object'
for col in df.select_dtypes(include=['object']).columns:

    # Print the column name and the unique values
    print(f"{col}: {df[col].unique()}")
```

```
experience_level: ['SE' 'MI' 'EN' 'EX']
employment_type: ['FT' 'CT' 'FL' 'PT']
job_title: ['Data Scientist' 'Machine Learning Engineer' 'Data Analyst'
 'Data Engineer' 'Uncategorized' 'Business Intelligence and Analytics'
 'Other']
employee_residence: ['Unknown' 'North America' 'Europe' 'Africa' 'Asia' 'Oceania'
 'South America']
company_location: ['Unknown' 'North America' 'Europe' 'Africa' 'Asia' 'Oceania'
 'South America']
company_size: ['L' 'S' 'M']
```

In [23]:
```
from sklearn import preprocessing

# Loop over each column in the DataFrame where dtype is 'object'
for col in df.select_dtypes(include=['object']).columns:

    # Initialize a LabelEncoder object
    label_encoder = preprocessing.LabelEncoder()

    # Fit the encoder to the unique values in the column
    label_encoder.fit(df[col].unique())

    # Transform the column using the encoder
    df[col] = label_encoder.transform(df[col])

    # Print the column name and the unique encoded values
    print(f"{col}: {df[col].unique()}")
```

```
experience_level: [3 2 0 1]
employment_type: [2 0 1 3]
job_title: [3 4 1 2 6 0 5]
employee_residence: [6 3 2 0 1 4 5]
company_location: [6 3 2 0 1 4 5]
company_size: [0 2 1]
```

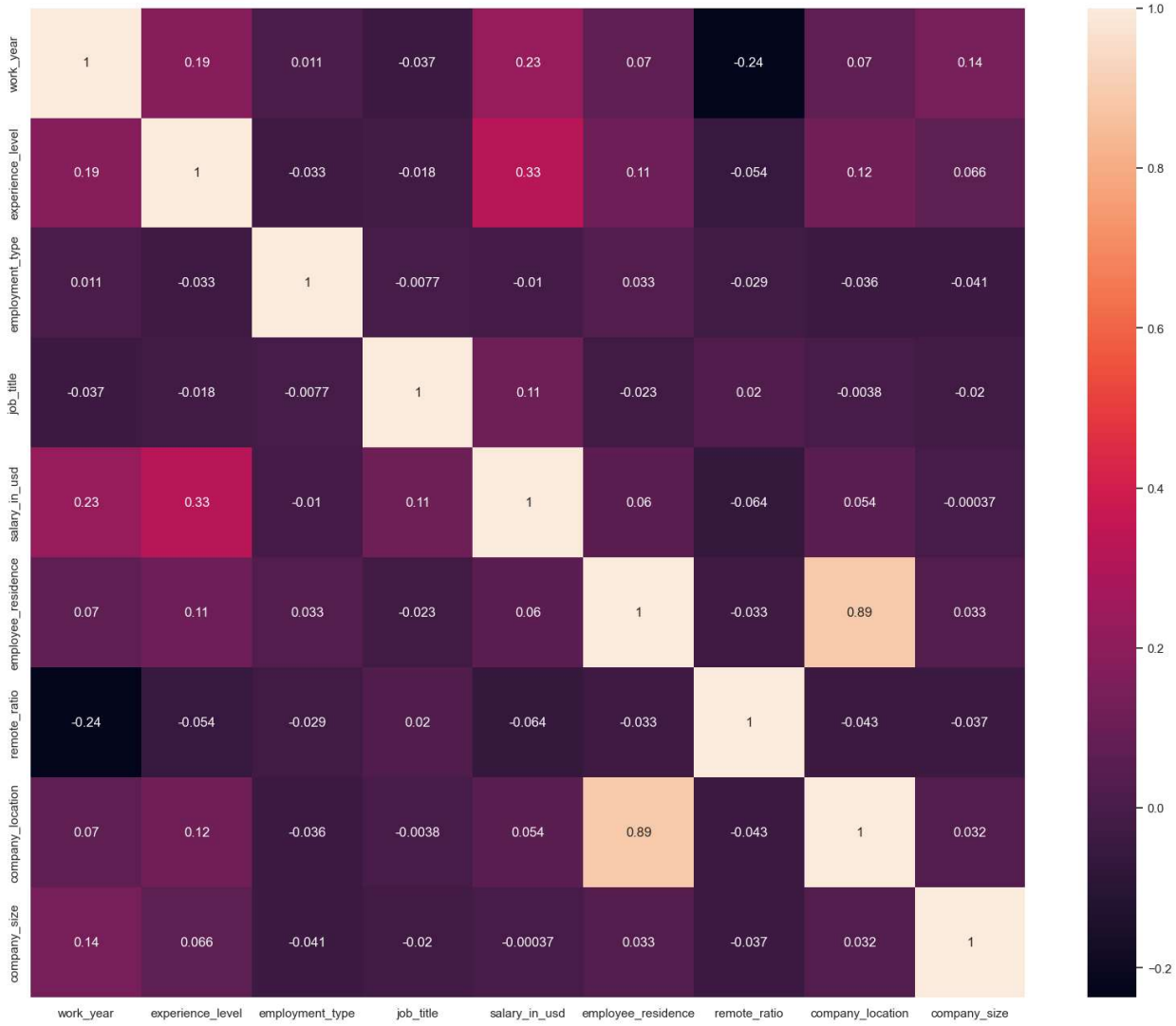
In [24]:
```
df.dtypes
```

Out[24]:
```
work_year             int64
experience_level      int32
employment_type       int32
job_title             int32
salary_in_usd         int64
employee_residence    int32
remote_ratio          int64
company_location      int32
company_size          int32
dtype: object
```

All of the data are categorial so that means, there are no outliers

In [25]:
```
#Correlation Heatmap
plt.figure(figsize=(20, 16))
sns.heatmap(df.corr(), fmt='.2g', annot=True)
```

Out[25]: <AxesSubplot:>



Train test Split

In [26]:
```
X = df.drop('salary_in_usd', axis=1)
y = df['salary_in_usd']
```

In [27]:
```
#test size 20% and train size 80%
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.2,random_state=0
```

Decision Tree Regressor

In [28]:
```
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.datasets import load_boston


# Create a DecisionTreeRegressor object
dtree = DecisionTreeRegressor()

# Define the hyperparameters to tune and their values
param_grid = {
    'max_depth': [2, 4, 6, 8],
    'min_samples_split': [2, 4, 6, 8],
    'min_samples_leaf': [1, 2, 3, 4],
    'max_features': ['auto', 'sqrt', 'log2']
}

# Create a GridSearchCV object
grid_search = GridSearchCV(dtree, param_grid, cv=5, scoring='neg_mean_squared_error')

# Fit the GridSearchCV object to the data
grid_search.fit(X_train, y_train)

# Print the best hyperparameters
print(grid_search.best_params_)
```

```
{'max_depth': 6, 'max_features': 'auto', 'min_samples_leaf': 3, 'min_samples_split':
4}
```

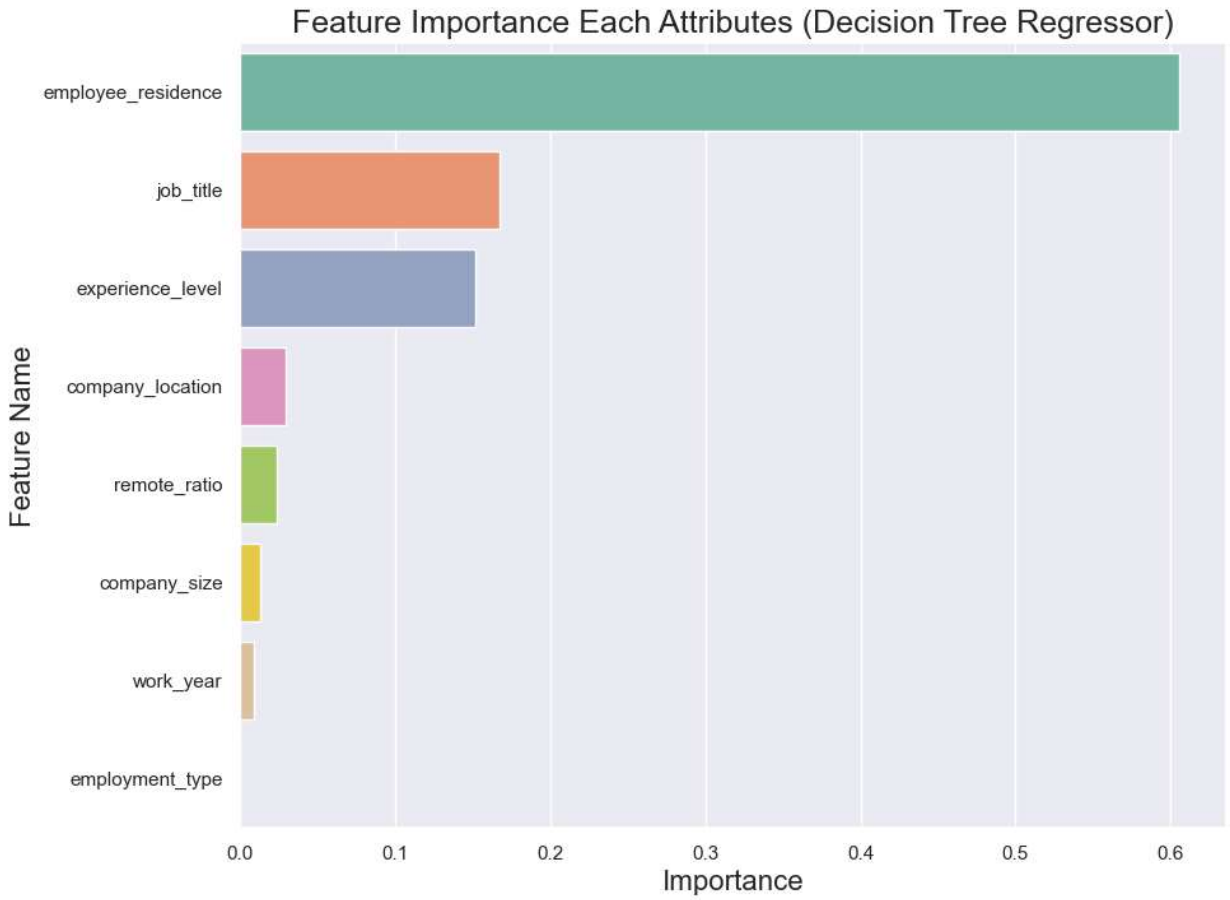
In [29]:
```
from sklearn.tree import DecisionTreeRegressor
dtree = DecisionTreeRegressor(random_state=0, max_depth=6, max_features='auto', min_s
dtree.fit(X_train, y_train)
```

Out[29]:
```
DecisionTreeRegressor(max_depth=6, max_features='auto', min_samples_leaf=3,
                      min_samples_split=4, random_state=0)
```

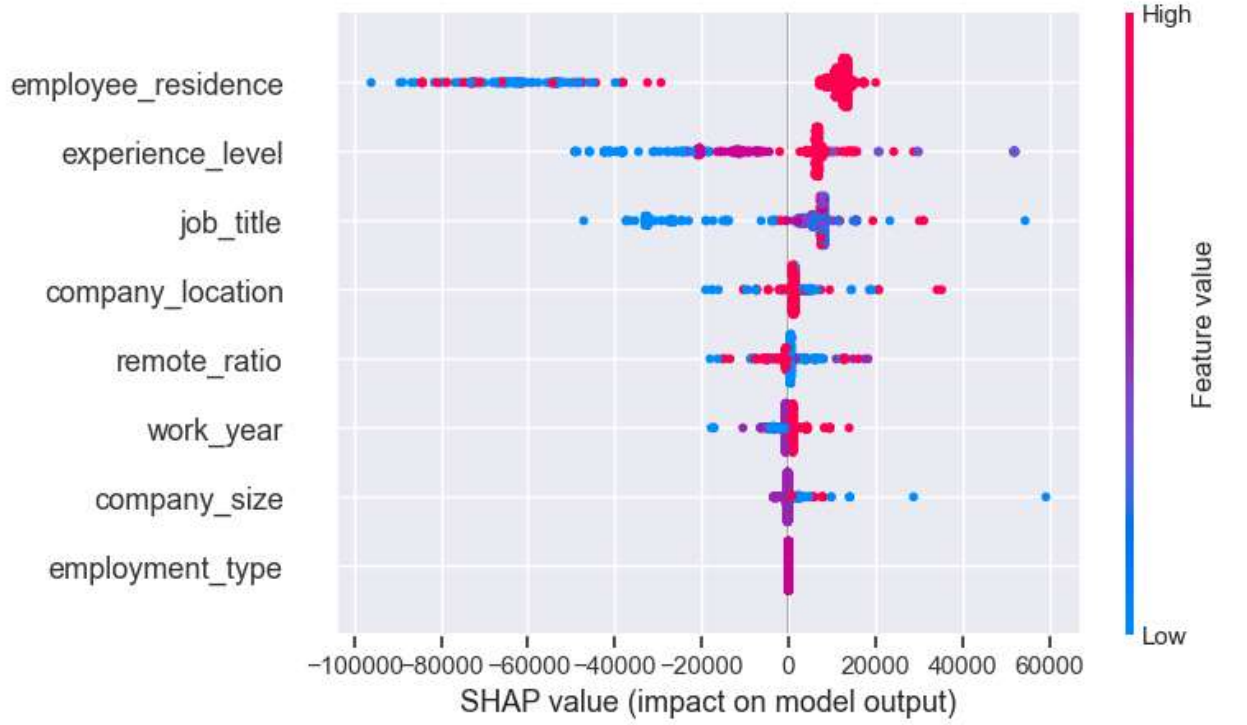
In [30]:
```
from sklearn import metrics
from sklearn.metrics import mean_absolute_percentage_error
import math
y_pred = dtree.predict(X_test)
mae = metrics.mean_absolute_error(y_test, y_pred)
mape = mean_absolute_percentage_error(y_test, y_pred)
mse = metrics.mean_squared_error(y_test, y_pred)
r2 = metrics.r2_score(y_test, y_pred)
rmse = math.sqrt(mse)

print('MAE is {}'.format(mae))
print('MAPE is {}'.format(mape))
print('MSE is {}'.format(mse))
print('R2 score is {}'.format(r2))
print('RMSE score is {}'.format(rmse))
```

```
MAE is 39982.989072133074
MAPE is 0.3966225664928661
MSE is 2890657437.6238027
R2 score is 0.32248347877074923
RMSE score is 53764.834581944015
```

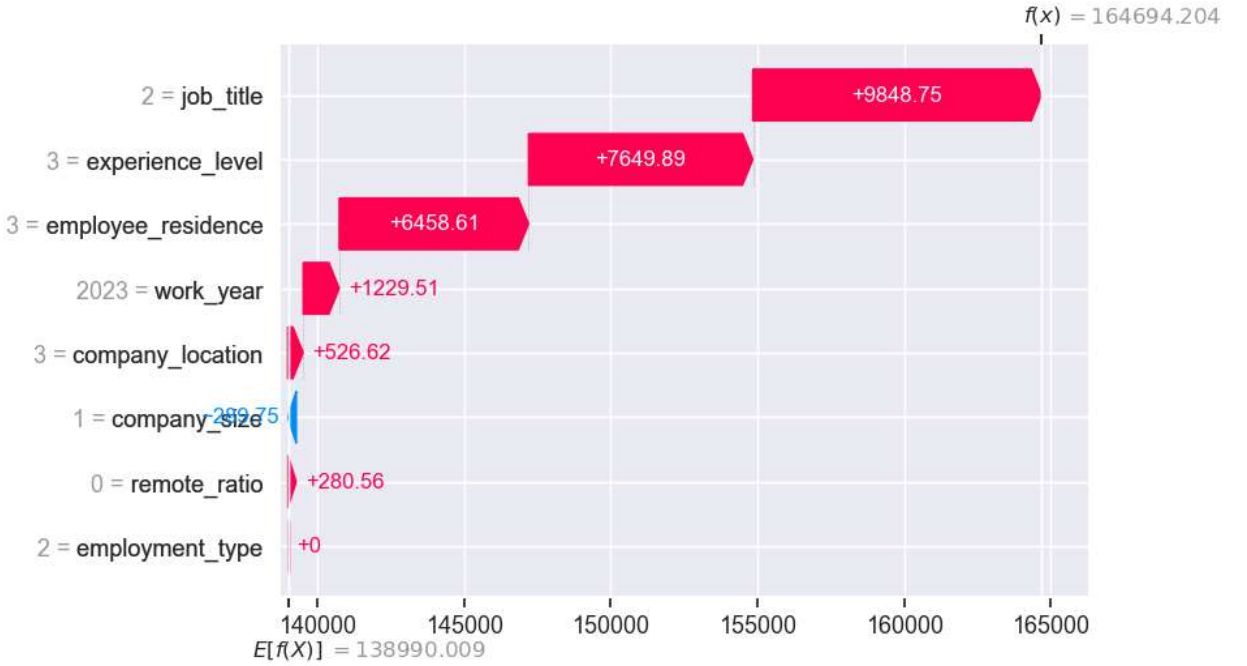
In [31]:
```
imp_df = pd.DataFrame({
    "Feature Name": X_train.columns,
    "Importance": dtree.feature_importances_
})
fi = imp_df.sort_values(by="Importance", ascending=False)

fi2 = fi.head(10)
plt.figure(figsize=(10,8))
sns.barplot(data=fi2, x='Importance', y='Feature Name')
plt.title('Feature Importance Each Attributes (Decision Tree Regressor)', fontsize=18
plt.xlabel ('Importance', fontsize=16)
plt.ylabel ('Feature Name', fontsize=16)
plt.show()
```



In [32]:
```
import shap
explainer = shap.TreeExplainer(dtree)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test)
```



In [33]:
```
explainer = shap.Explainer(dtree, X_test)
shap_values = explainer(X_test)
shap.plots.waterfall(shap_values[0])
```



Random Forest Regressor

In [34]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV

# Create a Random Forest Regressor object
rf = RandomForestRegressor()

# Define the hyperparameter grid
param_grid = {
    'max_depth': [3, 5, 7, 9],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['auto', 'sqrt']
}

# Create a GridSearchCV object
grid_search = GridSearchCV(rf, param_grid, cv=5, scoring='r2')

# Fit the GridSearchCV object to the training data
grid_search.fit(X_train, y_train)

# Print the best hyperparameters
print("Best hyperparameters: ", grid_search.best_params_)
```

```
Best hyperparameters:  {'max_depth': 7, 'max_features': 'auto', 'min_samples_leaf':
2, 'min_samples_split': 10}
```

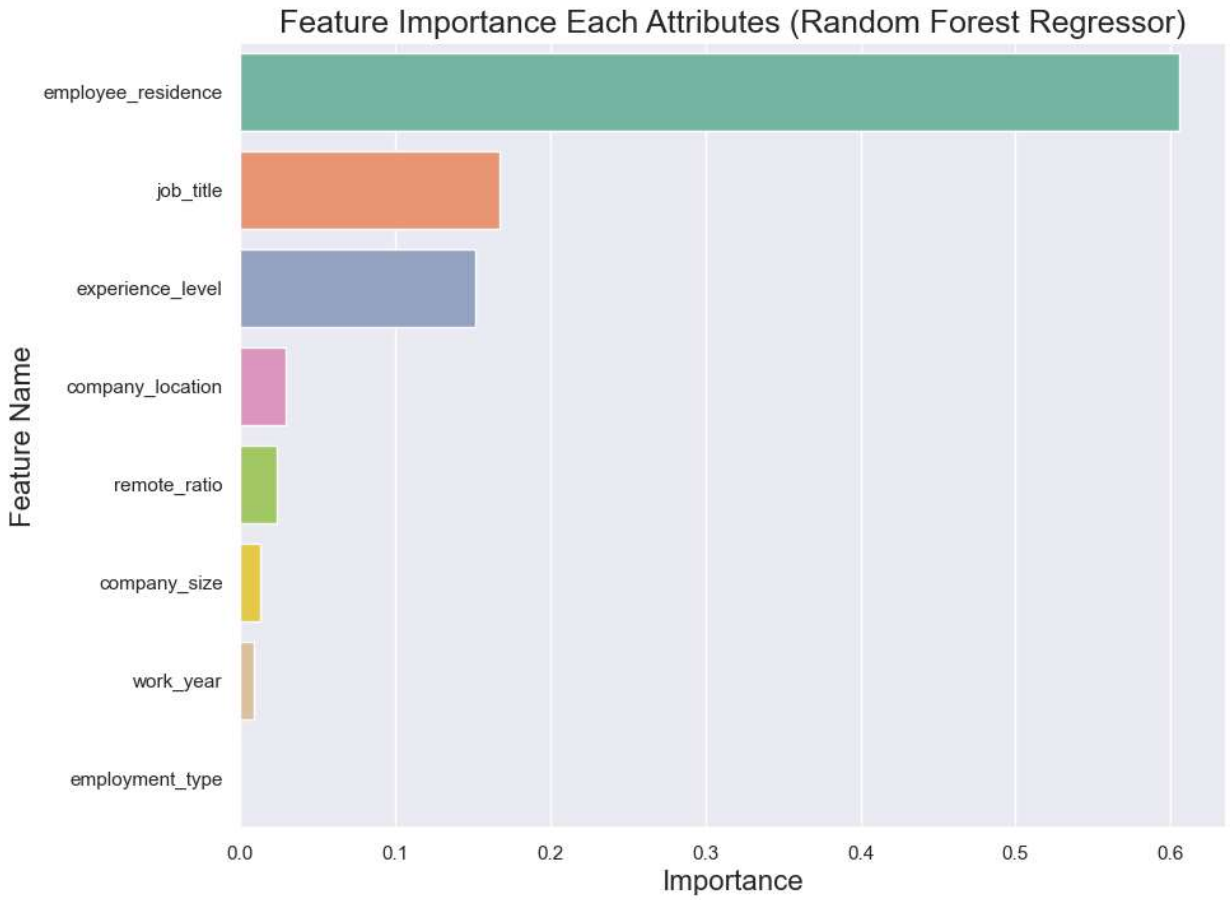
In [35]:
```
from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor(random_state=0, max_depth=7, min_samples_split=10, min_sam
                           max_features='auto')
rf.fit(X_train, y_train)
```

Out[35]:
```
RandomForestRegressor(max_depth=7, min_samples_leaf=2, min_samples_split=10,
                      random_state=0)
```

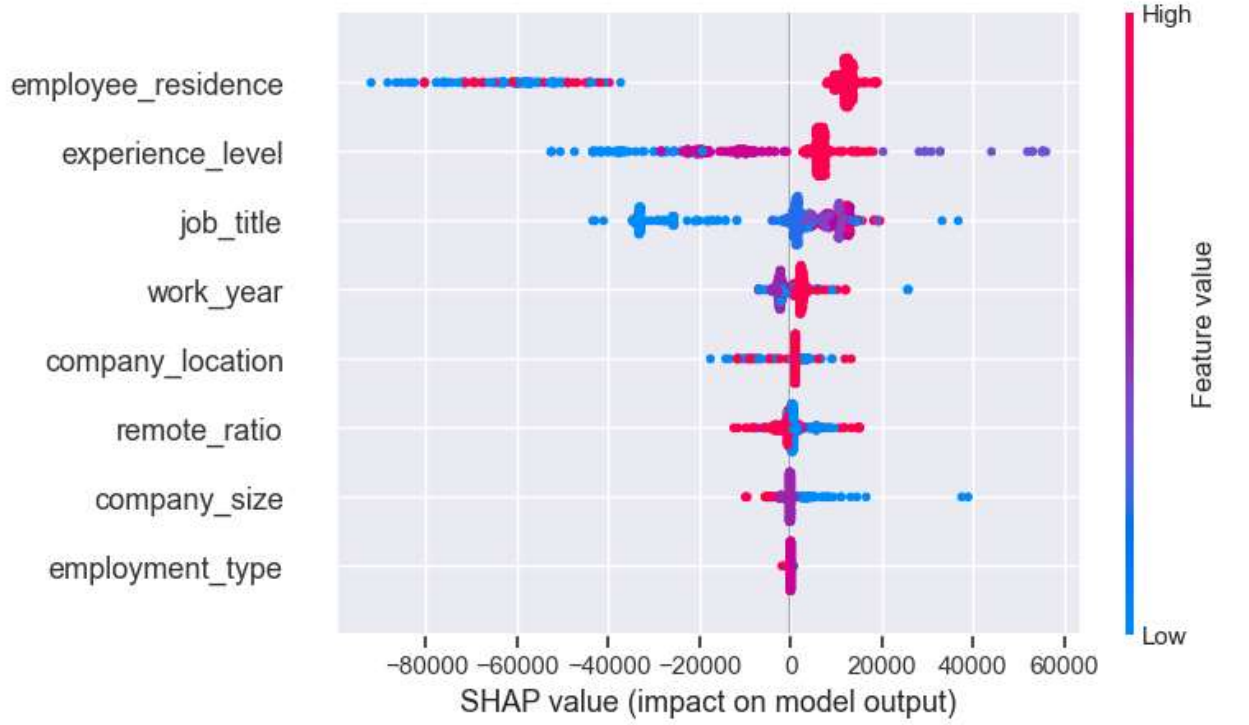
In [36]:
```
from sklearn import metrics
from sklearn.metrics import mean_absolute_percentage_error
import math
y_pred = rf.predict(X_test)
mae = metrics.mean_absolute_error(y_test, y_pred)
mape = mean_absolute_percentage_error(y_test, y_pred)
mse = metrics.mean_squared_error(y_test, y_pred)
r2 = metrics.r2_score(y_test, y_pred)
rmse = math.sqrt(mse)

print('MAE is {}'.format(mae))
print('MAPE is {}'.format(mape))
print('MSE is {}'.format(mse))
print('R2 score is {}'.format(r2))
print('RMSE score is {}'.format(rmse))
```

```
MAE is 39078.9148717654
MAPE is 0.37982999896431796
MSE is 2781586526.3528595
R2 score is 0.348047678599401
RMSE score is 52740.748253630794
```

In [37]:
```
imp_df = pd.DataFrame({
    "Feature Name": X_train.columns,
    "Importance": dtree.feature_importances_
})
fi = imp_df.sort_values(by="Importance", ascending=False)

fi2 = fi.head(10)
plt.figure(figsize=(10,8))
sns.barplot(data=fi2, x='Importance', y='Feature Name')
plt.title('Feature Importance Each Attributes (Random Forest Regressor)', fontsize=18
plt.xlabel ('Importance', fontsize=16)
plt.ylabel ('Feature Name', fontsize=16)
plt.show()
```



In [38]:
```
import shap
explainer = shap.TreeExplainer(rf)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test)
```



In [39]:
```
explainer = shap.Explainer(rf, X_test, check_additivity=False)
shap_values = explainer(X_test, check_additivity=False)
shap.plots.waterfall(shap_values[0])
```