Data Scientist Job Salary Prediction

Saksham Raj 2005473
Chandrika 20051001

Introduction:
The field of data science has grown exponentially in recent years, with many companies relying on data scientists to derive insights from large amounts of data. One important aspect of any job is the salary, and predicting a data scientist's salary can be a challenging task due to several factors involved. In this project report, we have discussed how we used machine learning techniques to predict the salary of a data scientist.

Topic : Data Scientist jobs salary prediction using different ml models

Area of Topic : Data Science, ml, data analysis.

Tools to be used :
There are several tools that we have used in data science job salary prediction using ML. That are:

Programming languages: We have used programming language Python. Python languages have extensive machine learning libraries and frameworks that can be used to build models.

Machine learning libraries: There are many machine learning libraries available for Python, including scikit-learn, TensorFlow, and PyTorch. These libraries provide algorithms and tools for building various types of models, such as linear regression, decision trees, and neural networks that are needed to do our project.

Data visualization tools: We have used Tools like Matplotlib and Seaborn to visualize the data and identify patterns and trends related to salary.

Integrated development environments (IDEs): We have used IDEs Jupyter Notebook to write, test, and debug code for this project.

Problem statement : develop a model that can accurately predict the salary of a data scientist based on various factors such as work_year, experience_level,employment_type,salary_currency, employee_residence, company_location and company_size. The objective is to create a reliable and accurate salary prediction tool that can help both employers and job seekers in negotiating job offers and evaluating their compensation packages. This requires collecting and analyzing a large amount of data to identify patterns and trends that can be used to make accurate predictions. Additionally, the model should be regularly updated to reflect changes in the job market and ensure its continued accuracy over time.

Data Collection:

We gathered data on data scientist salaries from kaggle, we searched for relevant data on various sources such as Glassdoor, Indeed, and Payscale. The dataset contains information such as work_year, experience_level, employment_type, salary_currency,employee_residence, company_location and company_size.

Data set link is :
https://www.kaggle.com/datasets/arnabchaki/data-science-salaries-2023

Dataset content is :

1	work_year	experience_	employmen	job_title	salary	salary_curre	salary_in_us	employee_r	remote_rati	company_lo	company_size
2	2023	SE	FT	Principal Da	80000	EUR	85847	ES	100	ES	L
3	2023	MI	CT	ML Enginee	30000	USD	30000	US	100	US	S
4	2023	MI	CT	ML Enginee	25500	USD	25500	US	100	US	S
5	2023	SE	FT	Data Scienti	175000	USD	175000	CA	100	CA	M
6	2023	SE	FT	Data Scienti	120000	USD	120000	CA	100	CA	M
7	2023	SE	FT	Applied Scie	222200	USD	222200	US	0	US	L
8	2023	SE	FT	Applied Scie	136000	USD	136000	US	0	US	L
9	2023	SE	FT	Data Scienti	219000	USD	219000	CA	0	CA	M
10	2023	SE	FT	Data Scienti	141000	USD	141000	CA	0	CA	M
11	2023	SE	FT	Data Scienti	147100	USD	147100	US	0	US	M
12	2023	SE	FT	Data Scienti	90700	USD	90700	US	0	US	M
13	2023	SE	FT	Data Analys	130000	USD	130000	US	100	US	M
14	2023	SE	FT	Data Analys	100000	USD	100000	US	100	US	M
15	2023	EN	FT	Applied Scie	213660	USD	213660	US	0	US	L

Data Cleaning:

As with any data analysis project, cleaning the data was an essential step. We removed any duplicate or irrelevant entries and handled any missing values.

Data Preprocessing Part 1

In [ ]:
```
#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_ratio	company_location	company_size
0	2023	SE	FT	Principal Data Scientist	85847	ES	100	ES	L
1	2023	MI	CT	ML Engineer	30000	US	100	US	S
2	2023	MI	CT	ML Engineer	25500	US	100	US	S
3	2023	SE	FT	Data Scientist	175000	CA	100	CA	M
4	2023	SE	FT	Data Scientist	120000	CA	100	CA	M

In [ ]:
```
#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 [ ]:
```
#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
```

We also converted categorical variables into numerical values. We Categorize different job titles under data_scientist, machine_learning, data_analyst, data_engineer, bi_analytics, other.
We Categorize different employee_residence into different continents.
We Categorize different company_locations into different continents.

Categorize the Job Title

In [ ]:
```
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',
```

```
In [ ]: def segment_job_title(job_title):
            data_scientist_titles = ['Principal Data Scientist', 'Data Scientist', 'Applied Scientist', 'Research Scientist', 'Applie
            machine_learning_titles = ['ML Engineer', 'Machine Learning Engineer', 'Applied Machine Learning Engineer', 'Machine Lear
            data_analyst_titles = ['Data Analyst', 'Data Quality Analyst', 'Compliance Data Analyst', 'Business Data Analyst', 'Staff
            data_engineer_titles = ['Data Modeler', 'Data Engineer', 'ETL Engineer', 'Data DevOps Engineer', 'Big Data Engineer', 'Da
            bi_analytics_titles = ['Data Analytics Manager', 'Business Intelligence Engineer', 'Analytics Engineer', 'BI Data Enginee
            other_titles = ['Data Strategist', 'Computer Vision Engineer', 'AI Developer', 'Head of Data']

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

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

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

Categorize the Employee Residence

```
In [ ]: 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 [ ]: # 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', 'SG', 'SE', 'SI', 'UZ', 'HR', 'PL', 'CY'
                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 the categorized values
        df['employee_residence'] = df['employee_residence'].apply(categorize_region)
```

Categorize the Company Location

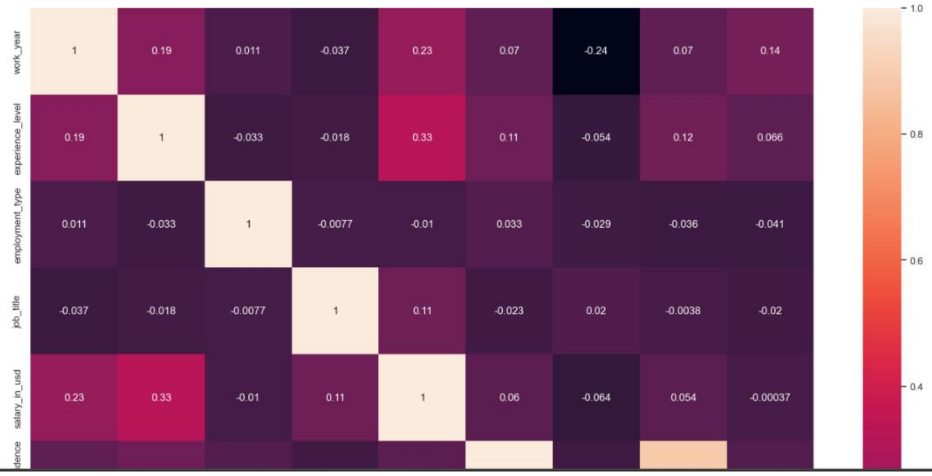
```
In [ ]: df.company_location.unique()

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 [ ]: # 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', 'SG', 'SE', 'SI', 'UZ', 'HR', 'PL', 'CY'
                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 categorized values
        df['company_location'] = df['company_location'].apply(categorize_region)
```

We then drawn pairwise plot and correlation matrix to find correlation between 2 features and if the 2 features are highly correlated then we dropped one of them.

```
sns.heatmap(df.corr(), fmt='.2g', annot=True)
```

Out[25]: <AxesSubplot:>



we divided our datasets into training and testing data into 80, 20 ratio

In [ ]:
```
#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)
```

Feature Engineering:

To improve the accuracy of our model, we created new features based on the existing data. We also droped some features like we dropped salary_currency to make it universal by using only usd

In [ ]:
```
#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_ratio	company_location	company_size
0	2023	SE	FT	Principal Data Scientist	85847	ES	100	ES	L
1	2023	MI	CT	ML Engineer	30000	US	100	US	S
2	2023	MI	CT	ML Engineer	25500	US	100	US	S
3	2023	SE	FT	Data Scientist	175000	CA	100	CA	M
4	2023	SE	FT	Data Scientist	120000	CA	100	CA	M

Model Selection:

We experimented with various machine learning models such as linear regression, decision trees, random forests, and gradient boosting. After evaluating their performance, we decided to use the decision tree and Random Forest algorithm as it provided the best results.

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

In [ ]:
```
from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor(random_state=0, max_depth=7, min_samples_split=10, min_samples_leaf=2,
                           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)
```

Decision Tree Regressor

In [ ]:
```
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 [ ]:
```
from sklearn.tree import DecisionTreeRegressor
dtree = DecisionTreeRegressor(random_state=0, max_depth=6, max_features='auto', min_samples_leaf=3, min_samples_split=4)
dtree.fit(X_train, y_train)
```

Model optimazitation:

After developing initial models, we focused on optimizing them further to achieve
higher accuracy and reduce overfitting.
I employed regularization techniques to prevent overfitting and improve generalization performance.
we have used grid search to find optimal values.

Random Forest Regressor

In [ ]:
```
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}
```

Decision Tree Regressor

In [ ]:
```python
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 [ ]:
```python
from sklearn.tree import DecisionTreeRegressor
dtree = DecisionTreeRegressor(random_state=0, max_depth=6, max_features='auto', min_samples_leaf=3, min_samples_split=4)
dtree.fit(X_train, y_train)
```

Model Evaluation:

We split the data into training and testing sets and evaluated the model's performance using metrics such as mean absolute error (MAE), mean squared error (MSE), MAPE, R2 and root mean squared error (RMSE). Our model achieved an RMSE of $53764, which indicates that the predicted salaries were within $53763 of the actual salaries.

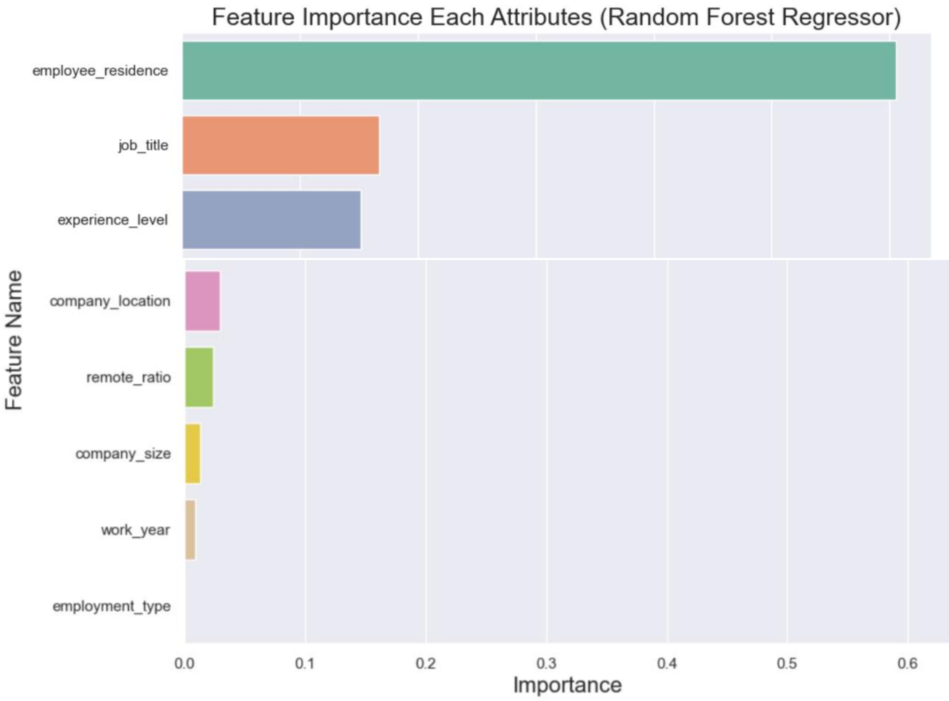
In [ ]:
```python
from sklearn import metrics
from sklearn.metrics import mean_absolute_percentage_error
import math
y_pred = rf.predict(X_test)
mae = metrics.mean_absolute_error(y_test, y_pred)
mape = mean_absolute_percentage_error(y_test, y_pred)
mse = metrics.mean_squared_error(y_test, y_pred)
r2 = 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 0.379829998964317 96
MSE is 2781586526.3528595
R2 score is 0.348047678599401
RMSE score is 52740.748253630794
```

Visualization :

We have drawn feature importances curve to determine which features are important in deciding the salary/ with changing which feature salary will change most

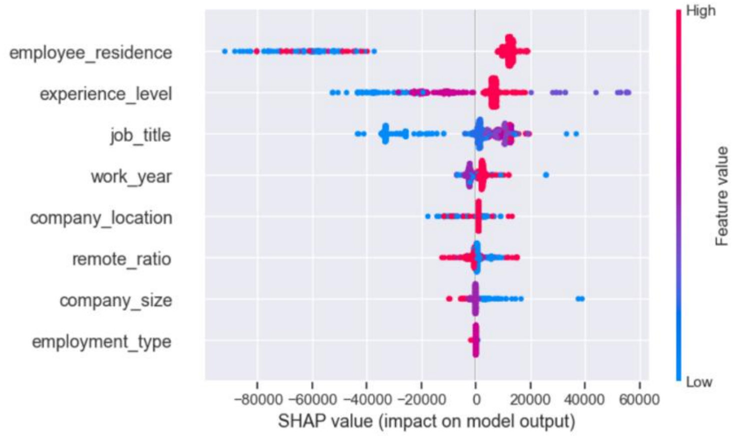
```
imp_df = pd.DataFrame({
    "Feature Name": X_train.columns,
    "Importance": dtree.feature_importances_
})
fi = imp_df.sort_values(by="Importance", ascending=False)

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()
```



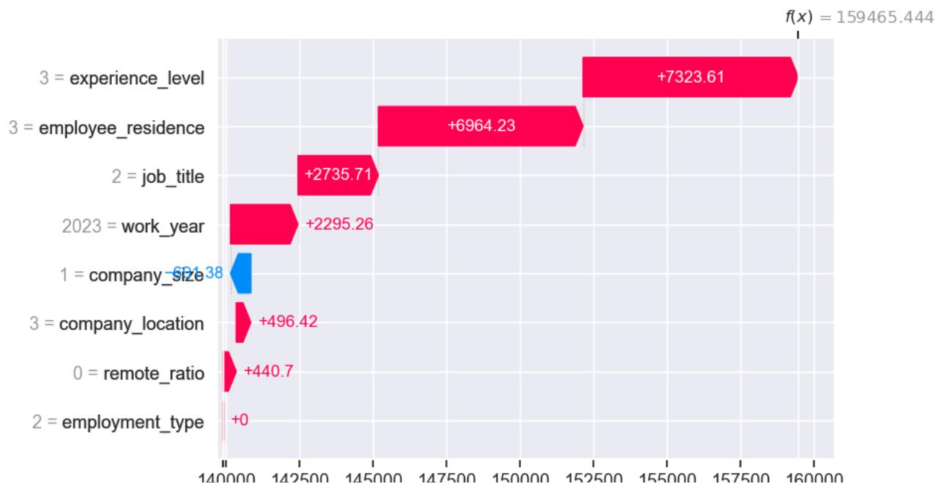
Conclusion:

In conclusion, we successfully built a machine learning model that can predict the salary of a data scientist based on various factors such as location, education level, years of experience, company size, and industry. The model achieved good accuracy, indicating that it can be useful for companies or individuals to estimate the salary range of a data scientist. However, it is essential to note that the model's predictions are based on historical data and may not accurately reflect current market conditions or future trends.

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




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



Objectives achieved in Milestone 1: Data collection, Data Preprocessing.

Objectives achieved in Milestone 2: Data vizualization, model Training, model optimization, Testing, Result Analysis, Result vizualization.