Predicting Data Science Salary in the United States

April Chia, Allison McKernan, and Jorge Roldan

Shiley-Marcos School of Engineering, University of San Diego

Abstract

This project investigates the factors that influence data science job positions and salaries in the United States, aiming to develop predictive models that identify these factors' significance. The main purpose of this analysis is to find which job positions in data science have the highest salaries using data mining methods. The primary question is: *What are the key determinants of data science job roles and salaries, and how accurately can they be predicted using machine learning models?* The demand for data science professionals has grown exponentially, making it one of the most sought-after career paths globally. Understanding what influences job roles and salaries helps individuals, employers, and policymakers align expectations, training, and hiring strategies. Using programming techniques, relationships between salaries and company size, and salaries and work types, can be explored. By analyzing a publicly available dataset, the analysis will shed light on how company size, work models, and experience levels impact salaries, providing actionable insights for stakeholders.

Keywords: data science, salary, machine learning, predictive modeling

Predicting Data Science Job Positions in the United States

The growing demand for data science professionals in the United States highlights the significance of understanding factors that influence salaries and job positions in this field. The aim of this study is to develop predictive models that identify and evaluate the impact of key attributes—such as experience level, work models, and company size—on salaries. Visualizing the dataset using data mining and machine learning techniques, will provide insights about salary trends across different data science positions in the United States. This analysis also seeks to achieve high predictive accuracy. This section outlines the steps taken to prepare, clean, and analyze the data, ensuring a robust foundation for building effective models.

Methodology

The raw public dataset used to perform the data mining tasks was sourced from Kaggle.com which contains public datasets and notebooks. The dataset is a single .csv file that includes 11 attributes and 6,599 records related to data science salary information from the year 2020 to 2024. The dataset includes categorical and numeric data types: *job_title, work_year, experience_level, employment_type, salary, salary_currency, salary_in_usd, exployee_residence, remote_ratio, company_location, company_size.* Python was used throughout the data mining process. Beginning with data preparation and cleaning, it was determined that there were no missing values or duplicate rows. The rows that do not have the United States as the company location were removed from the dataset. The employee residence column and salary currency column were also removed due to irrelevance to the project objective. Rows containing outliers were determined and removed. The method used was, if a salary is above and below 1.5 times the IQR, it was determined as an outlier. Finally, we encoded the categorical variables for analysis and transformed the *salary_in_usd* column to *salary_binary* to show records with a salary above the median of $147,000 as “High” and a salary below the median as “Low”. The cleaned data contained 5,204 records and nine columns, with 50.1% labeled “High” salary and 49.9% labeled “Low” salary, therefore, rebalancing was not needed to build our models.

Exploratory data analysis (EDA) was conducted to uncover patterns in the dataset. An initial finding from a distribution analysis showed that the salary data was skewed to the right which requires logarithmic transformation for modeling. Normalized bar charts with overlay were used to accurately analyze the trends. Figure 1 & 2, show that working on-site or remote were more likely to have higher salaries. Figure 2 compares the median salary across the three different work types: Hybrid, On-site, and Remote. In-office jobs have the highest median salary, slightly above 140,000. Hybrid work models reflect lower pay trends.

Figure 1

Normalized Bar Chart of Work Models with Salary Overlay

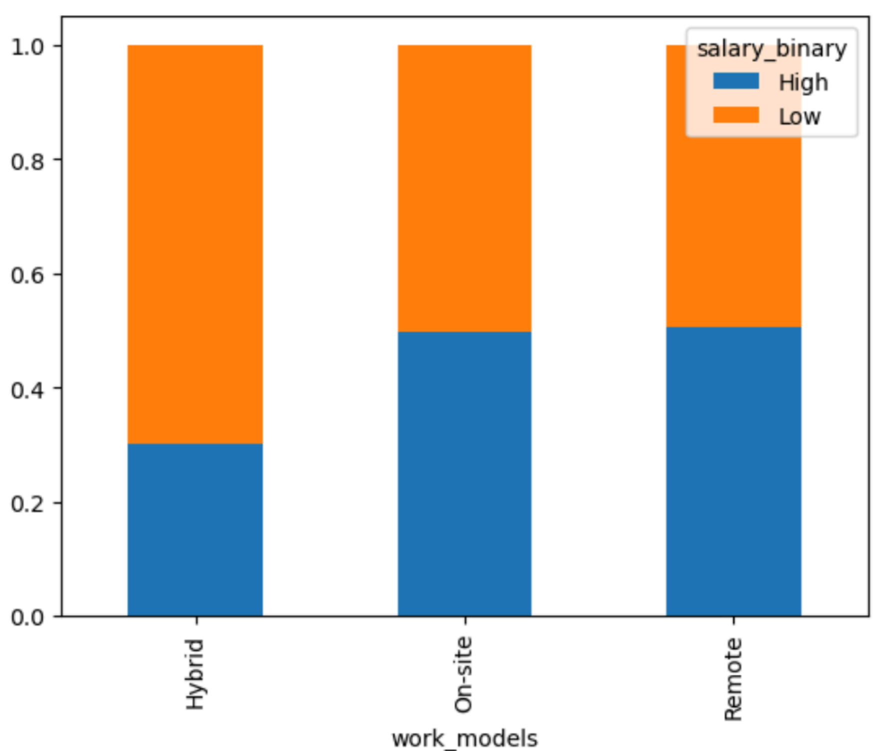
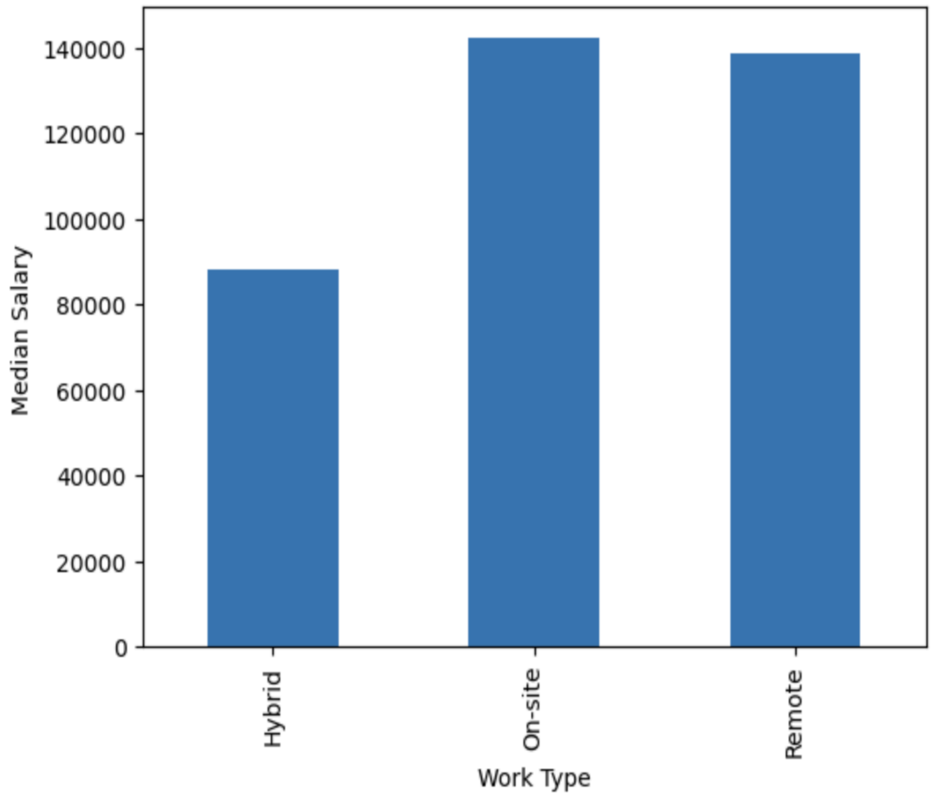


Figure 2

Median Salary By Work Type



Shown in Figure 3, working at an executive level showed to be most likely to have a high salary, followed by senior level, mid level, then entry level. Regarding company size, Figure 4 shows that large companies were more likely to offer higher salaries compared to smaller companies. Conducting a correlation analysis showed that salary correlated positively with experience level and company size.

Figure 3

Normalized Bar Chart of Experience Level with Salary Overlay

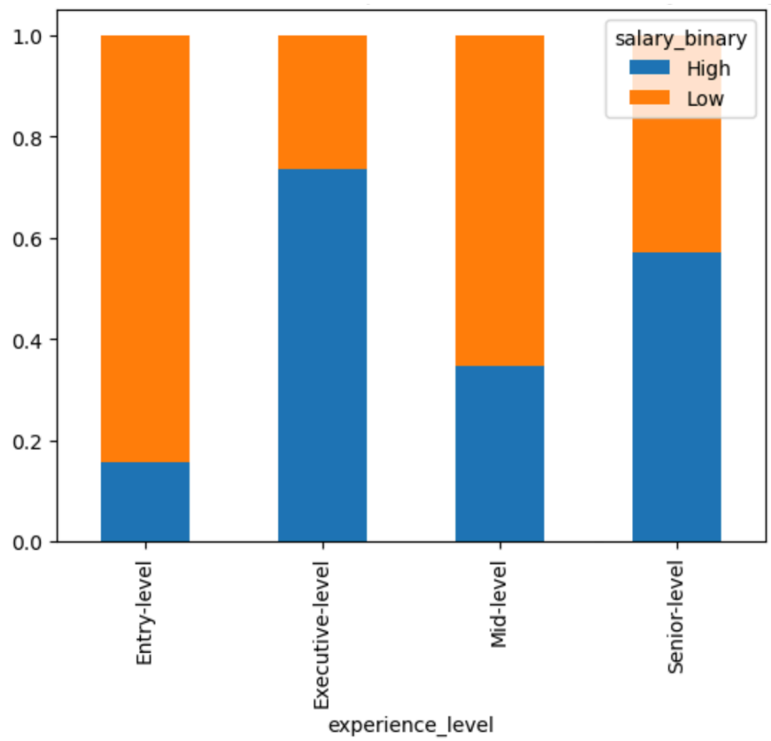
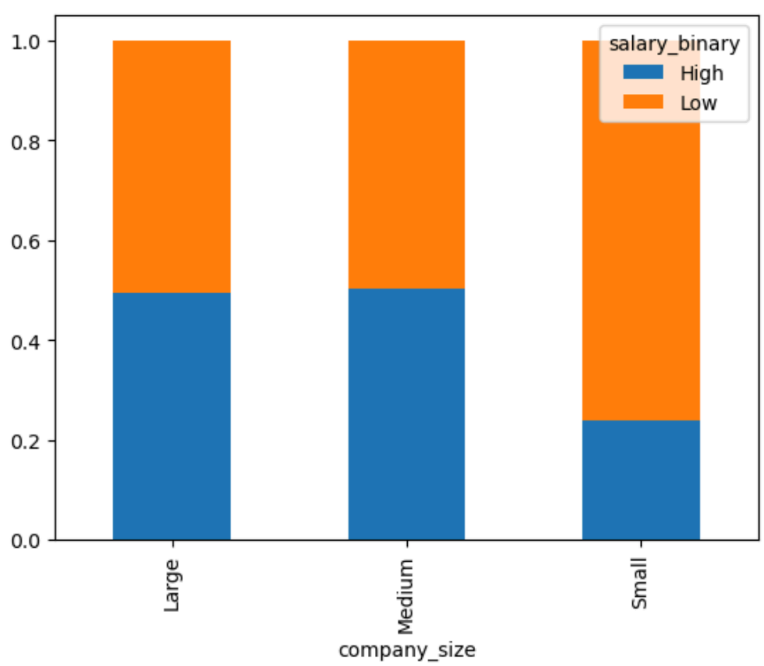


Figure 4

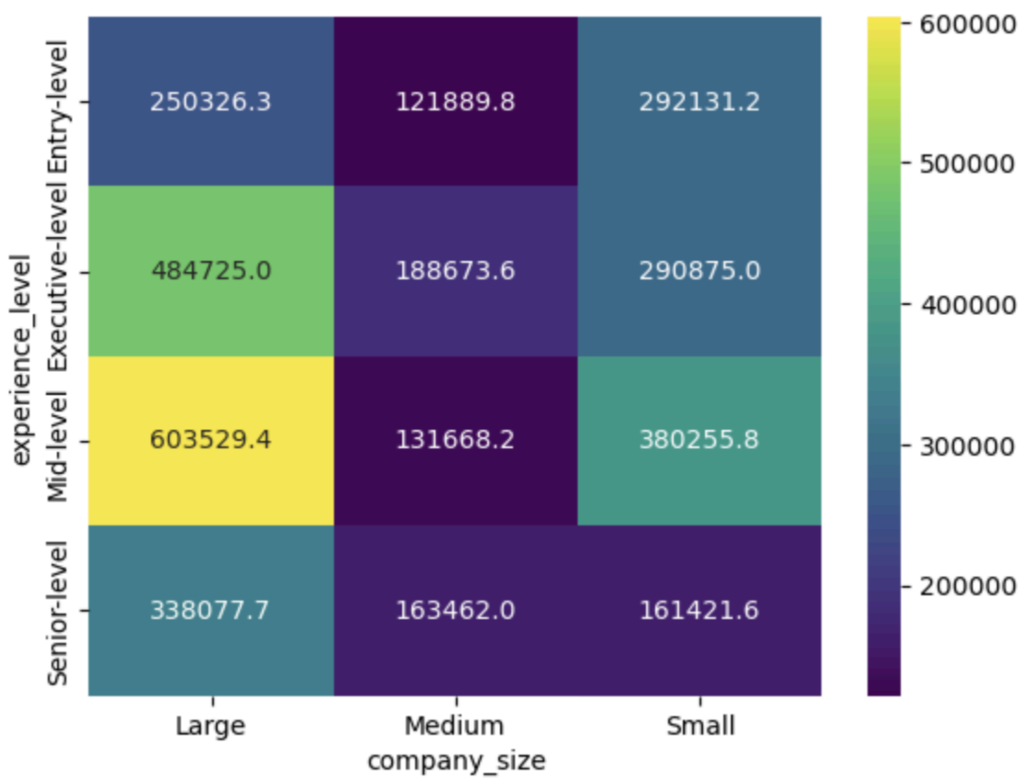
Normalized Bar Chart of Company Size with Salary Overlay



Additionally, the heatmap in Figure 5, visualizes the mean salary across different combinations of experience levels and company sizes. The cells are color-coded, with higher salaries represented by lighter colors (closer to yellow) and lower salaries by darker colors (closer to purple). The cell values represent the mean salary for each combination of company size and experience level. As expected, salaries increase with experience level across all company sizes. Along with the bar charts, the heatmap also illustrates that large companies are the most lucrative, generally paying the highest salaries at all experience levels, particularly at the Executive level. Smaller companies also have competitive salaries, in addition to their amounts of opportunities for growth.

Figure 5

Mean Salary By Experience Level and Company Size



Random Forest

Random forest is an ensemble classification method that builds decorrelated decision trees to improve the generalization performance (Tan et al., 2018). Random subsets of the data science salaries training dataset are chosen with replacement. Then, random subsets of the attributes are considered at each split and each record of the dataset is given a classification of "High" or "Low" salary to ensure diversity. The number of trees to be built was set to 100 trees.

Logistic Regression

The target attribute is binary which logistic regression modelling is well-suited for. The selected predictors are used to predict the salary response using the parametric logistic regression equation:

$$p(y) = \frac{exp\left(b_0 + b_1 x_1 + b_2 x_2 + ... + b_p x_p\right)}{1 + exp\left(b_0 + b_1 x_1 + b_2 x_2 + ... + b_p x_p\right)} + \varepsilon \tag{1}$$

Equation 2 shows the descriptive form of the selected predictors used to predict salary.

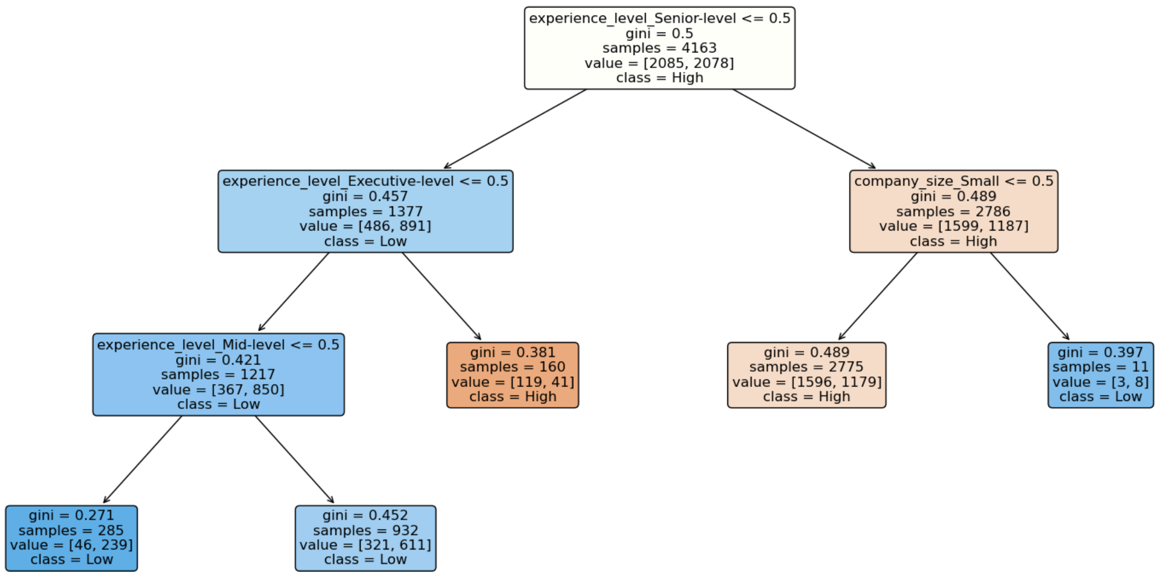
$$\hat{p}(salary) = \frac{exp\left(b_0 + b_1(company\ size) + b_2(experience\ level) + b_3(work\ models)\right)}{1 + exp\left(b_0 + b_1(company\ size) + b_2(experience\ level) + b_3(work\ models)\right)} \tag{2}$$

CART

The CART decision tree, shown in Figure 6, consists of one root node, three decision nodes, and five leaf nodes. Going down the tree, starting at the root node with experience level attribute, splitting the data based on whether the senior-level experience is present or not. This resulted in a nearly equal class distribution. Looking at the left subtree, it represents the cases with lower experience, further splitting the data based on executive-level and mid-level experience. This leads to leaf nodes with predictions dominated by the lower experience. Looking at the right subtree, it represents the cases with higher experience and larger company sizes, further splitting the data based on small company size. This leads to leaf nodes with predictions of either high or low salary based on the majority distribution. The lower the gini value is at each leaf node indicates a higher confidence level in the predictions.

Figure 6

CART Model

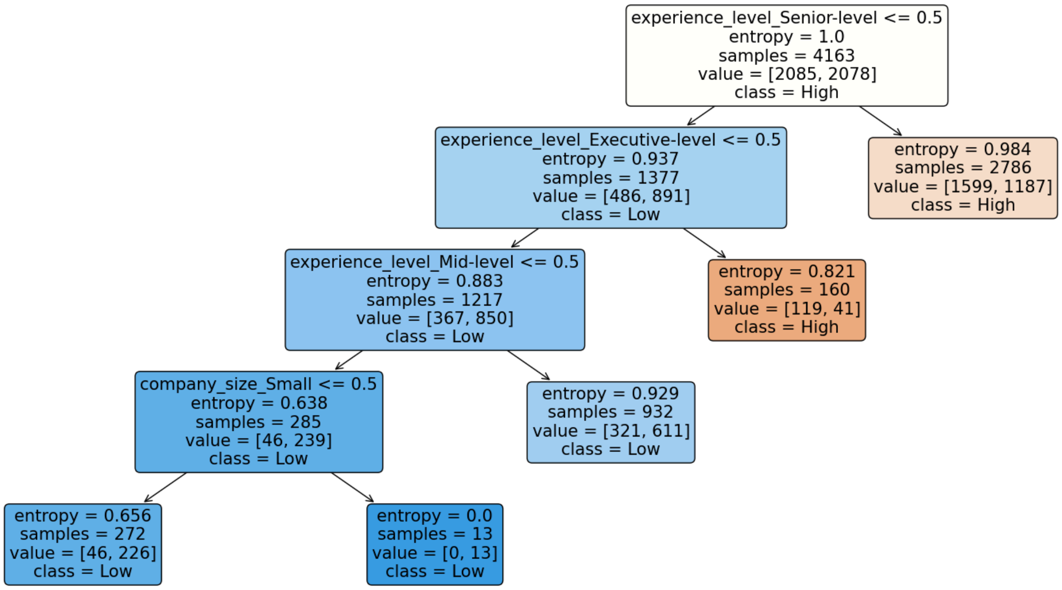


C5.0

As seen in Figure 7, the C5.0 decision tree consists of one root node, three decision nodes, and five leaf nodes. Starting at the root node, the data is split based on whether or not the senior-level experience is present or not. The entropy value at this node is 1.0 which indicates a maximum disorder and a nearly equal class distribution. The left subtree represents the cases with lower experience and splitting the cases by executive-level and mid-level experience. The right subtree shows the cases with senior-level experience and splitting on company size which shows a higher entropy of 0.984 compared to the left of 0.937. Looking at the leaf nodes, it is seen that the entropy values are decreasing which indicates higher purity and confidence in the predictions.

Figure 7

C5.0 Model



Naïve Bayes

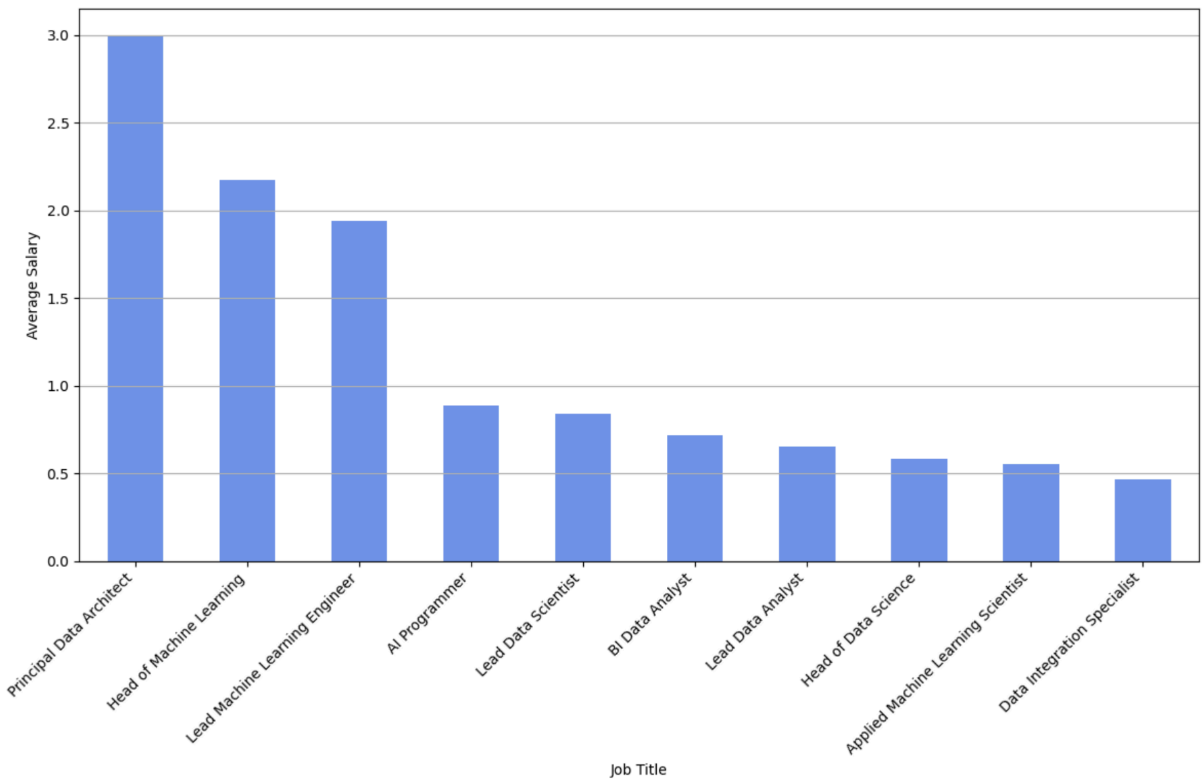
Naive Bayes classification was used to utilize probability to evaluate the relationship between the key attributes. This can be conducted using Bayes Theorem:

$$p\left(Y = y|X^*\right) = \frac{p\left(X^*|Y=y\right)p\left(Y=y^*\right)}{p\left(X^*\right)} \tag{3}$$

In this model, the attributes *experience_level*, *company_size*, and *work_models* were selected to be evaluated. Figure 8 visualizes the top 10 highest-paying jobs in data science based on their average salaries. Principal Data Architect is the highest-paying job, followed by Head of Machine Learning with an average salary exceeding 2 million, and Lead Machine Learning Engineer ranks third. The remaining roles, including AI Programmer, Lead Data Scientist, and others, still have high salaries ranging between 0.5 million and 1.5 million US dollars. The highest-paying roles are associated with leadership and strategic decision-making. Jobs related to machine learning and AI consistently appear, highlighting the high demand and compensation in these fields.

Figure 8

Top 10 Highest Paying Jobs In Data Science



Results

The study revealed that various machine learning models can be used to predict data science salaries in the United States with accuracies ranging from around 60-61%. Interpretable models, such as CART and C5.0, provided valuable decision rules for stakeholders. Despite these successes, some models struggled with nuanced patterns, particularly in scenarios with highly imbalanced data. The CART model and C5.0 model provided interpretable decision rules and an accuracy of 60.61%. The logistic regression model also achieved an accuracy of 60.61% which outperformed the random forest and naïve bayes models. The model that achieved the highest precision is logistic regression and CART at 56.89%. These two models performed best for predicting the target attribute.

Table 1

Evaluation Metrics

	Baseline	*Random Forest*	*Logistic Regression*	*CART*	*C5.0*	*Naïve Bayes*
Accuracy	0.4831892	0.6003842	0.6061479	0.6061479	0.6061479	0.6042267
Error Rate	0.5168108	0.3996158	0.3938521	0.3938521	0.3938521	0.3957733
Sensitivity	0.4882813	0.8222656	0.8222656	0.8222656	0.8261719	0.8164063
Specificity	0.4782609	0.3856333	0.3969754	0.3969754	0.3931947	0.3988658
Precision	0.4752852	0.5643432	0.5689189	0.5689189	0.5685484	0.5679348
F1	0.4816956	0.6693164	0.6725240	0.6725240	0.6735669	0.6698718
F2	0.4856255	0.7534001	0.7550215	0.7550215	0.7575215	0.7507184
F0.5	0.4778287	0.6021167	0.6062788	0.6062788	0.6063647	0.6047454

Conclusion

The study successfully analyzed the factors influencing data science salaries in the United States using data mining techniques and predictive modeling. The analysis highlights several actionable insights: Employers can attract top talent by offering remote work options and competitive salaries tied to experience, while job seekers should focus on gaining advanced skills and targeting larger companies for higher salaries. Additionally, the strong correlation between experience level, company size, and salary emphasizes the importance of structured career progression and strategic company policies in attracting and retaining skilled professionals.

This research has limitations, such as the dataset's temporal scope (2020–2024) and geographical focus on the United States, which may restrict generalizability. Future studies could benefit from incorporating additional features, such as educational background, certifications, and industry type, to enhance predictive accuracy. Furthermore, exploring advanced models like neural networks or integrating international datasets could provide a more comprehensive understanding of global trends in data science salaries.

By addressing these limitations and expanding the analysis, this research can serve as a foundation for future work, helping stakeholders make data-driven decisions in the dynamic field of data science. These insights can guide efforts in career planning in the fast changing and growing field of data science.

References

Islam, S. (2024). *Data Science Salaries 2024*, Version 1. Retrieved November 17, 2024 from https://www.kaggle.com/datasets/sazidthe1/data-science-salaries

Larose, C. D., & Larose, D. T. (2019). *Data Science Using Python and R*. Wiley Global Research (STMS). https://usd.vitalsource.com/books/9781119526841

Tan, P., Steinbach, M., & Kumar, V. (2018). *Introduction to Data Mining* (2nd ed.). Pearson Education (US). https://usd.vitalsource.com/books/9780134080284

Appendix

final_project

December 9, 2024

[34]:
```python
import pandas as pd
import numpy as np
import seaborn as sns
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier, plot_tree
import statsmodels.api as sm
from sklearn.metrics import confusion_matrix, classification_report,␣
    ↪accuracy_score
from sklearn.naive_bayes import MultinomialNB
from sklearn import metrics
from sklearn.dummy import DummyClassifier
```

Data Importing and Pre-processing

[35]:
```python
# Load the dataset
data = pd.read_csv('data_science_salaries.csv')
data.head()
```

[35]:
```
        job_title experience_level employment_type work_models  work_year  \
0   Data Engineer        Mid-level       Full-time      Remote       2024
1   Data Engineer        Mid-level       Full-time      Remote       2024
2  Data Scientist     Senior-level       Full-time      Remote       2024
3  Data Scientist     Senior-level       Full-time      Remote       2024
4    BI Developer        Mid-level       Full-time     On-site       2024

  employee_residence  salary salary_currency  salary_in_usd company_location  \
0      United States  148100             USD         148100    United States
1      United States   98700             USD          98700    United States
2      United States  140032             USD         140032    United States
3      United States  100022             USD         100022    United States
4      United States  120000             USD         120000    United States

  company_size
0       Medium
1       Medium
```

```
2       Medium
3       Medium
4       Medium
```

[36]:
```
# Find missing data
data.isnull().sum()
```

[36]:
```
job_title             0
experience_level      0
employment_type       0
work_models           0
work_year             0
employee_residence    0
salary                0
salary_currency       0
salary_in_usd         0
company_location      0
company_size          0
dtype: int64
```

[37]:
```
# Check for duplicates and remove if any
data.duplicated().sum()
```

[37]: 0

[38]:
```
# Remove irrelevant columns
data = data[data['employee_residence'].str.contains("United States")]
data = data.drop(columns = ['salary_currency', 'employee_residence'])
```

[39]:
```
data.dtypes
```

[39]:
```
job_title           object
experience_level    object
employment_type     object
work_models         object
work_year            int64
salary               int64
salary_in_usd        int64
company_location    object
company_size        object
dtype: object
```

[40]:
```
# Encode categorical variables
categorical_cols = ['job_title', 'experience_level', 'employment_type',
                    'work_models', 'company_location', 'company_size']

data_encoded = pd.get_dummies(data, columns = categorical_cols, drop_first =␣
 ↪False)
```

```python
# Check for outliers in salary_in_usd using the IQR method
q1, q3 = data['salary_in_usd'].quantile([0.25, 0.75])
iqr = q3 - q1
lower_bound = q1 - 1.5 * iqr
upper_bound = q3 + 1.5 * iqr

# Remove rows with outliers in salary_in_usd
data_cleaned = data[(data['salary_in_usd'] >= lower_bound) &
                    (data['salary_in_usd'] <= upper_bound)]
```

[41]:
```python
# Display summary of cleaned data
print(data_cleaned.info())
print(data_cleaned.head())
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 5204 entries, 0 to 6556
Data columns (total 9 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   job_title         5204 non-null   object
 1   experience_level  5204 non-null   object
 2   employment_type   5204 non-null   object
 3   work_models       5204 non-null   object
 4   work_year         5204 non-null   int64
 5   salary            5204 non-null   int64
 6   salary_in_usd     5204 non-null   int64
 7   company_location  5204 non-null   object
 8   company_size      5204 non-null   object
dtypes: int64(3), object(6)
memory usage: 406.6+ KB
None
        job_title experience_level employment_type work_models  work_year  \
0   Data Engineer        Mid-level       Full-time      Remote       2024
1   Data Engineer        Mid-level       Full-time      Remote       2024
2  Data Scientist     Senior-level       Full-time      Remote       2024
3  Data Scientist     Senior-level       Full-time      Remote       2024
4    BI Developer        Mid-level       Full-time     On-site       2024

   salary  salary_in_usd company_location company_size
0  148100         148100    United States       Medium
1   98700          98700    United States       Medium
2  140032         140032    United States       Medium
3  100022         100022    United States       Medium
4  120000         120000    United States       Medium
```

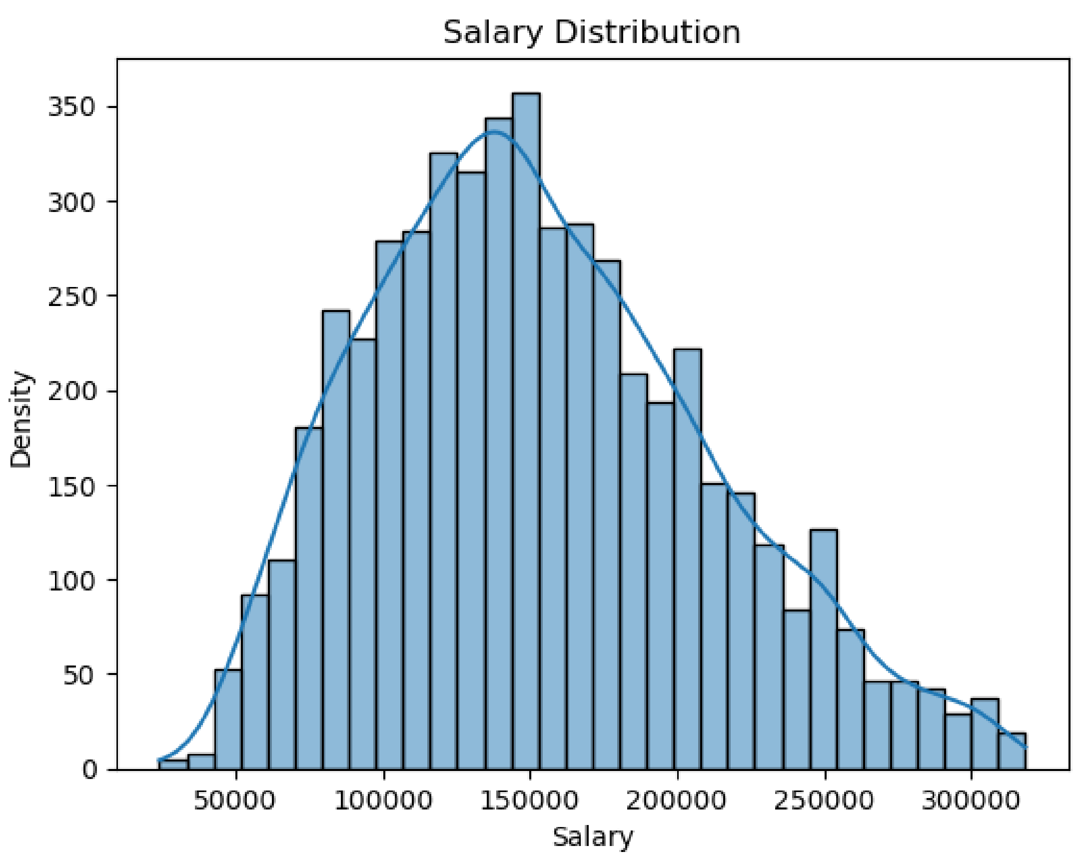
[42]:
```python
data_cleaned.describe()
```

[42]:

	work_year	salary	salary_in_usd
count	5204.000000	5204.000000	5204.000000
mean	2022.888163	153211.435050	153220.955419
std	0.590284	57568.049715	57552.569690
min	2020.000000	24000.000000	24000.000000
25%	2023.000000	110000.000000	110000.000000
50%	2023.000000	147000.000000	147000.000000
75%	2023.000000	190000.000000	190000.000000
max	2024.000000	318300.000000	318300.000000

Data Analysis and Visualization

[43]:
```
sns.histplot(data_cleaned['salary_in_usd'], kde = True)
plt.xlabel('Salary')
plt.ylabel('Density')
plt.title('Salary Distribution')
plt.show()
```

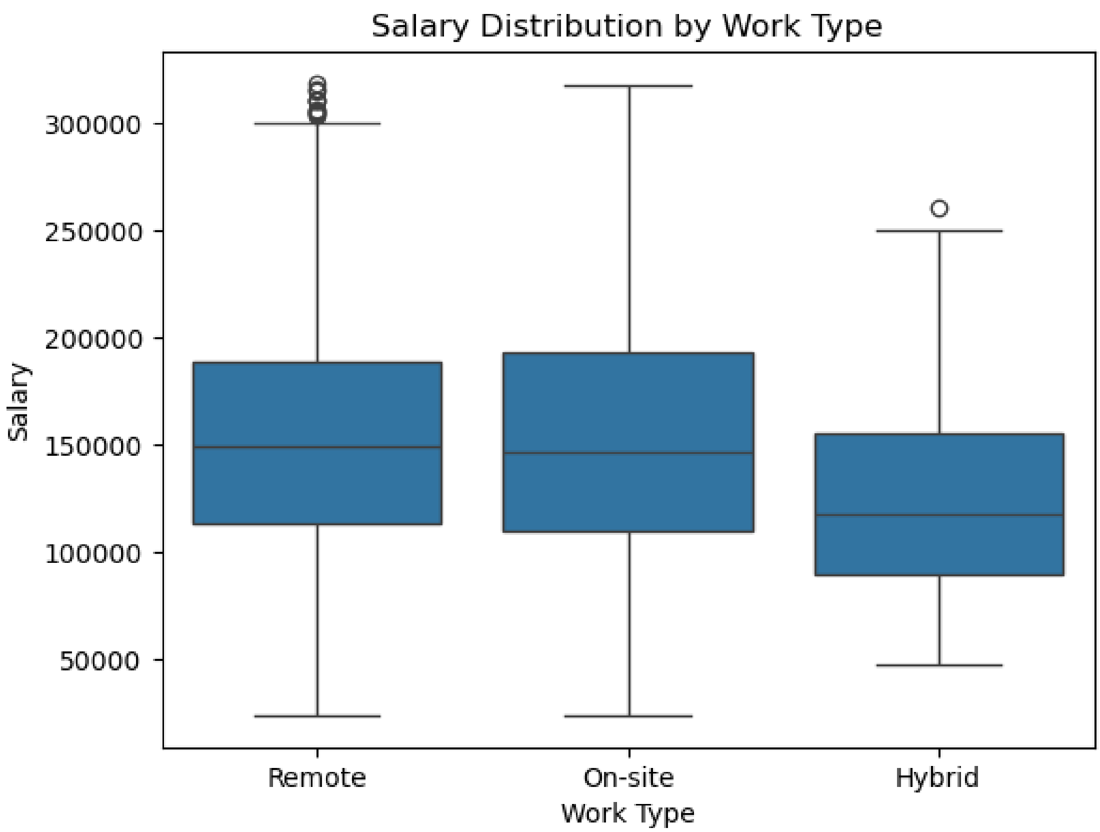


[44]:
```
# Salary distribution across experience level
sns.boxplot(x = 'experience_level', y = 'salary_in_usd', data = data_cleaned)
```

```
plt.xlabel('Experience Level')
plt.ylabel('Salary')
plt.title('Salary Distribution by Experience Level')
plt.show()
```

[45]:

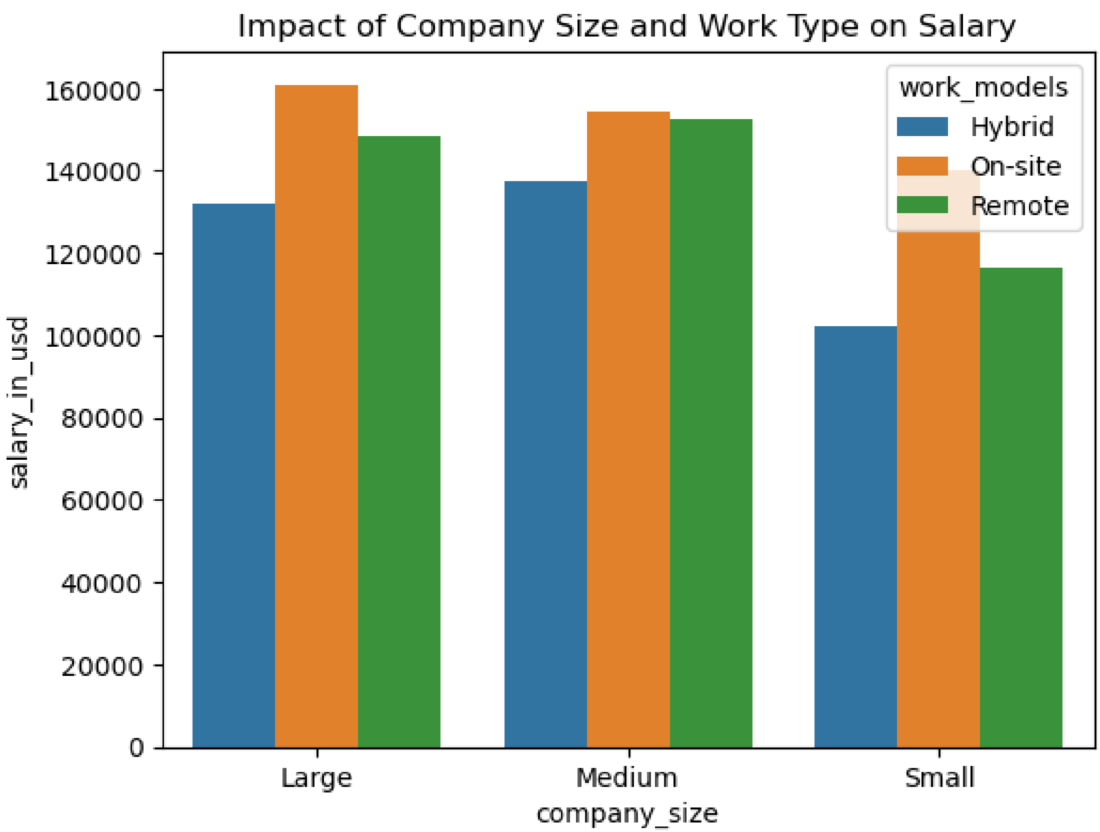
```
# Salary distribution across work type
sns.boxplot(x = 'work_models', y = 'salary_in_usd', data = data_cleaned)
plt.xlabel('Work Type')
plt.ylabel('Salary')
plt.title('Salary Distribution by Work Type')
plt.show()
```



[46]:
```
# Salary distribution across company size
sns.boxplot(x = 'company_size', y = 'salary_in_usd', data = data_cleaned)
plt.xlabel('Company Size')
plt.ylabel('Salary')
plt.title('Salary Distribution by Company Size')
plt.show()
```

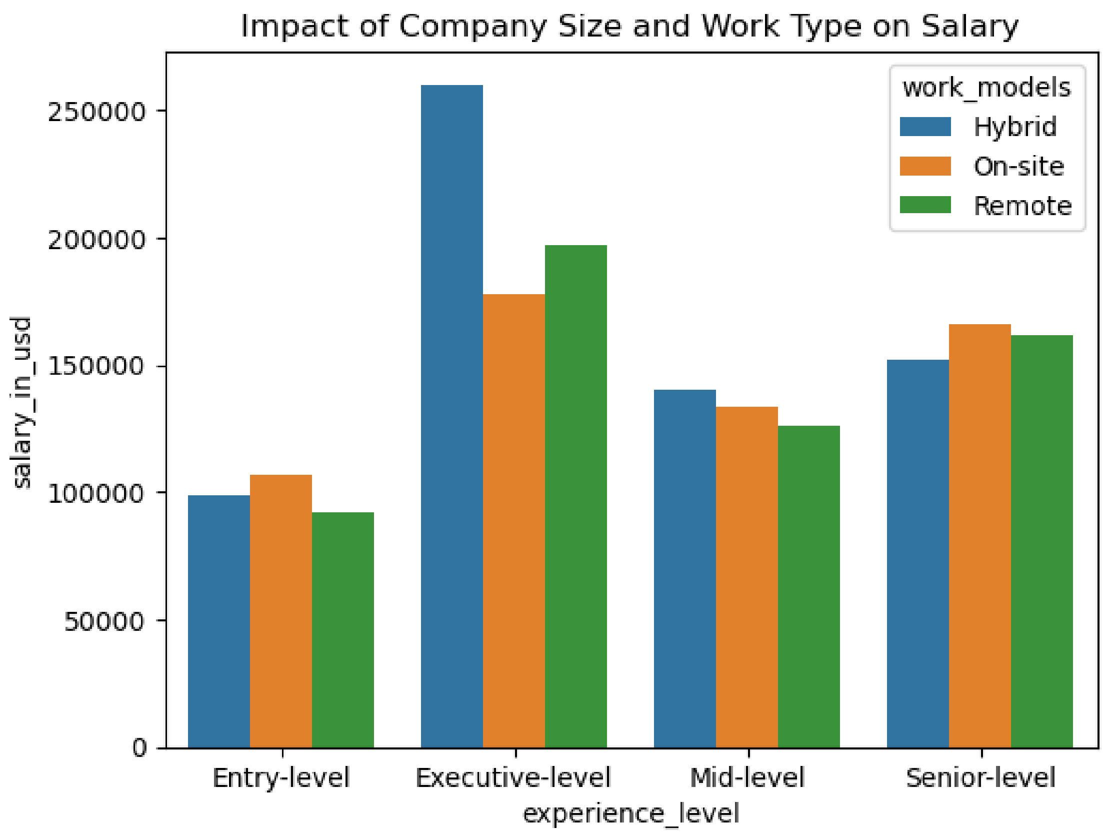
[47]:
```
interaction_effects = data_cleaned.groupby(['work_models',
↪'company_size'])['salary_in_usd'].mean().reset_index()
```

[48]:
```
# Bar plot to visualize company size and work type on salary
sns.barplot(data = interaction_effects, x = 'company_size', y =
↪'salary_in_usd', hue = 'work_models')
plt.title('Impact of Company Size and Work Type on Salary')
plt.show()
```



[49]:
```
interaction_effects2 = data_cleaned.groupby(['work_models',␣
↪'experience_level'])['salary_in_usd'].mean().reset_index()
```

[50]:
```
# Bar plot to visualize experience level and work type on salary
sns.barplot(data = interaction_effects2, x = 'experience_level', y =␣
↪'salary_in_usd', hue = 'work_models')
plt.title('Impact of Company Size and Work Type on Salary')
plt.show()
```



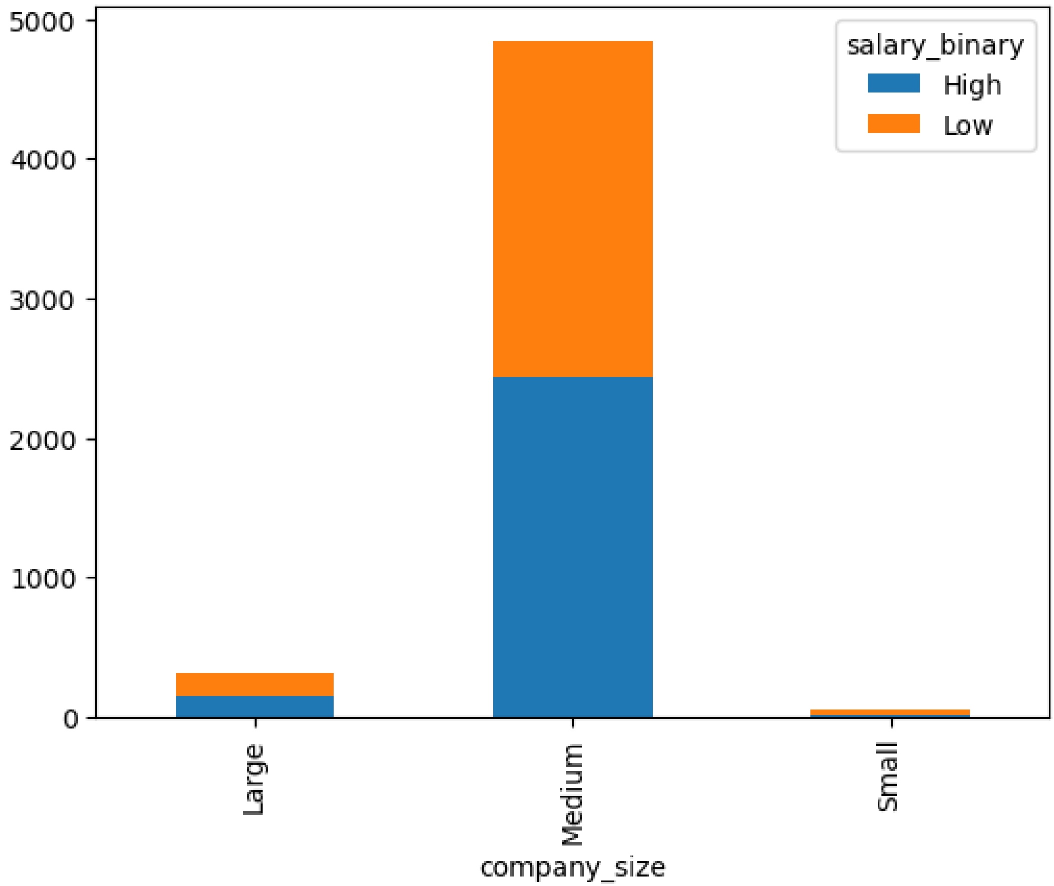
[51]:
```
# Median of salary
salary_median = data_cleaned['salary_in_usd'].median()
```

[52]:
```
# Convert 'salary_in_usd' to binary
data_cleaned['salary_binary'] = data_cleaned['salary_in_usd'].apply(lambda x:␣
↪'High' if x > salary_median else 'Low')
```

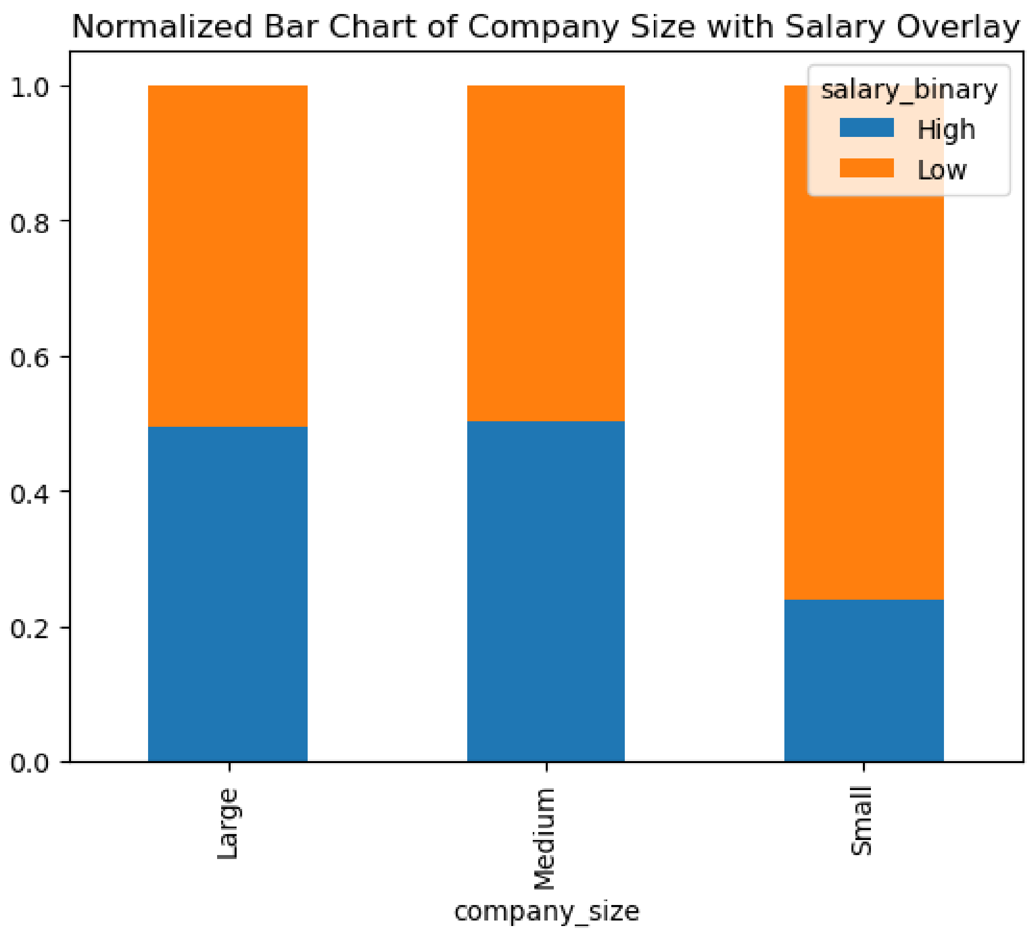
```
/var/folders/6j/qtnqw_nn0bj948l89lgcc38m0000gn/T/ipykernel_17069/884209481.py:2:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  data_cleaned['salary_binary'] = data_cleaned['salary_in_usd'].apply(lambda x:
'High' if x > salary_median else 'Low')
```

[53]:
```
# Bar graph of company_size by salary
company_salary = pd.crosstab(data_cleaned['company_size'],␣
↪data_cleaned['salary_binary'])
company_salary.plot(kind = 'bar', stacked = True)
```

[53]: <Axes: xlabel='company_size'>

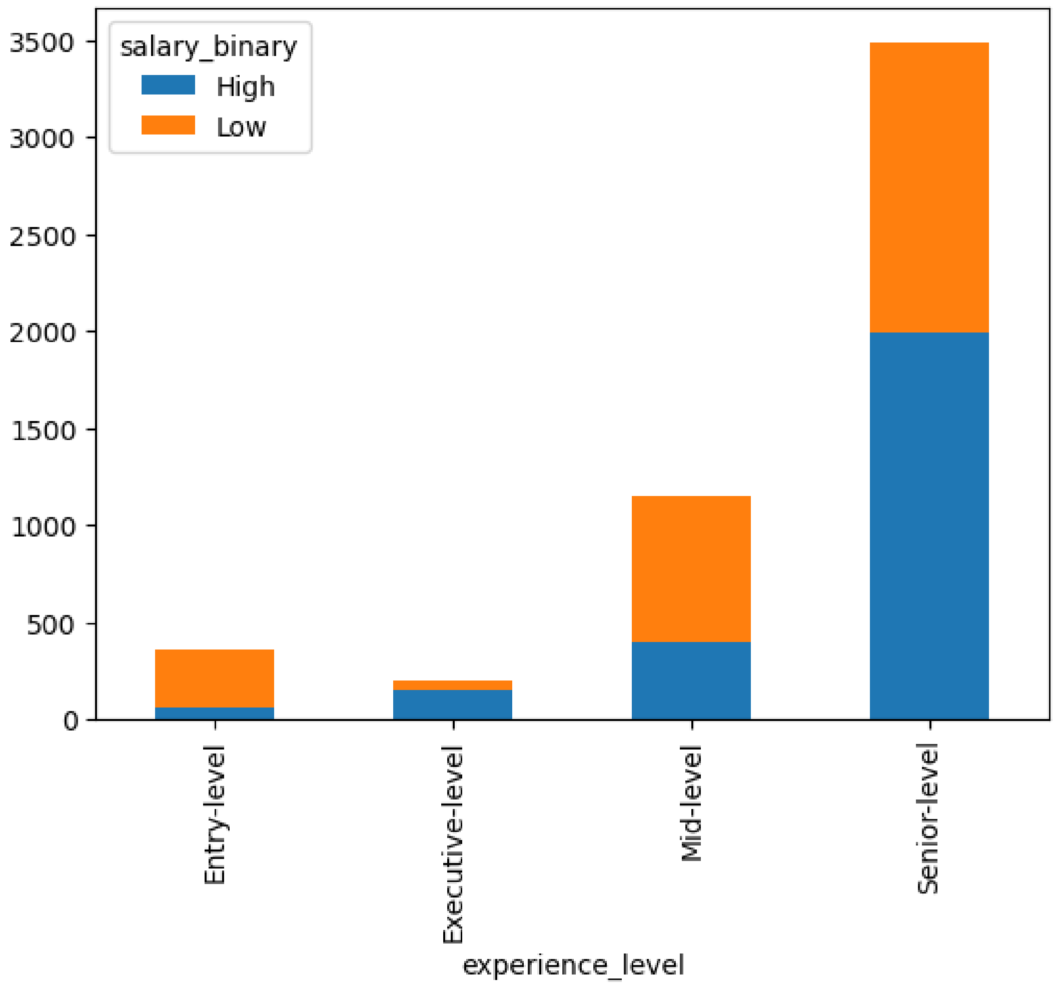


[54]:
```
# Normalized
company_salary_norm = company_salary.div(company_salary.sum(1), axis = 0)
company_salary_norm.plot(kind = 'bar', stacked = True)
plt.title('Normalized Bar Chart of Company Size with Salary Overlay')
plt.show()
```

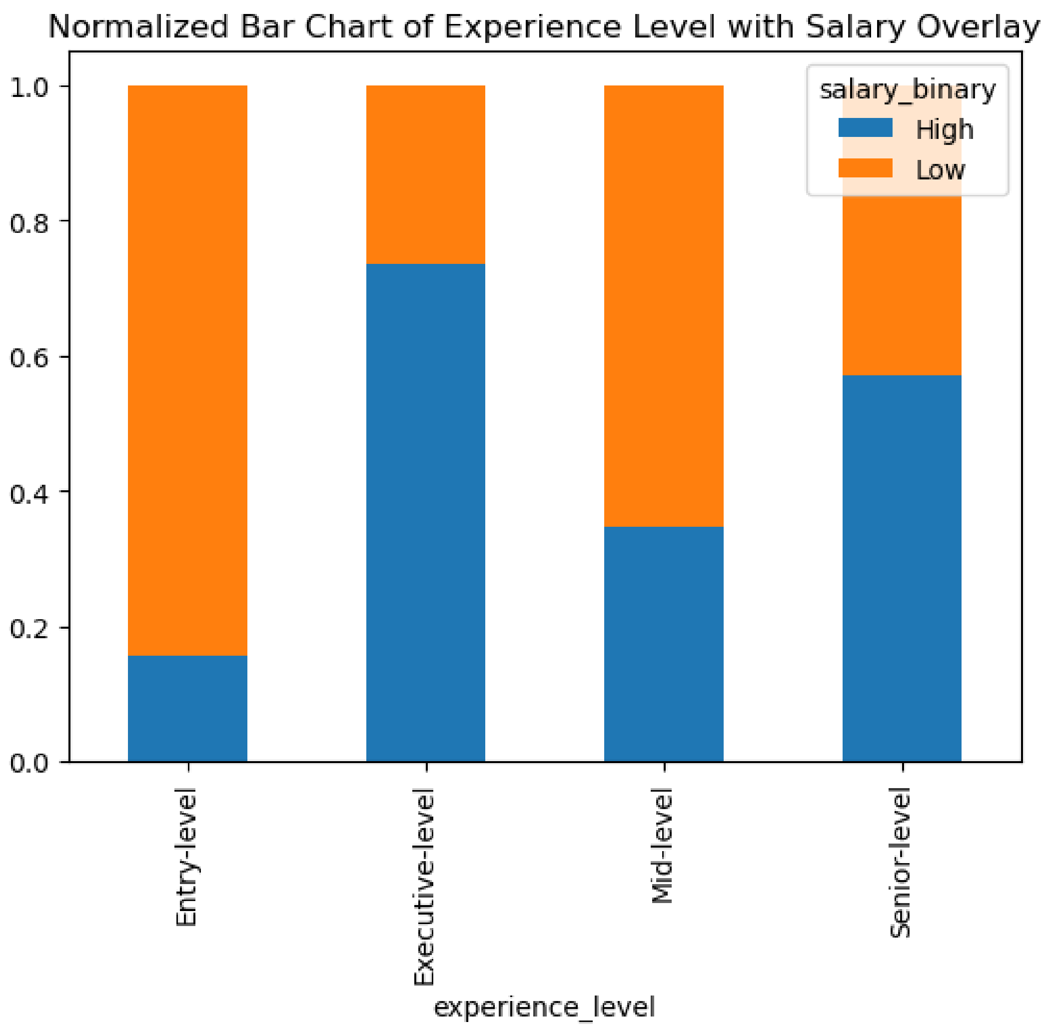


[55]:
```python
# Bar graph of experience_level by salary
experience_salary = pd.crosstab(data_cleaned['experience_level'],␣
 ↪data_cleaned['salary_binary'])
experience_salary.plot(kind = 'bar', stacked = True)
```

[55]: <Axes: xlabel='experience_level'>

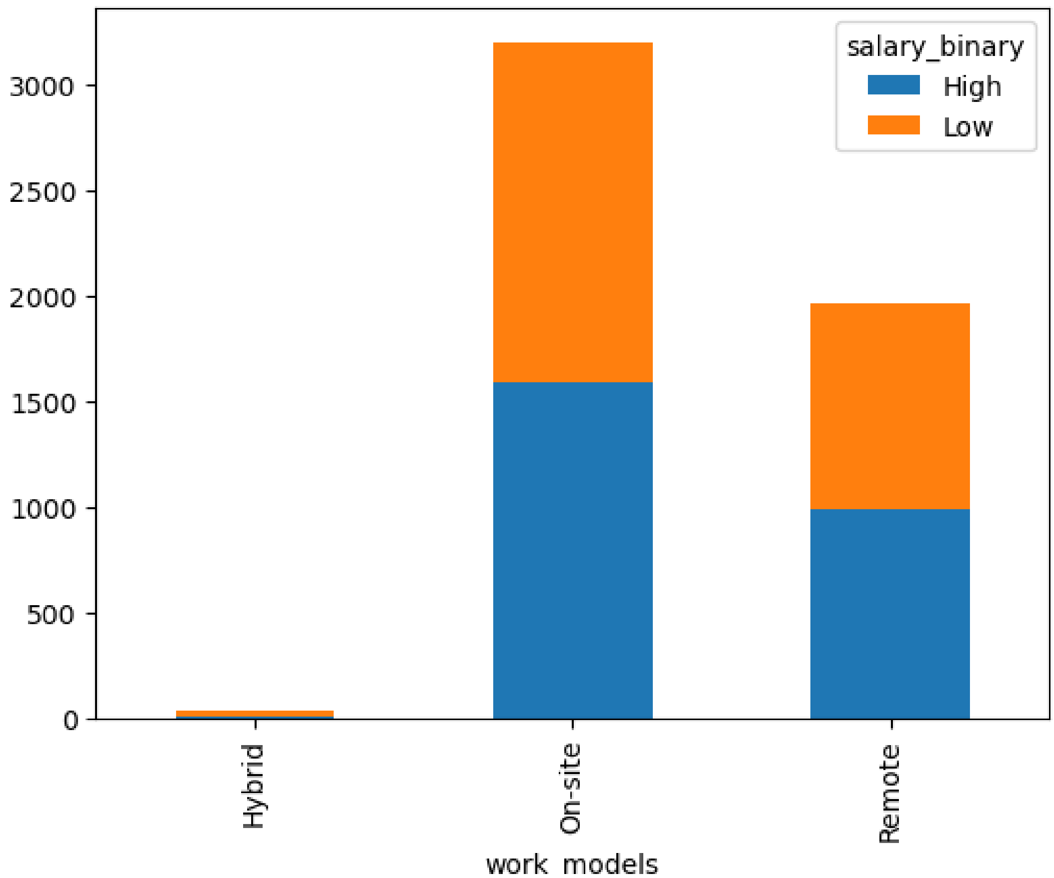


[56]:
```
# Normalized
experience_salary_norm = experience_salary.div(experience_salary.sum(1), axis =␣
↪0)
experience_salary_norm.plot(kind = 'bar', stacked = True)
plt.title('Normalized Bar Chart of Experience Level with Salary Overlay')
plt.show()
```

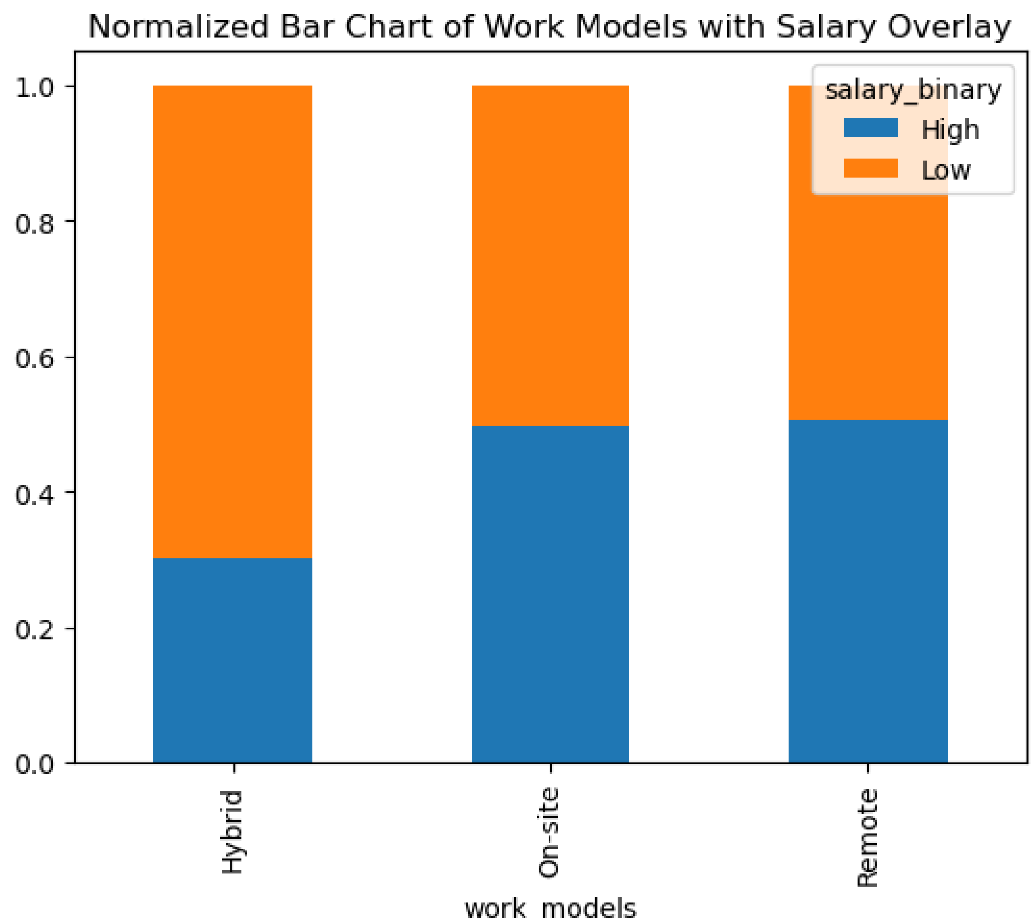



```
[57]: # Bar graph of work_models by salary
work_salary = pd.crosstab(data_cleaned['work_models'],
 ↪data_cleaned['salary_binary'])
work_salary.plot(kind = 'bar', stacked = True)
```

```
[57]: <Axes: xlabel='work_models'>
```



[58]:
```
# Normalized
work_salary_norm = work_salary.div(work_salary.sum(1), axis = 0)
work_salary_norm.plot(kind = 'bar', stacked = True)
plt.title('Normalized Bar Chart of Work Models with Salary Overlay')
plt.show()
```



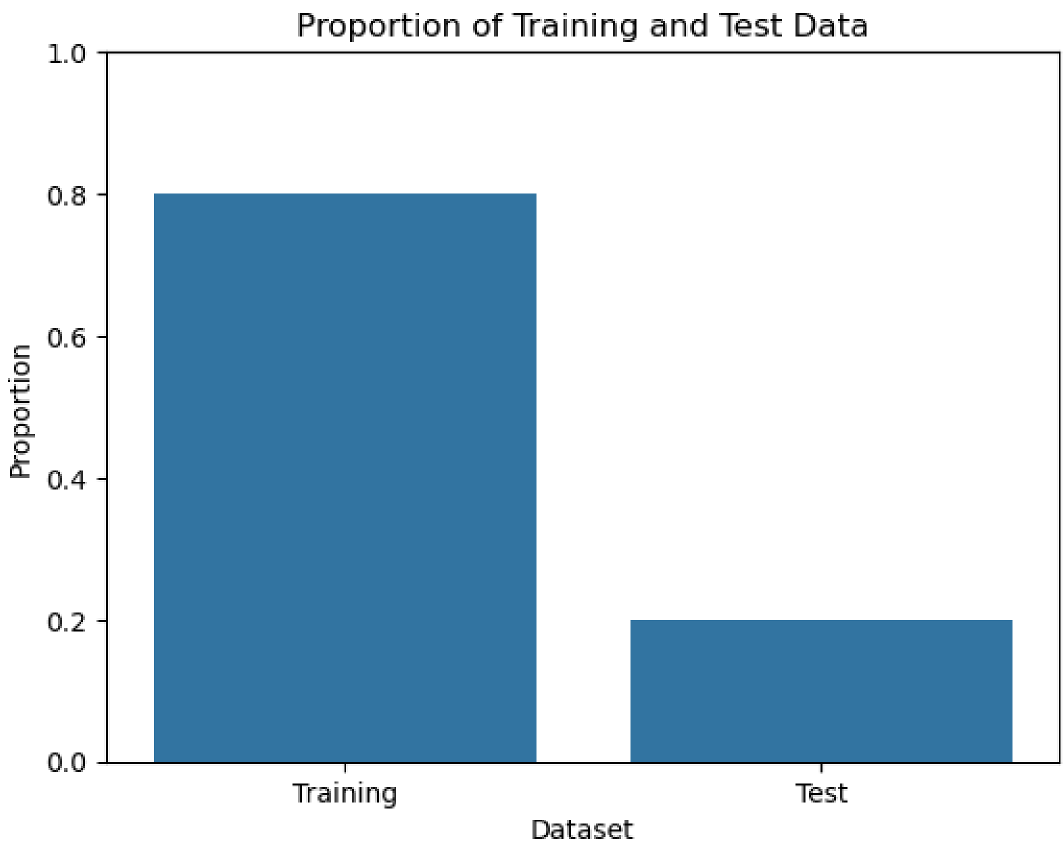
Split dataset to training and test

[59]:
```
# Partition dataset
data_train, data_test = train_test_split(data_cleaned, test_size = 0.2,␣
↪random_state = 7)
```

[60]:
```
# Confirm split proportions
split_proportions = pd.DataFrame({'Dataset': ['Training', 'Test'],
                                  'Proportion': [len(data_train) /␣
↪len(data_cleaned), len(data_test) / len(data_cleaned)]
})
```

[61]:
```
sns.barplot(x = 'Dataset', y = 'Proportion', data = split_proportions)

plt.title('Proportion of Training and Test Data')
plt.ylabel('Proportion')
plt.xlabel('Dataset')
plt.ylim(0, 1)
```

```
plt.show()
```



[62]:
```
# Check if rebalance is needed
data_train['salary_binary'].value_counts()
```

[62]:
```
salary_binary
High    2085
Low     2078
Name: count, dtype: int64
```

[63]:
```
# Prep data for modeling
y_train = data_train[['salary_binary']]
y_test = data_test[['salary_binary']]
data_train = pd.get_dummies(data_train,
                            prefix = None,
                            columns = ['company_size',␣
 ↪'experience_level','work_models'],
                            drop_first = False)
data_test = pd.get_dummies(data_test,
                           prefix = None,
```

```
                            columns = ['company_size',␣
↪'experience_level','work_models'],
                            drop_first = False)
```

[64]:
```
data_train.columns
```

[64]:
```
Index(['job_title', 'employment_type', 'work_year', 'salary', 'salary_in_usd',
       'company_location', 'salary_binary', 'company_size_Large',
       'company_size_Medium', 'company_size_Small',
       'experience_level_Entry-level', 'experience_level_Executive-level',
       'experience_level_Mid-level', 'experience_level_Senior-level',
       'work_models_Hybrid', 'work_models_On-site', 'work_models_Remote'],
      dtype='object')
```

[65]:
```
X_train = data_train[['company_size_Large', 'company_size_Medium',
                      'company_size_Small', 'experience_level_Entry-level',
                      'experience_level_Executive-level',
                      'experience_level_Mid-level',␣
↪'experience_level_Senior-level',
                      'work_models_Hybrid', 'work_models_On-site',
                      'work_models_Remote']]
X_test = data_test[['company_size_Large', 'company_size_Medium',
                    'company_size_Small', 'experience_level_Entry-level',
                    'experience_level_Executive-level',␣
↪'experience_level_Mid-level',
                    'experience_level_Senior-level', 'work_models_Hybrid',
                    'work_models_On-site', 'work_models_Remote']]
```

[66]:
```
X_names = ["company_size_Large", "company_size_Medium",
           "company_size_Small", "experience_level_Entry-level",
           "experience_level_Executive-level", "experience_level_Mid-level",
           "experience_level_Senior-level", "work_models_Hybrid",
           "work_models_On-site", "work_models_Remote"]
y_names = ["High", "Low"]
```

Random Forest

[67]:
```
rf = RandomForestClassifier(n_estimators = 100, criterion = "gini").
↪fit(X_train, y_train)
```

```
/var/folders/6j/qtnqw_nn0bj948l89lgcc38m0000gn/T/ipykernel_17069/3314980583.py:1
: DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples,), for example using
ravel().
  rf = RandomForestClassifier(n_estimators = 100, criterion =
"gini").fit(X_train, y_train)
```

[68]:
```
y_pred_rf = rf.predict(X_test)
```

[69]:
```python
cm = confusion_matrix(y_test, y_pred_rf)
cm
```

[69]:
```
array([[421,  91],
       [325, 204]])
```

[70]:
```python
print("Random Forest Accuracy:", accuracy_score(y_test, y_pred_rf))
print("Classification Report:\n", classification_report(y_test, y_pred_rf))
```

```
Random Forest Accuracy: 0.6003842459173871
Classification Report:
               precision    recall  f1-score   support

        High       0.56      0.82      0.67       512
         Low       0.69      0.39      0.50       529

    accuracy                           0.60      1041
   macro avg       0.63      0.60      0.58      1041
weighted avg       0.63      0.60      0.58      1041
```

[71]:
```python
TN = cm[1][1]
FP = cm[1][0]
FN = cm[0][1]
TP = cm[0][0]
TAN = TN + FP
TAP = FN + TP
TPN = TN + FN
TPP = FP + TP
GT = TN + FP + FN + TP

Accuracy = (TN + TP)/GT
ErrorRate = 1 - Accuracy
Sensitivity = TP/TAP
Recall = Sensitivity
Specificity = TN/TAN
Precision = TP/TPP
F1 = (2 * Precision * Recall)/(Precision + Recall)
F2 = (5 * Precision * Recall)/((4 * Precision) + Recall)
F0_5 = (1.25 * Precision * Recall)/((.25 * Precision) + Recall)
```

[72]:
```python
print("Random Forest Accuracy:", Accuracy)
print("Random Forest Error Rate:", ErrorRate)
print("Random Forest Sensitivity:", Recall)
print("Random Forest Specificity:", Specificity)
print("Random Forest Precision:", Precision)
print("Random Forest F1:", F1)
print("Random Forest F2:", F2)
```

```
print("Random Forest F0.5:", F0_5)
```

```
Random Forest Accuracy: 0.6003842459173871
Random Forest Error Rate: 0.39961575408261285
Random Forest Sensitivity: 0.822265625
Random Forest Specificity: 0.3856332703213611
Random Forest Precision: 0.564343163538874
Random Forest F1: 0.6693163751987282
Random Forest F2: 0.7534001431639228
Random Forest F0.5: 0.6021167048054921
```

Logistic Regression

[73]:
```
log_reg = LogisticRegression().fit(X_train, y_train.values.ravel())
```

[74]:
```
y_pred_logreg = log_reg.predict(X_test)
```

[75]:
```
cm_logreg = confusion_matrix(y_test, y_pred_logreg)
cm_logreg
```

[75]:
```
array([[421,  91],
       [319, 210]])
```

[76]:
```
print("Logistic Regression Accuracy:", accuracy_score(y_test, y_pred_logreg))
print("Classification Report:\n", classification_report(y_test, y_pred_logreg))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_logreg))
```

```
Logistic Regression Accuracy: 0.6061479346781941
Classification Report:
               precision    recall  f1-score   support

        High       0.57      0.82      0.67       512
         Low       0.70      0.40      0.51       529

    accuracy                           0.61      1041
   macro avg       0.63      0.61      0.59      1041
weighted avg       0.63      0.61      0.59      1041

Confusion Matrix:
 [[421  91]
 [319 210]]
```

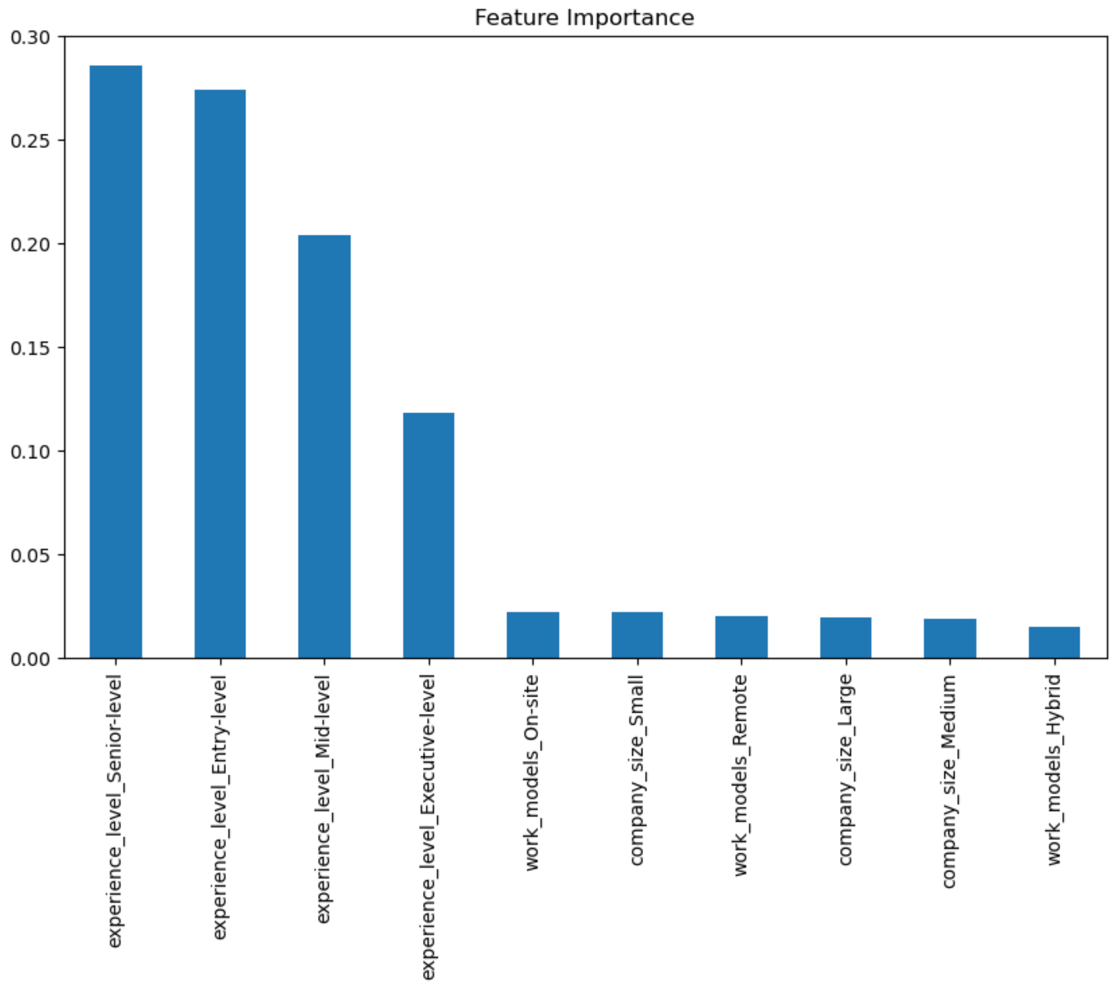
[77]:
```
TN = cm_logreg[1][1]
FP = cm_logreg[1][0]
FN = cm_logreg[0][1]
TP = cm_logreg[0][0]
TAN = TN + FP
TAP = FN + TP
TPN = TN + FN
```

```
TPP = FP + TP
GT = TN + FP + FN + TP

Accuracy = (TN + TP)/GT
ErrorRate = 1 - Accuracy
Sensitivity = TP/TAP
Recall = Sensitivity
Specificity = TN/TAN
Precision = TP/TPP
F1 = (2 * Precision * Recall)/(Precision + Recall)
F2 = (5 * Precision * Recall)/((4 * Precision) + Recall)
F0_5 = (1.25 * Precision * Recall)/((.25 * Precision) + Recall)
```

[78]:
```
print("Logistic Regression Accuracy:", Accuracy)
print("Logistic Regression Error Rate:", ErrorRate)
print("Logistic Regression Sensitivity:", Recall)
print("Logistic Regression Specificity:", Specificity)
print("Logistic Regression Precision:", Precision)
print("Logistic Regression F1:", F1)
print("Logistic Regression F2:", F2)
print("Logistic Regression F0.5:", F0_5)
```

```
Logistic Regression Accuracy: 0.6061479346781941
Logistic Regression Error Rate: 0.39385206532180594
Logistic Regression Sensitivity: 0.822265625
Logistic Regression Specificity: 0.39697542533081287
Logistic Regression Precision: 0.5689189189189189
Logistic Regression F1: 0.6725239616613419
Logistic Regression F2: 0.7550215208034432
Logistic Regression F0.5: 0.606278801843318
```

[79]:
```
# Analyze feature importance
importance = pd.Series(rf.feature_importances_, index = X_train.columns)
importance.sort_values(ascending = False).plot(kind = 'bar', figsize = (10, 6))
plt.title('Feature Importance')
plt.show()
```



CART

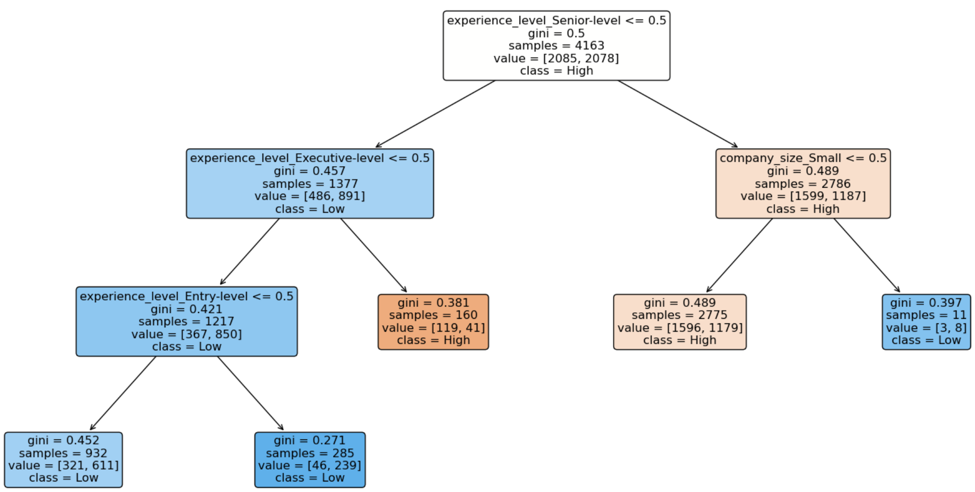
[80]:
```
cart = DecisionTreeClassifier(criterion = "gini", max_leaf_nodes = 5).
 ↪fit(X_train, y_train)
```

[81]:
```
plt.figure()
plt.figure(figsize = (20,10))
plot_tree(cart,
          feature_names = X_names,
          class_names = y_names,
          filled = True,
          rounded = True)
```

[81]:
```
[Text(0.5555555555555556, 0.875, 'experience_level_Senior-level <= 0.5\ngini =
0.5\nsamples = 4163\nvalue = [2085, 2078]\nclass = High'),
 Text(0.3333333333333333, 0.625, 'experience_level_Executive-level <= 0.5\ngini
= 0.457\nsamples = 1377\nvalue = [486, 891]\nclass = Low'),
```

```
 Text(0.2222222222222222, 0.375, 'experience_level_Entry-level <= 0.5\ngini =
0.421\nsamples = 1217\nvalue = [367, 850]\nclass = Low'),
 Text(0.1111111111111111, 0.125, 'gini = 0.452\nsamples = 932\nvalue = [321,
611]\nclass = Low'),
 Text(0.3333333333333333, 0.125, 'gini = 0.271\nsamples = 285\nvalue = [46,
239]\nclass = Low'),
 Text(0.4444444444444444, 0.375, 'gini = 0.381\nsamples = 160\nvalue = [119,
41]\nclass = High'),
 Text(0.7777777777777778, 0.625, 'company_size_Small <= 0.5\ngini =
0.489\nsamples = 2786\nvalue = [1599, 1187]\nclass = High'),
 Text(0.6666666666666666, 0.375, 'gini = 0.489\nsamples = 2775\nvalue = [1596,
1179]\nclass = High'),
 Text(0.8888888888888888, 0.375, 'gini = 0.397\nsamples = 11\nvalue = [3,
8]\nclass = Low')]
```

```
<Figure size 640x480 with 0 Axes>
```



[82]:
```python
y_pred_cart = cart.predict(X_test)
y_pred_cart
```

[82]:
```
array(['High', 'High', 'Low', …, 'High', 'High', 'High'], dtype=object)
```

[83]:
```python
cm_cart = confusion_matrix(y_test, y_pred_cart)
cm_cart
```

[83]:
```
array([[421,  91],
       [319, 210]])
```

[84]:
```python
print("CART Accuracy:", accuracy_score(y_test, y_pred_cart))
print("Classification Report:\n", classification_report(y_test, y_pred_cart))
```

```
CART Accuracy: 0.6061479346781941
Classification Report:
               precision    recall  f1-score   support

        High       0.57      0.82      0.67       512
         Low       0.70      0.40      0.51       529

    accuracy                           0.61      1041
   macro avg       0.63      0.61      0.59      1041
weighted avg       0.63      0.61      0.59      1041
```

[85]:
```python
TN = cm_cart[1][1]
FP = cm_cart[1][0]
FN = cm_cart[0][1]
TP = cm_cart[0][0]
TAN = TN + FP
TAP = FN + TP
TPN = TN + FN
TPP = FP + TP
GT = TN + FP + FN + TP

Accuracy = (TN + TP)/GT
ErrorRate = 1 - Accuracy
Sensitivity = TP/TAP
Recall = Sensitivity
Specificity = TN/TAN
Precision = TP/TPP
F1 = (2 * Precision * Recall)/(Precision + Recall)
F2 = (5 * Precision * Recall)/((4 * Precision) + Recall)
F0_5 = (1.25 * Precision * Recall)/((.25 * Precision) + Recall)
```

[86]:
```python
print("CART Accuracy:", Accuracy)
print("CART Error Rate:", ErrorRate)
print("CART Sensitivity:", Recall)
print("CART Specificity:", Specificity)
print("CART Precision:", Precision)
print("CART F1:", F1)
print("CART F2:", F2)
print("CART F0.5:", F0_5)
```

```
CART Accuracy: 0.6061479346781941
CART Error Rate: 0.39385206532180594
CART Sensitivity: 0.822265625
CART Specificity: 0.39697542533081287
```

```
CART Precision: 0.5689189189189189
CART F1: 0.6725239616613419
CART F2: 0.7550215208034432
CART F0.5: 0.606278801843318
```

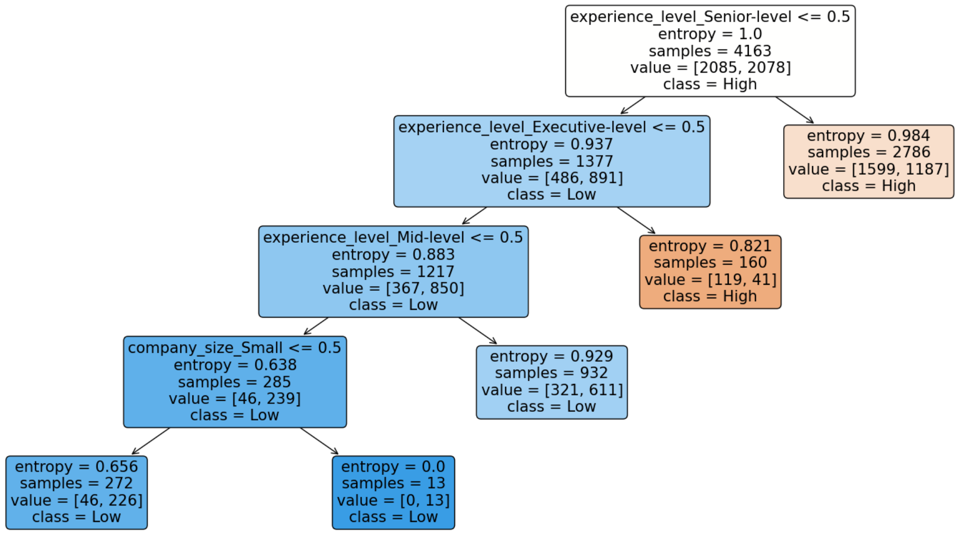
C5.0 Model

[87]:
```python
c50 = DecisionTreeClassifier(criterion = "entropy", max_leaf_nodes = 5).
↪fit(X_train, y_train)
```

[88]:
```python
plt.figure()
plt.figure(figsize = (20,10))
plot_tree(c50,
          feature_names = X_names,
          class_names = y_names,
          filled = True,
          rounded = True)
```

[88]:
```
[Text(0.7142857142857143, 0.9, 'experience_level_Senior-level <= 0.5\nentropy =
1.0\nsamples = 4163\nvalue = [2085, 2078]\nclass = High'),
 Text(0.5714285714285714, 0.7, 'experience_level_Executive-level <= 0.5\nentropy
= 0.937\nsamples = 1377\nvalue = [486, 891]\nclass = Low'),
 Text(0.42857142857142855, 0.5, 'experience_level_Mid-level <= 0.5\nentropy =
0.883\nsamples = 1217\nvalue = [367, 850]\nclass = Low'),
 Text(0.2857142857142857, 0.3, 'company_size_Small <= 0.5\nentropy =
0.638\nsamples = 285\nvalue = [46, 239]\nclass = Low'),
 Text(0.14285714285714285, 0.1, 'entropy = 0.656\nsamples = 272\nvalue = [46,
226]\nclass = Low'),
 Text(0.42857142857142855, 0.1, 'entropy = 0.0\nsamples = 13\nvalue = [0,
13]\nclass = Low'),
 Text(0.5714285714285714, 0.3, 'entropy = 0.929\nsamples = 932\nvalue = [321,
611]\nclass = Low'),
 Text(0.7142857142857143, 0.5, 'entropy = 0.821\nsamples = 160\nvalue = [119,
41]\nclass = High'),
 Text(0.8571428571428571, 0.7, 'entropy = 0.984\nsamples = 2786\nvalue = [1599,
1187]\nclass = High')]
```

```
<Figure size 640x480 with 0 Axes>
```



[89]:
```
y_pred_c50 = c50.predict(X_test)
```

[90]:
```
cm_c50 = confusion_matrix(y_test, y_pred_c50)
cm_c50
```

[90]:
```
array([[423,  89],
       [321, 208]])
```

[91]:
```
print("C5.0 Accuracy:", accuracy_score(y_test, y_pred_c50))
print("Classification Report:\n", classification_report(y_test, y_pred_c50))
```

```
C5.0 Accuracy: 0.6061479346781941
Classification Report:
               precision    recall  f1-score   support

        High       0.57      0.83      0.67       512
         Low       0.70      0.39      0.50       529

    accuracy                           0.61      1041
   macro avg       0.63      0.61      0.59      1041
weighted avg       0.64      0.61      0.59      1041
```

[92]:
```
TN = cm_c50[1][1]
FP = cm_c50[1][0]
FN = cm_c50[0][1]
TP = cm_c50[0][0]
TAN = TN + FP
TAP = FN + TP
```

```
TPN = TN + FN
TPP = FP + TP
GT = TN + FP + FN + TP

Accuracy = (TN + TP)/GT
ErrorRate = 1 - Accuracy
Sensitivity = TP/TAP
Recall = Sensitivity
Specificity = TN/TAN
Precision = TP/TPP
F1 = (2 * Precision * Recall)/(Precision + Recall)
F2 = (5 * Precision * Recall)/((4 * Precision) + Recall)
F0_5 = (1.25 * Precision * Recall)/((.25 * Precision) + Recall)
```

[93]:
```
print("C5.0 Accuracy:", Accuracy)
print("C5.0 Error Rate:", ErrorRate)
print("C5.0 Sensitivity:", Recall)
print("C5.0 Specificity:", Specificity)
print("C5.0 Precision:", Precision)
print("C5.0 F1:", F1)
print("C5.0 F2:", F2)
print("C5.0 F0.5:", F0_5)
```

```
C5.0 Accuracy: 0.6061479346781941
C5.0 Error Rate: 0.39385206532180594
C5.0 Sensitivity: 0.826171875
C5.0 Specificity: 0.3931947069943289
C5.0 Precision: 0.5685483870967742
C5.0 F1: 0.6735668789808917
C5.0 F2: 0.7575214899713466
C5.0 F0.5: 0.6063646788990825
```

Naive Bayes

[94]:
```
nb = MultinomialNB().fit(X_train, y_train.values.ravel())
```

[95]:
```
y_pred_nb = nb.predict(X_test)
```

[96]:
```
cm_nb = confusion_matrix(y_test, y_pred_nb)
cm_nb
```

[96]:
```
array([[418,  94],
       [318, 211]])
```

[97]:
```
print("Naive Bayes Accuracy:", accuracy_score(y_test, y_pred_nb))
print("Classification Report:\n", classification_report(y_test, y_pred_nb))
```

```
Naive Bayes Accuracy: 0.6042267050912584
Classification Report:
```

```
              precision    recall  f1-score   support

        High       0.57      0.82      0.67       512
         Low       0.69      0.40      0.51       529

    accuracy                           0.60      1041
   macro avg       0.63      0.61      0.59      1041
weighted avg       0.63      0.60      0.59      1041
```

[98]:
```
TN = cm_nb[1][1]
FP = cm_nb[1][0]
FN = cm_nb[0][1]
TP = cm_nb[0][0]
TAN = TN + FP
TAP = FN + TP
TPN = TN + FN
TPP = FP + TP
GT = TN + FP + FN + TP

Accuracy = (TN + TP)/GT
ErrorRate = 1 - Accuracy
Sensitivity = TP/TAP
Recall = Sensitivity
Specificity = TN/TAN
Precision = TP/TPP
F1 = (2 * Precision * Recall)/(Precision + Recall)
F2 = (5 * Precision * Recall)/((4 * Precision) + Recall)
F0_5 = (1.25 * Precision * Recall)/((.25 * Precision) + Recall)
```

[99]:
```
print("NB Accuracy:", Accuracy)
print("NB Error Rate:", ErrorRate)
print("NB Sensitivity:", Recall)
print("NB Specificity:", Specificity)
print("NB Precision:", Precision)
print("NB F1:", F1)
print("NB F2:", F2)
print("NB F0.5:", F0_5)
```

```
NB Accuracy: 0.6042267050912584
NB Error Rate: 0.3957732949087416
NB Sensitivity: 0.81640625
NB Specificity: 0.3988657844990548
NB Precision: 0.5679347826086957
NB F1: 0.6698717948717948
NB F2: 0.7507183908045977
NB F0.5: 0.6047453703703703
```

Baseline Model

[100]:
```
baseline = DummyClassifier(strategy = 'uniform', random_state = 7).fit(X_train,␣
↪y_train.values.ravel())
```

[101]:
```
y_pred_baseline = baseline.predict(X_test)
```

[102]:
```
cm_baseline = confusion_matrix(y_test, y_pred_baseline)
cm_baseline
```

[102]:
```
array([[250, 262],
       [276, 253]])
```

[103]:
```
TN = cm_baseline[1][1]
FP = cm_baseline[1][0]
FN = cm_baseline[0][1]
TP = cm_baseline[0][0]
TAN = TN + FP
TAP = FN + TP
TPN = TN + FN
TPP = FP + TP
GT = TN + FP + FN + TP

Accuracy = (TN + TP)/GT
ErrorRate = 1 - Accuracy
Sensitivity = TP/TAP
Recall = Sensitivity
Specificity = TN/TAN
Precision = TP/TPP
F1 = (2 * Precision * Recall)/(Precision + Recall)
F2 = (5 * Precision * Recall)/((4 * Precision) + Recall)
F0_5 = (1.25 * Precision * Recall)/((.25 * Precision) + Recall)
```

[104]:
```
print("Baseline Accuracy:", Accuracy)
print("Baseline Error Rate:", ErrorRate)
print("Baseline Sensitivity:", Recall)
print("Baseline Specificity:", Specificity)
print("Baseline Precision:", Precision)
print("Baseline F1:", F1)
print("Baseline F2:", F2)
print("Baseline F0.5:", F0_5)
```

```
Baseline Accuracy: 0.48318924111431316
Baseline Error Rate: 0.5168107588856868
Baseline Sensitivity: 0.48828125
Baseline Specificity: 0.4782608695652174
Baseline Precision: 0.4752851711026616
Baseline F1: 0.4816955684007707
Baseline F2: 0.48562548562548563
Baseline F0.5: 0.47782874617737
```

Data+Science+Salaries

December 9, 2024

[36]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from scipy.stats import zscore
import seaborn as sns
```

[37]:
```
df = pd.read_csv('data_science_salaries.csv')
```

[38]:
```
# Remove employee residence column
df.drop(columns = ['employee_residence'], inplace = True)
```

[39]:
```
# Filter data set so we only see United States for company location
df_us = df[df['company_location'] == 'United States']
```

[40]:
```
df_us['company_location'].nunique()
```

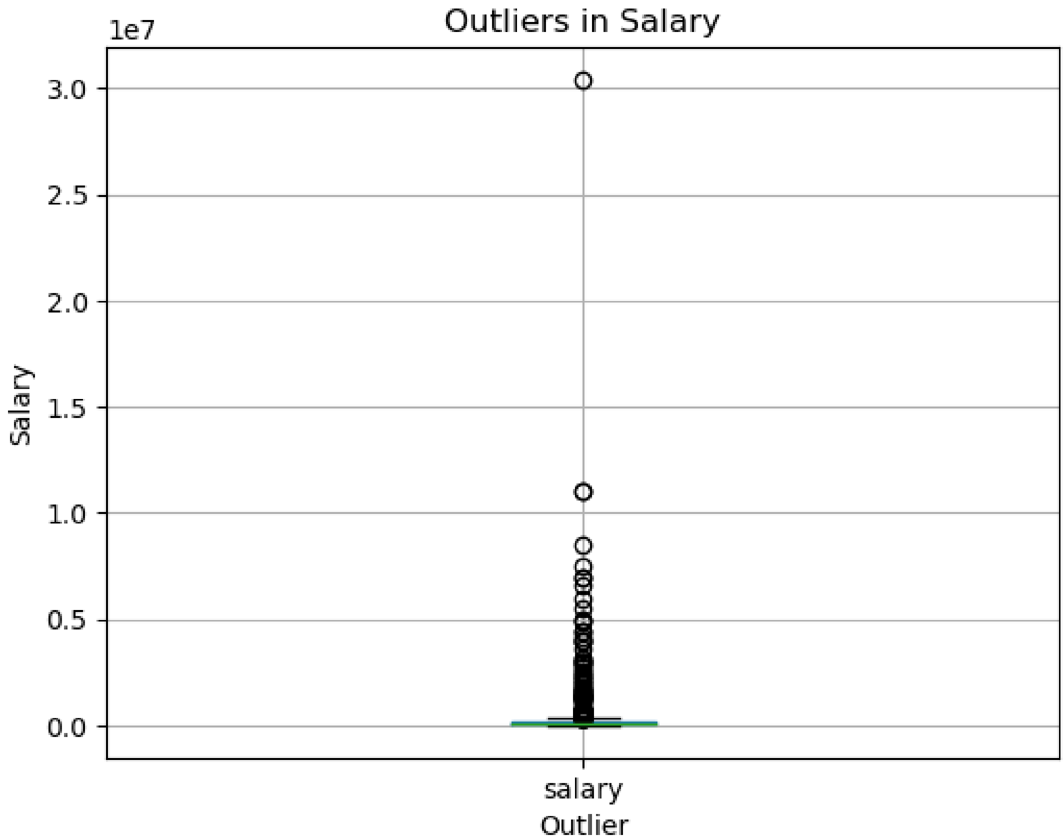
[40]:
```
1
```

[41]:
```
missing_values = df.isnull().sum()
print(missing_values)
```

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

[42]:
```
# Look for outliers
# Boxplot to visualize outliers
df.boxplot(column = 'salary')
plt.title('Outliers in Salary')
plt.suptitle('')
```

```
plt.xlabel('Outlier')
plt.ylabel('Salary')
plt.show()
```



[43]:
```
Q1 = df['salary'].quantile(0.2)
Q3 = df['salary'].quantile(0.8)
IQR = Q3 - Q1

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

outliers = df_us[(df_us['salary'] < lower_bound) | (df_us['salary'] >␣
↪upper_bound)]
outliers = outliers[['job_title','experience_level','work_models','salary']]
outliers.sort_values(by = 'salary', ascending = False, inplace = True)
print(f"lower {lower_bound}, upper {upper_bound}")
display(outliers)
```

```
lower -87500.0, upper 372500.0
                               job_title experience_level work_models  \
```

6353	BI Data Analyst	Mid-level	Hybrid
6381	Data Science Manager	Senior-level	Hybrid
5289	Data Scientist	Mid-level	Remote
1611	Data Scientist	Senior-level	On-site
1540	Data Engineer	Mid-level	On-site
296	Machine Learning Scientist	Mid-level	On-site
848	Machine Learning Scientist	Mid-level	On-site
852	Machine Learning Engineer	Mid-level	On-site
329	Research Scientist	Mid-level	On-site
321	Research Engineer	Mid-level	On-site
1509	Analytics Engineer	Mid-level	Remote
1095	Analytics Engineer	Senior-level	Remote
1354	Data Engineer	Executive-level	On-site
6529	Research Scientist	Mid-level	On-site
150	Research Engineer	Senior-level	On-site
84	Research Engineer	Senior-level	On-site
170	Research Scientist	Senior-level	On-site
129	Research Engineer	Mid-level	On-site
6403	Applied Machine Learning Scientist	Mid-level	Hybrid
82	Machine Learning Engineer	Senior-level	Remote
6388	Principal Data Scientist	Executive-level	Remote
203	Applied Scientist	Senior-level	On-site
6554	Data Scientist	Senior-level	Remote
6031	Data Analytics Lead	Senior-level	Remote
2337	Research Scientist	Mid-level	On-site
3414	Machine Learning Engineer	Senior-level	On-site
2697	Data Infrastructure Engineer	Senior-level	On-site
111	Data Engineer	Senior-level	On-site
113	Data Scientist	Mid-level	On-site
1355	Data Engineer	Executive-level	On-site
2453	Research Engineer	Senior-level	On-site
4306	Data Analyst	Senior-level	On-site
2811	ML Engineer	Senior-level	Remote
2433	Data Engineer	Senior-level	On-site
3344	ML Engineer	Senior-level	On-site
213	Data Architect	Senior-level	On-site
6032	Applied Data Scientist	Senior-level	Remote
4126	Data Architect	Senior-level	Remote
2768	Director of Data Science	Executive-level	Remote
5203	Machine Learning Software Engineer	Senior-level	Remote
5509	Data Science Tech Lead	Senior-level	Hybrid
3581	Research Scientist	Senior-level	On-site

	salary
6353	11000000
6381	4000000
5289	2500000
1611	750000

```
1540    750000
296     750000
848     750000
852     750000
329     720000
321     720000
1509    700000
1095    700000
1354    465000
6529    450000
150     450000
84      440000
170     440000
129     440000
6403    423000
82      418000
6388    416000
203     414000
6554    412000
6031    405000
2337    405000
3414    392000
2697    385000
111     385000
113     385000
1355    385000
2453    385000
4306    385000
2811    385000
2433    385000
3344    383910
213     381500
6032    380000
4126    376080
2768    375500
5203    375000
5509    375000
3581    374000
```

[44]:
```python
df_us.plot.scatter(x = 'work_models', y = 'salary')
```

[44]: <Axes: xlabel='work_models', ylabel='salary'>

[45]:
```python
# Understand data types and non-null counts
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6599 entries, 0 to 6598
```

```
Data columns (total 10 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   job_title         6599 non-null   object
 1   experience_level  6599 non-null   object
 2   employment_type   6599 non-null   object
 3   work_models       6599 non-null   object
 4   work_year         6599 non-null   int64
 5   salary            6599 non-null   int64
 6   salary_currency   6599 non-null   object
 7   salary_in_usd     6599 non-null   int64
 8   company_location  6599 non-null   object
 9   company_size      6599 non-null   object
dtypes: int64(3), object(7)
memory usage: 515.7+ KB
None
```

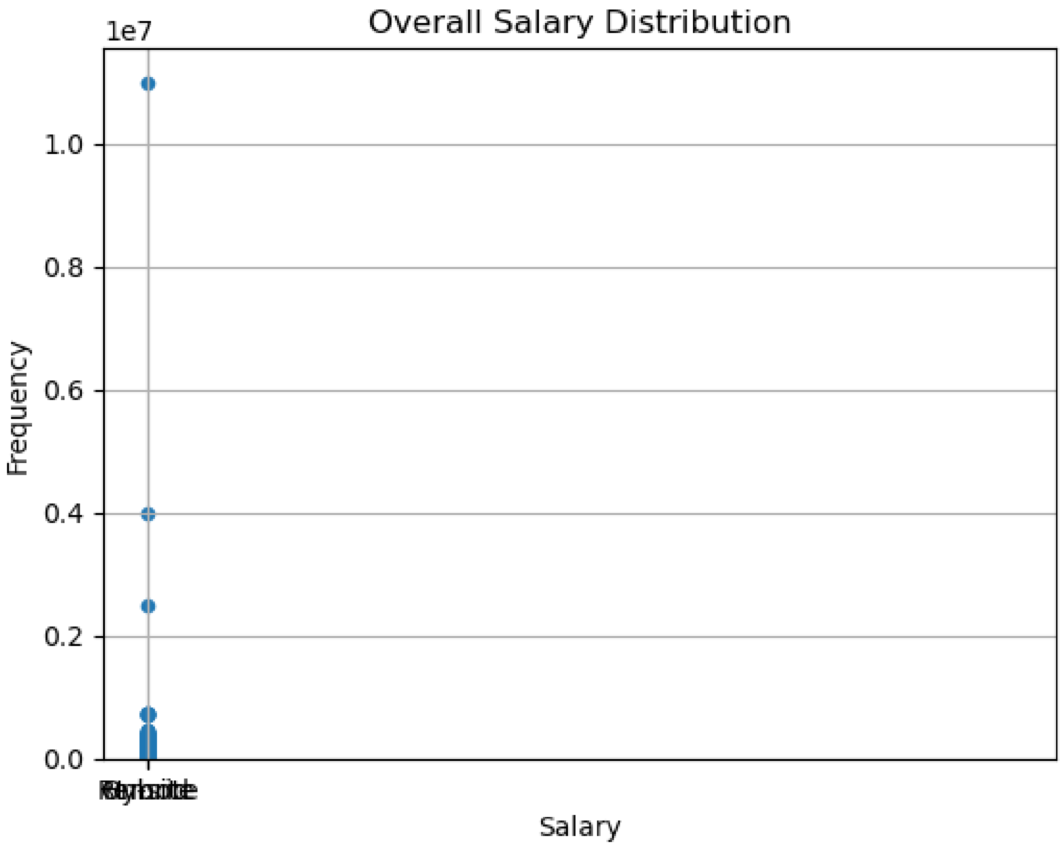
[46]:
```python
# Summary statistics for numeric columns
print(df.describe())
```

```
         work_year        salary  salary_in_usd
count  6599.000000  6.599000e+03    6599.000000
mean   2022.818457  1.792833e+05  145560.558569
std       0.674809  5.263722e+05   70946.838070
min    2020.000000  1.400000e+04   15000.000000
25%    2023.000000  9.600000e+04   95000.000000
50%    2023.000000  1.400000e+05  138666.000000
75%    2023.000000  1.875000e+05  185000.000000
max    2024.000000  3.040000e+07  750000.000000
```

[47]:
```python
# Check unique values for experience_level
print(df['experience_level'].value_counts())
```

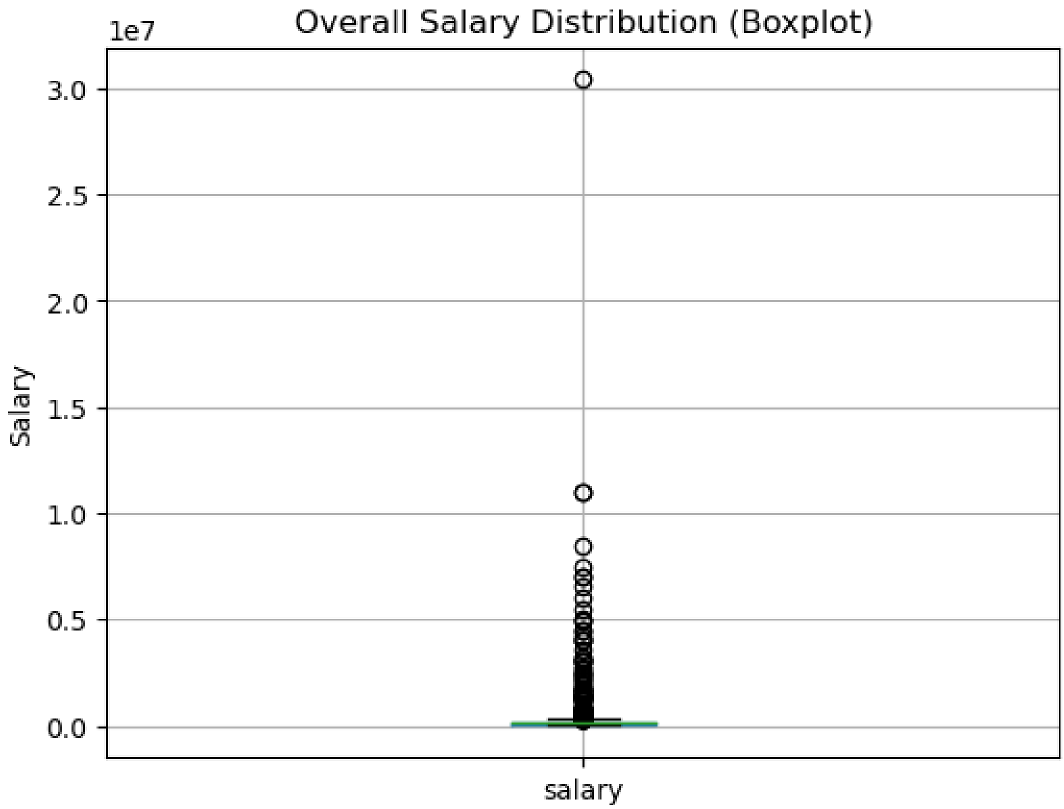
```
experience_level
Senior-level       4105
Mid-level          1675
Entry-level         565
Executive-level     254
Name: count, dtype: int64
```

[48]:
```python
# Visualize Overall Salary Distribution

# Histogram
df['salary'].hist(bins = 30)
plt.title('Overall Salary Distribution')
plt.xlabel('Salary')
plt.ylabel('Frequency')
plt.show()
```



[49]:
```
# Plot histogram for salary distribution
plt.figure(figsize = (10, 6))
plt.hist(df['salary'], bins = 30, color = 'lightblue', edgecolor = 'black')
plt.title('Overall Salary Distribution')
plt.xlabel('Salary')
plt.ylabel('Frequency')
plt.grid(axis = 'y', linestyle = '--', alpha = 0.7)

# Show plot
plt.tight_layout()
plt.show()
```

[50]:
```
# Boxplot of salary distribution
df.boxplot(column = 'salary')
plt.title('Overall Salary Distribution (Boxplot)')
plt.ylabel('Salary')
plt.show()
```



[51]:
```
# Analyze Salary by Experience Level
# Calculate summary statistics
experience_salary = df.groupby('experience_level')['salary'].describe()
print(experience_salary)
```

```
                   count           mean            std      min       25%  \
experience_level
Entry-level        565.0  162796.054867  530557.685605  14000.0   50000.0
Executive-level    254.0  215203.681102  375316.845869  15000.0  137000.0
Mid-level         1675.0  191858.816716  915254.963537  15000.0   73100.0
Senior-level      4105.0  174198.581973  237014.263156  24000.0  119000.0

                       50%       75%         max
experience_level
Entry-level        80000.0  119200.0   6600000.0
Executive-level   183580.0  230000.0   6000000.0
Mid-level         109238.0  152487.5  30400000.0
Senior-level      154600.0  200000.0   7500000.0
```

[52]:
```
# Visualize Salary by Experience Level
df.boxplot(column = 'salary', by = 'experience_level', grid = False)
```

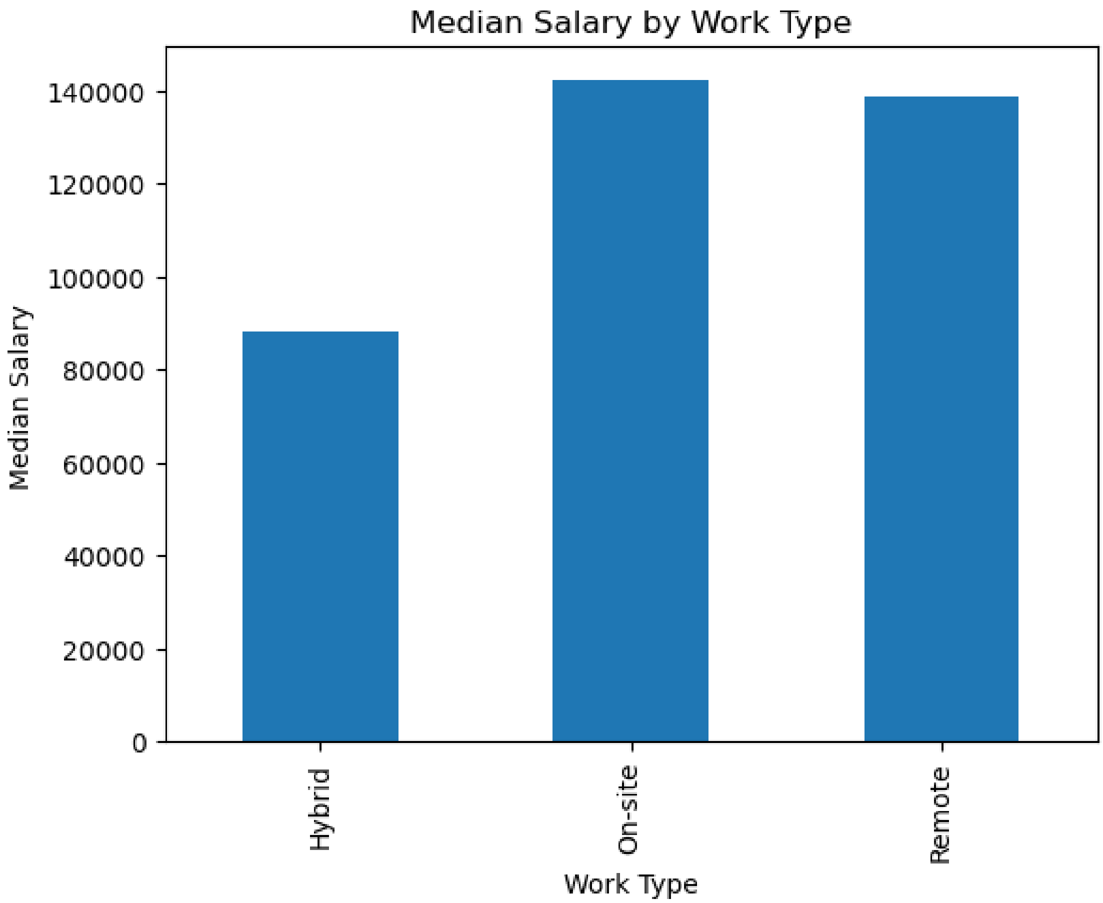
```
plt.title('Salary Distribution by Experience Level')
plt.suptitle('')
plt.xlabel('Experience Level')
plt.ylabel('Salary')
plt.show()
```

[53]:
```
# Analyze salary by work models
# Calculate mean/median salaries by work models
worktype_salary = df.groupby('work_models')['salary'].median()
print(worktype_salary)
```

```
work_models
Hybrid      88000.0
On-site    142200.0
Remote     138750.0
Name: salary, dtype: float64
```

[54]:
```
# Bar plot for median salaries by work type
worktype_salary.plot(kind = 'bar')
plt.title('Median Salary by Work Type')
plt.xlabel('Work Type')
```

```
plt.ylabel('Median Salary')
plt.show()
```



[55]:

```
# Analyze salary by Company Size
# Calculate summary statistics
company_size_salary = df.groupby('company_size')['salary'].mean()
print(company_size_salary)
```

```
company_size
Large     409937.574692
Medium    153820.125597
Small     284998.747059
Name: salary, dtype: float64
```

[56]:

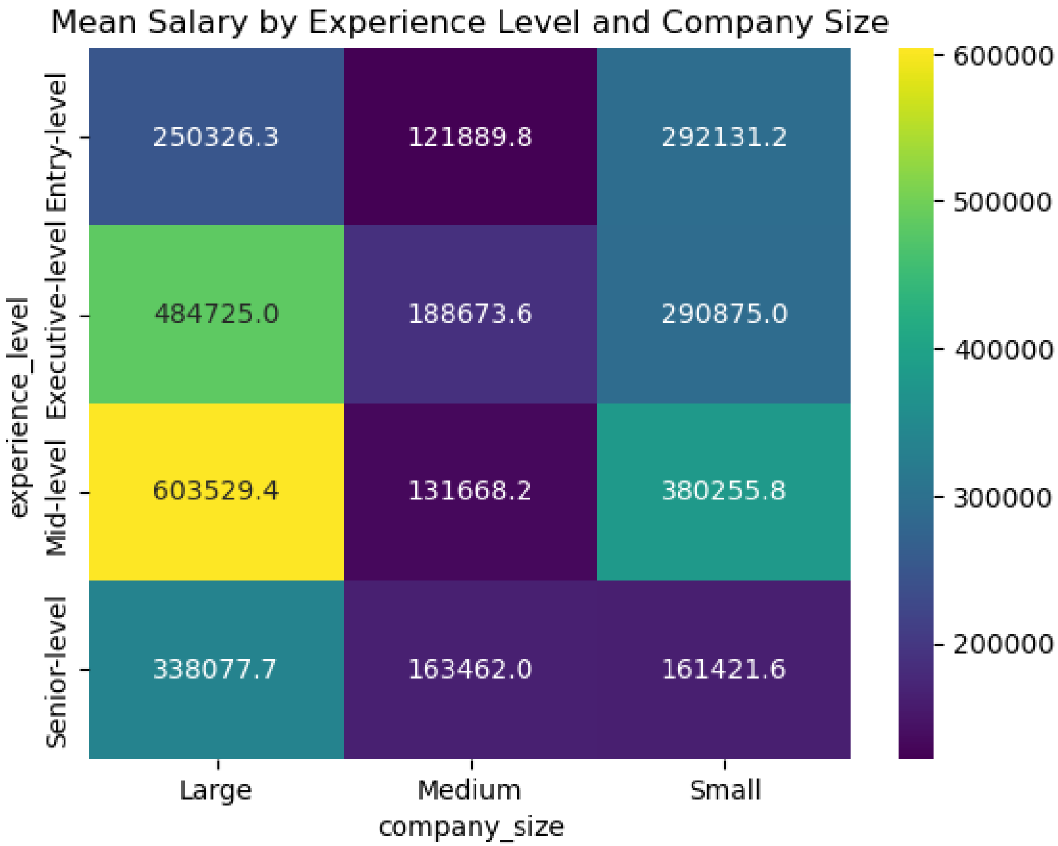
```
# Boxplot for salary distribution by company size
df.boxplot(column = 'salary', by = 'company_size', grid = False)
plt.title('Salary Distribution by Company Size')
plt.suptitle('')
plt.xlabel('Company Size')
```

```
plt.ylabel('Salary')
plt.show()
```

[57]:
```
# Pivot table for mean salary by experience level and company size
pivot_table = df.pivot_table(values = 'salary',
                             index = 'experience_level',
                             columns = 'company_size',
                             aggfunc = 'mean')
print(pivot_table)
```

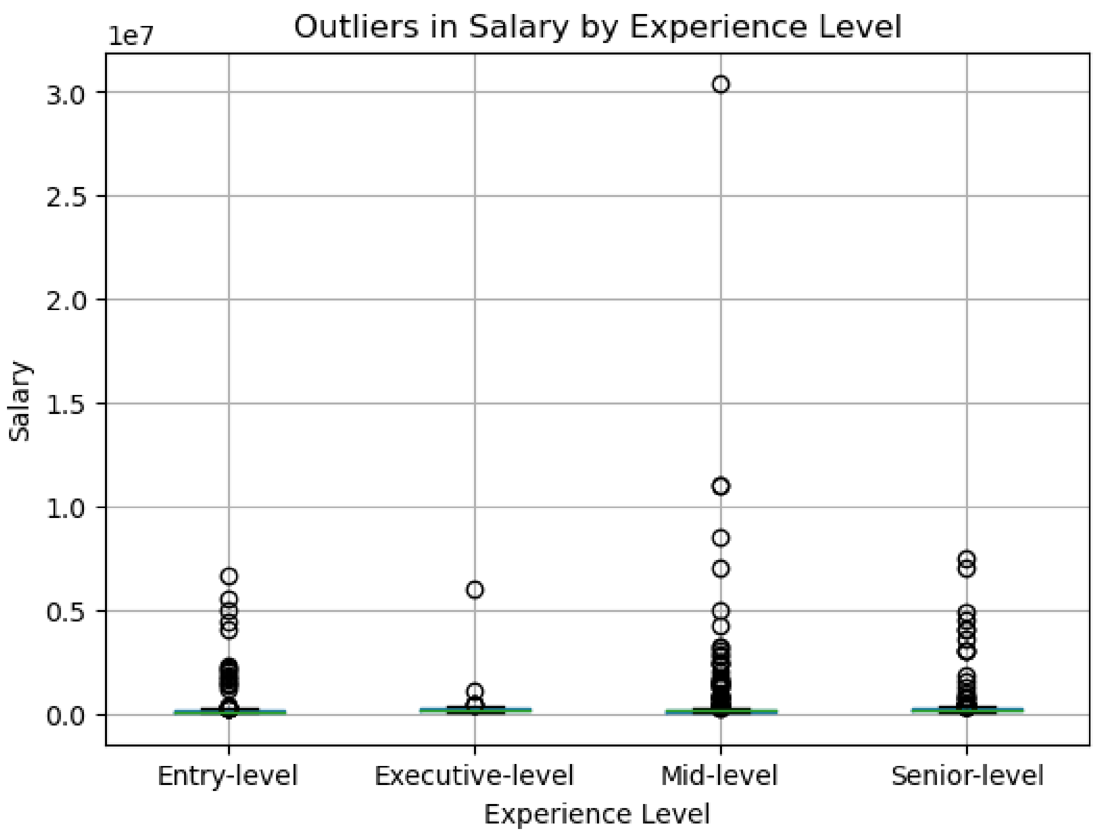
```
company_size              Large         Medium          Small
experience_level
Entry-level       250326.260870  121889.827930  292131.224490
Executive-level   484725.000000  188673.606195  290875.000000
Mid-level         603529.353591  131668.189944  380255.758065
Senior-level      338077.656126  163461.992107  161421.568627
```

[58]:
```
# Heatmap for visualization
sns.heatmap(pivot_table, annot = True, fmt = '.1f', cmap = 'viridis')
plt.title('Mean Salary by Experience Level and Company Size')
plt.show()
```



[59]:

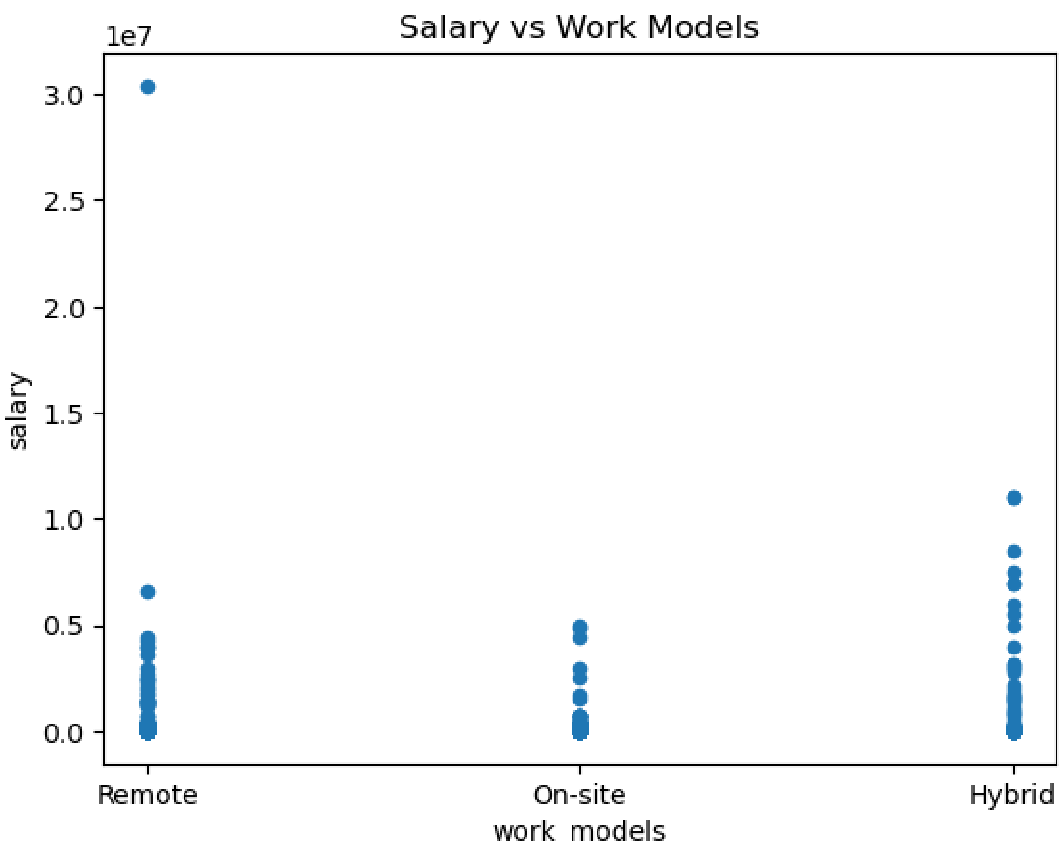
```
# Detect outliers
# Boxplot to visualize outliers
df.boxplot(column = 'salary', by = 'experience_level')
plt.title('Outliers in Salary by Experience Level')
plt.suptitle('')
plt.xlabel('Experience Level')
plt.ylabel('Salary')
plt.show()
```

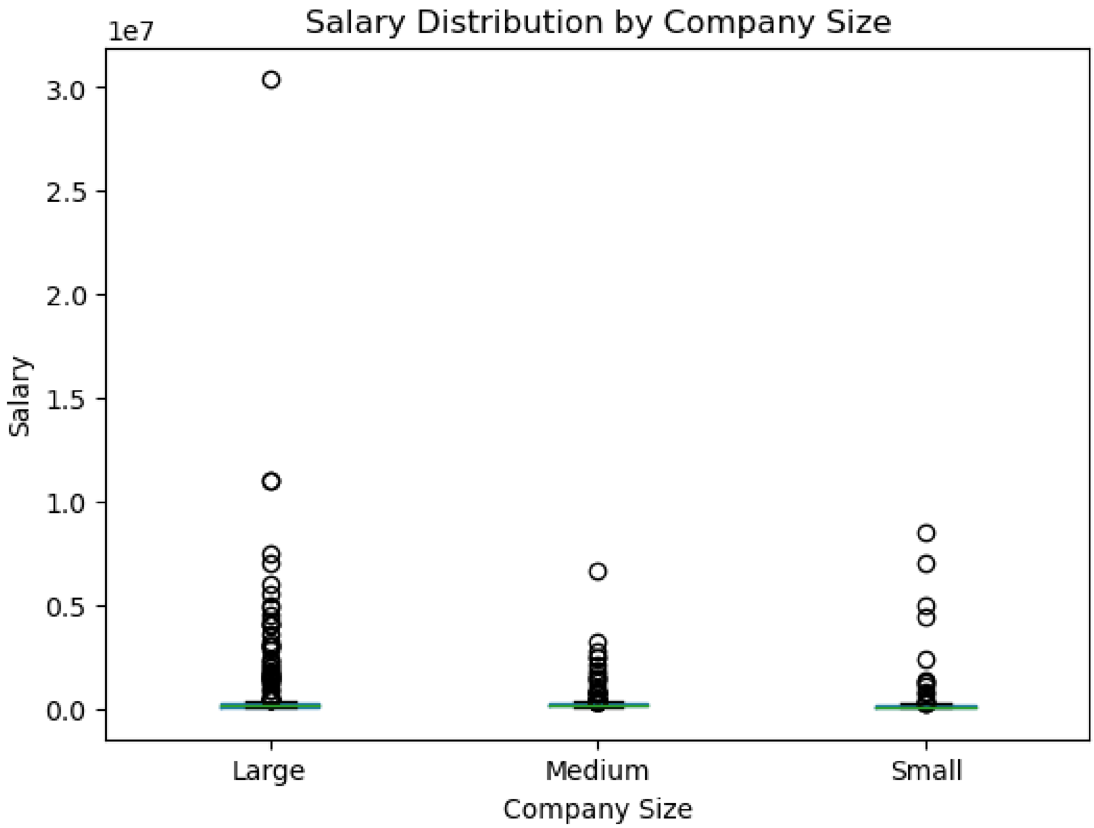


[60]:
```python
# Correlation between work_models and salary, and company_size and salary
data = {
    'work_models': ['Remote', 'Onsite', 'Hybrid'],
    'company_size': ['Small', 'Medium', 'Large'],
    'salary': [70000, 80000, 90000]
}

df_data = pd.DataFrame(data)
df_encoded = pd.get_dummies(df_data, columns = ['work_models', 'company_size'])
numeric_df = df_encoded.select_dtypes(include = [np.number])

correlation_matrix = numeric_df.corr()
salary_correlation = correlation_matrix['salary']

print("Correlation with Salary:")
print(salary_correlation)
```

```
Correlation with Salary:
salary    1.0
Name: salary, dtype: float64
```

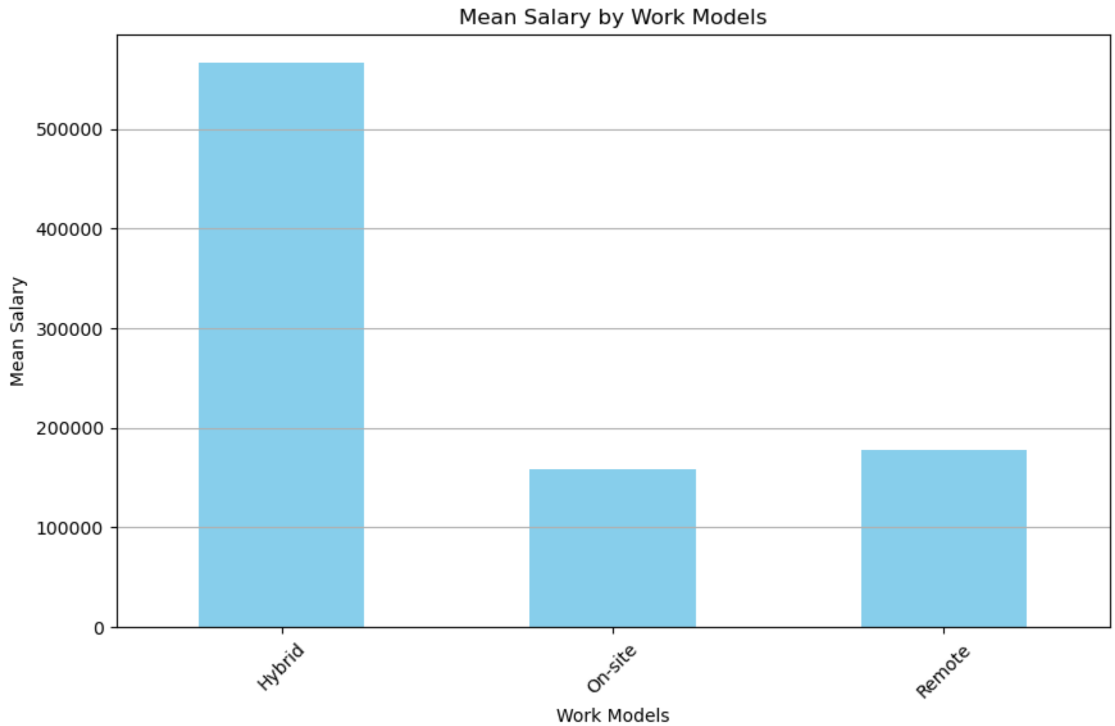
[61]:
```
df.plot.scatter(x = 'work_models',
                y = 'salary',
                title = 'Salary vs Work Models')

df.boxplot(column = 'salary', by = 'company_size', grid = False)
plt.title('Salary Distribution by Company Size')
plt.xlabel('Company Size')
plt.ylabel('Salary')
plt.suptitle('')
plt.show()
```

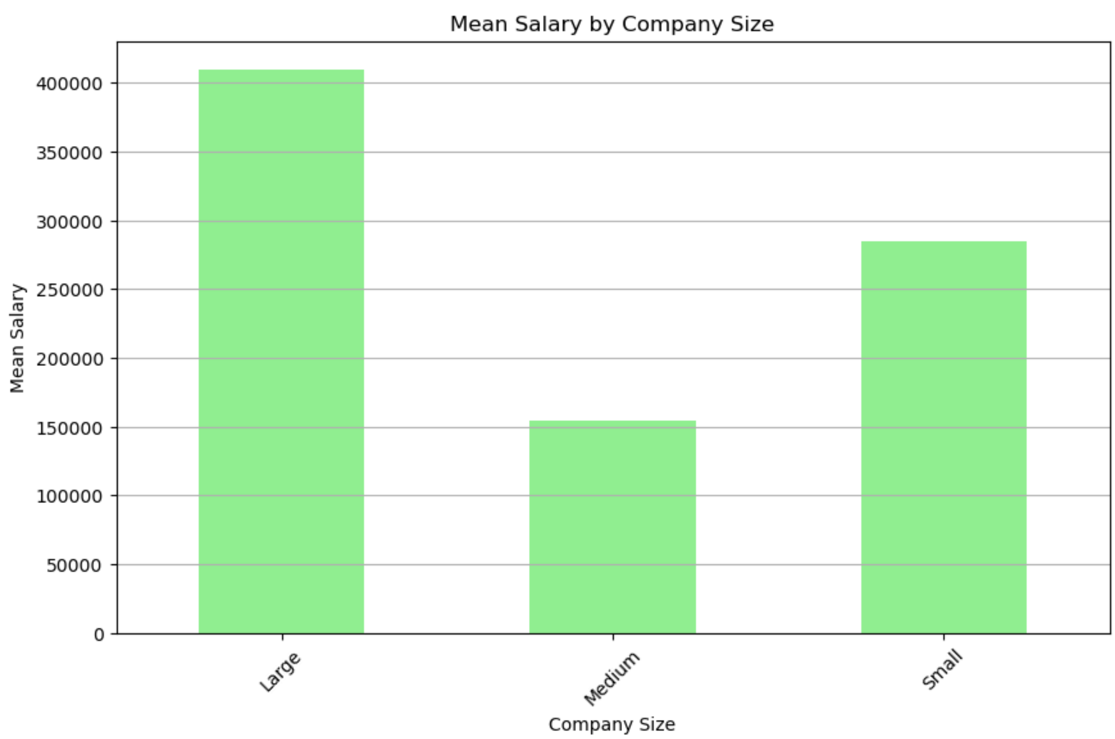




[62]:
```
mean_salary_by_work_models = df.groupby('work_models')['salary'].mean()

plt.figure(figsize = (10, 6))
mean_salary_by_work_models.plot(kind = 'bar', color = 'skyblue')
plt.title('Mean Salary by Work Models')
plt.xlabel('Work Models')
plt.ylabel('Mean Salary')
plt.xticks(rotation = 45)
plt.grid(axis = 'y')
plt.show()

mean_salary_by_company_size = df.groupby('company_size')['salary'].mean()

plt.figure(figsize = (10, 6))
mean_salary_by_company_size.plot(kind = 'bar', color = 'lightgreen')
plt.title('Mean Salary by Company Size')
plt.xlabel('Company Size')
plt.ylabel('Mean Salary')
plt.xticks(rotation = 45)
plt.grid(axis='y')
```

```
plt.show()
```

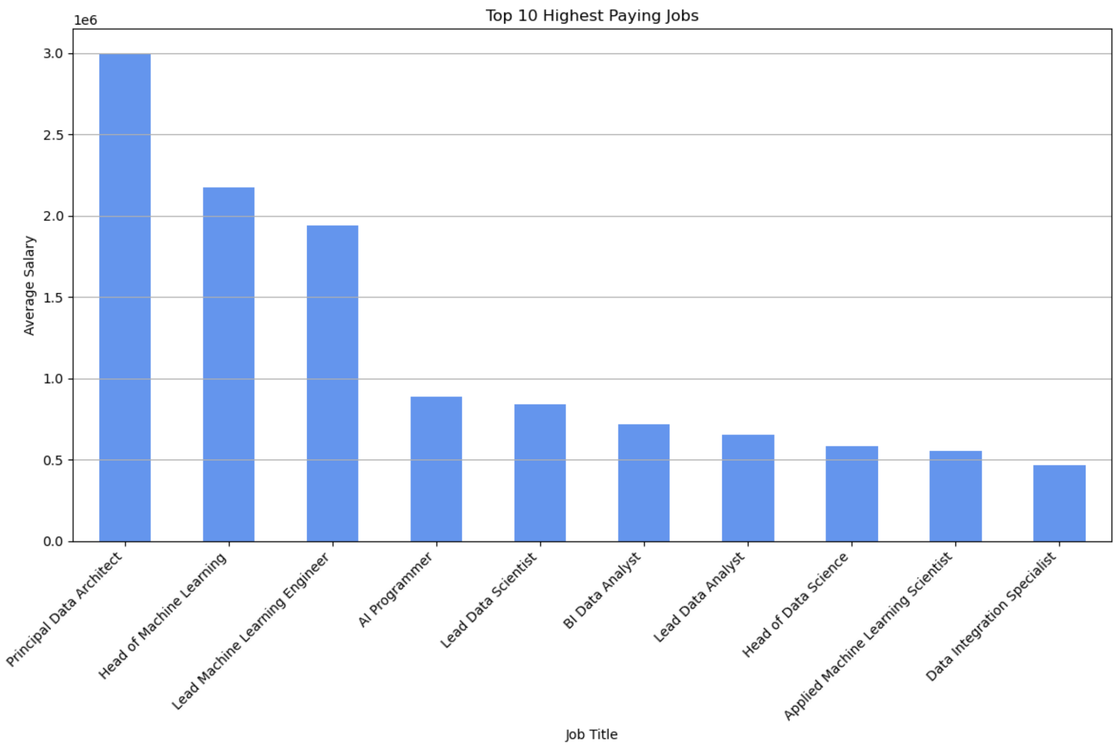

Mean Salary by Work Models
Mean Salary
500000
400000
300000
200000
100000
0
Hybrid
On-site
Remote
Work Models


Mean Salary by Company Size
Mean Salary
400000
350000
300000
250000
200000
150000
100000
50000
0
Large
Medium
Small
Company Size

[63]:

```python
mean_salary_by_job = df.groupby('job_title')['salary'].mean()

# Sort jobs by mean salary in descending order
sorted_salary_by_job = mean_salary_by_job.sort_values(ascending=False)

# Display the top N jobs with the highest salaries
top_n_jobs = sorted_salary_by_job.head(10)
print(top_n_jobs)

# Plot a bar chart for the top N jobs with the highest salaries
plt.figure(figsize = (12, 8))
top_n_jobs.plot(kind = 'bar', color = 'cornflowerblue')
plt.title('Top 10 Highest Paying Jobs')
plt.xlabel('Job Title')
plt.ylabel('Average Salary')
plt.xticks(rotation = 45, ha = 'right')
plt.grid(axis = 'y')

# Show plot
plt.tight_layout()
plt.show()
```

```
job_title
Principal Data Architect             3.000000e+06
Head of Machine Learning             2.172667e+06
Lead Machine Learning Engineer       1.940250e+06
AI Programmer                        8.868010e+05
Lead Data Scientist                  8.392368e+05
BI Data Analyst                      7.155882e+05
Lead Data Analyst                    6.550000e+05
Head of Data Science                 5.846304e+05
Applied Machine Learning Scientist   5.548429e+05
Data Integration Specialist          4.649562e+05
Name: salary, dtype: float64
```



Top 10 Highest Paying Jobs
1e6
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Average Salary
Principal Data Architect
Head of Machine Learning
Lead Machine Learning Engineer
AI Programmer
Lead Data Scientist
BI Data Analyst
Lead Data Analyst
Head of Data Science
Applied Machine Learning Scientist
Data Integration Specialist
Job Title