DataDough 💸

DataDough is a comprehensive analysis of publicly available data sourced from kaggle, conducted by Shreya Sudan. It encompasses a series of rigorous analytical procedures, including data cleansing, exploratory data analysis, data visualizations, hypothesis testing, and predictive analysis.

In [1]:
```
import pandas as pd
import numpy as np
import plotly.express as px
```

Introduction 🤓

Welcome to DataDough, a data science project focused on exploring the fascinating realm of data and uncovering insights related to salaries of data scientists. In today's data-driven world, understanding the factors that influence data scientists' salaries is crucial for both professionals and organizations alike. This project aims to shed light on the various factors that contribute to the salaries of data scientists and provide valuable insights into this ever-evolving field.

The dataset `salary` contains data on not only the salaries of data scientists, but also their experience, salary in USD, residence, company's location, and company's size.

This project not only seeks to answer questions about salary trends but also aims to explore the relationships between different variables and their impact on compensation. By examining the interplay between factors like work experience, employment type, and company size, we aim to uncover valuable insights that can aid professionals in negotiating salaries and assist organizations in formulating competitive compensation packages.

To accomplish these objectives, we employ a range of analytical techniques and tools, including data preprocessing, exploratory data analysis, feature engineering, and machine learning algorithms. The project utilizes popular libraries like pandas, scikit-learn, and plotly to perform data manipulation, modeling, and visualization.

Here are the components of our project:

- Part I : Inferential Analysis
  - Data Cleaning
  - Exploratory Data Analysis (EDA)
  - Hypothesis Testing
- Part II : Predictive Analysis
  - Linear Regression Model
    - Framing the Problem
    - Baseline Model
    - Final Model
  - Multiclass Classification
    - Random Forest Classifier
    - Decision Trees

By the end of DataDough, we aim to provide a comprehensive understanding of the salary landscape for data scientists, empowering both individuals and organizations with actionable insights. Whether you are a data scientist seeking to benchmark your salary or an organization looking to attract and retain top talent, this project strives to offer valuable information and assist in data-driven decision-making.

In [2]:
```
salary = pd.read_csv('ds_salaries.csv')
salary.head()
# print(salary.head().to_markdown(index=False))
```

Out[2]:

	work_year	experience_level	employment_type	job_title	salary	salary_currency	salary_in_usd	employee_residence	remote_ratio	company_location	company_size
0	2023	SE	FT	Principal Data Scientist	80000	EUR	85847	ES	100	ES	L
1	2023	MI	CT	ML Engineer	30000	USD	30000	US	100	US	S
2	2023	MI	CT	ML Engineer	25500	USD	25500	US	100	US	S
3	2023	SE	FT	Data Scientist	175000	USD	175000	CA	100	CA	M
4	2023	SE	FT	Data Scientist	120000	USD	120000	CA	100	CA	M

In [3]:
```
salary.describe()
# print(salary.describe().to_markdown())
```

Out[3]:

	work_year	salary	salary_in_usd	remote_ratio
count	3755.000000	3.755000e+03	3755.000000	3755.000000
mean	2022.373635	1.906956e+05	137570.389880	46.271638
std	0.691448	6.716765e+05	63055.625278	48.589050
min	2020.000000	6.000000e+03	5132.000000	0.000000
25%	2022.000000	1.000000e+05	95000.000000	0.000000
50%	2022.000000	1.380000e+05	135000.000000	0.000000
75%	2023.000000	1.800000e+05	175000.000000	100.000000
max	2023.000000	3.040000e+07	450000.000000	100.000000

Part I : Inferential Analysis 🔬🔬

Data Cleaning 🧹🧽

After looking at some preliminary statistics for `salary`, we decided to undertake the following steps to clean our data and prepare it for use:

1. `remote_ratio` column was divided by 100 to ensure scalability and correctness if the column were to be used as a numeric feature in a model.
2. `work_year` was converted to string type as it is a categorical feature here and the interpretation of the 'mean' of the `work_year` in the descriptive statistics is meaningless.

In [4]:
```
salary['remote_ratio'].unique()
```

Out[4]:
```
array([100,   0,  50])
```

In [5]:
```
salary['remote_ratio'] = salary['remote_ratio'] / 100
```

In [6]:
```
salary.head()
```

Out[6]:

	work_year	experience_level	employment_type	job_title	salary	salary_currency	salary_in_usd	employee_residence	remote_ratio	company_location	company_size
0	2023	SE	FT	Principal Data Scientist	80000	EUR	85847	ES	1.0	ES	L
1	2023	MI	CT	ML Engineer	30000	USD	30000	US	1.0	US	S
2	2023	MI	CT	ML Engineer	25500	USD	25500	US	1.0	US	S
3	2023	SE	FT	Data Scientist	175000	USD	175000	CA	1.0	CA	M
4	2023	SE	FT	Data Scientist	120000	USD	120000	CA	1.0	CA	M

In [7]:
```
salary.shape
```

Out[7]:
```
(3755, 11)
```

In [8]:
```
salary['work_year'].dtype
```

Out[8]:
```
dtype('int64')
```

In [9]:
```
salary['work_year'] = salary['work_year'].astype(str)
```

In [10]:
```
salary.head()
# print(salary.head().to_markdown(index=False))
```

Out[10]:

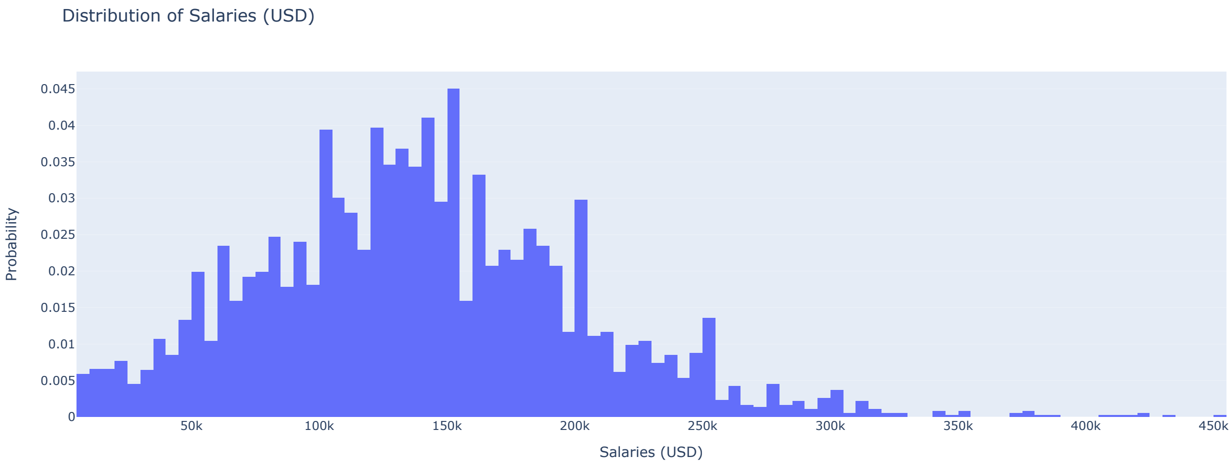
	work_year	experience_level	employment_type	job_title	salary	salary_currency	salary_in_usd	employee_residence	remote_ratio	company_location	company_size
0	2023	SE	FT	Principal Data Scientist	80000	EUR	85847	ES	1.0	ES	L
1	2023	MI	CT	ML Engineer	30000	USD	30000	US	1.0	US	S
2	2023	MI	CT	ML Engineer	25500	USD	25500	US	1.0	US	S
3	2023	SE	FT	Data Scientist	175000	USD	175000	CA	1.0	CA	M
4	2023	SE	FT	Data Scientist	120000	USD	120000	CA	1.0	CA	M

EDA 📊

As part of our EDA we explored the relationship and the distribution of the following variables:

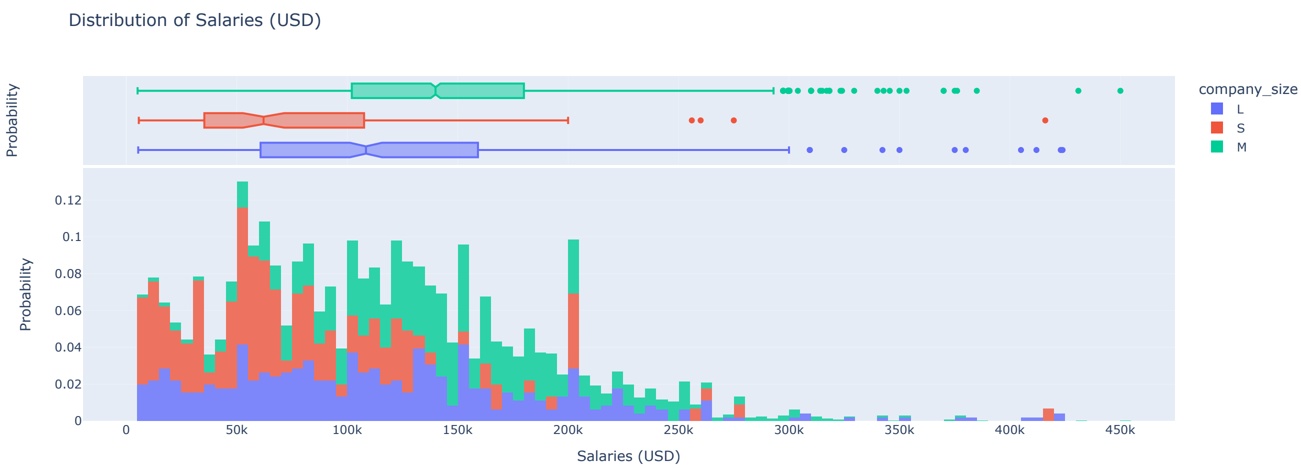
1. The Distribution of `salary_in_usd`
   - **Observations**: The distribution of salaries (USD) is fairly normal, and slightly right-skewed. It's a unimodal histogram with its peak at `$150,000-$155,000`
2. The Distribution of `salary_in_usd` with respect to `company_size`
   - **Observations**: We can derive from the overlaid histogram that the medium companies are generally taller than the smaller and the larger companies. This is because the dataset contains more data-points for medium-sized companies. The boxplots accompanying the histograms confirm that the medium sized companies have more outliers and a higher median than the other two groups.
3. The average `salary` and `salary_in_usd` as `work_year` increases
   - **Observations**: There is a steady increase in the mean `salary_in_usd` as the years increase. This could be explained by the usual rise in the general price level or inflation. However, the growth of the mean `salary` is quite peculiar. The `salary` column has a very high standard deviation, because of the disparities in the foreign ecxchange values of the different currencies. Thus, `salary_in_usd` provides a more standardized distribution of salaries.

In [11]:
```
fig = px.histogram(
    salary,
    x='salary_in_usd',
    histnorm='probability',
    title='Distribution of Salaries (USD)',
    labels={'salary_in_usd':'Salaries (USD)'},
#     hover_name='job_title',
#     hover_data=['salary_currency','salary'],
)
# fig.add_trace(marker_color='#EB89B5')
# fig.update_layout(hoverlabel=dict(bgcolor="lightgreen", font_size=12, font_family="Arial"))
fig.update_yaxes(title_text='Probability')
fig.show()
```



In [12]:
```
# fig.write_html('edaFirst.html', include_plotlyjs='cdn')
```

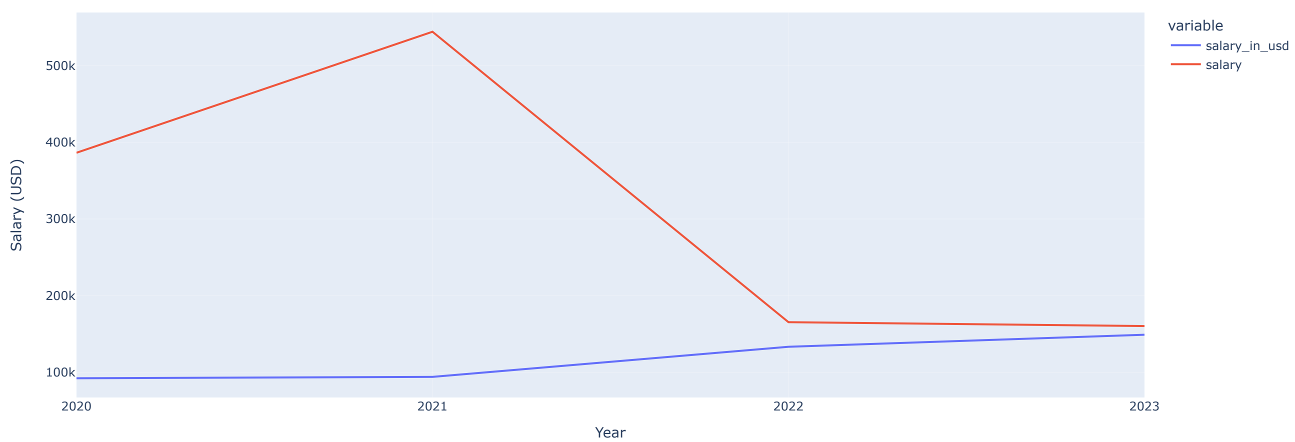
In [13]:
```
fig = px.histogram(
    salary,
    x='salary_in_usd',
    histnorm='probability',
    title='Distribution of Salaries (USD)',
    labels={'salary_in_usd':'Salaries (USD)'},
    color = 'company_size',
    opacity = 0.8,
    marginal="box"
)
fig.update_yaxes(title_text='Probability')
fig.show()
```



In [14]:
```
# fig.write_html('edaSecond.html', include_plotlyjs='cdn')
```

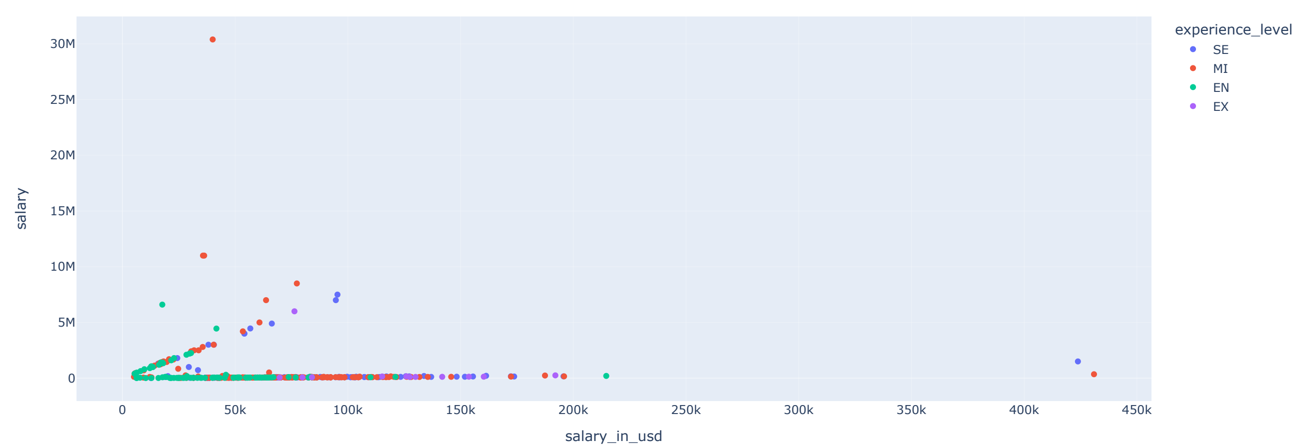
In [15]:
```
fig = px.line(salary.groupby('work_year')['salary_in_usd', 'salary'].mean(), y=['salary_in_usd', 'salary'])
fig.update_xaxes(title_text='Year')
fig.update_yaxes(title_text='Salary (USD)')
fig.show()
```

```
/var/folders/6g/cn7vfpqj1hq_5nknjxtfqhbh0000gn/T/ipykernel_61871/765794337.py:1: FutureWarning:

Indexing with multiple keys (implicitly converted to a tuple of keys) will be deprecated, use a list instead.
```




```
In [16]: # fig.write_html('edaThird.html', include_plotlyjs='cdn')
```

```
In [17]: fig = px.scatter(salary[salary['salary_currency'] != 'USD'], y='salary', x='salary_in_usd', color='experience_level', hover_name='salary_currency',hover_data=['employee
fig.show()
```



Hypothesis Testing 👩🏻‍🔬

Null Hypothesis: The distribution of salaries of different company sizes is drawn from the same population, and any differences in the distribution are purely conincidental.

Alternative Hypothesis: The differences in salaries are not purely coincidental

Test Statistic: Mean s

p-value: 0.0

Conclusion: It is highly unlikely that the difference in means are purely coincidental. Thus, we reject the null hypothesis

```
In [18]: observed_tvd = salary.groupby('company_size')['salary_in_usd'].mean().diff().iloc[-1]
observed_tvd
```

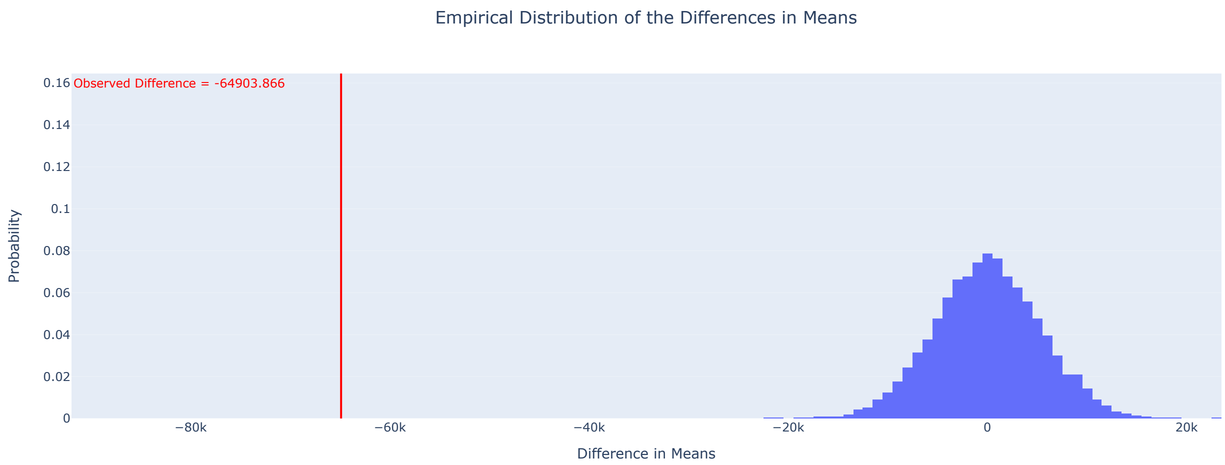
```
Out[18]: -64903.865934202535
```

```
In [19]: tvd = []
hyp = salary.copy()
for _ in range(10000):
    hyp['Shuffled'] = np.random.permutation(salary['company_size'])
    tvd.append(hyp.groupby('Shuffled')['salary_in_usd'].mean().diff().iloc[-1])
tvd[:10]
```

```
Out[19]: [-1041.6129812019353,
 1145.6211330264632,
 3888.3027575625165,
 -2132.826158699143,
 -4853.351319206937,
 -573.0251926522178,
 9952.11478128939,
 -1805.0297121574404,
 1434.0364603423513,
 2990.542471348599]
```

```
In [20]: fig = px.histogram(pd.DataFrame(tvd), x=0, nbins=50, histnorm='probability',
 title='Empirical Distribution of the Differences in Means')
fig.add_vline(x=observed_tvd, line_color='red')
fig.add_annotation(text=f'<span style="color:red">Observed Difference = {round(observed_tvd, 3)}</span>',
```

```
 x=1.25 * observed_tvd, showarrow=False, y=0.16)
fig.update_layout(xaxis_title='Difference in Means', yaxis_title='Probability', title_x=0.5)
```



In [21]:
```
# fig.write_html('hypothesis.html', include_plotlyjs='cdn')
```

In [22]:
```
pval = np.mean(np.array(tvd) <= observed_tvd)
pval
```

Out[22]: 0.0

Part II : Predictive Analysis 🔮🔮

Linear Regression 📈📈

Framing the Problem 🧱 🤔

Prediction Problem: Predict the salaries of data scientists in USD

Response Variable: `salary_in_usd`

Regressors: `work_year`, `employee_type`

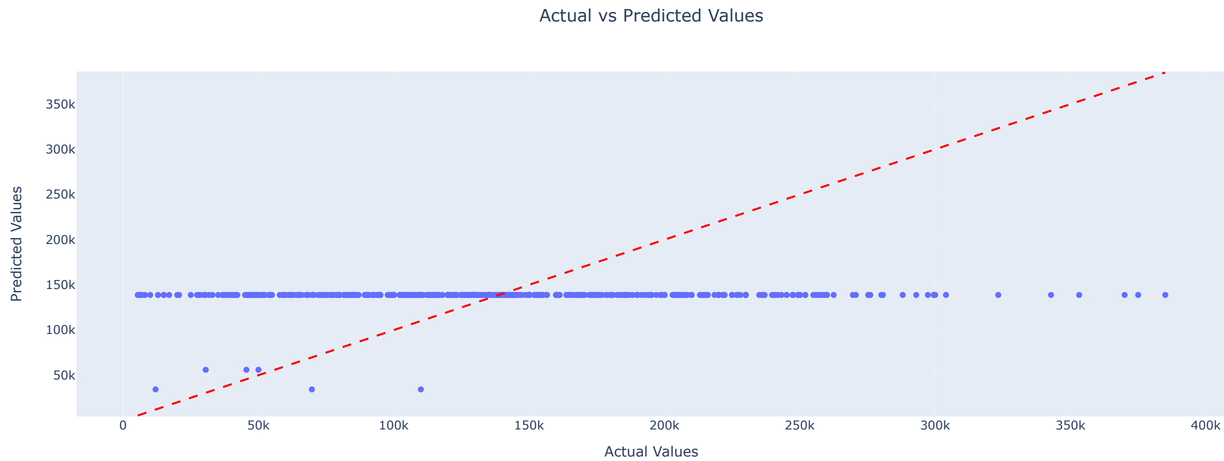
Evaluation Metric: R^2

In [23]:
```
def plot_predictions(y_true, y_pred):
 fig = px.scatter(x=y_true, y=y_pred, labels={'x': 'Actual Values', 'y': 'Predicted Values'},
 title='Actual vs Predicted Values')
 fig.add_shape(type='line', x0=min(y_true), y0=min(y_true), x1=max(y_true), y1=max(y_true),
 line=dict(color='red', dash='dash'))
 fig.update_layout(title_x=0.5)
 return fig
```

Baseline Model 🧩

In [24]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder, LabelEncoder, FunctionTransformer
from sklearn.linear_model import LinearRegression
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import r2_score
# salary['work_year'] = salary['work_year'].astype(int)
# Prepare the data
X = salary[['work_year', 'employment_type']]
y = salary['salary_in_usd']

# Define the encoding steps
encoder = ColumnTransformer([
    ('onehot', OneHotEncoder(), ['employment_type']),
#      ('label', LabelEncoder(), ['work_year'])
])
# Define the pipeline
pipeline = Pipeline([
    ('encoder', encoder),
    ('model', LinearRegression())
])

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Fit the pipeline (including encoding and model training) to the training data
pipeline.fit(X_train, y_train)

# Make predictions on the test set
y_pred = pipeline.predict(X_test)

# Evaluate the model
r2 = r2_score(y_test, y_pred)
print("R² score: {:.2f}".format(r2))
fig = plot_predictions(y_test, y_pred)
fig.show()
```

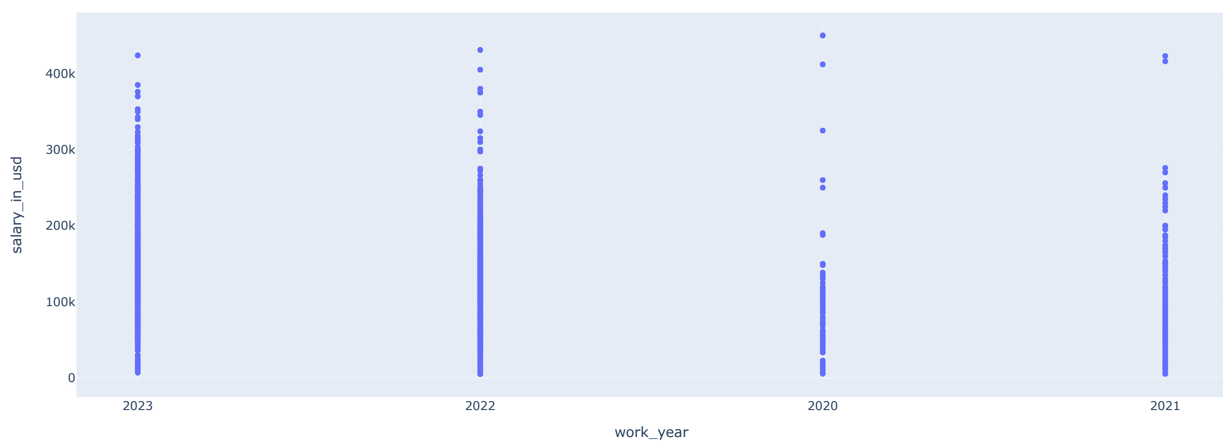
```
R² score: 0.01
```



In [25]:
```
# fig.write_html('baseline.html', include_plotlyjs='cdn')
```

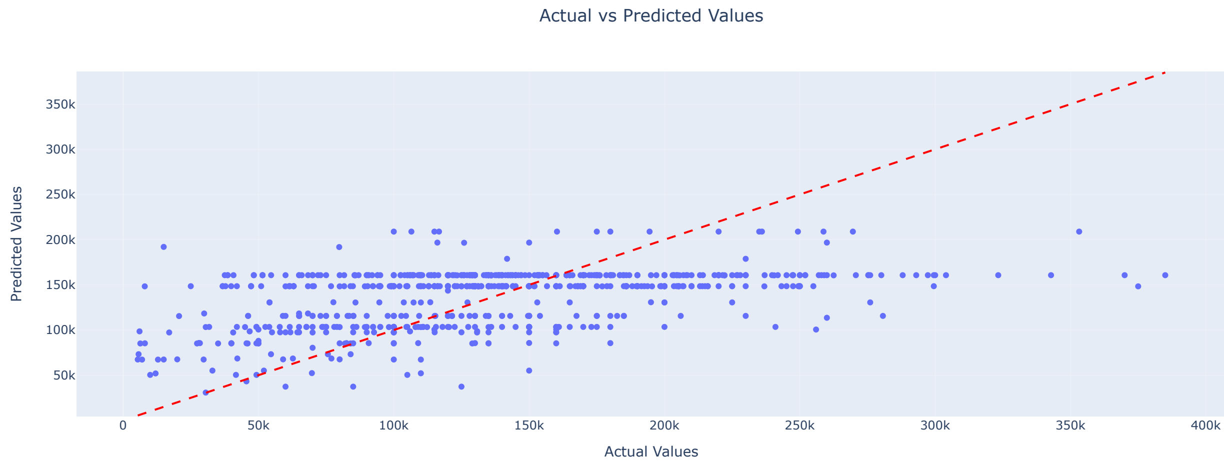
Final Model 🏆

In [26]:
```
fig = px.scatter(salary, x='work_year', y='salary_in_usd')
fig.show()
```



In [27]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder, LabelEncoder, FunctionTransformer
from sklearn.linear_model import LinearRegression
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import r2_score
# salary['work_year'] = salary['work_year'].astype(int)
# Prepare the data
X = salary[['work_year', 'employment_type','company_size', 'experience_level']]
y = salary['salary_in_usd']

# Define the encoding steps
encoder = ColumnTransformer([
    ('onehot', OneHotEncoder(), ['employment_type', 'work_year', 'company_size', 'experience_level']),
#      ('label', LabelEncoder(), ['work_year'])
])
# Define the pipeline
pipeline = Pipeline([
    ('encoder', encoder),
    ('model', LinearRegression())
])

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Fit the pipeline (including encoding and model training) to the training data
pipeline.fit(X_train, y_train)

# Make predictions on the test set
y_pred = pipeline.predict(X_test)

# Evaluate the model
r2 = r2_score(y_test, y_pred)
print("R² score: {:.2f}".format(r2))
# plot_predictions(y_test, y_pred)
fig = plot_predictions(y_test, y_pred)
fig.show()
```

```
R² score: 0.20
```

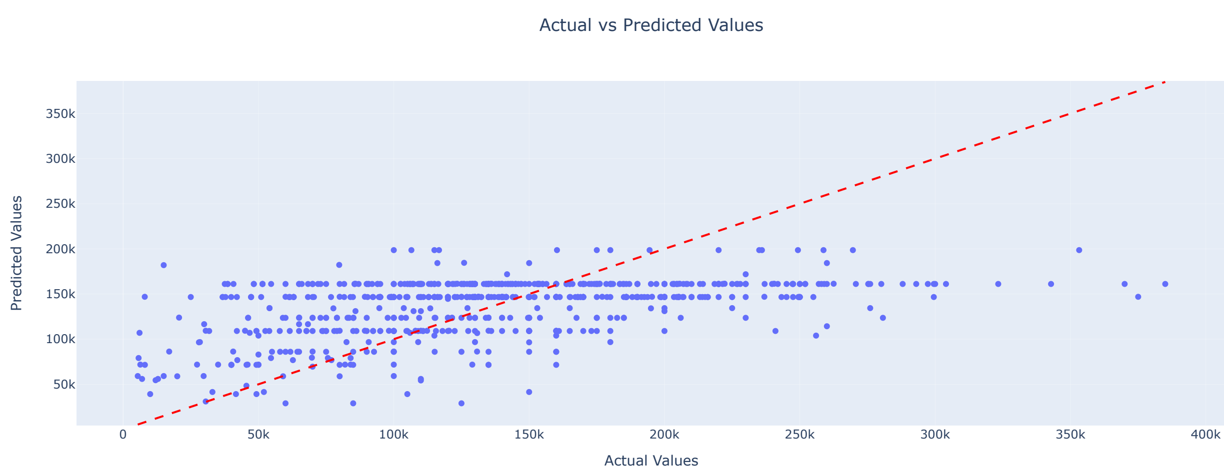


In [28]:
```
salary['experience_level'].unique()
```

Out[28]:
```
array(['SE', 'MI', 'EN', 'EX'], dtype=object)
```

In [29]:
```
from sklearn.preprocessing import OneHotEncoder, OrdinalEncoder
from sklearn.linear_model import LinearRegression
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split
X = salary[['work_year', 'employment_type', 'experience_level', 'company_size']]
y = salary['salary_in_usd']

# Define the encoding steps
encoder = ColumnTransformer([
    ('onehot', OneHotEncoder(), ['employment_type', 'work_year', 'company_size']),
    ('ordinal', OrdinalEncoder(categories=[['EN', 'MI', 'SE', 'EX']]), ['experience_level'])
])

# Define the pipeline
pipeline = Pipeline([
    ('encoder', encoder),
    ('model', LinearRegression())
])

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Fit the pipeline (including encoding and model training) to the training data
pipeline.fit(X_train, y_train)

# Make predictions on the test set
y_pred = pipeline.predict(X_test)

# Evaluate the model
r2 = r2_score(y_test, y_pred)
print("R² score: {:.2f}".format(r2))
fig = plot_predictions(y_test, y_pred)
fig.show()
```

```
R² score: 0.20
```

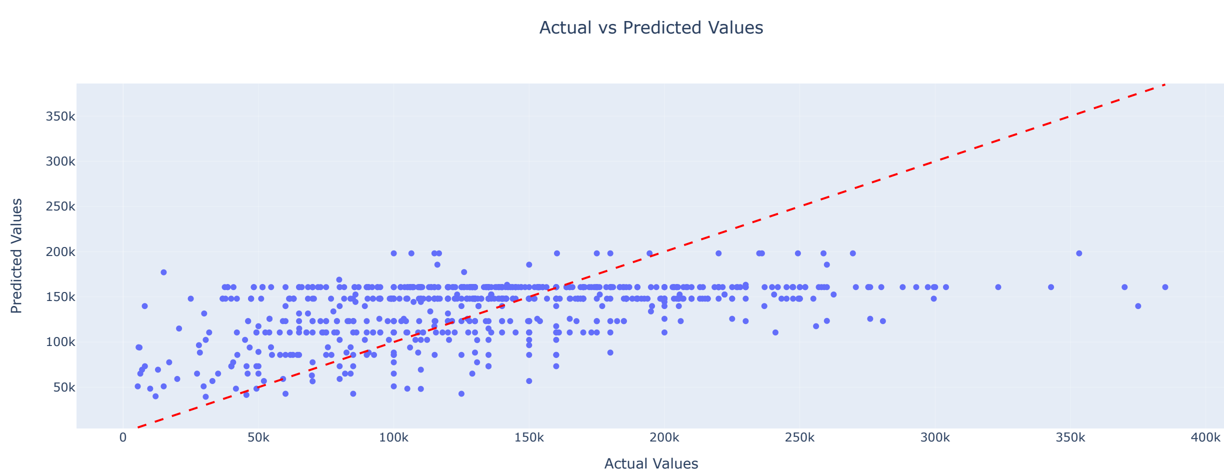


In [30]:
```
fig = px.histogram(salary, x="salary_in_usd", color="experience_level",
                   marginal="box", # or violin, rug
                   opacity = 0.8
                  )
fig.show()
```

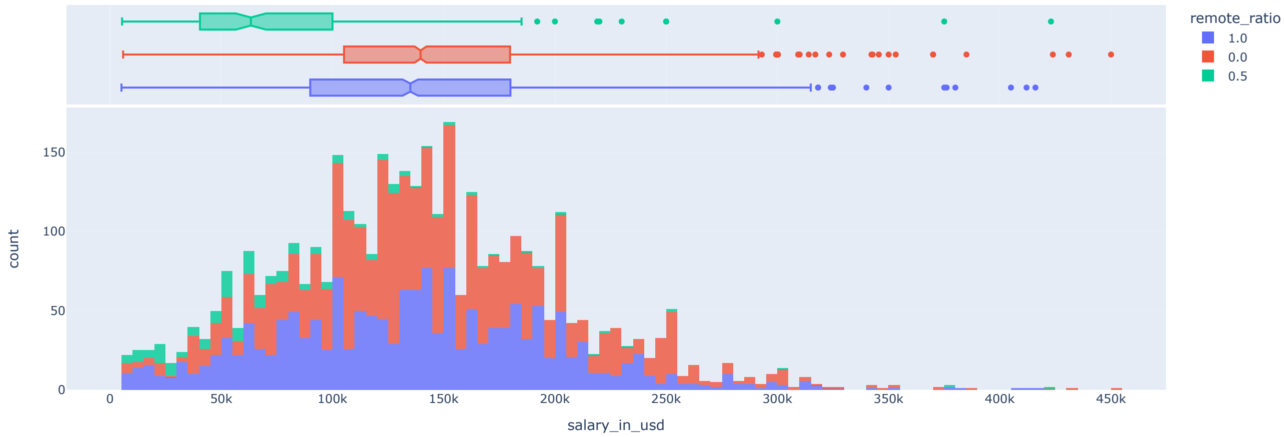
In [31]:
```
# fig.write_html('exp_lvl.html', include_plotlyjs='cdn')
```

In [32]:
```
from sklearn.preprocessing import OneHotEncoder, OrdinalEncoder
from sklearn.linear_model import LinearRegression
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split
X = salary[['work_year', 'employment_type', 'experience_level', 'company_size']]
y = salary['salary_in_usd']

# Define the encoding steps
encoder = ColumnTransformer([
    ('onehot', OneHotEncoder(), ['employment_type', 'work_year']),
    ('ordinal', OrdinalEncoder(categories=[['EN', 'MI', 'SE', 'EX'], ['S', 'L', 'M']]), ['experience_level', 'company_size'])
])

# Define the pipeline
pipeline = Pipeline([
    ('encoder', encoder),
    ('model', LinearRegression())
])

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Fit the pipeline (including encoding and model training) to the training data
pipeline.fit(X_train, y_train)

# Make predictions on the test set
y_pred = pipeline.predict(X_test)

# Evaluate the model
r2 = r2_score(y_test, y_pred)
print("R² score: {:.2f}".format(r2))
fig = plot_predictions(y_test, y_pred)
fig.show()
```

```
R² score: 0.20
```



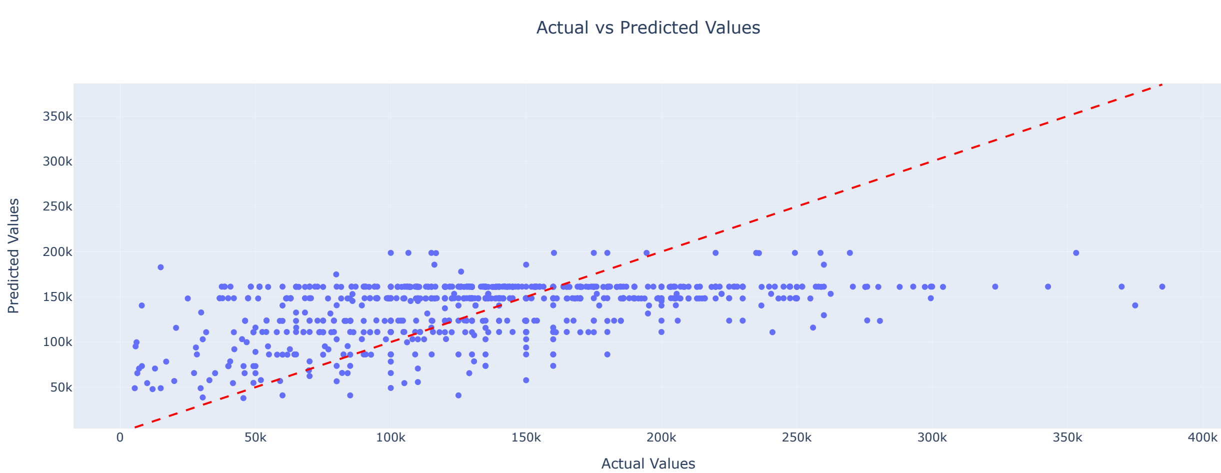
In [33]:
```
fig = px.histogram(salary, x="salary_in_usd", color="remote_ratio",
                   marginal="box", # or violin, rug
                   opacity = 0.8
                  )
fig.show()
```



In [34]:
```
# fig.write_html('ratio.html', include_plotlyjs='cdn')
```

In [35]:
```
from sklearn.preprocessing import OneHotEncoder, OrdinalEncoder
from sklearn.linear_model import LinearRegression
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split
X = salary[['work_year', 'employment_type', 'experience_level', 'company_size', 'remote_ratio']]
y = salary['salary_in_usd']

# Define the encoding steps
encoder = ColumnTransformer([
    ('onehot', OneHotEncoder(), ['employment_type', 'work_year']),
    ('ordinal', OrdinalEncoder(categories=[['EN', 'MI', 'SE', 'EX'], ['S', 'L', 'M']]), ['experience_level', 'company_size'])
], remainder='passthrough')

# Define the pipeline
pipeline = Pipeline([
    ('encoder', encoder),
    ('model', LinearRegression())
])

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Fit the pipeline (including encoding and model training) to the training data
pipeline.fit(X_train, y_train)

# Make predictions on the test set
y_pred = pipeline.predict(X_test)

# Evaluate the model
r2 = r2_score(y_test, y_pred)
print("R² score: {:.2f}".format(r2))
fig = plot_predictions(y_test, y_pred)
fig.show()
```

```
R² score: 0.20
```



In [36]:
```
# fig.write_html('final_mdl.html', include_plotlyjs='cdn')
```

Multiclass Classification 📋

Framing the Problem 🧱 🤔

Prediction Problem: Predict the experience level of data scientists

Response Variable: `experience_level`

Features: `employee_type`, `company_size`

Evaluation Metric: Accuracy

Random Forest Classifier 🌴🌴

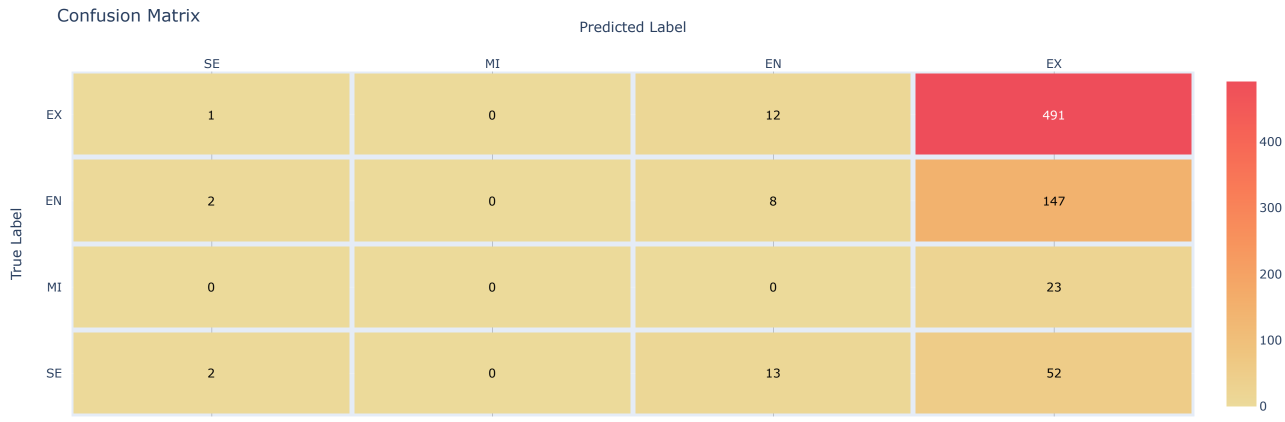
In [37]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score

# Prepare the data
X = salary[['employment_type', 'company_size']]
y = salary['experience_level']

# Perform one-hot encoding on the categorical features
X_encoded = pd.get_dummies(X)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_encoded, y, test_size=0.2, random_state=42)

# Create and train the random forest classifier
model = RandomForestClassifier()
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy: {:.2f}".format(accuracy))
precision = precision_score(y_test, y_pred, average=None)

# Display precision for each class
for class_label, class_precision in zip(model.classes_, precision):
    print("Precision for class {}: {:.2f}".format(class_label, class_precision))
precision
```

```
Accuracy: 0.67
Precision for class EN: 0.40
Precision for class EX: 0.00
Precision for class MI: 0.24
Precision for class SE: 0.69
```

```
/Users/shreya_sudan/anaconda3/envs/projects2023/lib/python3.8/site-packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning:

Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
```

Out[37]:
```
array([0.4       , 0.        , 0.24242424, 0.68863955])
```

In [38]:
```
from sklearn.metrics import confusion_matrix
import plotly.graph_objects as go
import plotly.figure_factory as ff
# Assuming you have already trained your classification model and made predictions
y_true = y_test  # True labels from the test set
y_pred = model.predict(X_test)  # Predicted labels

# Create a confusion matrix
cm = confusion_matrix(y_true, y_pred)

# Define the labels for the confusion matrix
labels = list(set(y_true))

# Create the confusion matrix figure using Plotly
fig = ff.create_annotated_heatmap(
    z=cm,
    x=labels,
    y=labels,
    colorscale='oryel',
    showscale=True,
    xgap=5,
    ygap=5
)

# Update the layout
fig.update_layout(
    title="Confusion Matrix",
    xaxis_title="Predicted Label",
    yaxis_title="True Label"
)

# Display the confusion matrix
fig.show()
```



In [39]:
```
# fig.write_html('randomforest.html', include_plotlyjs='cdn')
```

Decision Trees 🌳🌳

In [41]:
```
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import precision_score

# Prepare the data
X = salary[['employment_type', 'company_size']]
y = salary['experience_level']

# Perform one-hot encoding on the categorical features
X_encoded = pd.get_dummies(X)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_encoded, y, test_size=0.2, random_state=42)

# Define the parameter grid for hyperparameter search
param_grid = {'max_depth': [3, 5, 7, 9]}

# Create the decision tree classifier
tree = DecisionTreeClassifier()

# Perform grid search with cross-validation
grid_search = GridSearchCV(tree, param_grid, cv=5)
grid_search.fit(X_train, y_train)

# Retrieve the best model
best_tree = grid_search.best_estimator_

# Make predictions on the test set using the best model
y_pred = best_tree.predict(X_test)

# Calculate precision for each class
precision = precision_score(y_test, y_pred, average=None)

# Display precision for each class
for class_label, class_precision in zip(best_tree.classes_, precision):
    print("Precision for class {}: {:.2f}".format(class_label, class_precision))

# Print the best hyperparameters
print("Best hyperparameters:", grid_search.best_params_)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy: {:.2f}".format(accuracy))
```

```
Precision for class EN: 0.40
Precision for class EX: 0.00
Precision for class MI: 0.24
Precision for class SE: 0.69
Best hyperparameters: {'max_depth': 5}
Accuracy: 0.67
```

```
/Users/shreya_sudan/anaconda3/envs/projects2023/lib/python3.8/site-packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning:

Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
```

In [46]:
```
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import precision_score, recall_score

# Prepare the data
X = salary[['employment_type', 'company_size', 'salary_in_usd', 'job_title']]
y = salary['experience_level']

# Perform one-hot encoding on the categorical features
X_encoded = pd.get_dummies(X)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_encoded, y, test_size=0.2, random_state=42)

# Define the parameter grid for hyperparameter search
param_grid = {'max_depth': [3, 5, 7, 9]}

# Create the decision tree classifier
tree = DecisionTreeClassifier()
```

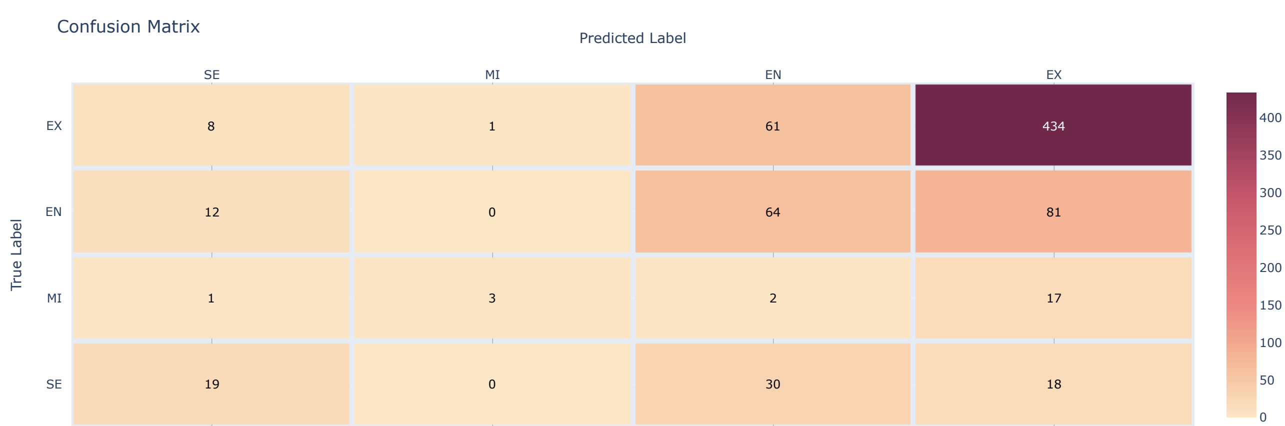
```
# Perform grid search with cross-validation
grid_search = GridSearchCV(tree, param_grid, cv=5)
grid_search.fit(X_train, y_train)

# Retrieve the best model
best_tree = grid_search.best_estimator_

# Make predictions on the test set using the best model
y_pred = best_tree.predict(X_test)

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

accuracy = accuracy_score(y_test, y_pred)
print("Accuracy: {:.2f}".format(accuracy))
```

```
Best hyperparameters: {'max_depth': 9}
Accuracy: 0.69
```

In [47]:
```
precision = precision_score(y_test, y_pred, average=None)

# Display precision for each class
for class_label, class_precision in zip(best_tree.classes_, precision):
    print("Precision for class {}: {:.2f}".format(class_label, class_precision))
recall = recall_score(y_test, y_pred, average=None)
# Display precision for each class
for class_label, class_precision in zip(best_tree.classes_, recall):
    print("Recall for class {}: {:.2f}".format(class_label, class_precision))
```

```
Precision for class EN: 0.47
Precision for class EX: 0.75
Precision for class MI: 0.41
Precision for class SE: 0.79
Recall for class EN: 0.28
Recall for class EX: 0.13
Recall for class MI: 0.41
Recall for class SE: 0.86
```

In [48]:
```
import plotly.figure_factory as ff
from sklearn.metrics import confusion_matrix

# Assuming you have already trained your classification model and made predictions
y_true = y_test  # True labels from the test set
y_pred = best_tree.predict(X_test)  # Predicted labels

# Create a confusion matrix
cm = confusion_matrix(y_true, y_pred)

# Define the labels for the confusion matrix
labels = list(set(y_true))

# Create the confusion matrix figure using Plotly
fig = ff.create_annotated_heatmap(
    z=cm,
    x=labels,
    y=labels,
    colorscale='burgyl',
    showscale=True,
    xgap=5,
    ygap=5
)

# Update the layout
fig.update_layout(
    title="Confusion Matrix",
    xaxis_title="Predicted Label",
    yaxis_title="True Label"
)

# Display the confusion matrix
fig.show()
#burgyl, deep, magenta, matter, mint, oryel, peach, pinkyl, purp, purpor, redor, sunset?, sunsetdark, teal and tealgrn???,
#
```



In [49]:
```
# fig.write_html('decisiontree.html', include_plotlyjs='cdn')
# ['aggrnyl', 'agsunset', 'algae', 'amp', 'armyrose', 'balance',
#              'blackbody', 'bluered', 'blues', 'blugrn', 'bluyl', 'brbg',
#              'brwnyl', 'bugn', 'bupu', 'burg', 'burgyl', 'cividis', 'curl',
#              'darkmint', 'deep', 'delta', 'dense', 'earth', 'edge', 'electric',
#              'emrld', 'fall', 'geyser', 'gnbu', 'gray', 'greens', 'greys',
#              'haline', 'hot', 'hsv', 'ice', 'icefire', 'inferno', 'jet',
```

```
#              'magenta', 'magma', 'matter', 'mint', 'mrybm', 'mygbm', 'oranges',
#              'orrd', 'oryel', 'oxy', 'peach', 'phase', 'picnic', 'pinkyl',
#              'piyg', 'plasma', 'plotly3', 'portland', 'prgn', 'pubu', 'pubugn',
#              'puor', 'purd', 'purp', 'purples', 'purpor', 'rainbow', 'rdbu',
#              'rdgy', 'rdpu', 'rdylbu', 'rdylgn', 'redor', 'reds', 'solar',
#              'spectral', 'speed', 'sunset', 'sunsetdark', 'teal', 'tealgrn',
#              'tealrose', 'tempo', 'temps', 'thermal', 'tropic', 'turbid',
#              'turbo', 'twilight', 'viridis', 'ylgn', 'ylgnbu', 'ylorbr',
#              'ylorrd']
```

In [50]:

```
from imblearn.over_sampling import SMOTE
from sklearn.preprocessing import StandardScaler
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score

# Prepare the data
X = salary[['employment_type', 'company_size']]
y = salary['experience_level']

# Perform one-hot encoding on the categorical features
X_encoded = pd.get_dummies(X)

smote = SMOTE()
X_resampled, y_resampled = smote.fit_resample(X_encoded, y)

# Scale the features using StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_resampled)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_resampled, test_size=0.2, random_state=42)

# Create and train the random forest classifier
model = RandomForestClassifier()
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy: {:.2f}".format(accuracy))
```

```
Accuracy: 0.36
```

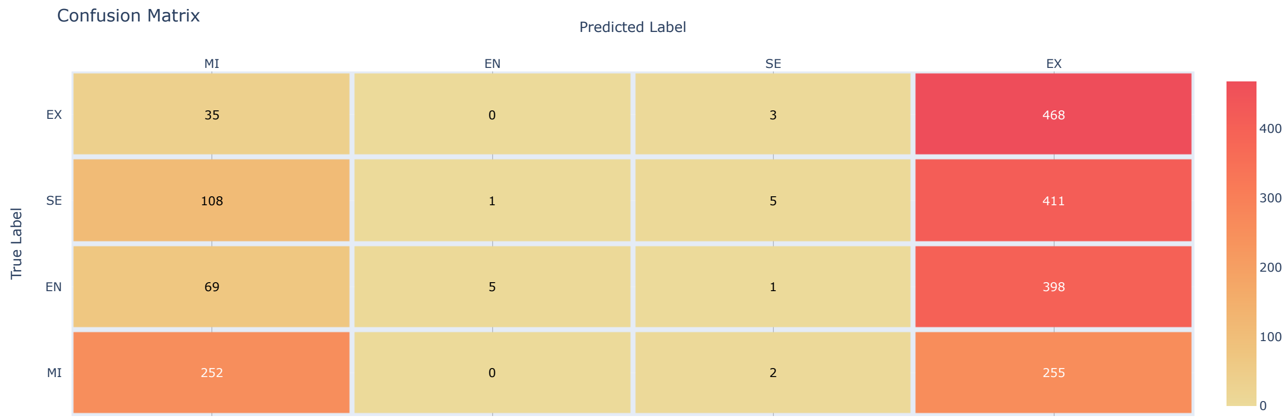
In [51]:

```
precision = precision_score(y_test, y_pred, average=None)

# Display precision for each class
for class_label, class_precision in zip(best_tree.classes_, precision):
    print("Precision for class {}: {:.2f}".format(class_label, class_precision))
recall = recall_score(y_test, y_pred, average=None)

# Display precision for each class
for class_label, class_precision in zip(model.classes_, recall):
    print("Recall for class {}: {:.2f}".format(class_label, class_precision))
```

```
Precision for class EN: 0.54
Precision for class EX: 0.83
Precision for class MI: 0.45
Precision for class SE: 0.31
Recall for class EN: 0.50
Recall for class EX: 0.01
Recall for class MI: 0.01
Recall for class SE: 0.92
```

In [52]:

```
from sklearn.metrics import confusion_matrix
import plotly.graph_objects as go
import plotly.figure_factory as ff
# Assuming you have already trained your classification model and made predictions
y_true = y_test  # True labels from the test set
y_pred = model.predict(X_test)  # Predicted labels

# Create a confusion matrix
cm = confusion_matrix(y_true, y_pred)

# Define the labels for the confusion matrix
labels = list(set(y_true))

# Create the confusion matrix figure using Plotly
fig = ff.create_annotated_heatmap(
    z=cm,
    x=labels,
    y=labels,
    colorscale='oryel',
    showscale=True,
    xgap=5,
    ygap=5
)

# Update the layout
fig.update_layout(
    title="Confusion Matrix",
    xaxis_title="Predicted Label",
    yaxis_title="True Label"
)

# Display the confusion matrix
fig.show()
```

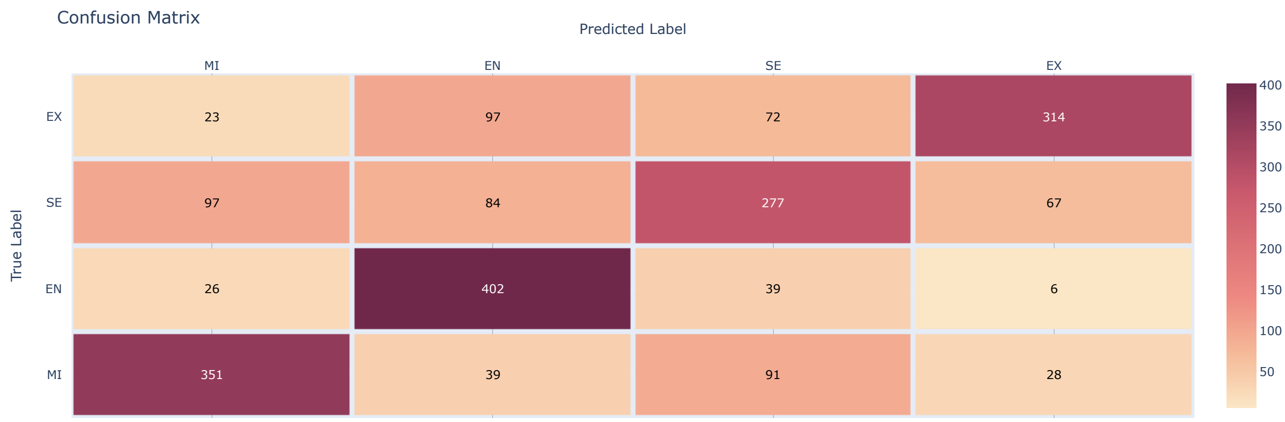


In [56]:
```python
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import precision_score

# Prepare the data
X = salary[['employment_type', 'company_size', 'salary_in_usd', 'job_title']]
y = salary['experience_level']

# Perform one-hot encoding on the categorical features
X_encoded = pd.get_dummies(X)

smote = SMOTE()
X_resampled, y_resampled = smote.fit_resample(X_encoded, y)

# Scale the features using StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_resampled)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_resampled, test_size=0.2, random_state=42)

# Define the parameter grid for hyperparameter search
param_grid = {'max_depth': [3, 5, 7, 9]}

# Create the decision tree classifier
tree = DecisionTreeClassifier()

# Perform grid search with cross-validation
grid_search = GridSearchCV(tree, param_grid, cv=5)
grid_search.fit(X_train, y_train)

# Retrieve the best model
best_tree = grid_search.best_estimator_

# Make predictions on the test set using the best model
y_pred = best_tree.predict(X_test)

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

accuracy = accuracy_score(y_test, y_pred)
print("Accuracy: {:.2f}".format(accuracy))
```

```
Best hyperparameters: {'max_depth': 9}
Accuracy: 0.67
```

In [57]:
```python
import plotly.figure_factory as ff
from sklearn.metrics import confusion_matrix

# Assuming you have already trained your classification model and made predictions
y_true = y_test  # True labels from the test set
y_pred = best_tree.predict(X_test)  # Predicted labels

# Create a confusion matrix
cm = confusion_matrix(y_true, y_pred)

# Define the labels for the confusion matrix
labels = list(set(y_true))

# Create the confusion matrix figure using Plotly
fig = ff.create_annotated_heatmap(
    z=cm,
    x=labels,
    y=labels,
    colorscale='burgyl',
    showscale=True,
    xgap=5,
    ygap=5
)

# Update the layout
fig.update_layout(
    title="Confusion Matrix",
    xaxis_title="Predicted Label",
    yaxis_title="True Label"
)

# Display the confusion matrix
fig.show()
```

```
#burgyl, deep, magenta, matter, mint, oryel, peach, pinkyl, purp, purpor, redor, sunset?, sunsetdark, teal and tealgrn???,
#
```



In [58]:
```
precision = precision_score(y_test, y_pred, average=None)

# Display precision for each class
for class_label, class_precision in zip(best_tree.classes_, precision):
    print("Precision for class {}: {:.2f}".format(class_label, class_precision))
recall = recall_score(y_test, y_pred, average=None)

# Display precision for each class
for class_label, class_precision in zip(best_tree.classes_, recall):
    print("Recall for class {}: {:.2f}".format(class_label, class_precision))
```

```
Precision for class EN: 0.71
Precision for class EX: 0.65
Precision for class MI: 0.58
Precision for class SE: 0.76
Recall for class EN: 0.69
Recall for class EX: 0.85
Recall for class MI: 0.53
Recall for class SE: 0.62
```

In [59]:
```
# fig.write_html('final_classification.html', include_plotlyjs='cdn')
```

Conclusion 🏁🏁