In [54]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
sns.set_theme(color_codes = True)
import numpy.ma as ma
```

In [55]:
```
df = pd.read_csv("employee_data.csv")
df
```

Out[55]:

	Salary	Job Satisfaction	Years of Experience	Employment Status
0	56.217808	2.666196	7	stay
1	96.550001	5.877109	3	stay
2	81.239576	8.856513	14	stay
3	71.906094	7.590024	10	stay
4	40.921305	8.259050	16	stay
...	...	...	...	...
995	36.410745	6.912596	19	leave
996	94.211950	9.609532	1	stay
997	39.577304	1.620622	7	leave
998	96.516615	1.513492	17	stay
999	61.220404	3.539684	3	stay

1000 rows × 4 columns

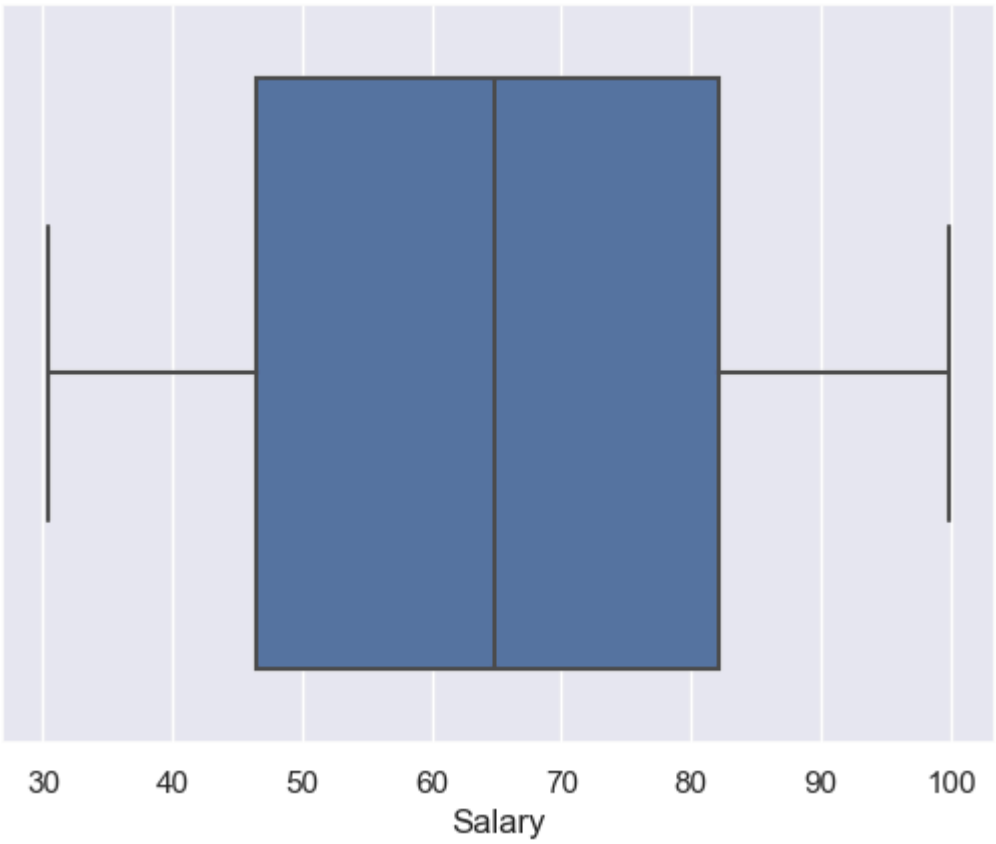
In [56]:
```
df.isnull().sum()
```

Out[56]:
```
Salary                0
Job Satisfaction      0
Years of Experience   0
Employment Status     0
dtype: int64
```

In [57]:
```
sns.boxplot(x=df["Salary"])
```

```
C:\Users\sjkar\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use isin
stance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
```

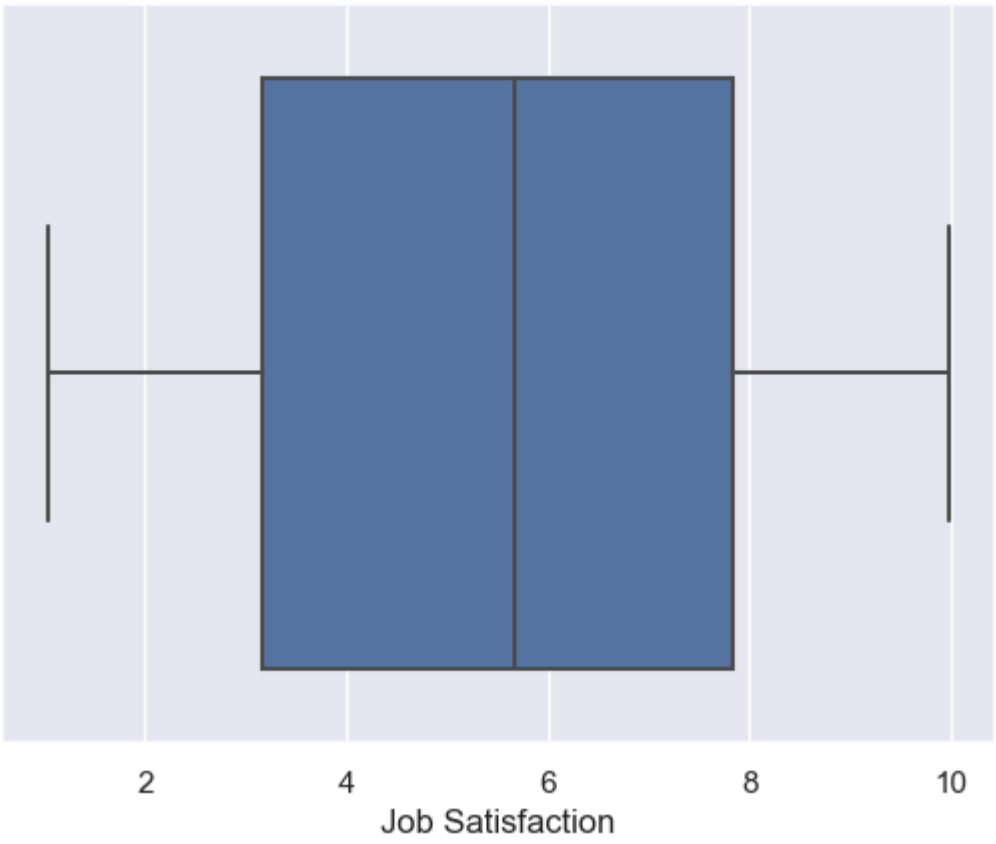
Out[57]:
```
<Axes: xlabel='Salary'>
```



In [58]:
```
sns.boxplot(x=df["Job Satisfaction"])
```

```
C:\Users\sjkar\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use isin
stance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
```

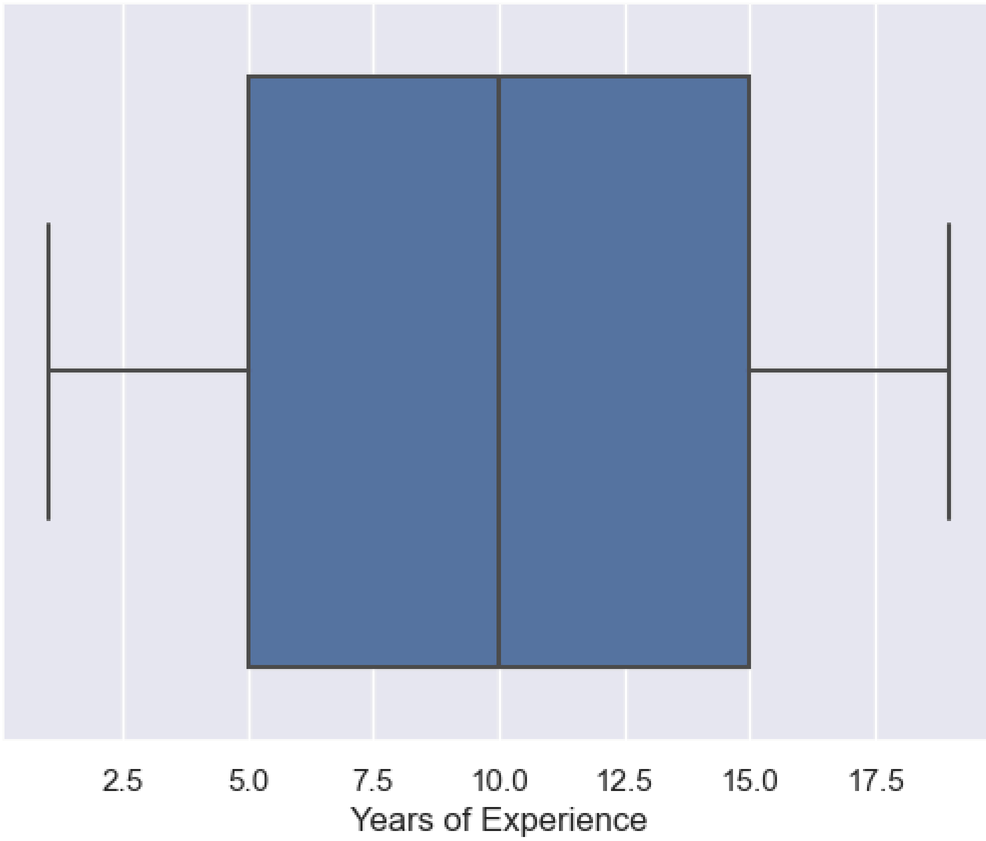
Out[58]:
```
<Axes: xlabel='Job Satisfaction'>
```



In [59]:
```
sns.boxplot(x=df["Years of Experience"])
```

```
C:\Users\sjkar\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use isin
stance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
```

Out[59]:
```
<Axes: xlabel='Years of Experience'>
```

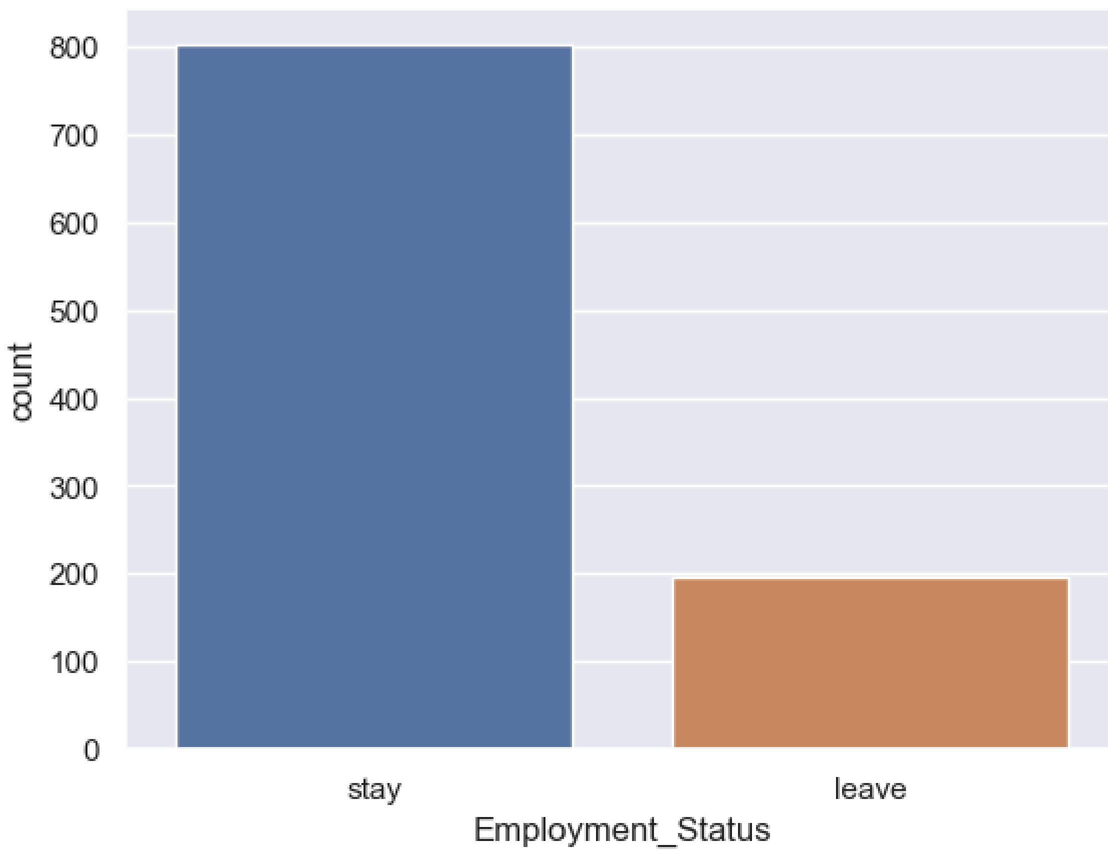


In [60]:
```
df.rename(columns={'Employment Status': 'Employment_Status'}, inplace=True)
```

In [61]:
```
sns.countplot(data=df, x='Employment_Status')
print(df.Employment_Status.value_counts())
```

```
Employment_Status
stay     803
leave    197
Name: count, dtype: int64
```

```
C:\Users\sjkar\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use isin
stance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\sjkar\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use isin
stance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\sjkar\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use isin
stance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
```



Upsampling the minority

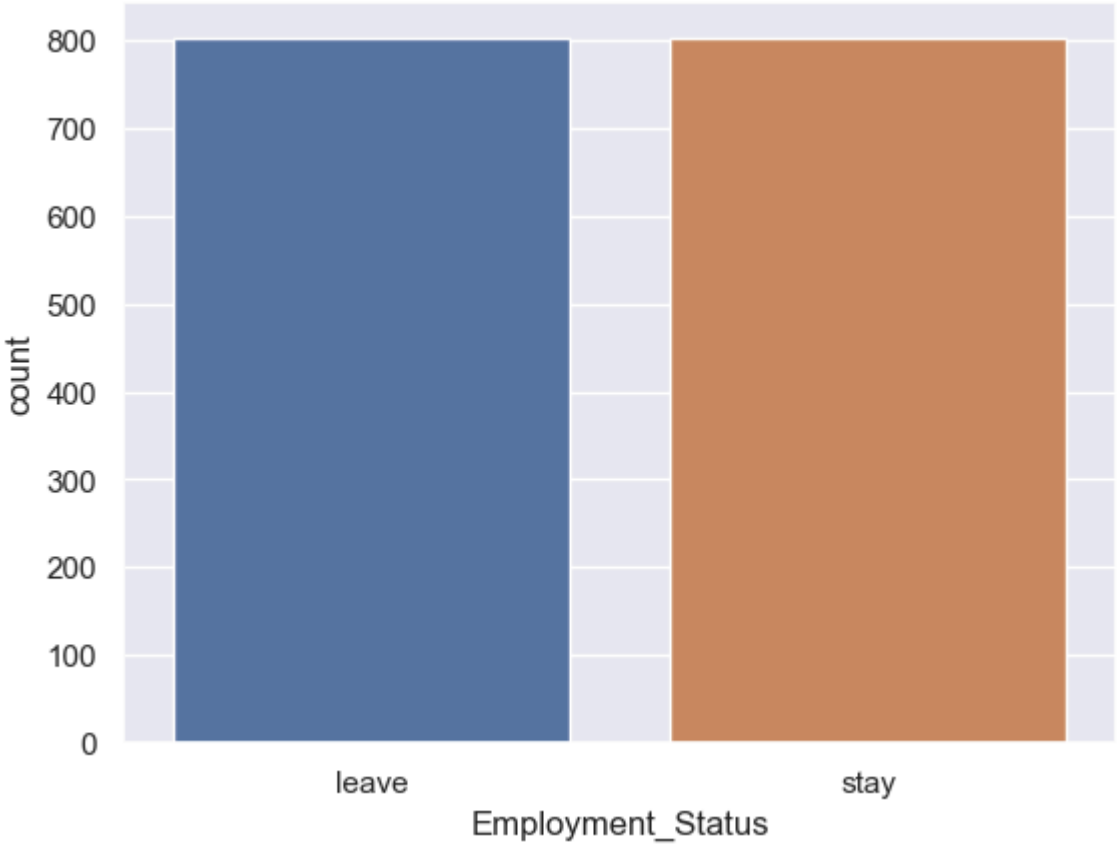
In [62]:
```
from sklearn.utils import resample
#create two different dataframe of majority and minority class
df_majority = df[(df['Employment_Status']=='stay')]
df_minority = df[(df['Employment_Status']=='leave')]
# upsample minority class
df_minority_upsampled = resample(df_minority,
                                 n_samples= 803,
                                 random_state=0)

# Combine majority class with upsampled minority class
df2 = pd.concat([df_minority_upsampled, df_majority])
```

In [63]:
```
sns.countplot(data=df2, x='Employment_Status')
print(df2.Employment_Status.value_counts())
```

```
C:\Users\sjkar\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use isin
stance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\sjkar\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use isin
stance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\sjkar\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use isin
stance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
```

```
Employment_Status
leave    803
stay     803
Name: count, dtype: int64
```



In [64]:
```python
X = df2.drop(columns=["Employment_Status"])
y = df2["Employment_Status"]

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

Implementing Decision Tree from scratch(Using Gini impurity or entropy criterion)

In [65]:
```python
def calculate_gini_impurity(y):
    classes = np.unique(y)
    gini = 0
    total_samples = len(y)
    for cls in classes:
        p = np.sum(y == cls) / total_samples
        gini += p * (1 - p)
    return gini
```

In [66]:
```python
def calculate_entropy(y):
    classes = np.unique(y)
    entropy = 0
    total_samples = len(y)
    for cls in classes:
        p = np.sum(y == cls) / total_samples
```

```
        entropy -= p * np.log2(p)
    return entropy
```

In [67]:
```
class TreeNode:
    def __init__(self, feature=None, threshold=None, left=None, right=None, value=N
        self.feature = feature
        self.threshold = threshold
        self.left = left
        self.right = right
        self.value = value
```

In [68]:
```
class DecisionTree:
    def __init__(self, max_depth=None):
        self.root = None
        self.criterion = "gini"
        self.max_depth = max_depth

    def _best_split(self, X, y):
        best_gini = np.inf
        best_feature_idx = None
        best_threshold = None

        for feature_idx in range(X.shape[1]):
            thresholds = np.unique(X[:, feature_idx])
            for threshold in thresholds:
                left_indices = X[:, feature_idx] < threshold
                right_indices = X[:, feature_idx] >= threshold

                gini_left = calculate_gini_impurity(y[left_indices])
                gini_right = calculate_gini_impurity(y[right_indices])
                gini = (
                        len(y[left_indices]) * gini_left
                        + len(y[right_indices]) * gini_right
                    ) / len(y)

                if gini < best_gini:
                    best_gini = gini
                    best_feature_idx = feature_idx
                    best_threshold = threshold

        return best_feature_idx, best_threshold

    def _build_tree(self, X, y, depth):
        if len(np.unique(y)) == 1:
            return TreeNode(value=y[0])

        if self.max_depth is not None and depth >= self.max_depth:
            return TreeNode(value=np.bincount(y).argmax())

        best_feature_idx, best_threshold = self._best_split(X, y)

        if best_feature_idx is None:
            return TreeNode(value=np.bincount(y).argmax())

        left_indices = X[:, best_feature_idx] < best_threshold
```

```
        right_indices = X[:, best_feature_idx] >= best_threshold

        left_subtree = self._build_tree(X[left_indices], y[left_indices], depth + 1
        right_subtree = self._build_tree(X[right_indices], y[right_indices], depth

        return TreeNode(
            feature=best_feature_idx,
            threshold=best_threshold,
            left=left_subtree,
            right=right_subtree,
        )

    def fit(self, X, y):
        self.root = self._build_tree(X, y, depth=0)

    def _predict_one(self, x, node):
        if node.value is not None:
            return node.value
        if x[node.feature] < node.threshold:
            return self._predict_one(x, node.left)
        else:
            return self._predict_one(x, node.right)

    def predict(self, X):
        predictions = []
        for x in X:
            predictions.append(self._predict_one(x, self.root))
        return np.array(predictions)
```

Evaluation of model

In [69]:
```
tree = DecisionTree( max_depth=None)

tree.fit(X_train.values, y_train.values)

test_predictions = tree.predict(X_test.values)

accuracy = np.mean(test_predictions == y_test)

print("Accuracy:", accuracy)
```

```
Accuracy: 1.0
```

In [70]:
```
employee_new = np.array([[75, 7, 5]])
prediction = tree.predict(employee_new)
print("Predicted employment status:", prediction[0])
```

```
Predicted employment status: stay
```

Plotting

In [71]:
```
data_5_years_exp = df[df["Years of Experience"] == 5]

x_min, x_max = (
    data_5_years_exp["Salary"].min() - 1,
    data_5_years_exp["Salary"].max() + 1,
```

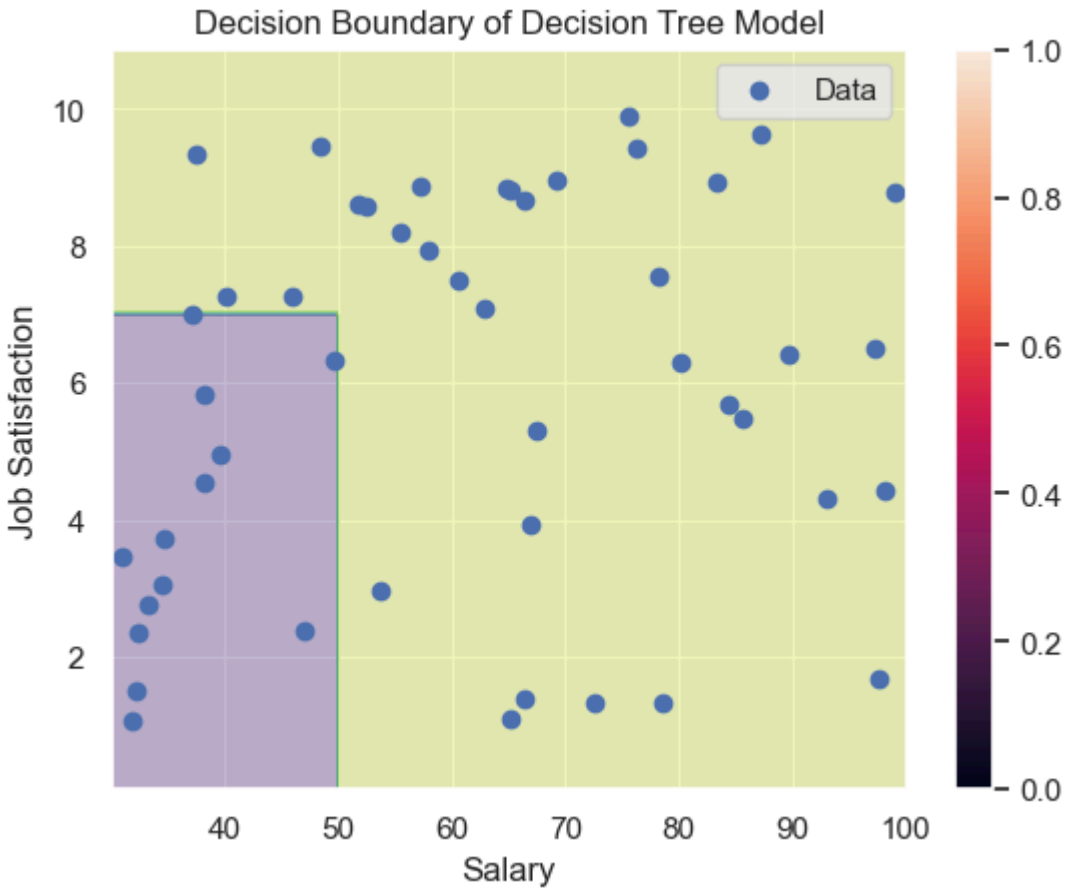
```
)
y_min, y_max = (
    data_5_years_exp["Job Satisfaction"].min() - 1,
    data_5_years_exp["Job Satisfaction"].max() + 1,
)
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1), np.arange(y_min, y_max, 0.1))

mesh_data = np.c_[
    xx.ravel(), yy.ravel(), np.ones_like(xx.ravel()) * 5
]
Z = tree.predict(mesh_data).reshape(xx.shape)

Z_numeric = np.where(Z == "leave", 0, 1)

plt.contourf(xx, yy, Z_numeric, alpha=0.3, cmap="viridis")
plt.scatter(
    data_5_years_exp["Salary"],
    data_5_years_exp["Job Satisfaction"],
    cmap="viridis",
    label="Data",
)
plt.xlabel("Salary")
plt.ylabel("Job Satisfaction")
plt.title("Decision Boundary of Decision Tree Model")
plt.legend()
plt.colorbar()
plt.show()
```

```
C:\Users\sjkar\AppData\Local\Temp\ipykernel_40692\569100420.py:21: UserWarning: No d
ata for colormapping provided via 'c'. Parameters 'cmap' will be ignored
  plt.scatter(
```


Decision Boundary of Decision Tree Model
Data
Job Satisfaction
Salary
10
8
6
4
2
40
50
60
70
80
90
100
1.0
0.8
0.6
0.4
0.2
0.0