Machine Learning Lab 6: Decision Tree Classification

Name: Deshmukh Pratik Bhushanrao

Roll No: 2448513

In [2]:
```
import pandas as pd
import numpy as np
import seaborn as sns

#inferenceimporting the essential libraries
```

In [3]:
```
df = pd.read_excel('employee_data.xlsx')
df.head()

#importing the dataset
```

Out[3]:

	avg_monthly_hrs	department	filed_complaint	last_evaluation	n_projects	recently_p
0	221	engineering	NaN	0.932868	4	
1	232	support	NaN	NaN	3	
2	184	sales	NaN	0.788830	3	
3	206	sales	NaN	0.575688	4	
4	249	sales	NaN	0.845217	3	

1. Data Exploration and Visualization:

In [6]:
```
df.info()

# checking for missing values
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14249 entries, 0 to 14248
Data columns (total 10 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   avg_monthly_hrs    14249 non-null  int64  
 1   department         13540 non-null  object 
 2   filed_complaint    2058 non-null   float64
 3   last_evaluation    12717 non-null  float64
 4   n_projects         14249 non-null  int64  
 5   recently_promoted  300 non-null    float64
 6   salary             14249 non-null  object 
 7   satisfaction       14068 non-null  float64
 8   status             14249 non-null  object 
 9   tenure             14068 non-null  float64
dtypes: float64(5), int64(2), object(3)
memory usage: 1.1+ MB
```

In [7]:

```
df.isna().sum()

#checking for null values
```

Out[7]:

```
avg_monthly_hrs          0
department             709
filed_complaint      12191
last_evaluation       1532
n_projects               0
recently_promoted    13949
salary                   0
satisfaction           181
status                   0
tenure                 181
dtype: int64
```

In [8]:

```
df.shape

#checking the shape of the dataset
```

Out[8]:

```
(14249, 10)
```

In [10]:

```
df.describe()

#checking the descriptive statistics
```

Out[10]:

	avg_monthly_hrs	filed_complaint	last_evaluation	n_projects	recently_promot
count	14249.000000	2058.0	12717.000000	14249.000000	30(
mean	199.795775	1.0	0.718477	3.773809	'
std	50.998714	0.0	0.173062	1.253126	(
min	49.000000	1.0	0.316175	1.000000	'
25%	155.000000	1.0	0.563866	3.000000	'
50%	199.000000	1.0	0.724939	4.000000	'
75%	245.000000	1.0	0.871358	5.000000	'
max	310.000000	1.0	1.000000	7.000000	'

In [11]:

```
df.drop(['filed_complaint', 'recently_promoted'], axis=1, inplace=True)
df.head()

#dropping the unnecessary columns
```

Out[11]:

	avg_monthly_hrs	department	last_evaluation	n_projects	salary	satisfaction	s
0	221	engineering	0.932868	4	low	0.829896	
1	232	support	NaN	3	low	0.834544	Emp
2	184	sales	0.788830	3	medium	0.834988	Emp
3	206	sales	0.575688	4	low	0.424764	Emp
4	249	sales	0.845217	3	low	0.779043	Emp

In [12]:

```
df.isna().sum()

#again checking for null values
```

Out[12]:

```
avg_monthly_hrs       0
department          709
last_evaluation    1532
n_projects            0
salary                0
satisfaction        181
status                0
tenure              181
dtype: int64
```

Data imputation

In [13]:

```
df['department'].fillna(df['department'].mode()[0], inplace=True)
df['last_evaluation'].fillna(df['last_evaluation'].mean(), inplace=True)
df['satisfaction'].fillna(df['satisfaction'].mean(), inplace=True)
df['tenure'].fillna(df['tenure'].mean(), inplace=True)
```

```
#filling the missing values
```

```
C:\Users\prati\AppData\Local\Temp\ipykernel_1016\3877897122.py:1: FutureWarning:
A value is trying to be set on a copy of a DataFrame or Series through chained as
signment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work becau
se the intermediate object on which we are setting values always behaves as a cop
y.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.meth
od({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to pe
rform the operation inplace on the original object.


  df['department'].fillna(df['department'].mode()[0], inplace=True)
C:\Users\prati\AppData\Local\Temp\ipykernel_1016\3877897122.py:2: FutureWarning:
A value is trying to be set on a copy of a DataFrame or Series through chained as
signment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work becau
se the intermediate object on which we are setting values always behaves as a cop
y.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.meth
od({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to pe
rform the operation inplace on the original object.


  df['last_evaluation'].fillna(df['last_evaluation'].mean(), inplace=True)
C:\Users\prati\AppData\Local\Temp\ipykernel_1016\3877897122.py:3: FutureWarning:
A value is trying to be set on a copy of a DataFrame or Series through chained as
signment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work becau
se the intermediate object on which we are setting values always behaves as a cop
y.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.meth
od({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to pe
rform the operation inplace on the original object.


  df['satisfaction'].fillna(df['satisfaction'].mean(), inplace=True)
C:\Users\prati\AppData\Local\Temp\ipykernel_1016\3877897122.py:4: FutureWarning:
A value is trying to be set on a copy of a DataFrame or Series through chained as
signment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work becau
se the intermediate object on which we are setting values always behaves as a cop
y.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.meth
od({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to pe
rform the operation inplace on the original object.


  df['tenure'].fillna(df['tenure'].mean(), inplace=True)
```

In [14]:

```
df.isna().sum()

#checking for null values
```

Out[14]:
```
avg_monthly_hrs    0
department         0
last_evaluation    0
n_projects         0
salary             0
satisfaction       0
status             0
tenure             0
dtype: int64
```

In [15]:
```
df.head()

#checking the dataset
```

Out[15]:

	avg_monthly_hrs	department	last_evaluation	n_projects	salary	satisfaction	s
0	221	engineering	0.932868	4	low	0.829896	
1	232	support	0.718477	3	low	0.834544	Emp
2	184	sales	0.788830	3	medium	0.834988	Emp
3	206	sales	0.575688	4	low	0.424764	Emp
4	249	sales	0.845217	3	low	0.779043	Emp

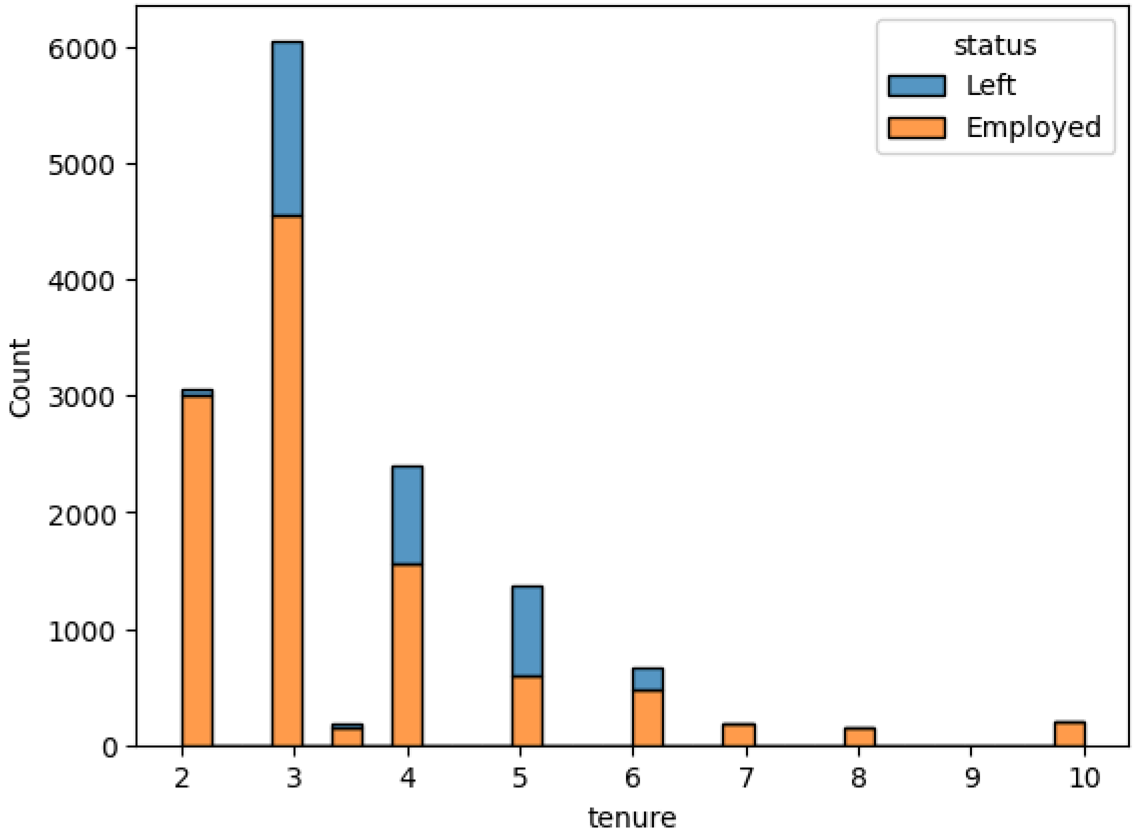
In [16]:
```
df['department'].unique()

#checking for unique values
```

Out[16]:
```
array(['engineering', 'support', 'sales', 'IT', 'product', 'marketing',
       'temp', 'procurement', 'finance', 'management',
       'information_technology', 'admin'], dtype=object)
```

visualizations

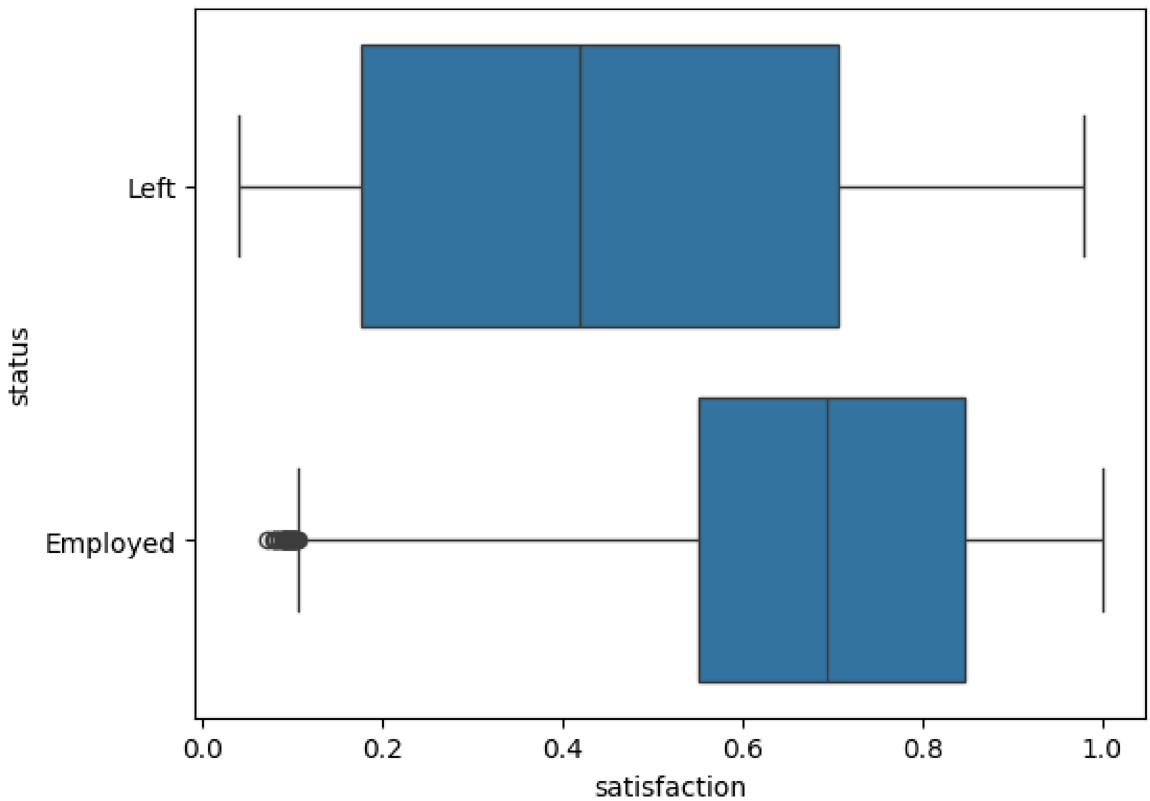
In [17]:
```
sns.histplot(data=df,x='tenure',hue='status',bins=30,multiple='stack')

#checking the distribution of the target variable
```

Out[17]:
```
<Axes: xlabel='tenure', ylabel='Count'>
```



In [18]:
```
sns.boxplot(data=df,x='satisfaction',y='status')

#checking for outliers
```

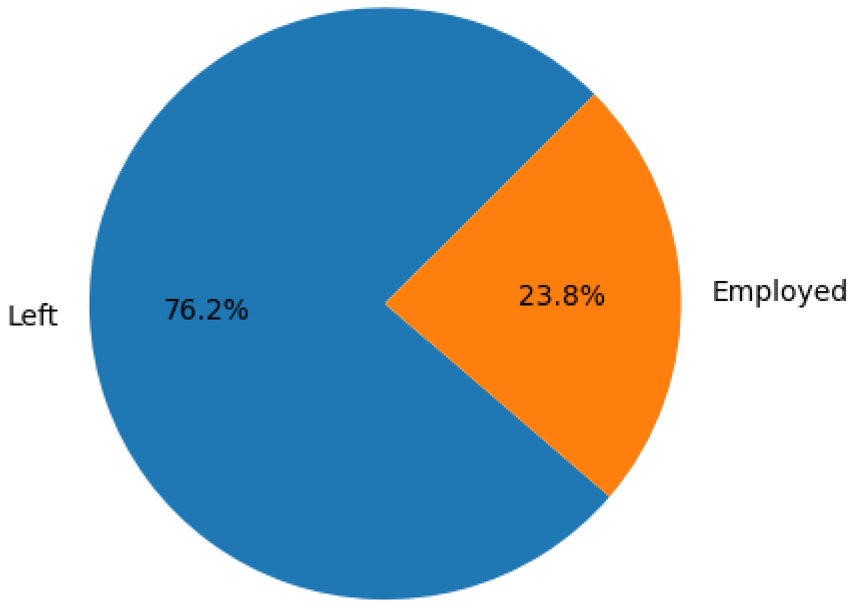
Out[18]:
```
<Axes: xlabel='satisfaction', ylabel='status'>
```



In [20]:
```
import matplotlib.pyplot as plt
```

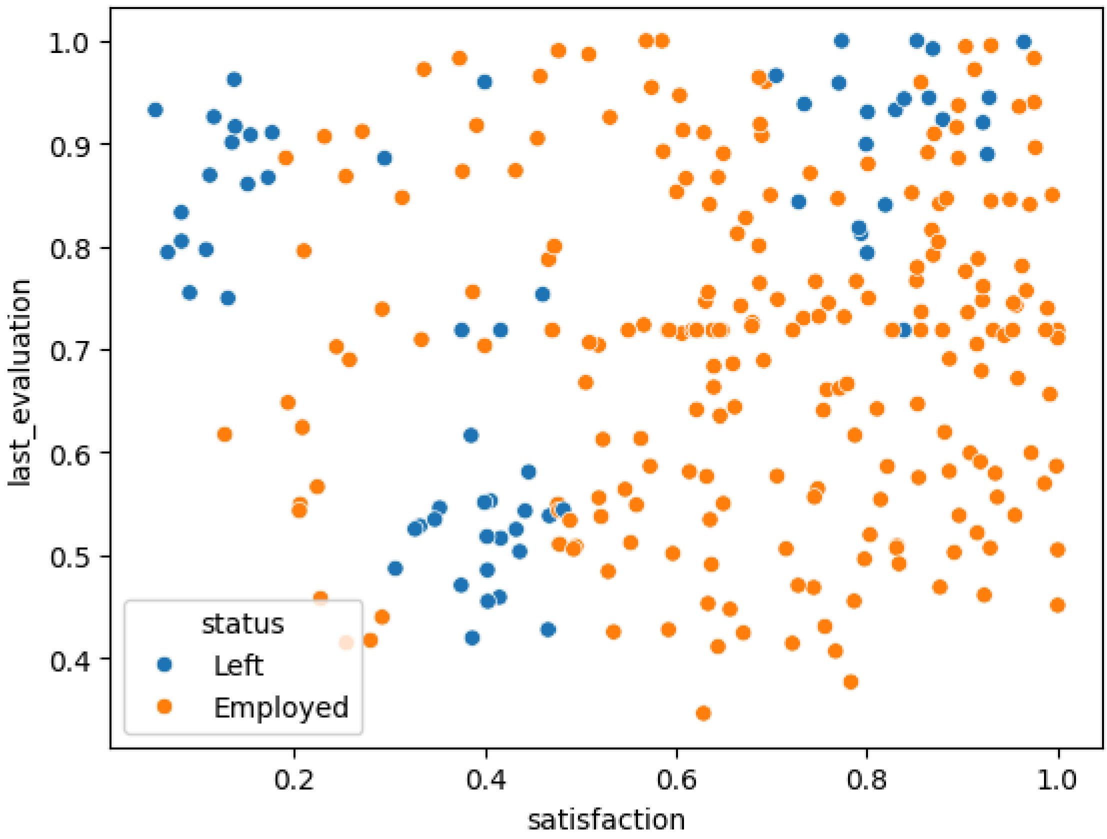
In [21]:
```
plt.pie(df["status"].value_counts(),labels=df["status"].unique(),autopct='%1.1f%
plt.plot()

#checking the distribution of the target variable
```

Out[21]: []



In [22]:
```
sns.scatterplot(data=df[::50],x='satisfaction',y='last_evaluation',hue='status')

#checking for correlation
```

Out[22]: <Axes: xlabel='satisfaction', ylabel='last_evaluation'>



Encoding

In [24]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df['department'] = le.fit_transform(df['department'])
df['salary'] = le.fit_transform(df['salary'])
df['status'] = le.fit_transform(df['status'])
df.head()

#encoding the categorical variables
```

Out[24]:

	avg_monthly_hrs	department	last_evaluation	n_projects	salary	satisfaction	status
0	221	2	0.932868	4	1	0.829896	1
1	232	10	0.718477	3	1	0.834544	0
2	184	9	0.788830	3	2	0.834988	0
3	206	9	0.575688	4	1	0.424764	0
4	249	9	0.845217	3	1	0.779043	0

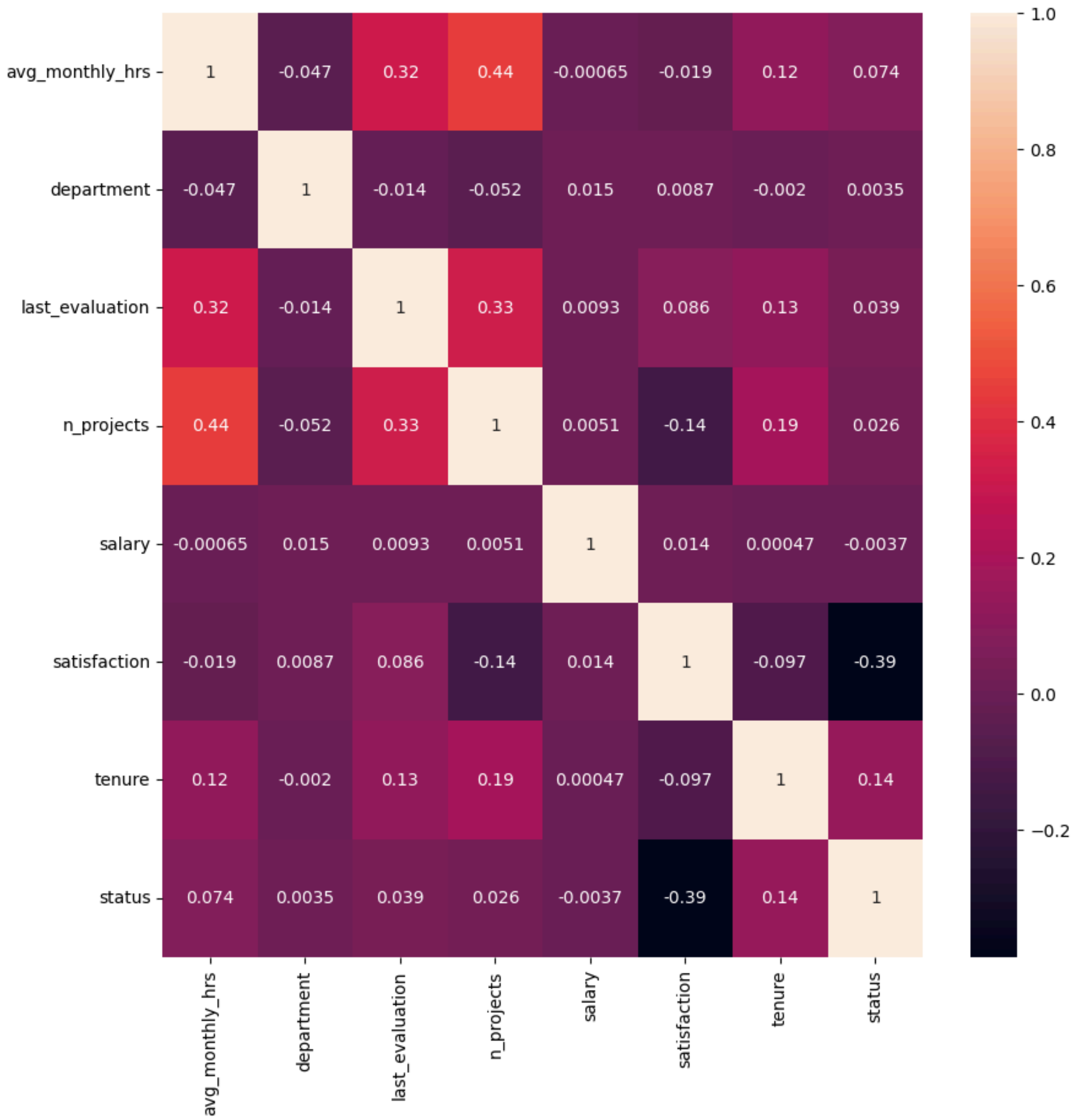
In [25]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x = pd.DataFrame(sc.fit_transform(df.drop('status', axis=1)), columns=df.columns
df_scaled = pd.concat([x, df['status']], axis=1)
df_scaled.head()

#standardizing(scaling) the dataset
```

Out[25]:

	avg_monthly_hrs	department	last_evaluation	n_projects	salary	satisfaction	
0	0.415794	-1.205166	1.311357	0.180508	-0.561893	0.838210	1.
1	0.631493	1.055552	0.000000	-0.617524	-0.561893	0.856885	-1.
2	-0.309740	0.772962	0.430327	-0.617524	1.040992	0.858668	-0.
3	0.121659	0.772962	-0.873391	0.180508	-0.561893	-0.789711	-1.
4	0.964847	0.772962	0.775231	-0.617524	-0.561893	0.633869	-0.

In [26]:

```
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 10))
sns.heatmap(df_scaled.corr(), annot=True)
plt.show()

#checking for correlation
```



In [ ]:

2. Decision Tree Modeling with Tree Pruning and Split Criteria:

Split the dataset into a training set and a testing set (e.g., 80% training, 20% testing).

In [27]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(df_scaled.drop('status', a

#splitting the dataset into train and test sets with 80:20 ratio
```

Build a decision tree classifier to predict employee attrition based on selected features (e.g., tenure, satisfaction, number of projects).

In [28]:
```
from sklearn.tree import DecisionTreeClassifier
cnf_gini = DecisionTreeClassifier(criterion='gini')
cnf_gini.fit(X_train, y_train)
y_pred = cnf_gini.predict(X_test)

cnf_gini.score(X_test, y_test)

#checking building the model with Gini impurity criterion and checking the accur
```

Out[28]:
```
0.9578947368421052
```

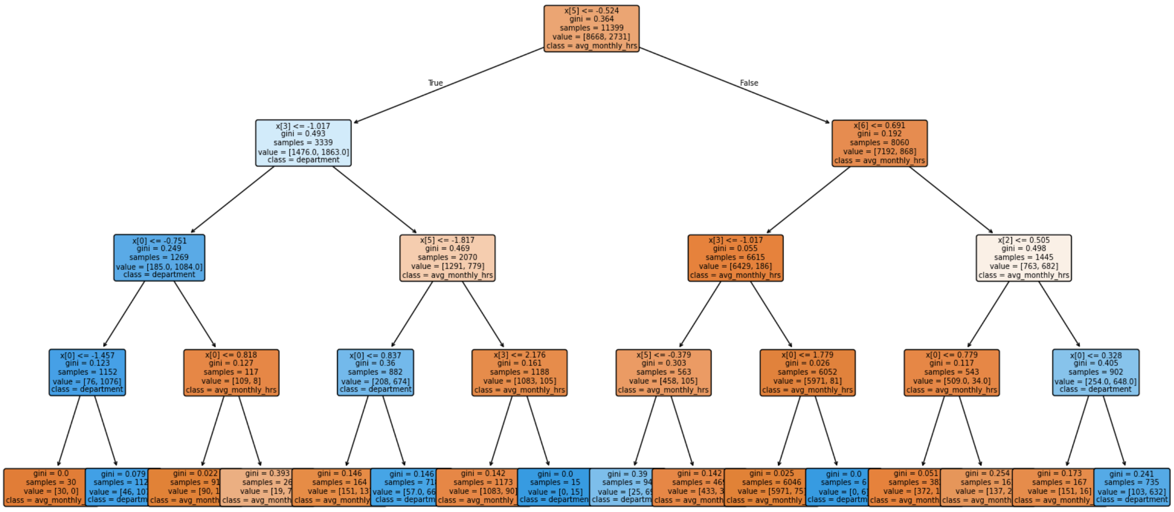
In [29]:
```
cnf_entropy = DecisionTreeClassifier(criterion='entropy')
cnf_entropy.fit(X_train, y_train)
y_pred = cnf_entropy.predict(X_test)

cnf_entropy.score(X_test, y_test)

#checking building the model with Entropy criterion and checking the accuracy
```

Out[29]:
```
0.9652631578947368
```

Visualize the decision tree structure. How deep is the tree, and what are the most influential features for predicting attrition?

In [30]:
```
from sklearn.tree import export_text, plot_tree

example_tree = DecisionTreeClassifier(criterion='gini', max_depth=4, min_samples
plt.figure(figsize=(20, 10))
plot_tree(example_tree, class_names=X_train.columns, filled=True, rounded=True,
plt.plot()

#plotting the decision tree
```

Out[30]:
```
[]
```

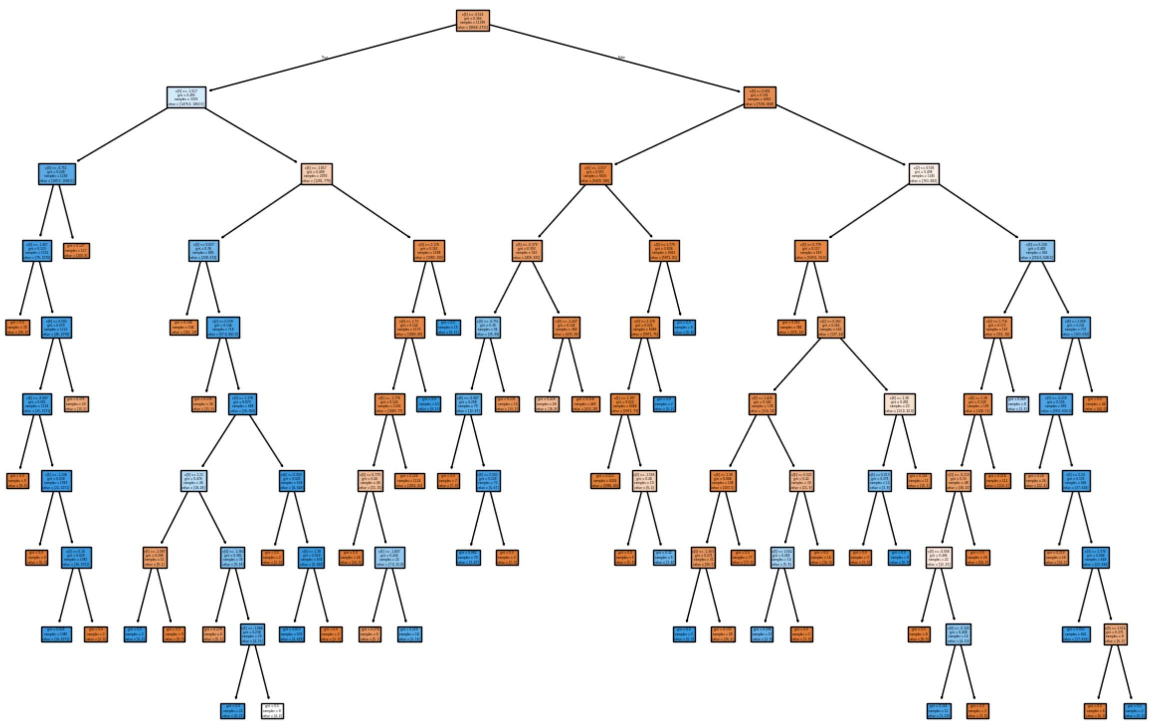


Apply post-pruning techniques to control the complexity of the tree and prevent overfitting. Experiment with different pruning strategies, such as minimum leaf size or maximum depth, to find the optimal tree size

In [31]:
```
path = cnf_gini.cost_complexity_pruning_path(X_train, y_train)
ccp_alphas, impurities = path.ccp_alphas, path.impurities

pruned_models = []

for ccp_alphas in ccp_alphas:
    pruned_model = DecisionTreeClassifier(criterion='gini', ccp_alpha=ccp_alphas
    pruned_models.append(pruned_model)

best_accuracy = 0
best_pruned_model = None

for pruned_model in pruned_models:
    accuracy = pruned_model.score(X_test, y_test)
    if accuracy > best_accuracy:
        best_accuracy = accuracy
        best_pruned_model = pruned_model

# finding the best pruned model based on accuracy
```

In [32]:
```
plt.figure(figsize=(15, 10))
plot_tree(best_pruned_model, filled=True)
plt.title("Pruned Decision Tree")
plt.show()

# plotting the pruned decision tree
```

Pruned Decision Tree



In [33]:
```
from sklearn.model_selection import GridSearchCV
model = DecisionTreeClassifier()
params = {
    'criterion': ['gini','entropy'],
    'max_depth': list(np.random.randint(2, 20, 1)),
    'min_samples_leaf': [1,2,4,6]
}
search = GridSearchCV(model, params, scoring='accuracy').fit(X_train, y_train)

#finding the best model
```

In [34]:
```
print(search.best_params_)
print(search.best_score_)

#finding the best parameters and best score
```

```
{'criterion': 'gini', 'max_depth': 4, 'min_samples_leaf': 1}
0.9535926037120005
```

In [35]:
```
bestModel = search.best_estimator_
y_pred = bestModel.predict(X_test)
bestModel.score(X_test, y_test)

#checking the accuracy of the best model
```

Out[35]: 0.9624561403508772

3. Model Evaluation

In [37]:
```
from sklearn.metrics import confusion_matrix, classification_report
```

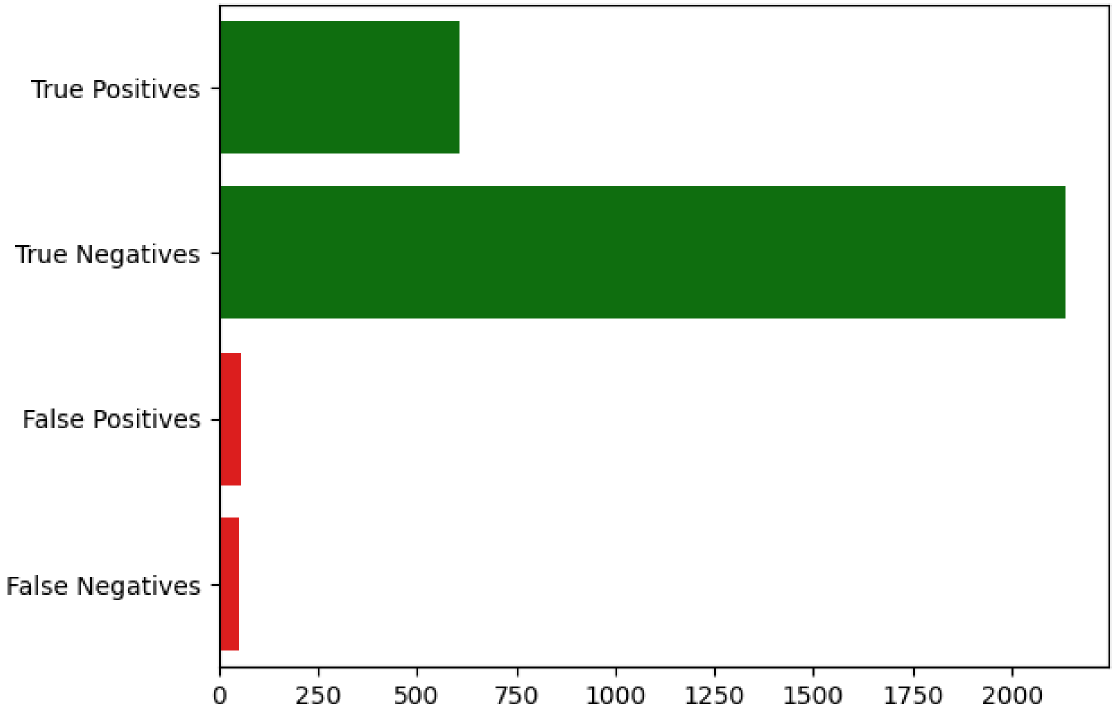
In [38]:
```
conf_mat = confusion_matrix(y_test, y_pred)
print(conf_mat, end="\n\n")
print(classification_report(y_test, y_pred))

#checking the confusion matrix and classification report
```

```
[[2135   54]
 [  53  608]]

              precision    recall  f1-score   support

           0       0.98      0.98      0.98      2189
           1       0.92      0.92      0.92       661

    accuracy                           0.96      2850
   macro avg       0.95      0.95      0.95      2850
weighted avg       0.96      0.96      0.96      2850
```

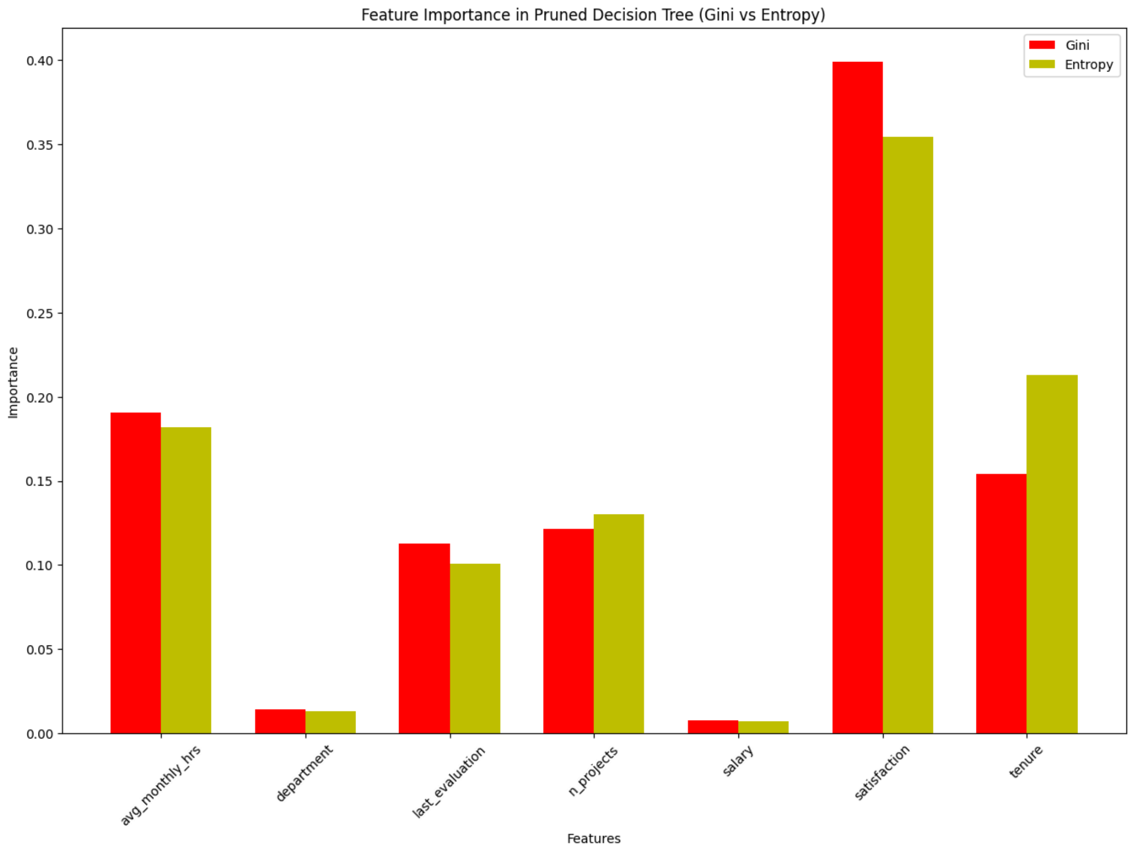
4. Feature Importance Visualization:

In [40]:
```
tn, fp, fn, tp = conf_mat.ravel()
labels = ['True Positives', 'True Negatives', 'False Positives', 'False Negat

sns.barplot(y=labels, x=[tp, tn, fp, fn], palette=['green', 'green', 'red', '
plt.show()

#plotting the confusion matrix with barchart
```

```
C:\Users\prati\AppData\Local\Temp\ipykernel_1016\342439456.py:4: FutureWarnin
g:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(y=labels, x=[tp, tn, fp, fn], palette=['green', 'green', 'red',
'red'])
```



In [41]:
```
entropy_imp = cnf_entropy.feature_importances_
gini_imp = cnf_gini.feature_importances_
features = X_test.columns

# Plotting the bar chart
fig, ax = plt.subplots(figsize=(15, 10))
bar_width = 0.35
index = np.arange(len(features))

# Gini bar chart
bar1 = plt.bar(index, gini_imp, bar_width, label='Gini', color='r')

# Entropy bar chart
bar2 = plt.bar(index + bar_width, entropy_imp, bar_width, label='Entropy', co

# Adding Labels and Titles
plt.xlabel('Features')
plt.ylabel('Importance')
plt.title('Feature Importance in Pruned Decision Tree (Gini vs Entropy)')
plt.xticks(index + bar_width / 2, features, rotation=45)
plt.legend()


plt.show()

#plotting the feature importance after pruning
```


Feature Importance in Pruned Decision Tree (Gini vs Entropy)
Gini
Entropy
Importance
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
avg_monthly_hrs
department
last_evaluation
n_projects
salary
satisfaction
tenure
Features

------------------EOD------------------