HR ATTRIBUTION

In [2]:
```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import make_scorer, f1_score
import numpy as np
from sklearn.metrics import confusion_matrix, roc_curve, roc_auc_score, auc
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import numpy as np
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import make_scorer, roc_auc_score
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import accuracy_score
```

1.) Import, split data into X/y, plot y data as bar charts, turn X categorical variables binary and tts.

In [3]:
```
df = pd.read_csv("HR_Analytics.csv")
```

In [4]:
```
df.head()
```

Out[4]:

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationField	Em
0	41	Yes	Travel_Rarely	1102	Sales	1	2	Life Sciences	
1	49	No	Travel_Frequently	279	Research & Development	8	1	Life Sciences	
2	37	Yes	Travel_Rarely	1373	Research & Development	2	2	Other	
3	33	No	Travel_Frequently	1392	Research & Development	3	4	Life Sciences	
4	27	No	Travel_Rarely	591	Research & Development	2	1	Medical	

5 rows × 35 columns

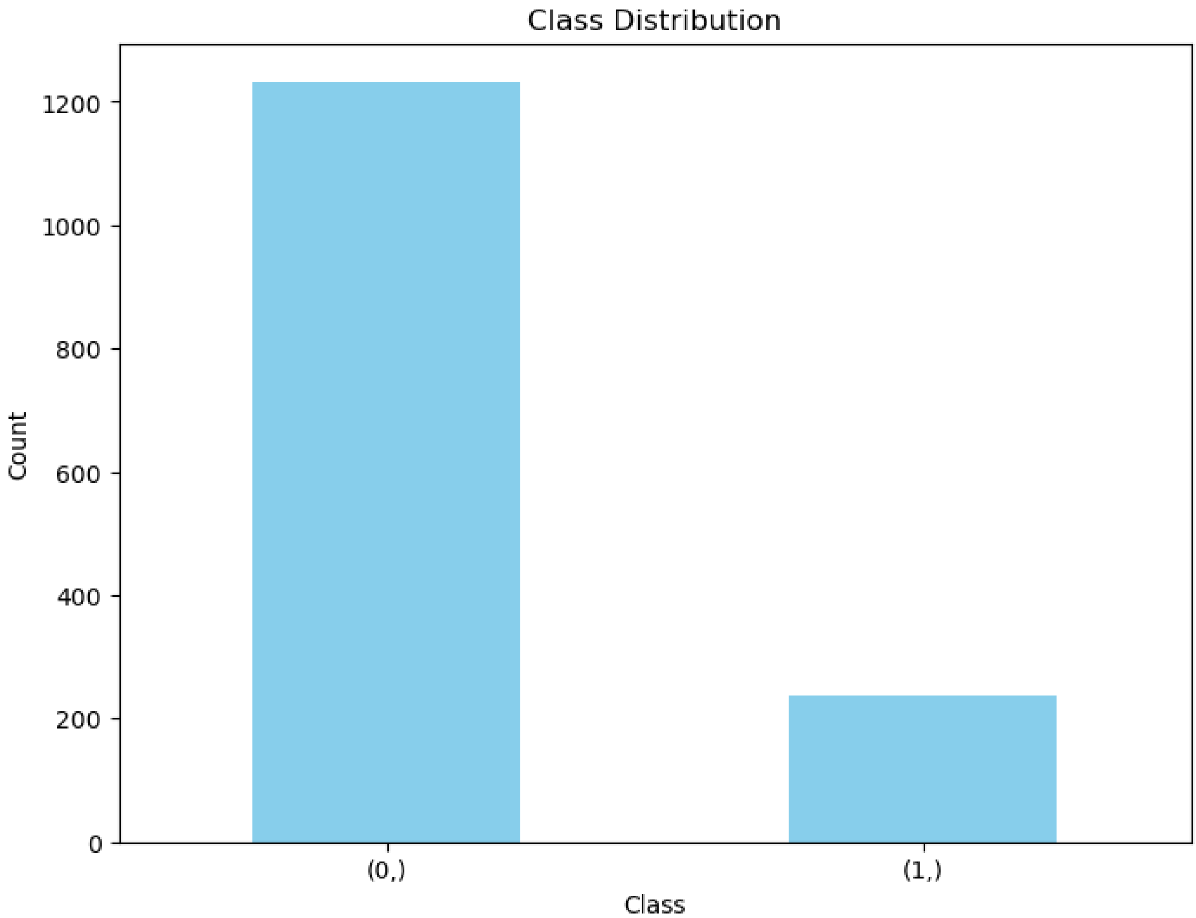
In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1470 entries, 0 to 1469
Data columns (total 35 columns):
 #   Column                    Non-Null Count  Dtype 
---  ------                    --------------  ----- 
 0   Age                       1470 non-null   int64 
 1   Attrition                 1470 non-null   object
 2   BusinessTravel            1470 non-null   object
 3   DailyRate                 1470 non-null   int64 
 4   Department                1470 non-null   object
 5   DistanceFromHome          1470 non-null   int64 
 6   Education                 1470 non-null   int64 
 7   EducationField            1470 non-null   object
 8   EmployeeCount             1470 non-null   int64 
 9   EmployeeNumber            1470 non-null   int64 
 10  EnvironmentSatisfaction   1470 non-null   int64 
 11  Gender                    1470 non-null   object
 12  HourlyRate                1470 non-null   int64 
 13  JobInvolvement            1470 non-null   int64 
 14  JobLevel                  1470 non-null   int64 
 15  JobRole                   1470 non-null   object
 16  JobSatisfaction           1470 non-null   int64 
 17  MaritalStatus             1470 non-null   object
 18  MonthlyIncome             1470 non-null   int64 
 19  MonthlyRate               1470 non-null   int64 
 20  NumCompaniesWorked        1470 non-null   int64 
 21  Over18                    1470 non-null   object
 22  OverTime                  1470 non-null   object
 23  PercentSalaryHike         1470 non-null   int64 
 24  PerformanceRating         1470 non-null   int64 
 25  RelationshipSatisfaction  1470 non-null   int64 
 26  StandardHours             1470 non-null   int64 
 27  StockOptionLevel          1470 non-null   int64 
 28  TotalWorkingYears         1470 non-null   int64 
 29  TrainingTimesLastYear     1470 non-null   int64 
 30  WorkLifeBalance           1470 non-null   int64 
 31  YearsAtCompany            1470 non-null   int64 
 32  YearsInCurrentRole        1470 non-null   int64 
 33  YearsSinceLastPromotion   1470 non-null   int64 
 34  YearsWithCurrManager      1470 non-null   int64 
dtypes: int64(26), object(9)
memory usage: 402.1+ KB
```

In [5]:
```
y = df[["Attrition"]].copy()
X = df.drop("Attrition", axis = 1)
```

In [6]:
```
y["Attrition"] = [1 if i == "Yes" else 0 for i in y["Attrition"]]
```

In [7]:
```
class_counts = y.value_counts()


plt.figure(figsize=(8, 6))
class_counts.plot(kind='bar', color='skyblue')
plt.xlabel('Class')
plt.ylabel('Count')
plt.title('Class Distribution')
plt.xticks(rotation=0)  # Remove rotation of x-axis labels
plt.show()
```



In [8]:
```
# Step 1: Identify string columns
string_columns = X.columns[X.dtypes == 'object']

# Step 2: Convert string columns to categorical
for col in string_columns:
    X[col] = pd.Categorical(X[col])

# Step 3: Create dummy columns
X = pd.get_dummies(X, columns=string_columns, prefix=string_columns,drop_first=True)
```

In [9]:
```
x_train,x_test,y_train,y_test=train_test_split(X,
 y, test_size=0.20, random_state=42)
```

2.) Using the default Decision Tree. What is the IN/Out of Sample accuracy?

In [10]:
```
clf = DecisionTreeClassifier()
clf.fit(x_train,y_train)
y_pred=clf.predict(x_train)
acc=accuracy_score(y_train,y_pred)
print("IN SAMPLE ACCURACY : " , round(acc,2))

y_pred=clf.predict(x_test)
acc=accuracy_score(y_test,y_pred)
print("OUT OF SAMPLE ACCURACY : " , round(acc,2))
```

```
IN SAMPLE ACCURACY :  1.0
OUT OF SAMPLE ACCURACY :  0.77
```

3.) Run a grid search cross validation using F1 score to find the best metrics. What is the In and

Out of Sample now?

In [11]:
```
# Define the hyperparameter grid to search through
param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': np.arange(1, 11),  # Range of max_depth values to try
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}


dt_classifier = DecisionTreeClassifier(random_state=42)

scoring = make_scorer(f1_score, average='weighted')

grid_search = GridSearchCV(estimator=dt_classifier, param_grid=param_grid, scoring=scoring, cv=

grid_search.fit(x_train, y_train)

# Get the best parameters and the best score
best_params = grid_search.best_params_
best_score = grid_search.best_score_

print("Best Parameters:", best_params)
print("Best F1-Score:", best_score)
```

```
Best Parameters: {'criterion': 'gini', 'max_depth': 6, 'min_samples_leaf': 2, 'min_samples_spli
t': 2}
Best F1-Score: 0.8214764475510983
```

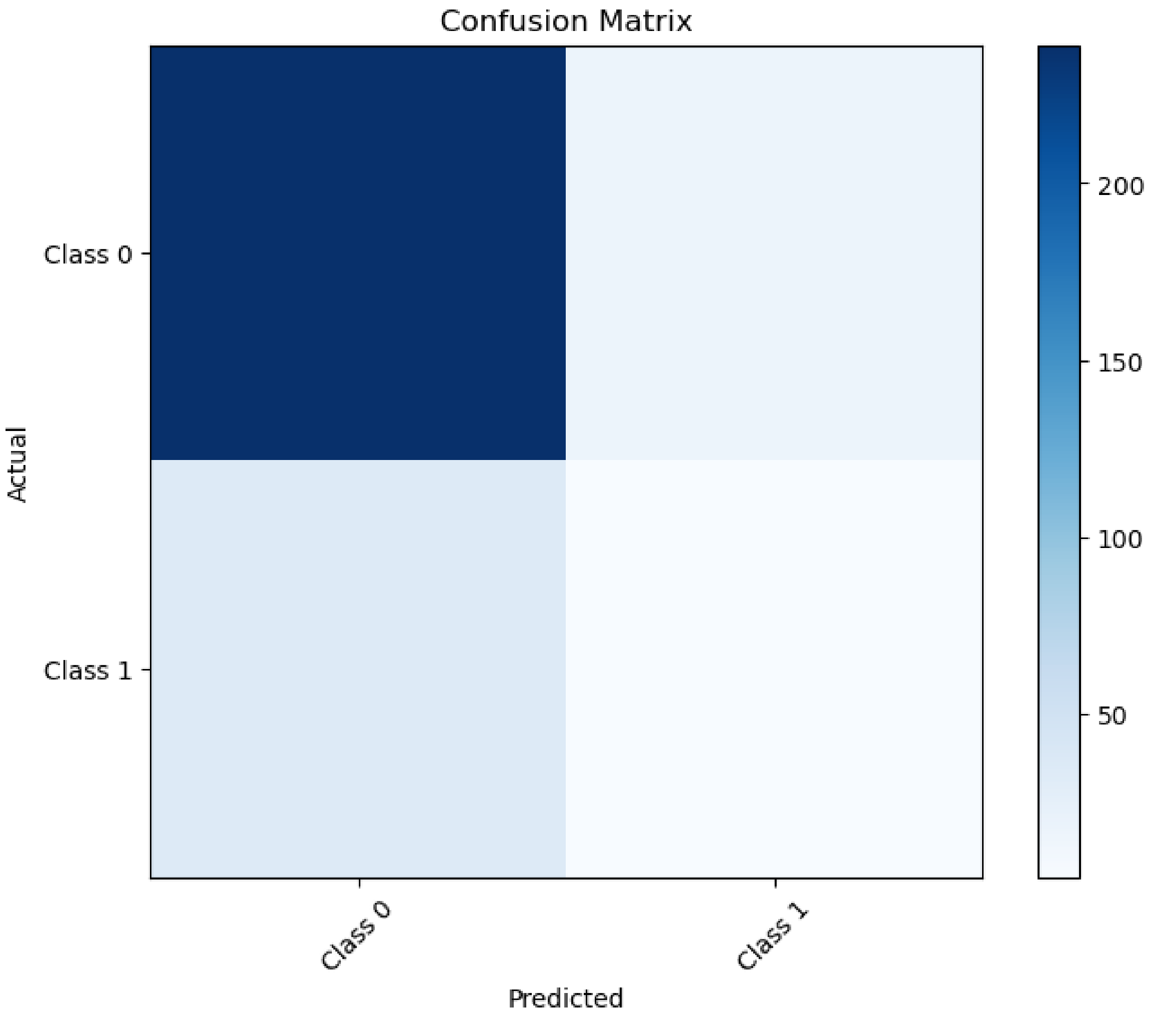
In [12]:
```
clf = tree.DecisionTreeClassifier(**best_params, random_state =42)
clf.fit(x_train,y_train)
y_pred=clf.predict(x_train)
acc=accuracy_score(y_train,y_pred)
print("IN SAMPLE ACCURACY : " , round(acc,2))

y_pred=clf.predict(x_test)
acc=accuracy_score(y_test,y_pred)
print("OUT OF SAMPLE ACCURACY : " , round(acc,2))
```

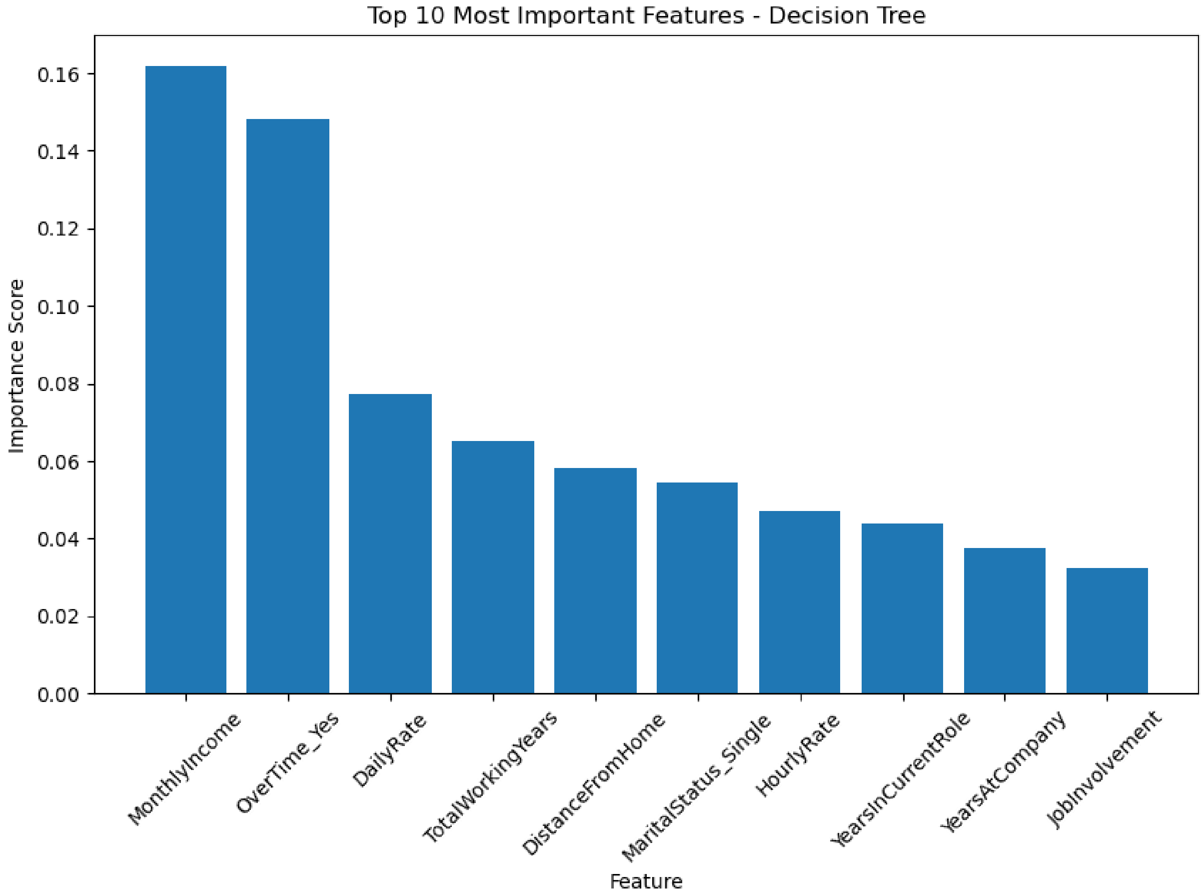
```
IN SAMPLE ACCURACY :  0.91
OUT OF SAMPLE ACCURACY :  0.83
```

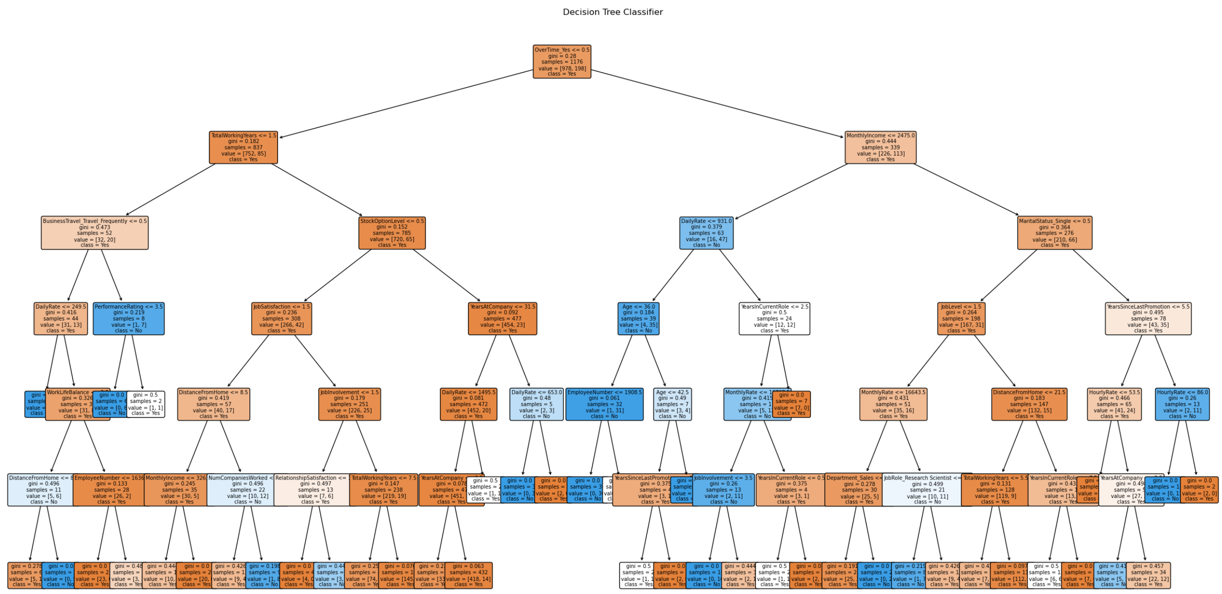
4.) Plot ......

In [13]:
```
# Make predictions on the test data
y_pred = clf.predict(x_test)
y_prob = clf.predict_proba(x_test)[:, 1]

# Calculate the confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Plot the confusion matrix
plt.figure(figsize=(8, 6))
plt.imshow(conf_matrix, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
plt.colorbar()
tick_marks = np.arange(len(conf_matrix))
plt.xticks(tick_marks, ['Class 0', 'Class 1'], rotation=45)
plt.yticks(tick_marks, ['Class 0', 'Class 1'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()




feature_importance = clf.feature_importances_
```

```python
# Sort features by importance and select the top 10
top_n = 10
top_feature_indices = np.argsort(feature_importance)[::-1][:top_n]
top_feature_names = X.columns[top_feature_indices]
top_feature_importance = feature_importance[top_feature_indices]

# Plot the top 10 most important features
plt.figure(figsize=(10, 6))
plt.bar(top_feature_names, top_feature_importance)
plt.xlabel('Feature')
plt.ylabel('Importance Score')
plt.title('Top 10 Most Important Features - Decision Tree')
plt.xticks(rotation=45)
plt.show()

# Plot the Decision Tree for better visualization of the selected features
plt.figure(figsize=(30, 15))
plot_tree(clf, filled=True, feature_names=X.columns, class_names=["Yes", "No"], rounded=True,
plt.title('Decision Tree Classifier')
plt.show()
```







In [14]:

Out[14]:

```
0       1
1       0
2       1
3       0
4       0
       ..
1465    0
1466    0
1467    0
1468    0
1469    0
Name: Attrition, Length: 1470, dtype: int64
```

In [ ]:

5.) Looking at the graphs. What would be your suggestions to try to improve employee retention? What additional information would you need for a better plan. Calculate anything you think would assist in your assessment.

ANSWER :

According to the graph of decision tree, we figure out top 10 most important features: $monthlyincome$, $dailyrate$, $overtimeyes$, age, $monthlyrate$, $totalworkingyears$, $yearsincurrentrole$, $distancefromhome$. They make the biggest contribution to the attraction of job positions. Also, we look at the decision tree and find that $monthlyincome$, $dailyrate$, $overtimeyes$, $totalworkingyears$ are the most improtant factors contributing to employee retention.

In [15]:
```
np.corrcoef(np.array(X["OverTime_Yes"]),y["Attrition"])
```

Out[15]:
```
array([[1.        , 0.24611799],
       [0.24611799, 1.        ]])
```

In [15]:
```
np.corrcoef(np.array(X['MonthlyIncome']), y['Attrition'])
```

Out[15]:
```
array([[ 1.        , -0.15983958],
       [-0.15983958,  1.        ]])
```

In [16]:
```
np.corrcoef(np.array(X['TotalWorkingYears']), y['Attrition'])
```

Out[16]:
```
array([[ 1.        , -0.17106325],
       [-0.17106325,  1.        ]])
```

In [18]:
```
np.corrcoef(np.array(X['DailyRate']), y['Attrition'])
```

Out[18]:
```
array([[ 1.        , -0.05665199],
       [-0.05665199,  1.        ]])
```

ANSWER :

Looking at the correlation coefficients, we conclude that workers with higher overtime working rate, lower monthly income, fewer working years and lower daiy rate is more likely to leave. So in order to improve the employee retention, company should improve their salary structure and reduce the overtime working.

6.) Using the Training Data, if they made everyone work overtime. What would have been the expected difference in employee retention?

In [19]:
```
x_train_experiment = x_train.copy()
```

In [20]:
```
x_train_experiment["OverTime_Yes"] = 0.
```

In [21]:
```
y_pred_experiment = clf.predict(x_train_experiment)
y_pred = clf.predict(x_train)
```

In [22]:
```
print("Stopping overtime work would have prevented people from leaving:",sum(y_pred - y_pred_exp
```

```
Stopping overtime work would have prevented people from leaving: 59
```

The expected difference in employee retention would be 59.

7.) If they company loses an employee, there is a cost to train a new employee for a role ~2.8 * their monthly income.

To make someone not work overtime costs the company 2K per person.

Is it profitable for the company to remove overtime? If so/not by how much?

What do you suggest to maximize company profits?

In [23]:
```
x_train_experiment["Y"] = y_pred
x_train_experiment["Y_exp"] = y_pred_experiment
x_train_experiment["Ret_Change"] = x_train_experiment["Y"] - x_train_experiment["Y_exp"]
```

In [24]:
```
# Savings
savings = sum(x_train_experiment["Ret_Change"] * 2.8 * x_train_experiment["MonthlyIncome"])
```

In [25]:
```
cost = 2000 * len(x_train[x_train["OverTime_Yes"] == 1.])
```

In [26]:
```
print("profit form this experiment: ", savings - cost)
```

```
profit form this experiment:  -117593.99999999977
```

ANSWER :

It's not profitable to do that because from our results the profit would decrease greatly by 117594.

In order to improve the employee rentention and not reduce profits to that much extent together, company could consider reducing the overtime working instead of 100% removing the overtime working, that may help maximize company profits and maintain employee extention simultanouesly.

8.) Use your model and get the expected change in retention for raising and lowering peoples income. Plot the outcome of the experiment. Comment on the outcome of the experiment and your suggestions to maximize profit.

In [27]:
```
raise_amount = 500
```

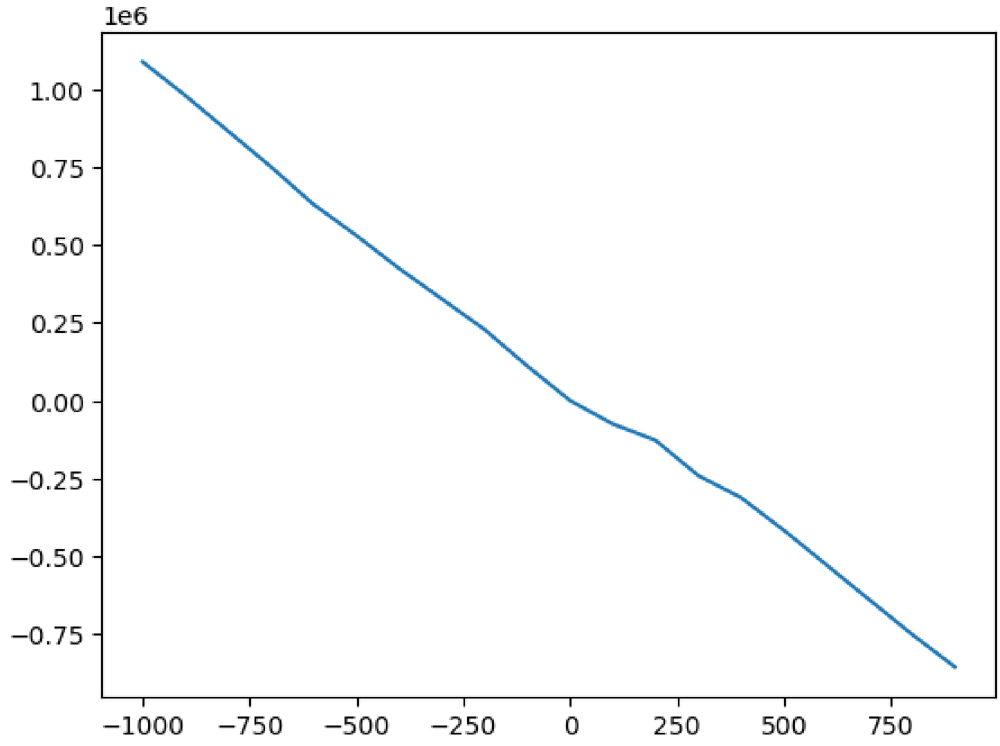
In [28]:
```
profits = []
for raise_amount in range(-1000,1000,100):
    x_train_experiment = x_train.copy()
    x_train_experiment["MonthlyIncome"] = x_train_experiment["MonthlyIncome"] + raise_amount
    y_pred_experiment = clf.predict(x_train_experiment)
```

```python
    y_pred = clf.predict(x_train)
    x_train_experiment["Y"] = y_pred
    x_train_experiment["Y_exp"] = y_pred_experiment
    x_train_experiment["Ret_Change"] = x_train_experiment["Y"] - x_train_experiment["Y_exp"]

    # Savings
    print("Retention different: ", sum(x_train_experiment["Ret_Change"]))
    savings = sum(x_train_experiment["Ret_Change"] * 2.8 * x_train_experiment["MonthlyIncome"])

    # Cost of lost overtime
    cost = raise_amount * len(x_train)

    print("Profit is: ", savings - cost)
    profits.append(savings - cost)
```

```
Retention different:  -16
Profit is:  1087584.4
Retention different:  -14
Profit is:  979524.0
Retention different:  -13
Profit is:  864992.8
Retention different:  -12
Profit is:  750738.8
Retention different:  -12
Profit is:  629778.8
Retention different:  -9
Profit is:  530138.0
Retention different:  -7
Profit is:  424200.0
Retention different:  -4
Profit is:  326096.4
Retention different:  -1
Profit is:  228440.8
Retention different:  -1
Profit is:  110714.8
Retention different:  0
Profit is:  0.0
Retention different:  6
Profit is:  -75328.40000000001
Retention different:  15
Profit is:  -127503.60000000002
Retention different:  15
Profit is:  -240914.8
Retention different:  21
Profit is:  -311586.80000000005
Retention different:  22
Profit is:  -416449.6000000001
Retention different:  22
Profit is:  -527889.6000000001
Retention different:  22
Profit is:  -639329.6000000001
Retention different:  22
Profit is:  -750769.6000000001
Retention different:  23
Profit is:  -854999.6000000001
```

In [29]:

```python
plt.plot(range(-1000,1000,100), profits)
plt.show()
```



ANSWER :

Looking at the graph, we could see that there is negative relationship between the raise amount of money company pays to employees and the profit it earns. In order to maximize the profit, I suggest that company shouldn't do any change in current employee salary structure, thus they can retain profits and have a relatively stable employee structure just as current situation.