HR ATTRIBUTION

In [1]:
```
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 [2]:
```
df = pd.read_csv("HR_Analytics.csv")
```

In [7]:
```
df.head()
#attrition =yes is bad, leave
```

Out[7]:

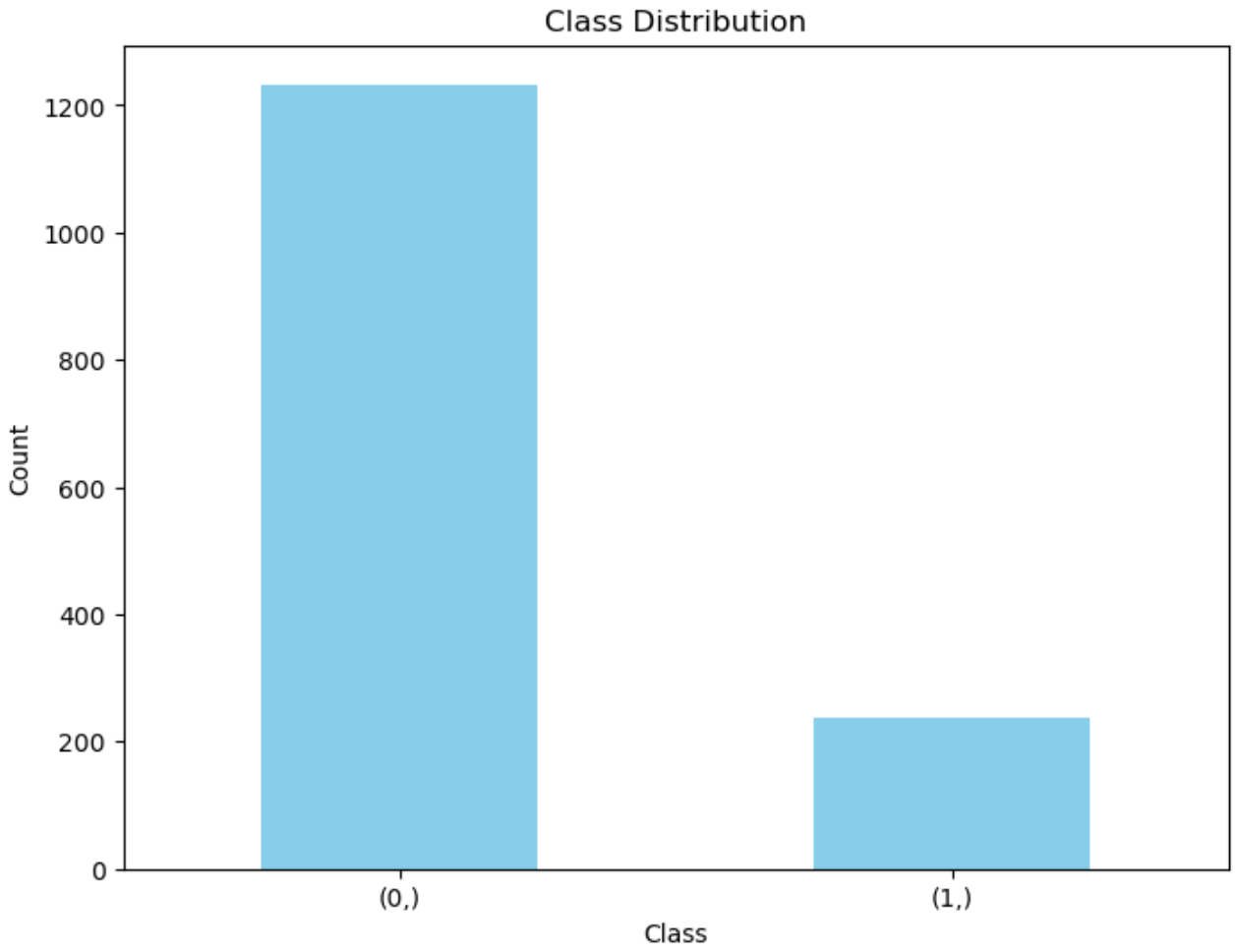
	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education
0	41	Yes	Travel_Rarely	1102	Sales	1	2
1	49	No	Travel_Frequently	279	Research & Development	8	1
2	37	Yes	Travel_Rarely	1373	Research & Development	2	2
3	33	No	Travel_Frequently	1392	Research & Development	3	4
4	27	No	Travel_Rarely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

5 rows × 35 columns

In [3]:
```
y = df[["Attrition"]].copy()
X = df.drop("Attrition", axis = 1)
```

In [4]:
```
y["Attrition"] = [1 if i == "Yes" else 0 for i in y["Attrition"]]
```

In [5]:
```
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_fir
```

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
```

In [11]:
```
#overfitting with the insample data
```

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 [12]:
```
# 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, try
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

#test through cv=5 all the values from all the values in the parameters on t
dt_classifier = DecisionTreeClassifier(random_state=42)

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

grid_search = GridSearchCV(estimator=dt_classifier, param_grid=param_grid, s

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)
```

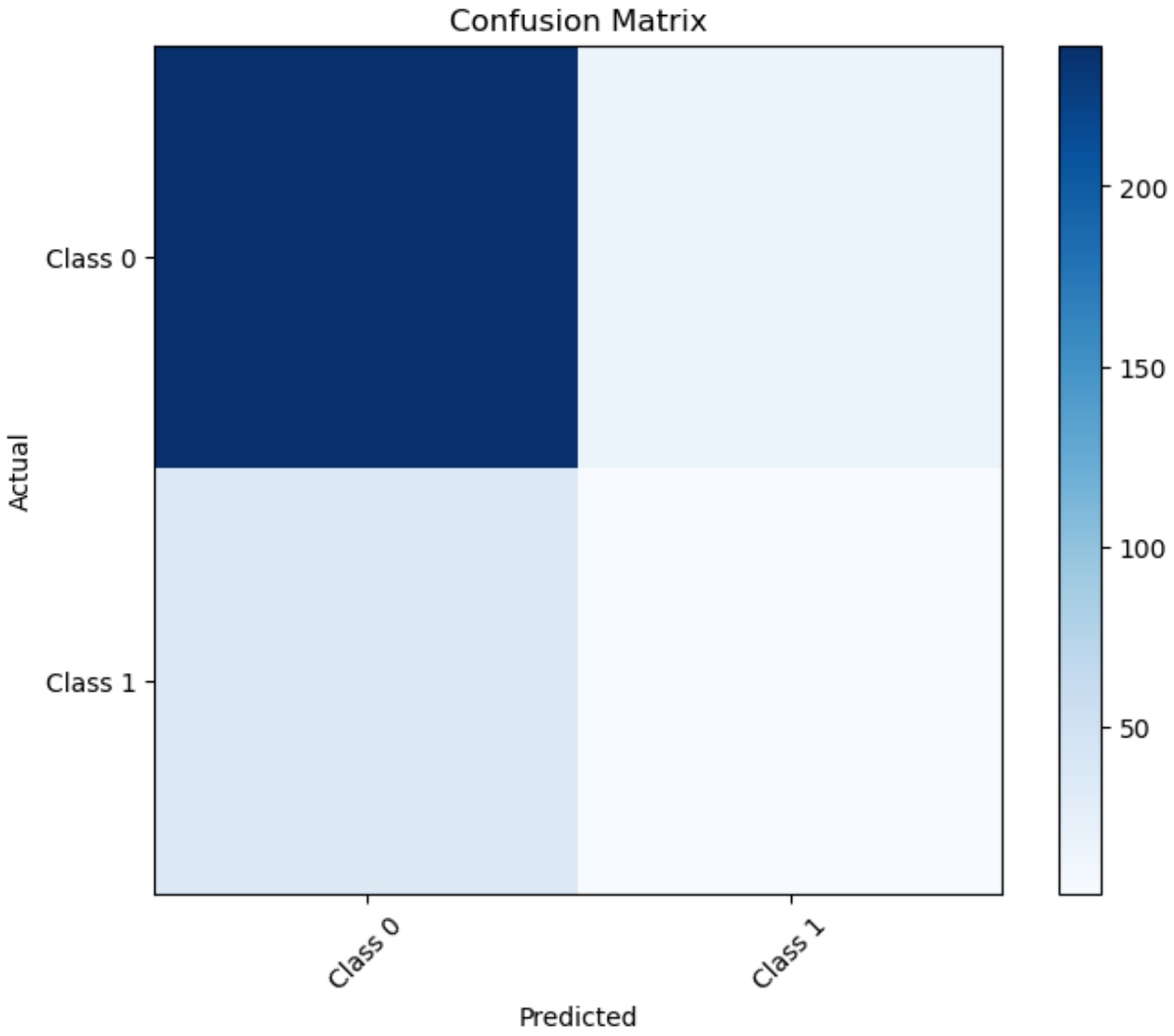
```
Best Parameters: {'criterion': 'gini', 'max_depth': 6, 'min_samples_leaf': 2
, 'min_samples_split': 2}
Best F1-Score: 0.8214764475510983
```

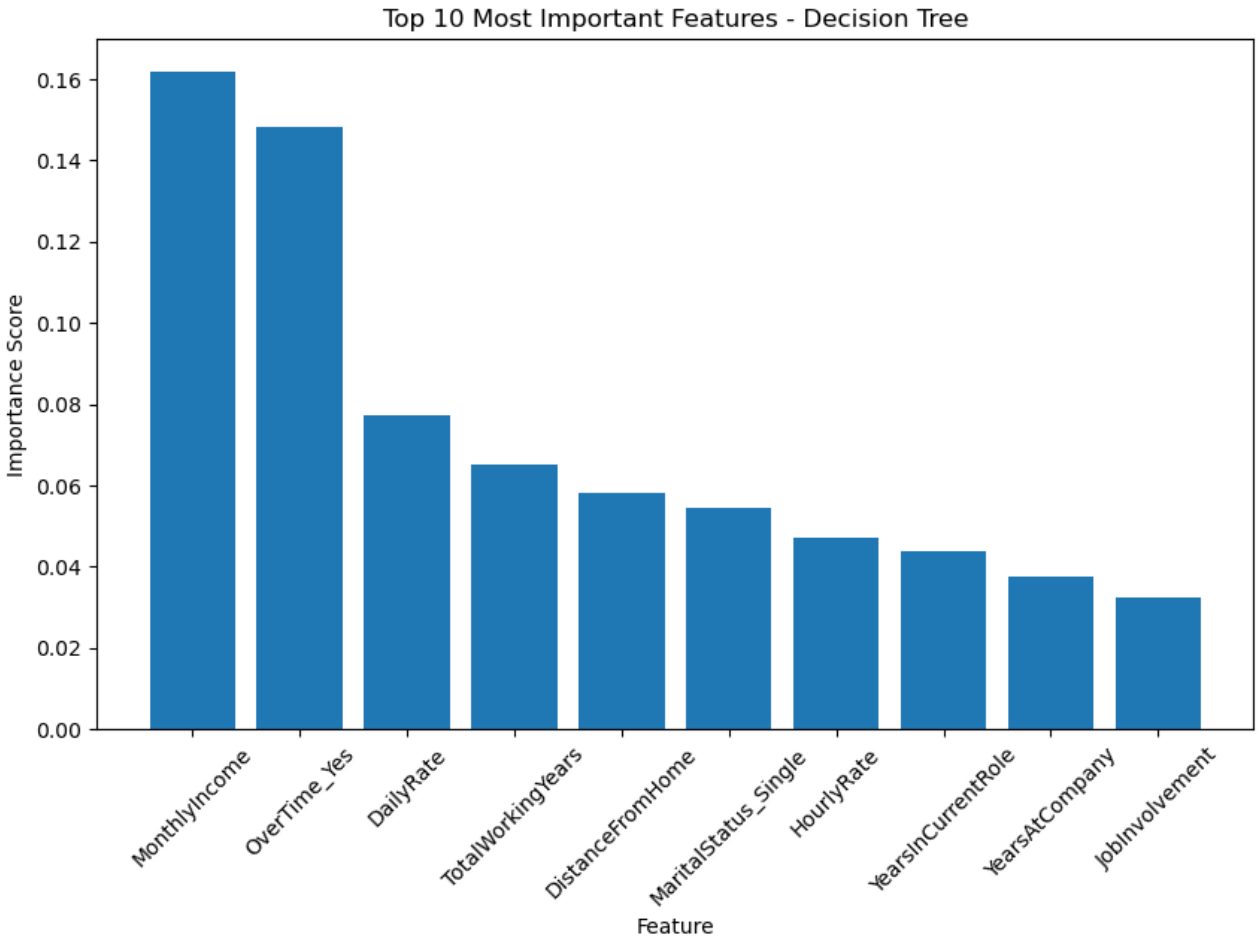
In [13]:
```
clf = tree.DecisionTreeClassifier(**best_params, random_state =42)
clf.fit(x_train,y_train)
y_pred=clf.predict(x_train)
acc=accuracy_score(y_train,y_pred)
print("IN SAMPLE ACCURACY : " , round(acc,2))

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 [15]:
```
# 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_
#what is getting split first

# 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=(12, 6))
plot_tree(clf, filled=True, feature_names=X.columns, class_names=["Yes", "Nc
plt.title('Decision Tree Classifier')
plt.show()
```


Confusion Matrix
Class 0
Class 1
Actual
Class 0
Class 1
Predicted
200
150
100
50




```
---------------------------------------------------------------------------
InvalidParameterError                     Traceback (most recent call last)
Cell In[15], line 43
     41 # Plot the Decision Tree for better visualization of the selected fe
atures
     42 plt.figure(figsize=(12, 6))
---> 43 plot_tree(clf, filled=True, feature_names=X.columns, class_names=["Y
es", "No"], rounded=True, fontsize=7)
     44 plt.title('Decision Tree Classifier')
     45 plt.show()

File ~/anaconda3/lib/python3.11/site-packages/sklearn/utils/_param_validatio
n.py:201, in validate_params.<locals>.decorator.<locals>.wrapper(*args, **kw
args)
    198 to_ignore += ["self", "cls"]
    199 params = {k: v for k, v in params.arguments.items() if k not in
to_ignore}
--> 201 validate_parameter_constraints(
    202     parameter_constraints, params, caller_name=func.__qualname__
    203 )
    205 try:
    206     with config_context(
    207         skip_parameter_validation=(
    208             prefer_skip_nested_validation or global_skip_validation
```

```
    209         )
    210     ):

File ~/anaconda3/lib/python3.11/site-packages/sklearn/utils/_param_validatio
n.py:95, in validate_parameter_constraints(parameter_constraints, params, ca
ller_name)
     89 else:
     90     constraints_str = (
     91         f"{', '.join([str(c) for c in constraints[:-1]])} or"
     92         f" {constraints[-1]}"
     93     )
---> 95 raise InvalidParameterError(
     96     f"The {param_name!r} parameter of {caller_name} must be"
     97     f" {constraints_str}. Got {param_val!r} instead."
     98 )

InvalidParameterError: The 'feature_names' parameter of plot_tree must be an
instance of 'list' or None. Got Index(['Age', 'DailyRate', 'DistanceFromHome
', 'Education', 'EmployeeCount',
       'EmployeeNumber', 'EnvironmentSatisfaction', 'HourlyRate',
       'JobInvolvement', 'JobLevel', 'JobSatisfaction', 'MonthlyIncome',
       'MonthlyRate', 'NumCompaniesWorked', 'PercentSalaryHike',
       'PerformanceRating', 'RelationshipSatisfaction', 'StandardHours',
       'StockOptionLevel', 'TotalWorkingYears', 'TrainingTimesLastYear',
       'WorkLifeBalance', 'YearsAtCompany', 'YearsInCurrentRole',
       'YearsSinceLastPromotion', 'YearsWithCurrManager',
       'BusinessTravel_Travel_Frequently', 'BusinessTravel_Travel_Rarely',
       'Department_Research & Development', 'Department_Sales',
       'EducationField_Life Sciences', 'EducationField_Marketing',
       'EducationField_Medical', 'EducationField_Other',
       'EducationField_Technical Degree', 'Gender_Male',
       'JobRole_Human Resources', 'JobRole_Laboratory Technician',
       'JobRole_Manager', 'JobRole_Manufacturing Director',
       'JobRole_Research Director', 'JobRole_Research Scientist',
       'JobRole_Sales Executive', 'JobRole_Sales Representative',
       'MaritalStatus_Married', 'MaritalStatus_Single', 'OverTime_Yes'],
      dtype='object') instead.
```

```
<Figure size 1200x600 with 0 Axes>
```

In [ ]:

In [ ]:

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

ANSWER :

In [16]:
```
#According to the feature importance plot, intuitively it is better to incre
#overtime in order to keep employee in the company.
#But the thing is the fearture importance cannot tell us the direction of th
```

In [ ]:
```
from scipy.stats import pearsonr
```

In [ ]:
```
def calculate_correlation(X,feature_name,y):
    feature=X[feature_name]
    coef,_ =pearsonr(feature,y)
    return(coef)
```

In [20]:
```
np.corrcoef(np.array(X["OverTime_Yes"]),np.array(y["Attrition"]))
```

Out[20]:
```
array([[1.        , 0.24611799],
       [0.24611799, 1.        ]])
```

In [ ]:
```
calculate_correlation(x_train,"Monthly Income",y_train)
```

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

In [22]:
```
x_train_experiment=x_train.copy()
```

In [23]:
```
x_train_experiment['OverTime_Yes']=0
```

In [24]:
```
y_pred=clf.predict(x_train)
y_pred_experiment=clf.predict(x_train_experiment)
```

In [25]:
```
diff=sum(y_pred_experiment-y_pred)
```

In [26]:
```
print(diff)
```

```
-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 [44]:
```
x_train_experiment["Y"]=y_pred
x_train_experiment["Y_exp"]=y_pred_experiment
```

In [45]:
```
x_train_experiment["RetChange"]=x_train_experiment["Y_exp"]-x_train_experime
```

In [46]:
```
sav=sum(-2.8*x_train_experiment["RetChange"]*x_train_experiment["MonthlyInco
```

In [47]:
```
cost=len(x_train[x_train["OverTime_Yes"]==1])*2000 #having them not work ove
```

In [48]:
```
sav-cost
```

Out[48]:
```
-474599.6000000001
```

ANSWER :

In [ ]:
```
#It is way more profitable to keep them working overtime.
```

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 [36]:
```
raise_amount=100
```

In [50]:
```
x_train_experiment=x_train.copy()
x_train_experiment['MonthlyIncome']=x_train_experiment['MonthlyIncome']+rais
y_pred=clf.predict(x_train)
y_pred_experiment=clf.predict(x_train_experiment)
diff=sum(y_pred_experiment-y_pred)
print(diff)
x_train_experiment["Y"]=y_pred
x_train_experiment["Y_exp"]=y_pred_experiment
x_train_experiment["RetChange"]=x_train_experiment["Y_exp"]-x_train_experime
sav=sum(-2.8*x_train_experiment["RetChange"]*x_train_experiment["MonthlyInco
cost=len(x_train)*raise_amount
print("Profits",sav-cost)
```

```
-23
Profits -854999.6000000001
```

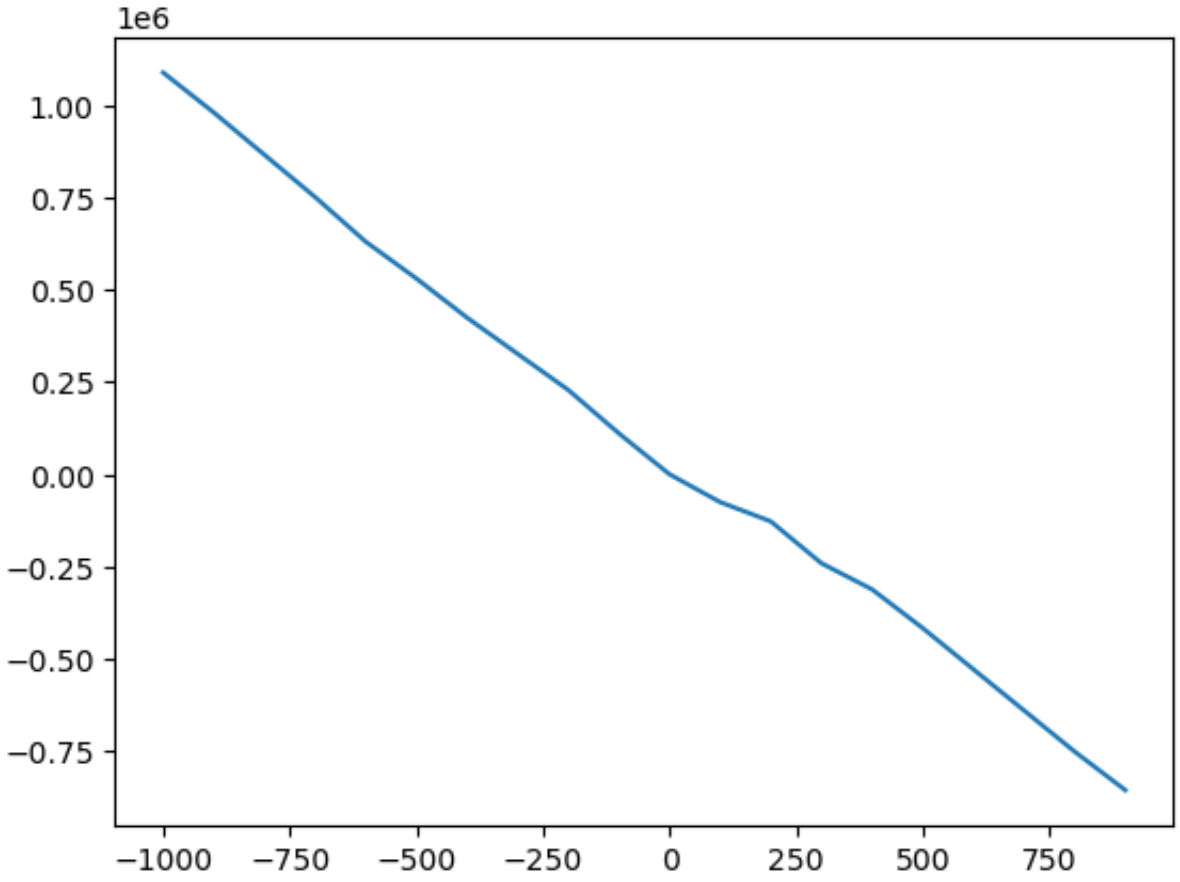
In [58]:
```
profits=[]
for raise_amount in range(-1000,1000,100):
    x_train_experiment=x_train.copy()
    x_train_experiment['MonthlyIncome']=x_train_experiment['MonthlyIncome']+
    y_pred=clf.predict(x_train)
    y_pred_experiment=clf.predict(x_train_experiment)
    diff=sum(y_pred_experiment-y_pred)
    x_train_experiment["Y"]=y_pred
    x_train_experiment["Y_exp"]=y_pred_experiment
    x_train_experiment["RetChange"]=x_train_experiment["Y_exp"]-x_train_expe
    sav=sum(-2.8*x_train_experiment["RetChange"]*x_train_experiment["Monthly
    cost=len(x_train)*raise_amount
    print("Profits",sav-cost)
    profits.append(sav-cost)
```

```
Profits 1087584.4
Profits 979524.0
Profits 864992.8
Profits 750738.8
Profits 629778.8
Profits 530138.0
Profits 424200.0
Profits 326096.4
Profits 228440.8
Profits 110714.8
Profits 0.0
Profits -75328.40000000001
Profits -127503.60000000002
Profits -240914.8
Profits -311586.80000000005
Profits -416449.6000000001
Profits -527889.6000000001
Profits -639329.6000000001
Profits -750769.6000000001
Profits -854999.6000000001
```

In [59]:
```
print(profits)
```

```
[1087584.4, 979524.0, 864992.8, 750738.8, 629778.8, 530138.0, 424200.0, 3260
96.4, 228440.8, 110714.8, 0.0, -75328.40000000001, -127503.60000000002, -240
914.8, -311586.80000000005, -416449.6000000001, -527889.6000000001, -639329.
6000000001, -750769.6000000001, -854999.6000000001]
```

In [60]:
```
plt.plot(range(-1000,1000,100), profits)
plt.show()
```



ANSWER :

In [ ]:
```
#This means even though the company raises the salary, there is still cost m
#to just keep workers working overtime.
```