In [1]:
```python
import pandas as pd

# Load the dataset with the correct delimiter
data = pd.read_csv("bank.csv", sep=";")

# Display the column names
print(data.columns)
```

```
Index(['age', 'job', 'marital', 'education', 'default', 'balance', 'housing',
       'loan', 'contact', 'day', 'month', 'duration', 'campaign', 'pdays',
       'previous', 'poutcome', 'y'],
      dtype='object')
```

In [5]:
```python
# Features and target
X = data.drop("y", axis=1)  # Drop the target column
y = data["y"]               # Target variable
```

In [7]:
```python
import pandas as pd

# Load dataset
data = pd.read_csv("bank.csv", sep=";")

# Clean column names
data.columns = data.columns.str.strip().str.replace('"', '')

# Display the updated column names
print(data.columns)

# Features and target
X = data.drop("y", axis=1)  # Drop target column
y = data["y"]               # Extract target variable
```

```
Index(['age', 'job', 'marital', 'education', 'default', 'balance', 'housing',
       'loan', 'contact', 'day', 'month', 'duration', 'campaign', 'pdays',
       'previous', 'poutcome', 'y'],
      dtype='object')
```

In [9]:
```python
from sklearn.model_selection import train_test_split

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

print("Training features shape:", X_train.shape)
print("Testing features shape:", X_test.shape)
```

```
Training features shape: (3616, 16)
Testing features shape: (905, 16)
```

In [13]:
```python
from sklearn.preprocessing import LabelEncoder

# Encode categorical columns
encoder = LabelEncoder()
for col in X.select_dtypes(include="object").columns:
    X[col] = encoder.fit_transform(X[col])
```

In [15]:

```
X = pd.get_dummies(X, drop_first=True)b
```

In [21]:

```
categorical_columns = X.select_dtypes(include=["object"]).columns
print("Categorical columns:", categorical_columns)
```

```
Categorical columns: Index([], dtype='object')
```

In [23]:

```
X_encoded = pd.get_dummies(X, drop_first=True)
print(X_encoded.head())
```

```
   age  job  marital  education  default  balance  housing  loan  contact  \
0   30   10        1          0        0     1787        0     0        0   
1   33    7        1          1        0     4789        1     1        0   
2   35    4        2          2        0     1350        1     0        0   
3   30    4        1          2        0     1476        1     1        2   
4   59    1        1          1        0        0        1     0        2   

   day  month  duration  campaign  pdays  previous  poutcome  
0   19     10        79         1     -1         0         3  
1   11      8       220         1    339         4         0  
2   16      0       185         1    330         1         0  
3    3      6       199         4     -1         0         3  
4    5      8       226         1     -1         0         3  
```

In [25]:

```
print(X.dtypes)
```

```
age          int64
job          int32
marital      int32
education    int32
default      int32
balance      int64
housing      int32
loan         int32
contact      int32
day          int64
month        int32
duration     int64
campaign     int64
pdays        int64
previous     int64
poutcome     int32
dtype: object
```

In [33]:

```
import pandas as pd

X_encoded = pd.get_dummies(X, drop_first=True)  # Encodes and avoids multicollinear
print(X_encoded.head())
```

```
   age  job  marital  education  default  balance  housing  loan  contact  \
0   30   10        1          0        0     1787        0     0        0   
1   33    7        1          1        0     4789        1     1        0   
2   35    4        2          2        0     1350        1     0        0   
3   30    4        1          2        0     1476        1     1        2   
4   59    1        1          1        0        0        1     0        2   

   day  month  duration  campaign  pdays  previous  poutcome  
0   19     10        79         1     -1         0         3  
1   11      8       220         1    339         4         0  
2   16      0       185         1    330         1         0  
3    3      6       199         4     -1         0         3  
4    5      8       226         1     -1         0         3  
```

In [37]:

```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X_encoded, y, test_size=0.3, ra

# Train the model
model = DecisionTreeClassifier()
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate
accuracy = accuracy_score(y_test, y_pred)
print("Model Accuracy:", accuracy)
```

```
Model Accuracy: 0.8629329403095063
```

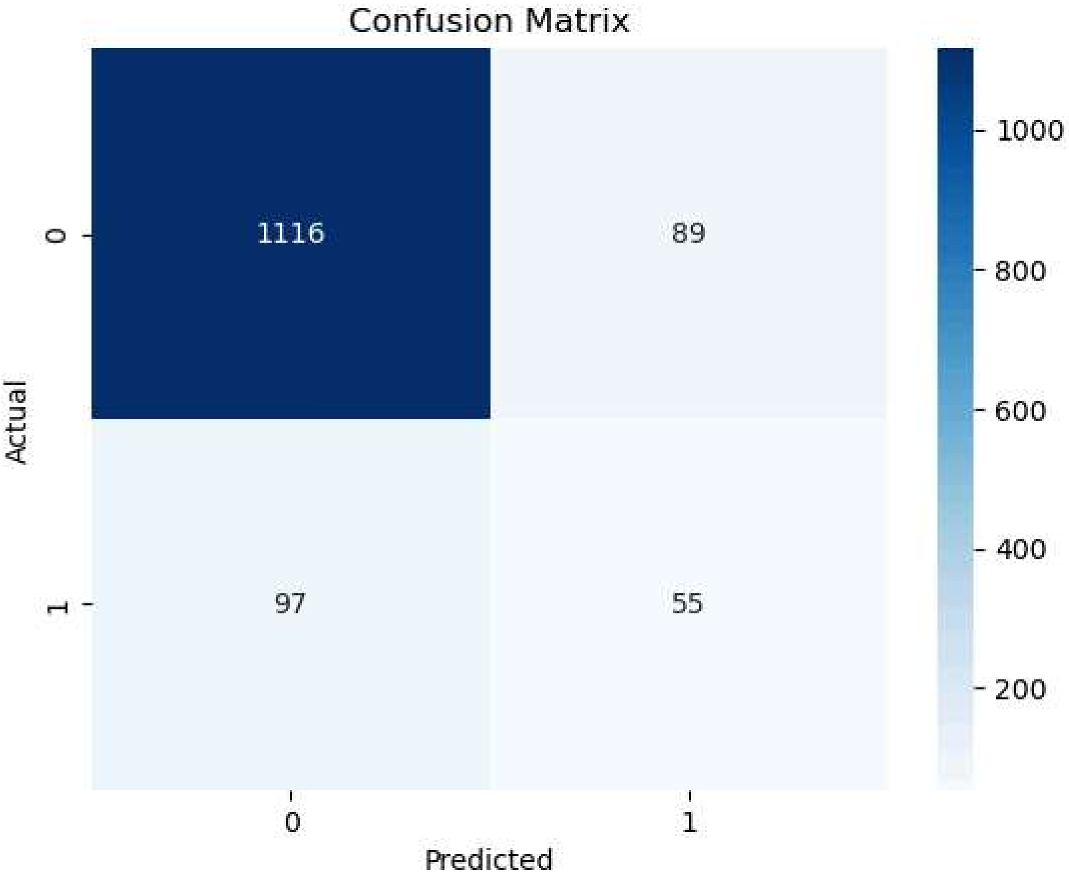
In [39]:

```
from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

          no       0.92      0.93      0.92      1205
         yes       0.38      0.36      0.37       152

    accuracy                           0.86      1357
   macro avg       0.65      0.64      0.65      1357
weighted avg       0.86      0.86      0.86      1357
```

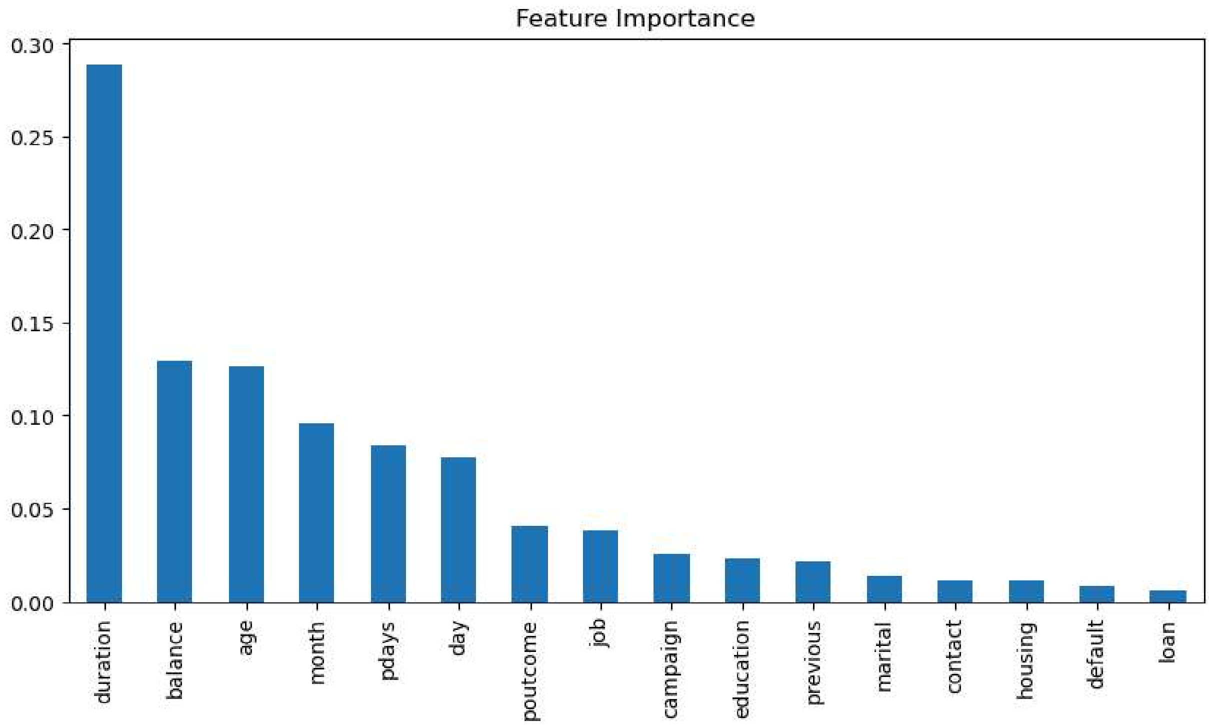
In [41]:

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()
```



In [43]:

```
import pandas as pd
import matplotlib.pyplot as plt

# Get feature importances
feature_importances = pd.Series(model.feature_importances_, index=X_encoded.columns

# Sort and plot
feature_importances = feature_importances.sort_values(ascending=False)
feature_importances.plot(kind="bar", figsize=(10, 5), title="Feature Importance")
plt.show()
```



In [45]:
```
from sklearn.model_selection import GridSearchCV

# Define parameter grid
param_grid = {
    "max_depth": [3, 5, 10, None],
    "min_samples_split": [2, 5, 10],
    "min_samples_leaf": [1, 2, 4]
}

# Grid Search
grid_search = GridSearchCV(DecisionTreeClassifier(), param_grid, cv=5, scoring="acc
grid_search.fit(X_train, y_train)

# Best parameters
print("Best Parameters:", grid_search.best_params_)

# Best model
best_model = grid_search.best_estimator_
```

```
Best Parameters: {'max_depth': 5, 'min_samples_leaf': 4, 'min_samples_split': 5}
```

In [47]:
```
from sklearn.model_selection import RandomizedSearchCV

random_search = RandomizedSearchCV(DecisionTreeClassifier(), param_grid, n_iter=10,
random_search.fit(X_train, y_train)

print("Best Parameters:", random_search.best_params_)
```

```
Best Parameters: {'min_samples_split': 5, 'min_samples_leaf': 4, 'max_depth': 5}
```

In [49]:
```
import joblib

# Save model
joblib.dump(model, "decision_tree_model.pkl")
```

```python
# Load model
loaded_model = joblib.load("decision_tree_model.pkl")
```

In [51]:

```python
from sklearn.ensemble import RandomForestClassifier

rf_model = RandomForestClassifier()
rf_model.fit(X_train, y_train)
y_pred_rf = rf_model.predict(X_test)
print("Random Forest Accuracy:", accuracy_score(y_test, y_pred_rf))
```

```
Random Forest Accuracy: 0.8975681650700074
```

In [ ]: