Predicting Term Deposit Suscriptions

In [68]:

```
from IPython.display import Image
Image("G:/ML portfolio projects/Own Projects\Predicting Term Deposit Suscriptions/1.png")
```

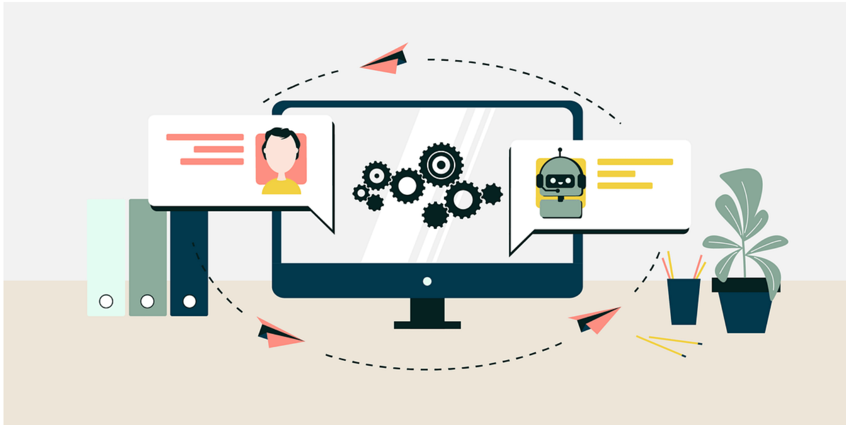
Out[68]:

Data Exploration and Preprocessing

In [70]:

```
Image("G:/ML portfolio projects/Own Projects\Predicting Term Deposit Suscriptions/3.png")
```

Out[70]:



- Import the required libraries

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

- Load the dataset into a Pandas DataFrame

In [2]:

```
df = pd.read_csv('bank.csv')
```

- Inspect the first few rows of the DataFrame to get a glimpse of the data

In [3]:

```
df.head()
```

Out[3]:

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration
0	59	admin.	married	secondary	no	2343	yes	no	unknown	5	may	1042
1	56	admin.	married	secondary	no	45	no	no	unknown	5	may	1467
2	41	technician	married	secondary	no	1270	yes	no	unknown	5	may	1389
3	55	services	married	secondary	no	2476	yes	no	unknown	5	may	579
4	54	admin.	married	tertiary	no	184	no	no	unknown	5	may	673

- Handle missing values:

Identify missing values in the dataset

In [4]:

```
df.isnull().sum()
```

Out[4]:

```
age          0
job          0
marital      0
education    0
default      0
balance      0
housing      0
loan         0
contact      0
day          0
month        0
duration     0
campaign     0
pdays        0
previous     0
poutcome     0
deposit      0
dtype: int64
```

- Encode categorical variables:

Identify categorical columns-

In [5]:

```
categorical_cols = df.select_dtypes(include=['object']).columns
```

- Convert categorical variables into numerical representations using one-hot encoding or label encoding. For one-hot encoding

In [6]:

```
df_encoded = pd.get_dummies(df, columns=categorical_cols, drop_first=True)
```

- Perform exploratory data analysis (EDA):

Inspect the first few rows of the DataFrame to get a glimpse of the data

In [8]:

```
df_encoded.head()
```

Out[8]:

	age	balance	day	duration	campaign	pdays	previous	job_blue-collar	job_entrepreneur	job_housemaid
0	59	2343	5	1042	1	-1	0	0	0	0
1	56	45	5	1467	1	-1	0	0	0	0
2	41	1270	5	1389	1	-1	0	0	0	0
3	55	2476	5	579	1	-1	0	0	0	0
4	54	184	5	673	2	-1	0	0	0	0

5 rows × 43 columns

- Check the dimensions of the dataset (number of rows and columns)

In [9]:

```
df_encoded.shape
```

Out[9]:

```
(11162, 43)
```

- Explore the summary statistics of numerical features

In [10]:

```python
df_encoded.describe()
```

Out[10]:

I	...	month_jun	month_mar	month_may	month_nov	month_oct	month_sep	poutcome_other
)	...	11162.000000	11162.000000	11162.000000	11162.000000	11162.000000	11162.000000	11162.000000
3	...	0.109479	0.024727	0.253001	0.084483	0.035119	0.028579	0.048110
)	...	0.312253	0.155298	0.434751	0.278123	0.184089	0.166628	0.214008
)	...	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
)	...	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
)	...	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
)	...	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000
)	...	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

- Check the data types of each column

In [11]:

```python
df_encoded.dtypes
```

Out[11]:

```
age                    int64
balance                int64
day                    int64
duration               int64
campaign               int64
pdays                  int64
previous               int64
job_blue-collar        uint8
job_entrepreneur       uint8
job_housemaid          uint8
job_management         uint8
job_retired            uint8
job_self-employed      uint8
job_services           uint8
job_student            uint8
job_technician         uint8
job_unemployed         uint8
job_unknown            uint8
marital_married        uint8
marital_single         uint8
education_secondary    uint8
education_tertiary     uint8
education_unknown      uint8
default_yes            uint8
housing_yes            uint8
loan_yes               uint8
contact_telephone      uint8
contact_unknown        uint8
month_aug              uint8
month_dec              uint8
month_feb              uint8
month_jan              uint8
month_jul              uint8
month_jun              uint8
month_mar              uint8
month_may              uint8
month_nov              uint8
month_oct              uint8
month_sep              uint8
poutcome_other         uint8
poutcome_success       uint8
poutcome_unknown       uint8
deposit_yes            uint8
dtype: object
```

- Analyze the distribution of the target variable ('deposit_yes' column)

In [13]:

```python
deposit_yes['deposit_yes'].value_counts()
```

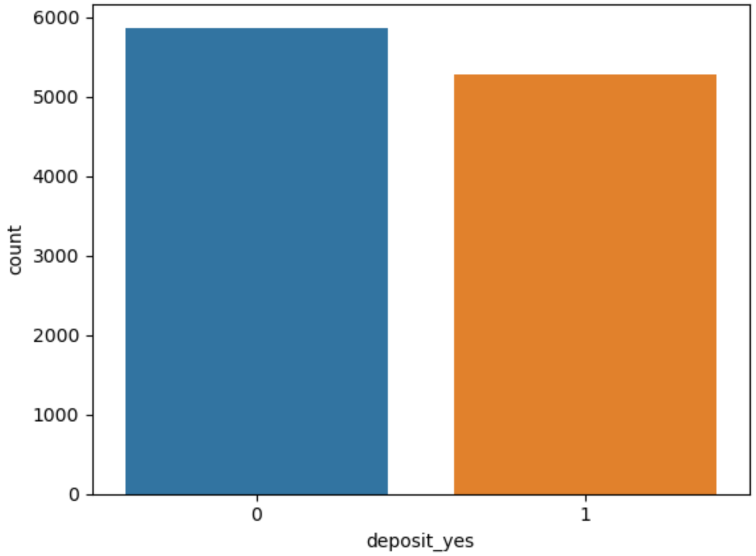
Out[13]:

```
0    5873
1    5289
Name: deposit_yes, dtype: int64
```

- Visualize the distribution of the target variable using a bar plot

In [16]:

```
sns.countplot(x='deposit_yes', data=df_encoded)
plt.show()
```



- Split the dataset into training and testing sets

In [17]:

```
from sklearn.model_selection import train_test_split

X = df_encoded.drop('deposit_yes', axis=1)  # Features (excluding the target variable)
y = df_encoded['deposit_yes']  # Target variable

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Model Selection and Training

In [72]:

```
Image("G:/ML portfolio projects/Own Projects\Predicting Term Deposit Suscriptions/4.jpg")
```

Out[72]:

- Import the necessary libraries

In [18]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_scor
```

- Initialize the models

In [19]:

```
logistic_regression = LogisticRegression()
decision_tree = DecisionTreeClassifier()
random_forest = RandomForestClassifier()
gradient_boosting = GradientBoostingClassifier()
support_vector_machine = SVC()
```

- Train the models on the training set

In [21]:

```
logistic_regression.fit(X_train, y_train)
```

```
C:\Users\SOMNATH\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:460:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-lear
n.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
(https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
```

Out[21]:

```
▾ LogisticRegression
LogisticRegression()
```

In [22]:

```
decision_tree.fit(X_train, y_train)
```

Out[22]:

```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

In [23]:

```
random_forest.fit(X_train, y_train)
```

Out[23]:

```
▾ RandomForestClassifier
RandomForestClassifier()
```

In [24]:

```
gradient_boosting.fit(X_train, y_train)
```

Out[24]:

```
▾ GradientBoostingClassifier
GradientBoostingClassifier()
```

In [25]:

```
support_vector_machine.fit(X_train, y_train)
```

Out[25]:

```
▾ SVC
SVC()
```

- Make predictions on the training set

In [26]:

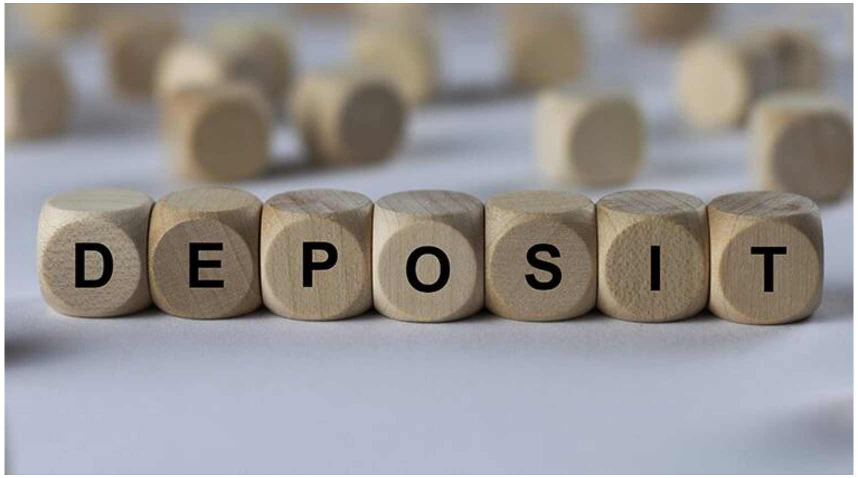
```
logistic_regression_train_preds = logistic_regression.predict(X_train)
decision_tree_train_preds = decision_tree.predict(X_train)
random_forest_train_preds = random_forest.predict(X_train)
gradient_boosting_train_preds = gradient_boosting.predict(X_train)
support_vector_machine_train_preds = support_vector_machine.predict(X_train)
```

- Evaluate the performance of each model on the training set

In [74]:

```
Image("G:/ML portfolio projects/Own Projects\Predicting Term Deposit Suscriptions/5.jpg")
```

Out[74]:



In [29]:

```
print("Training Set Metrics:")
print("Logistic Regression:")
print("Accuracy:", accuracy_score(y_train, logistic_regression_train_preds))
print("Precision:", precision_score(y_train, logistic_regression_train_preds))
print("Recall:", recall_score(y_train, logistic_regression_train_preds))
print("F1-Score:", f1_score(y_train, logistic_regression_train_preds))
print("ROC-AUC:", roc_auc_score(y_train, logistic_regression_train_preds))

# Repeat the same evaluation steps for the other models
```

```
Training Set Metrics:
Logistic Regression:
Accuracy: 0.8007615634449546
Precision: 0.7691121392377176
Recall: 0.8268593083846518
F1-Score: 0.7969409884716357
ROC-AUC: 0.8021060935379813
```

In [ ]:

```
print("Training Set Metrics:")
print("Decision Tree:")
print("Accuracy:", accuracy_score(y_train, decision_tree_train_preds))
print("Precision:", precision_score(y_train, decision_tree_train_preds))
print("Recall:", recall_score(y_train, decision_tree_train_preds))
print("F1-Score:", f1_score(y_train, decision_tree_train_preds))
print("ROC-AUC:", roc_auc_score(y_train, decision_tree_train_preds))
```

In [30]:

```
print("Training Set Metrics:")
print("Random Forest:")
print("Accuracy:", accuracy_score(y_train, random_forest_train_preds))
print("Precision:", precision_score(y_train, random_forest_train_preds))
print("Recall:", recall_score(y_train, random_forest_train_preds))
print("F1-Score:", f1_score(y_train, random_forest_train_preds))
print("ROC-AUC:", roc_auc_score(y_train, random_forest_train_preds))
```

```
Training Set Metrics:
Random Forest:
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
F1-Score: 1.0
ROC-AUC: 1.0
```

In [31]:

```
print("Training Set Metrics:")
print("Gradient Boosting:")
print("Accuracy:", accuracy_score(y_train, gradient_boosting_train_preds))
print("Precision:", precision_score(y_train, gradient_boosting_train_preds))
print("Recall:", recall_score(y_train, gradient_boosting_train_preds))
print("F1-Score:", f1_score(y_train, gradient_boosting_train_preds))
print("ROC-AUC:", roc_auc_score(y_train, gradient_boosting_train_preds))
```

```
Training Set Metrics:
Gradient Boosting:
Accuracy: 0.8625825960353903
Precision: 0.8404182768811094
Recall: 0.8756513500710563
F1-Score: 0.8576731237675443
ROC-AUC: 0.863255885360576
```

In [32]:

```
print("Training Set Metrics:")
print("Support Vector Machine:")
print("Accuracy:", accuracy_score(y_train, support_vector_machine_train_preds))
print("Precision:", precision_score(y_train, support_vector_machine_train_preds))
print("Recall:", recall_score(y_train, support_vector_machine_train_preds))
print("F1-Score:", f1_score(y_train, support_vector_machine_train_preds))
print("ROC-AUC:", roc_auc_score(y_train, support_vector_machine_train_preds))
```

```
Training Set Metrics:
Support Vector Machine:
Accuracy: 0.7437562996976145
Precision: 0.7814318975552969
Recall: 0.6359545239223117
F1-Score: 0.7012274745364325
ROC-AUC: 0.7382024584769833
```

Model Evaluation and Comparison

In [75]:

```
Image("G:/ML portfolio projects/Own Projects\Predicting Term Deposit Suscriptions/6.jpg")
```

Out[75]:

- Import the necessary libraries

In [34]:

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_scor
```

- Define a list to store the evaluation metrics for each model

In [35]:

```
models = [logistic_regression, decision_tree, random_forest, gradient_boosting, support_vector_ma
model_names = ['Logistic Regression', 'Decision Tree', 'Random Forest', 'Gradient Boosting', 'Sup
accuracy_scores = []
precision_scores = []
recall_scores = []
f1_scores = []
roc_auc_scores = []
```

- Iterate over the models to evaluate their performance on the test set

In [36]:

```
for model in models:
    preds = model.predict(X_test)
    accuracy_scores.append(accuracy_score(y_test, preds))
    precision_scores.append(precision_score(y_test, preds))
    recall_scores.append(recall_score(y_test, preds))
    f1_scores.append(f1_score(y_test, preds))
    roc_auc_scores.append(roc_auc_score(y_test, preds))
```

- Create a dataframe to compare the evaluation metrics of different models

In [37]:

```
evaluation_df = pd.DataFrame({
    'Model': model_names,
    'Accuracy': accuracy_scores,
    'Precision': precision_scores,
    'Recall': recall_scores,
    'F1-Score': f1_scores,
    'ROC-AUC': roc_auc_scores
})
```

- Print or visualize the evaluation metrics for model comparison

In [38]:

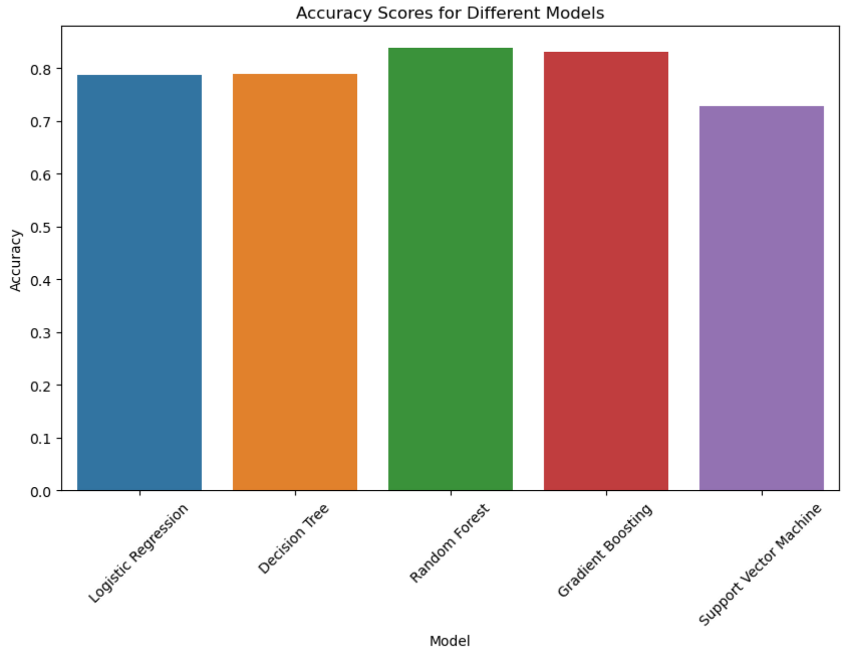
```
print(evaluation_df)
```

```
                    Model  Accuracy  Precision    Recall  F1-Score   ROC-AUC
0     Logistic Regression  0.786386   0.756522  0.815370  0.784844  0.787616
1           Decision Tree  0.789521   0.780809  0.777882  0.779343  0.789027
2           Random Forest  0.837438   0.808772  0.864105  0.835523  0.838570
3       Gradient Boosting  0.831169   0.815934  0.835052  0.825382  0.831334
4  Support Vector Machine  0.726825   0.750823  0.641050  0.691608  0.723183
```

- create visualizations such as bar plots or line plots to compare the evaluation metrics of different models.

In [39]:

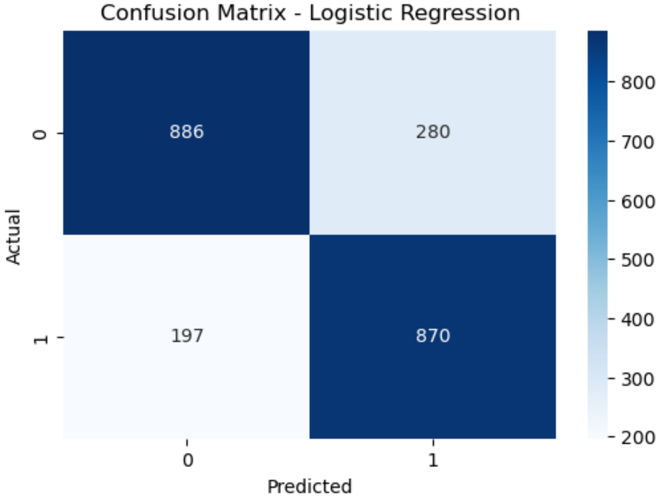
```
plt.figure(figsize=(10, 6))
sns.barplot(x='Model', y='Accuracy', data=evaluation_df)
plt.xlabel('Model')
plt.ylabel('Accuracy')
plt.title('Accuracy Scores for Different Models')
plt.xticks(rotation=45)
plt.show()
```

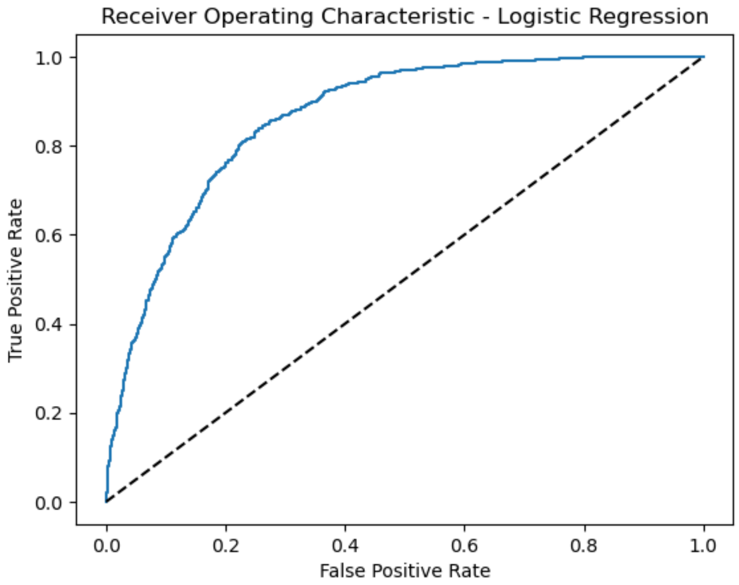


- Logistic Regression

In [40]:

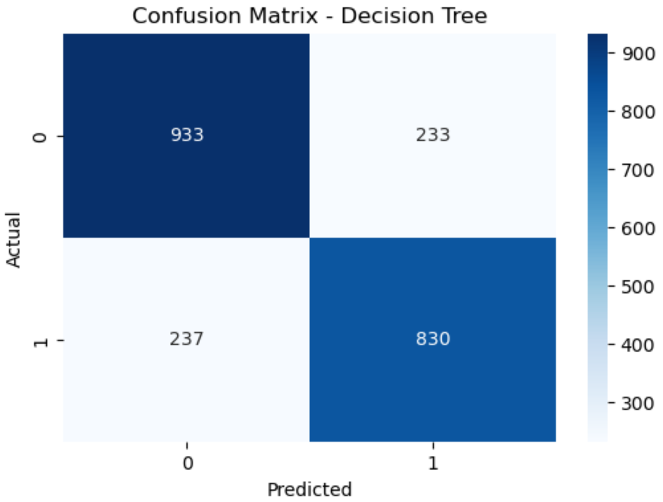
```
from sklearn.metrics import confusion_matrix
import seaborn as sns

preds = logistic_regression.predict(X_test)
cm = confusion_matrix(y_test, preds)

plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, cmap='Blues', fmt='d')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - Logistic Regression')
plt.show()

from sklearn.metrics import roc_curve

preds = logistic_regression.predict_proba(X_test)[:, 1]  # Probability of positive class
fpr, tpr, thresholds = roc_curve(y_test, preds)

plt.plot(fpr, tpr)
plt.plot([0, 1], [0, 1], 'k--')  # Diagonal line for random classifier
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic - Logistic Regression')
plt.show()
```

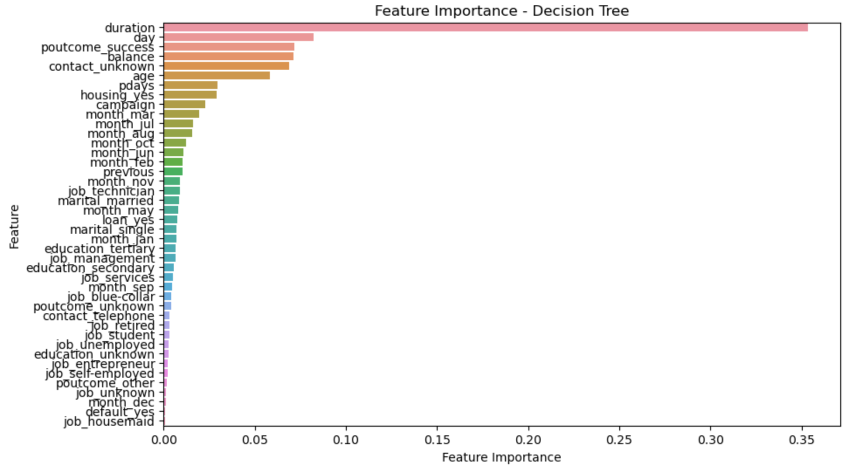




In [41]:

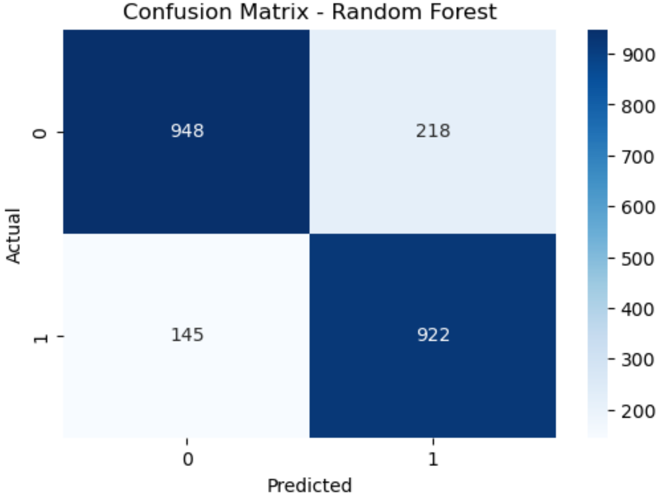
```
from sklearn.metrics import confusion_matrix
import seaborn as sns

preds = decision_tree.predict(X_test)
cm = confusion_matrix(y_test, preds)

plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, cmap='Blues', fmt='d')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - Decision Tree')
plt.show()

feature_importances = decision_tree.feature_importances_
feature_names = X.columns

feature_importance_df = pd.DataFrame({'Feature': feature_names, 'Importance': feature_importances
feature_importance_df = feature_importance_df.sort_values('Importance', ascending=False)

plt.figure(figsize=(10, 6))
sns.barplot(x='Importance', y='Feature', data=feature_importance_df)
plt.xlabel('Feature Importance')
plt.ylabel('Feature')
plt.title('Feature Importance - Decision Tree')
plt.show()
```

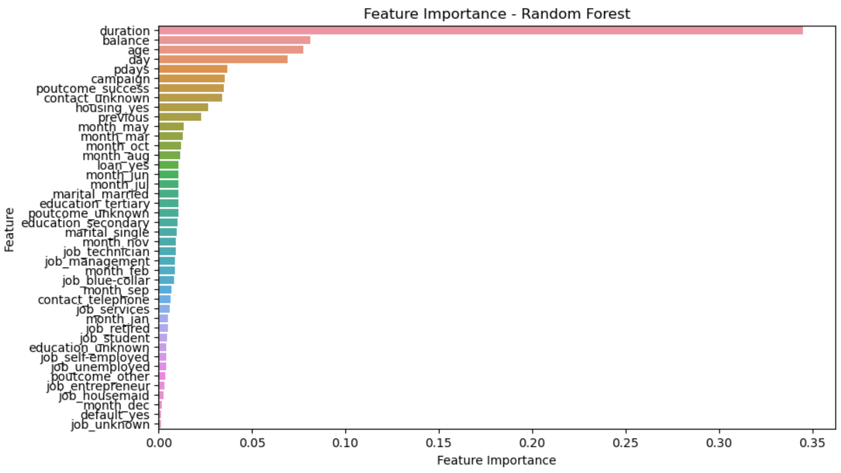




In [42]:

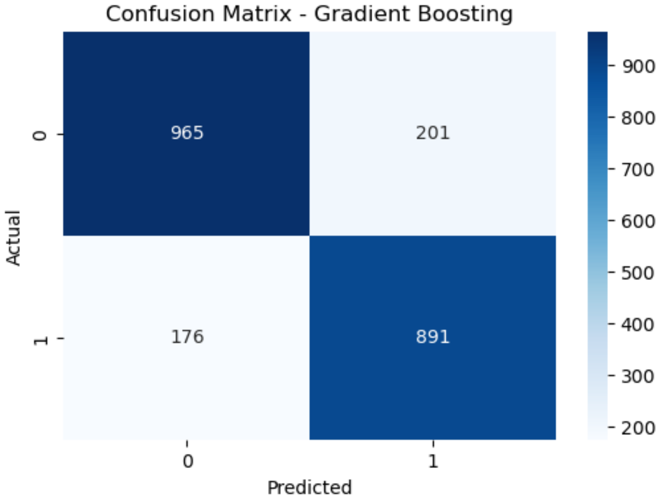
```python
from sklearn.metrics import confusion_matrix
import seaborn as sns

preds = random_forest.predict(X_test)
cm = confusion_matrix(y_test, preds)

plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, cmap='Blues', fmt='d')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - Random Forest')
plt.show()

feature_importances = random_forest.feature_importances_
feature_names = X.columns

feature_importance_df = pd.DataFrame({'Feature': feature_names, 'Importance': feature_importances
feature_importance_df = feature_importance_df.sort_values('Importance', ascending=False)

plt.figure(figsize=(10, 6))
sns.barplot(x='Importance', y='Feature', data=feature_importance_df)
plt.xlabel('Feature Importance')
plt.ylabel('Feature')
plt.title('Feature Importance - Random Forest')
plt.show()
```

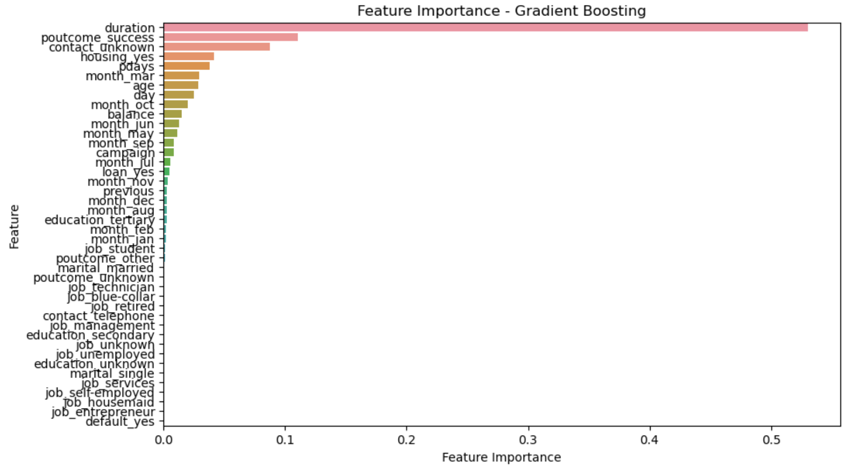




In [43]:

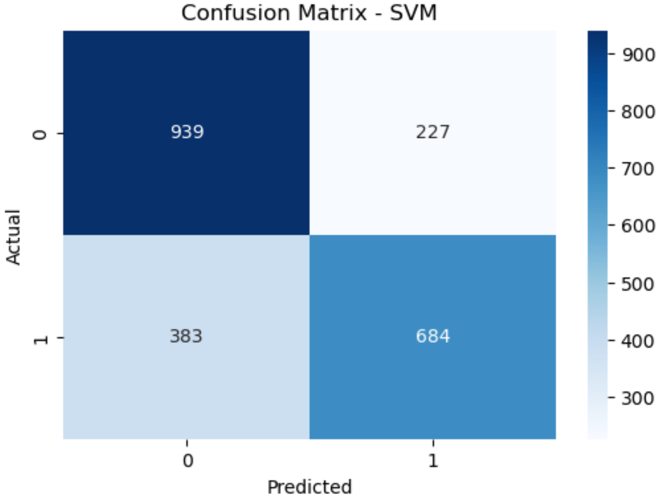
```
from sklearn.metrics import confusion_matrix
import seaborn as sns

preds = gradient_boosting.predict(X_test)
cm = confusion_matrix(y_test, preds)

plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, cmap='Blues', fmt='d')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - Gradient Boosting')
plt.show()

feature_importances = gradient_boosting.feature_importances_
feature_names = X.columns

feature_importance_df = pd.DataFrame({'Feature': feature_names, 'Importance': feature_importances
feature_importance_df = feature_importance_df.sort_values('Importance', ascending=False)

plt.figure(figsize=(10, 6))
sns.barplot(x='Importance', y='Feature', data=feature_importance_df)
plt.xlabel('Feature Importance')
plt.ylabel('Feature')
plt.title('Feature Importance - Gradient Boosting')
plt.show()
```

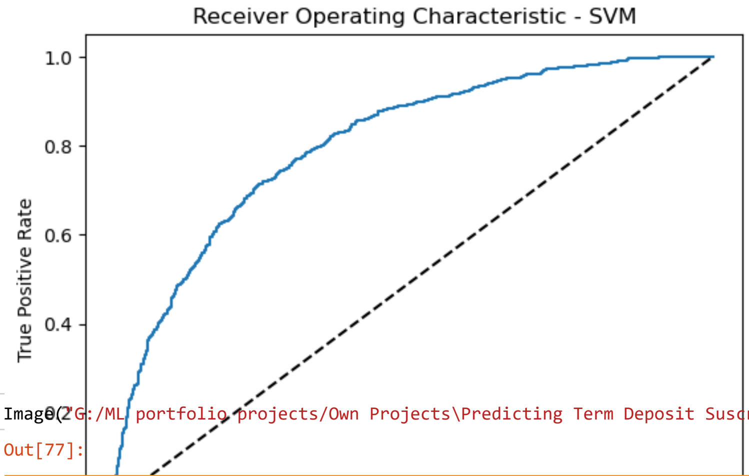




In [44]:

```
from sklearn.metrics import confusion_matrix
import seaborn as sns

preds = support_vector_machine.predict(X_test)
cm = confusion_matrix(y_test, preds)

plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, cmap='Blues', fmt='d')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - SVM')
plt.show()

from sklearn.metrics import roc_curve

preds = support_vector_machine.decision_function(X_test)  # Distance to the hyperplane
fpr, tpr, thresholds = roc_curve(y_test, preds)

plt.plot(fpr, tpr)
plt.plot([0, 1], [0, 1], 'k--')  # Diagonal line for random classifier
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic - SVM')
plt.show()
```






```
Image(r"G:/ML portfolio projects/Own Projects\Predicting Term Deposit Suscriptions/8.jpg")
```

Out[77]:

- Import the necessary libraries

In [45]:

```
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
```

- Define the parameter grid for hyperparameter tuning: For Random Forest

In [46]:

```
param_grid_rf = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 5, 10],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['sqrt', 'log2'],
}
```

- Perform hyperparameter tuning using grid search or random search: For Random Forest

In [47]:

```
rf_grid_search = GridSearchCV(estimator=random_forest, param_grid=param_grid_rf, scoring='accurac
rf_grid_search.fit(X_train, y_train)
best_rf_model = rf_grid_search.best_estimator_
```

- Get the best hyperparameters and retrain the models: For Random Forest

In [48]:

```
print("Best Random Forest Model:")
print(best_rf_model)
best_rf_model.fit(X_train, y_train)
```

```
Best Random Forest Model:
RandomForestClassifier(max_features='log2', min_samples_split=5)
```

Out[48]:

```
RandomForestClassifier
RandomForestClassifier(max_features='log2', min_samples_split=5)
```

- For Gradient Boosting

In [49]:

```
param_grid_gb = {
    'n_estimators': [100, 200, 300],
    'learning_rate': [0.01, 0.1, 1],
    'max_depth': [3, 5, 10],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
}
```

- For Gradient Boosting

In [50]:

```
gb_random_search = RandomizedSearchCV(estimator=gradient_boosting, param_distributions=param_grid_
gb_random_search.fit(X_train, y_train)
best_gb_model = gb_random_search.best_estimator_
```

- For Gradient Boosting

In [51]:

```
print("Best Gradient Boosting Model:")
print(best_gb_model)
best_gb_model.fit(X_train, y_train)
```

```
Best Gradient Boosting Model:
GradientBoostingClassifier(min_samples_leaf=2, n_estimators=300)
```

Out[51]:

```
GradientBoostingClassifier
GradientBoostingClassifier(min_samples_leaf=2, n_estimators=300)
```

- Make predictions on the test set using the retrained models

In [52]:

```
rf_preds = best_rf_model.predict(X_test)
gb_preds = best_gb_model.predict(X_test)
```

- Evaluate the performance of each model using evaluation metrics

In [53]:

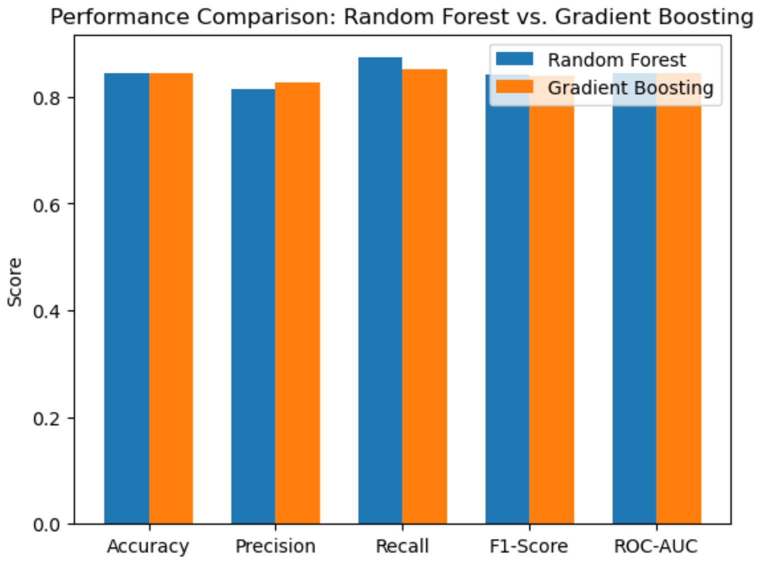
```
print("Random Forest Metrics:")
print("Accuracy:", accuracy_score(y_test, rf_preds))
print("Precision:", precision_score(y_test, rf_preds))
print("Recall:", recall_score(y_test, rf_preds))
print("F1-Score:", f1_score(y_test, rf_preds))
print("ROC-AUC:", roc_auc_score(y_test, rf_preds))
print("\n")

print("Gradient Boosting Metrics:")
print("Accuracy:", accuracy_score(y_test, gb_preds))
print("Precision:", precision_score(y_test, gb_preds))
print("Recall:", recall_score(y_test, gb_preds))
print("F1-Score:", f1_score(y_test, gb_preds))
print("ROC-AUC:", roc_auc_score(y_test, gb_preds))
```

```
Random Forest Metrics:
Accuracy: 0.8441558441558441
Precision: 0.8139737991266376
Recall: 0.8734770384254921
F1-Score: 0.8426763110307415
ROC-AUC: 0.8454006118371027


Gradient Boosting Metrics:
Accuracy: 0.8441558441558441
Precision: 0.8265213442325159
Recall: 0.8528584817244611
F1-Score: 0.8394833948339483
ROC-AUC: 0.8445252957507383
```

- Compare the models using a bar plot

In [54]:

```
import matplotlib.pyplot as plt

metrics = ['Accuracy', 'Precision', 'Recall', 'F1-Score', 'ROC-AUC']
rf_scores = [accuracy_score(y_test, rf_preds), precision_score(y_test, rf_preds),
             recall_score(y_test, rf_preds), f1_score(y_test, rf_preds), roc_auc_score(y_test, rf
gb_scores = [accuracy_score(y_test, gb_preds), precision_score(y_test, gb_preds),
             recall_score(y_test, gb_preds), f1_score(y_test, gb_preds), roc_auc_score(y_test, gb

x = range(len(metrics))
width = 0.35

fig, ax = plt.subplots()
ax.bar(x, rf_scores, width, label='Random Forest')
ax.bar([i + width for i in x], gb_scores, width, label='Gradient Boosting')

ax.set_ylabel('Score')
ax.set_title('Performance Comparison: Random Forest vs. Gradient Boosting')
ax.set_xticks([i + width/2 for i in x])
ax.set_xticklabels(metrics)
ax.legend()

plt.show()
```



In [78]:

```
Image("G:/ML portfolio projects/Own Projects\Predicting Term Deposit Suscriptions/2.png")
```

Out[78]:

In [ ]: