term-deposit-prediction-project

June 3, 2024

1 Term Deposit Prediction Project to aid Marketing Activities

2 By Peter Tinashe Mundowa

Abstract: Marketing campaigns are characterized by focusing on the customer needs and their overall satisfaction. Nevertheless, there are different variables that determine whether a marketing campaign will be successful or not. There are certain variables that we need to take into consideration when making a marketing campaign. A Term deposit is a deposit that a bank or a financial institution offers with a fixed rate (often better than just opening a deposit account) in which your money will be returned back at a specific maturity time.

Problem Statement: Predict if a customer subscribes to a term deposits or not, when contacted by a marketing agent, by understanding the different features and performing predictive analytics

3 Importing Libraries

[24]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, roc_curve,␣
↪auc
from sklearn.model_selection import GridSearchCV
from IPython.display import display
```

4 Loading the dataset

[3]:
```
bank=pd.read_excel("C:\\Users\\user\\OneDrive\\Documents\\Projects for␣
↪Portfolio\\bank_data.xlsx")
bank.head()
```

[3]:
```
    age        job  marital    education  default housing loan    contact  \
0  56.0  housemaid  married     basic.4y       no      no   no  telephone
```

```
1  57.0   services  married  high.school  unknown      no   no  telephone
2  37.0   services  married  high.school       no     yes   no  telephone
3  40.0     admin.  married     basic.6y       no      no   no  telephone
4  56.0   services  married  high.school       no      no  yes  telephone

  month day_of_week  …  campaign  pdays  previous     poutcome emp.var.rate  \
0   may         mon  …       1.0  999.0       0.0  nonexistent          1.1
1   may         mon  …       1.0  999.0       0.0  nonexistent          1.1
2   may         mon  …       1.0  999.0       0.0  nonexistent          1.1
3   may         mon  …       1.0  999.0       0.0  nonexistent          1.1
4   may         mon  …       1.0  999.0       0.0  nonexistent          1.1

   cons.price.idx  cons.conf.idx  euribor3m  nr.employed   y
0          93.994          -36.4      4.857       5191.0  no
1          93.994          -36.4      4.857       5191.0  no
2          93.994          -36.4      4.857       5191.0  no
3          93.994          -36.4      4.857       5191.0  no
4          93.994          -36.4      4.857       5191.0  no

[5 rows x 21 columns]
```

5 Inspecting the data

[4]:
```
bank.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41188 entries, 0 to 41187
Data columns (total 21 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   age             41188 non-null  float64
 1   job             41188 non-null  object
 2   marital         41188 non-null  object
 3   education       41188 non-null  object
 4   default         41188 non-null  object
 5   housing         41188 non-null  object
 6   loan            41188 non-null  object
 7   contact         41188 non-null  object
 8   month           41188 non-null  object
 9   day_of_week     41188 non-null  object
 10  duration        41188 non-null  float64
 11  campaign        41188 non-null  float64
 12  pdays           41188 non-null  float64
 13  previous        41188 non-null  float64
 14  poutcome        41188 non-null  object
 15  emp.var.rate    41188 non-null  float64
```

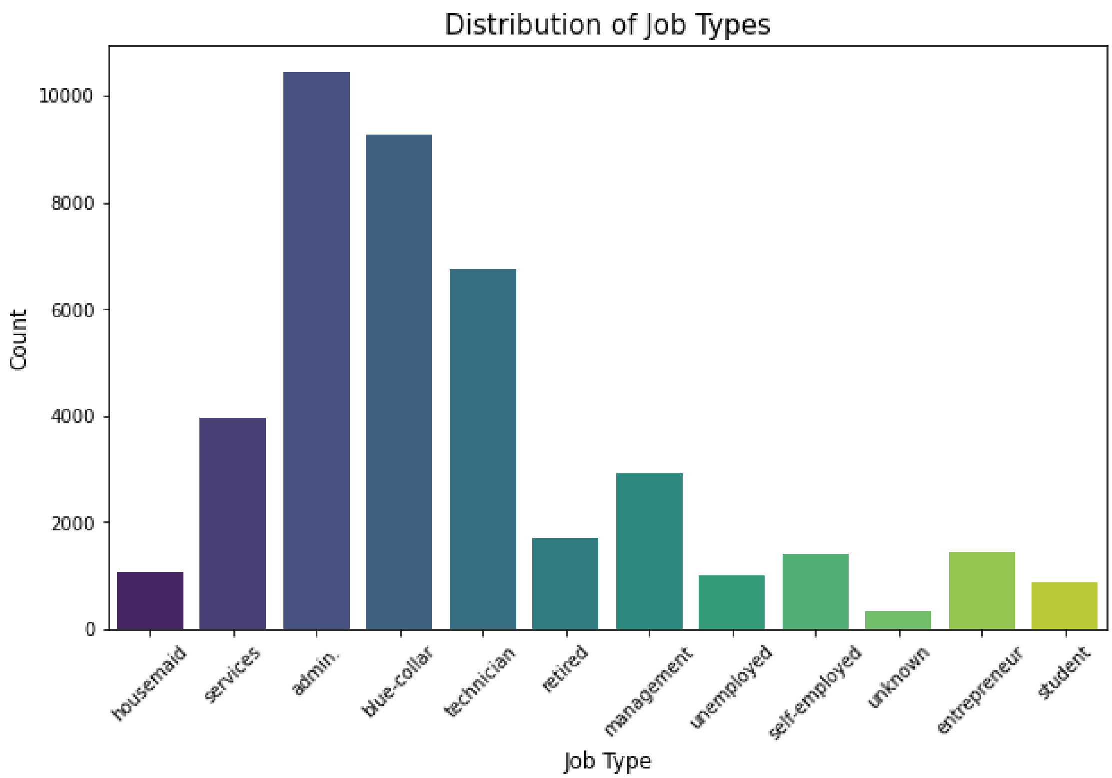
```
 16   cons.price.idx  41188 non-null  float64
 17   cons.conf.idx   41188 non-null  float64
 18   euribor3m       41188 non-null  float64
 19   nr.employed     41188 non-null  float64
 20   y               41188 non-null  object
dtypes: float64(10), object(11)
memory usage: 6.6+ MB
```

[5]:
```
bank.describe()
```

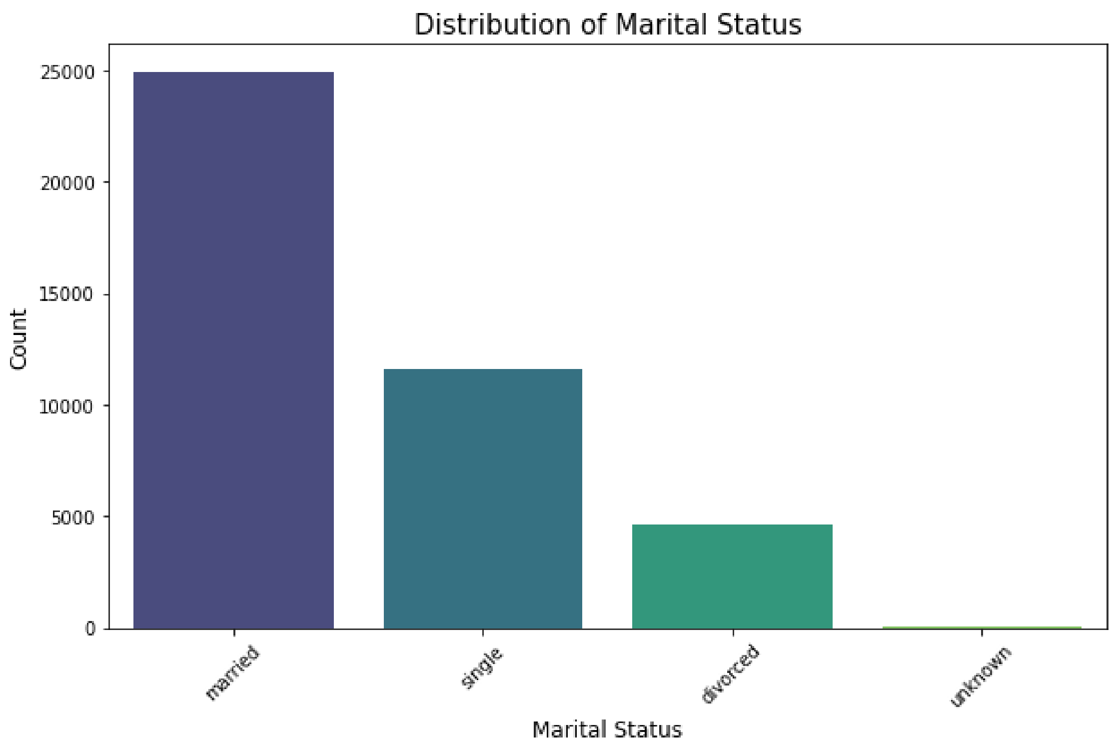
[5]:
```
               age      duration      campaign         pdays      previous  \
count  41188.00000  41188.000000  41188.000000  41188.000000  41188.000000
mean      40.02406    258.285010      2.567593    962.475454      0.172963
std       10.42125    259.279249      2.770014    186.910907      0.494901
min       17.00000      0.000000      1.000000      0.000000      0.000000
25%       32.00000    102.000000      1.000000    999.000000      0.000000
50%       38.00000    180.000000      2.000000    999.000000      0.000000
75%       47.00000    319.000000      3.000000    999.000000      0.000000
max       98.00000   4918.000000     56.000000    999.000000      7.000000

       emp.var.rate  cons.price.idx  cons.conf.idx     euribor3m   nr.employed
count  41188.000000    41188.000000   41188.000000  41188.000000  41188.000000
mean       0.081886       93.575664     -40.502600      3.621291   5167.035911
std        1.570960        0.578840       4.628198      1.734447     72.251528
min       -3.400000       92.201000     -50.800000      0.634000   4963.600000
25%       -1.800000       93.075000     -42.700000      1.344000   5099.100000
50%        1.100000       93.749000     -41.800000      4.857000   5191.000000
75%        1.400000       93.994000     -36.400000      4.961000   5228.100000
max        1.400000       94.767000     -26.900000      5.045000   5228.100000
```

6 Data Exploration

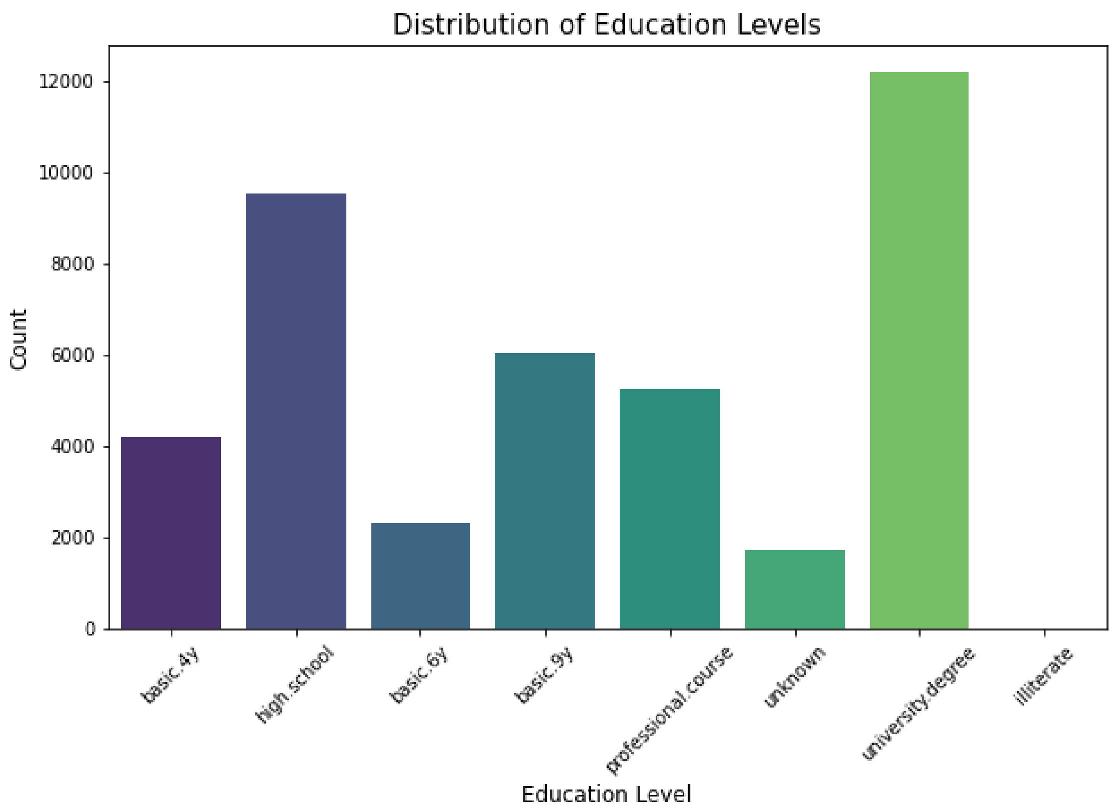
[10]:
```
#Exploring categorical variables
plt.figure(figsize=(10, 6))
sns.countplot(data=bank, x='job', palette='viridis')
plt.xlabel('Job Type', fontsize=12)
plt.ylabel('Count', fontsize=12)
plt.title('Distribution of Job Types', fontsize=15)
plt.xticks(rotation=45)
plt.show()
```



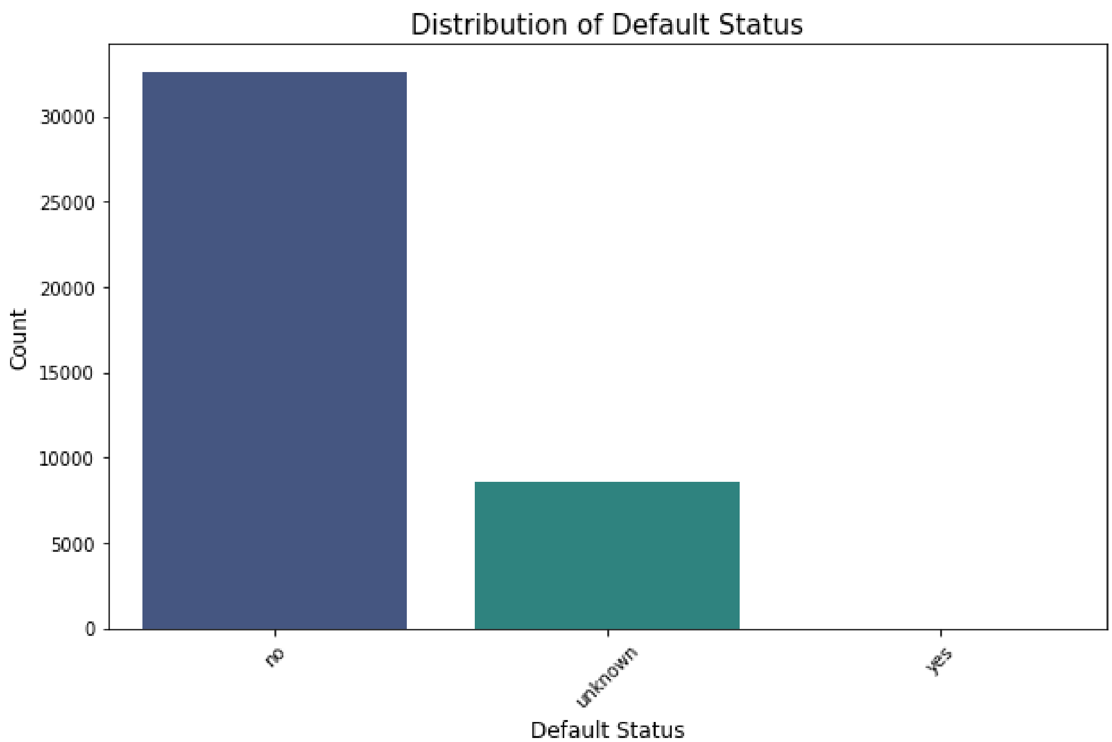
[11]:
```
#Marital status
plt.figure(figsize=(10, 6))
sns.countplot(data=bank, x='marital', palette='viridis')
plt.xlabel('Marital Status', fontsize=12)
plt.ylabel('Count', fontsize=12)
plt.title('Distribution of Marital Status', fontsize=15)
plt.xticks(rotation=45)
plt.show()
```



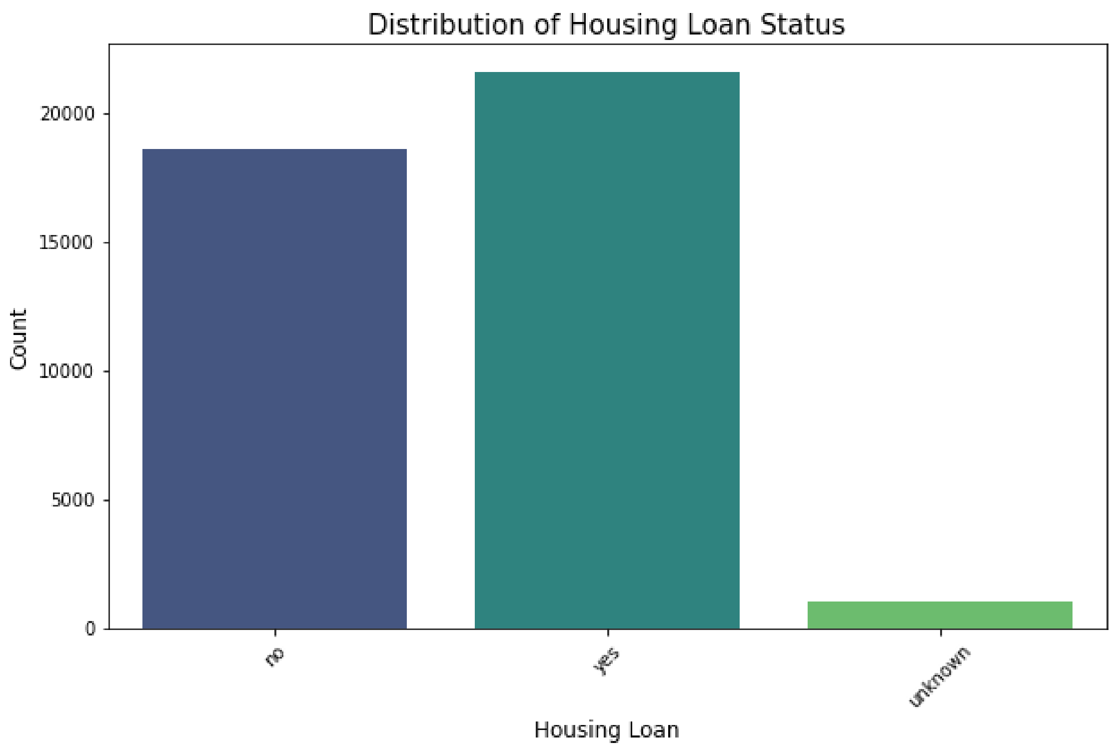
[12]:
```
#Education
plt.figure(figsize=(10, 6))
sns.countplot(data=bank, x='education', palette='viridis')
plt.xlabel('Education Level', fontsize=12)
plt.ylabel('Count', fontsize=12)
plt.title('Distribution of Education Levels', fontsize=15)
plt.xticks(rotation=45)
plt.show()
```



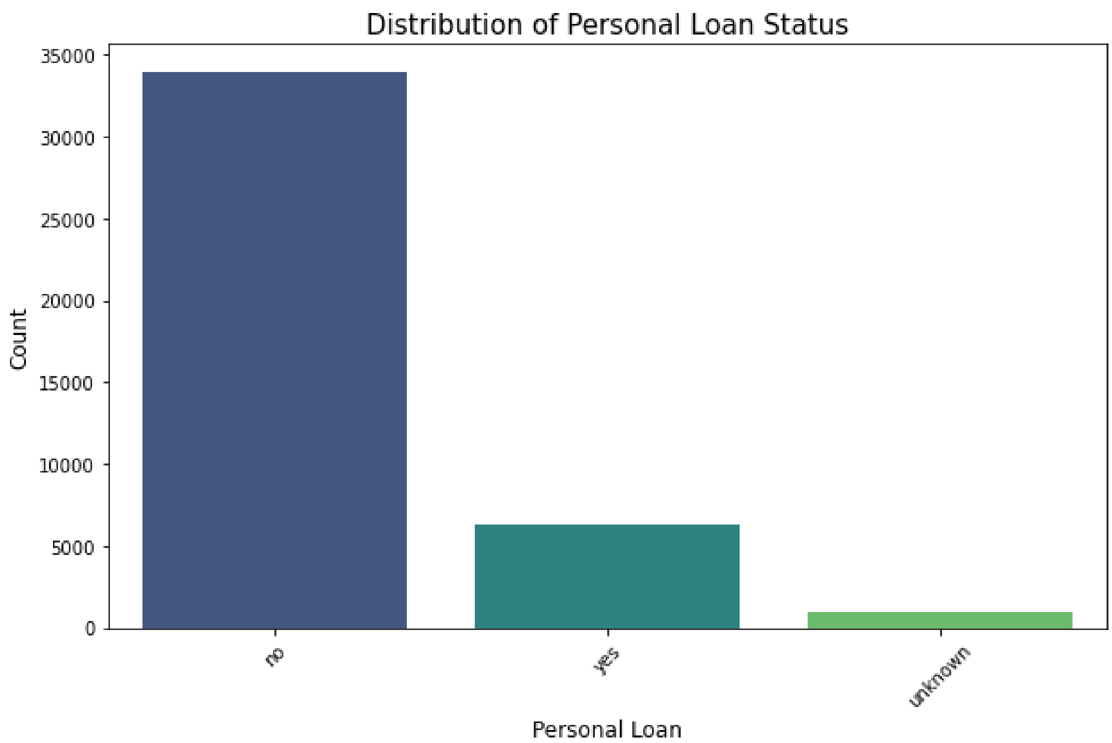
[13]:
```
#Default distribution
plt.figure(figsize=(10, 6))
sns.countplot(data=bank, x='default', palette='viridis')
plt.xlabel('Default Status', fontsize=12)
plt.ylabel('Count', fontsize=12)
plt.title('Distribution of Default Status', fontsize=15)
plt.xticks(rotation=45)
plt.show()
```



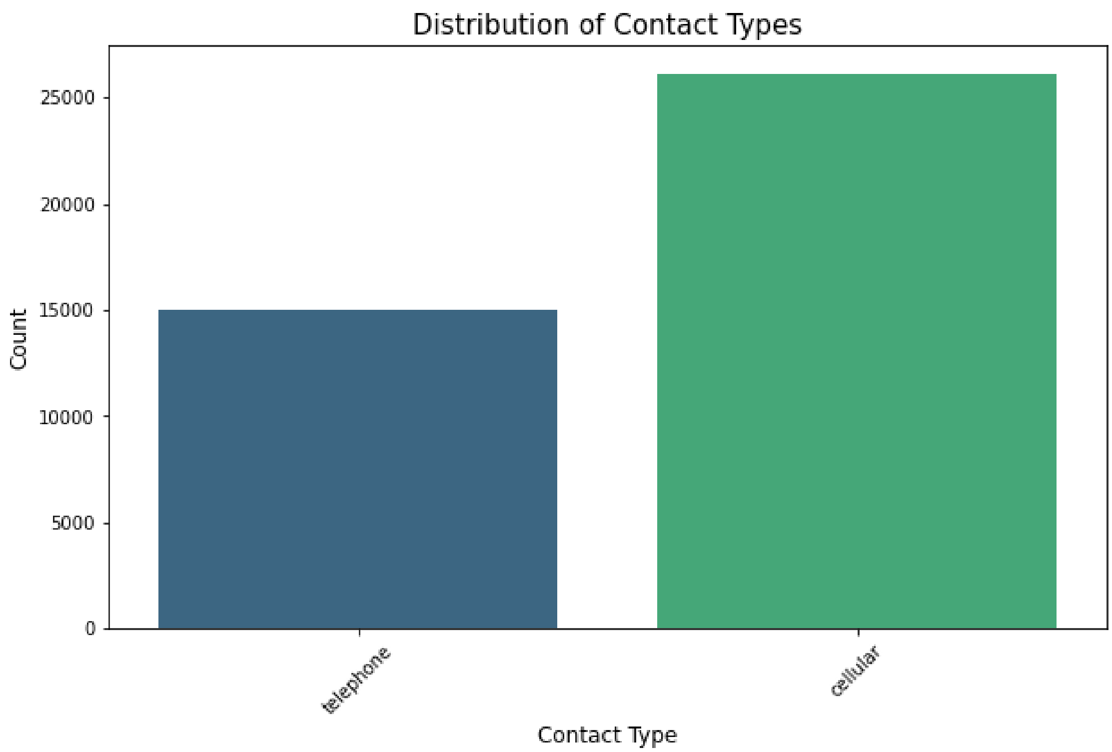
[14]:
```
#Housing loan status distribution
plt.figure(figsize=(10, 6))
sns.countplot(data=bank, x='housing', palette='viridis')
plt.xlabel('Housing Loan', fontsize=12)
plt.ylabel('Count', fontsize=12)
plt.title('Distribution of Housing Loan Status', fontsize=15)
plt.xticks(rotation=45)
plt.show()
```



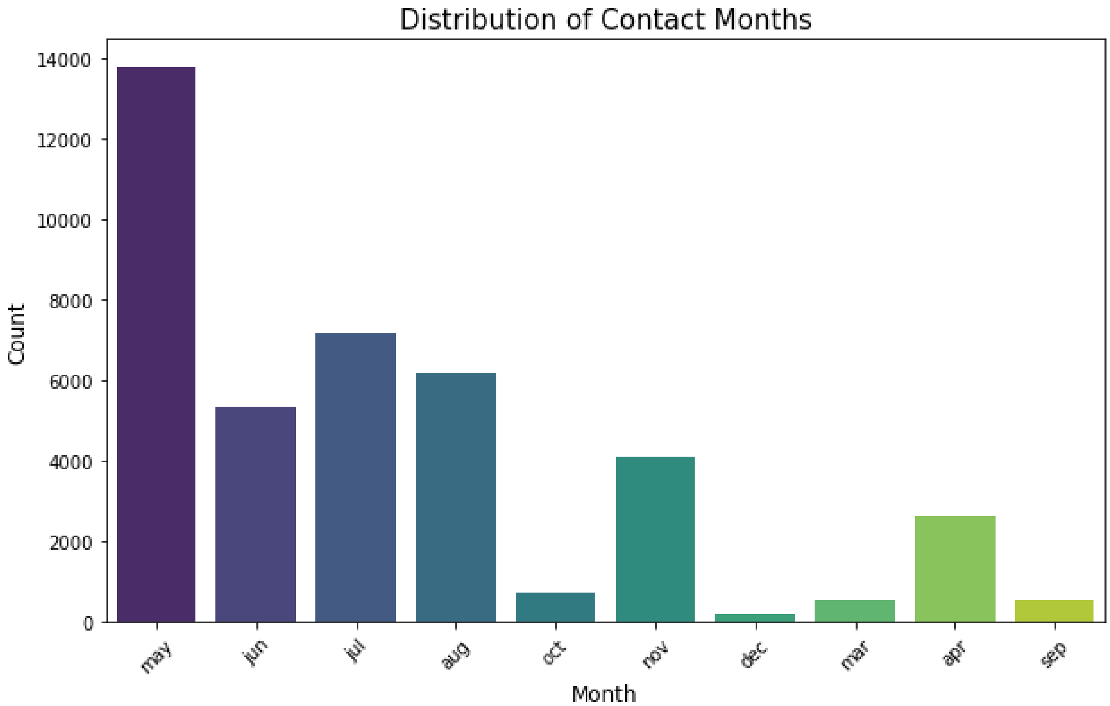
[15]:
```
#Distribution of loan status
plt.figure(figsize=(10, 6))
sns.countplot(data=bank, x='loan', palette='viridis')
plt.xlabel('Personal Loan', fontsize=12)
plt.ylabel('Count', fontsize=12)
plt.title('Distribution of Personal Loan Status', fontsize=15)
plt.xticks(rotation=45)
plt.show()
```



[16]:
```
#Distribution of contact types
plt.figure(figsize=(10, 6))
sns.countplot(data=bank, x='contact', palette='viridis')
plt.xlabel('Contact Type', fontsize=12)
plt.ylabel('Count', fontsize=12)
plt.title('Distribution of Contact Types', fontsize=15)
plt.xticks(rotation=45)
plt.show()
```

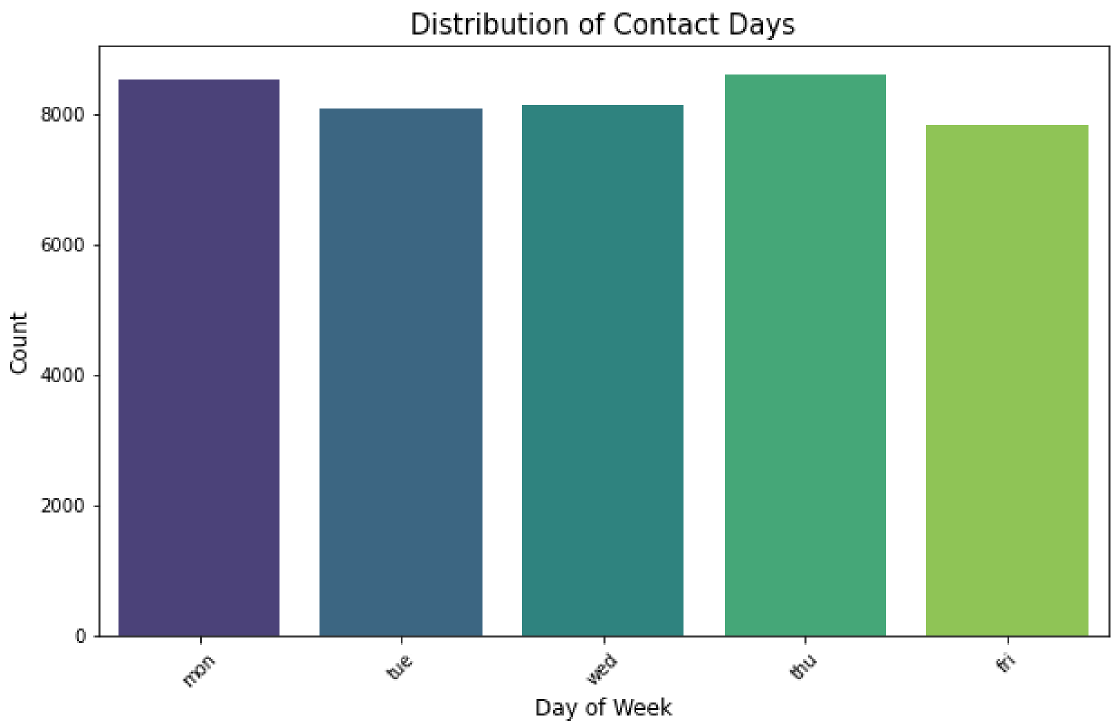


[17]:
```
#Distribution of month contact
plt.figure(figsize=(10, 6))
sns.countplot(data=bank, x='month', palette='viridis')
plt.xlabel('Month', fontsize=12)
plt.ylabel('Count', fontsize=12)
plt.title('Distribution of Contact Months', fontsize=15)
plt.xticks(rotation=45)
plt.show()
```

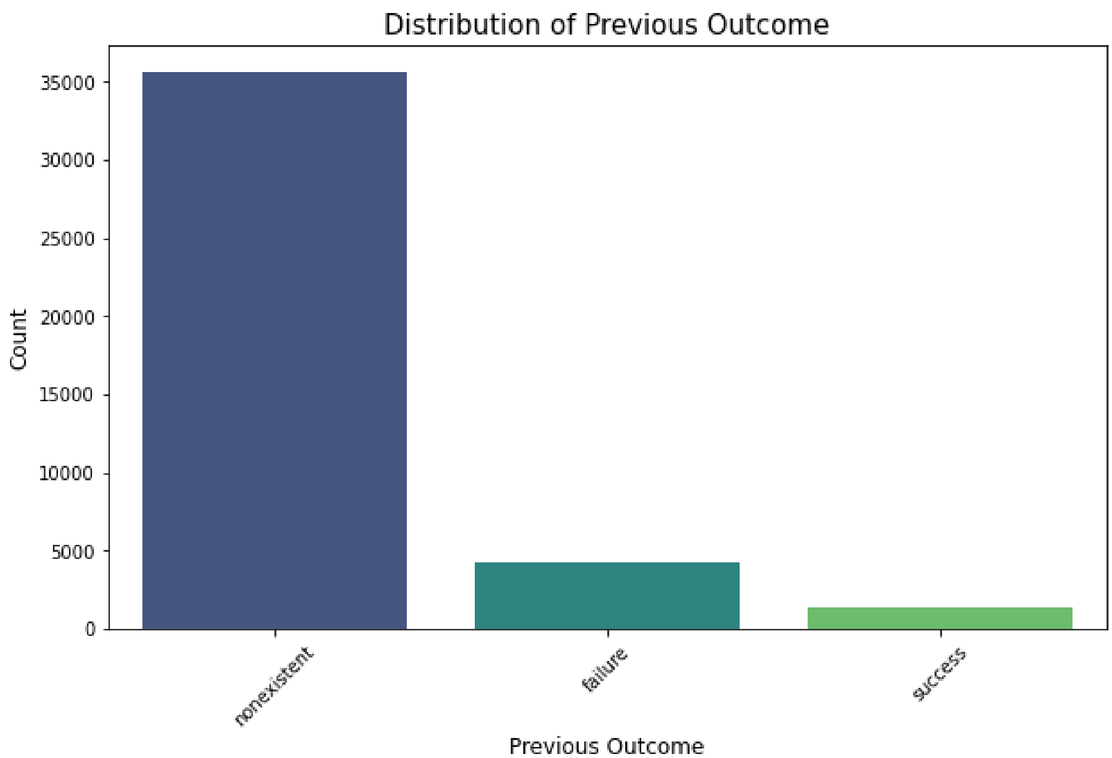


[18]:

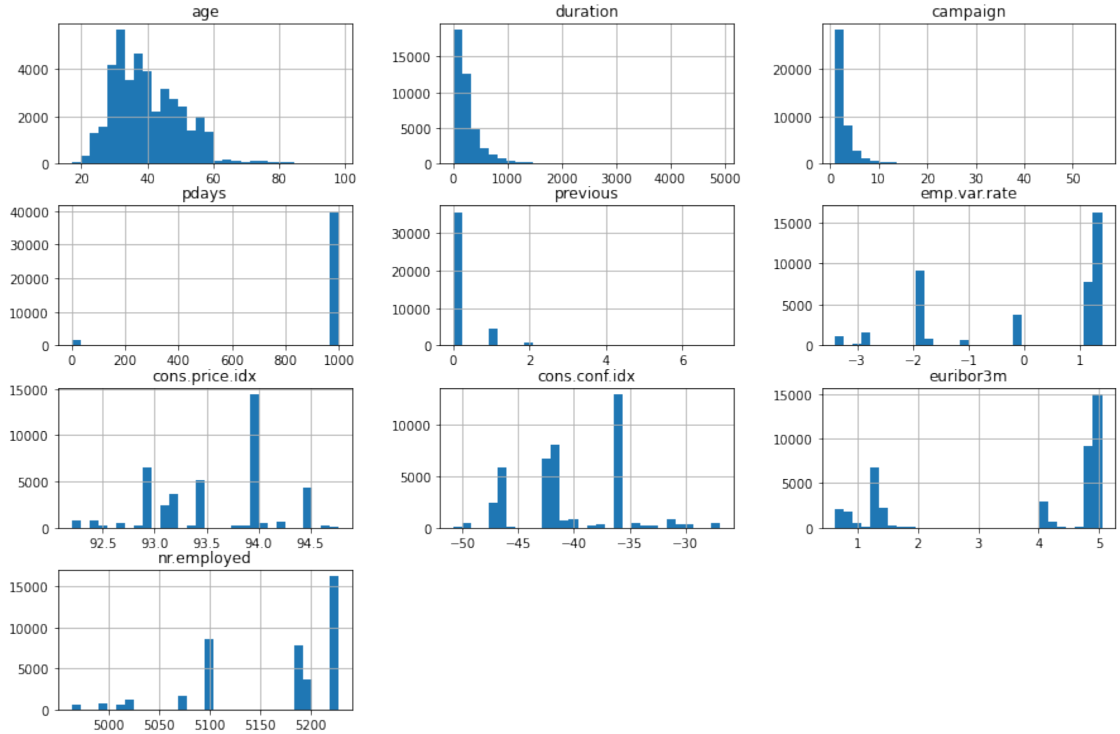
```
#Distribution of contact days
plt.figure(figsize=(10, 6))
sns.countplot(data=bank, x='day_of_week', palette='viridis')
plt.xlabel('Day of Week', fontsize=12)
plt.ylabel('Count', fontsize=12)
plt.title('Distribution of Contact Days', fontsize=15)
plt.xticks(rotation=45)
plt.show()
```



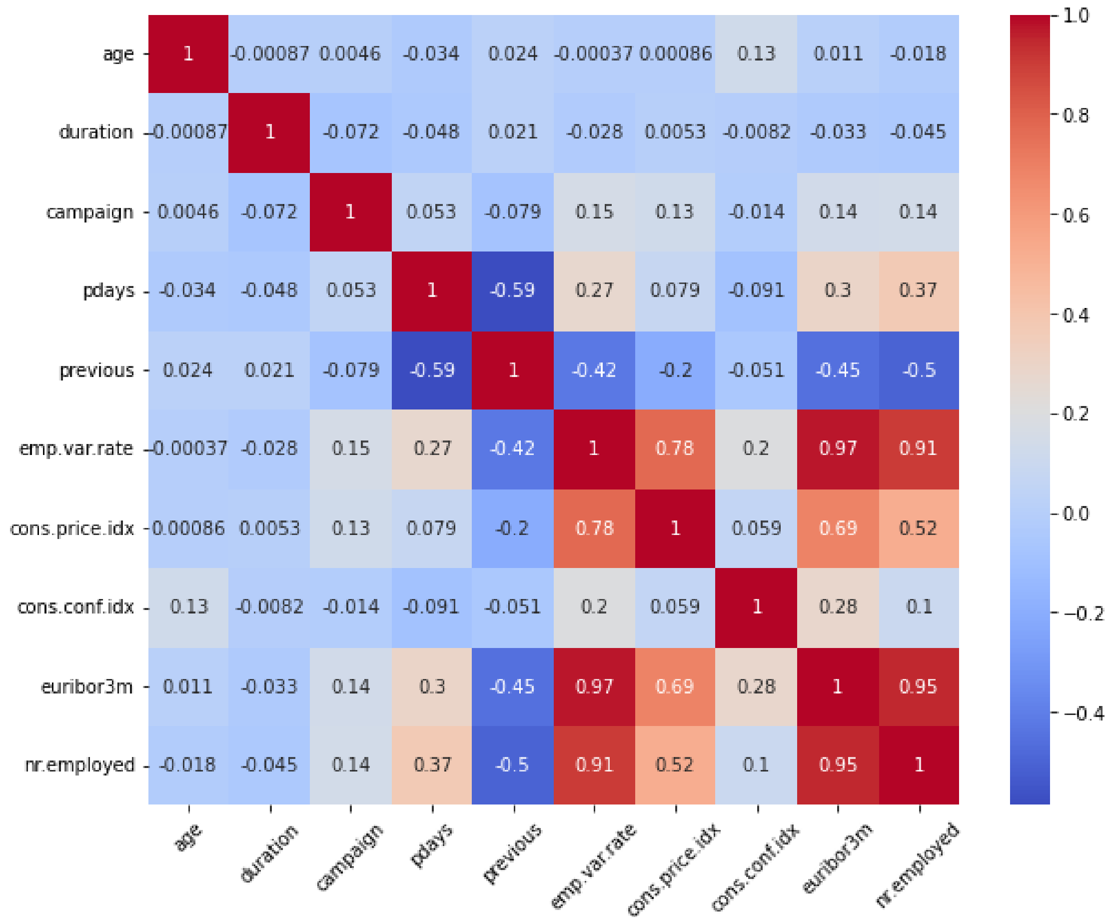
[19]:
```
#Distribution of previous outcome
plt.figure(figsize=(10, 6))
sns.countplot(data=bank, x='poutcome', palette='viridis')
plt.xlabel('Previous Outcome', fontsize=12)
plt.ylabel('Count', fontsize=12)
plt.title('Distribution of Previous Outcome', fontsize=15)
plt.xticks(rotation=45)
plt.show()
```



[20]:
```
#Exploring numerical variables
numerical_features = ['age', 'duration', 'campaign', 'pdays', 'previous', 'emp.
↪var.rate', 'cons.price.idx', 'cons.conf.idx', 'euribor3m', 'nr.employed']
bank[numerical_features].hist(bins=30, figsize=(15, 10))
plt.show()
```



[22]:
```
#Correlation matrix
corr_matrix = bank.corr()

plt.figure(figsize=(10, 8))  #Adjusting the size of the figure as desired

# Creating the heatmap with annotated values
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')

# Rotating x-labels by 45 degrees for better visibility
plt.xticks(rotation=45)

plt.show()
```



7 Data Preprocessing

[26]:
```
#Checking for missing values
if bank.isnull().sum().sum() == 0:
    # Display happy emoji
    display('  Dataset has no missing values')
else:
    # Display sad emoji and message
    display(' There are missing values in the dataset. Action is needed.')
```

```
'  Dataset has no missing values'
```

[27]:
```
#Encoding categorical variables
bank = pd.get_dummies(bank, columns=categorical_features, drop_first=True)
```

[28]:
```
#Label encoding the target variable
le = LabelEncoder()
```

```
bank['y'] = le.fit_transform(bank['y'])
```

[29]:
```
#Feature Scaling
scaler = StandardScaler()
bank[numerical_features] = scaler.fit_transform(bank[numerical_features])
```

8 Model Building

[30]:
```
#train-test split
X = bank.drop('y', axis=1)
y = bank['y']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

[44]:
```
#Fitting Random forest model
rf_model = RandomForestClassifier(random_state=42)
rf_model.fit(X_train, y_train)
```

[44]:
```
RandomForestClassifier(random_state=42)
```

9 Hyperparameter tuning

[45]:
```
#Grid search for best parameters
param_grid = {
    'n_estimators': [100, 200],
    'max_features': ['auto', 'sqrt'],
    'max_depth': [6, 8],
    'criterion': ['gini']
}

grid_search = GridSearchCV(estimator=rf_model, param_grid=param_grid, cv=3,␣
↪n_jobs=-1, verbose=2)
grid_search.fit(X_train, y_train)
best_rf_model = grid_search.best_estimator_

y_pred_best = best_rf_model.predict(X_test)
print(confusion_matrix(y_test, y_pred_best))
print(classification_report(y_test, y_pred_best))
```

```
Fitting 3 folds for each of 8 candidates, totalling 24 fits
[[7214   89]
 [ 687  248]]
              precision    recall  f1-score   support

           0       0.91      0.99      0.95      7303
```

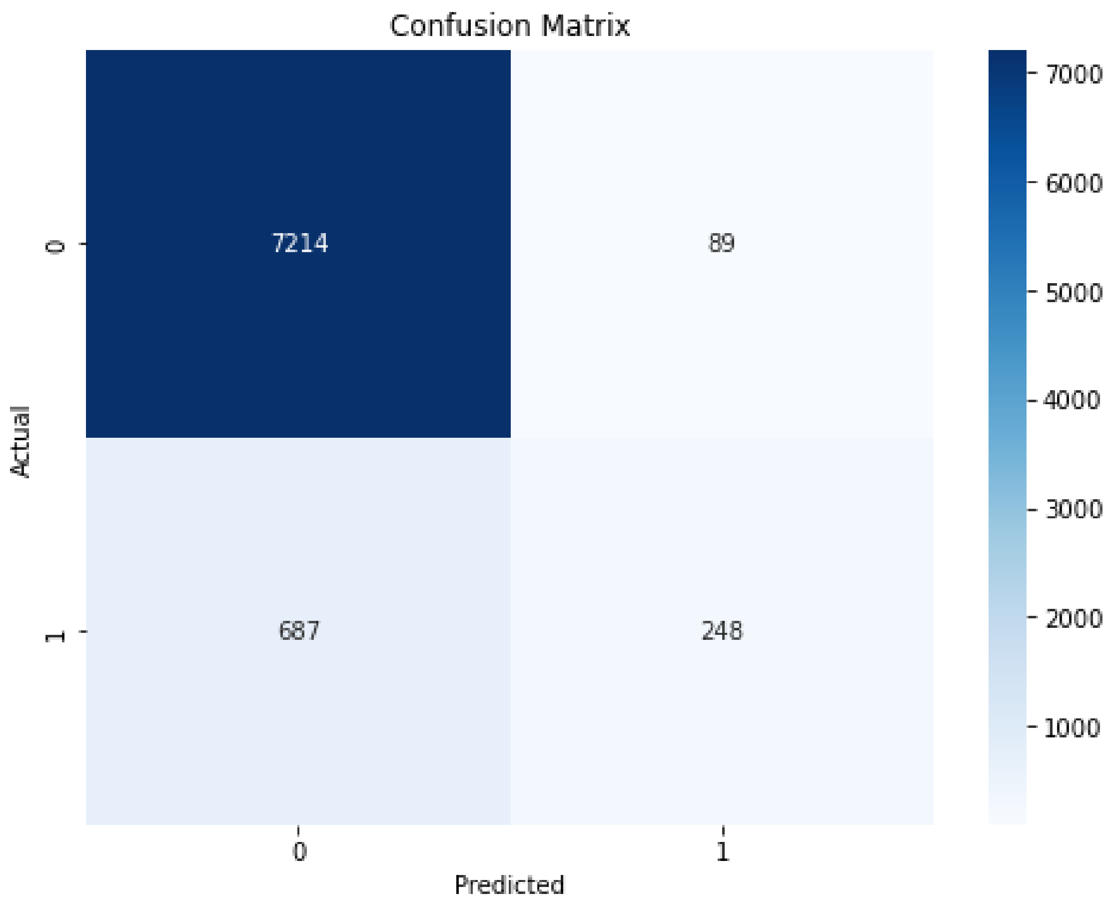
```
           1       0.74      0.27      0.39       935

    accuracy                           0.91      8238
   macro avg       0.82      0.63      0.67      8238
weighted avg       0.89      0.91      0.89      8238
```

10 Evaluating the model

[46]:
```
#Evaluating the model on test dataset
y_pred = best_rf_model.predict(X_test)
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[7214   89]
 [ 687  248]]
              precision    recall  f1-score   support

           0       0.91      0.99      0.95      7303
           1       0.74      0.27      0.39       935

    accuracy                           0.91      8238
   macro avg       0.82      0.63      0.67      8238
weighted avg       0.89      0.91      0.89      8238
```

[47]:
```
#Confusion matrix visualization

cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

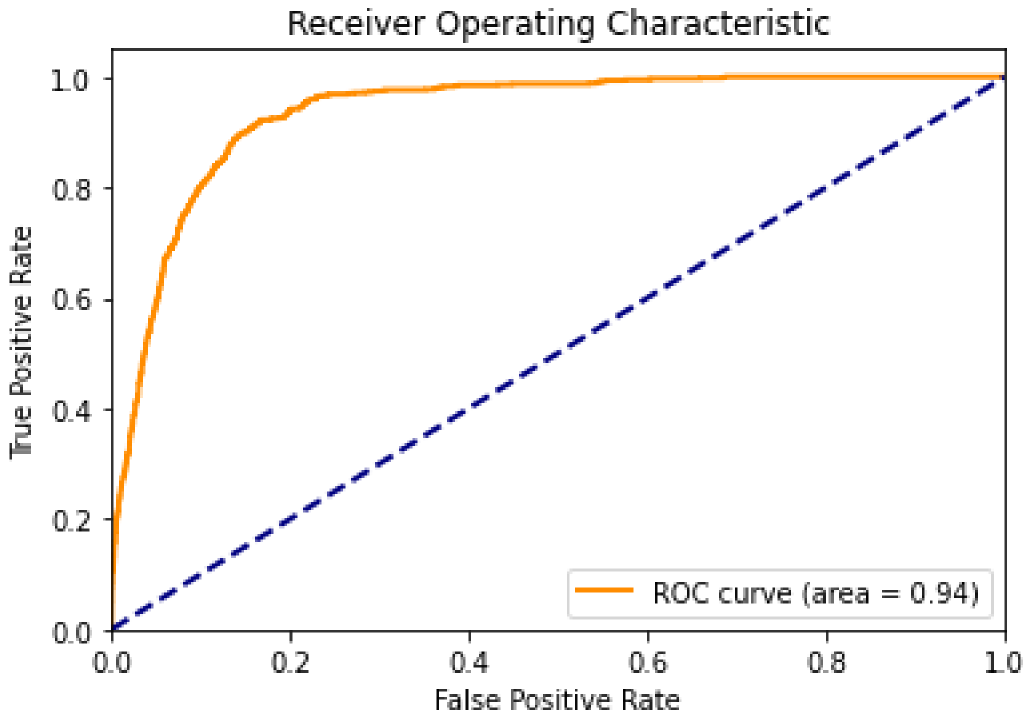


Explanation:

True Positives (TP): Correctly predicted positive cases. True Negatives (TN): Correctly predicted negative cases. False Positives (FP): Incorrectly predicted positive cases. False Negatives (FN): Incorrectly predicted negative cases.

[48]:
```
#ROC Curve
y_prob = best_rf_model.predict_proba(X_test)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, y_prob)
roc_auc = auc(fpr, tpr)

plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' %␣
 ↪roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
```

```
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()
```

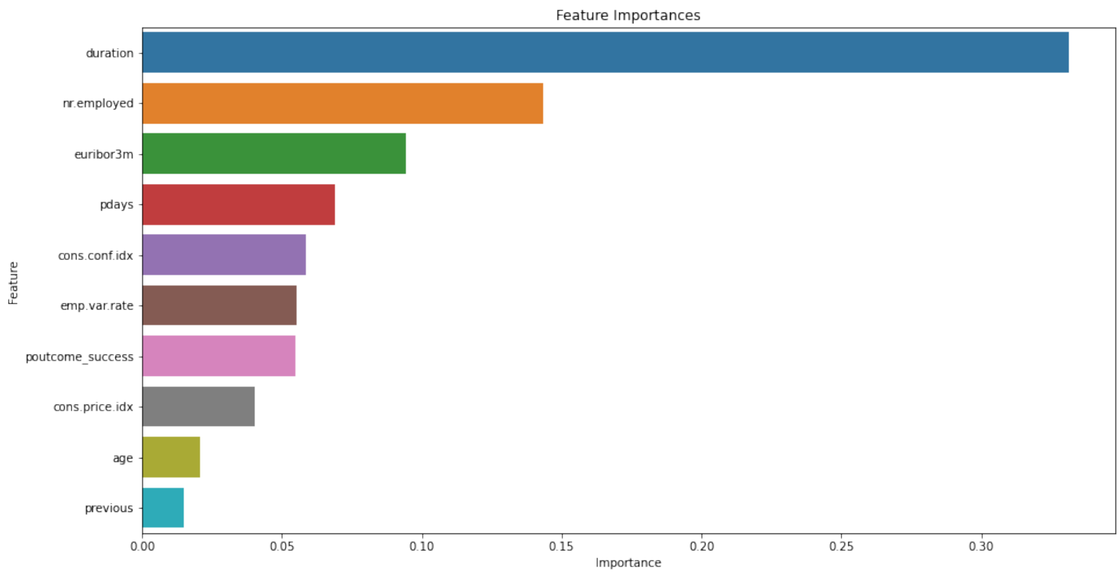


Explanation:

The ROC Curve illustrates the model's ability to distinguish between positive and negative classes. The area under the curve (AUC) quantifies the overall performance; the closer to 1, the better.

11 Interpretation and Reporting

[52]:
```
#Visualizing top 10 feature importance
feature_importances = best_rf_model.feature_importances_
features = X.columns
importances_df = pd.DataFrame({'Feature': features, 'Importance':␣
 ↪feature_importances})
importances_df = importances_df.sort_values(by='Importance', ascending=False).
 ↪head(10)

plt.figure(figsize=(15, 8))
sns.barplot(x='Importance', y='Feature', data=importances_df)
plt.title('Feature Importances')
plt.show()
```



Explanation:

The bar plot shows the importance of each feature in making predictions. Higher importance indicates a greater influence on the model's predictions.

12 Business impact

My predictive model helps identify customers who are more likely to subscribe to term deposits. By focusing on these customers, the marketing team can optimize their efforts and resources, leading to higher conversion rates and increased revenue.

Feature importance analysis reveals that certain variables play a more significant role in predicting term deposit subscriptions. Notably, "duration" of the call, "nr.employed" (number of employees), and "euribor3m" (Euro Interbank Offered Rate) are among the top features contributing to the model's predictive power.

By leveraging these insights, the business can tailor its marketing strategies to focus on factors that drive term deposit subscriptions. For instance, prioritizing longer call durations and targeting customers during periods of lower unemployment rates (reflected by "nr.employed" and "euribor3m") can potentially increase subscription rates.

Additionally, features such as "poutcome_success" (outcome of the previous marketing campaign) and "month_oct" (month of contact) also hold significance, suggesting the importance of past campaign success and timing in influencing subscription decisions.

Overall, this project equips the business with actionable insights to optimize its marketing campaigns, enhance customer targeting strategies, and ultimately increase term deposit subscriptions. By leveraging feature importance analysis, the business can make data-driven decisions to drive better outcomes and achieve its marketing objectives.

```
[ ]:
```