Bank Deposit Subscription

with Data Science

Denilson Panzo

Introduction

The dataset we're using is made up of information about a bank's customers with details of the campaigns, such as type of contact, duration of the call and the customer's financial history. The target variable is binary **(yes/no)** to indicate whether the customer has subscribed to the deposit.

The bank wants to predict whether customers will subscribe to a term deposit, using data from telephone marketing campaigns. Implementing a predictive solution will allow the bank to target its campaigns more efficiently, saving time and resources by focusing on the customers most likely to convert.

Analytical Objective:
To predict the probability of customers subscribing to term deposits.

Why? The majority of clients do not subscribe to deposits (88 per cent ‘no’ against 12 per cent ‘yes’),

Como? Can the bank improve its time deposit underwriting strategy?

Outline

Exploratory Data Analysis

EDA visualizations

Predictive Models

Conclusion

Next Steps

EDA

Exploratory Data Analysis

The dataset contains **45211 bank customer data**, each with details such as **age**, **job** and 15 more details which are column names. In addition, we can see there are some missing values in **employment, education, contact, poutcome** (they have the value '**unknown**') as we can see these missing values in the 10 rows shown below.

```python
# For data visualization
import matplotlib.pyplot as plt  # Biblioteca para visualização de gráficos
import seaborn as sns  # Biblioteca para visualização de dados com gráficos estatísticos

# For displaying all of the columns in dataframes
pd.set_option('display.max_columns', None)  # Para exibir todas as colunas dos dataframes

# Iniciar a medição do tempo
import time

from IPython import get_ipython

# Ativar comandos mágicos (se necessário)
get_ipython().run_line_magic('time', '')

# For data modeling
from xgboost import XGBClassifier  # Classificador XGBoost
from xgboost import XGBRegressor  # Regressor XGBoost
from xgboost import plot_importance  # Função para plotar a importância das variáveis no XGBoost

from sklearn.linear_model import LogisticRegression  # Regressão logística para modelagem preditiva
from sklearn.tree import DecisionTreeClassifier  # Modelo de árvore de decisão
from sklearn.ensemble import RandomForestClassifier  # Classificador de Random Forest

# For metrics and helpful functions
from sklearn.model_selection import GridSearchCV, train_test_split  # Para pesquisa de parâmetros e divisão de dados
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score  # Métricas de avaliação de mode
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay  # Matriz de confusão e visualização
from sklearn.metrics import classification_report  # Relatório de classificação com várias métricas
from sklearn.metrics import roc_auc_score, roc_curve  # AUC-ROC para avaliação de desempenho de classificadores
from sklearn.tree import plot_tree  # Para visualização de árvores de decisão

# For saving models
import pickle  # Biblioteca para salvar e carregar modelos treinados

# Adding KNeighborsClassifier and SVC
from sklearn.neighbors import KNeighborsClassifier  # Classificador KNN (K-Nearest Neighbors) para classificação bas
from sklearn.svm import SVC  # Suport Vector Classifier (SVC) para classificação com vetores de suporte
```

```python
train_df = pd.read_csv('train.csv', sep=";")
test_df = pd.read_csv('test.csv', sep=";")

train_df.head()
```

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	y
0	58	management	married	tertiary	no	2143	yes	no	unknown	5	may	261	1	-1	0	unknown	no
1	44	technician	single	secondary	no	29	yes	no	unknown	5	may	151	1	-1	0	unknown	no
2	33	entrepreneur	married	secondary	no	2	yes	yes	unknown	5	may	76	1	-1	0	unknown	no
3	47	blue-collar	married	unknown	no	1506	yes	no	unknown	5	may	92	1	-1	0	unknown	no
4	33	unknown	single	unknown	no	1	no	no	unknown	5	may	198	1	-1	0	unknown	no

```python
train_df.tail()
```

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	y
45206	51	technician	married	tertiary	no	825	no	no	cellular	17	nov	977	3	-1	0	unknown	yes
45207	71	retired	divorced	primary	no	1729	no	no	cellular	17	nov	456	2	-1	0	unknown	yes
45208	72	retired	married	secondary	no	5715	no	no	cellular	17	nov	1127	5	184	3	success	yes
45209	57	blue-collar	married	secondary	no	668	no	no	telephone	17	nov	508	4	-1	0	unknown	no
45210	37	entrepreneur	married	secondary	no	2971	no	no	cellular	17	nov	361	2	188	11	other	no

We can see that there are 17 characteristics of each client. Age ranges from **18 to 95 (average 40.9)**. The bank balance shows values from **-8,019 to 102,127,** indicating high dispersion. The **y** variable is highly unbalanced **(88% 'no' and 12% 'yes').**

```
train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45211 entries, 0 to 45210
Data columns (total 17 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   age        45211 non-null  int64 
 1   job        45211 non-null  object
 2   marital    45211 non-null  object
 3   education  45211 non-null  object
 4   default    45211 non-null  object
 5   balance    45211 non-null  int64 
 6   housing    45211 non-null  object
 7   loan       45211 non-null  object
 8   contact    45211 non-null  object
 9   day        45211 non-null  int64 
 10  month      45211 non-null  object
 11  duration   45211 non-null  int64 
 12  campaign   45211 non-null  int64 
 13  pdays      45211 non-null  int64 
 14  previous   45211 non-null  int64 
 15  poutcome   45211 non-null  object
 16  y          45211 non-null  object
dtypes: int64(7), object(10)
memory usage: 5.9+ MB
```

```
train_df1['subscription'].value_counts()
```

```
subscription
no     39922
yes     5289
Name: count, dtype: int64
```

```
# 1. Converter a variável alvo 'y' em valores binários (1 para 'yes', 0 para 'no')
train_df1['subscription'] = train_df1['subscription'].map({'yes': 1, 'no': 0})
```

```
train_df.duplicated().sum()
```

```
0
```

```
train_df.isnull().sum()
```

```
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
y            0
dtype: int64
```

```
train_df.describe()
```

	age	balance	day	duration	campaign	pdays	previous
count	45211.000000	45211.000000	45211.000000	45211.000000	45211.000000	45211.000000	45211.000000
mean	40.936210	1362.272058	15.806419	258.163080	2.763841	40.197828	0.580323
std	10.618762	3044.765829	8.322476	257.527812	3.098021	100.128746	2.303441
min	18.000000	-8019.000000	1.000000	0.000000	1.000000	-1.000000	0.000000
25%	33.000000	72.000000	8.000000	103.000000	1.000000	-1.000000	0.000000
50%	39.000000	448.000000	16.000000	180.000000	2.000000	-1.000000	0.000000
75%	48.000000	1428.000000	21.000000	319.000000	3.000000	-1.000000	0.000000
max	95.000000	102127.000000	31.000000	4918.000000	63.000000	871.000000	275.000000

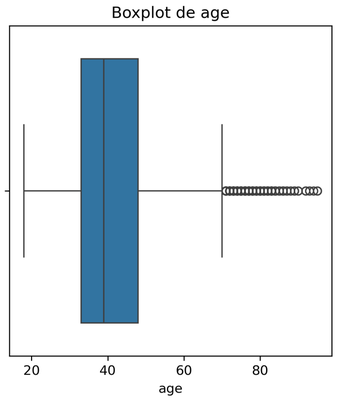
```
# 2. Criar variáveis dummy para as variáveis categóricas
train_df1 = pd.get_dummies(train_df, columns=['job', 'marital', 'education', 'default',
                                              'housing', 'loan', 'contact', 'month',
                                              'poutcome'], drop_first=True)
```

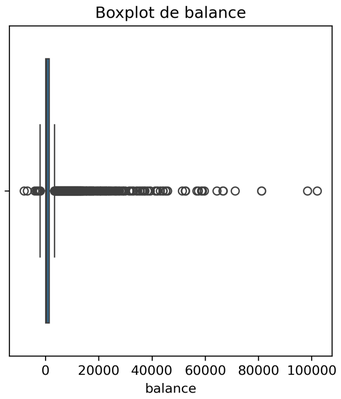
Descriptive statistics of categorical columns

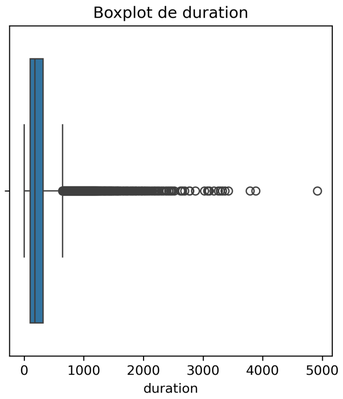
```
train_df.describe(include ='object')
```

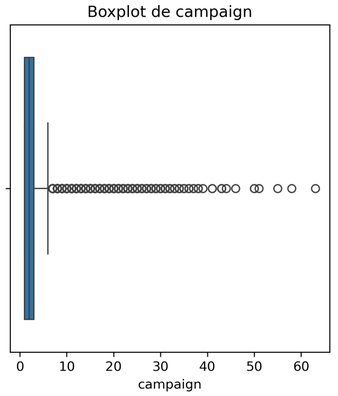
	job	marital	education	default	housing	loan	contact	month	poutcome	y
count	45211	45211	45211	45211	45211	45211	45211	45211	45211	45211
unique	12	3	4	2	2	2	3	12	4	2
top	blue-collar	married	secondary	no	yes	no	cellular	may	unknown	no
freq	9732	27214	23202	44396	25130	37967	29285	13766	36959	39922

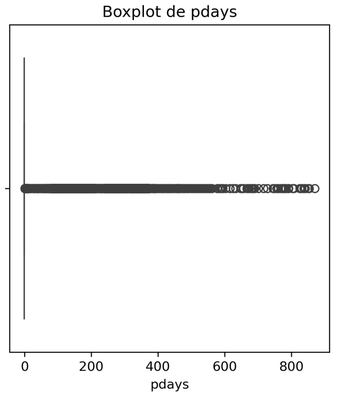
The boxplots show that the variables **balance, duration, campaign, pdays** and **previous** have significant outliers.

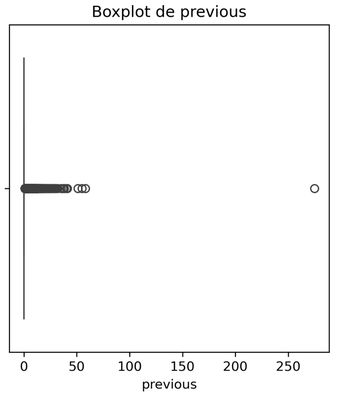






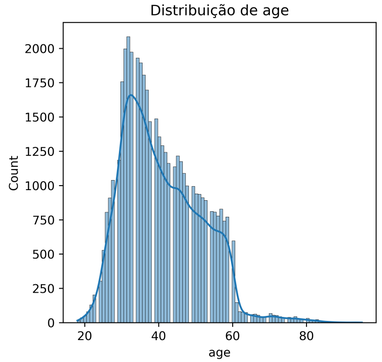




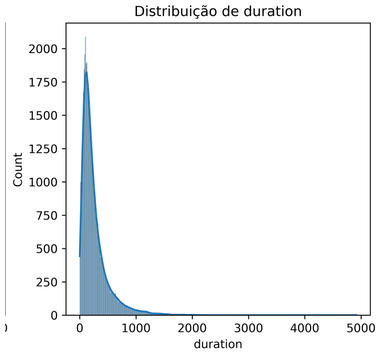


EDA Visualization

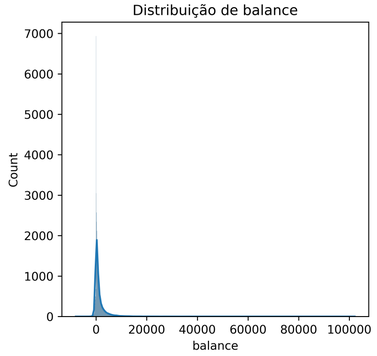
Distribution of numerical variables



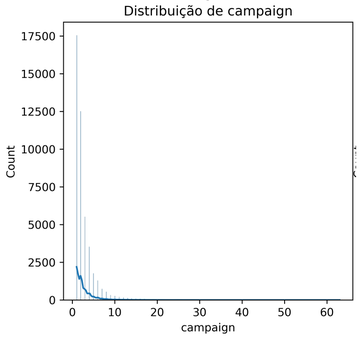
Age: The distribution is slightly asymmetrical, with the majority of clients concentrated between the ages of 30 and 50.



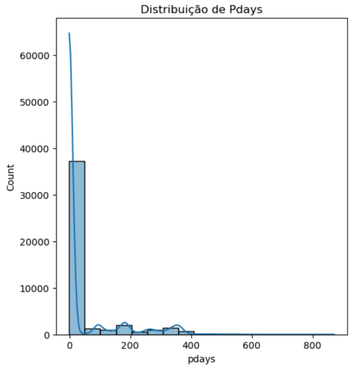
Duration: Most connections last less than 500 seconds, but there are some outliers with very long connections.



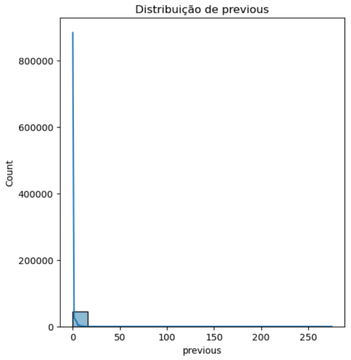
Balance: The distribution is quite uneven, with many clients showing low or negative balances, but a few with very high balances.



Campaign: Most clients were contacted once or twice, with a few cases of clients with many contacts.

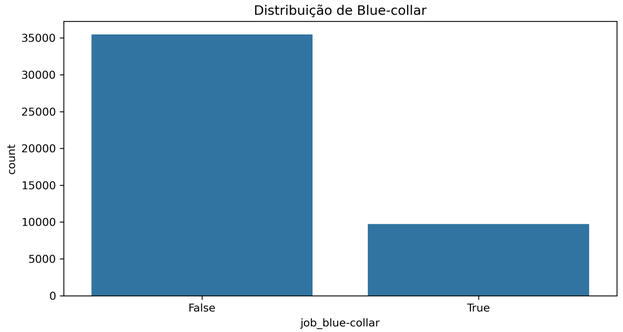


Pdays: Most customers have never been contacted before (-1 indicates this), but there are some who have been contacted before.

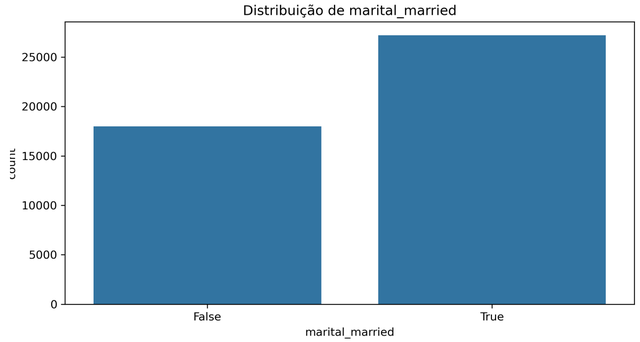


Previous: The majority of clients had no previous contacts, with a few having multiple previous contacts.

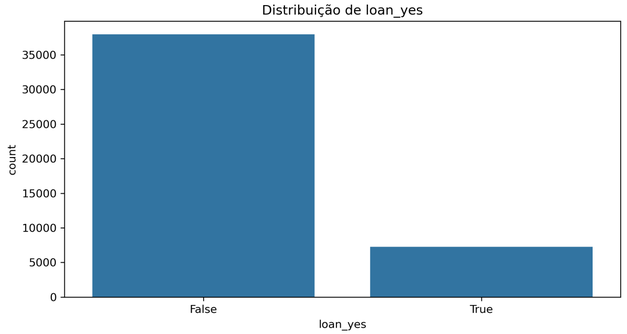
Distribution of categorical variables



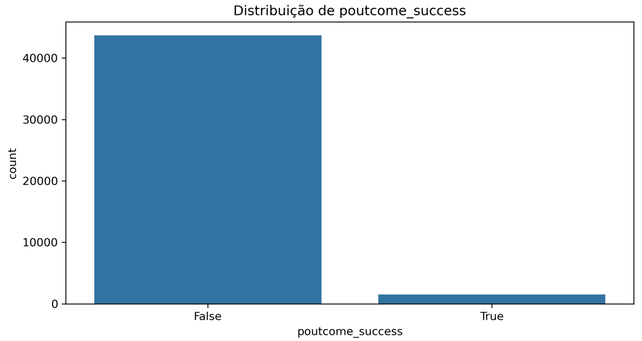
Profession: ‘Blue-collar’ is one of the most common categories among customers.



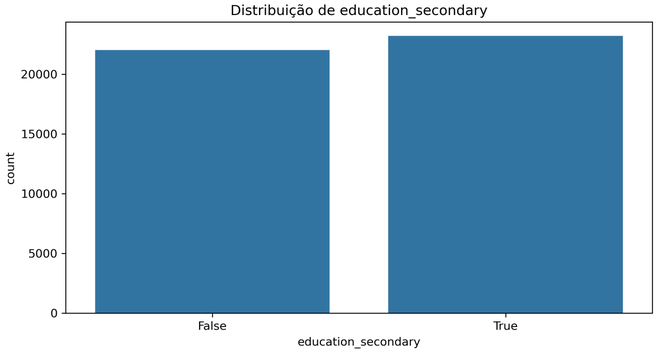
Marital status: Most clients are married.



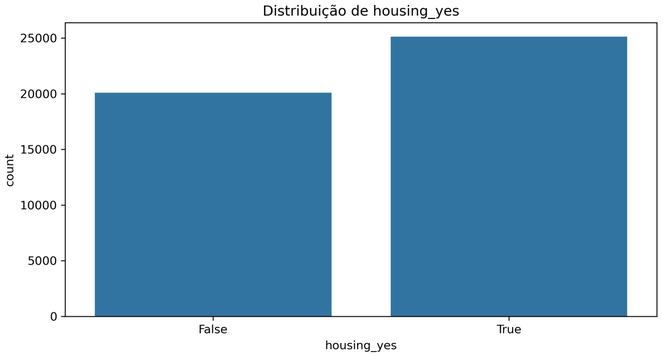
Personal loans: Fewer customers have personal loans compared to housing units



poutcome_sucess: Most clients were unsuccessful in previous campaigns



Education: Most clients have secondary education.

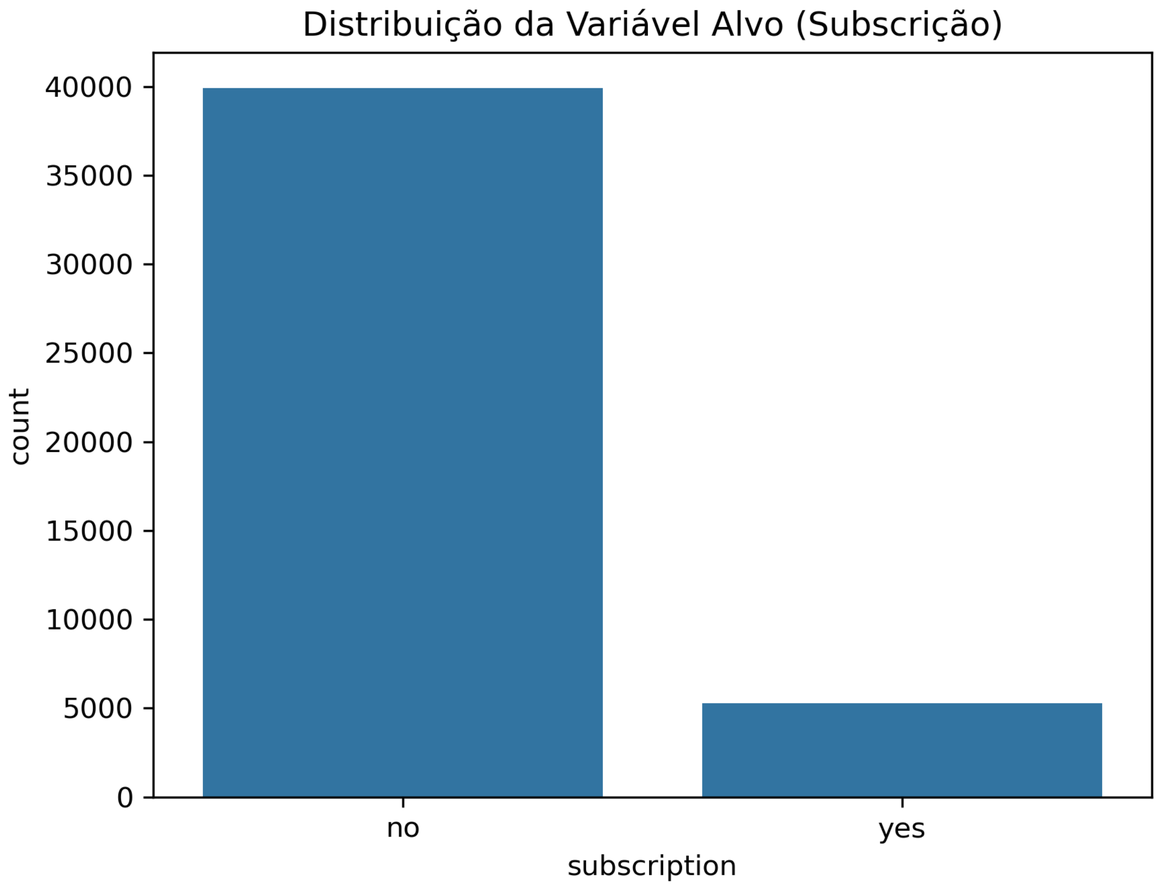


Housing loans: Many customers have a home loan.

Distribution of target variable “Subscription”

- The subscription target variable is severely unbalanced: around **88%** of customers have not subscribed **(no)**, while only **12%** have subscribed **(yes)**.

- This imbalance requires specific techniques, such as **SMOTE**, to improve the performance of predictive models.

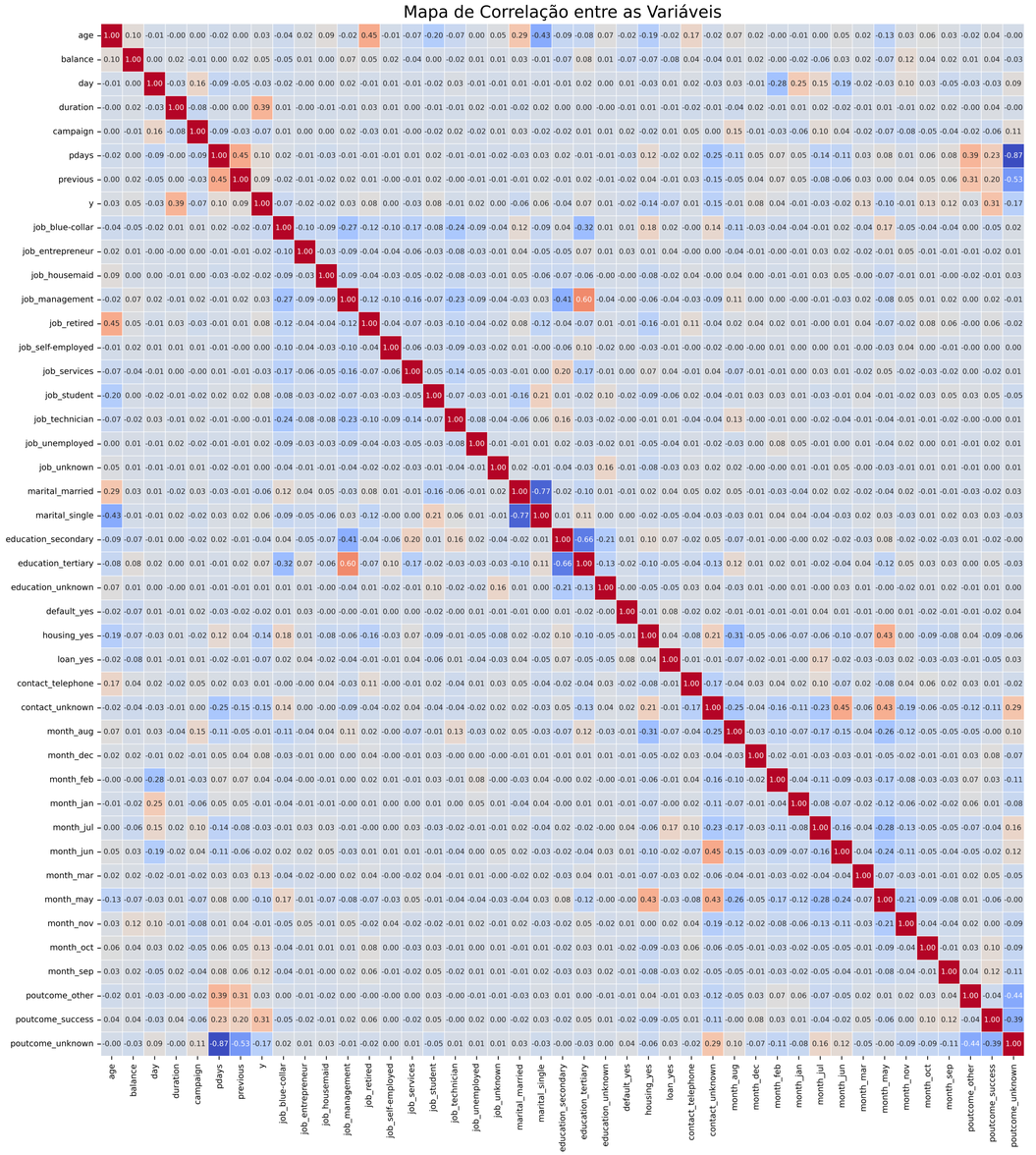


Correlation map between variables

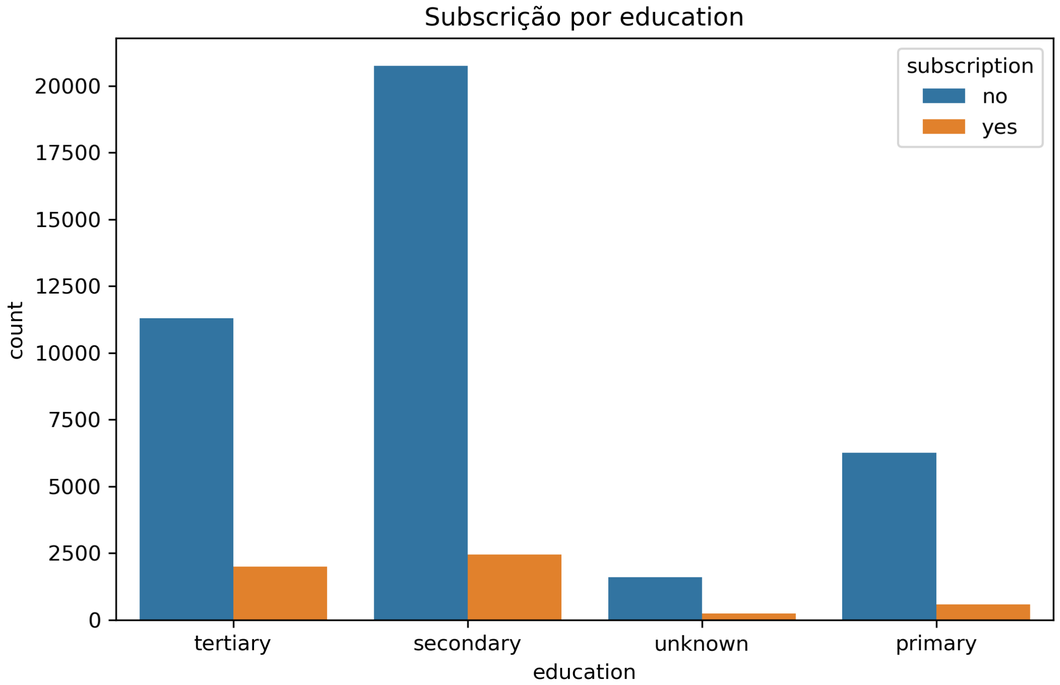
- **Correlation:** The correlation heat map gives us an insight into the relationships between numerical variables such as age, balance and duration.

- **Age:** Age can be a factor that influences a customer's decision to take out a term deposit.

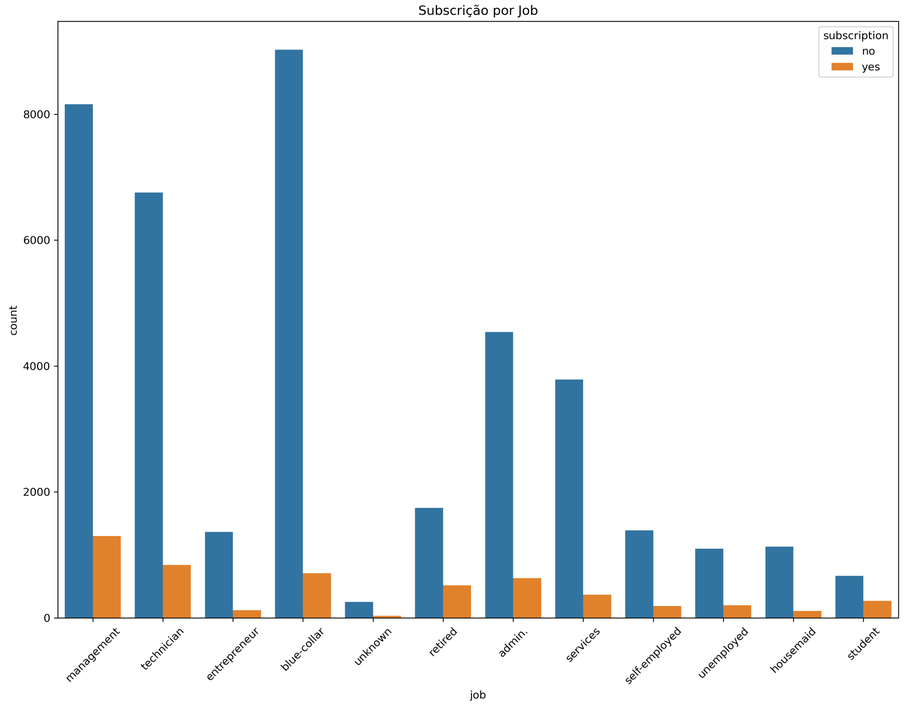
- **Distribution of the target variable (y):** There was a significant imbalance between customers who subscribed and those who did not.



Subscription by education and work

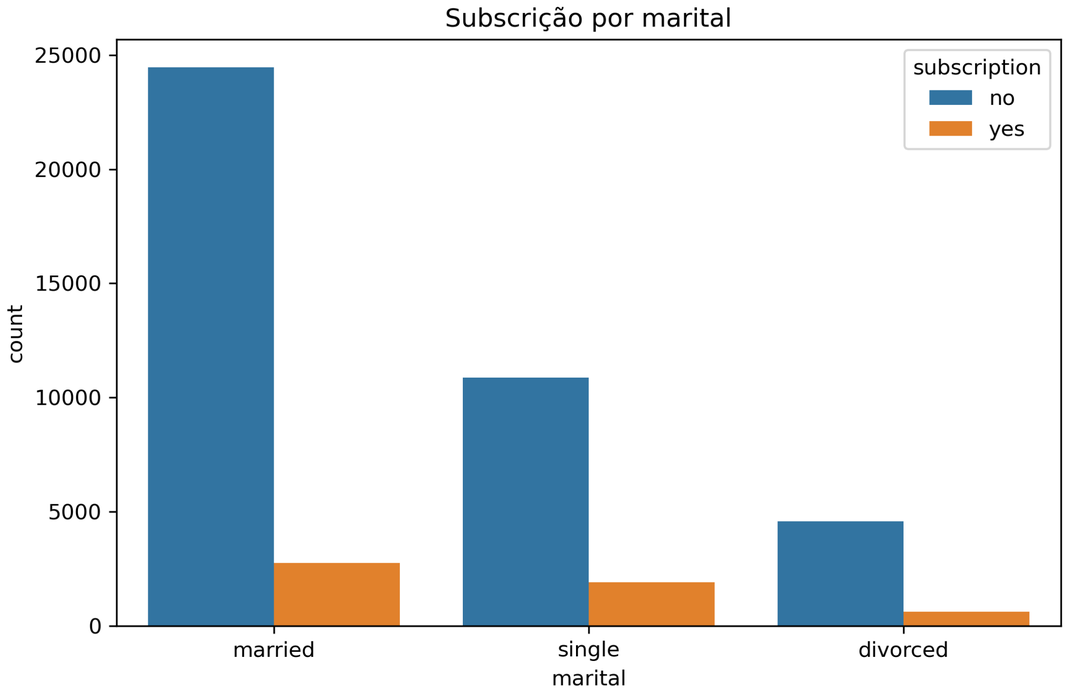


Education: Customers with a higher level of education may be more likely to apply.

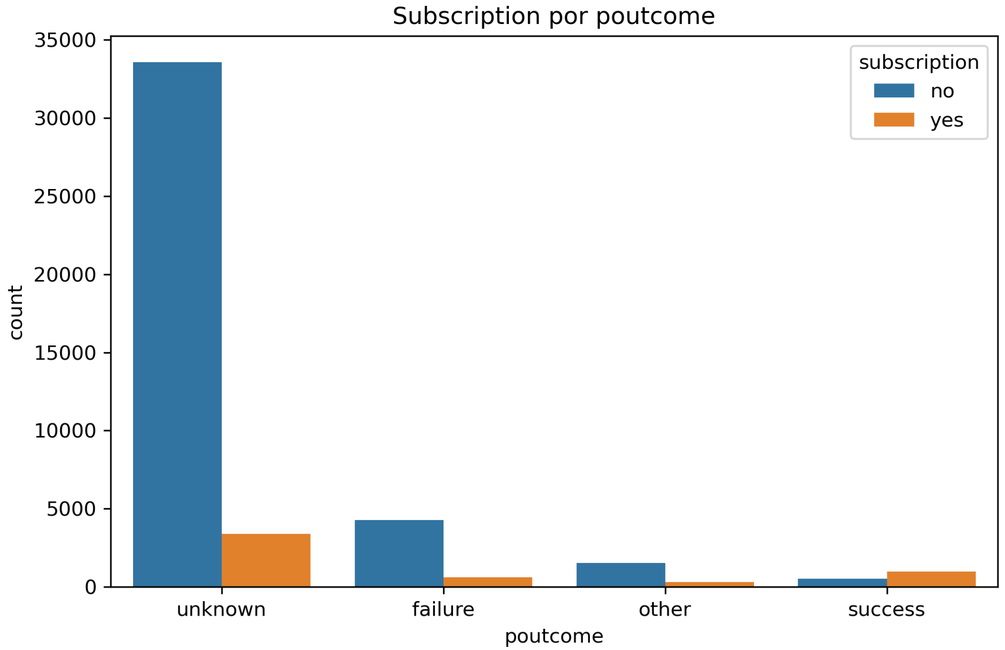


Type of Job: There is a greater tendency for underwriting among certain professions, such as management or blue-collar.

Signature by marital status and result in the previous campaign



Marital Status: Explore whether single, married or divorced customers are more likely to sign up.

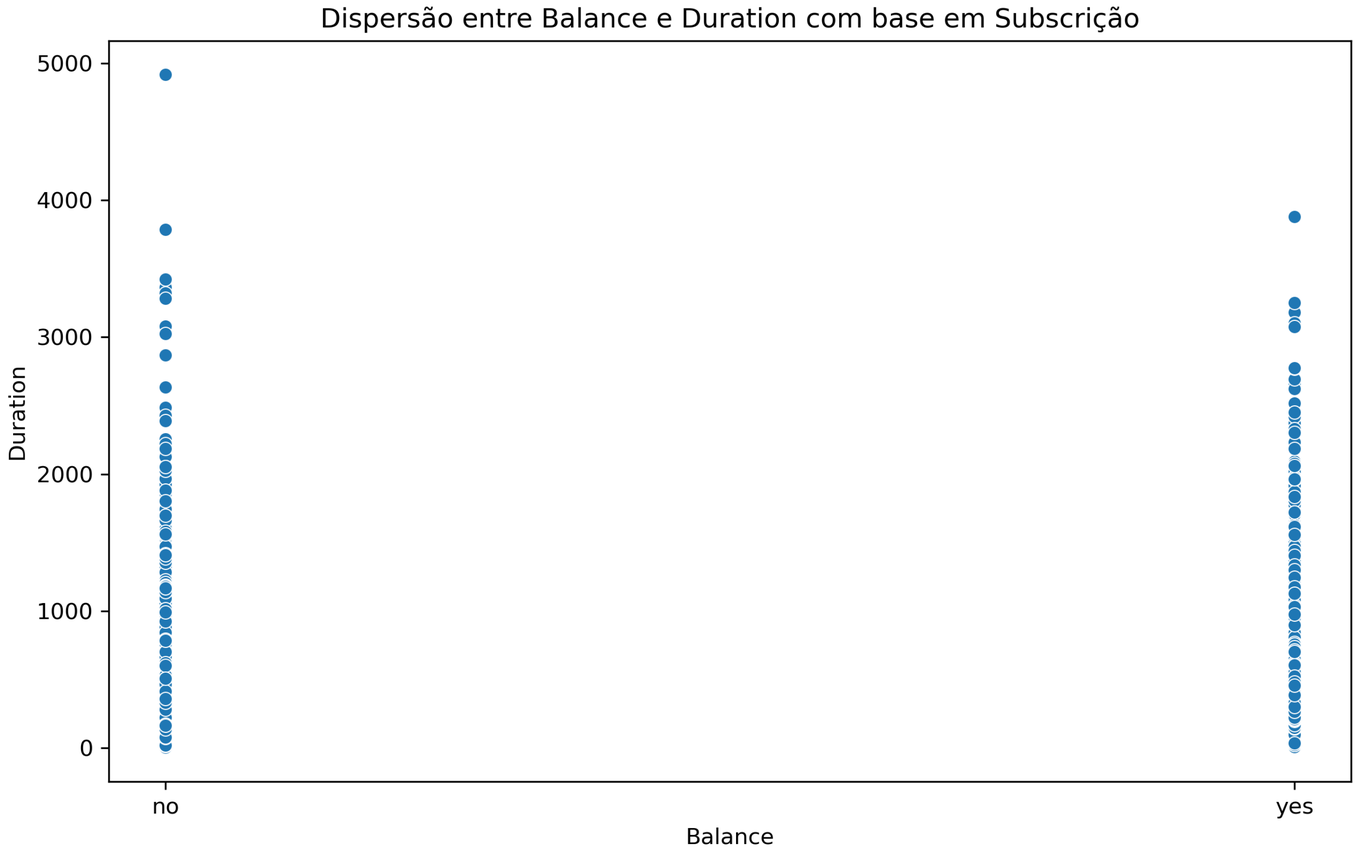


Poutcome: Customers who were “**successful**” in the previous campaign have a higher subscription rate, while “**failure**” and “**other**” have low conversions. The majority of records are “**unknown**,” indicating a lack of prior contact.

Scatter plot between balance and duration

- Customers who have subscribed (**yes**) tend to have a longer duration of contact (**duration**), while the bank balance (**balance**) shows similar dispersion in both classes.

- This reinforces the impact of longer contacts on the subscription decision.



Predictive Models

Featuring Engineering

- The target variable **subscription**, representing deposit subscription, was separated for modeling, with **39,922 (“0” not subscribed)** and **5,289 (“1” subscribed)** records.

```
# Separar a variável alvo (y) - A variável alvo é 'y' (subscrição do depósito)
X = train_df1.drop('y', axis=1)  # Excluir a coluna 'y' para usar como X (variáveis independentes)
y = train_df1['y']  # 'y' será nossa variável alvo (dependente)
```

```
train_df1['y'].value_counts()
```

```
y
0    39922
1     5289
Name: count, dtype: int64
```

- **StandardScaler** was applied for normalization.

```
# Importar o StandardScaler para normalizar os dados
from sklearn.preprocessing import StandardScaler

# Criar o scaler e ajustar nos dados de treino
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Verificar as primeiras linhas dos dados normalizados
print("Primeiras linhas dos dados normalizados (X_train):")
print(X_train_scaled[:5])
```

- The data was divided into **75% for training** (33,908 records) and **25% for testing** (11,303 records).

```
# Dividir os dados em conjunto de treino e teste (75% treino, 25% teste)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, stratify=y, random_state=42)

# Verificar as dimensões dos conjuntos de treino e teste
print("Tamanho do conjunto de treino:", X_train.shape)
print("Tamanho do conjunto de teste:", X_test.shape)
```

```
Tamanho do conjunto de treino: (33908, 42)
Tamanho do conjunto de teste: (11303, 42)
```

- **SMOTE** to balance classes in the training set, resulting in equal proportions **(29,941 for each class).**

```
# Importar a biblioteca SMOTE
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split

# Aplicar SMOTE para balanceamento de classes
smote = SMOTE(random_state=42)
X_train_balanced, y_train_balanced = smote.fit_resample(X_train_scaled, y_train)

# Verificar o balanceamento das classes após o SMOTE
print("Distribuição das classes após o SMOTE:", np.bincount(y_train_balanced))
```

```
Distribuição das classes após o SMOTE: [29941 29941]
```

Logistic Regression

Classification accuracy

The **logistic regression** model was adjusted to address imbalance, achieving **91% AUC-ROC.** It identified **82%** of customers who signed up, maintaining high overall accuracy. Techniques such as weight adjustment and automatic balancing have improved performance.

```python
from sklearn.linear_model import LogisticRegression

# Passo 1: Usar 'class_weight=balanced' para lidar com o desbalanceamento de classes
log_model_balanced = LogisticRegression(max_iter=1000, random_state=42, class_weight='balanced')
log_model_balanced.fit(X_train_scaled, y_train)

# Previsões no conjunto de teste
y_pred_log_balanced = log_model_balanced.predict(X_test_scaled)

# Avaliar o modelo
print("Relatório de Classificação - Regressão Logística com Balanceamento")
print(classification_report(y_test, y_pred_log_balanced))
print("AUC-ROC Score:", roc_auc_score(y_test, log_model_balanced.predict_proba(X_test_scaled)[:, 1]))
```

```
Relatório de Classificação - Regressão Logística com Balanceamento
              precision    recall  f1-score   support

           0       0.97      0.85      0.91      9981
           1       0.42      0.82      0.56      1322

    accuracy                           0.85     11303
   macro avg       0.70      0.84      0.73     11303
weighted avg       0.91      0.85      0.87     11303

AUC-ROC Score: 0.9085716719558388
```

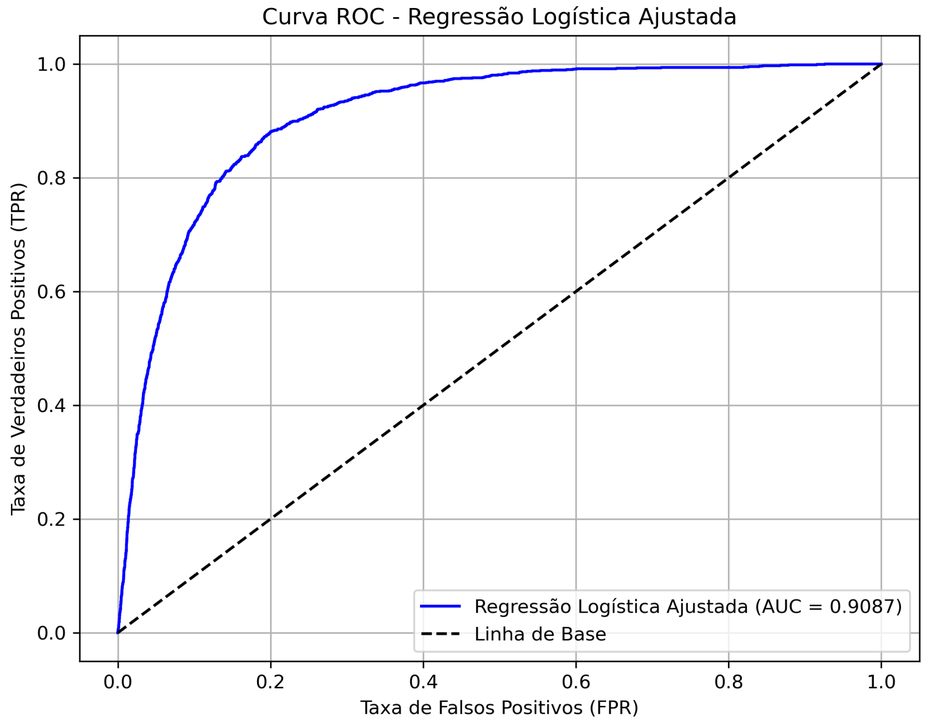
```python
# Melhor combinação de hiperparâmetros
print("Melhores Hiperparâmetros:", grid_search.best_params_)
```

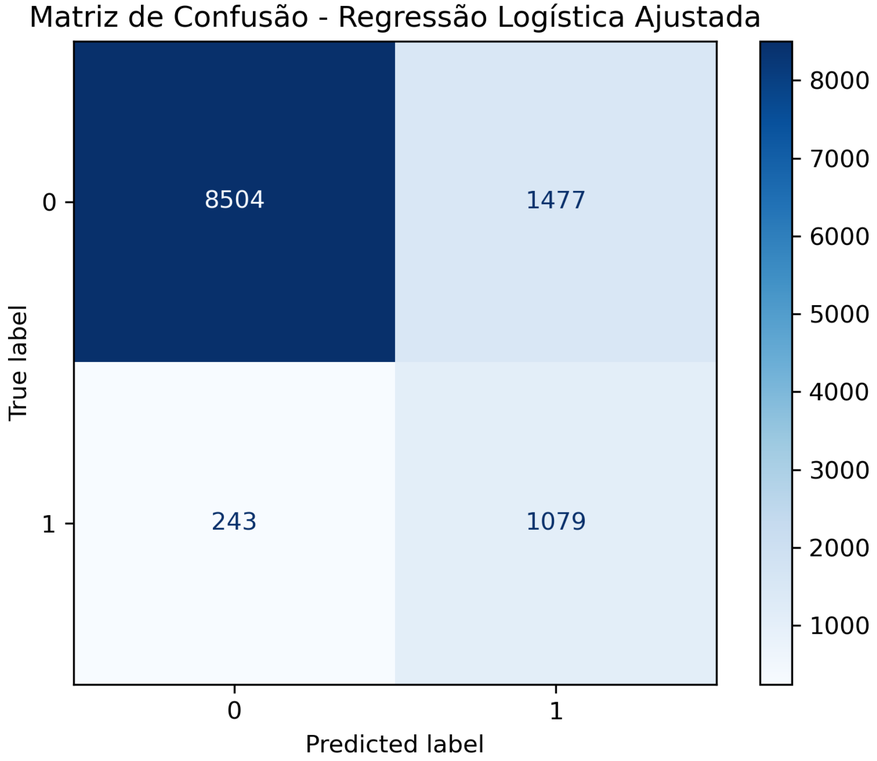
```
Melhores Hiperparâmetros: {'C': 0.1, 'class_weight': 'balanced', 'penalty': 'l1', 'solver': 'liblinear'}
```

```python
# Previsões com o melhor modelo encontrado
best_log_model = grid_search.best_estimator_
y_pred_best_log = best_log_model.predict(X_test_scaled)

# Avaliar o modelo ajustado
print("Relatório de Classificação - Regressão Logística Ajustada")
print(classification_report(y_test, y_pred_best_log))
print("AUC-ROC Score:", roc_auc_score(y_test, best_log_model.predict_proba(X_test_scaled)[:, 1]))
```

```
Relatório de Classificação - Regressão Logística Ajustada
              precision    recall  f1-score   support

           0       0.97      0.85      0.91      9981
           1       0.42      0.82      0.56      1322

    accuracy                           0.85     11303
   macro avg       0.70      0.83      0.73     11303
weighted avg       0.91      0.85      0.87     11303

AUC-ROC Score: 0.9086762579612307
```

The **ROC curve** shows that the model has excellent performance with 91% overall accuracy **(AUC-ROC = 0.91)**. The **confusion matrix** indicates that the model correctly identified the majority of customers who did not sign (**8,504**) and correctly identified **1,079 of the 1,322** who signed. This demonstrates that the model is well adjusted to predict who is most likely to sign.





K-Nearest Neighbors

Classification accuracy

The adjusted **KNN** model improved the identification of customers who subscribed (66% recall vs. 27% in the baseline). With an **AUC-ROC of 84%**, the adjustment balanced the performance between precision and recall, making it more effective at prediction.

```python
# Melhor modelo encontrado
best_knn = grid_knn.best_estimator_
```

```python
# Previsões no conjunto de teste
y_pred_knn = best_knn.predict(X_test_scaled)
```

```python
# Avaliar o modelo
print("Relatório de Classificação - KNN")
print(classification_report(y_test, y_pred_knn))
print("AUC-ROC Score:", roc_auc_score(y_test, best_knn.predict_proba(X_test_scaled)[:, 1]))
```

```
Relatório de Classificação - KNN
              precision    recall  f1-score   support

           0       0.91      0.98      0.94      9981
           1       0.61      0.27      0.38      1322

    accuracy                           0.89     11303
   macro avg       0.76      0.63      0.66     11303
weighted avg       0.88      0.89      0.88     11303

AUC-ROC Score: 0.8412394669387723
```

```python
# Melhor modelo encontrado
best_knn = grid_knn.best_estimator_
print("Melhor KNN encontrado:", best_knn)
```

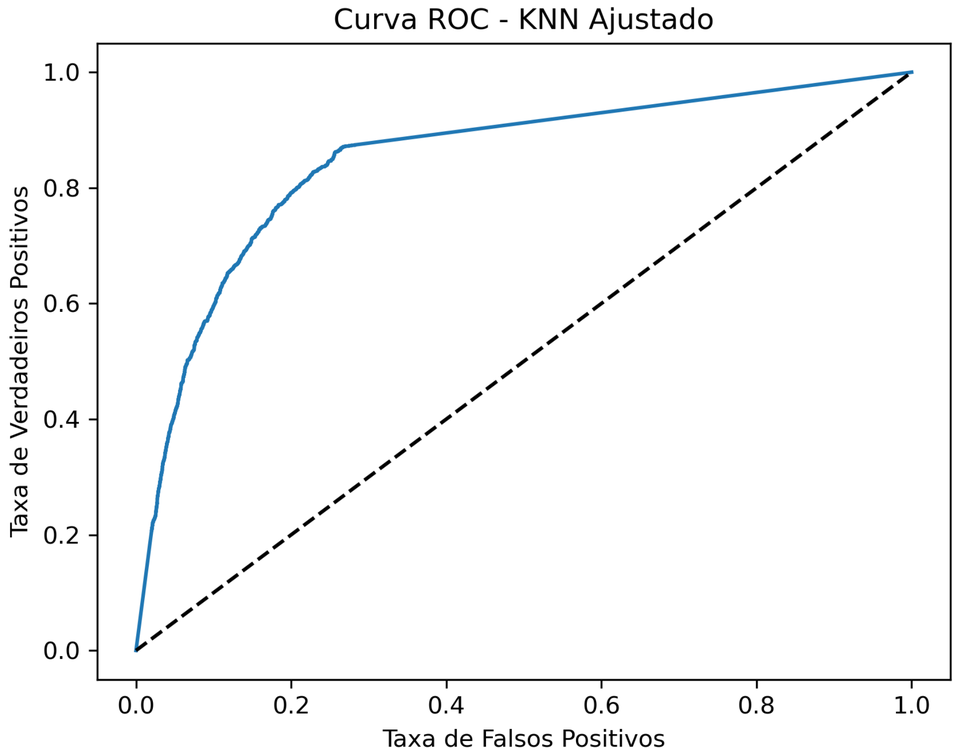
```
Melhor KNN encontrado: KNeighborsClassifier(metric='manhattan', n_neighbors=9, weights='distance')
```

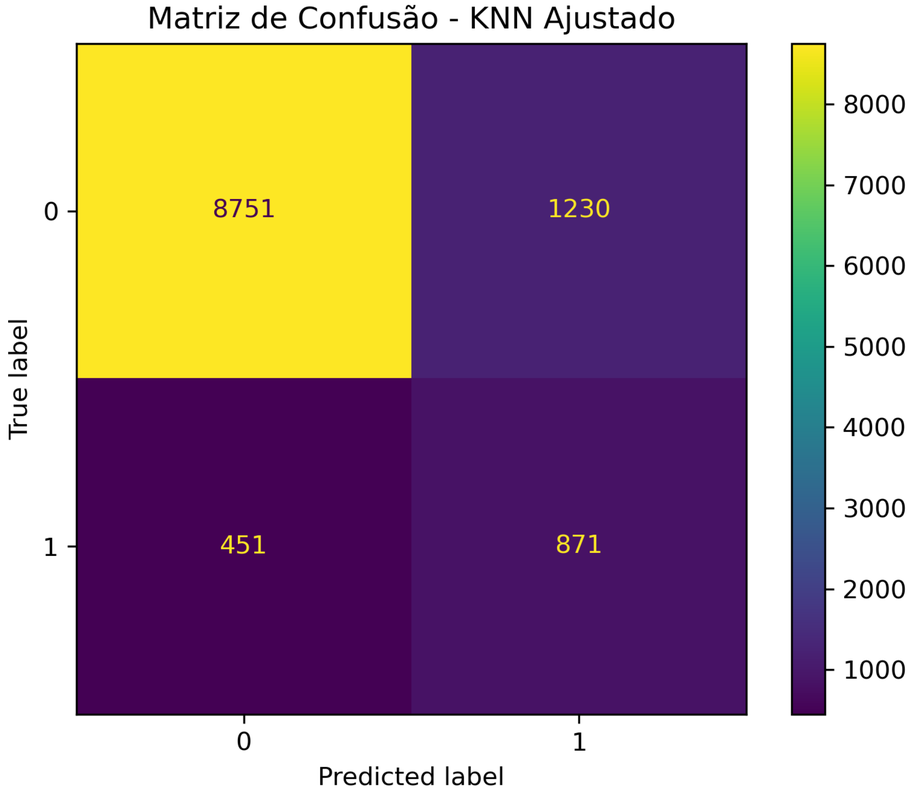
```python
# Previsões no conjunto de teste
y_pred_knn = best_knn.predict(X_test_scaled)
```

```python
# Avaliar o modelo
from sklearn.metrics import classification_report, roc_auc_score

print("Relatório de Classificação - KNN (Ajustado)")
print(classification_report(y_test, y_pred_knn))
print("AUC-ROC Score:", roc_auc_score(y_test, best_knn.predict_proba(X_test_scaled)[:, 1]))
```

```
Relatório de Classificação - KNN (Ajustado)
              precision    recall  f1-score   support

           0       0.95      0.88      0.91      9981
           1       0.41      0.66      0.51      1322

    accuracy                           0.85     11303
   macro avg       0.68      0.77      0.71     11303
weighted avg       0.89      0.85      0.87     11303

AUC-ROC Score: 0.8495291583509424
```

The **adjusted KNN** achieved an **AUC-ROC of 84.9%**, correctly identifying **871** customers who subscribed and **87.5%** of those who did not. Despite the improvement, **451** customers who subscribed were lost, showing good overall performance, but with room for refinement.





Random Forest

Classification accuracy

The **basic Random Forest** model had an **AUC-ROC of 92.5%**, with a precision of 92% and a recall of 40% for customers who subscribed. After adjustments (increasing estimators to 300), the model achieved a similar **AUC-ROC (92.4%),** but improved the recall of the '**yes**' **class from 40% to 61%,** indicating a greater ability to identify customers who subscribed, despite a slight reduction in the precision of the '**no**' class. The **adjusted model** showed a better balance between precision and recall, being more effective in general predictions.

```python
# Definir o modelo Random Forest
rf_model = RandomForestClassifier(random_state=42)

# Ajustar o modelo Random Forest
rf_model.fit(X_train_scaled, y_train)

# Previsões no conjunto de teste
y_pred_rf = rf_model.predict(X_test_scaled)

# Avaliar o modelo
print("Relatório de Classificação - Random Forest")
print(classification_report(y_test, y_pred_rf))
print("AUC-ROC Score:", roc_auc_score(y_test, rf_model.predict_proba(X_test_scaled)[:, 1]))
```

```
Relatório de Classificação - Random Forest
              precision    recall  f1-score   support

           0       0.92      0.97      0.95      9981
           1       0.67      0.40      0.50      1322

    accuracy                           0.91     11303
   macro avg       0.80      0.69      0.72     11303
weighted avg       0.89      0.91      0.90     11303

AUC-ROC Score: 0.9259175640979586
```

```python
# Melhor modelo encontrado
best_rf = grid_rf.best_estimator_
print("Melhor Random Forest encontrado:", best_rf)
```

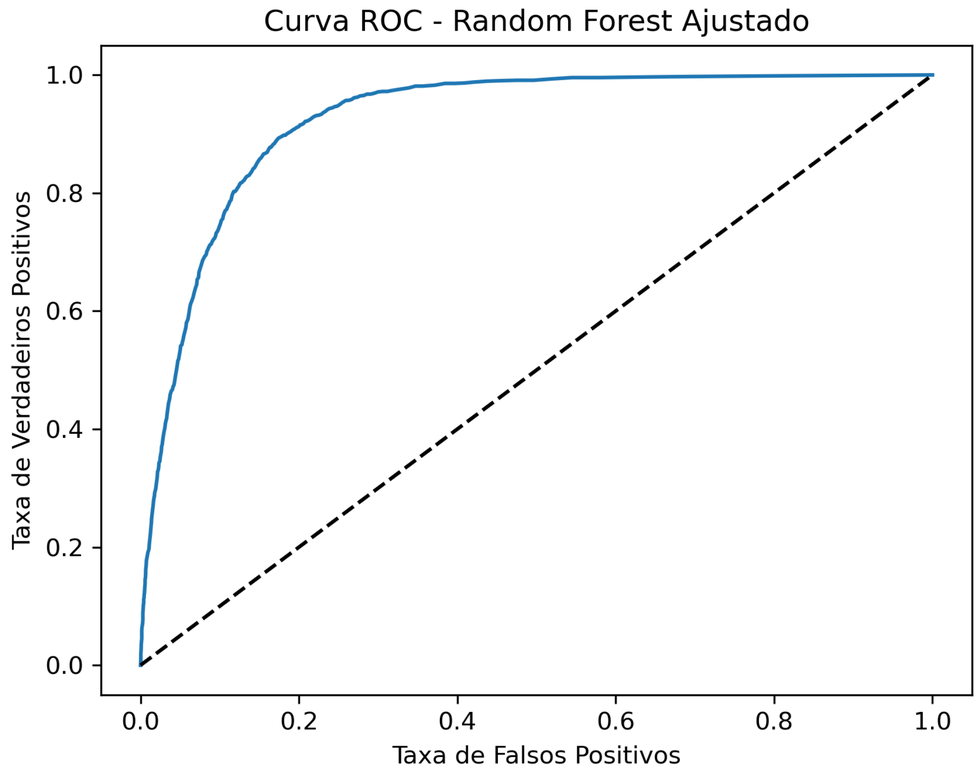
```
Melhor Random Forest encontrado: RandomForestClassifier(n_estimators=300, random_state=42)
```

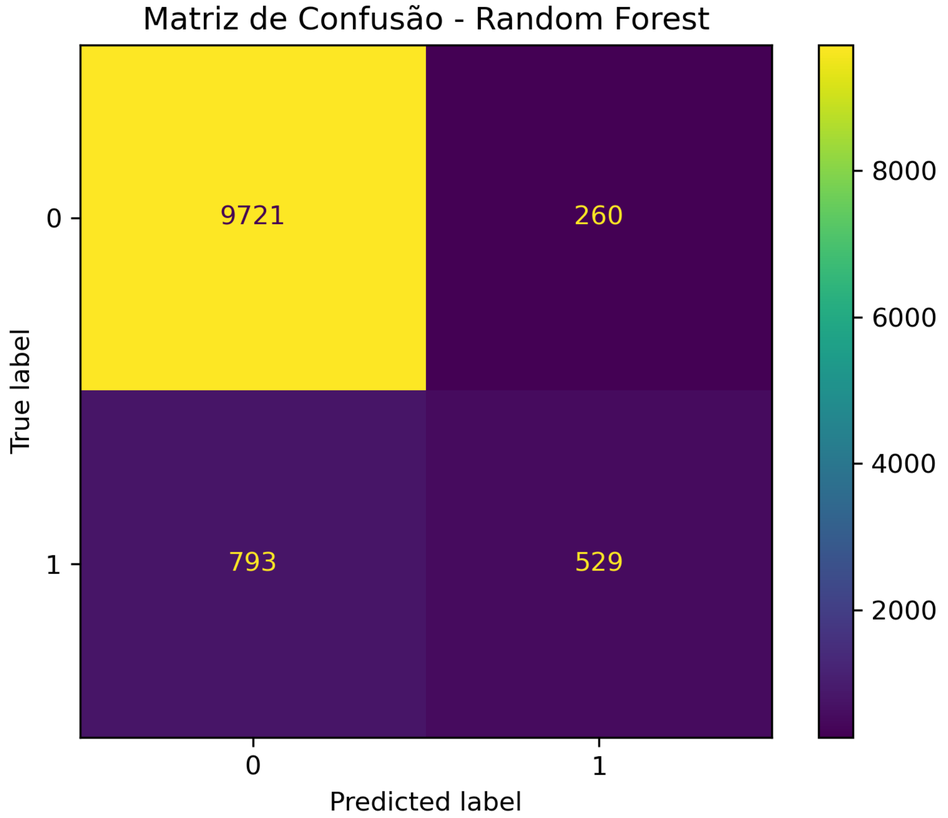
```python
# Previsões no conjunto de teste
y_pred_rf = best_rf.predict(X_test_scaled)
```

```python
# Avaliar o modelo
from sklearn.metrics import classification_report, roc_auc_score

print("Relatório de Classificação - Random Forest (Ajustado)")
print(classification_report(y_test, y_pred_rf))
print("AUC-ROC Score (Ajustado):", roc_auc_score(y_test, best_rf.predict_proba(X_test_scaled)[:, 1]))
```

```
Relatório de Classificação - Random Forest (Ajustado)
              precision    recall  f1-score   support

           0       0.95      0.94      0.94      9981
           1       0.56      0.61      0.58      1322

    accuracy                           0.90     11303
   macro avg       0.76      0.77      0.76     11303
weighted avg       0.90      0.90      0.90     11303

AUC-ROC Score (Ajustado): 0.9246665866356365
```

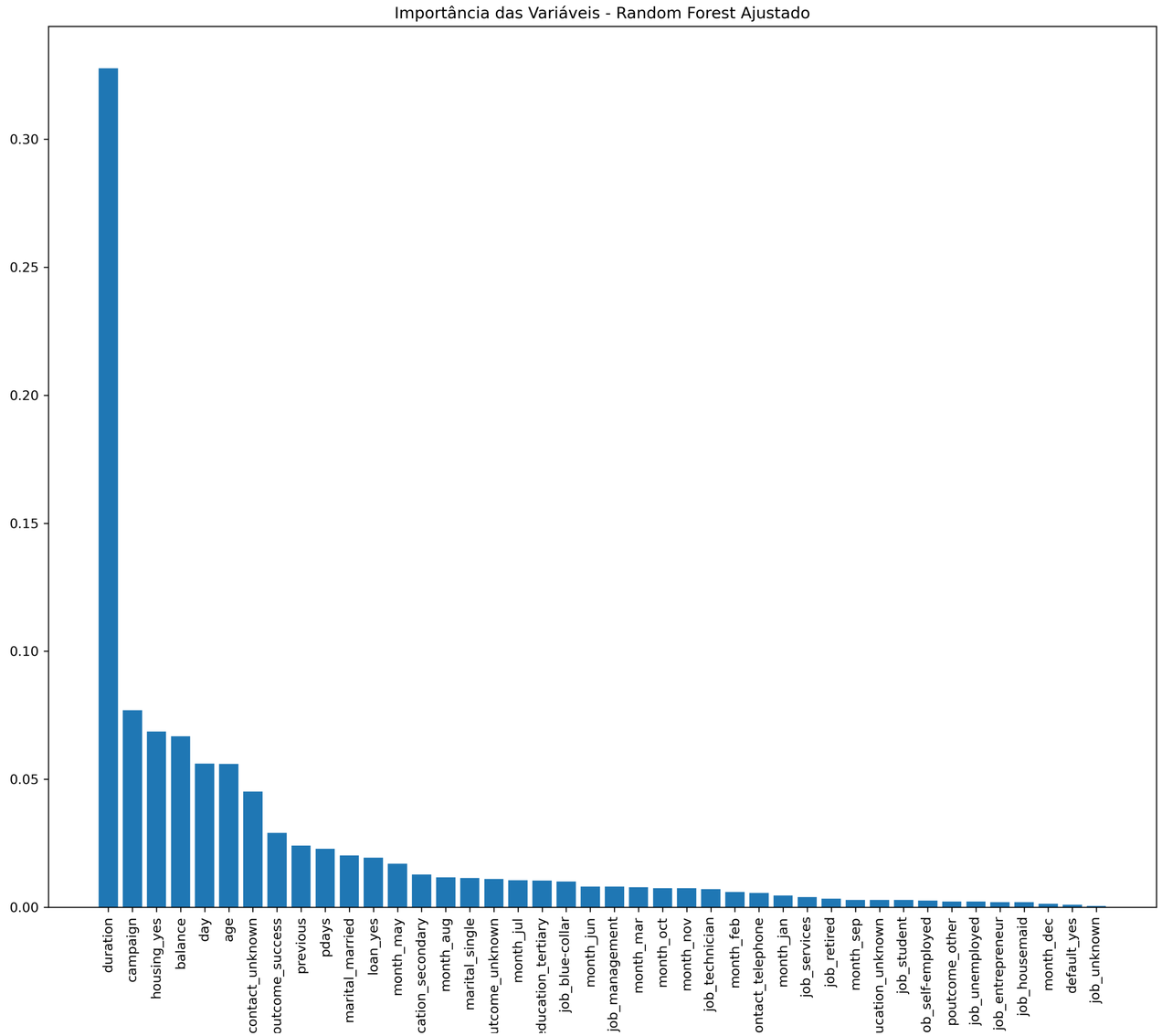
The **adjusted Random Forest** achieved an **AUC-ROC of 92.4%**, showing excellent performance. It correctly identified **802** customers who subscribed and **93.5%** of those who did not. Although **520** subscribed customers were not detected, the model is highly effective for general predictions.





Importance of variables

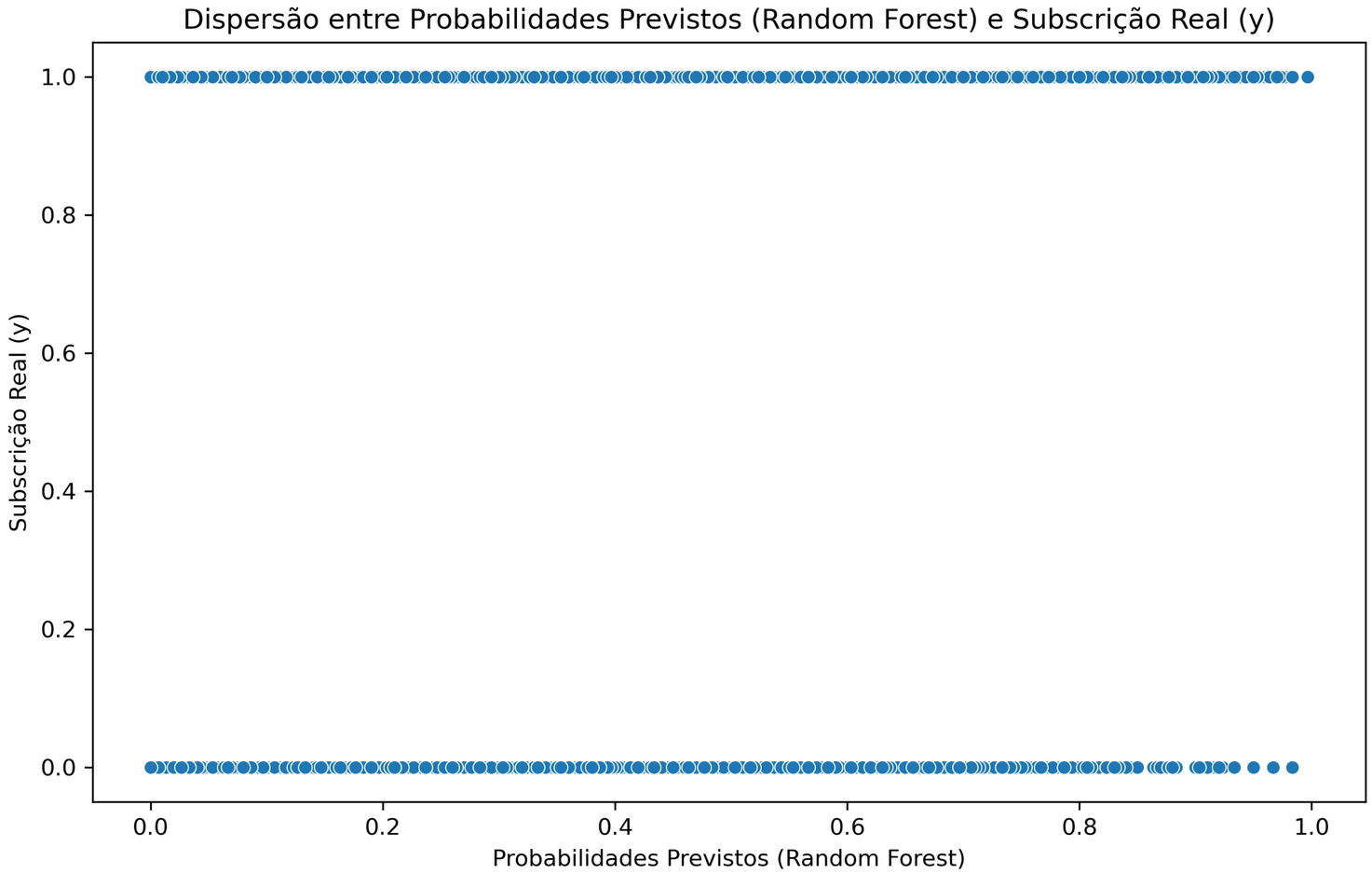
- The duration variable had the greatest impact on forecasts, followed by **campaign** (number of contacts made) and **housing_yes** (housing loan).
- Other relevant variables include **balance** and **age**. Variables such as **poutcome_success** (result of previous campaign) also had an influence, but to a lesser extent.
- This shows that factors related to direct interaction and customers' financial history are crucial for predicting underwriting.



Predicted Probabilities and Actual Subscription (y)

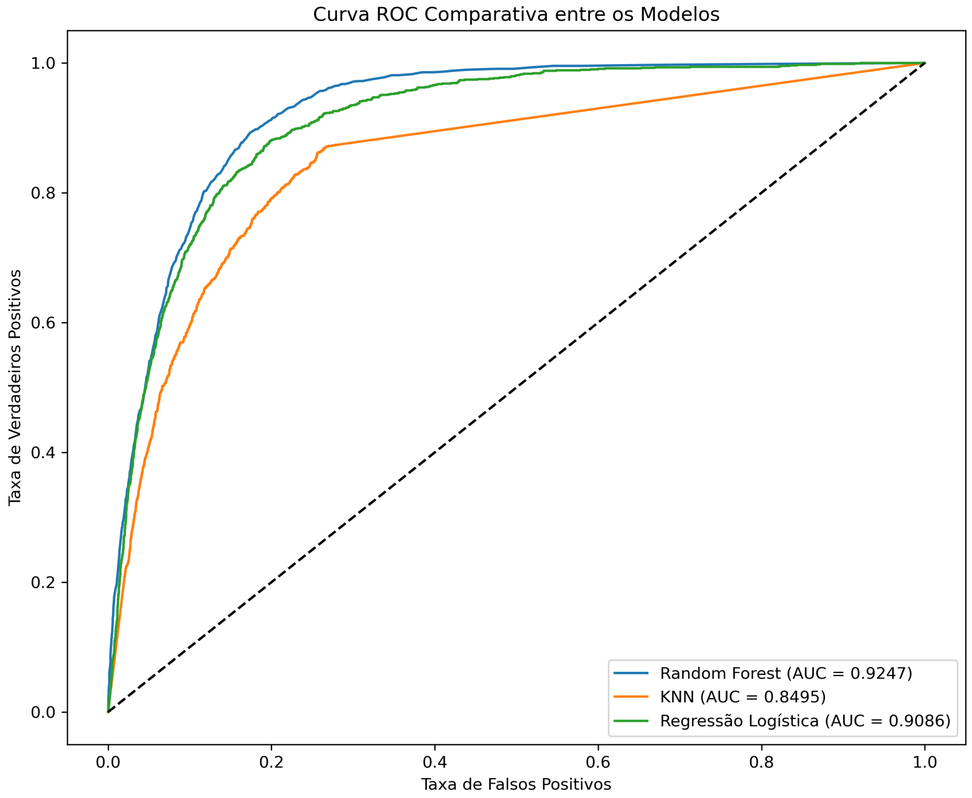
- The graph shows the relationship between the probabilities predicted by the Random Forest model and the actual subscription values.

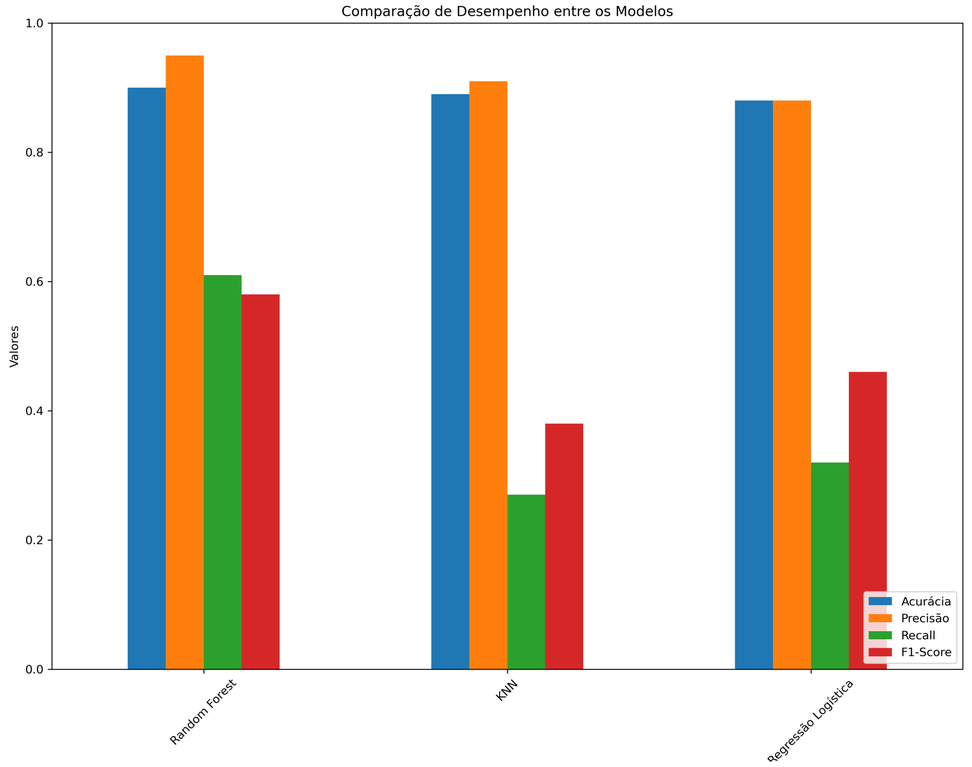
- Probabilities close to 1 indicate customers who probably subscribed, while low values represent those who did not subscribe.



Comparison Between Models

Random Forest was the best model, with an AUC-ROC of **92.4%** and a balance between precision and recall. **Logistic Regression** performed well (AUC-ROC **90.8%**), but had lower recall. **KNN** was less effective (AUC-ROC **84.9%**) due to poor identification of the positive class.





Conclusion

- Target campaigns at customers with a high probability of subscribing, **based on the model's predictions.**

- Personalise messages taking into account key variables such **as duration, balance and age.**

- Adjust strategies based on the patterns identified, such **as prioritising customers who have been successful in previous campaigns.**

- **Develop retention and loyalty strategies** for customers who have already subscribed to deposits, increasing the long-term relationship.

Next Steps

1. **Model implementation:** Integrate the Random Forest model into the marketing system to prioritise customers most likely to subscribe.

2. **Model Refinement:** Explore ensemble models (such as Stacking) to further improve minority class identification.

3. **Temporal Analysis:** Evaluate seasonal patterns that may influence subscription and adjust campaigns accordingly.

4. **Additional Data Collection:** Reduce the proportion of ‘unknown’ values in important variables to improve the quality of the model.

5. **Ongoing Monitoring:** Evaluate the performance of the model in production, refining it with new data and feedback from campaigns.

Thank You