Week 11 — Explain the model, analyze risk, bias and ethical considerations

In [1]:
```
import pandas as pd
import numpy as np
# import dask.dataframe as dd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from datetime import datetime
from sklearn.metrics import roc_auc_score, f1_score, confusion_matrix
from sklearn.linear_model import LogisticRegression
from collections import defaultdict
from sklearn.preprocessing import MinMaxScaler
from joblib import Parallel, delayed
```

In [2]:
```
pd.set_option('display.max_columns', None)

train = pd.read_csv('train_final.csv', low_memory=False)
validation = pd.read_csv('val_set_final.csv')
test = pd.read_csv('test_4_11.csv')
```

In [3]:
```
train.head()
```

Out[3]:

	Unnamed: 0	date	customer_code	employee_index	country_spain	female	age	new_cust	sen
0	0	2016-04-28	1334092	N	1	0	0.234694	0	
1	1	2015-07-28	1024586	N	1	0	0.234694	0	
2	2	2016-04-28	856204	N	1	0	0.306122	0	
3	3	2015-08-28	295807	N	1	0	0.489796	0	
4	4	2016-03-28	942624	N	1	1	0.224490	0	

In [4]:
```
validation.head()
```

Out[4]:

	Unnamed: 0	date	customer_code	employee_index	country_spain	female	age	first_contract_(
0	0	2016-05-28	1212130	N	1	0	0.204082	2013-1
1	1	2015-07-28	84306	N	1	0	0.500000	1998-0
2	2	2015-07-28	883630	N	1	0	0.418367	2010-0
3	3	2016-05-28	1464700	N	1	1	0.183673	2015-0
4	4	2015-12-28	487783	N	1	1	0.418367	2004-1

In [5]:
```
test.head()
```

Out[5]:

	Unnamed: 0	date	customer_code	employee_index	country_spain	female	age	new_cust	sen
0	0	2015-06-28	49335	N	1	0	0.734694	0	
1	1	2016-02-28	1174349	N	1	0	0.214286	0	
2	2	2015-07-28	1393286	N	1	0	0.244898	1	
3	3	2016-03-28	1454346	N	1	0	0.183673	0	
4	4	2016-02-28	1074431	N	1	0	0.234694	0	

Changing columns name and dropping columns so both datasets are the same

In [6]:
```
train = train.rename(columns={'country': 'country_spain'})
```

In [7]:
```
train = train.drop(columns=['Unnamed: 0'])
validation = validation.drop(columns=['Unnamed: 0'])
drop = ['join_channel', 'province_name', 'employee_index', 'segment', 'total_products'
train = train.drop(columns=drop)
validation = validation.drop(columns=drop + ['payroll_acct.1', 'first_contract_date',

test = test.drop(columns=['Unnamed: 0'])
test = test.drop(columns=drop + ['payroll_acct.1'])
```

Reading into the data

Setting products we want to predict

In [8]:
```
products = ['savings_acct', 'guarantees', 'current_acct',
       'derivada_acct', 'payroll_acct', 'junior_acct', 'mas_particular_acct',
       'particular_acct', 'particular_plus_acct', 'short_term_depo',
       'medium_term_depo', 'long_term_depo', 'e_acct', 'funds', 'mortgage',
       'pension', 'loans', 'taxes', 'credit_card', 'securities', 'home_acct',
       'pensions_2', 'direct_debt']
```

Transformation #1

Change #1: Instead of dropping these duplicates on customer column and use only the last instance we will keep those duplicates since it could capture some patterns such as if a client buys product x first, it will likely buy y product next.

We will create copies of the original train and test datasets so we don't change the original one.

In [9]:
```
train_2 = train.copy()
test_2 = test.copy()
```

Pre-processing

Transformation #2

For tranformation #2 we will add the date column as one of the features. For that, we will calculate the time since purchase using the month we are trying to predict on June 2016. For this transformation to make sense, we will also keep the first transformation, since the time line of purchase matters now, we will keep the duplicate clients' purchases instead of only keeping the last one

In [10]:
```
train_2['date'] = pd.to_datetime(train_2['date'], format='%Y-%m-%d')

train_2['date'] = train_2['date'].dt.to_period('M').dt.to_timestamp()

# Setting our prediction date, June 28, 2016, as the reference date
reference_date = pd.to_datetime("2016-06-28")

# Calculate time since purchase
train_2['months_since_purchase'] = (reference_date.year - train_2['date'].dt.year) * 1
                                   (reference_date.month - train_2['date'].dt.month)

print(train_2[['date', 'months_since_purchase']])
```

```
              date  months_since_purchase
0       2016-04-01                      2
1       2015-07-01                     11
2       2016-04-01                      2
3       2015-08-01                     10
4       2016-03-01                      3
...            ...                    ...
5757281 2016-05-01                      1
5757282 2015-08-01                     10
5757283 2015-11-01                      7
5757284 2016-05-01                      1
5757285 2016-01-01                      5

[5757286 rows x 2 columns]
```

In [11]:
```
# Adding feature on test dateased
test_2['date'] = pd.to_datetime(test_2['date'], format='%Y-%m-%d')
test_2['date'] = test_2['date'].dt.to_period('M').dt.to_timestamp()

test_2['months_since_purchase'] = (reference_date.year - test_2['date'].dt.year) * 12
                                  (reference_date.month - test_2['date'].dt.month)

print(test_2[['date', 'months_since_purchase']])
```

```
              date  months_since_purchase
0       2015-06-01                     12
1       2016-02-01                      4
2       2015-07-01                     11
3       2016-03-01                      3
4       2016-02-01                      4
...            ...                    ...
1236739 2016-02-01                      4
1236740 2016-02-01                      4
1236741 2015-08-01                     10
1236742 2016-05-01                      1
1236743 2016-04-01                      2

[1236744 rows x 2 columns]
```

In [12]:
```
X_train_2 = train_2.drop(['customer_code', 'date'] + products, axis=1)
y_train_2 = train_2[products]

X_test_2 = test_2.drop(['customer_code', 'date'] + products, axis=1)
y_test_2 = test_2[products]
```

Training

In [13]:
```
# Defining the best training parameter
params = {'C': 10, 'solver': 'liblinear', 'max_iter': 300}
```

Database with second transformation

In [14]:
```
# Initialize dictionary for storing metrics
metrics = defaultdict(lambda: defaultdict(dict))

# Train and evaluate the model on the 'train_2' dataset
for product in products:
```

```
    clf = LogisticRegression(**params)

    # Train data and labels for each product
    y_train_2_product = y_train_2[product].values
    y_test_2_product = y_test_2[product].values

    # Train the model
    clf.fit(X_train_2, y_train_2_product)

    # Predictions
    y_train_2_pred = clf.predict(X_train_2)
    y_test_2_pred = clf.predict(X_test_2)
    y_train_2_pred_proba = clf.predict_proba(X_train_2)[:, 1]
    y_test_2_pred_proba = clf.predict_proba(X_test_2)[:, 1]

    # Calculate metrics
    metrics['train_2']['train'][product] = {
        'ROC AUC': roc_auc_score(y_train_2_product, y_train_2_pred_proba),
        'F1 Score': f1_score(y_train_2_product, y_train_2_pred),
        'Confusion Matrix': confusion_matrix(y_train_2_product, y_train_2_pred)
    }

    metrics['train_2']['test'][product] = {
        'ROC AUC': roc_auc_score(y_test_2_product, y_test_2_pred_proba),
        'F1 Score': f1_score(y_test_2_product, y_test_2_pred),
        'Confusion Matrix': confusion_matrix(y_test_2_product, y_test_2_pred)
    }
```

In [15]:

```
# Summarize the average metrics across all products
summary_data_2 = []
for dataset in ['train', 'test']:
    avg_roc_auc = np.mean([metrics['train_2'][dataset][p]['ROC AUC'] for p in products
    avg_f1 = np.mean([metrics['train_2'][dataset][p]['F1 Score'] for p in products])
    summary_data_2.append(['train_2', dataset, avg_roc_auc, avg_f1])

# Create summary DataFrame
summary_df_2 = pd.DataFrame(summary_data_2, columns=['Dataset', 'Type', 'Avg ROC AUC',
print("\nEvaluated Model on Dataset: train_2")
print(summary_df_2.to_string(index=False))
```

```
Evaluated Model on Dataset: train_2
Dataset  Type  Avg ROC AUC  Avg F1 Score
train_2 train     0.876659      0.084174
train_2  test     0.874053      0.185602
```

Feature Importance Analysis

In [ ]:

```
# Dictionary to store feature importances
feature_importances = {}

# Iterate over each product
for product in products:
    clf = LogisticRegression(**params)
    clf.fit(X_train_2, y_train_2[product].values)

    feature_importances[product] = clf.coef_[0]

importances_df = pd.DataFrame(feature_importances, index=X_train_2.columns)
```

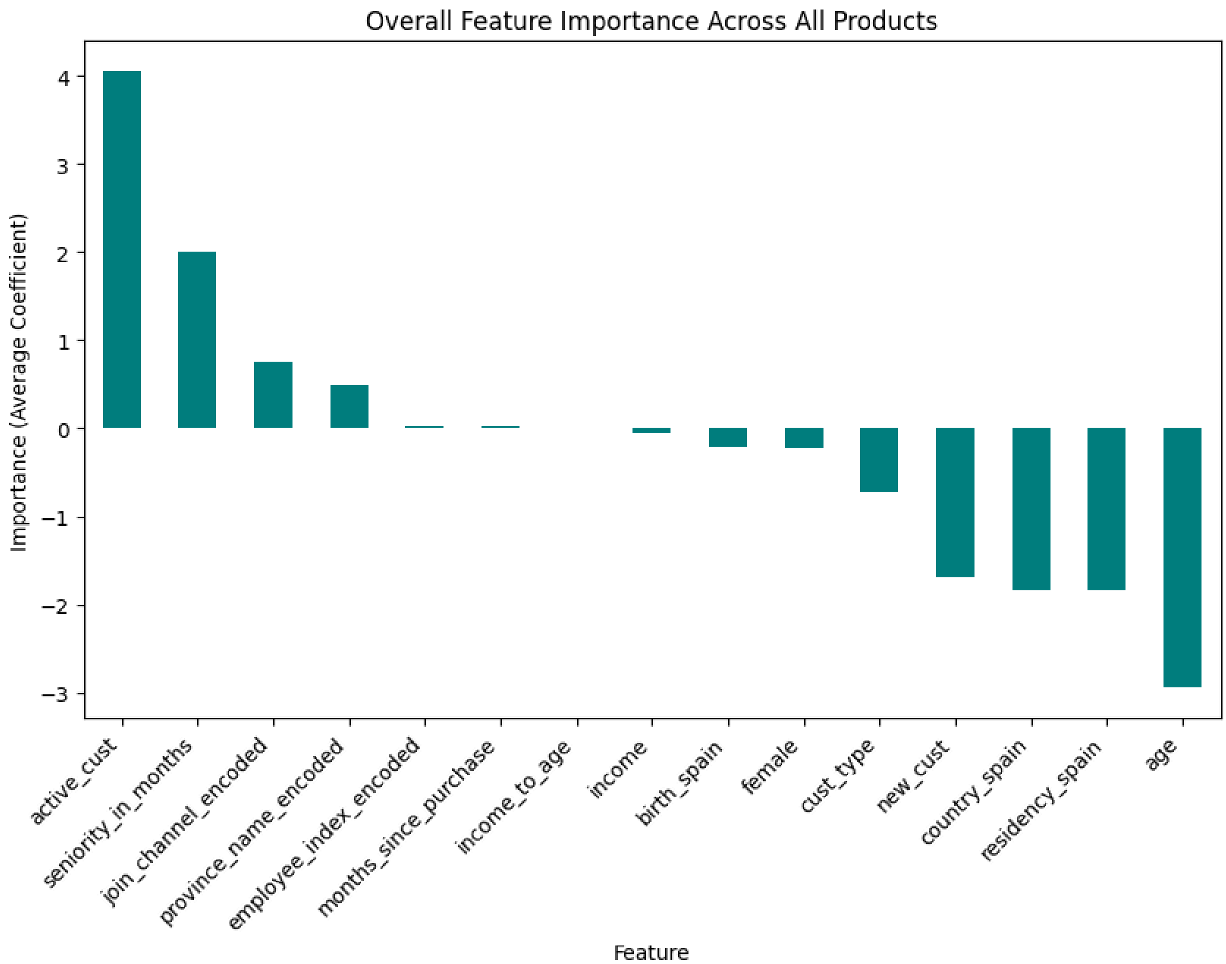
```
importances_df['Mean_Importance'] = importances_df.mean(axis=1)
sorted_importances = importances_df.sort_values(by='Mean_Importance', ascending=False)
```

In [27]:
```
sorted_importances
```

Out[27]:

	savings_acct	guarantees	current_acct	derivada_acct	payroll_acct	junior_acc
active_cust	0.830705	4.264000	1.414410	2.427595	6.842190	3.03010
seniority_in_months	4.603687	3.920184	0.148217	1.993498	0.956385	5.37899
join_channel_encoded	0.909934	-0.647531	-1.445911	0.368749	1.030798	2.54202
province_name_encoded	-0.805320	6.485177	-0.823748	-0.552544	0.626762	0.89281
employee_index_encoded	0.250569	0.528488	0.143572	0.657579	0.122628	0.48636
months_since_purchase	0.022000	0.026316	0.024692	0.010854	0.002302	0.03050
income_to_age	0.000004	-0.000041	-0.000020	0.000002	-0.000012	-0.00007
income	0.029702	0.014780	-0.000532	0.003247	-0.060334	-0.01263
birth_spain	0.615684	-3.797005	-0.192945	-0.808166	0.135772	-0.09866
female	-0.581870	-0.833165	0.029119	-1.419952	0.013041	0.00647
cust_type	-3.278677	-5.275618	0.531083	-0.524133	0.487662	-0.87524
new_cust	-2.920691	-3.626413	-0.168880	-1.249166	-0.418773	-0.66440
country_spain	-1.702266	-5.265817	0.772075	-2.432985	-3.752419	0.37180
residency_spain	-1.702266	-5.265817	0.772075	-2.432985	-3.752419	0.37180
age	-2.054414	-4.500836	-0.370727	1.399406	-2.178210	-69.70432

In [22]:
```
plt.figure(figsize=(10, 6))
sorted_importances['Mean_Importance'].plot(kind='bar', color='teal')
plt.title('Overall Feature Importance Across All Products')
plt.ylabel('Importance (Average Coefficient)')
plt.xlabel('Feature')
plt.xticks(rotation=45, ha='right')
plt.show()
```



LIME example on variable 'savings_acct'

Here we are trying to get deeper and understand which variables affect the variable 'savings_acct' the most. For the purpose of this exercise we will just run the code on 'savings_acct'

In [18]:

```
# pip install lime
```

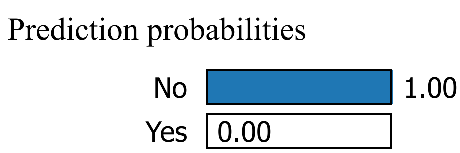
In [23]:

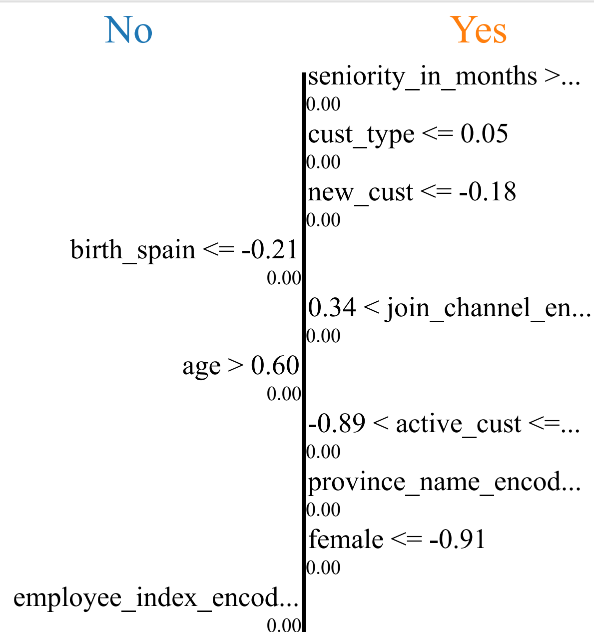
```
from lime.lime_tabular import LimeTabularExplainer
from sklearn.preprocessing import StandardScaler
```

In [ ]:

```
example_product = 'savings_acct'

#Standardizing features for LIME
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_2)
X_test_scaled = scaler.transform(X_test_2)

clf = LogisticRegression(**params)
clf.fit(X_train_scaled, y_train_2[example_product].values)

# Initializing LIME Tabular Explainer
explainer = LimeTabularExplainer(
    training_data=X_train_scaled,
    training_labels=y_train_2[example_product],
```

```
    feature_names=X_train_2.columns,
    class_names=['No', 'Yes'],
    mode='classification'
)

# Choosing an instance to explain
instance_idx = 0
instance = X_test_scaled[instance_idx]

# Generating explanation
exp = explainer.explain_instance(
    data_row=instance,
    predict_fn=clf.predict_proba  # Probability prediction function
)
exp.show_in_notebook(show_table=True)  # For Jupyter Notebook
# exp.save_to_file('lime_explanation.html')  # Save to HTML file
```





seniority_in_months	1.78
cust_type	0.05
new_cust	-0.18
birth_spain	-0.21
join_channel_encoded	1.21
age	1.95
active_cust	1.12
province_name_encoded	-0.88
female	-0.91
employee_index_encoded	-0.02

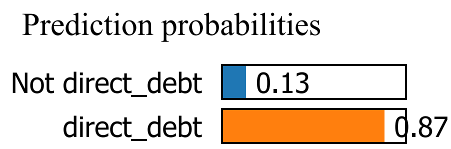
Select 5 predictions at random, explain how the model generated those predictions (which features matter more than others), which features need to change and by how much to move the output in a significant way (e.g., to flip the prediction from one class to another)

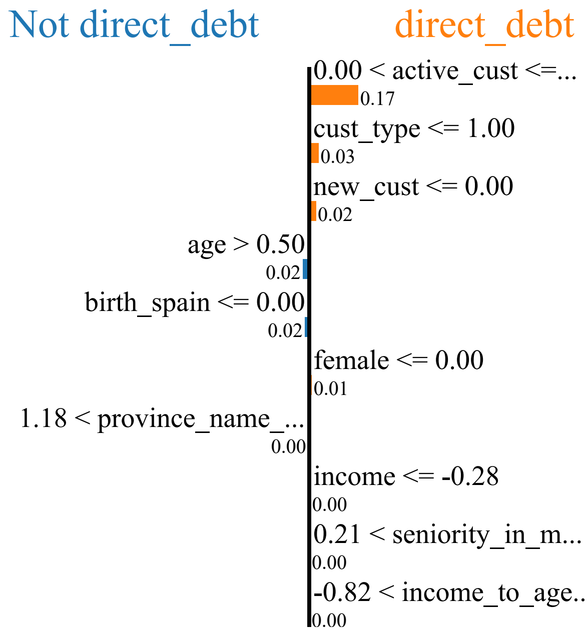
In [24]:
```
import lime
import lime.lime_tabular
import random
```

In [25]:
```
# Randomly selecting 5 samples
random_indices = random.sample(range(X_test_2.shape[0]), 5)
selected_samples = X_test_2.iloc[random_indices]
selected_labels = y_test_2[product].iloc[random_indices]

# LIME explainer
explainer = lime.lime_tabular.LimeTabularExplainer(
    X_train_2.values,
    feature_names=X_train_2.columns,
    class_names=[f"Not {product}", product],
    verbose=True,
    mode="classification"
)

# Explain each sample
for i, idx in enumerate(random_indices):
    print(f"\nExplaining Prediction {i+1} (Row {idx}):")
    sample = X_test_2.iloc[idx].values

    # Probability for the sample
    prediction = clf.predict_proba([sample])[0]
    predicted_class = np.argmax(prediction)
    print(f"Predicted Class: {predicted_class} (Probability: {prediction[predicted_cla

    # Generate explanation
    exp = explainer.explain_instance(sample, clf.predict_proba, num_features=10)
    exp.show_in_notebook(show_table=True)

    # Print explanation in text format
    explanation = exp.as_list()
    print("Feature Contributions:")
    for feature, contribution in explanation:
        print(f"{feature}: {contribution:.4f}")

    # Identify feature changes to flip prediction
    important_feature, contribution = explanation[0]
    print(f"\nTo flip the prediction, try adjusting '{important_feature}' by a signifi

# Visualize the explanation for one of the samples
exp.show_in_notebook(show_table=True)
```

```
Explaining Prediction 1 (Row 950916):
Predicted Class: 1 (Probability: 0.87)
Intercept -0.039240922130545156
Prediction_local [0.15935189]
Right: 0.8716594547837396
```

```
c:\Users\MARIA\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\bas
e.py:464: UserWarning: X does not have valid feature names, but LogisticRegression wa
s fitted with feature names
  warnings.warn(
c:\Users\MARIA\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\bas
e.py:464: UserWarning: X does not have valid feature names, but LogisticRegression wa
s fitted with feature names
  warnings.warn(
```

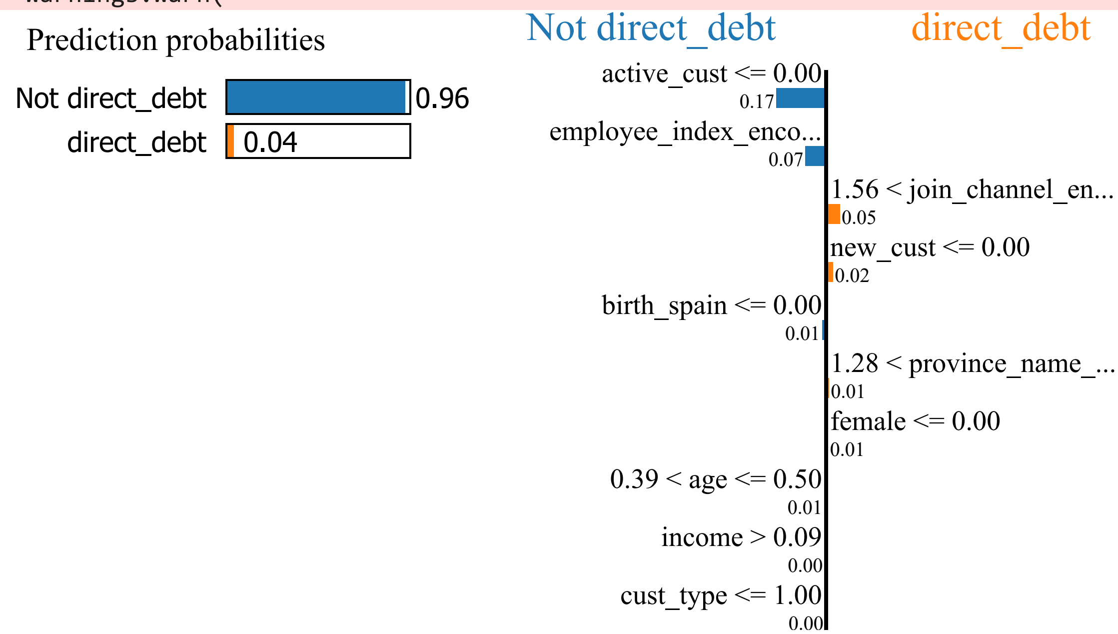




Feature	Value
active_cust	1.00
cust_type	1.00
new_cust	0.00
age	0.66
birth_spain	0.00
female	0.00
province_name_encoded	1.24
income	-0.46
seniority_in_months	0.54

```
Feature Contributions:
0.00 < active_cust <= 1.00: 0.1687
cust_type <= 1.00: 0.0320
new_cust <= 0.00: 0.0229
age > 0.50: -0.0219
birth_spain <= 0.00: -0.0151
female <= 0.00: 0.0084
1.18 < province_name_encoded <= 1.28: -0.0037
income <= -0.28: 0.0034
0.21 < seniority_in_months <= 0.54: 0.0023
-0.82 < income_to_age <= -0.35: 0.0016

To flip the prediction, try adjusting '0.00 < active_cust <= 1.00' by a significant a
mount.

Explaining Prediction 2 (Row 3239):
Predicted Class: 0 (Probability: 0.96)
Intercept 0.21402389014112455
Prediction_local [0.03316361]
Right: 0.037688068101520936
```

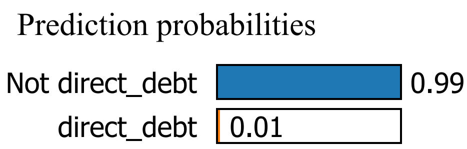
```
c:\Users\MARIA\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\bas
e.py:464: UserWarning: X does not have valid feature names, but LogisticRegression wa
s fitted with feature names
  warnings.warn(
c:\Users\MARIA\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\bas
e.py:464: UserWarning: X does not have valid feature names, but LogisticRegression wa
s fitted with feature names
  warnings.warn(
```

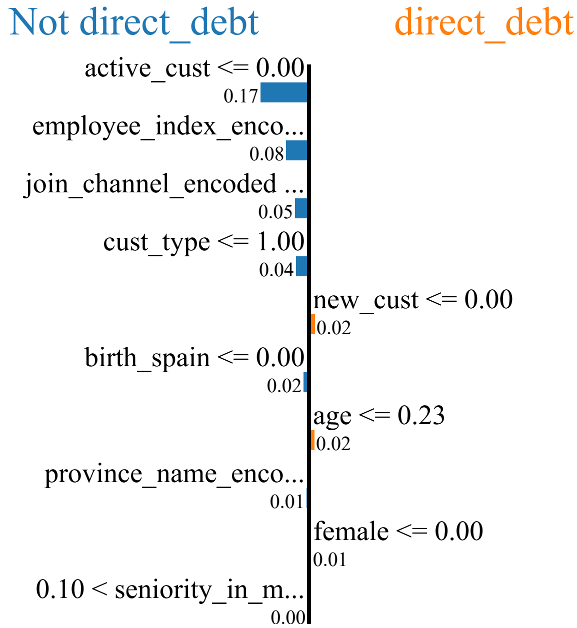


Feature	Value
active_cust	0.00
employee_index_encoded	1.41
join_channel_encoded	1.81
new_cust	0.00
birth_spain	0.00
province_name_encoded	1.75
female	0.00
age	0.50
income	0.45

```
Feature Contributions:
active_cust <= 0.00: -0.1708
employee_index_encoded <= 1.41: -0.0710
1.56 < join_channel_encoded <= 1.94: 0.0472
new_cust <= 0.00: 0.0232
birth_spain <= 0.00: -0.0136
1.28 < province_name_encoded <= 1.75: 0.0093
female <= 0.00: 0.0069
0.39 < age <= 0.50: -0.0056
income > 0.09: -0.0043
cust_type <= 1.00: -0.0021

To flip the prediction, try adjusting 'active_cust <= 0.00' by a significant amount.

Explaining Prediction 3 (Row 458596):
Predicted Class: 0 (Probability: 0.99)
Intercept 0.30015283526118897
Prediction_local [-0.02265368]
Right: 0.014329623063726801
```

```
c:\Users\MARIA\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\bas
e.py:464: UserWarning: X does not have valid feature names, but LogisticRegression wa
s fitted with feature names
  warnings.warn(
c:\Users\MARIA\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\bas
e.py:464: UserWarning: X does not have valid feature names, but LogisticRegression wa
s fitted with feature names
  warnings.warn(
```





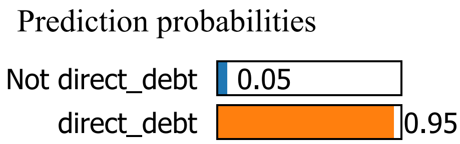
Feature	Value
active_cust	0.00
employee_index_encoded	1.41
join_channel_encoded	0.89
cust_type	1.00
new_cust	0.00
birth_spain	0.00
age	0.20
province_name_encoded	1.18
female	0.00

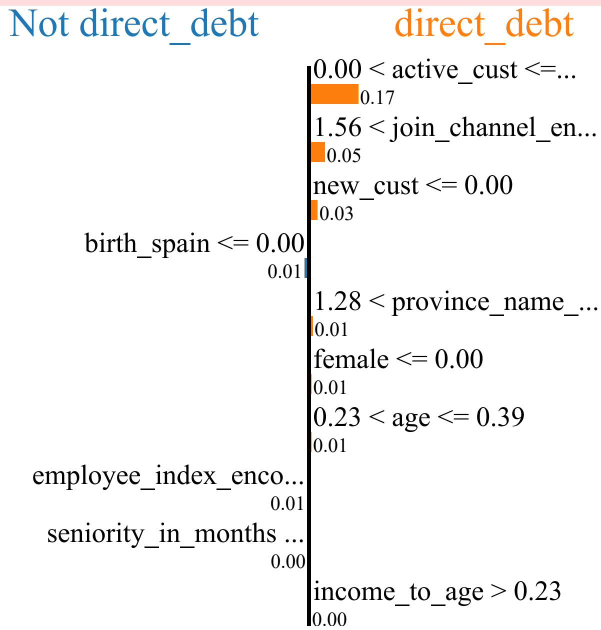
```
Feature Contributions:
active_cust <= 0.00: -0.1668
employee_index_encoded <= 1.41: -0.0780
join_channel_encoded <= 0.89: -0.0475
cust_type <= 1.00: -0.0443
new_cust <= 0.00: 0.0195
birth_spain <= 0.00: -0.0178
age <= 0.23: 0.0173
province_name_encoded <= 1.18: -0.0074
female <= 0.00: 0.0067
0.10 < seniority_in_months <= 0.21: -0.0044

To flip the prediction, try adjusting 'active_cust <= 0.00' by a significant amount.

Explaining Prediction 4 (Row 1115023):
```

```
c:\Users\MARIA\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\bas
e.py:464: UserWarning: X does not have valid feature names, but LogisticRegression wa
s fitted with feature names
  warnings.warn(
```

```
Predicted Class: 1 (Probability: 0.95)
Intercept -0.03681746983404906
Prediction_local [0.22245744]
Right: 0.9472943668943028
```

```
c:\Users\MARIA\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\bas
e.py:464: UserWarning: X does not have valid feature names, but LogisticRegression wa
s fitted with feature names
  warnings.warn(
```





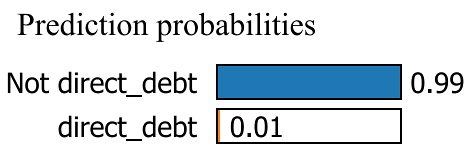
active_cust	1.00
join_channel_encoded	1.94
new_cust	0.00
birth_spain	0.00
province_name_encoded	1.75
female	0.00
age	0.36
employee_index_encoded	1.41
seniority_in_months	0.05
income_to_age	0.51

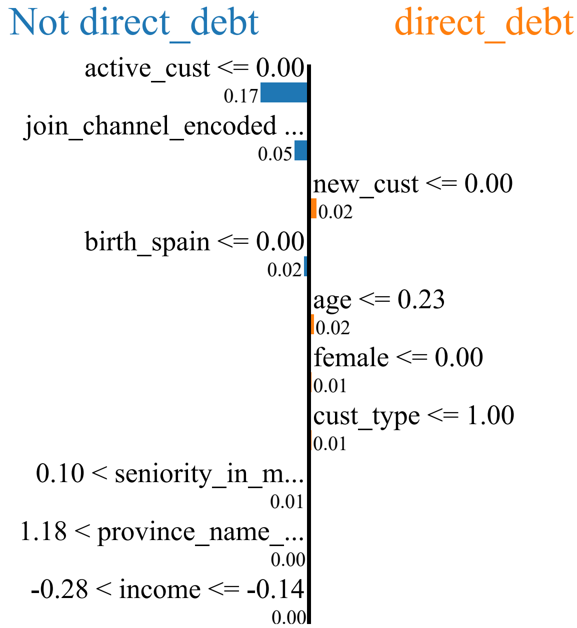
```
Feature Contributions:
0.00 < active_cust <= 1.00: 0.1695
1.56 < join_channel_encoded <= 1.94: 0.0546
new_cust <= 0.00: 0.0291
birth_spain <= 0.00: -0.0150
1.28 < province_name_encoded <= 1.75: 0.0127
female <= 0.00: 0.0079
0.23 < age <= 0.39: 0.0071
employee_index_encoded <= 1.41: -0.0055
seniority_in_months <= 0.10: -0.0045
income_to_age > 0.23: 0.0035

To flip the prediction, try adjusting '0.00 < active_cust <= 1.00' by a significant a
mount.

Explaining Prediction 5 (Row 557414):
Predicted Class: 0 (Probability: 0.99)
```

```
c:\Users\MARIA\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\bas
e.py:464: UserWarning: X does not have valid feature names, but LogisticRegression wa
s fitted with feature names
  warnings.warn(
```

```
Intercept 0.1629545436552834
Prediction_local [-0.02360619]
Right: 0.014531659456663631
```

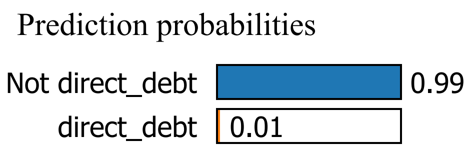
```
c:\Users\MARIA\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\bas
e.py:464: UserWarning: X does not have valid feature names, but LogisticRegression wa
s fitted with feature names
  warnings.warn(
```

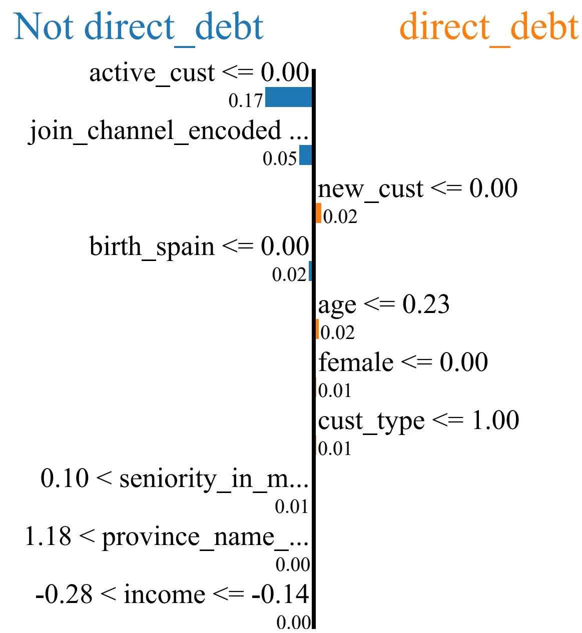




Feature	Value
active_cust	0.00
join_channel_encoded	0.89
new_cust	0.00
birth_spain	0.00
age	0.23
female	0.00
cust_type	1.00
seniority_in_months	0.21
province_name_encoded	1.28

```
Feature Contributions:
active_cust <= 0.00: -0.1665
join_channel_encoded <= 0.89: -0.0483
new_cust <= 0.00: 0.0249
birth_spain <= 0.00: -0.0164
age <= 0.23: 0.0161
female <= 0.00: 0.0079
cust_type <= 1.00: 0.0074
0.10 < seniority_in_months <= 0.21: -0.0063
1.18 < province_name_encoded <= 1.28: -0.0042
-0.28 < income <= -0.14: -0.0012

To flip the prediction, try adjusting 'active_cust <= 0.00' by a significant amount.
```





Feature	Value
active_cust	0.00
join_channel_encoded	0.89
new_cust	0.00
birth_spain	0.00
age	0.23
female	0.00
cust_type	1.00
seniority_in_months	0.21
province_name_encoded	1.28