Group 3: ML Assignment 5

Let's start coding!

```
# !pip install lime
```

```
# !pip install SALib
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OrdinalEncoder, StandardScaler
from matplotlib import pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, recall_score, precision_score, roc_curve, auc, confusion_matrix
from lime import lime_tabular
import random
from imblearn.over_sampling import SMOTE
from SALib.sample import saltelli
from SALib.analyze import sobol
from lime import lime_tabular
from sklearn.linear_model import LogisticRegression
```

Read the dataset

```
path = '/content/sample_data/Telecom.xlsx'
df = pd.read_excel(path, sheet_name='Telecom')
df.info()
df.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4888 entries, 0 to 4887
Data columns (total 20 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   CustomerID              4888 non-null   int64  
 1   PlanTaken               4888 non-null   int64  
 2   Age                     4662 non-null   float64
 3   TypeofContact           4863 non-null   object 
 4   CityTier                4888 non-null   int64  
 5   DurationOfPitch         4637 non-null   float64
 6   Occupation              4888 non-null   object 
 7   Gender                  4888 non-null   object 
 8   NumberOfPersons         4888 non-null   int64  
 9   NumberOfFollowups       4843 non-null   float64
 10  PlanPitched             4888 non-null   object 
 11  PreferredServiceStar    4862 non-null   float64
 12  MaritalStatus           4888 non-null   object 
 13  NumberOfUpgrades        4748 non-null   float64
 14  iPhone                  4888 non-null   int64  
 15  PitchSatisfactionScore  4888 non-null   int64  
 16  PhoneContract           4888 non-null   int64  
 17  NumberOfChildren        4822 non-null   float64
 18  Designation             4888 non-null   object 
 19  MonthlyIncome           4655 non-null   float64
dtypes: float64(7), int64(7), object(6)
memory usage: 763.9+ KB
```

	CustomerID	PlanTaken	Age	CityTier	DurationOfPitch	NumberOf
count	4888.000000	4888.000000	4662.000000	4888.000000	4637.000000	4888
mean	202443.500000	0.188216	37.622265	1.654255	15.490835	2
std	1411.188388	0.390925	9.316387	0.916583	8.519643	0
min	200000.000000	0.000000	18.000000	1.000000	5.000000	1
25%	201221.750000	0.000000	31.000000	1.000000	9.000000	2
50%	202443.500000	0.000000	36.000000	1.000000	13.000000	3
75%	203665.250000	0.000000	44.000000	3.000000	20.000000	3
max	204887.000000	1.000000	61.000000	3.000000	127.000000	5

Feature Engineering

Remove duplicates

```
df
df.drop_duplicates()
```

	CustomerID	PlanTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupati
0	200000	1	41.0	Self Enquiry	3	6.0	Salari
1	200001	0	49.0	Company Invited	1	14.0	Salari
2	200002	1	37.0	Self Enquiry	1	8.0	Free Lanc
3	200003	0	33.0	Company Invited	1	9.0	Salari
4	200004	0	NaN	Self Enquiry	1	8.0	Sm Busine
...	...	...	...	...	...	...	
4883	204883	1	49.0	Self Enquiry	3	9.0	Sm Busine
4884	204884	1	28.0	Company Invited	1	31.0	Salari
4885	204885	1	52.0	Self Enquiry	3	17.0	Salari
4886	204886	1	19.0	Self Enquiry	3	16.0	Sm Busine
4887	204887	1	36.0	Self Enquiry	1	14.0	Salari

4888 rows × 20 columns

Drop Occupation - Free Lancer

```
df = df.drop(df[df["Occupation"] == 'Free Lancer'].index)
```

Select features

- Select independent features as X
- Select dependent feature for y

```
def select_features_for_model(df):
    df=df.drop(columns='CustomerID', axis=1)
    X = df.drop("PlanTaken", axis=1)
    y = df["PlanTaken"]
    X.info()
    return X, y

X, y = select_features_for_model(df)
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4886 entries, 0 to 4887
Data columns (total 18 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   Age                    4660 non-null   float64
 1   TypeofContact          4861 non-null   object 
 2   CityTier               4886 non-null   int64  
 3   DurationOfPitch        4635 non-null   float64
 4   Occupation             4886 non-null   object 
 5   Gender                 4886 non-null   object 
 6   NumberOfPersons        4886 non-null   int64  
 7   NumberOfFollowups      4841 non-null   float64
 8   PlanPitched            4886 non-null   object 
 9   PreferredServiceStar   4860 non-null   float64
 10  MaritalStatus          4886 non-null   object 
 11  NumberOfUpgrades       4746 non-null   float64
 12  iPhone                 4886 non-null   int64  
 13  PitchSatisfactionScore 4886 non-null   int64  
 14  PhoneContract          4886 non-null   int64  
 15  NumberOfChildren       4820 non-null   float64
 16  Designation            4886 non-null   object 
 17  MonthlyIncome          4653 non-null   float64
dtypes: float64(7), int64(5), object(6)
memory usage: 725.3+ KB
```

Split Train & Test

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

X_train, X_test, y_train, y_test = split_train_test(X, y)
```

Data Cleaning

Gender

```
# Checking the unique values of gender in X_train
X_train['Gender'].unique()
```

```
array(['Male', 'Female', 'Fe Male'], dtype=object)
```

```
X_train['Gender'].value_counts()
```

```
Male       2334
Female     1452
Fe Male     122
Name: Gender, dtype: int64
```

```
# Correcting the Fe Male gender as Female
def clean_Gender(df):
    df['Gender'] = df['Gender'].map({'Male': 'Male', 'Female':'Female', 'Fe Male': 'Female'})
    return df

# Clean Train
X_train = clean_Gender(X_train)

# Clean Test
X_test = clean_Gender(X_test)
```

```
X_train['Gender'].value_counts()
```

```
Male      2334
Female    1574
Name: Gender, dtype: int64
```

MaritalStatus

```
X_train['MaritalStatus'].value_counts()
```

```
Married      1895
Divorced      751
Single        717
Unmarried     545
Name: MaritalStatus, dtype: int64
```

```
# Merging Unmarried to Single
def clean_MaritalStatus(df):
    df['MaritalStatus'] = df['MaritalStatus'].map({'Married': 'Married',
                                                   'Divorced':'Divorced',
                                                   'Single': 'Single',
                                                   'Unmarried': 'Single'})
    return df

# Clean Train
X_train = clean_MaritalStatus(X_train)

# Clean Test
X_test = clean_MaritalStatus(X_test)
```

```
X_train['MaritalStatus'].value_counts()
```

```
Married     1895
Single      1262
Divorced     751
Name: MaritalStatus, dtype: int64
```

Impute missing values

```
X_train.isnull().sum()
```

```
Age                       176
TypeofContact              19
CityTier                    0
DurationOfPitch           211
Occupation                  0
Gender                      0
NumberOfPersons             0
NumberOfFollowups          37
PlanPitched                 0
PreferredServiceStar       20
MaritalStatus               0
NumberOfUpgrades          122
iPhone                      0
PitchSatisfactionScore      0
PhoneContract               0
NumberOfChildren           54
Designation                 0
MonthlyIncome             182
dtype: int64
```

```
def impute_features(df):
    numeric_cols = ['DurationOfPitch', 'MonthlyIncome', 'Age', 'NumberOfUpgrades', 'NumberOfChildren', 'NumberOfFollowups', 'PreferredS
    mean_values = df[numeric_cols].mean()

    df[numeric_cols] = df[numeric_cols].fillna(mean_values)

    categorical_cols = ['TypeofContact']
    df[categorical_cols] = df[categorical_cols].fillna(df[categorical_cols].mode().iloc[0])


    return df
```

```
X_train = impute_features(X_train)
```

```
X_train.isnull().sum()
```

```
X_test = impute_features(X_test)
```

Handling Outliers

- Income can be dependent on Occupation and Designation, therefore removing the outliers based on them

```
def handle_outliers(df):

    numeric_columns = df.select_dtypes(include=['float64', 'int64']).columns

    # Calculateing IQR
    Q1 = df[numeric_columns].quantile(0.25)
    Q3 = df[numeric_columns].quantile(0.75)
    IQR = Q3 - Q1

    # Outlier threshold - 1.5 times IQR
    threshold = 1.5

    # Finding the outliers using the threshold value
    outliers = np.logical_or(df[numeric_columns] < (Q1 - threshold * IQR), df[numeric_columns] > (Q3 + threshold * IQR))

    # Total outliers in each numerical columns
    outliers_count = outliers.sum(axis=0)

    columns_to_replace_with_median = ['DurationOfPitch', 'NumberOfUpgrades']
    df[columns_to_replace_with_median] = np.where(outliers[columns_to_replace_with_median], df[columns_to_replace_with_median].median()

    # Set income thresholds based on the 95th percentile of MonthlyIncome for each designation
    thresholds_by_designation = df.groupby('Designation')['MonthlyIncome'].quantile(0.95)

    # Replace outliers based on IQR for each designation and occupation
    for (designation, occupation), group in df.groupby(['Designation', 'Occupation']):
        # Use the threshold corresponding to the designation
        income_threshold = thresholds_by_designation.get(designation, 0)

        Q1 = group['MonthlyIncome'].quantile(0.25)
        Q3 = group['MonthlyIncome'].quantile(0.75)
        IQR = Q3 - Q1

        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR

        # Replace outliers with the threshold value
        df.loc[(df['Designation'] == designation) & (df['Occupation'] == occupation) &
               ((df['MonthlyIncome'] < lower_bound) | (df['MonthlyIncome'] > upper_bound)),
               'MonthlyIncome'] = income_threshold
    return df

X_train = handle_outliers(X_train)
```

Encoding

PlanPitched - Ordinal encoding

```
X_train['PlanPitched'].unique()
```

```
array(['Basic', 'Deluxe', 'Super Deluxe', 'Standard', 'King'],
      dtype=object)
```

```
def encode_PlanPitched(df):
    # Define the custom order
    custom_order_PlanPitched = ['Basic', 'Standard', 'Deluxe', 'Super Deluxe', 'King']

    # Initialize the OrdinalEncoder with the custom order
    encoder = OrdinalEncoder(categories=[custom_order_PlanPitched])

    # Fit and transform the labels
    df['PlanPitched'] = encoder.fit_transform(df['PlanPitched'].values.reshape(-1,1))

    return df

# Encoding on Train
X_train = encode_PlanPitched(X_train)

# Encoding on Test
X_test = encode_PlanPitched(X_test)
```

Encode other categorical features

```
def encode_features(df):
    columns_to_encode = ['Gender', 'TypeofContact', 'Occupation', 'MaritalStatus', 'Designation']
    df = pd.get_dummies(df, columns=columns_to_encode)
    return df

# Encoding on Train
X_train = encode_features(X_train)

# Encoding on Test
X_test = encode_features(X_test)


X_train_columns = X_train.columns
X_test_columns = X_test.columns
```

Scale data

```
def scale_data(df):
    scaler = StandardScaler()
    df = scaler.fit_transform(df)
    return df

X_train = scale_data(X_train)
X_test = scale_data(X_test)
```

Helpers

Helper to display model metrics and AUC under ROC curve

```
def show_auc_under_roc(y_test, y_pred, y_prob):

    # Compute the false positive rate, true positive rate, and thresholds
    fpr, tpr, thresholds = roc_curve(y_test, y_prob)

    accuracy = accuracy_score(y_test, y_pred)
    auc_score = auc(fpr, tpr)

    print("Accuracy: ", accuracy)
    print("Precision:", precision_score(y_test, y_pred, average="weighted"))
    print("Recall:", recall_score(y_test, y_pred, average="weighted"))

    plt.figure()
    plt.plot(fpr, tpr, label='ROC curve (AUC = %0.2f)' % auc_score)
    plt.plot([0, 1], [0, 1], 'k--')
    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()
```

Helper to display confusion matrix

```
def show_confusion_matrix(y_test, y_pred):
    cm = confusion_matrix(y_test, y_pred)

    # Create a heatmap of the confusion matrix
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")

    # Add labels, title, and ticks to the plot
    plt.xlabel("Predicted")
    plt.ylabel("True")
    plt.title("Confusion Matrix for PlanTaken")
    plt.xticks(ticks=[0, 1])
    plt.yticks(ticks=[0, 1])

    # Show the plot
    plt.show()
```

SVM with Regulariztion & `rbf` kernel & SMOTE

```
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

svm_model = SVC(kernel='rbf', C=13.0, probability=True)
svm_model.fit(X_train_resampled, y_train_resampled)

# Make predictions on the test set
y_pred = svm_model.predict(X_test)
y_prob = svm_model.predict_proba(X_test)[:, 1]

#show_auc_under_roc(y_test, y_pred, y_prob)
#show_confusion_matrix(y_test, y_pred)
```

LIME initialize

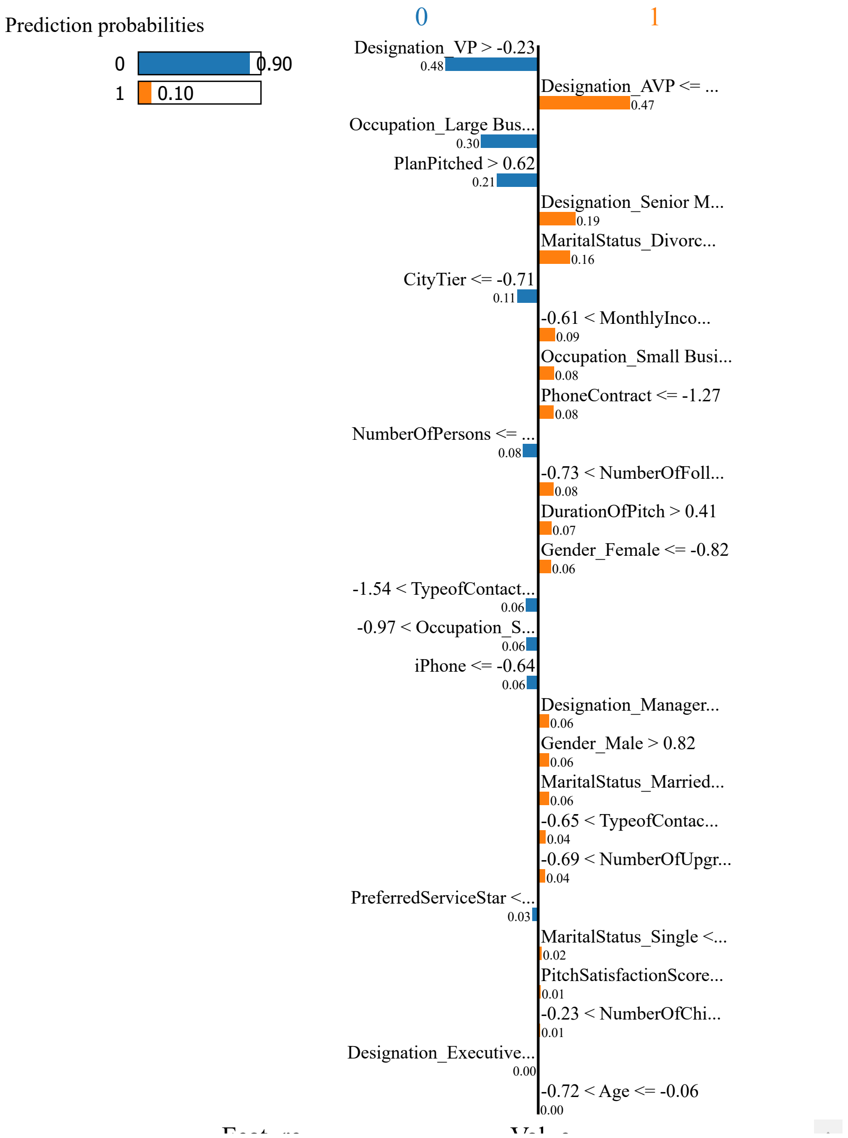
```
explainer = lime_tabular.LimeTabularExplainer(X_train, mode="classification",
                                              class_names=['0','1'],
                                              feature_names=X_train_columns,
                                             )
```

```
y_test=np.array(y_test)
idx = random.randint(1, len(X_test))

predicted_label = svm_model.predict(X_test[idx].reshape(1, -1))[0]

print("Prediction : ", predicted_label)
print("Actual : ",y_test[idx] )
```

```
Prediction :  0
Actual :  0
```

Visualizing Feature Importance

```
explanation = explainer.explain_instance(X_test[idx], svm_model.predict_proba,
                                         num_features=X_train.shape[1])

explanation.show_in_notebook()
```



In case of complex models(Black-Box) that have lots of features interpreting the model becomes hard. In these circumstances we can use LIME local interpretability to analyse which features are affecting predicted outcome the most. We can understand that:

1. Model has high confidence in predicting that in this instance the class is 0 based on the probability value of 1.
2. Features that push the prediction towards class 0 are shown in blue, while those pushing towards class 1 are in orange.
3. 'Designation_VP' and 'Occupation_Large Business' are the most significant features that contribute to the prediction of class 0 having the values -0.20 and -0.30 respectively.
4. Features such as 'DurationOfPitch' and 'MonthlyIncome' do not have much contribution.

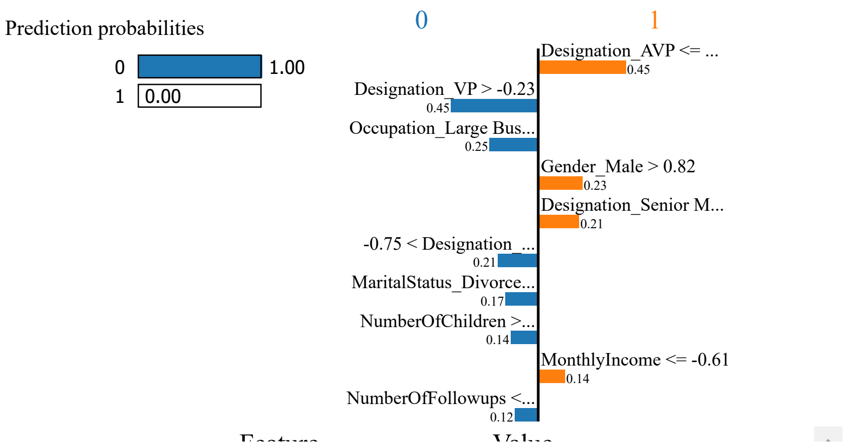
Visualizing Features Importance for Wrong Predictions

```
y_test=np.array(y_test)
preds = svm_model.predict(X_test)

false_preds = np.argwhere((preds != y_test)).flatten()

idx  = random.choice(false_preds)

predicted_label = svm_model.predict(X_test[idx].reshape(1, -1))[0]

print("Prediction : ", predicted_label)
print("Actual :     ",y_test[idx] )
```

```
Prediction :  0
Actual :      1
```

```
explanation = explainer.explain_instance(X_test[idx], svm_model.predict_proba)
```

```
explanation.show_in_notebook()
```



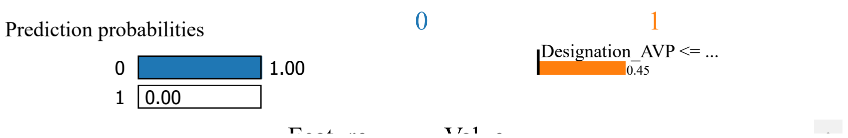
1. The bar chart at the top left corner shows the prediction probabilities for two classes (0 and 1). The model predicts class 1 with a probability of 0.59 and class 0 with a probability of 0.41, indicating the model does not have a lot of confidence in its prediction.
2. The features that have the most significant impact on the prediction are 'Designation_Manager' with a value of 1.40 and 'MaritalStatus_Married' with a value of 1.09, both pushing towards class 1. In contrast, 'Occupation_Large Business' with a value of -0.30 and 'Designation_VP' with a value of -0.20 are pushing towards class 0.

Local Feature importance interpreted for 1 instance

```
explanation = explainer.explain_instance(X_test[idx], svm_model.predict_proba,
                                         num_features=1)
```

If we need to check which feature affects outcome the most we can pass number of feature as 1 to n or all.

```
explanation.show_in_notebook()
```

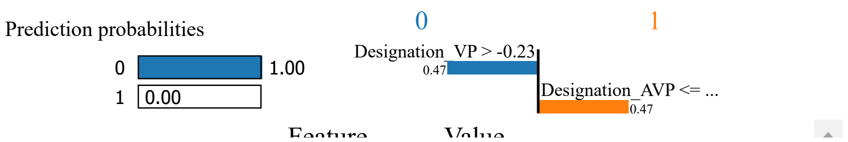


1. LIME with 1 feature provides us the feature which is affecting outcome the most.
2. We can see that 'Designation_AVP' contributes to 1 outcome the most in this instance.

Local Feature importance interpreted for 2 instances

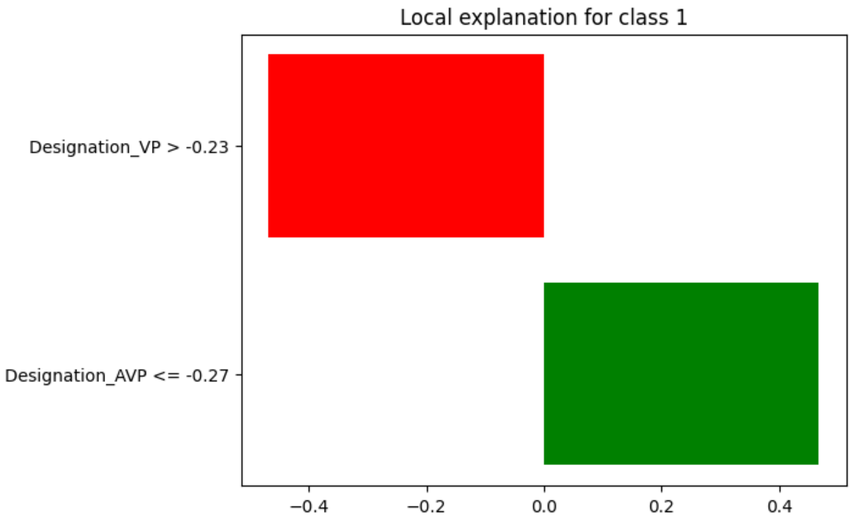
```
explanation = explainer.explain_instance(X_test[idx], svm_model.predict_proba,
                                         num_features=2)
```

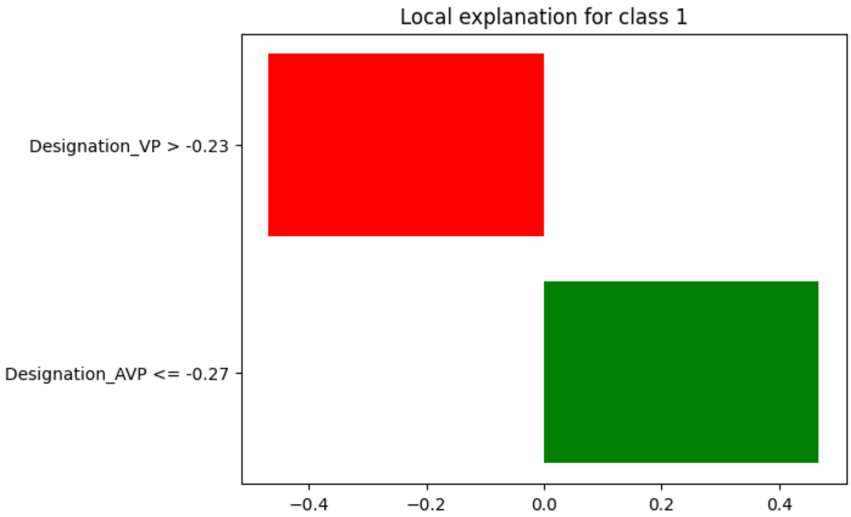
```
explanation.show_in_notebook(show_table = True)
```



1. Taking 2 features of LIME (Local interpretation) provides the 2 best features which affect the predicted outcome the most.
2. Designation_VP and Designation_AVP are the features that affect the outcome the most.
3. Feature Designation_VP affect probability of 0 outcome
4. Feature Designation_AVP affect probability of 1 outcome
5. From pyplot figure we will get better understanding

```
explanation.as_pyplot_figure()
```





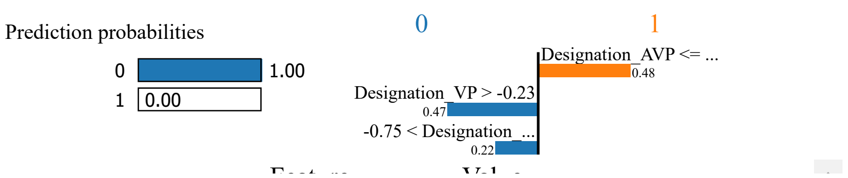
The length of the bars represents the magnitude of the contribution of each feature to the model's prediction. A longer bar means a stronger influence. The direction of the bars (left for negative, right for positive) indicates whether the feature makes the prediction of class 1 more or less likely. Both charts are almost identical.

1. Designation_AVP has a negative weight of -0.27, suggesting that the value of this feature for the instance being explained is less than or equal to -0.27, which is contributing negatively to the prediction of class 1.
2. Designation_VP has a positive weight, indicating that the value of this feature is greater than -0.23, contributing positively to the prediction of class 1.

Local Feature importance interpreted for 3 instances

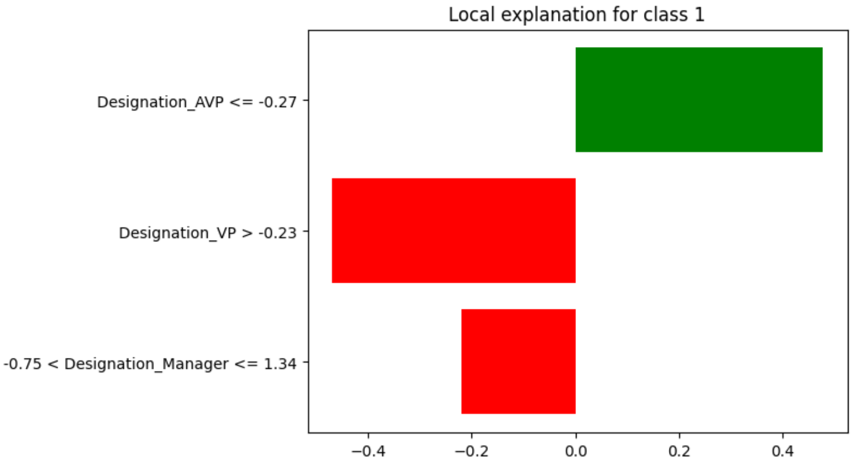
```
explanation = explainer.explain_instance(X_test[idx], svm_model.predict_proba,
                                         num_features=3)
```

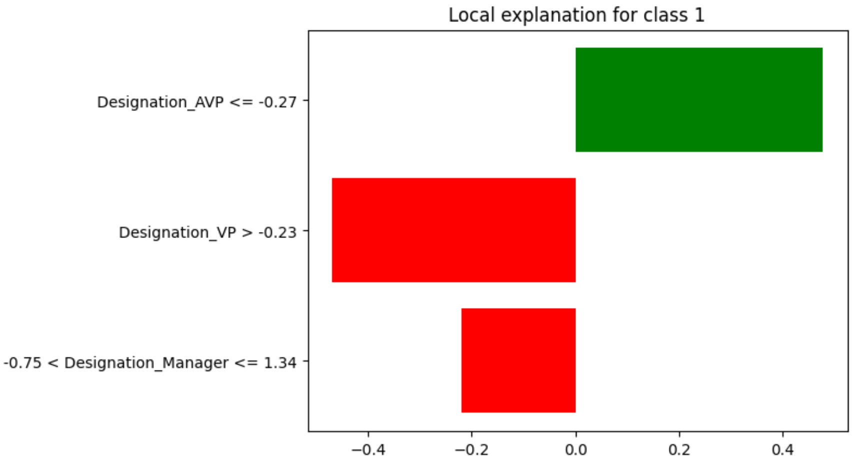
```
explanation.show_in_notebook(show_table = True)
```



1. Feature Designation_AVP contribute to probability of 1
2. Designation_VP and Designation_Manager Contribute to probability of 0
3. For three features interpretation, from the statements above we can determine that class 0 outcome has max weightage on predicated outcome compared to class 0.
4. To understand it better, we will use pyplot graph

```
explanation.as_pyplot_figure()
```



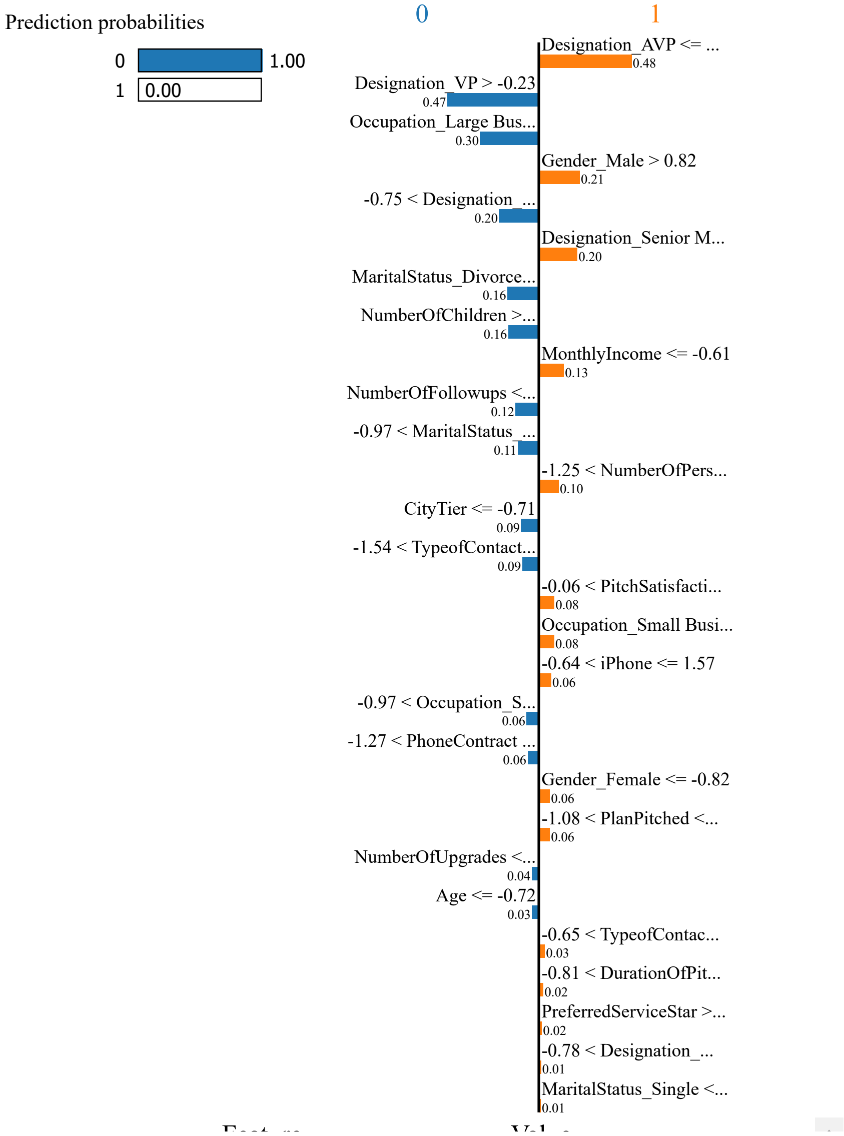


1. Designation_VP has a positive weight towards class 1. This plot suggests that when the 'Designation_VP' feature is greater than -0.23, it positively influences the prediction towards class 1.
2. In contrast Designation_AVP has a negative weight towards class 1. The condition "Designation_AVP <= -0.27" indicates that when 'Designation_AVP' is less than or equal to -0.27, it negatively influences the prediction towards class 1.
3. Similar to 'Designation_VP', 'Occupation_Large Business' also has a positive weight towards class 1, with the plot indicating its influence when "Occupation_Large Business > -0.32".

Local Feature importance interpreted for all instances

```
explanation = explainer.explain_instance(X_test[idx], svm_model.predict_proba,
                                         num_features=X_train.shape[1])
```

```
explanation.show_in_notebook(show_table = True)
```

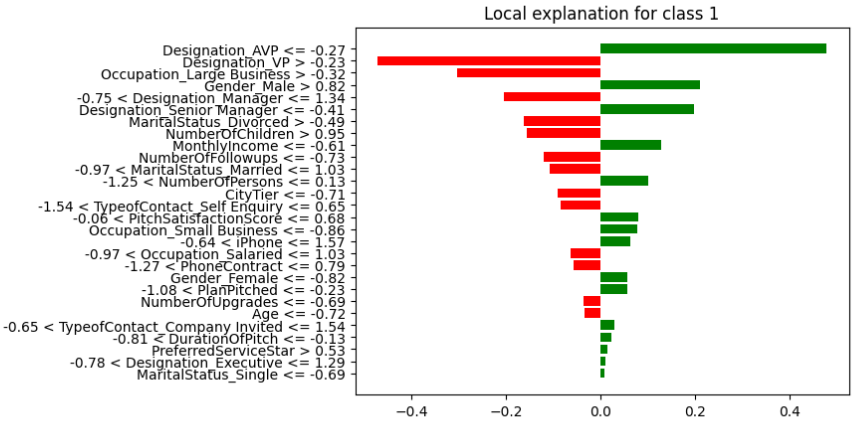


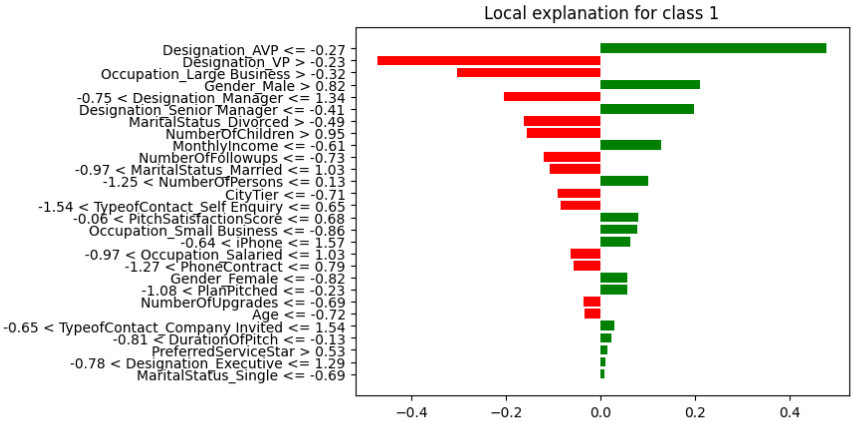
1. Here we have seperated which features contribute to probability of 0 or 1.

Features	Probability
Designation_AVP	
Gender_Male	
MaritalStatus_Divorced	
CityTier	
PitchSatisfactionScore	
NumberOfFollowups	
MonthlyIncome	1
Occupation_Small Business	
Gender_Female	
PreferredServiceStar	
iPhone	
TypeofContact_Company Invited	
DurationOfPitch	
Designation_VP	
Occupation_Large Business	
Designation_Manager	
Designation_Senior Manager	
MaritalStatus_Married	
NumberOfPersons	
PhoneContract	
TypeofContact_Self Enquiry	0
Occupation_Salaried	
NumberOfUpgrades	
MaritalStatus_Single	
Designation_Executive	
Age	
PlanPitched	
NumberOfChildren	

1. From above comparison we understand that more features contribute to 0 probability.
2. We can deduct that probability of class 0 features have more weightage on predicated outcome than class 1.

```
explanation.as_pyplot_figure()
```





Shap initialize

```
!pip install numpy==1.24
!pip install --upgrade numba
!pip install shap
import shap
```

```
Collecting numpy==1.24
  Downloading numpy-1.24.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 17.3/17.3 MB 13.5 MB/s eta 0:00:00
Installing collected packages: numpy
  Attempting uninstall: numpy
    Found existing installation: numpy 1.23.5
    Uninstalling numpy-1.23.5:
      Successfully uninstalled numpy-1.23.5
ERROR: pip's dependency resolver does not currently take into account all the package
lida 0.0.10 requires fastapi, which is not installed.
lida 0.0.10 requires kaleido, which is not installed.
lida 0.0.10 requires python-multipart, which is not installed.
lida 0.0.10 requires uvicorn, which is not installed.
seaborn 0.12.2 requires numpy!=1.24.0,>=1.17, but you have numpy 1.24.0 which is inc
Successfully installed numpy-1.24.0
WARNING: The following packages were previously imported in this runtime:
  [numpy]
You must restart the runtime in order to use newly installed versions.
```

RESTART SESSION

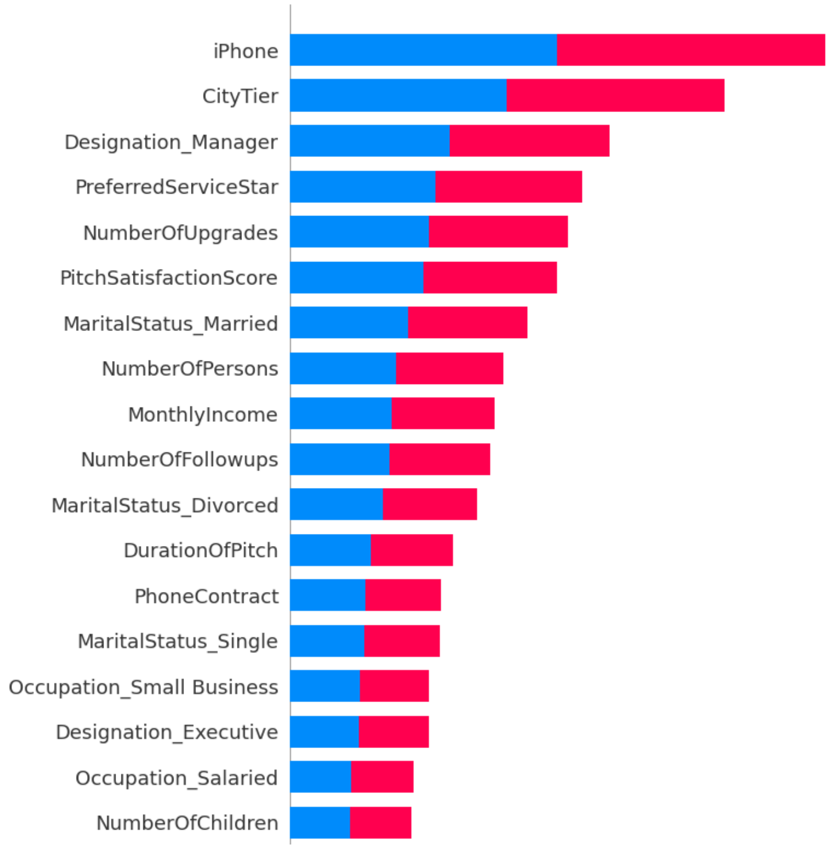
```
Requirement already satisfied: numba in /usr/local/lib/python3.10/dist-packages (0.58
Requirement already satisfied: llvmlite<0.42,>=0.41.0dev0 in /usr/local/lib/python3.1
Requirement already satisfied: numpy<1.27,>=1.22 in /usr/local/lib/python3.10/dist-pa
Collecting shap
  Downloading shap-0.44.0-cp310-cp310-manylinux_2_12_x86_64.manylinux2010_x86_64.many
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 533.5/533.5 kB 4.8 MB/s eta 0:00:00
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (fro
Requirement already satisfied: tqdm>=4.27.0 in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: packaging>20.9 in /usr/local/lib/python3.10/dist-packa
Collecting slicer==0.0.7 (from shap)
  Downloading slicer-0.0.7-py3-none-any.whl (14 kB)
Requirement already satisfied: numba in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: cloudpickle in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: llvmlite<0.42,>=0.41.0dev0 in /usr/local/lib/python3.1
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10/di
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (
Installing collected packages: slicer, shap
Successfully installed shap-0.44.0 slicer-0.0.7
```

```
# Taking 50 samples as shap is taking too much time to run
sample = 50
b = shap.sample(X_train, sample)

# Creating a KernelExplainer using the SVM model's predict_proba function
explainer = shap.KernelExplainer(svm_model.predict_proba, b,feature_names=X_train.shape[1])

# Computing shapley values for a subset
shap_values = explainer.shap_values(X_test[:sample])
```

100% 50/50 [13:47<00:00, 15.54s/it]

Global Feature Importance

```
# Summary plot for global feature importance
shap.summary_plot(shap_values, X_test,feature_names=X_train_columns)
```



This Bar chart represents the mean SHAP values for different features in a dataset, with two classes indicated by blue and red bars. The SHAP values indicate the impact of each feature on the model's output. The features are listed on the y-axis of this graph, and their corresponding