

```
In [1]: import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
sns.set_theme(color_codes=True)
pd.set_option('display.max_columns', None)
```

```
In [2]: df = pd.read_csv('df_Clean.csv')
df.head()
```

Out[2]:

Issue	AC_1003_Issue	TV_2001_Issue	TV_2002_Issue	TV_2003_Issue	Claim_Value	Service_Centre	Product_Age	Purchased_from	Call_details	Pur
0	0	1	2	0	15000.0	10	60	Manufacturer	0.5	Com
1	0	0	0	0	20000.0	12	10	Dealer	1.0	Com
1	2	0	0	0	18000.0	14	10	Dealer	1.4	(
0	0	1	1	0	12000.0	16	20	Manufacturer	2.0	Com
0	0	0	1	2	25000.0	15	6	Dealer	1.3	(

Data Preprocessing Part 1

```
In [3]: #Check the number of unique value from all of the object datatype
df.select_dtypes(include='object').nunique()
```

```
Out[3]: Region      8
State      20
Area       2
City      27
Consumer_profile  2
Product_category  2
Product_type    2
Purchased_from   3
Purpose        3
dtype: int64
```

```
In [4]: # Drop Unnamed : 0 column because this is identifier column
# Drop State and City column because there are alot of unique value. We can use Region column instead City and State
df.drop(columns=['Unnamed: 0', 'State', 'City'], inplace=True)
df.head()
```

Out[4]:

Issue	AC_1003_Issue	TV_2001_Issue	TV_2002_Issue	TV_2003_Issue	Claim_Value	Service_Centre	Product_Age	Purchased_from	Call_details	Pur
0	0	1	2	0	15000.0	10	60	Manufacturer	0.5	Com
1	0	0	0	0	20000.0	12	10	Dealer	1.0	Com
1	2	0	0	0	18000.0	14	10	Dealer	1.4	(
0	0	1	1	0	12000.0	16	20	Manufacturer	2.0	Com
0	0	0	1	2	25000.0	15	6	Dealer	1.3	(

```
In [5]: #Check the number of unique value from all of the object datatype
df.select_dtypes(include='object').nunique()
```

```
Out[5]: Region      8
Area       2
Consumer_profile  2
Product_category  2
Product_type    2
Purchased_from   3
Purpose        3
dtype: int64
```

Exploratory Data Analysis

```
In [7]: # list of categorical variables to plot
cat_vars = ['Region', 'Area', 'Consumer_profile',
            'Product_category', 'Product_type', 'Purchased_from',
            'Purpose']

# create figure with subplots
fig, axs = plt.subplots(nrows=3, ncols=3, figsize=(15, 15))
axs = axs.flatten()

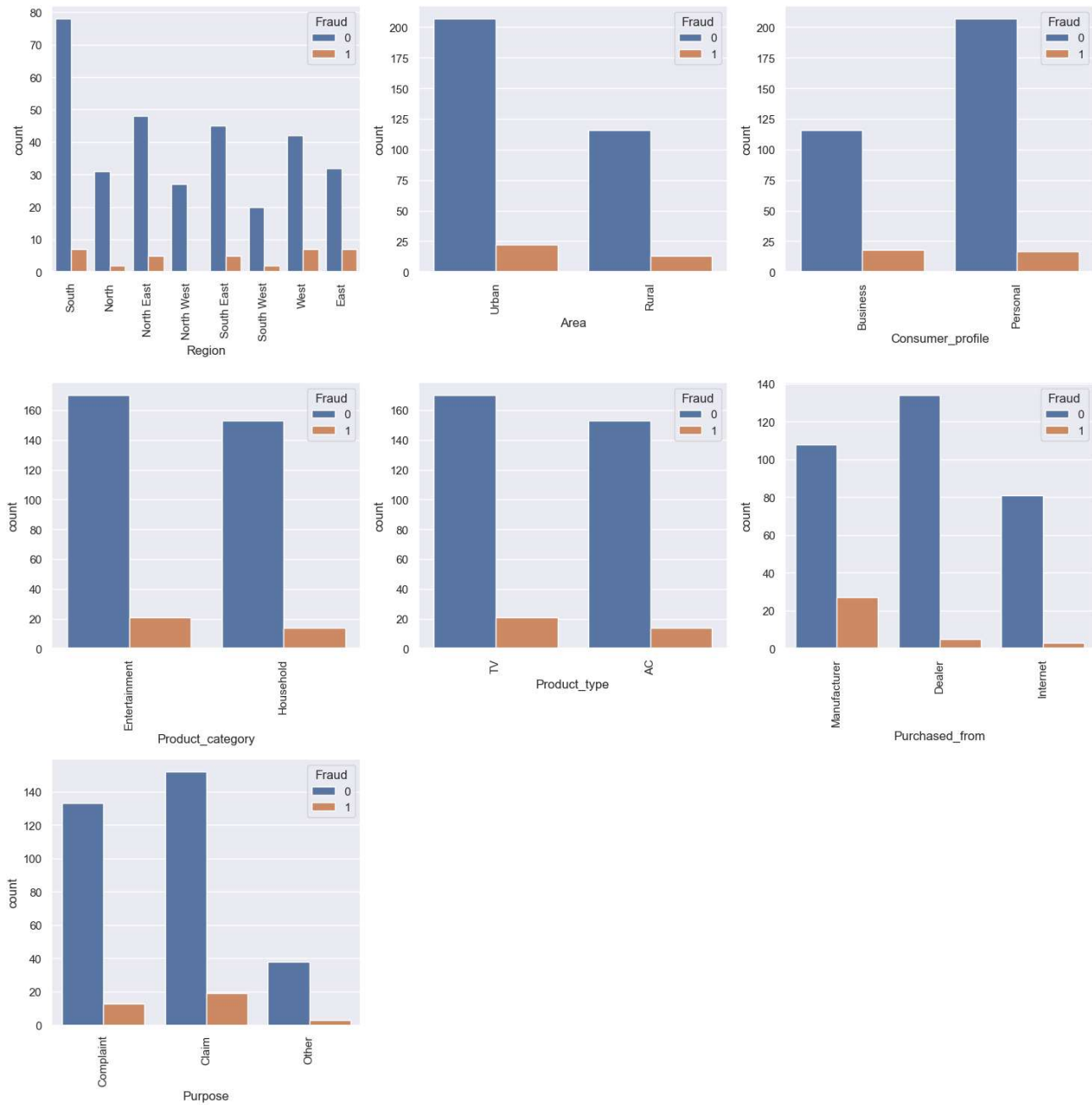
# create barplot for each categorical variable
for i, var in enumerate(cat_vars):
    sns.countplot(x=var, hue='Fraud', data=df, ax=axs[i])
    axs[i].set_xticklabels(axs[i].get_xticklabels(), rotation=90)

# adjust spacing between subplots
fig.tight_layout()

# remove the eighth subplot
fig.delaxes(axs[7])

# remove the ninth subplot
fig.delaxes(axs[8])

# show plot
plt.show()
```



```
In [8]: import warnings
warnings.filterwarnings("ignore")
# get list of categorical variables
cat_vars = ['Region', 'Area', 'Consumer_profile',
            'Product_category', 'Product_type', 'Purchased_from',
            'Purpose']

# create figure with subplots
fig, axs = plt.subplots(nrows=3, ncols=3, figsize=(15, 15))
axs = axs.flatten()

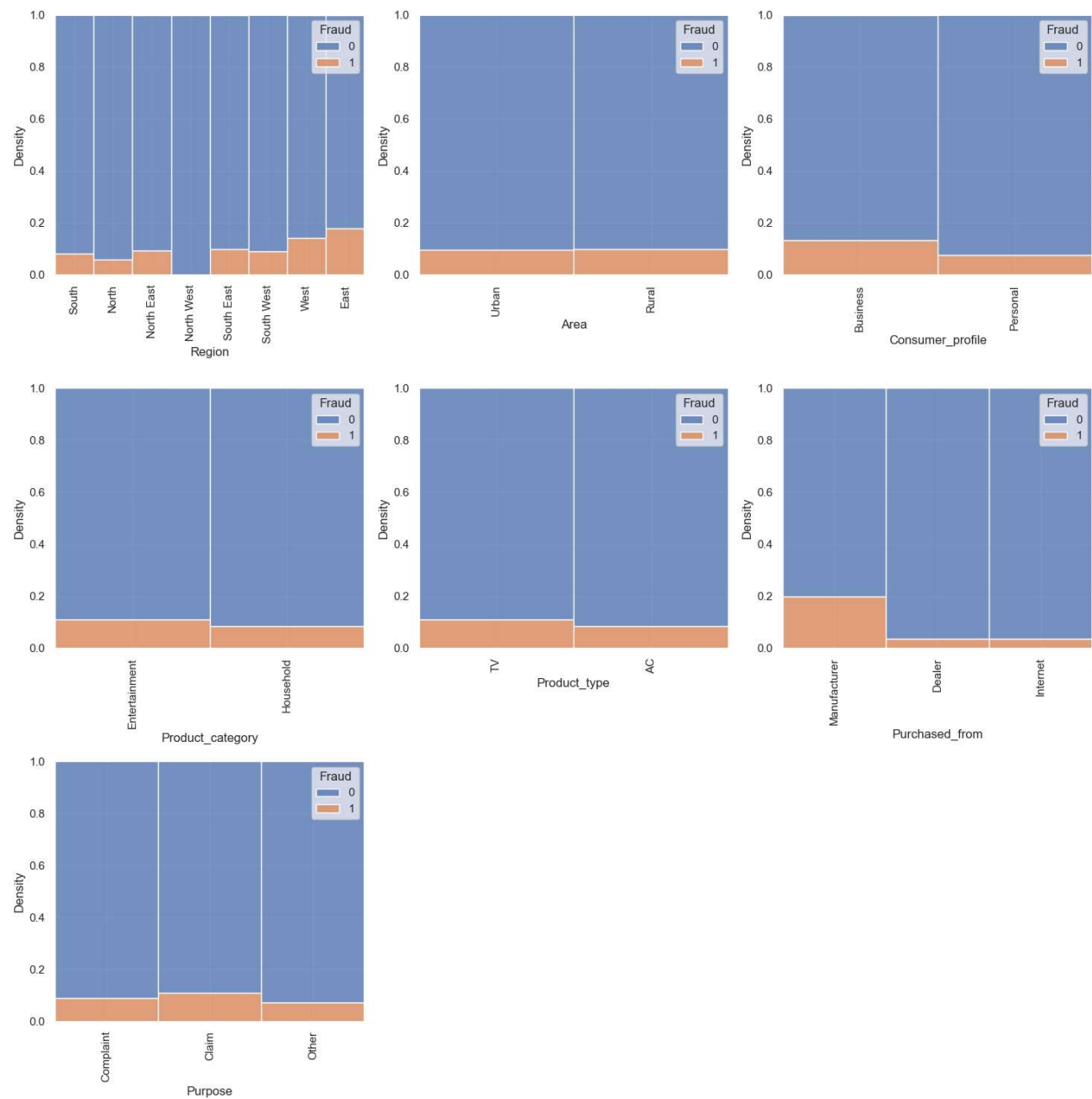
# create histplot for each categorical variable
for i, var in enumerate(cat_vars):
    sns.histplot(x=var, hue='Fraud', data=df, ax=axs[i], multiple="fill", kde=False, element="bars", fill=True, stat='count')
    axs[i].set_xticklabels(df[var].unique(), rotation=90)
    axs[i].set_xlabel(var)

# adjust spacing between subplots
fig.tight_layout()

# remove the eighth subplot
fig.delaxes(axs[7])

# remove the ninth subplot
fig.delaxes(axs[8])

# show plot
plt.show()
```



```
In [9]: # Specify the maximum number of categories to show individually
max_categories = 5

cat_vars = ['Region', 'Area', 'Consumer_profile',
            'Product_category', 'Product_type', 'Purchased_from',
            'Purpose']

# Create a figure and axes
fig, axs = plt.subplots(nrows=3, ncols=3, figsize=(15, 15))

# Create a pie chart for each categorical variable
for i, var in enumerate(cat_vars):
    if i < len(axs.flat):
        # Count the number of occurrences for each category
        cat_counts = df[var].value_counts()

        # Group categories beyond the top max_categories as 'Other'
        if len(cat_counts) > max_categories:
            cat_counts_top = cat_counts[:max_categories]
            cat_counts_other = pd.Series(cat_counts[max_categories:].sum(), index=['Other'])
            cat_counts = cat_counts_top.append(cat_counts_other)

        # Create a pie chart
        axs.flat[i].pie(cat_counts, labels=cat_counts.index, autopct='%1.1f%%', startangle=90)

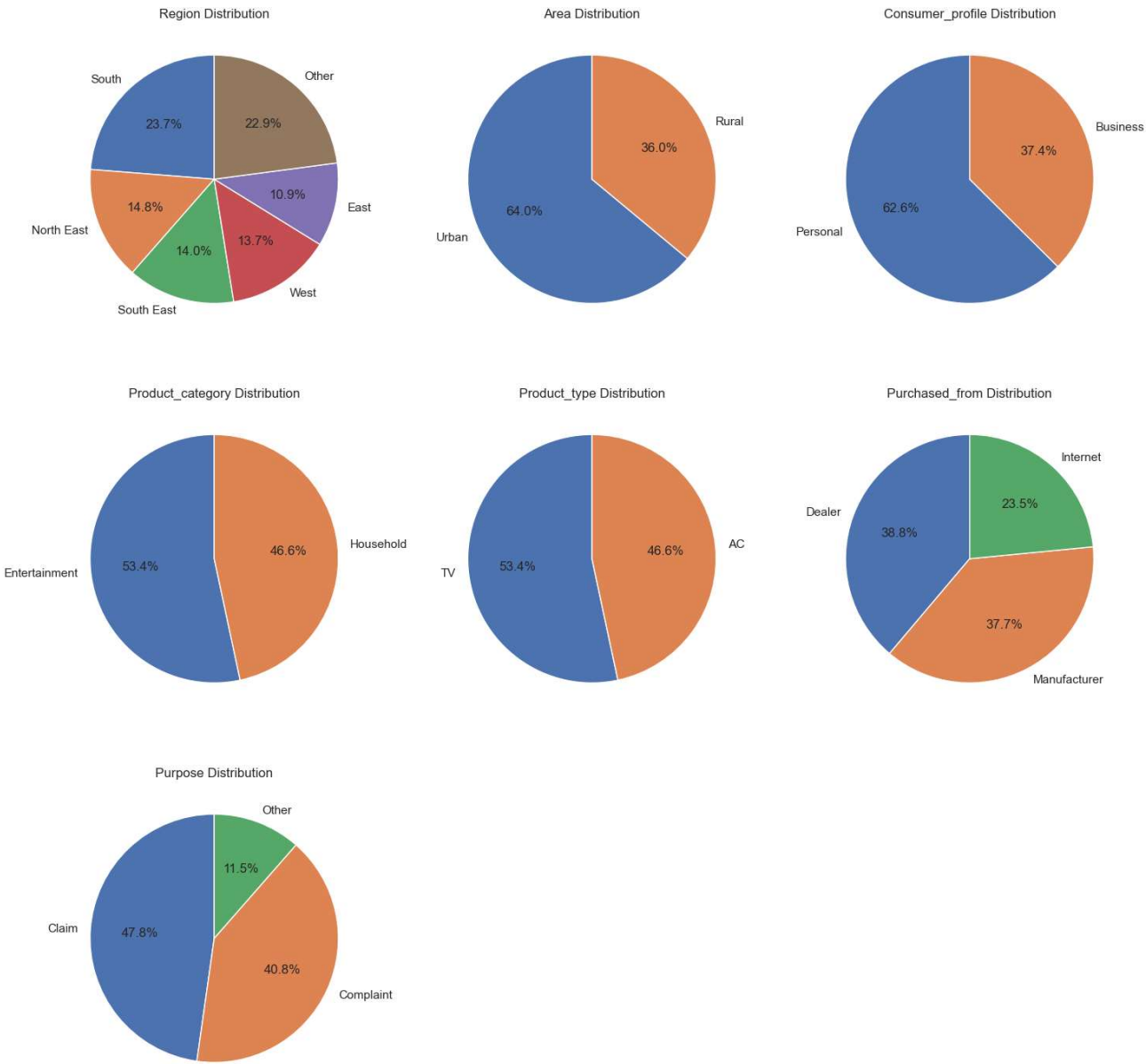
        # Set a title for each subplot
        axs.flat[i].set_title(f'{var} Distribution')

# Adjust spacing between subplots
fig.tight_layout()

# remove eighth plot
fig.delaxes(axs[2][1])

# remove ninth plot
fig.delaxes(axs[2][2])

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



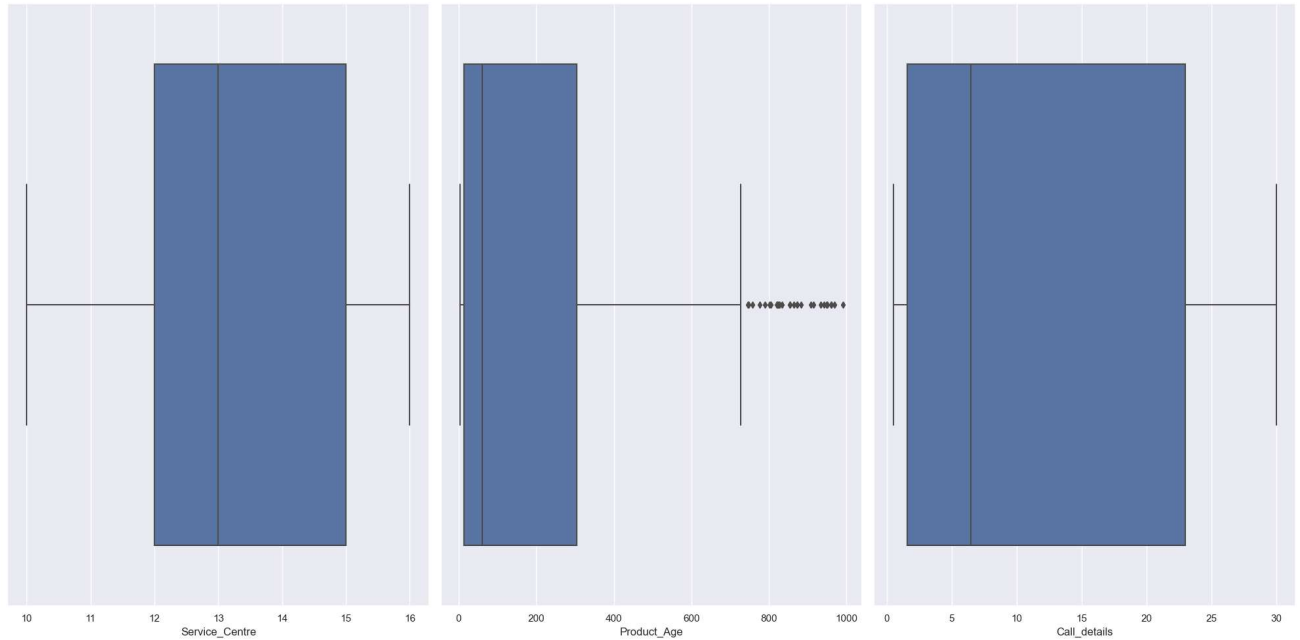
```
In [10]: num_vars = ['Service_Centre', 'Product_Age', 'Call_details']

fig, axs = plt.subplots(nrows=1, ncols=3, figsize=(20, 10))
axs = axs.flatten()

for i, var in enumerate(num_vars):
    sns.boxplot(x=var, data=df, ax=axs[i])

fig.tight_layout()

plt.show()
```



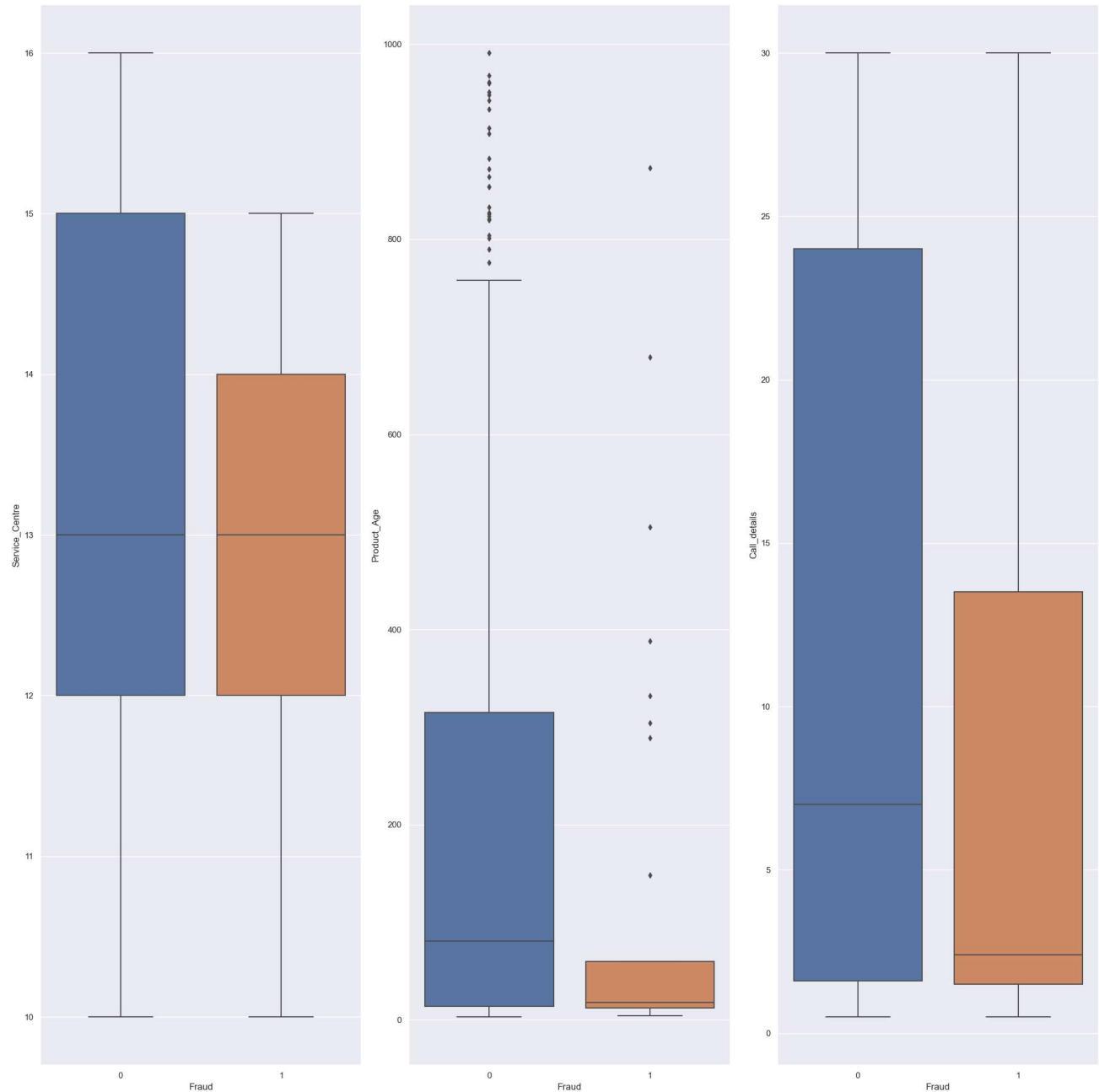

```
In [12]: num_vars = ['Service_Centre', 'Product_Age', 'Call_details']

fig, axs = plt.subplots(nrows=1, ncols=3, figsize=(20, 20))
axs = axs.flatten()

for i, var in enumerate(num_vars):
    sns.boxplot(y=var, x='Fraud', data=df, ax=axs[i])

fig.tight_layout()

plt.show()
```



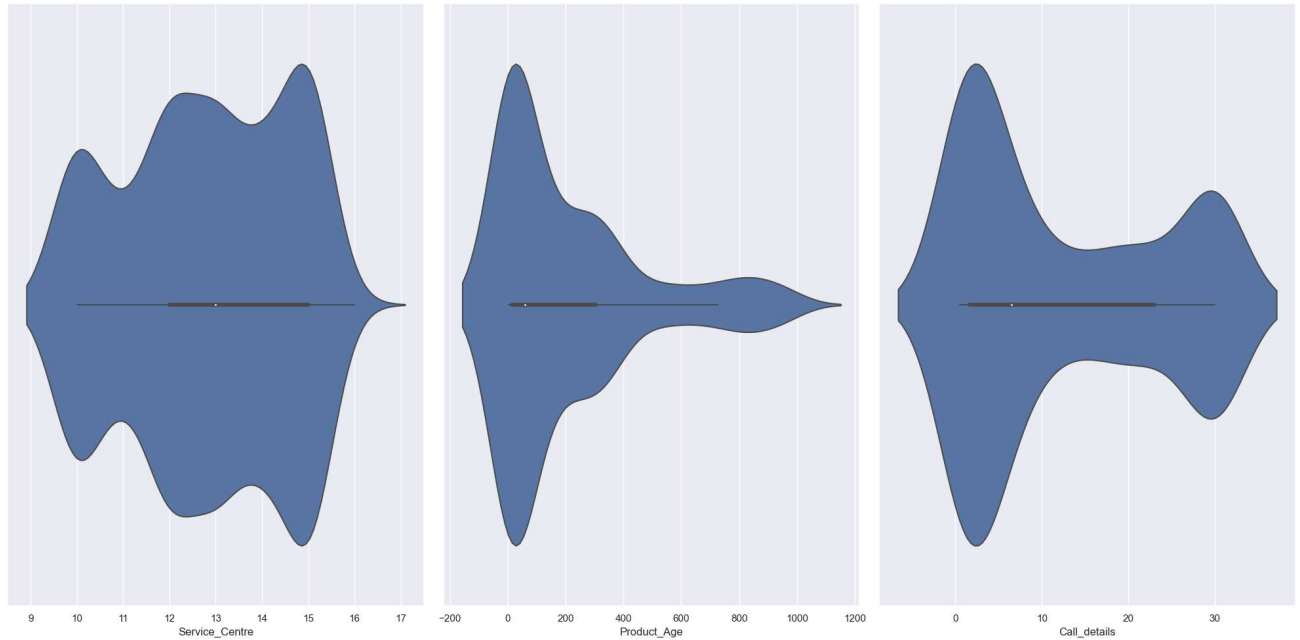
```
In [13]: num_vars = ['Service_Centre', 'Product_Age', 'Call_details']

fig, axs = plt.subplots(nrows=1, ncols=3, figsize=(20, 10))
axs = axs.flatten()

for i, var in enumerate(num_vars):
    sns.violinplot(x=var, data=df, ax=axs[i])

fig.tight_layout()

plt.show()
```



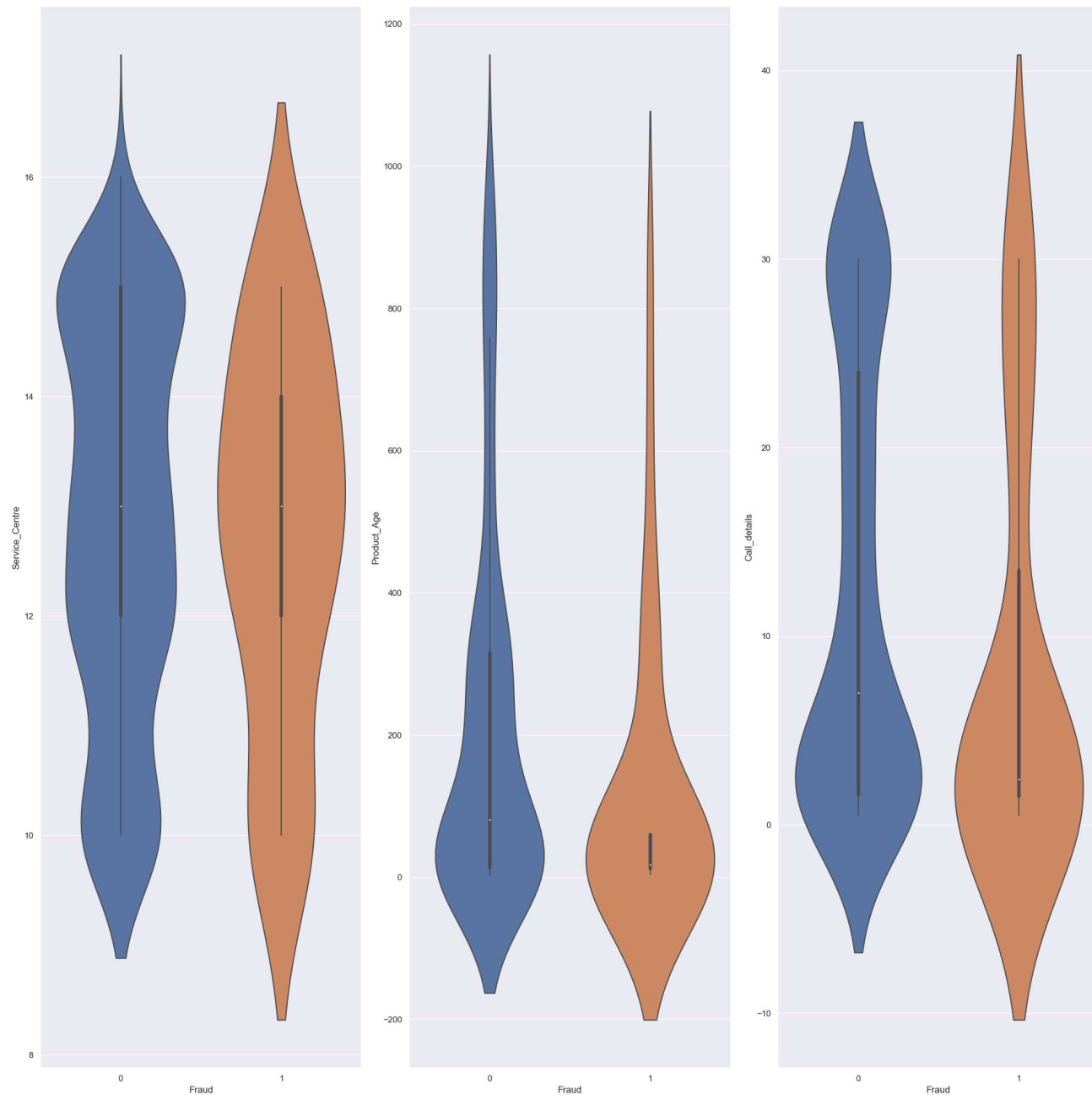
```
In [15]: num_vars = ['Service_Centre', 'Product_Age', 'Call_details']

fig, axs = plt.subplots(nrows=1, ncols=3, figsize=(20, 20))
axs = axs.flatten()

for i, var in enumerate(num_vars):
    sns.violinplot(y=var, data=df, x='Fraud', ax=axs[i])

fig.tight_layout()

plt.show()
```



Data Preprocessing Part 2

```
In [16]: #Check missing value
check_missing = df.isnull().sum() * 100 / df.shape[0]
check_missing[check_missing > 0].sort_values(ascending=False)
```

```
Out[16]: Series([], dtype: float64)
```

Label Encoding for each Object datatype

```
In [17]: # Loop over each column in the DataFrame where dtype is 'object'
for col in df.select_dtypes(include=['object']).columns:

    # Print the column name and the unique values
    print(f"{col}: {df[col].unique()}")

Region: ['South' 'North' 'North East' 'North West' 'South East' 'South West'
        'West' 'East']
Area: ['Urban' 'Rural']
Consumer_profile: ['Business' 'Personal']
Product_category: ['Entertainment' 'Household']
Product_type: ['TV' 'AC']
Purchased_from: ['Manufacturer' 'Dealer' 'Internet']
Purpose: ['Complaint' 'Claim' 'Other']
```

```
In [18]: from sklearn import preprocessing

# Loop over each column in the DataFrame where dtype is 'object'
for col in df.select_dtypes(include=['object']).columns:

    # Initialize a LabelEncoder object
    label_encoder = preprocessing.LabelEncoder()

    # Fit the encoder to the unique values in the column
    label_encoder.fit(df[col].unique())

    # Transform the column using the encoder
    df[col] = label_encoder.transform(df[col])

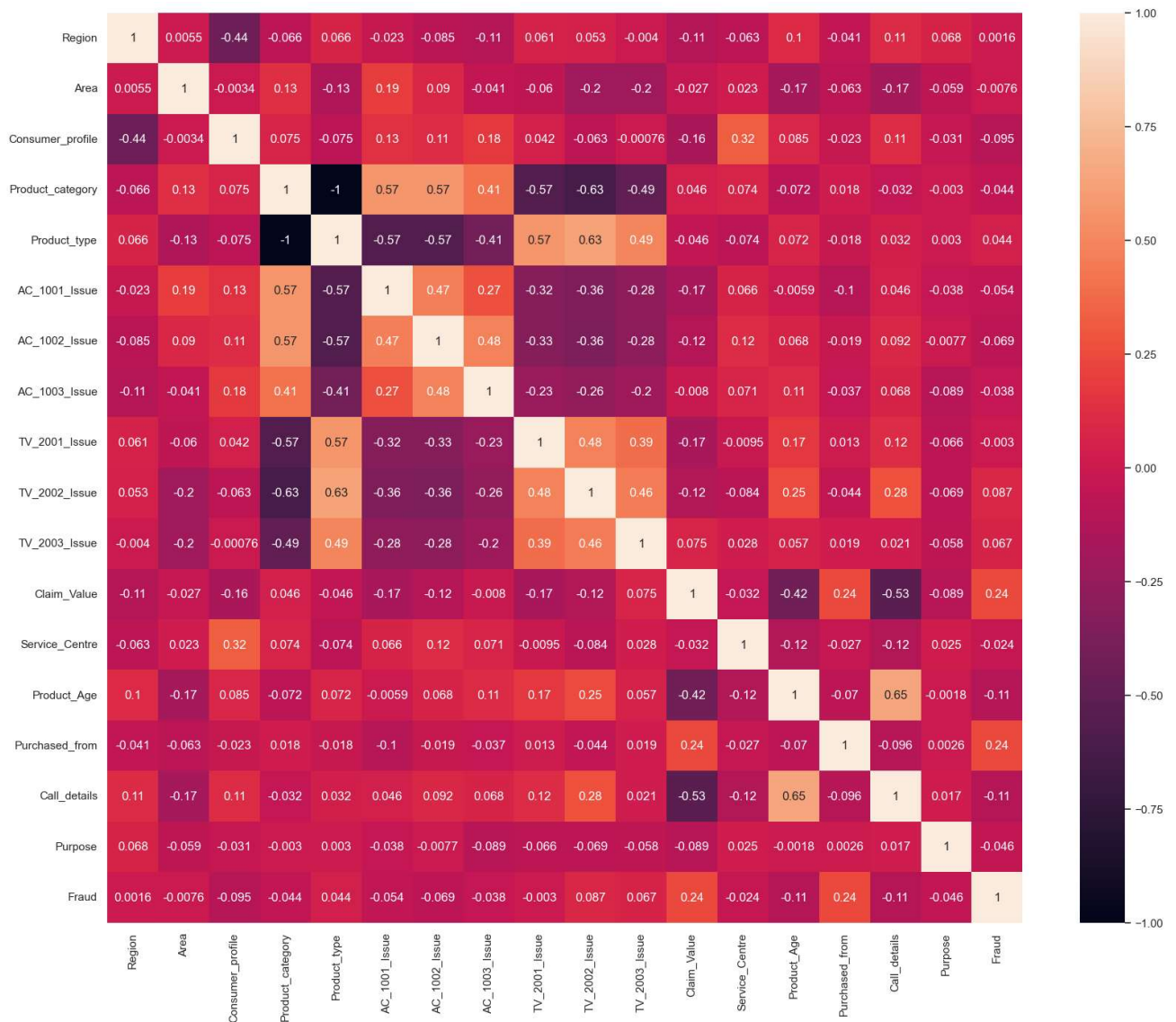
    # Print the column name and the unique encoded values
    print(f"{col}: {df[col].unique()}")
```

```
Region: [4 1 2 3 5 6 7 0]
Area: [1 0]
Consumer_profile: [0 1]
Product_category: [0 1]
Product_type: [1 0]
Purchased_from: [2 0 1]
Purpose: [1 0 2]
```

Correlation Heatmap

```
In [19]: #Correlation Heatmap (print the correlation score each variables)
plt.figure(figsize=(20, 16))
sns.heatmap(df.corr(), fmt='.2g', annot=True)
```

Out[19]: <AxesSubplot:>



Train Test Split

```
In [20]: from sklearn.model_selection import train_test_split
# Select the features (X) and the target variable (y)
X = df.drop('Fraud', axis=1)
y = df['Fraud']

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

Remove the Outlier from train data using Z-Score

```
In [22]: from scipy import stats

# Define the columns for which you want to remove outliers
selected_columns = ['Service_Centre', 'Product_Age', 'Call_details']

# Calculate the Z-scores for the selected columns in the training data
z_scores = np.abs(stats.zscore(X_train[selected_columns]))

# Set a threshold value for outlier detection (e.g., 3)
threshold = 3

# Find the indices of outliers based on the threshold
outlier_indices = np.where(z_scores > threshold)[0]

# Remove the outliers from the training data
X_train = X_train.drop(X_train.index[outlier_indices])
y_train = y_train.drop(y_train.index[outlier_indices])
```

Decision Tree

```
In [23]: from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
dtree = DecisionTreeClassifier(class_weight='balanced')
param_grid = {
    'max_depth': [3, 4, 5, 6, 7, 8],
    'min_samples_split': [2, 3, 4],
    'min_samples_leaf': [1, 2, 3, 4],
    'random_state': [0, 42]
}

# Perform a grid search with cross-validation to find the best hyperparameters
grid_search = GridSearchCV(dtree, param_grid, cv=5)
grid_search.fit(X_train, y_train)

# Print the best hyperparameters
print(grid_search.best_params_)

{'max_depth': 8, 'min_samples_leaf': 1, 'min_samples_split': 2, 'random_state': 0}
```

```
In [24]: from sklearn.tree import DecisionTreeClassifier
dtree = DecisionTreeClassifier(random_state=0, max_depth=8, min_samples_leaf=1, min_samples_split=2, class_weight='balanced')
dtree.fit(X_train, y_train)
```

Out[24]: DecisionTreeClassifier(class_weight='balanced', max_depth=8, random_state=0)

```
In [25]: from sklearn.metrics import accuracy_score
y_pred = dtree.predict(X_test)
print("Accuracy Score :", round(accuracy_score(y_test, y_pred)*100, 2), "%")

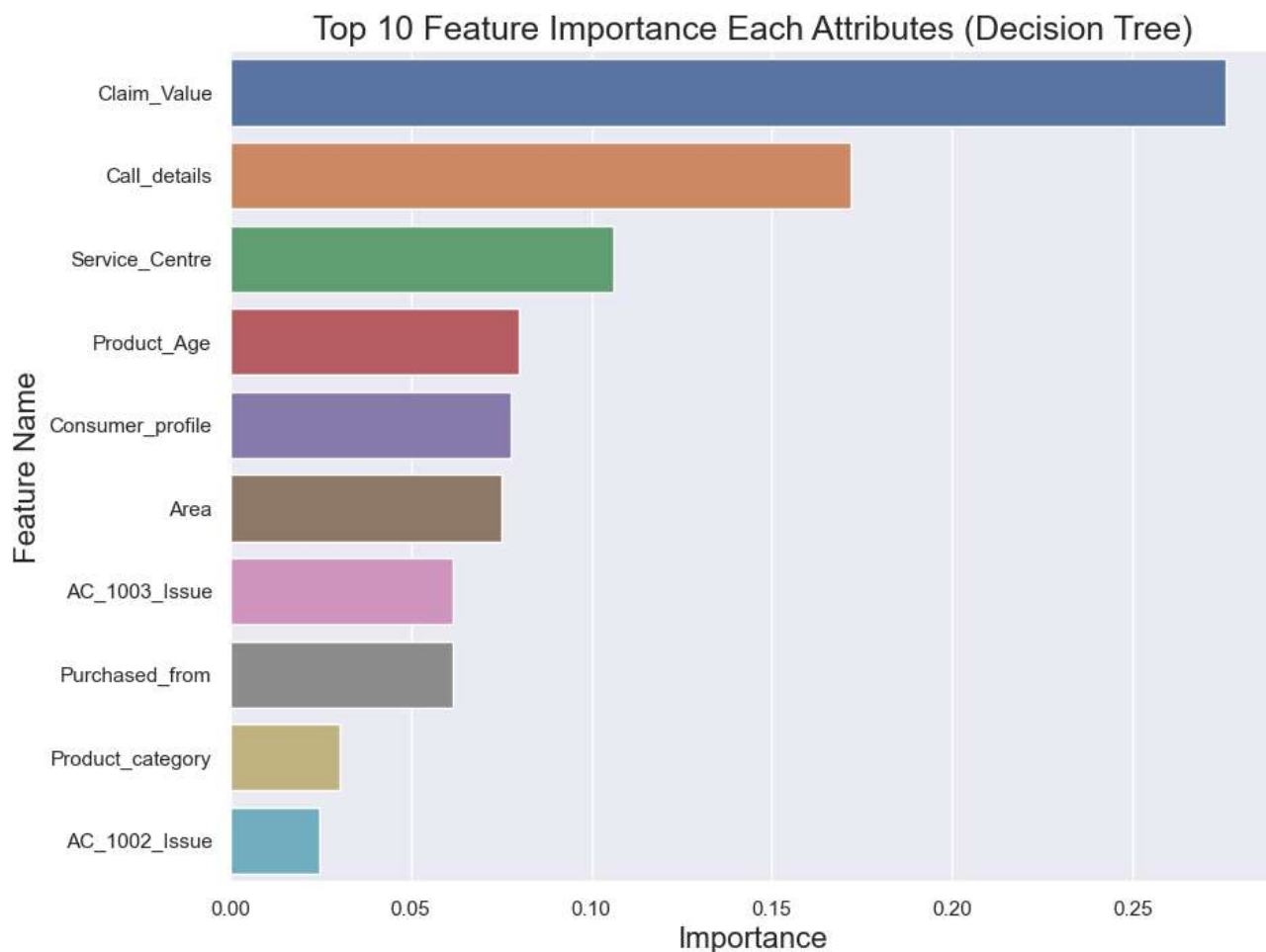
Accuracy Score : 84.72 %
```

```
In [26]: from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, jaccard_score, log_loss
print('F-1 Score : ', (f1_score(y_test, y_pred, average='micro')))
print('Precision Score : ', (precision_score(y_test, y_pred, average='micro')))
print('Recall Score : ', (recall_score(y_test, y_pred, average='micro')))
print('Jaccard Score : ', (jaccard_score(y_test, y_pred, average='micro')))
print('Log Loss : ', (log_loss(y_test, y_pred)))

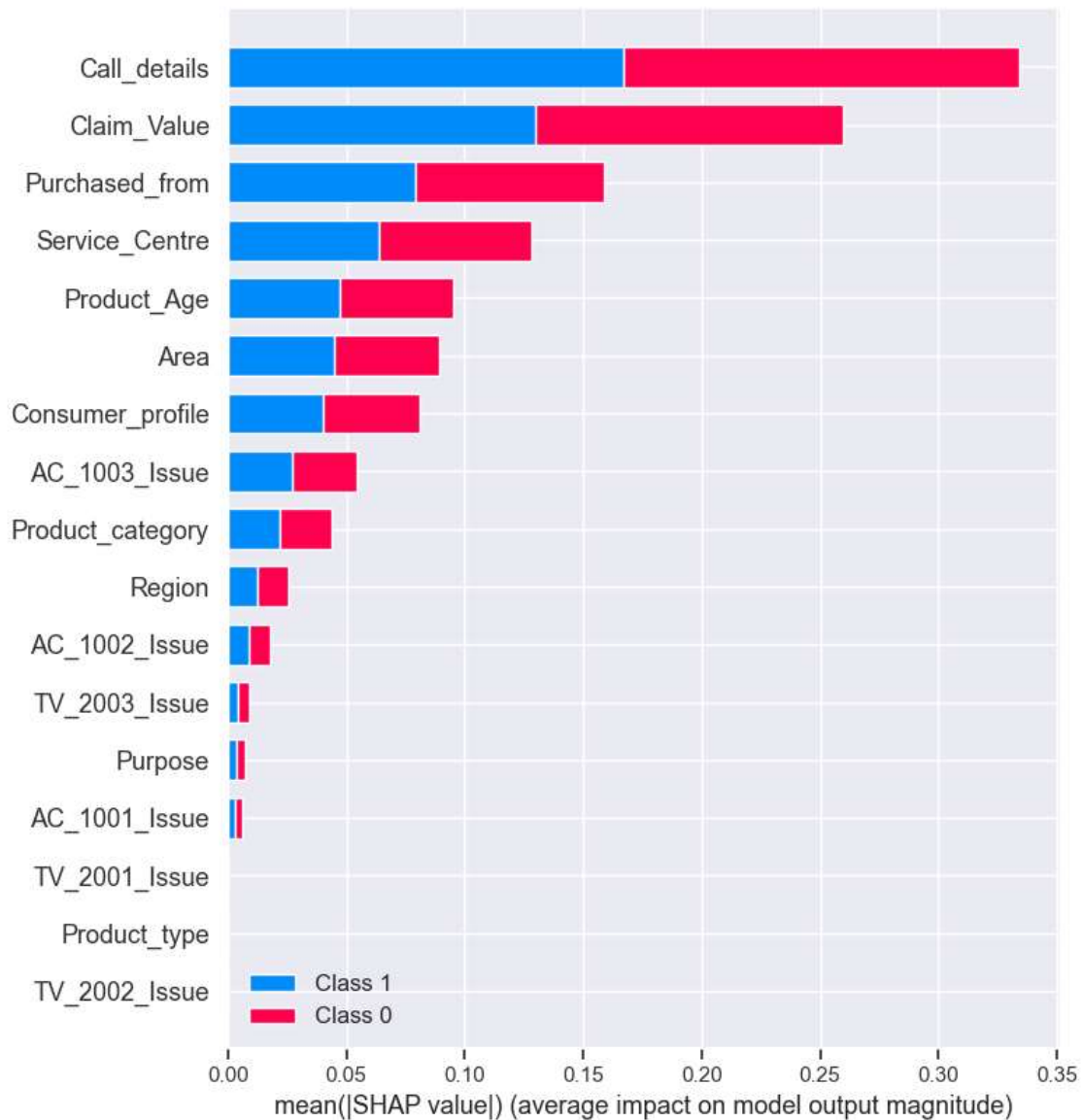
F-1 Score : 0.8472222222222222
Precision Score : 0.8472222222222222
Recall Score : 0.8472222222222222
Jaccard Score : 0.7349397590361446
Log Loss : 5.276846348936932
```

```
In [27]: imp_df = pd.DataFrame({
    "Feature Name": X_train.columns,
    "Importance": dtree.feature_importances_
})
fi = imp_df.sort_values(by="Importance", ascending=False)

fi2 = fi.head(10)
plt.figure(figsize=(10,8))
sns.barplot(data=fi2, x='Importance', y='Feature Name')
plt.title('Top 10 Feature Importance Each Attributes (Decision Tree)', fontsize=18)
plt.xlabel ('Importance', fontsize=16)
plt.ylabel ('Feature Name', fontsize=16)
plt.show()
```



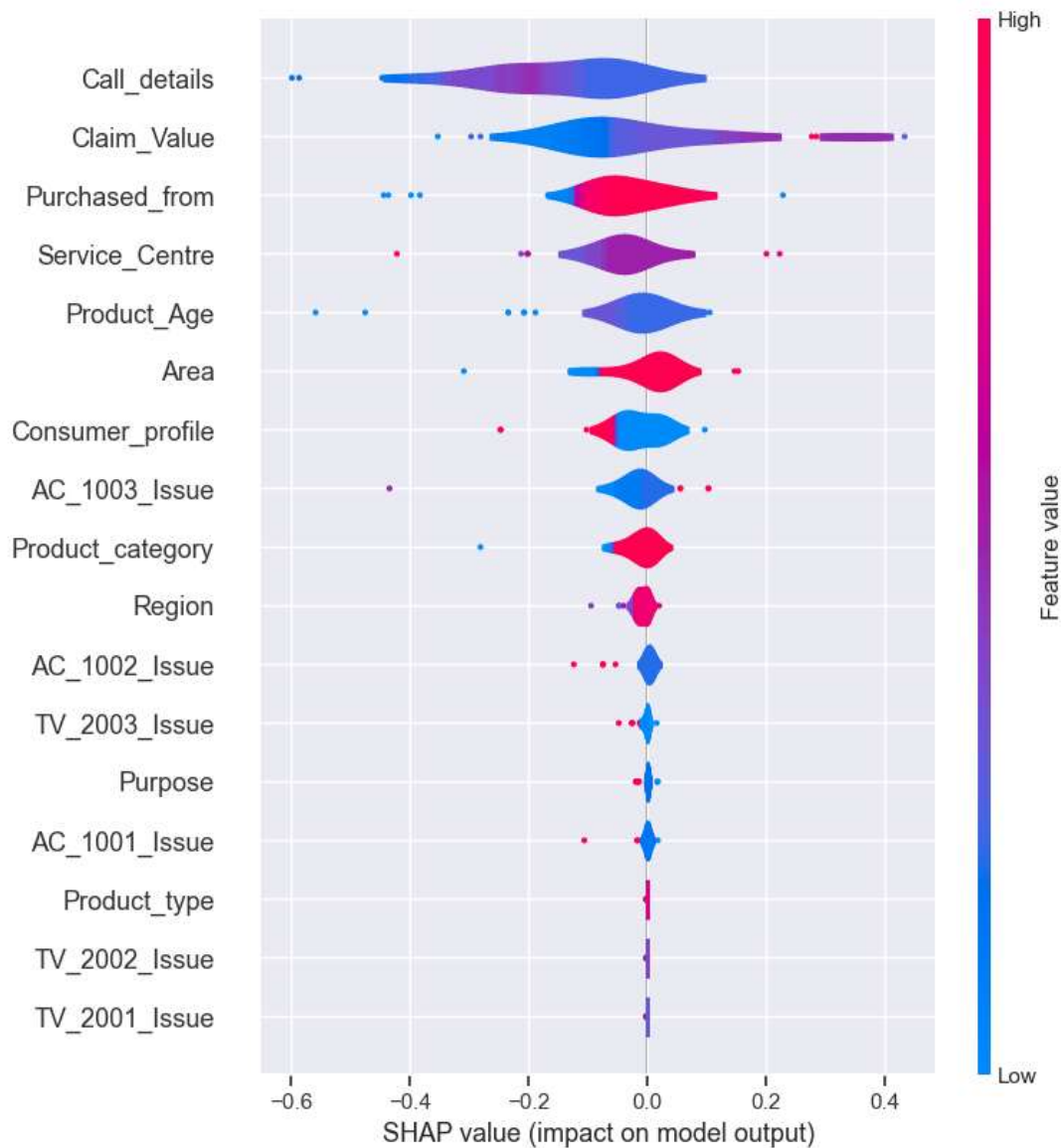
```
In [28]: import shap
explainer = shap.TreeExplainer(dtree)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test)
```




```
In [29]: # compute SHAP values
explainer = shap.TreeExplainer(dtree)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values[1], X_test.values, feature_names = X_test.columns)
```



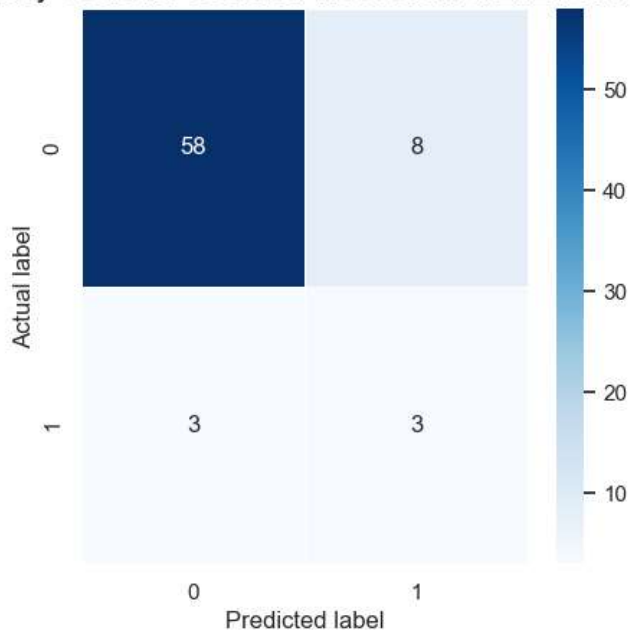
```
In [30]: # compute SHAP values
explainer = shap.TreeExplainer(dtree)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values[1], X_test.values, feature_names = X_test.columns, plot_type="violin")
```



```
In [31]: from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(5,5))
sns.heatmap(data=cm,linewidths=.5, annot=True, cmap = 'Blues')
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
all_sample_title = 'Accuracy Score for Decision Tree: {}'.format(dtreescore(X_test, y_test))
plt.title(all_sample_title, size = 15)
```

Out[31]: Text(0.5, 1.0, 'Accuracy Score for Decision Tree: 0.847222222222222')

Accuracy Score for Decision Tree: 0.847222222222222



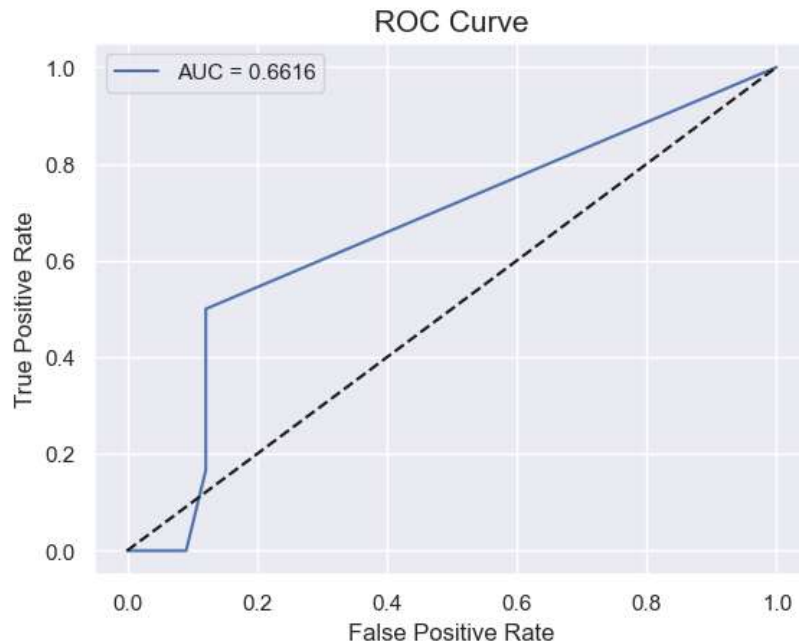
```
In [32]: from sklearn.metrics import roc_curve, roc_auc_score
y_pred_proba = dtree.predict_proba(X_test)[:,:1]

df_actual_predicted = pd.concat([pd.DataFrame(np.array(y_test), columns=['y_actual']), pd.DataFrame(y_pred_proba, columns=['y_pred_proba'])], axis=1)
df_actual_predicted.index = y_test.index

fpr, tpr, tr = roc_curve(df_actual_predicted['y_actual'], df_actual_predicted['y_pred_proba'])
auc = roc_auc_score(df_actual_predicted['y_actual'], df_actual_predicted['y_pred_proba'])

plt.plot(fpr, tpr, label='AUC = %0.4f' % auc)
plt.plot(fpr, fpr, linestyle='--', color='k')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve', size=15)
plt.legend()
```

Out[32]: <matplotlib.legend.Legend at 0x235e5420070>



Random Forest

```
In [33]: from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
rfc = RandomForestClassifier(class_weight='balanced')
param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [None, 5, 10],
    'max_features': ['sqrt', 'log2', None],
    'random_state': [0, 42]
}

# Perform a grid search with cross-validation to find the best hyperparameters
grid_search = GridSearchCV(rfc, param_grid, cv=5)
grid_search.fit(X_train, y_train)

# Print the best hyperparameters
print(grid_search.best_params_)

{'max_depth': 10, 'max_features': 'sqrt', 'n_estimators': 100, 'random_state': 42}
```

```
In [34]: from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(random_state=42, max_features='sqrt', n_estimators=100, max_depth=10, class_weight='balanced')
rfc.fit(X_train, y_train)
```

Out[34]: RandomForestClassifier(class_weight='balanced', max_depth=10, max_features='sqrt', random_state=42)

```
In [35]: y_pred = rfc.predict(X_test)
print("Accuracy Score :", round(accuracy_score(y_test, y_pred)*100 ,2), "%")
```

Accuracy Score : 91.67 %

```
In [36]: from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, jaccard_score, log_loss
print('F-1 Score : ',(f1_score(y_test, y_pred, average='micro')))
print('Precision Score : ',(precision_score(y_test, y_pred, average='micro')))
print('Recall Score : ',(recall_score(y_test, y_pred, average='micro')))
print('Jaccard Score : ',(jaccard_score(y_test, y_pred, average='micro')))
print('Log Loss : ',(log_loss(y_test, y_pred)))
```

F-1 Score : 0.9166666666666666

Precision Score : 0.9166666666666666

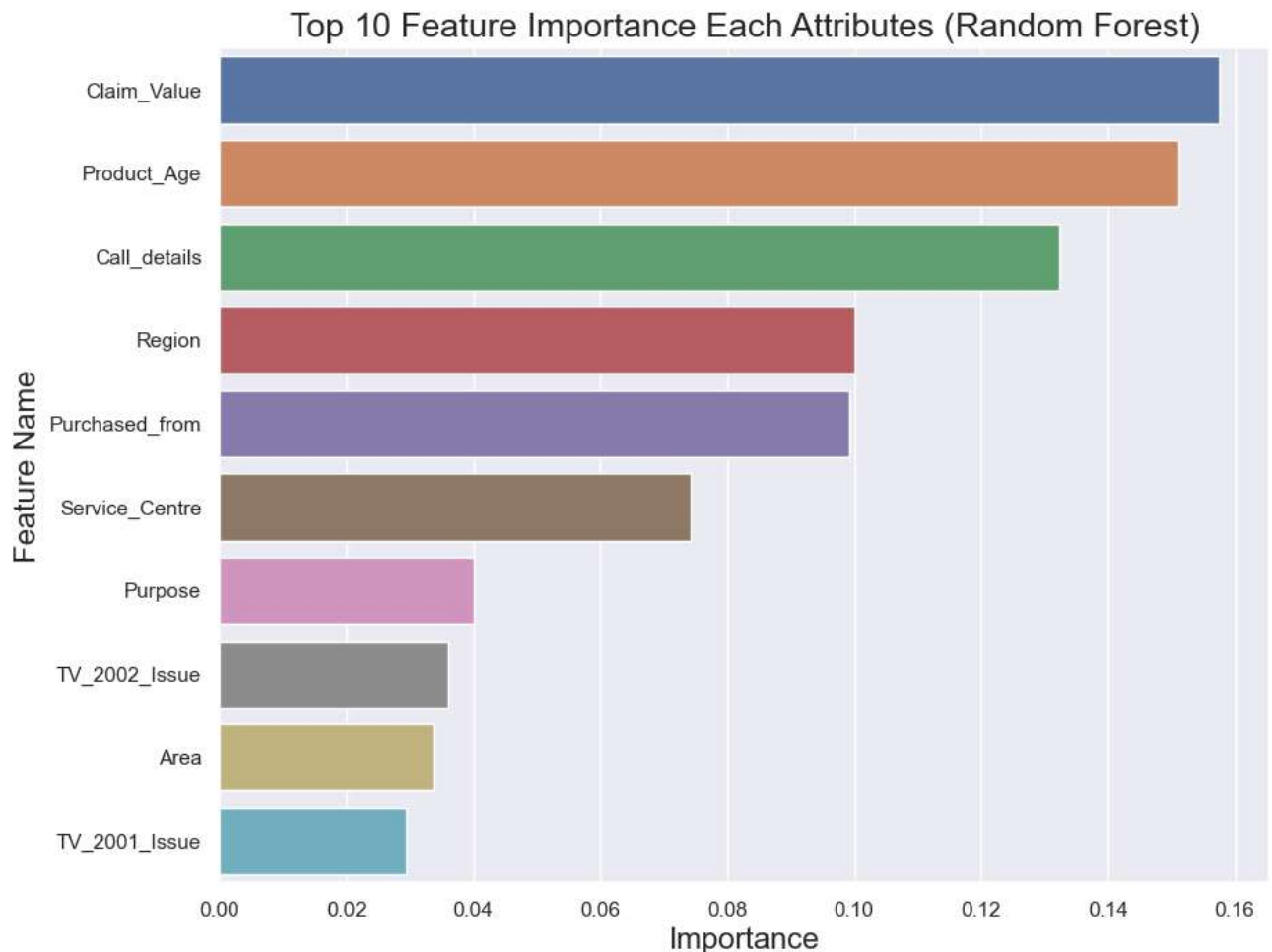
Recall Score : 0.9166666666666666

Jaccard Score : 0.8461538461538461

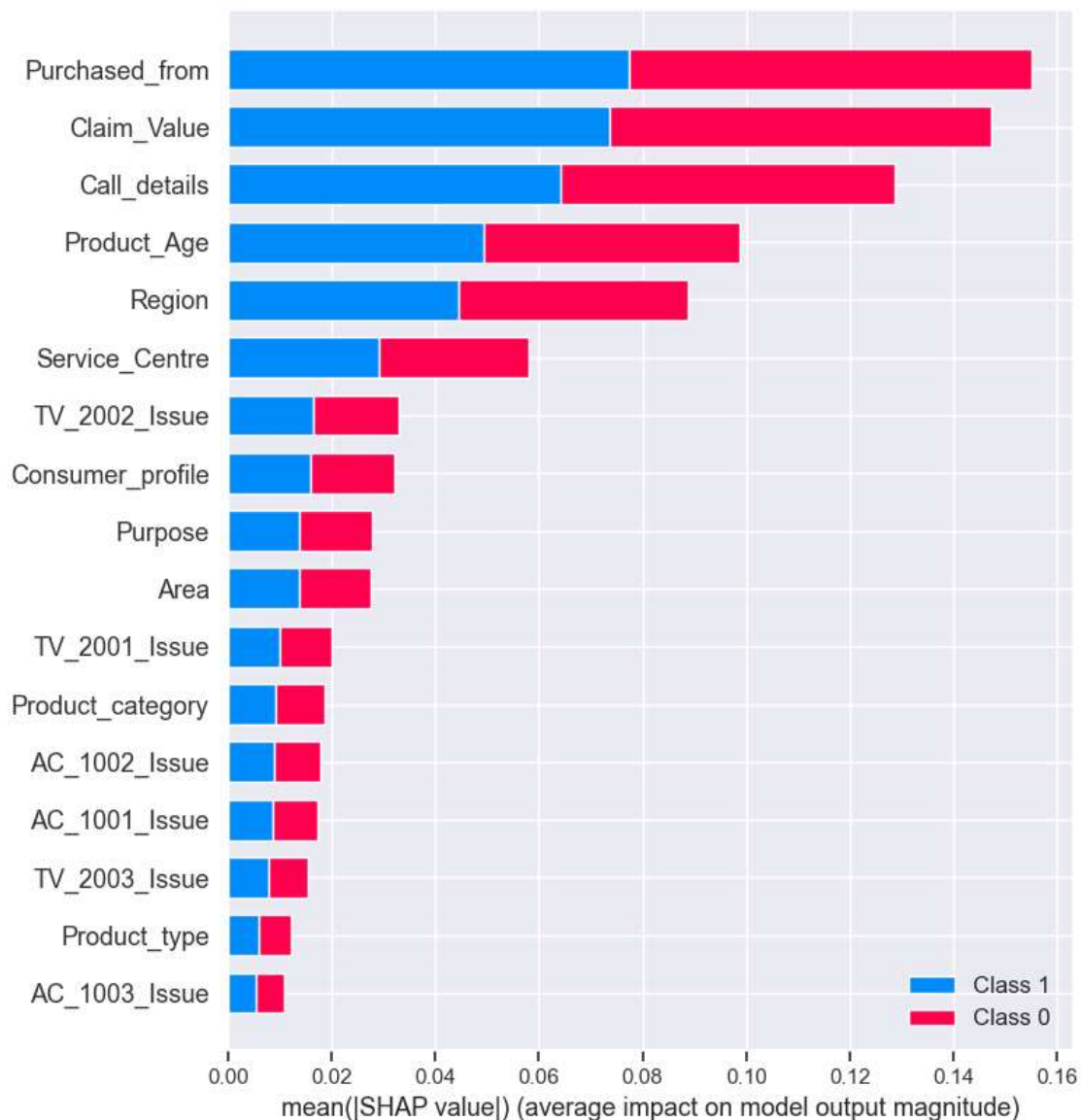
Log Loss : 2.878253577282285

```
In [37]: imp_df = pd.DataFrame({
    "Feature Name": X_train.columns,
    "Importance": rfc.feature_importances_
})
fi = imp_df.sort_values(by="Importance", ascending=False)

fi2 = fi.head(10)
plt.figure(figsize=(10,8))
sns.barplot(data=fi2, x='Importance', y='Feature Name')
plt.title('Top 10 Feature Importance Each Attributes (Random Forest)', fontsize=18)
plt.xlabel('Importance', fontsize=16)
plt.ylabel('Feature Name', fontsize=16)
plt.show()
```



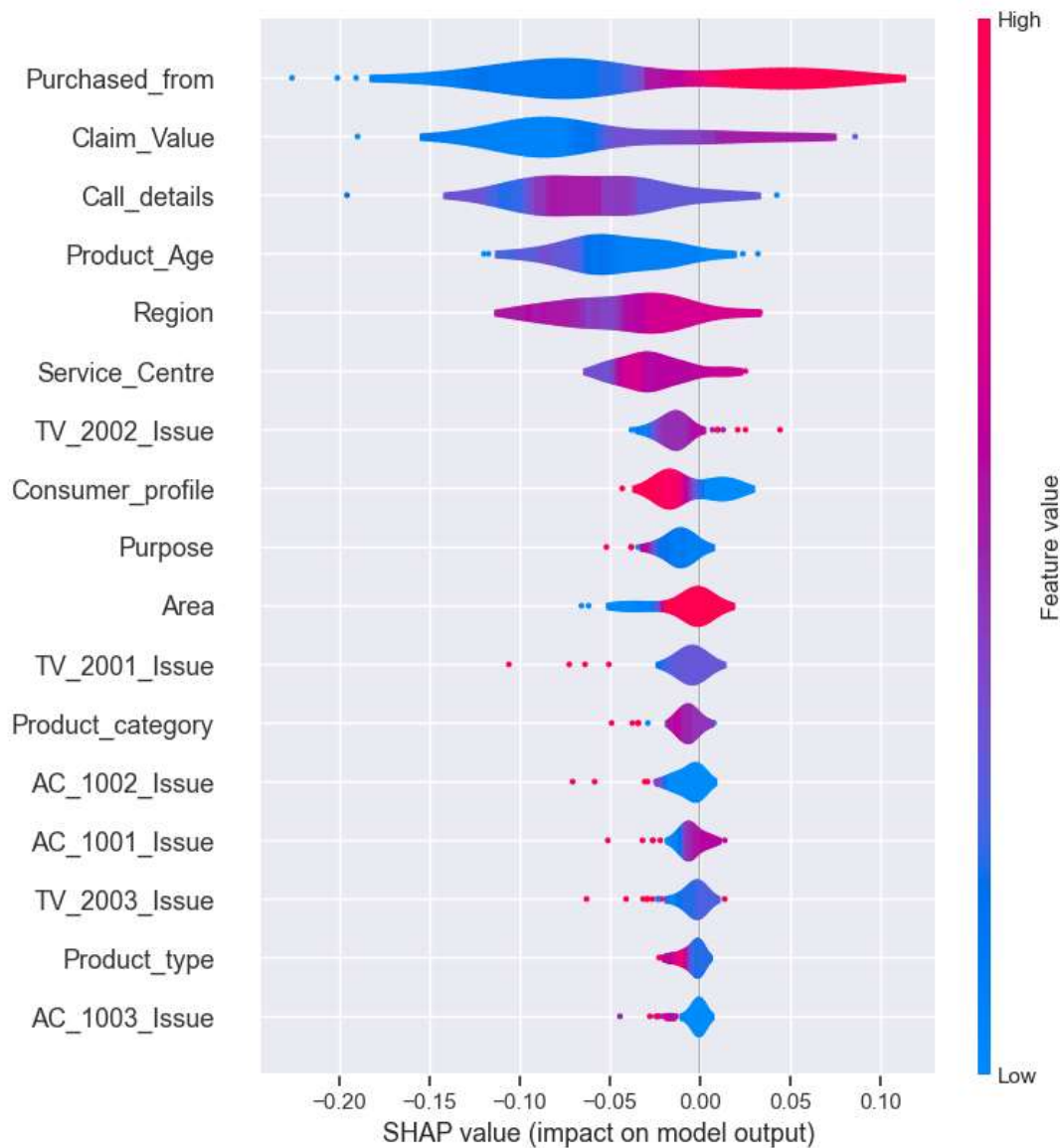
```
In [38]: import shap
explainer = shap.TreeExplainer(rfc)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test)
```



```
In [39]: # compute SHAP values
explainer = shap.TreeExplainer(rfc)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values[1], X_test.values, feature_names = X_test.columns)
```



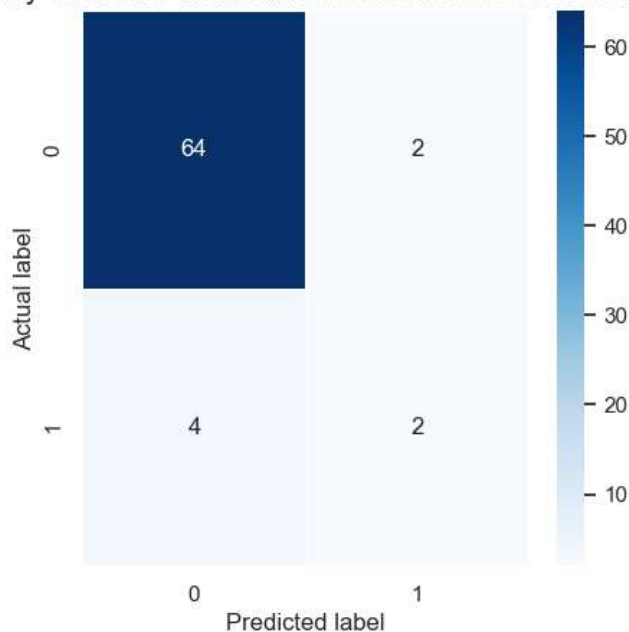
```
In [40]: # compute SHAP values
explainer = shap.TreeExplainer(rfc)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values[1], X_test.values, feature_names = X_test.columns, plot_type="violin")
```




```
In [41]: from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(5,5))
sns.heatmap(data=cm,linewidths=.5, annot=True, cmap = 'Blues')
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
all_sample_title = 'Accuracy Score for Random Forest: {}'.format(rfc.score(X_test, y_test))
plt.title(all_sample_title, size = 15)
```

Out[41]: Text(0.5, 1.0, 'Accuracy Score for Random Forest: 0.9166666666666666')

Accuracy Score for Random Forest: 0.9166666666666666



```
In [42]: from sklearn.metrics import roc_curve, roc_auc_score
y_pred_proba = rfc.predict_proba(X_test)[:][:,1]

df_actual_predicted = pd.concat([pd.DataFrame(np.array(y_test), columns=['y_actual']), pd.DataFrame(y_pred_proba, columns=['y_pred_proba'])], axis=1)
df_actual_predicted.index = y_test.index

fpr, tpr, _ = roc_curve(df_actual_predicted['y_actual'], df_actual_predicted['y_pred_proba'])
auc = roc_auc_score(df_actual_predicted['y_actual'], df_actual_predicted['y_pred_proba'])

plt.plot(fpr, tpr, label='AUC = %0.4f' % auc)
plt.plot(fpr, fpr, linestyle='--', color='k')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve', size=15)
plt.legend()
```

Out[42]: <matplotlib.legend.Legend at 0x235e63cfac0>

