Bank Churn Data Exploration and Churn Prediction

Table Of Content

Libraries And Utilities

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as ex
import plotly.graph_objs as go
import plotly.figure_factory as ff
from plotly.subplots import make_subplots
import plotly.offline as pyo
pyo.init_notebook_mode()
sns.set_style('darkgrid')
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split,cross_val_score
from sklearn.ensemble import RandomForestClassifier,AdaBoostClassifier
from sklearn.svm import SVC
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import f1_score as f1
from sklearn.metrics import confusion_matrix
import scikitplot as skplt

plt.rc('figure',figsize=(18,9))
%pip install imbalanced-learn
from imblearn.over_sampling import SMOTE
```

```
Requirement already satisfied: imbalanced-learn in /opt/conda/lib/python3.7/site-packages (0.7.0)
Requirement already satisfied: joblib>=0.11 in /opt/conda/lib/python3.7/site-packages (from imbalanced-learn) (0.14.1)
Requirement already satisfied: scikit-learn>=0.23 in /opt/conda/lib/python3.7/site-packages (from imbalanced-learn) (0.23.2)
Requirement already satisfied: numpy>=1.13.3 in /opt/conda/lib/python3.7/site-packages (from imbalanced-learn) (1.18.5)
Requirement already satisfied: scipy>=0.19.1 in /opt/conda/lib/python3.7/site-packages (from imbalanced-learn) (1.4.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in /opt/conda/lib/python3.7/site-packages (from scikit-learn>=0.23->imbalanced-learn) (2.1.0)
WARNING: You are using pip version 20.2.4; however, version 23.3.1 is available.
You should consider upgrading via the '/opt/conda/bin/python3.7 -m pip install --upgrade pip' command.
Note: you may need to restart the kernel to use updated packages.
```

Data Loading

In [2]:
```
c_data = pd.read_csv('/input/credit-card-customers/BankChurners.csv')
c_data = c_data[c_data.columns[:-2]]
c_data.head(3)
```

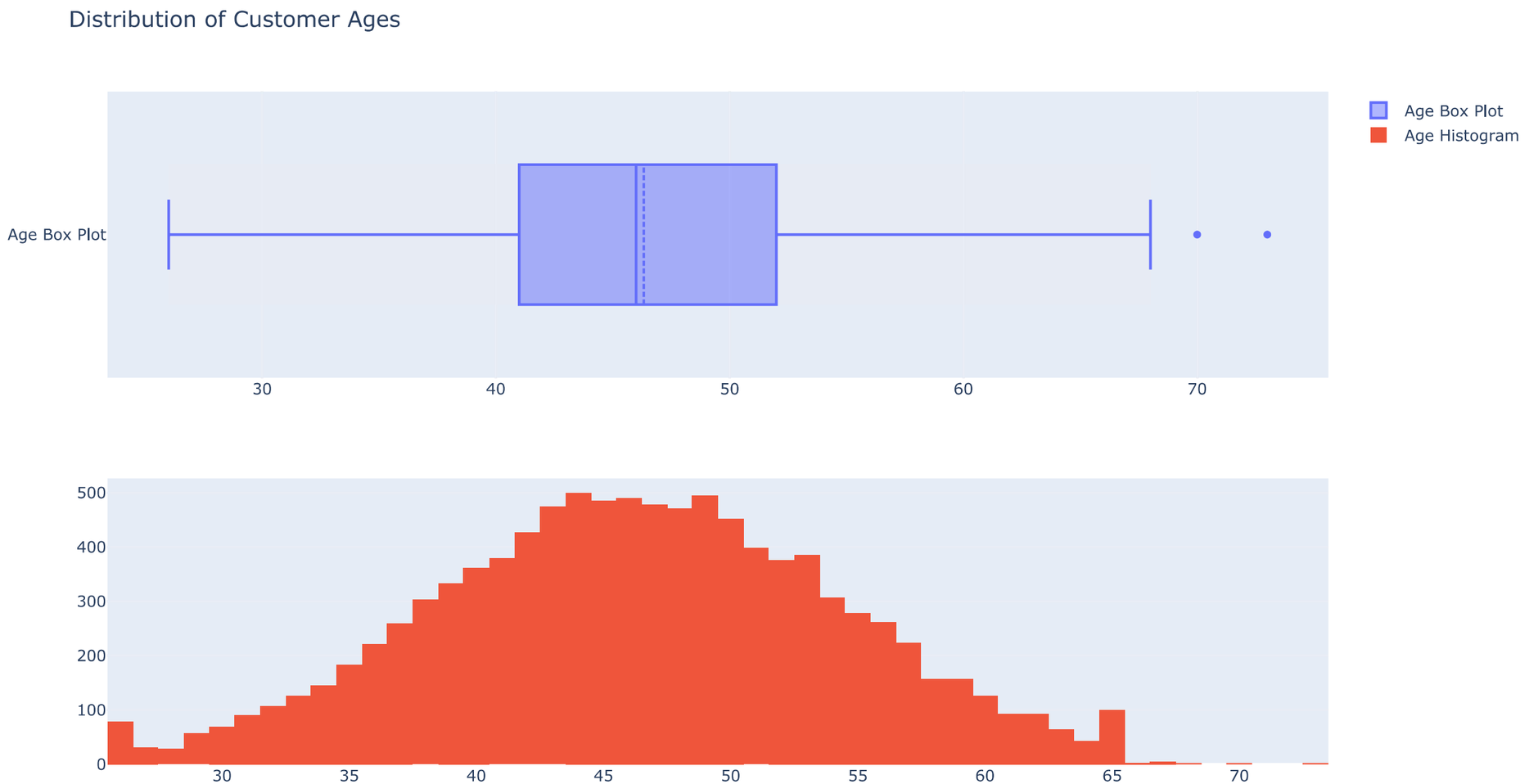
Out[2]:

	CLIENTNUM	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	Marital_Status	Income_Category	Card_Category	Months_on_book	...	Months_Inactive_12_mon	Contacts_Count_12_mon	Credit_Limit	Total_Revolv
0	768805383	Existing Customer	45	M	3	High School	Married	$60K - 80K$	Blue	39	...	1	3	12691.0	
1	818770008	Existing Customer	49	F	5	Graduate	Single	Less than $40K	Blue	44	...	1	2	8256.0	
2	713982108	Existing Customer	51	M	3	Graduate	Married	$80K - 120K$	Blue	36	...	1	0	3418.0	

3 rows × 21 columns

Exploratory Data Analysis

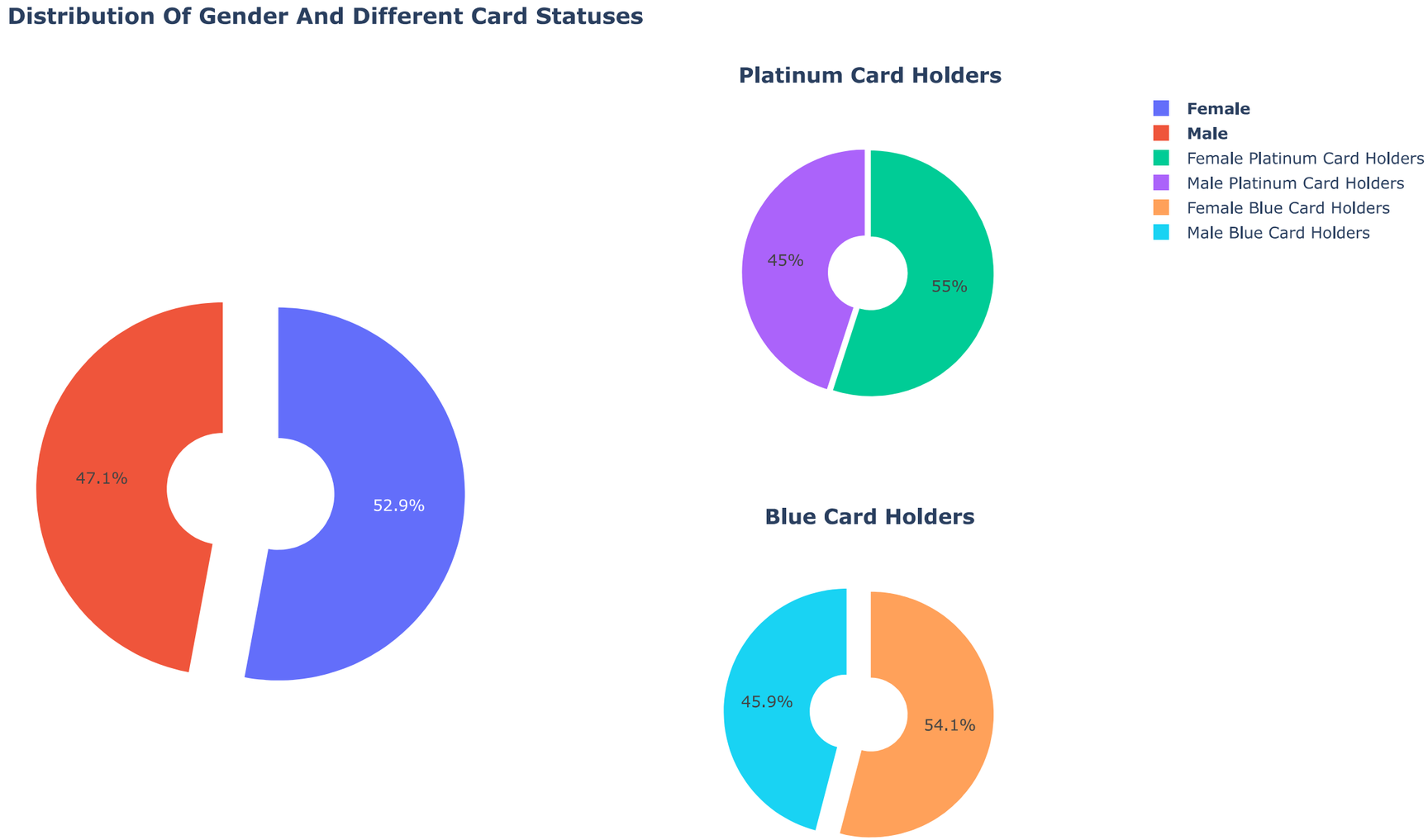
In [3]:
```
fig = make_subplots(rows=2, cols=1)

tr1=go.Box(x=c_data['Customer_Age'],name='Age Box Plot',boxmean=True)
tr2=go.Histogram(x=c_data['Customer_Age'],name='Age Histogram')

fig.add_trace(tr1,row=1,col=1)
fig.add_trace(tr2,row=2,col=1)

fig.update_layout(height=700, width=1200, title_text="Distribution of Customer Ages")
fig.show()
```



We can see that the distribution of customer ages in our dataset follows a fairly normal distribution; thus, further use of the age feature can be done with the normality assumption.

In [4]:
```
fig = make_subplots(
    rows=2, cols=2,subplot_titles=('','<b>Platinum Card Holders','<b>Blue Card Holders<b>','Residuals'),
    vertical_spacing=0.09,
    specs=[[{"type": "pie","rowspan": 2}       ,{"type": "pie"}] ,
           [None                               ,{"type": "pie"}]            ,
          ]
)

fig.add_trace(
    go.Pie(values=c_data.Gender.value_counts().values,labels=['<b>Female<b>','<b>Male<b>'],hole=0.3,pull=[0,0.3]),
    row=1, col=1
)
```

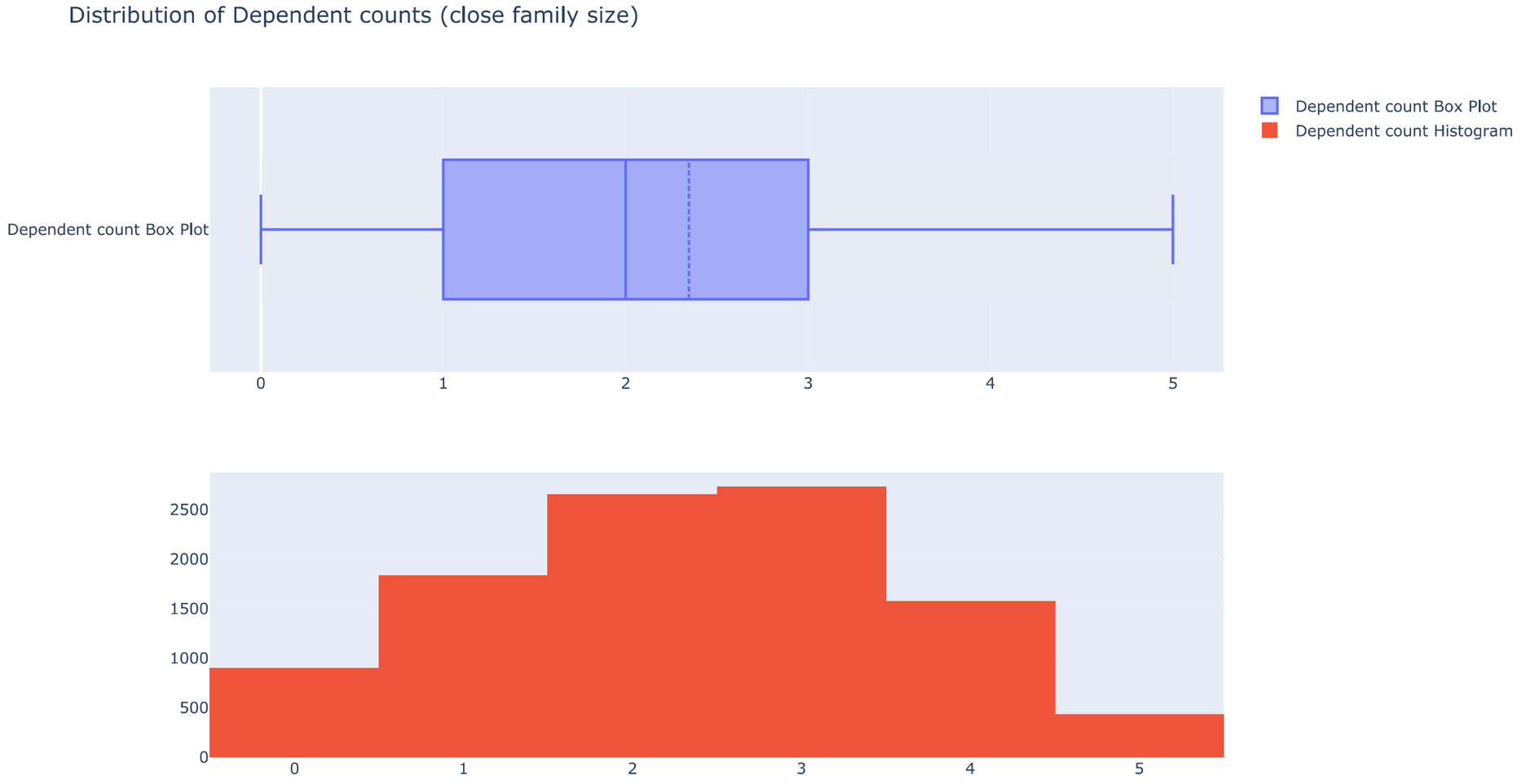
```
fig.add_trace(
    go.Pie(
        labels=['Female Platinum Card Holders','Male Platinum Card Holders'],
        values=c_data.query('Card_Category=="Platinum"').Gender.value_counts().values,
        pull=[0,0.05,0.5],
        hole=0.3

    ),
    row=1, col=2
)

fig.add_trace(
    go.Pie(
        labels=['Female Blue Card Holders','Male Blue Card Holders'],
        values=c_data.query('Card_Category=="Blue"').Gender.value_counts().values,
        pull=[0,0.2,0.5],
        hole=0.3
    ),
    row=2, col=2
)



fig.update_layout(
    height=800,
    showlegend=True,
    title_text="<b>Distribution Of Gender And Different Card Statuses<b>",
)

fig.show()
```



More samples of females in our dataset are compared to males, but the percentage of difference is not that significant, so we can say that genders are uniformly distributed.

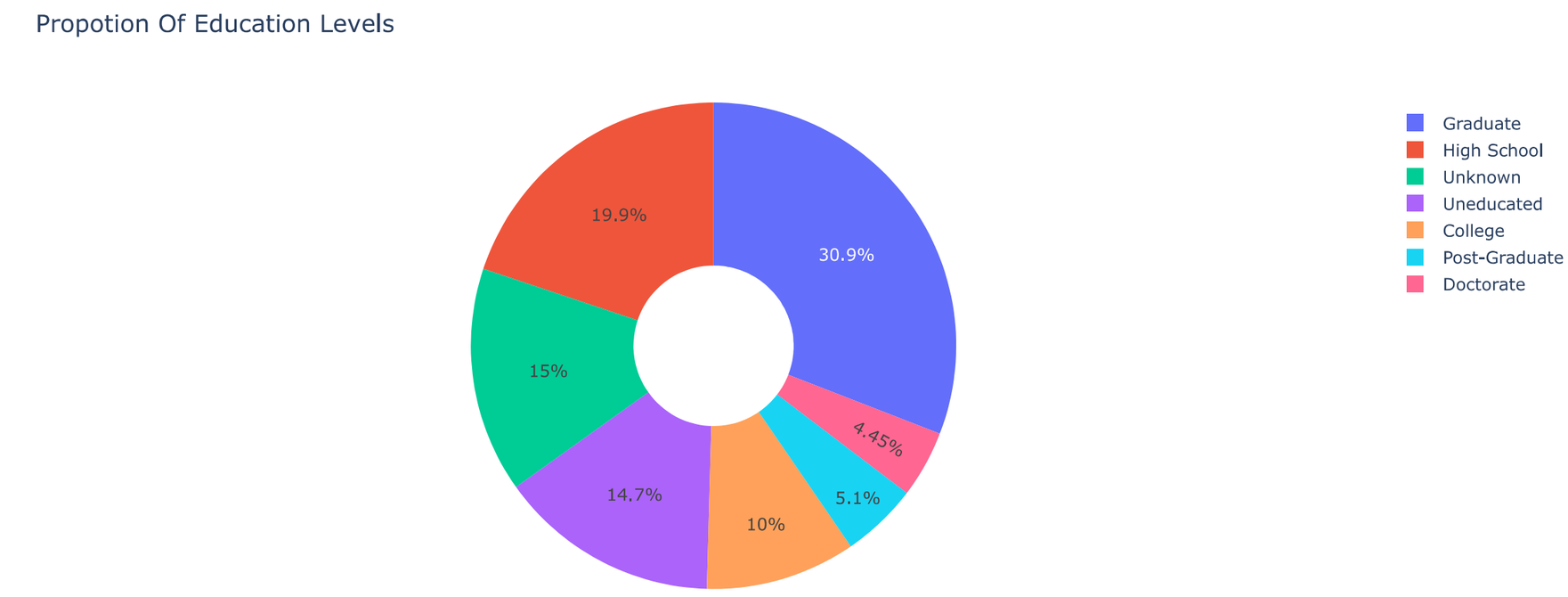
In [5]:
```
fig = make_subplots(rows=2, cols=1)

tr1=go.Box(x=c_data['Dependent_count'],name='Dependent count Box Plot',boxmean=True)
tr2=go.Histogram(x=c_data['Dependent_count'],name='Dependent count Histogram')

fig.add_trace(tr1,row=1,col=1)
fig.add_trace(tr2,row=2,col=1)
```

```
fig.update_layout(height=700, width=1200, title_text="Distribution of Dependent counts (close family size)")
fig.show()
```



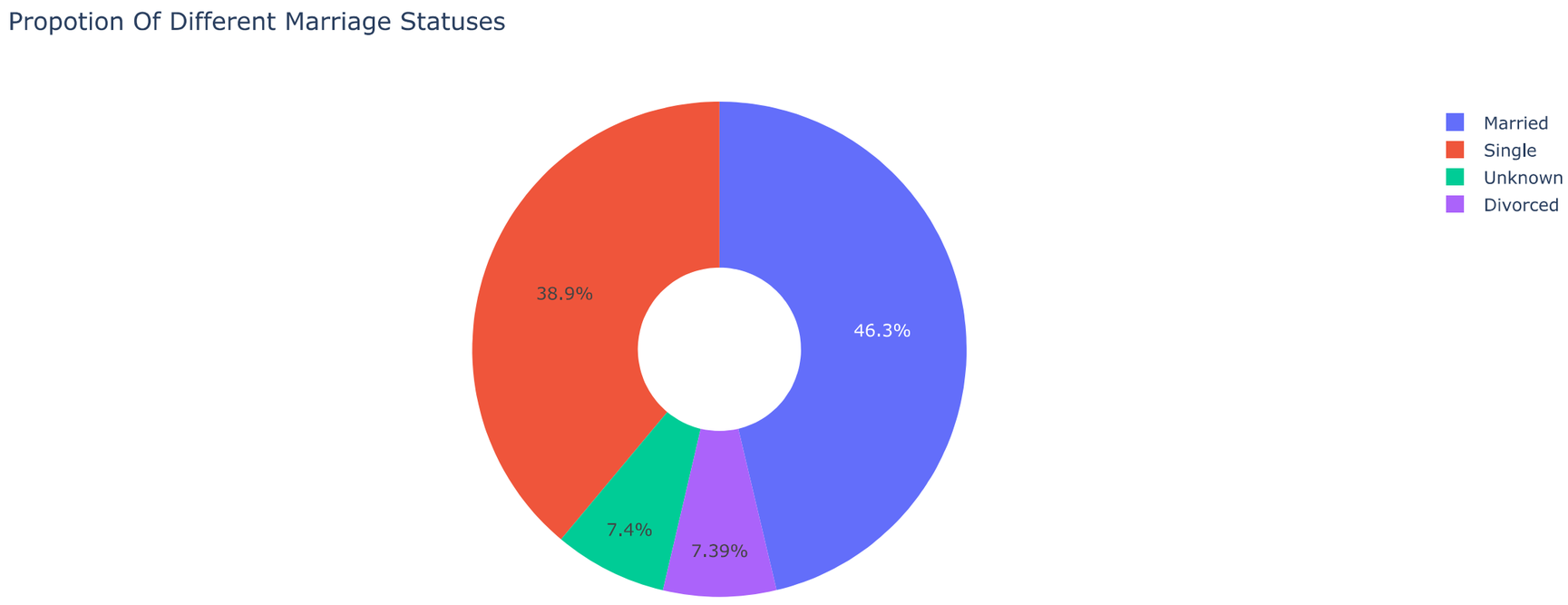
The distribution of Dependent counts is fairly normally distributed with a slight right skew.

In [6]:
```
ex.pie(c_data,names='Education_Level',title='Propotion Of Education Levels',hole=0.33)
```



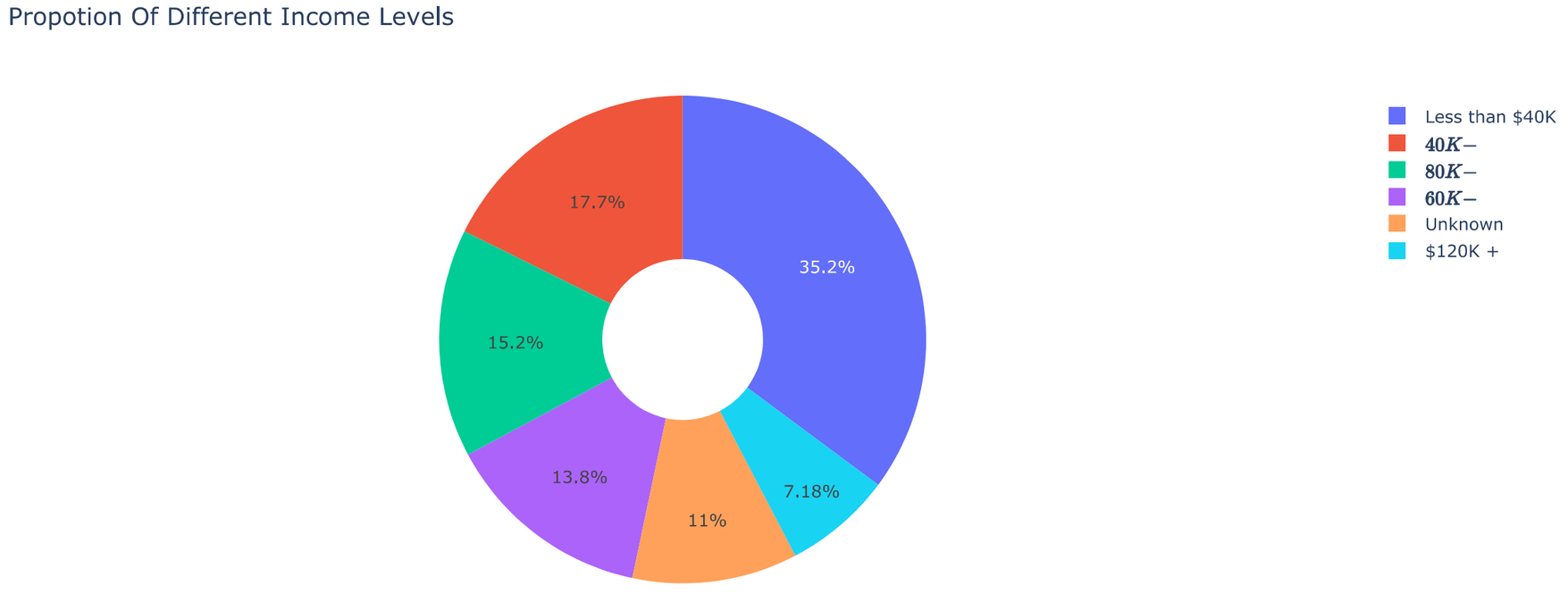
If most of the customers with unknown education status lack any education, we can state that more than 70% of the customers have a formal education level. About 35% have a higher level of education.

In [7]:
```
ex.pie(c_data,names='Marital_Status',title='Propotion Of Different Marriage Statuses',hole=0.33)
```

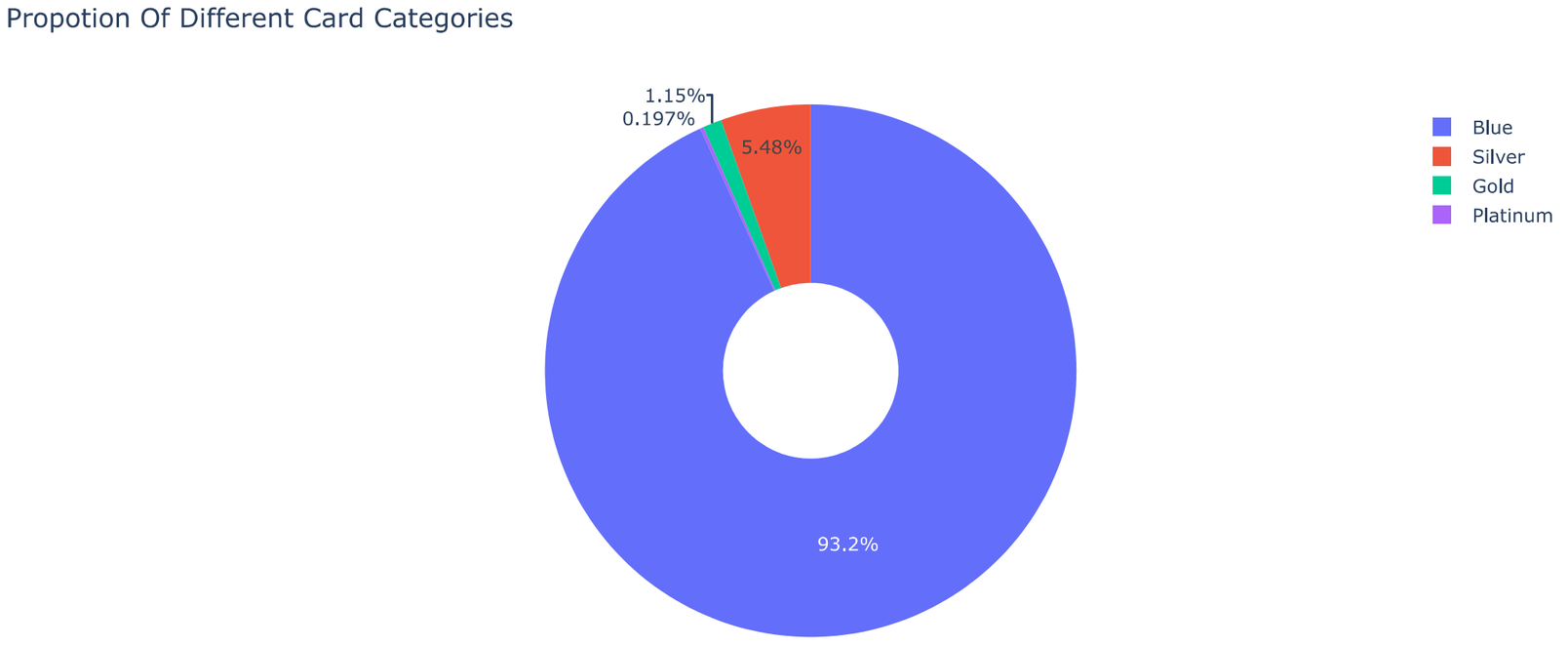


Almost half of the bank customers are married, and interestingly enough, almost the entire other half are single customers. only about 7% of the customers are divorced, which is surprising considering the worldwide divorce rate statistics.

In [8]:
```
ex.pie(c_data,names='Income_Category',title='Propotion Of Different Income Levels',hole=0.33)
```



In [9]:
```
ex.pie(c_data,names='Card_Category',title='Propotion Of Different Card Categories',hole=0.33)
```



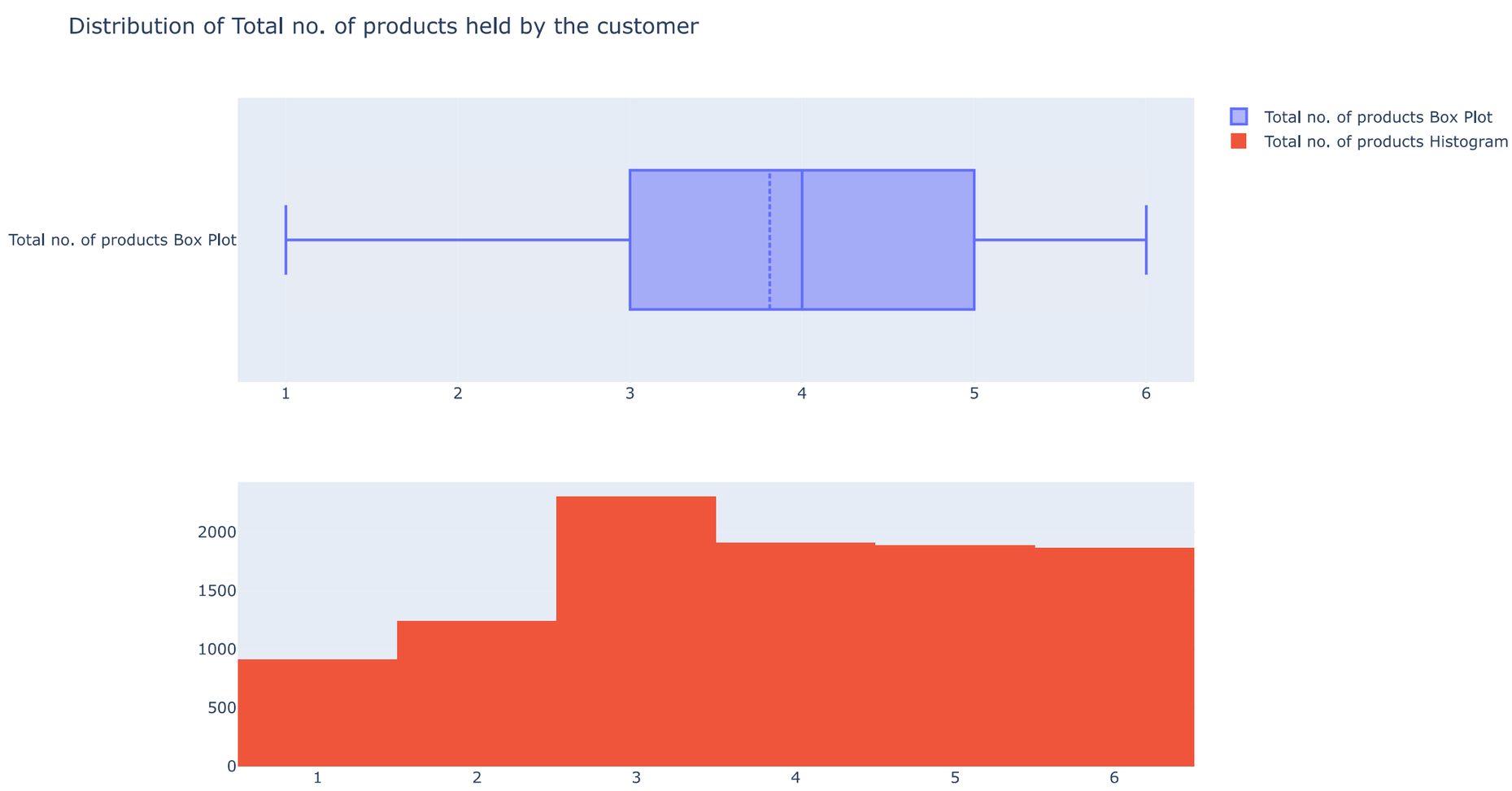
In [10]:
```
fig = make_subplots(rows=2, cols=1)

tr1=go.Box(x=c_data['Months_on_book'],name='Months on book Box Plot',boxmean=True)
tr2=go.Histogram(x=c_data['Months_on_book'],name='Months on book Histogram')

fig.add_trace(tr1,row=1,col=1)
fig.add_trace(tr2,row=2,col=1)

fig.update_layout(height=700, width=1200, title_text="Distribution of months the customer is part of the bank")
fig.show()
```

In [11]:
```
print('Kurtosis of Months on book features is : {}'.format(c_data['Months_on_book'].kurt()))
```

```
Kurtosis of Months on book features is : 0.40010012019986707
```

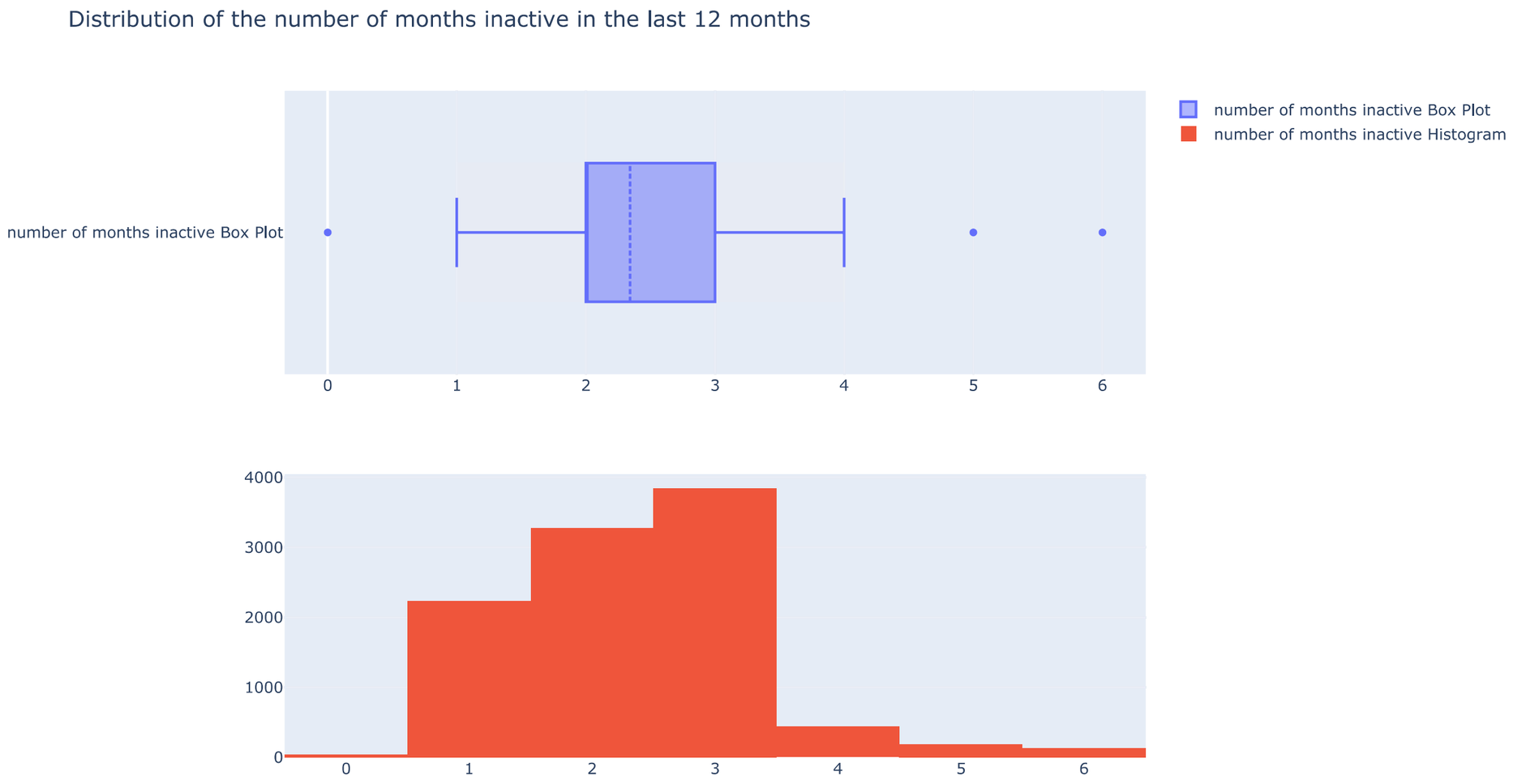
We have a low kurtosis value pointing to a very flat shaped distribution (as shown in the plots above as well), meaning we cannot assume normality of the feature.

In [12]:
```
fig = make_subplots(rows=2, cols=1)

tr1=go.Box(x=c_data['Total_Relationship_Count'],name='Total no. of products Box Plot',boxmean=True)
tr2=go.Histogram(x=c_data['Total_Relationship_Count'],name='Total no. of products Histogram')

fig.add_trace(tr1,row=1,col=1)
fig.add_trace(tr2,row=2,col=1)

fig.update_layout(height=700, width=1200, title_text="Distribution of Total no. of products held by the customer")
fig.show()
```

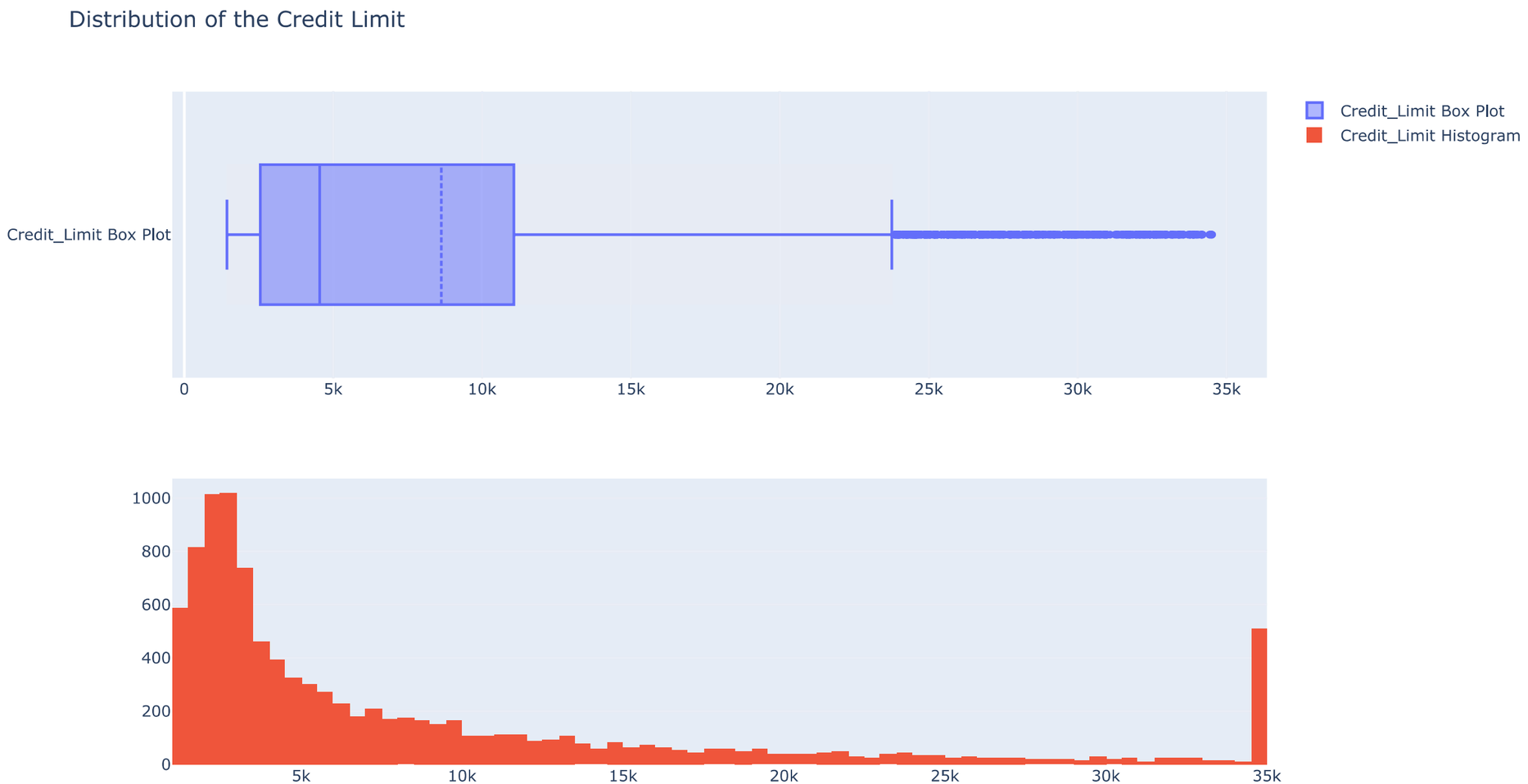


The distribution of the total number of products held by the customer seems closer to a uniform distribution and may appear useless as a predictor for churn status.

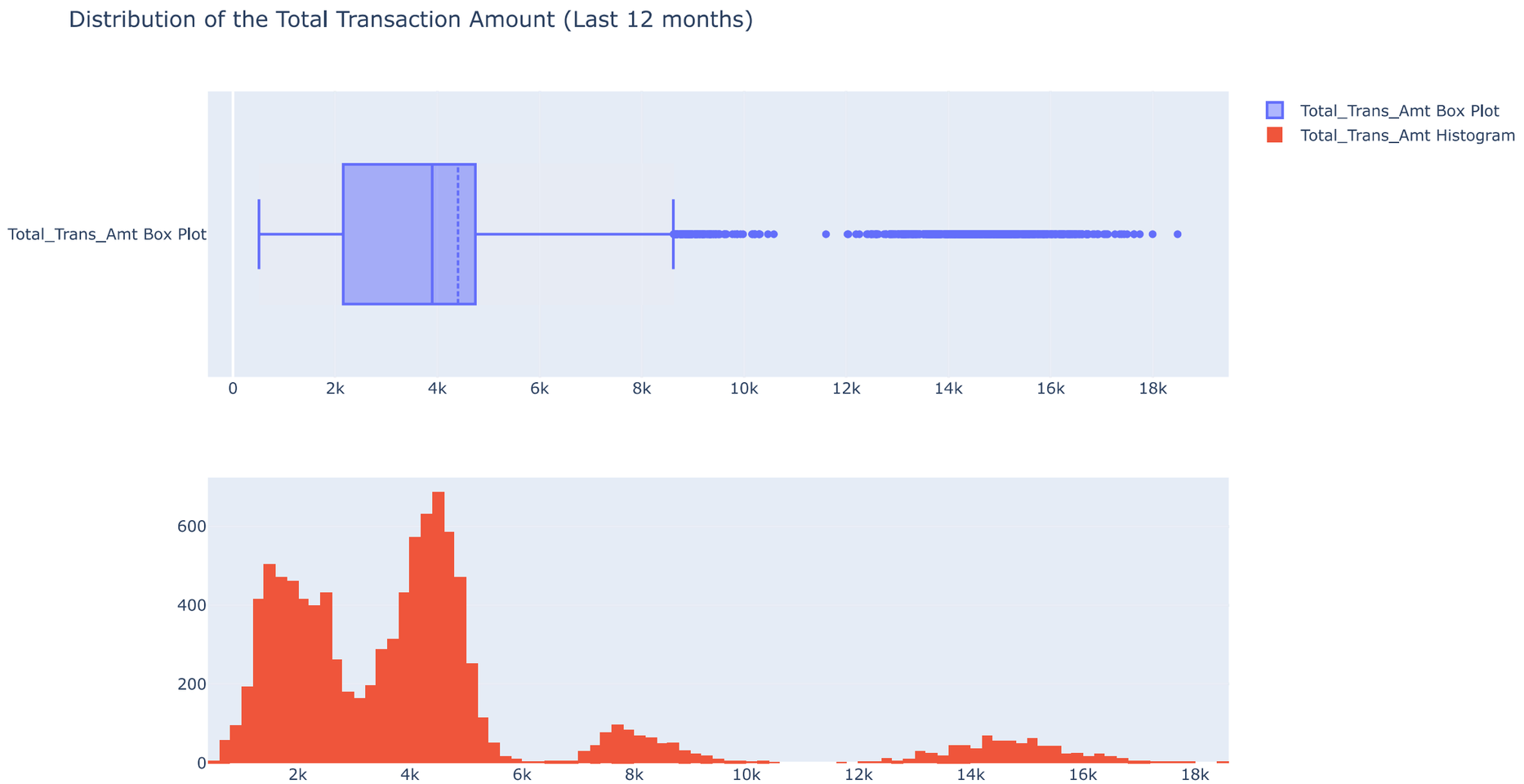
In [13]:
```
fig = make_subplots(rows=2, cols=1)

tr1=go.Box(x=c_data['Months_Inactive_12_mon'],name='number of months inactive Box Plot',boxmean=True)
tr2=go.Histogram(x=c_data['Months_Inactive_12_mon'],name='number of months inactive Histogram')

fig.add_trace(tr1,row=1,col=1)
fig.add_trace(tr2,row=2,col=1)

fig.update_layout(height=700, width=1200, title_text="Distribution of the number of months inactive in the last 12 months")
fig.show()
```



In [14]:
```
fig = make_subplots(rows=2, cols=1)

tr1=go.Box(x=c_data['Credit_Limit'],name='Credit_Limit Box Plot',boxmean=True)
tr2=go.Histogram(x=c_data['Credit_Limit'],name='Credit_Limit Histogram')

fig.add_trace(tr1,row=1,col=1)
fig.add_trace(tr2,row=2,col=1)

fig.update_layout(height=700, width=1200, title_text="Distribution of the Credit Limit")
fig.show()
```

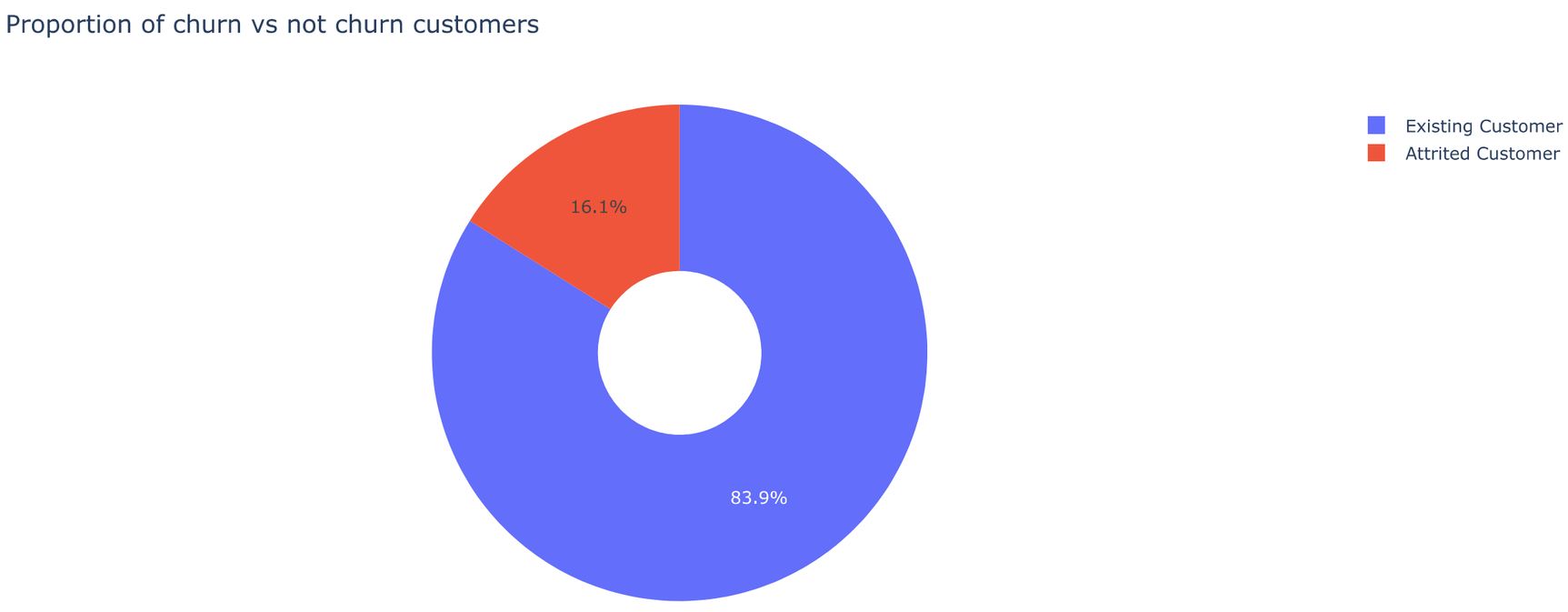


In [15]:
```
fig = make_subplots(rows=2, cols=1)

tr1=go.Box(x=c_data['Total_Trans_Amt'],name='Total_Trans_Amt Box Plot',boxmean=True)
tr2=go.Histogram(x=c_data['Total_Trans_Amt'],name='Total_Trans_Amt Histogram')

fig.add_trace(tr1,row=1,col=1)
fig.add_trace(tr2,row=2,col=1)

fig.update_layout(height=700, width=1200, title_text="Distribution of the Total Transaction Amount (Last 12 months)")
fig.show()
```



We see that the distribution of the total transactions (Last 12 months) displays a multimodal distribution, meaning we have some underlying groups in our data; it can be an interesting experiment to try and cluster the different groups and view the similarities between them and what describes best the different groups which create the different modes in our distribution.

In [16]:
```
ex.pie(c_data,names='Attrition_Flag',title='Proportion of churn vs not churn customers',hole=0.33)
```



As we can see, only 16% of the data samples represent churn customers; in the following steps, I will use SMOTE to upsample the churn samples to match them with the regular customer sample size to give the later selected models a better chance of catching on small details which will almost definitely be missed out with such a size difference.

In [17]:
```
c_data
```

Out[17]:

	CLIENTNUM	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	Marital_Status	Income_Category	Card_Category	Months_on_book	...	Months_Inactive_12_mon	Contacts_Count_12_mon	Credit_Limit	Total_R
0	768805383	Existing Customer	45	M	3	High School	Married	$60K-80K	Blue	39	...	1	3	12691.0	
1	818770008	Existing Customer	49	F	5	Graduate	Single	Less than $40K	Blue	44	...	1	2	8256.0	
2	713982108	Existing Customer	51	M	3	Graduate	Married	$80K-120K	Blue	36	...	1	0	3418.0	
3	769911858	Existing Customer	40	F	4	High School	Unknown	Less than $40K	Blue	34	...	4	1	3313.0	
4	709106358	Existing Customer	40	M	3	Uneducated	Married	$60K-80K	Blue	21	...	1	0	4716.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
10122	772366833	Existing Customer	50	M	2	Graduate	Single	$40K-60K	Blue	40	...	2	3	4003.0	
10123	710638233	Attrited Customer	41	M	2	Unknown	Divorced	$40K-60K	Blue	25	...	2	3	4277.0	
10124	716506083	Attrited Customer	44	F	1	High School	Married	Less than $40K	Blue	36	...	3	4	5409.0	
10125	717406983	Attrited Customer	30	M	2	Graduate	Unknown	$40K-60K	Blue	36	...	3	3	5281.0	
10126	714337233	Attrited Customer	43	F	2	Graduate	Married	Less than $40K	Silver	25	...	2	4	10388.0	

10127 rows × 21 columns

Data Preprocessing

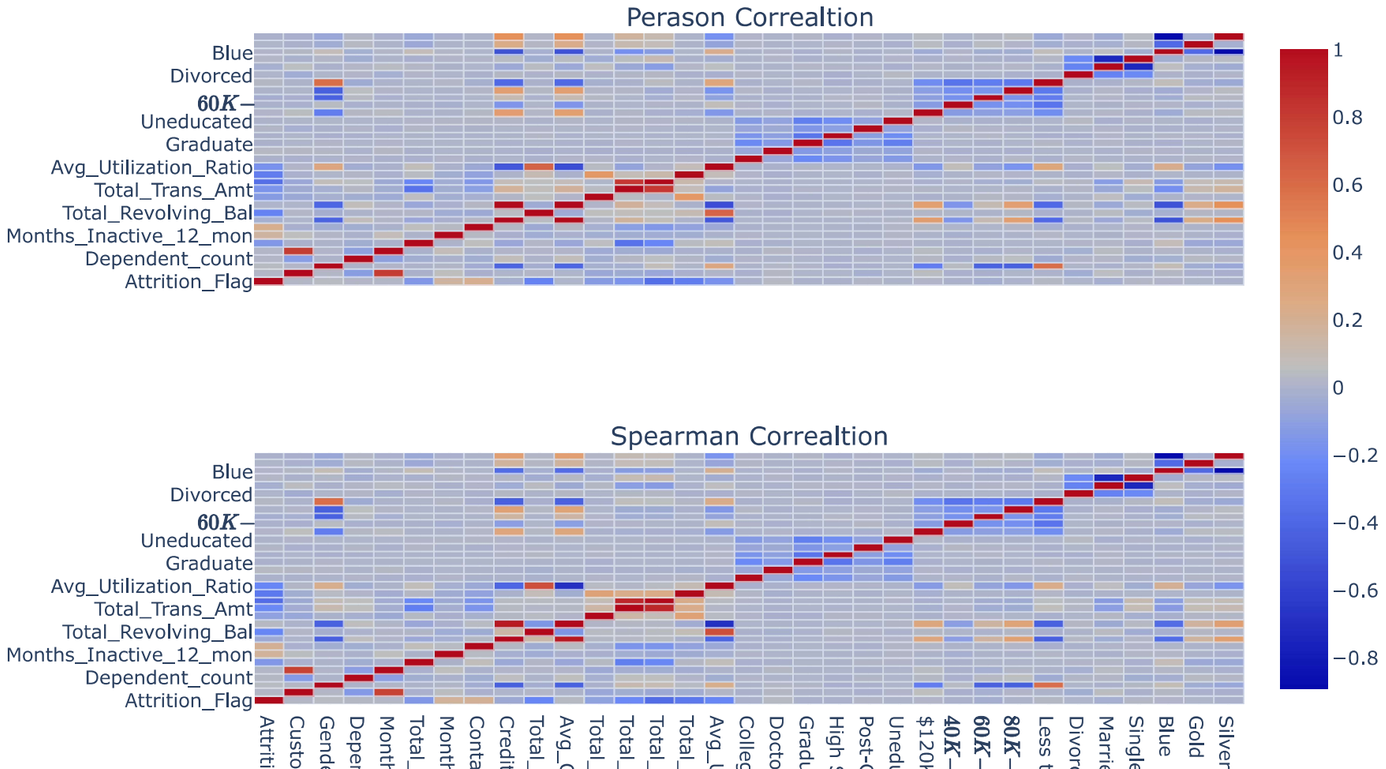
In [18]:
```
c_data.Attrition_Flag = c_data.Attrition_Flag.replace({'Attrited Customer':1,'Existing Customer':0})
c_data.Gender = c_data.Gender.replace({'F':1,'M':0})
c_data = pd.concat([c_data,pd.get_dummies(c_data['Education_Level']).drop(columns=['Unknown'])],axis=1)
c_data = pd.concat([c_data,pd.get_dummies(c_data['Income_Category']).drop(columns=['Unknown'])],axis=1)
c_data = pd.concat([c_data,pd.get_dummies(c_data['Marital_Status']).drop(columns=['Unknown'])],axis=1)
c_data = pd.concat([c_data,pd.get_dummies(c_data['Card_Category']).drop(columns=['Platinum'])],axis=1)
c_data.drop(columns = ['Education_Level','Income_Category','Marital_Status','Card_Category','CLIENTNUM'],inplace=True)
```

Here we use one hot encode for all the categorical features describing different statuses of a customer.

In [19]:
```
fig = make_subplots(rows=2, cols=1,shared_xaxes=True,subplot_titles=('Perason Correaltion',  'Spearman Correaltion'))
colorscale=     [[1.0              , "rgb(165,0,38)"],
                 [0.8888888888888888, "rgb(215,48,39)"],
                 [0.7777777777777778, "rgb(244,109,67)"],
                 [0.6666666666666666, "rgb(253,174,97)"],
                 [0.5555555555555556, "rgb(254,224,144)"],
                 [0.4444444444444444, "rgb(224,243,248)"],
                 [0.3333333333333333, "rgb(171,217,233)"],
                 [0.2222222222222222, "rgb(116,173,209)"],
                 [0.1111111111111111, "rgb(69,117,180)"],
                 [0.0               , "rgb(49,54,149)"]]
```

```
s_val =c_data.corr('pearson')
s_idx = s_val.index
s_col = s_val.columns
s_val = s_val.values
fig.add_trace(
    go.Heatmap(x=s_col,y=s_idx,z=s_val,name='pearson',showscale=False,xgap=0.7,ygap=0.7,colorscale=colorscale),
    row=1, col=1
)


s_val =c_data.corr('spearman')
s_idx = s_val.index
s_col = s_val.columns
s_val = s_val.values
fig.add_trace(
    go.Heatmap(x=s_col,y=s_idx,z=s_val,xgap=0.7,ygap=0.7,colorscale=colorscale),
    row=2, col=1
)
fig.update_layout(
    hoverlabel=dict(
        bgcolor="white",
        font_size=16,
        font_family="Rockwell"
    )
)
fig.update_layout(height=700, width=900, title_text="Numeric Correaltions")
fig.show()
```

Numeric Correaltions



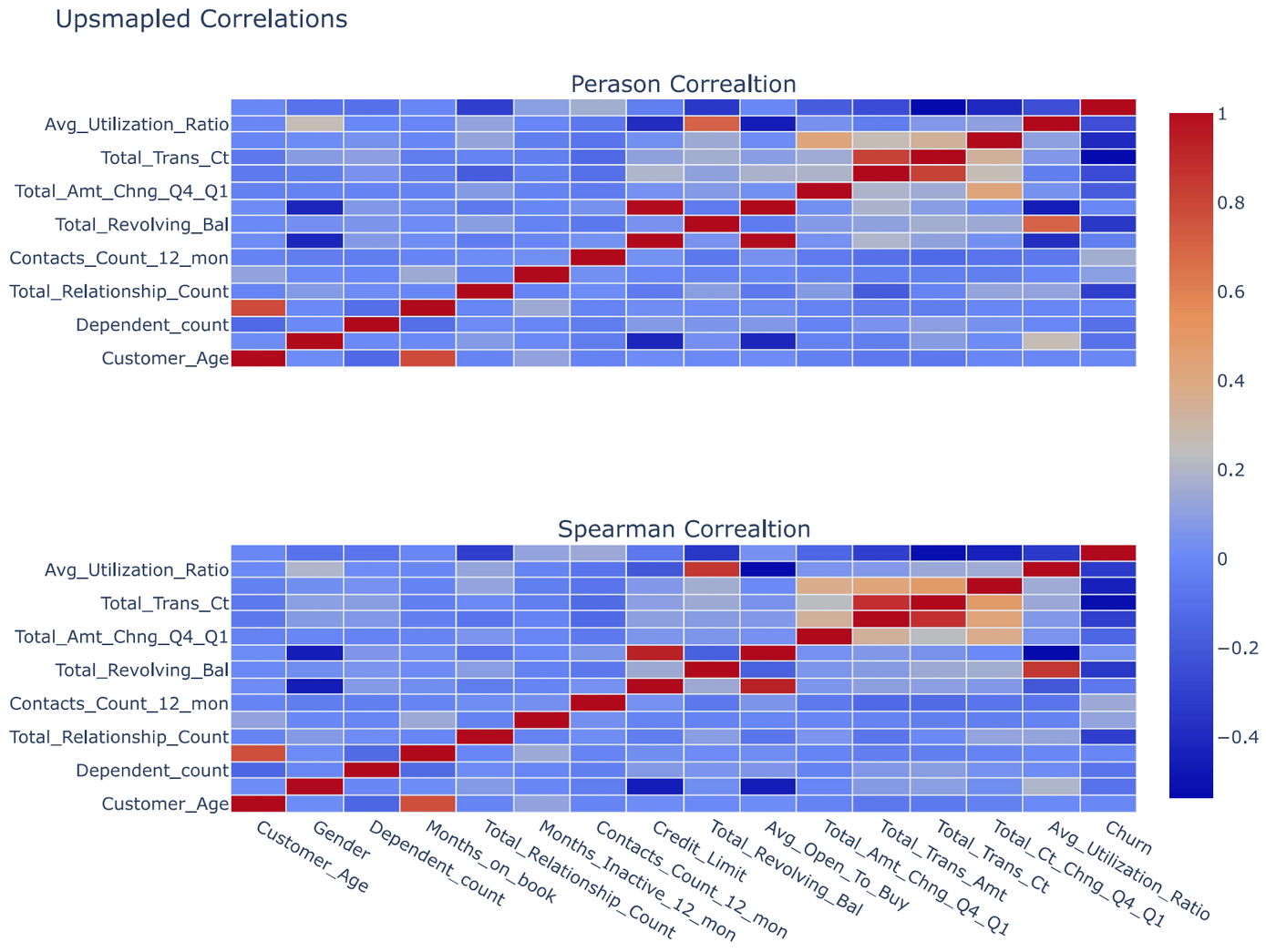
Data Upsampling Using SMOTE

In [20]:
```
oversample = SMOTE()
X, y = oversample.fit_resample(c_data[c_data.columns[1:]], c_data[c_data.columns[0]])
usampled_df = X.assign(Churn = y)
```

In [21]:
```
ohe_data =usampled_df[usampled_df.columns[15:-1]].copy()

usampled_df = usampled_df.drop(columns=usampled_df.columns[15:-1])
```

In [22]:
```
fig = make_subplots(rows=2, cols=1,shared_xaxes=True,subplot_titles=('Perason Correaltion',  'Spearman Correaltion'))
colorscale=     [[1.0              , "rgb(165,0,38)"],
                [0.8888888888888888, "rgb(215,48,39)"],
                [0.7777777777777778, "rgb(244,109,67)"],
```

```
                [0.6666666666666666, "rgb(253,174,97)"],
                [0.5555555555555556, "rgb(254,224,144)"],
                [0.4444444444444444, "rgb(224,243,248)"],
                [0.3333333333333333, "rgb(171,217,233)"],
                [0.2222222222222222, "rgb(116,173,209)"],
                [0.1111111111111111, "rgb(69,117,180)"],
                [0.0               , "rgb(49,54,149)"]]

s_val =usampled_df.corr('pearson')
s_idx = s_val.index
s_col = s_val.columns
s_val = s_val.values
fig.add_trace(
    go.Heatmap(x=s_col,y=s_idx,z=s_val,name='pearson',showscale=False,xgap=1,ygap=1,colorscale=colorscale),
    row=1, col=1
)


s_val =usampled_df.corr('spearman')
s_idx = s_val.index
s_col = s_val.columns
s_val = s_val.values
fig.add_trace(
    go.Heatmap(x=s_col,y=s_idx,z=s_val,xgap=1,ygap=1,colorscale=colorscale),
    row=2, col=1
)
fig.update_layout(
    hoverlabel=dict(
        bgcolor="white",
        font_size=16,
        font_family="Rockwell"
    )
)
fig.update_layout(height=700, width=900, title_text="Upsmapled Correlations")
fig.show()
```

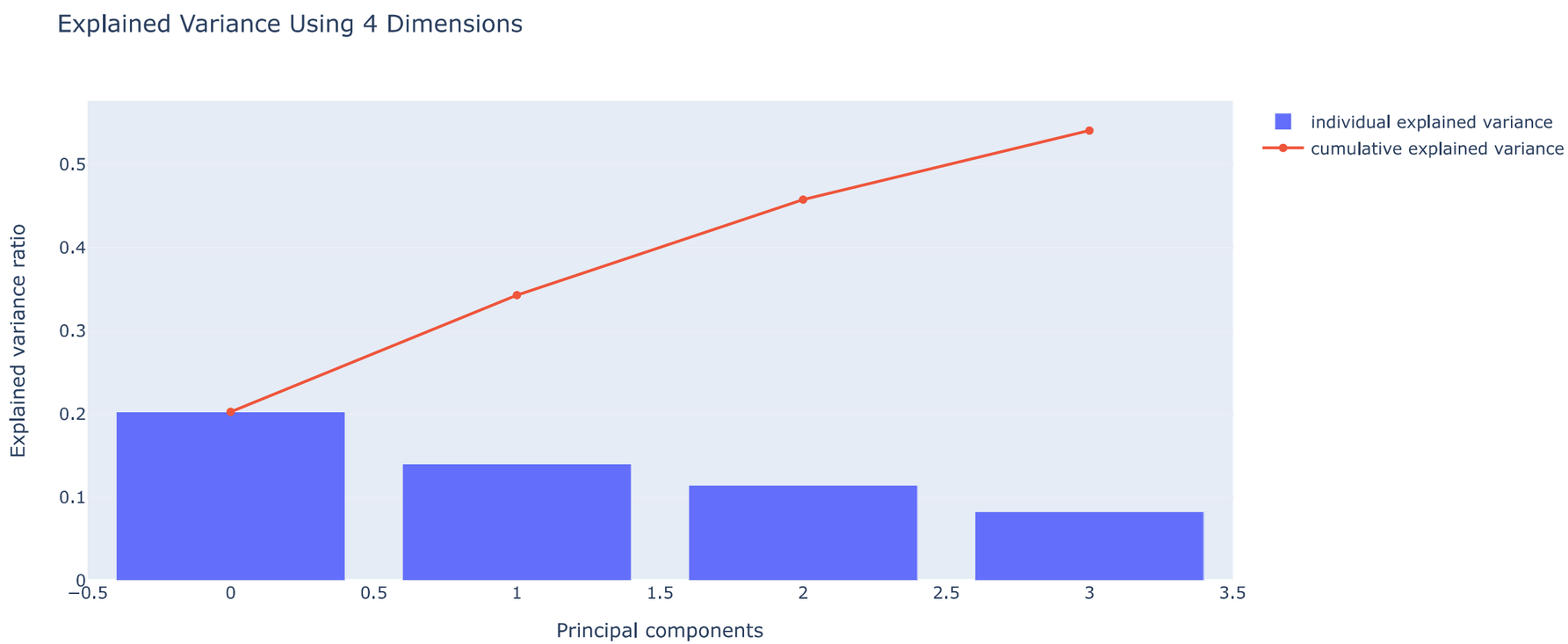


Principal Component Analysis Of One Hot Encoded Data

We will use principal component analysis to reduce the dimensionality of the one-hot encoded categorical variables losing some of the variances, but simultaneously, using a couple of principal components instead of tens of one-hot encoded features will help me construct a better model.

In [23]:
```
N_COMPONENTS = 4

pca_model = PCA(n_components = N_COMPONENTS )

pc_matrix = pca_model.fit_transform(ohe_data)

evr = pca_model.explained_variance_ratio_
total_var = evr.sum() * 100
cumsum_evr = np.cumsum(evr)
```

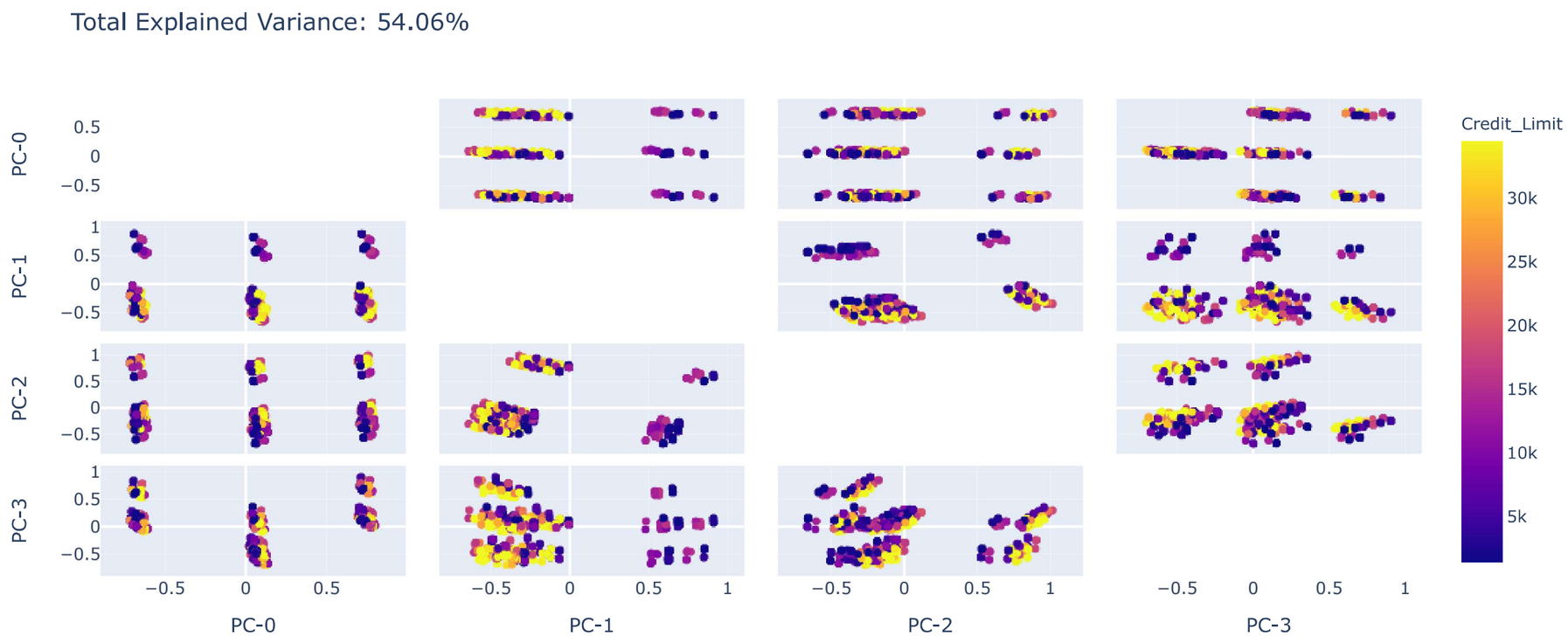
```
trace1 = {
    "name": "individual explained variance",
    "type": "bar",
    'y':evr}
trace2 = {
    "name": "cumulative explained variance",
    "type": "scatter",
     'y':cumsum_evr}
data = [trace1, trace2]
layout = {
    "xaxis": {"title": "Principal components"},
    "yaxis": {"title": "Explained variance ratio"},
  }
fig = go.Figure(data=data, layout=layout)
fig.update_layout(     title='Explained Variance Using {} Dimensions'.format(N_COMPONENTS))
fig.show()
```



In [24]:
```
usampled_df_with_pcs = pd.concat([usampled_df,pd.DataFrame(pc_matrix,columns=['PC-{}'.format(i) for i in range(0,N_COMPONENTS)])],axis=1)
```

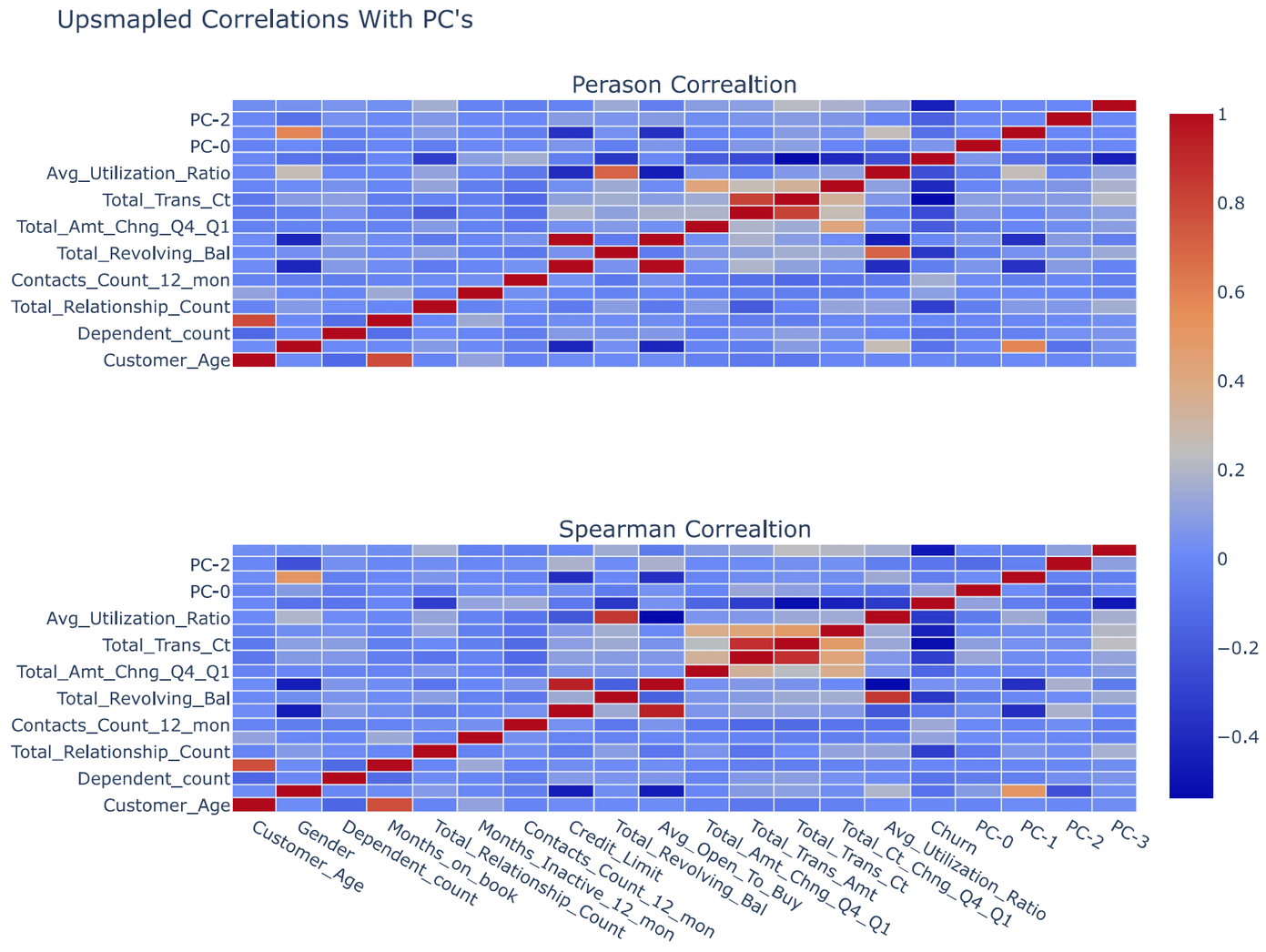
In [25]:
```
fig = ex.scatter_matrix(
    usampled_df_with_pcs[['PC-{}'.format(i) for i in range(0,N_COMPONENTS)]].values,
    color=usampled_df_with_pcs.Credit_Limit,
    dimensions=range(N_COMPONENTS),
    labels={str(i):'PC-{}'.format(i) for i in range(0,N_COMPONENTS)},
    title=f'Total Explained Variance: {total_var:.2f}%')
```

```
fig.update_traces(diagonal_visible=False)
fig.update_layout(
    coloraxis_colorbar=dict(
        title="Credit_Limit",
    ),
)
fig.show()
```



In [26]:
```
fig = make_subplots(rows=2, cols=1,shared_xaxes=True,subplot_titles=('Perason Correaltion',  'Spearman Correaltion'))


s_val =usampled_df_with_pcs.corr('pearson')
s_idx = s_val.index
s_col = s_val.columns
s_val = s_val.values
fig.add_trace(
    go.Heatmap(x=s_col,y=s_idx,z=s_val,name='pearson',showscale=False,xgap=1,ygap=1,colorscale=colorscale),
    row=1, col=1
)


s_val =usampled_df_with_pcs.corr('spearman')
s_idx = s_val.index
s_col = s_val.columns
s_val = s_val.values
fig.add_trace(
```

```
    go.Heatmap(x=s_col,y=s_idx,z=s_val,xgap=1,ygap=1,colorscale=colorscale),
    row=2, col=1
)
fig.update_layout(
    hoverlabel=dict(
        bgcolor="white",
        font_size=16,
        font_family="Rockwell"
    )
)
fig.update_layout(height=700, width=900, title_text="Upsmapled Correlations With PC\'s")
fig.show()
```



Model Selection And Evaluation

In [27]:
```
X_features = ['Total_Trans_Ct','PC-3','PC-1','PC-0','PC-2','Total_Ct_Chng_Q4_Q1','Total_Relationship_Count']

X = usampled_df_with_pcs[X_features]
y = usampled_df_with_pcs['Churn']
```

In [28]:
```
train_x,test_x,train_y,test_y = train_test_split(X,y,random_state=42)
```

Cross Validation

In [29]:
```
rf_pipe = Pipeline(steps =[ ('scale',StandardScaler()), ("RF",RandomForestClassifier(random_state=42)) ])
ada_pipe = Pipeline(steps =[ ('scale',StandardScaler()), ("RF",AdaBoostClassifier(random_state=42,learning_rate=0.7)) ])
svm_pipe = Pipeline(steps =[ ('scale',StandardScaler()), ("RF",SVC(random_state=42,kernel='rbf')) ])


f1_cross_val_scores = cross_val_score(rf_pipe,train_x,train_y,cv=5,scoring='f1')
ada_f1_cross_val_scores=cross_val_score(ada_pipe,train_x,train_y,cv=5,scoring='f1')
svm_f1_cross_val_scores=cross_val_score(svm_pipe,train_x,train_y,cv=5,scoring='f1')
```

In [30]:
```
fig = make_subplots(rows=3, cols=1,shared_xaxes=True,subplot_titles=('Random Forest Cross Val Scores',
                                                                     'Adaboost Cross Val Scores',
                                                                    'SVM Cross Val Scores'))

fig.add_trace(
    go.Scatter(x=list(range(0,len(f1_cross_val_scores))),y=f1_cross_val_scores,name='Random Forest'),
    row=1, col=1
)
fig.add_trace(
    go.Scatter(x=list(range(0,len(ada_f1_cross_val_scores))),y=ada_f1_cross_val_scores,name='Adaboost'),
    row=2, col=1
)
fig.add_trace(
    go.Scatter(x=list(range(0,len(svm_f1_cross_val_scores))),y=svm_f1_cross_val_scores,name='SVM'),
    row=3, col=1
)

fig.update_layout(height=700, width=900, title_text="Different Model 5 Fold Cross Validation")
fig.update_yaxes(title_text="F1 Score")
fig.update_xaxes(title_text="Fold #")

fig.show()
```

Model Evaluation

In [31]:
```
rf_pipe.fit(train_x,train_y)
rf_prediction = rf_pipe.predict(test_x)

ada_pipe.fit(train_x,train_y)
ada_prediction = ada_pipe.predict(test_x)

svm_pipe.fit(train_x,train_y)
svm_prediction = svm_pipe.predict(test_x)
```

In [32]:
```
fig = go.Figure(data=[go.Table(header=dict(values=['<b>Model<b>', '<b>F1 Score On Test Data<b>'],
                                           line_color='darkslategray',
    fill_color='whitesmoke',
```

```
    align=['center','center'],
    font=dict(color='black', size=18),
    height=40),

                cells=dict(values=[['<b>Random Forest<b>', '<b>AdaBoost<b>','<b>SVM<b>'], [np.round(f1(rf_prediction,test_y),2),
                                                                                           np.round(f1(ada_prediction,test_y),2),
                                                                                           np.round(f1(svm_prediction,test_y),2)]]))
                    ])

fig.update_layout(title='Model Results On Test Data')
fig.show()
```

Model Results On Test Data

Model	F1 Score On Test Data
Random Forest	0.91
AdaBoost	0.89
SVM	0.89

Model Evaluation On Original Data (Before Upsampling)

In [33]:
```
ohe_data =c_data[c_data.columns[16:]].copy()
pc_matrix = pca_model.fit_transform(ohe_data)
original_df_with_pcs = pd.concat([c_data,pd.DataFrame(pc_matrix,columns=['PC-{}'.format(i) for i in range(0,N_COMPONENTS)])],axis=1)

unsampled_data_prediction_RF = rf_pipe.predict(original_df_with_pcs[X_features])
unsampled_data_prediction_ADA = ada_pipe.predict(original_df_with_pcs[X_features])
unsampled_data_prediction_SVM = svm_pipe.predict(original_df_with_pcs[X_features])
```

In [34]:
```
fig = go.Figure(data=[go.Table(header=dict(values=['<b>Model<b>', '<b>F1 Score On Original Data (Before Upsampling)<b>'],
                                           line_color='darkslategray',
    fill_color='whitesmoke',
```

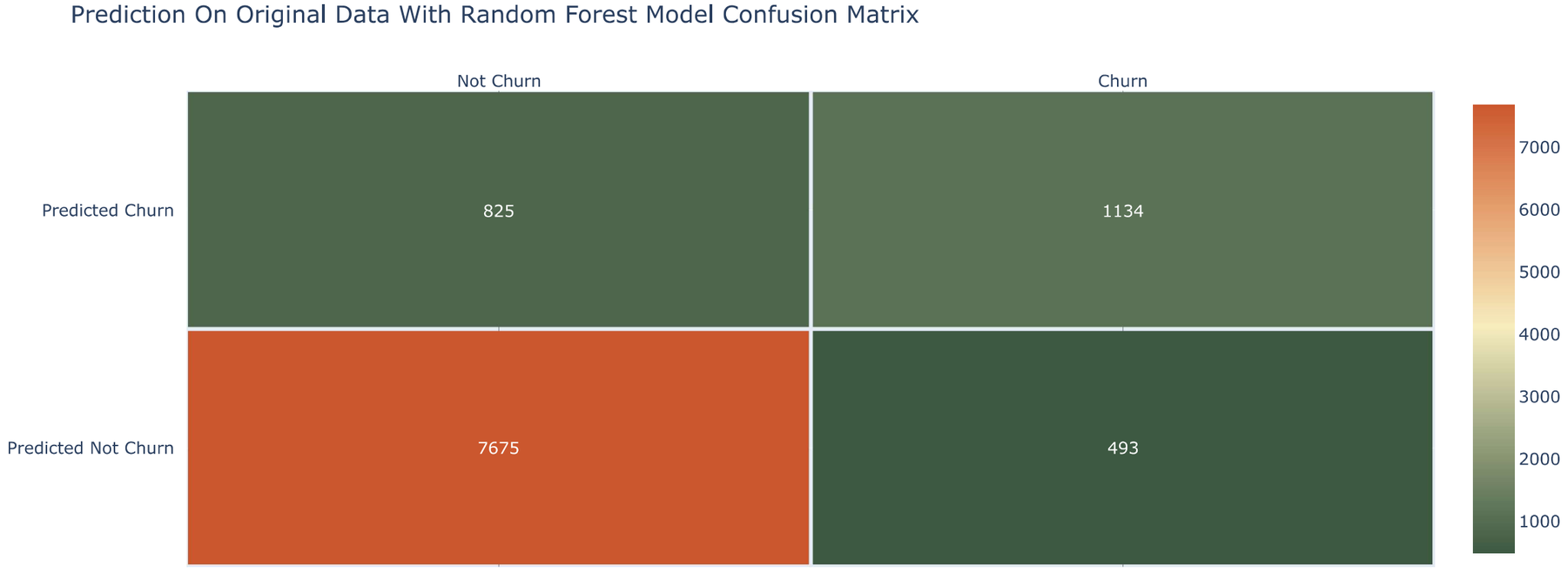
```
    align=['center','center'],
    font=dict(color='black', size=18),
    height=40),

                cells=dict(values=[['<b>Random Forest<b>', '<b>AdaBoost<b>','<b>SVM<b>'], [np.round(f1(unsampled_data_prediction_RF,original_df_with_pcs['Attrition_Flag']),2),
                                                                                           np.round(f1(unsampled_data_prediction_ADA,original_df_with_pcs['Attrition_Flag']),2),
                                                                                           np.round(f1(unsampled_data_prediction_SVM,original_df_with_pcs['Attrition_Flag']),2)]]))
                    ])

fig.update_layout(title='Model Result On Original Data (Without Upsampling)')
fig.show()
```

Model Result On Original Data (Without Upsampling)

Model	F1 Score On Original Data (Before Upsampling)
Random Forest	0.63
AdaBoost	0.52
SVM	0.55

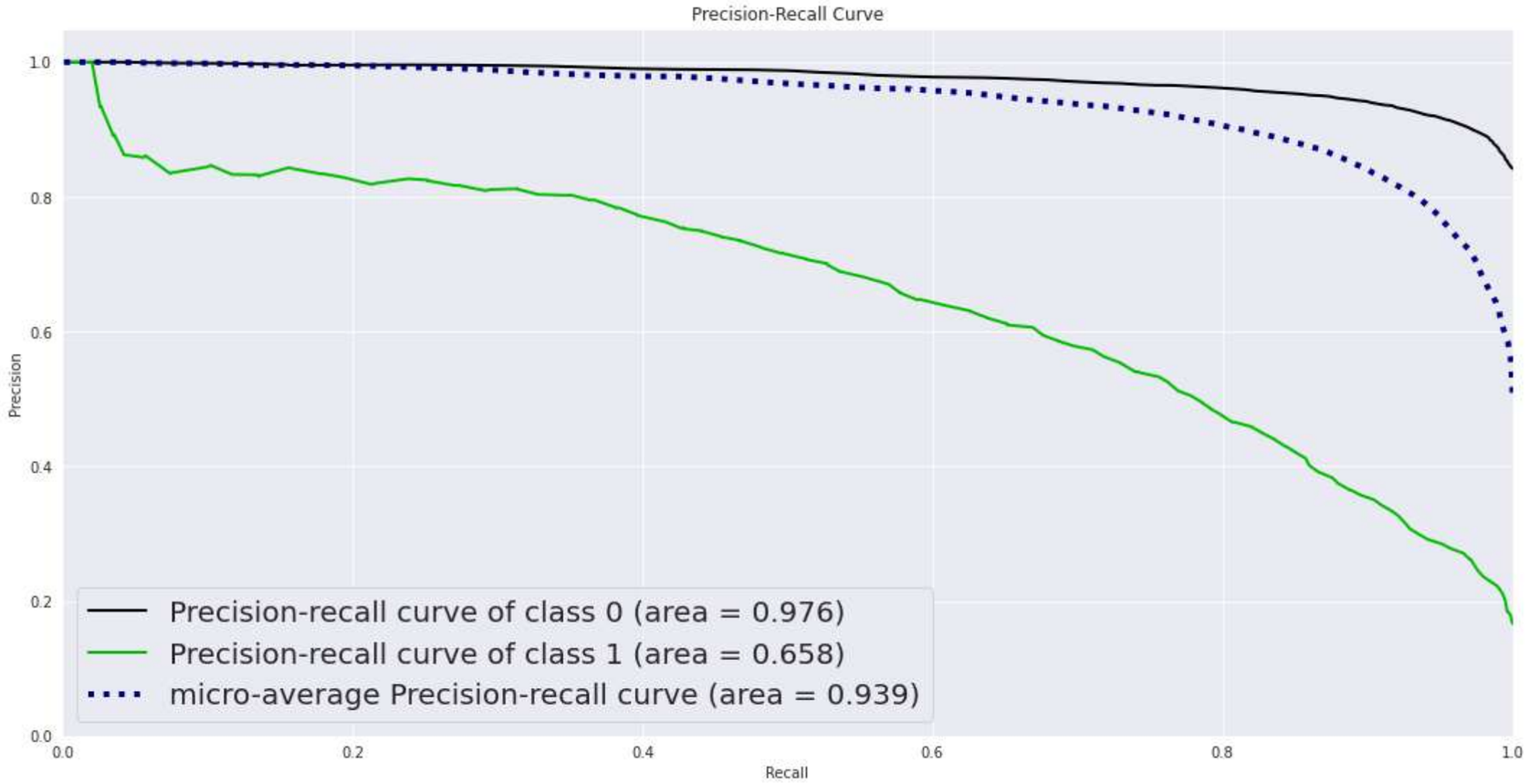
Results

In [35]:
```
z=confusion_matrix(unsampled_data_prediction_RF,original_df_with_pcs['Attrition_Flag'])
fig = ff.create_annotated_heatmap(z, x=['Not Churn','Churn'], y=['Predicted Not Churn','Predicted Churn'], colorscale='Fall',xgap=3,ygap=3)
fig['data'][0]['showscale'] = True
fig.update_layout(title='Prediction On Original Data With Random Forest Model Confusion Matrix')
fig.show()
```



In [36]:
```
unsampled_data_prediction_RF = rf_pipe.predict_proba(original_df_with_pcs[X_features])
skplt.metrics.plot_precision_recall(original_df_with_pcs['Attrition_Flag'], unsampled_data_prediction_RF)
plt.legend(prop={'size': 20})
```

Out[36]:
```
<matplotlib.legend.Legend at 0x7c2e6e075b90>
```


Precision-Recall Curve
Precision
Recall
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
Precision-recall curve of class 0 (area = 0.976)
Precision-recall curve of class 1 (area = 0.658)
micro-average Precision-recall curve (area = 0.939)