Holiday Package Prediction

Libraries

In [1]:

```
import numpy as np
import pandas as pd

import missingno as msno
import plotly.express as px
import plotly.graph_objs as go
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.preprocessing import LabelEncoder
from sklearn.impute import KNNImputer

from sklearn.feature_selection import f_classif
from sklearn.preprocessing import MinMaxScaler
from sklearn.feature_selection import SelectKBest, chi2

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.metrics import accuracy_score, classification_report

from sklearn.ensemble import RandomForestClassifier
import xgboost as xgb
```

Dataset

In [2]:

```
holiday = pd.read_csv(r'C:\Users\Tanmayee\OneDrive\Documents\Personal\Other\AI\Adarsh\Tr
holiday
```

Out[2]:

	CustomerID	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	C
0	200000	1	41.0	Self Enquiry	3	6.0	Salaried	F
1	200001	0	49.0	Company Invited	1	14.0	Salaried	
2	200002	1	37.0	Self Enquiry	1	8.0	Free Lancer	
3	200003	0	33.0	Company Invited	1	9.0	Salaried	F
4	200004	0	NaN	Self Enquiry	1	8.0	Small Business	
...	...	...	...	...	...	...	...	
4883	204883	1	49.0	Self Enquiry	3	9.0	Small Business	
4884	204884	1	28.0	Company Invited	1	31.0	Salaried	
4885	204885	1	52.0	Self Enquiry	3	17.0	Salaried	F
4886	204886	1	19.0	Self Enquiry	3	16.0	Small Business	
4887	204887	1	36.0	Self Enquiry	1	14.0	Salaried	

4888 rows × 20 columns

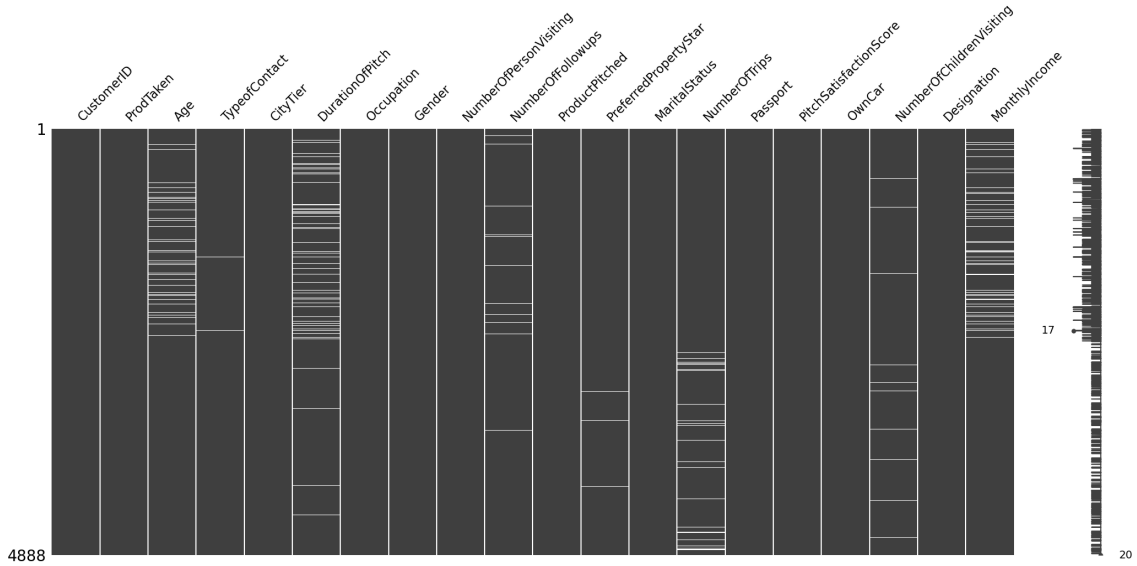
Checking Null Values

In [3]:

```
holiday.isnull().sum()
msno.matrix(holiday)
```

Out[3]:

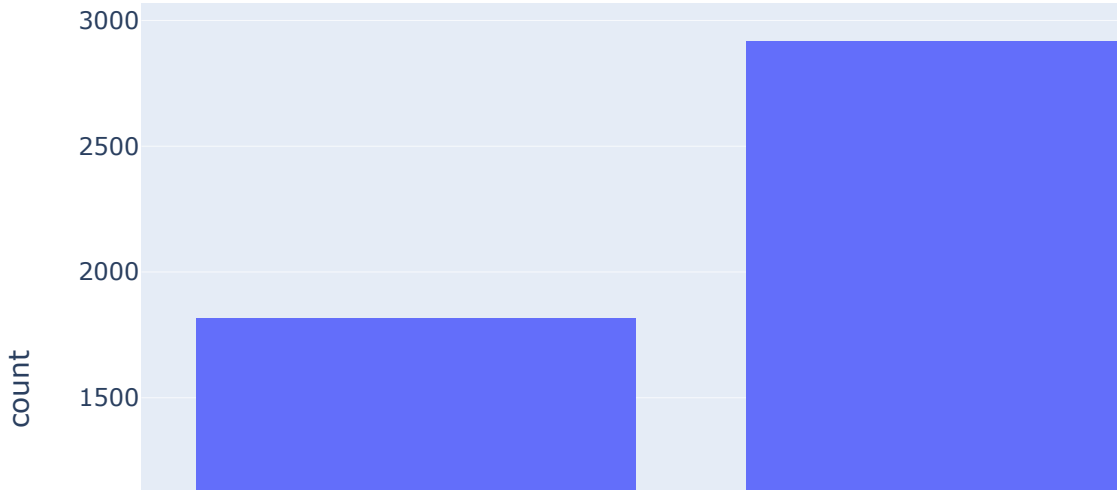
<AxesSubplot:>



Exploratory Data Analysis

In [4]:

```
fig = px.histogram(holiday,x='Gender')
fig.show()
```

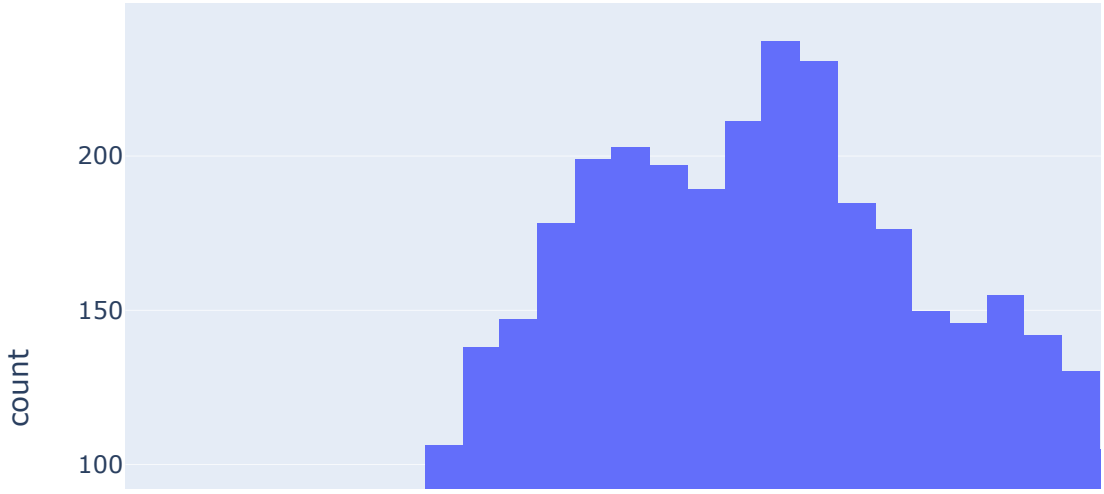


In [5]:

```
fig = px.histogram(holiday,x='TypeofContact')
fig.show()
```

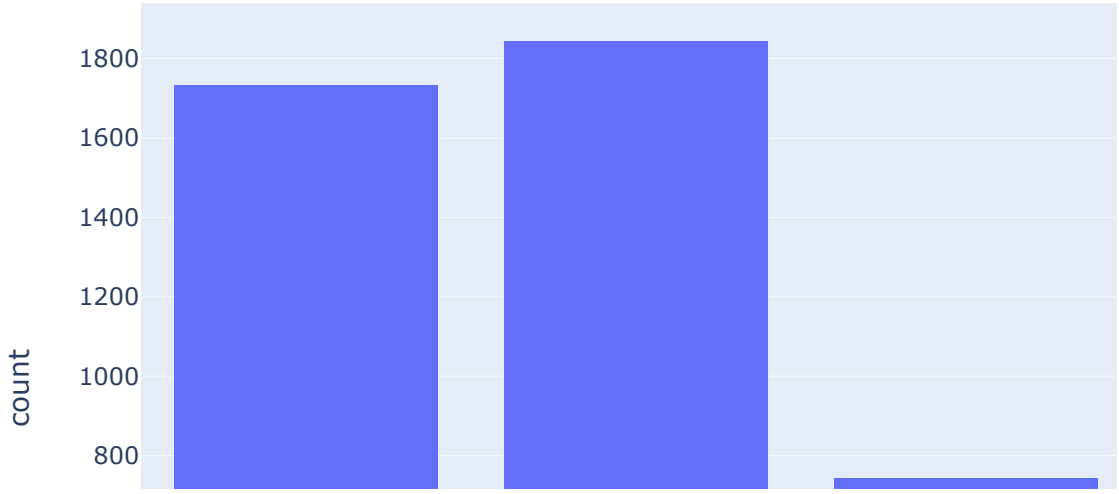
In [6]:

```
fig = px.histogram(holiday,x='Age')
fig.show()
```



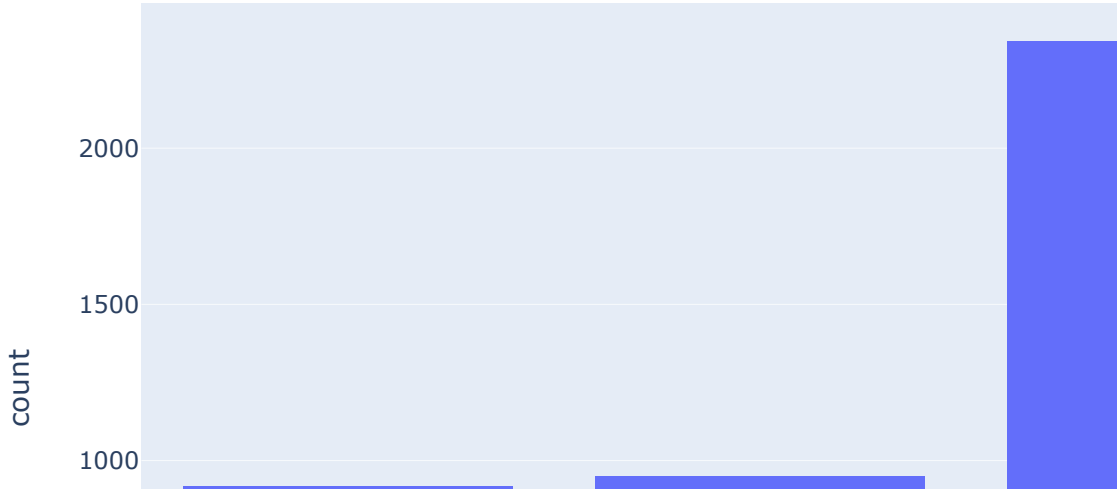
In [7]:

```
fig = px.histogram(holiday,x='ProductPitched')
fig.show()
```



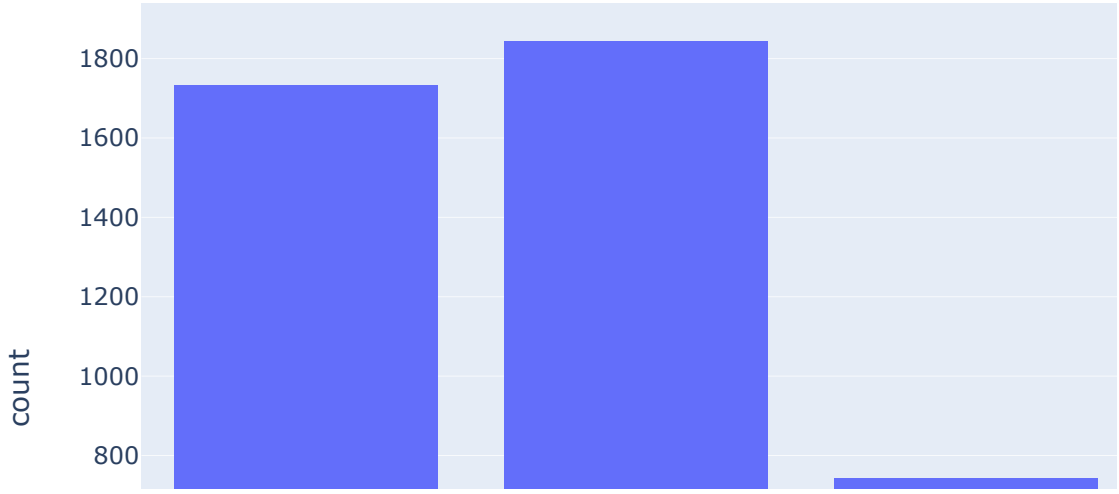
In [8]:

```
fig = px.histogram(holiday,x='MaritalStatus')
fig.show()
```



In [9]:

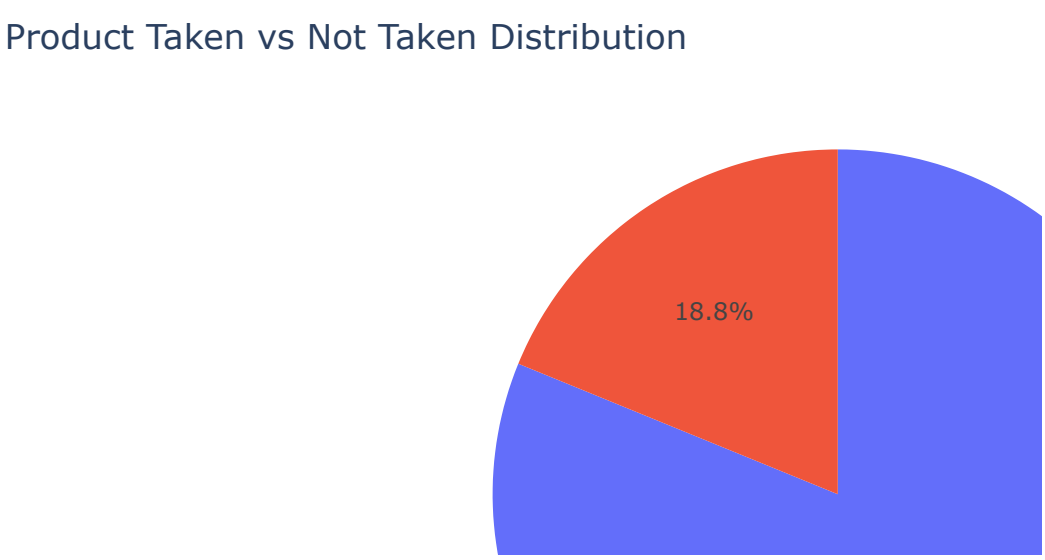
```
fig = px.histogram(holiday,x='Designation')
fig.show()
```



Using the df DataFrame and the data and labels supplied in the names and parameters, make a pie chart. The title of the chart is set using the title parameter. By adjusting the different options offered by the px.pie() method, you may change how the chart looks. Finally, you can use the fig.show() function to display the chart.

In [10]:

```
fig = px.pie(holiday, values=holiday['ProdTaken'].value_counts(), names=['Not Taken', 'T
             title='Product Taken vs Not Taken Distribution')
fig.show()
```



Code initially compiles a list of column names for all of the object- or int64-type columns in holiday. The unique values in each column are then extracted by iterating over each name of the column in this list. The output is formatted using the f-string in a manner that is similar to the original code.

In [11]:

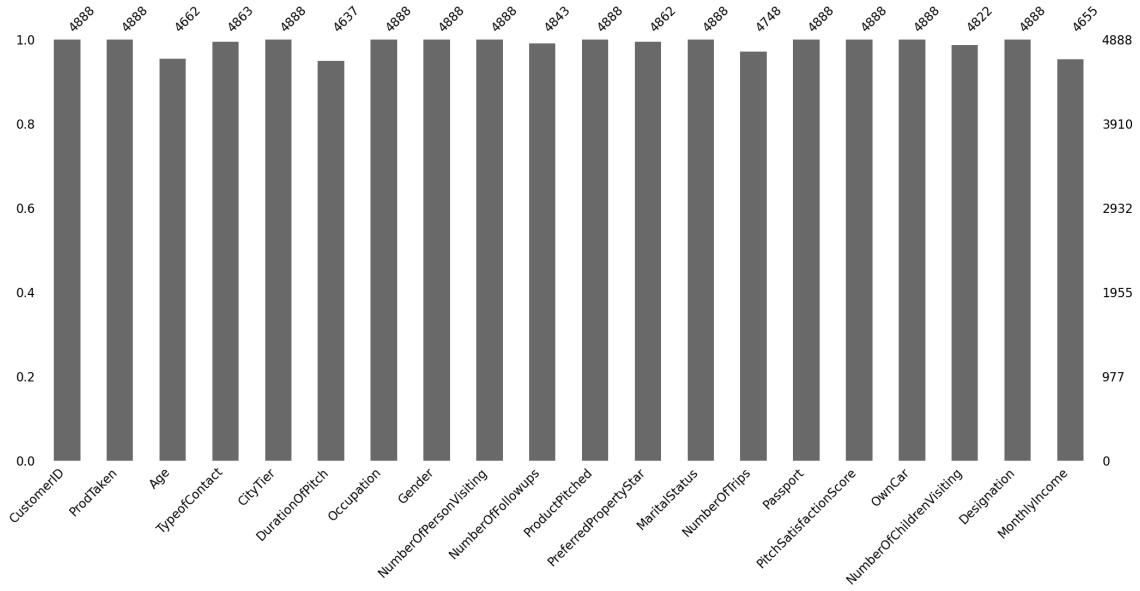
```
categorical_cols = holiday.select_dtypes(include=['object', 'int64']).columns.tolist()

for col in categorical_cols:
    unique_vals = holiday[col].unique()
    #print(f"{col} {'-' * (50 - len(col))} {unique_vals}")
```

In [12]:

```
holiday.replace('Fe Male','Female', inplace = True)
```

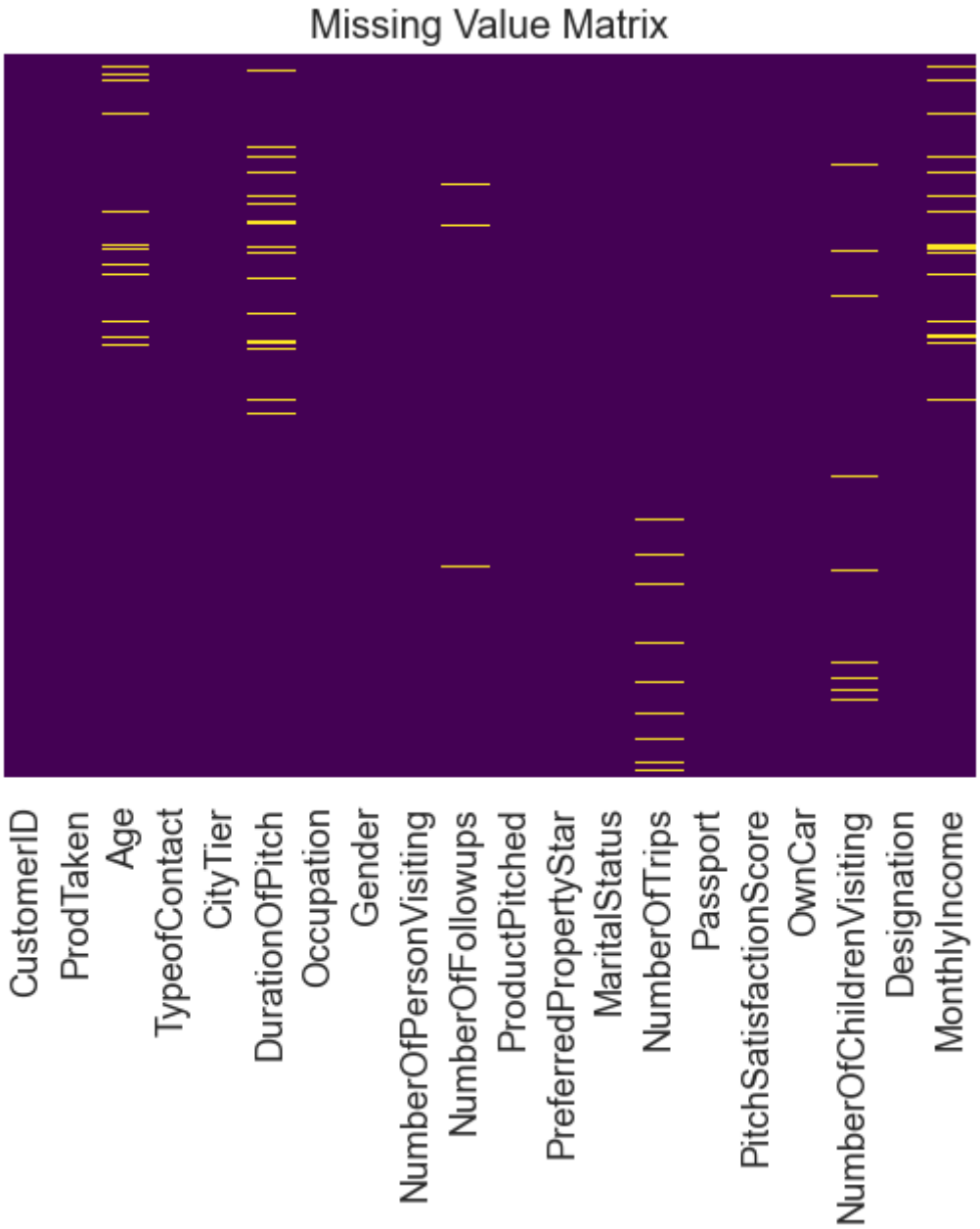
In [13]:

```
# Generate the missing value matrix plot
msno_bar = msno.bar(holiday)
```



In [14]:

```
sns.set(style='whitegrid', font_scale=1.2)
msno_matrix = sns.heatmap(holiday.isnull(), cmap='viridis',
                          cbar=False, yticklabels=False)

# Set the chart title
msno_matrix.set_title('Missing Value Matrix')

# Display the chart
plt.show()
```

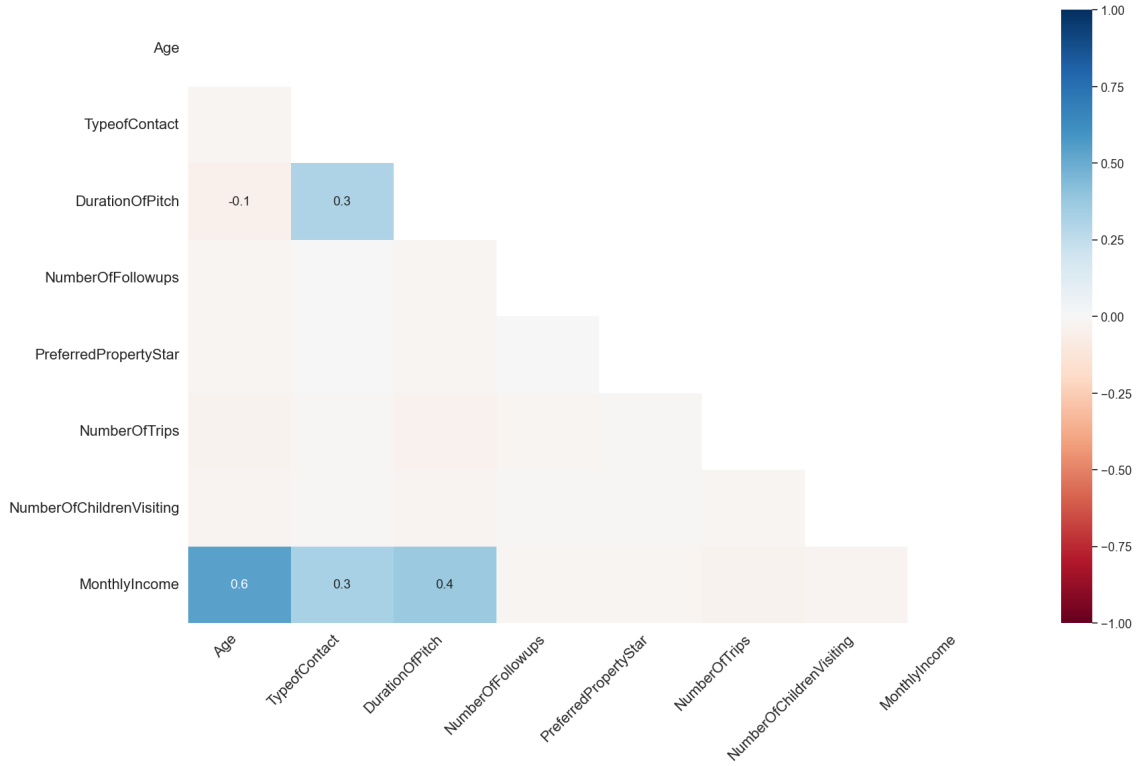


In [15]:

```
msno.heatmap(holiday)
```

Out[15]:

<AxesSubplot:>



Pre-Processing

In [16]:

```
holiday = holiday.drop(['CustomerID'],axis=1)
```

In [17]:

```
# Define the categorical columns
categorical = [col for col in holiday.columns if holiday[col].dtype == 'object' or holic

# Define the numerical columns
numerical = [col for col in holiday.columns if holiday[col].dtype in ['int64', 'float64'
```

In [18]:

```
# Impute missing values for numerical columns using KNNImputer
imputer = KNNImputer(n_neighbors=1)
holiday[numerical] = imputer.fit_transform(holiday[numerical])

# Impute missing values for categorical columns using mode
for col in categorical:
    holiday[col].fillna(holiday[col].mode()[0], inplace=True)

# Check for any remaining missing values
holiday.isnull()
```

Out[18]:

	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	Num
0	False	False	False	False	False	False	False	
1	False	False	False	False	False	False	False	
2	False	False	False	False	False	False	False	
3	False	False	False	False	False	False	False	
4	False	False	False	False	False	False	False	
...	...	...	...	...	...	...	...	
4883	False	False	False	False	False	False	False	
4884	False	False	False	False	False	False	False	
4885	False	False	False	False	False	False	False	
4886	False	False	False	False	False	False	False	
4887	False	False	False	False	False	False	False	

4888 rows × 19 columns

In [19]:

```
y = holiday.ProdTaken
```

In [20]:

```
cat_df = holiday[categorical].apply(LabelEncoder().fit_transform)
cat_df.head()
```

Out[20]:

	ProdTaken	TypeofContact	CityTier	Occupation	Gender	NumberOfPersonVisiting	Numbe
0	1	1	2	2	0	2	
1	0	0	0	2	1	2	
2	1	1	0	0	1	2	
3	0	0	0	2	0	1	
4	0	1	0	3	1	1	

In [21]:

```
scaler = MinMaxScaler(feature_range=(0,1))
normalized_numerical = pd.DataFrame(scaler.fit_transform(holiday[numerical]), columns=nu
normalized_numerical
```

Out[21]:

	Age	DurationOfPitch	MonthlyIncome
0	0.534884	0.008197	0.204683
1	0.720930	0.073770	0.195848
2	0.441860	0.024590	0.164725
3	0.348837	0.032787	0.173110
4	0.209302	0.024590	0.178832
...	...	...	...
4883	0.720930	0.032787	0.261840
4884	0.232558	0.213115	0.206925
4885	0.790698	0.098361	0.315527
4886	0.023256	0.090164	0.197475
4887	0.418605	0.073770	0.235887

4888 rows × 3 columns

In [22]:

```
bestfeatures_num = SelectKBest(score_func=f_classif, k=3)
fit_num = bestfeatures_num.fit(normalized_numerical,y)
dfscores_num = pd.DataFrame(fit_num.scores_)
dfcolumns_num = pd.DataFrame(normalized_numerical.columns)
#concat two dataframes for better visualization
featureScores_num = pd.concat([dfcolumns_num,dfscores_num],axis=1)
featureScores_num.columns = ['Specs','Score'] #naming the dataframe columns
print(featureScores_num.nlargest(3,'Score')) #print best features
featureScores_num.nlargest(3,'Score').to_csv("numerical.csv")
```

```
             Specs       Score
0              Age  102.943931
2    MonthlyIncome   77.318846
1  DurationOfPitch   28.181641
```

In [23]:

```
df = holiday[['Passport', 'ProductPitched', 'MaritalStatus']]
df.head()
```

Out[23]:

	Passport	ProductPitched	MaritalStatus
0	1	Deluxe	Single
1	0	Deluxe	Divorced
2	1	Basic	Single
3	1	Basic	Divorced
4	0	Basic	Divorced

In [24]:

```
dog_df = pd.get_dummies(df)
dog_df.head()
```

Out[24]:

	Passport	ProductPitched_Basic	ProductPitched_Deluxe	ProductPitched_King	ProductPitc
0	1	0	1	0	
1	0	0	1	0	
2	1	1	0	0	
3	1	1	0	0	
4	0	1	0	0	

In [25]:

```
num_dog_df = pd.concat([normalized_numerical, dog_df], axis=1)
df = pd.concat([y, num_dog_df], axis=1)
df.head()
```

Out[25]:

	ProdTaken	Age	DurationOfPitch	MonthlyIncome	Passport	ProductPitched_Basic	P
0	1	0.534884	0.008197	0.204683	1	0	
1	0	0.720930	0.073770	0.195848	0	0	
2	1	0.441860	0.024590	0.164725	1	1	
3	0	0.348837	0.032787	0.173110	1	1	
4	0	0.209302	0.024590	0.178832	0	1	

In [26]:

```
X = df.drop(['ProdTaken'], axis = 1)
y = df['ProdTaken']
```

Train and Test

In [27]:

```
X_train,X_test,y_train,y_test=train_test_split(X ,y,test_size=0.2,random_state=1)
```

Logistic Regression

In [28]:

```
reg = 0.01
LogRegModel = LogisticRegression(C=1/reg, solver = 'liblinear').fit(X_train, y_train)
```

Accuracy, Precision, Recall and F1 Score

In [29]:

```
accuracy_logi = LogRegModel.score(X_test, y_test)
preds = LogRegModel.predict(X_test)
recall_logi = metrics.recall_score(y_test, preds)
precision_logi = metrics.precision_score(y_test, preds)
f1_score_logi = metrics.f1_score(y_test, preds)
```

Printing Metrics

In [30]:

```
print('accuracy:', accuracy_logi)
print('recall:', recall_logi)
print('precision:', precision_logi)
print('f1-score:', f1_score_logi)
```

```
accuracy: 0.8282208588957055
recall: 0.2205128205128205
precision: 0.7288135593220338
f1-score: 0.3385826771653543
```

XGBoost

In [31]:

```
# define the XGBClassifier model
xgb_model = xgb.XGBClassifier().fit(X_train, y_train)

# predict the labels on test data
y_pred = xgb_model.predict(X_test)

# calculate accuracy score
accuracy = accuracy_score(y_test, y_pred)
```

Accuracy, Precision, Recall and F1 Score

In [32]:

```
accuracy_xgb = xgb_model.score(X_test, y_test)
preds = xgb_model.predict(X_test)
recall_xgb = metrics.recall_score(y_test, preds)
precision_xgb = metrics.precision_score(y_test, preds)
f1_score_xgb = metrics.f1_score(y_test, preds)
```

Printing Accuracy, Precision, Recall and F1 Score

In [33]:

```
print('accuracy:', accuracy_xgb)
print('recall:', recall_xgb)
print('precision:', precision_xgb)
print('f1-score:', f1_score_xgb)
```

```
accuracy: 0.8793456032719836
recall: 0.5282051282051282
precision: 0.7984496124031008
f1-score: 0.6358024691358025
```

Random Forest Tree

In [34]:

```
# Create a random forest classifier with 100 trees
rf = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the model on the training data
rf.fit(X_train, y_train)

# Predict on the test data
preds = rf.predict(X_test)
```

Accuracy, Precision, Recall and F1 Score

In [35]:

```
accuracy_rf = rf.score(X_test, y_test)
preds = rf.predict(X_test)
recall_rf = metrics.recall_score(y_test, preds)
precision_rf = metrics.precision_score(y_test, preds)
f1_score_rf = metrics.f1_score(y_test, preds)
```

Printing Accuracy, Precision, Recall and F1 Score

In [36]:

```
print('accuracy:', accuracy_rf)
print('recall:', recall_rf)
print('precision:', precision_rf)
print('f1-score:', f1_score_rf)
```

```
accuracy: 0.8977505112474438
recall: 0.6153846153846154
precision: 0.8275862068965517
f1-score: 0.7058823529411765
```

Storing data into Dataframes for plotting it

In [37]:

```
models = ['Logistic Regression', 'XGBoost', 'Random Forest']
accuracy = [accuracy_logi, accuracy_xgb, accuracy_rf]
recall = [recall_logi, recall_xgb, recall_rf]
precision = [precision_logi, precision_xgb, precision_rf]
f1_score = [f1_score_logi, f1_score_xgb, f1_score_rf]
```

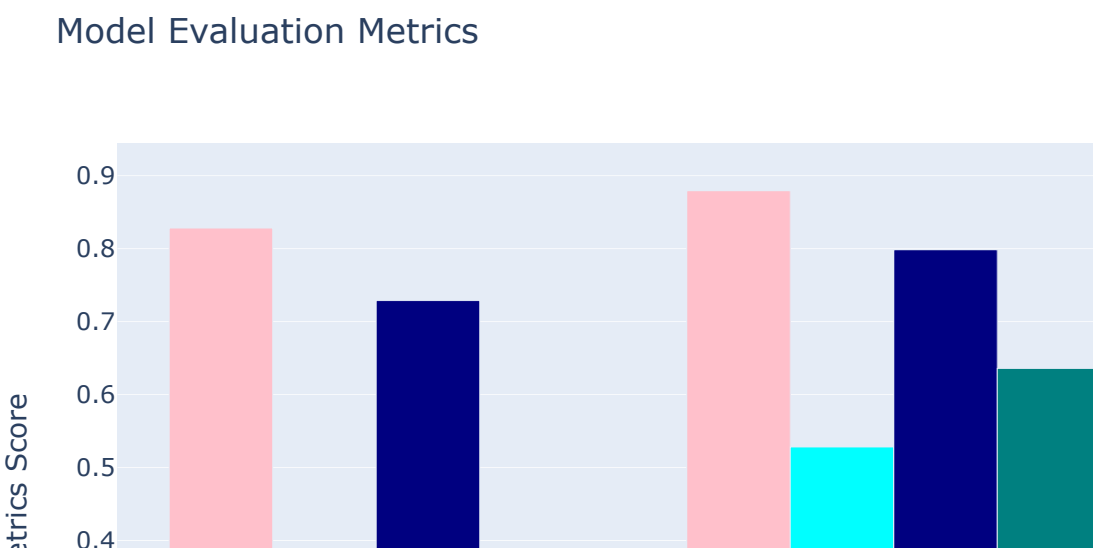
In [38]:

```
# Create trace for accuracy
trace1 = go.Bar(x=models, y=accuracy, name='Accuracy', marker_color='pink')

# Create trace for recall
trace2 = go.Bar(x=models, y=recall, name='Recall', marker_color='cyan')

# Create trace for precision
trace3 = go.Bar(x=models, y=precision, name='Precision', marker_color='navy')

# Create trace for F1-score
trace4 = go.Bar(x=models, y=f1_score, name='F1-score', marker_color='teal')
```

Plotting the graph

In [39]:

```
# Create layout for the graph
layout = go.Layout(title='Model Evaluation Metrics',
                   xaxis=dict(title='Models'),
                   yaxis=dict(title='Metrics Score'),
                   barmode='group')

# Create figure object
fig = go.Figure(data=[trace1, trace2, trace3, trace4], layout=layout)

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



In [ ]:

```

```