In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
sns.set()


# ML Algorithms

from sklearn.model_selection import train_test_split, cross_validate, cross_val_score, cross_val_predict, StratifiedKFold
from imblearn.over_sampling import SMOTE
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC


# DL Models

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout

# Hide warnings for cleaner production
import warnings
warnings.filterwarnings('ignore')

# Evaluation Metrics Libraries

from sklearn.metrics import accuracy_score, ConfusionMatrixDisplay, f1_score, recall_score, precision_score, roc_auc_score, co
```

In [3]:
```
df = pd.read_csv("Shoppers_Behaviour_and_Revenue.csv")
df
```

Out[3]:

	Administrative	Administrative_Duration	Informational	Informational_Duration	ProductRelated	ProductRelated_Duration	BounceR
0	0	0.0	0	0.0	1	0.000000	0.200
1	0	0.0	0	0.0	2	64.000000	0.000
2	0	0.0	0	0.0	1	0.000000	0.200
3	0	0.0	0	0.0	2	2.666667	0.050
4	0	0.0	0	0.0	10	627.500000	0.020
...	...	...	...	...	...	...	
12325	3	145.0	0	0.0	53	1783.791667	0.007
12326	0	0.0	0	0.0	5	465.750000	0.000
12327	0	0.0	0	0.0	6	184.250000	0.083
12328	4	75.0	0	0.0	15	346.000000	0.000
12329	0	0.0	0	0.0	3	21.250000	0.000

12330 rows × 18 columns

Data Preprocessing:

In [4]:
```
df.shape
```

Out[4]:
```
(12330, 18)
```

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12330 entries, 0 to 12329
Data columns (total 18 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   Administrative           12330 non-null  int64  
 1   Administrative_Duration  12330 non-null  float64
 2   Informational            12330 non-null  int64  
 3   Informational_Duration   12330 non-null  float64
 4   ProductRelated           12330 non-null  int64  
 5   ProductRelated_Duration  12330 non-null  float64
 6   BounceRates              12330 non-null  float64
 7   ExitRates                12330 non-null  float64
 8   PageValues               12330 non-null  float64
 9   SpecialDay               12330 non-null  float64
 10  Month                    12330 non-null  object 
 11  OperatingSystems         12330 non-null  int64  
 12  Browser                  12330 non-null  int64  
 13  Region                   12330 non-null  int64  
 14  TrafficType              12330 non-null  int64  
 15  VisitorType              12330 non-null  object 
 16  Weekend                  12330 non-null  bool   
 17  Revenue                  12330 non-null  bool   
dtypes: bool(2), float64(7), int64(7), object(2)
memory usage: 1.5+ MB
```

In [6]:
```
df.isna().sum()
```

Out[6]:
```
Administrative             0
Administrative_Duration    0
Informational              0
Informational_Duration     0
ProductRelated             0
ProductRelated_Duration    0
BounceRates                0
ExitRates                  0
PageValues                 0
SpecialDay                 0
Month                      0
OperatingSystems           0
Browser                    0
Region                     0
TrafficType                0
VisitorType                0
Weekend                    0
Revenue                    0
dtype: int64
```

In [7]:
```
df.duplicated().sum()
```

Out[7]:
```
125
```

In [8]:
```
# Drop Duplicates
df.drop_duplicates(inplace=True)
```

In [9]:
```
#Dummies Target Column "Revenue"
df['Revenue'] = pd.get_dummies(df['Revenue'], drop_first=True, dtype=int) #--> 1 for True
```

In [10]:
```
df['Revenue'].value_counts()
```

Out[10]:
```
Revenue
0    10297
1     1908
Name: count, dtype: int64
```

EDA:

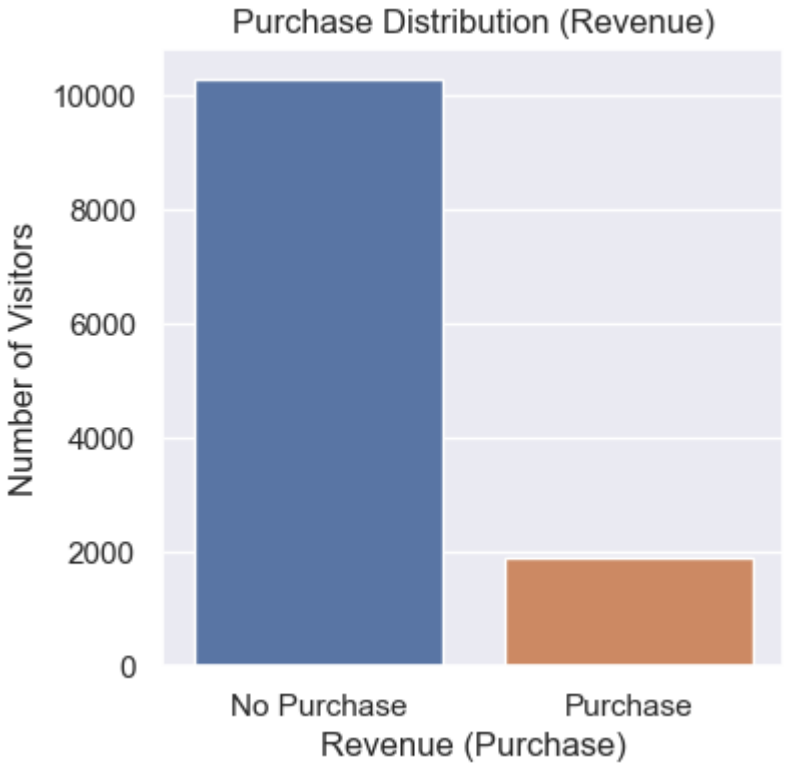
In [11]:
```
df.describe().round(1).T
```

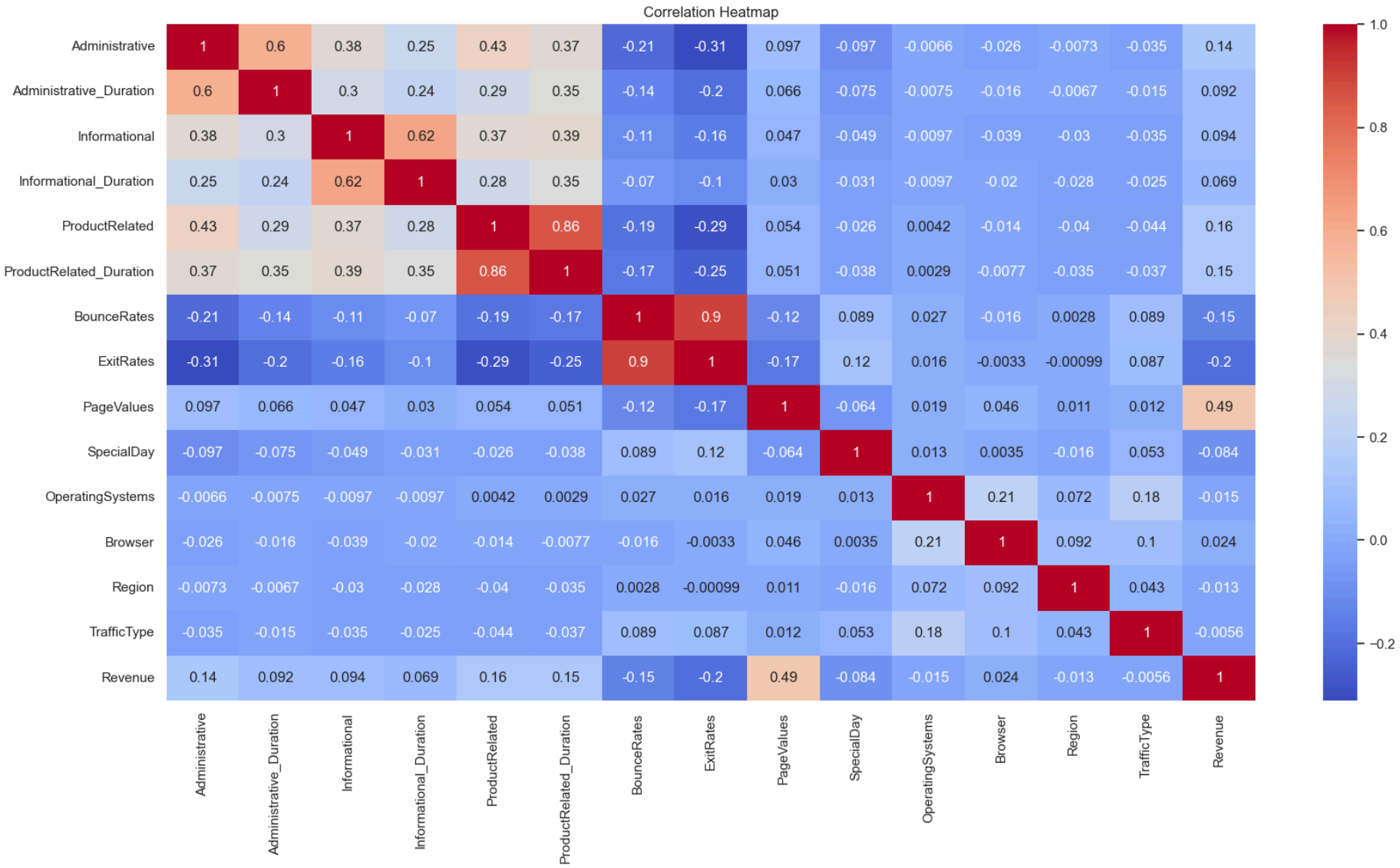
Out[11]:

	count	mean	std	min	25%	50%	75%	max
Administrative	12205.0	2.3	3.3	0.0	0.0	1.0	4.0	27.0
Administrative_Duration	12205.0	81.6	177.5	0.0	0.0	9.0	94.7	3398.8
Informational	12205.0	0.5	1.3	0.0	0.0	0.0	0.0	24.0
Informational_Duration	12205.0	34.8	141.4	0.0	0.0	0.0	0.0	2549.4
ProductRelated	12205.0	32.0	44.6	0.0	8.0	18.0	38.0	705.0
ProductRelated_Duration	12205.0	1207.0	1919.6	0.0	193.0	608.9	1477.2	63973.5
BounceRates	12205.0	0.0	0.0	0.0	0.0	0.0	0.0	0.2
ExitRates	12205.0	0.0	0.0	0.0	0.0	0.0	0.0	0.2
PageValues	12205.0	5.9	18.7	0.0	0.0	0.0	0.0	361.8
SpecialDay	12205.0	0.1	0.2	0.0	0.0	0.0	0.0	1.0
OperatingSystems	12205.0	2.1	0.9	1.0	2.0	2.0	3.0	8.0
Browser	12205.0	2.4	1.7	1.0	2.0	2.0	2.0	13.0
Region	12205.0	3.2	2.4	1.0	1.0	3.0	4.0	9.0
TrafficType	12205.0	4.1	4.0	1.0	2.0	2.0	4.0	20.0
Revenue	12205.0	0.2	0.4	0.0	0.0	0.0	0.0	1.0

In [84]:
```
# Plot the distribution of the Revenue column
plt.figure(figsize=(4, 4))
sns.countplot(data=df, x='Revenue', palette='deep')

plt.title('Purchase Distribution (Revenue)')
plt.xlabel('Revenue (Purchase)')
plt.ylabel('Number of Visitors')
```

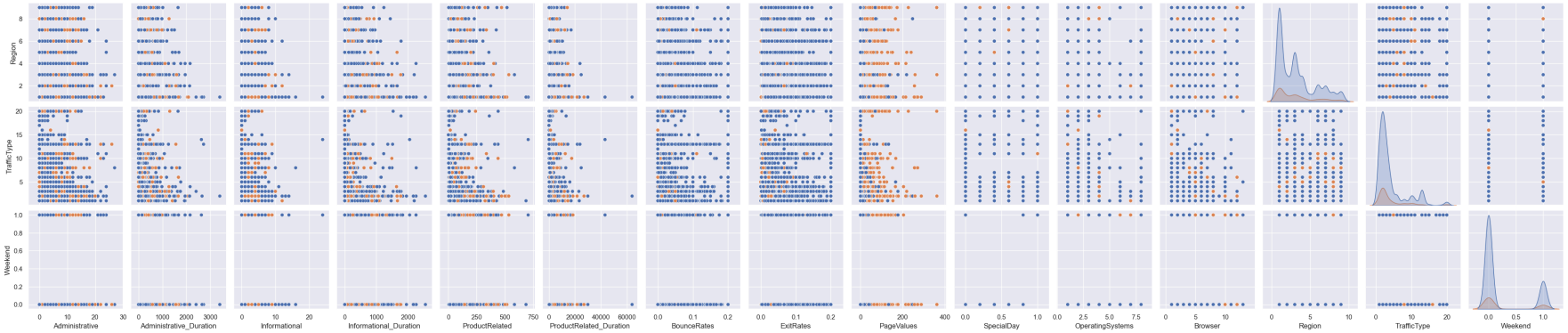
```
# Rename tick labels for better readability
plt.xticks([0, 1], ['No Purchase', 'Purchase'])
plt.show()


# The Relationship between variables
plt.figure(figsize=(20,10))
sns.heatmap(df.select_dtypes(include=['number']).corr(), annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```



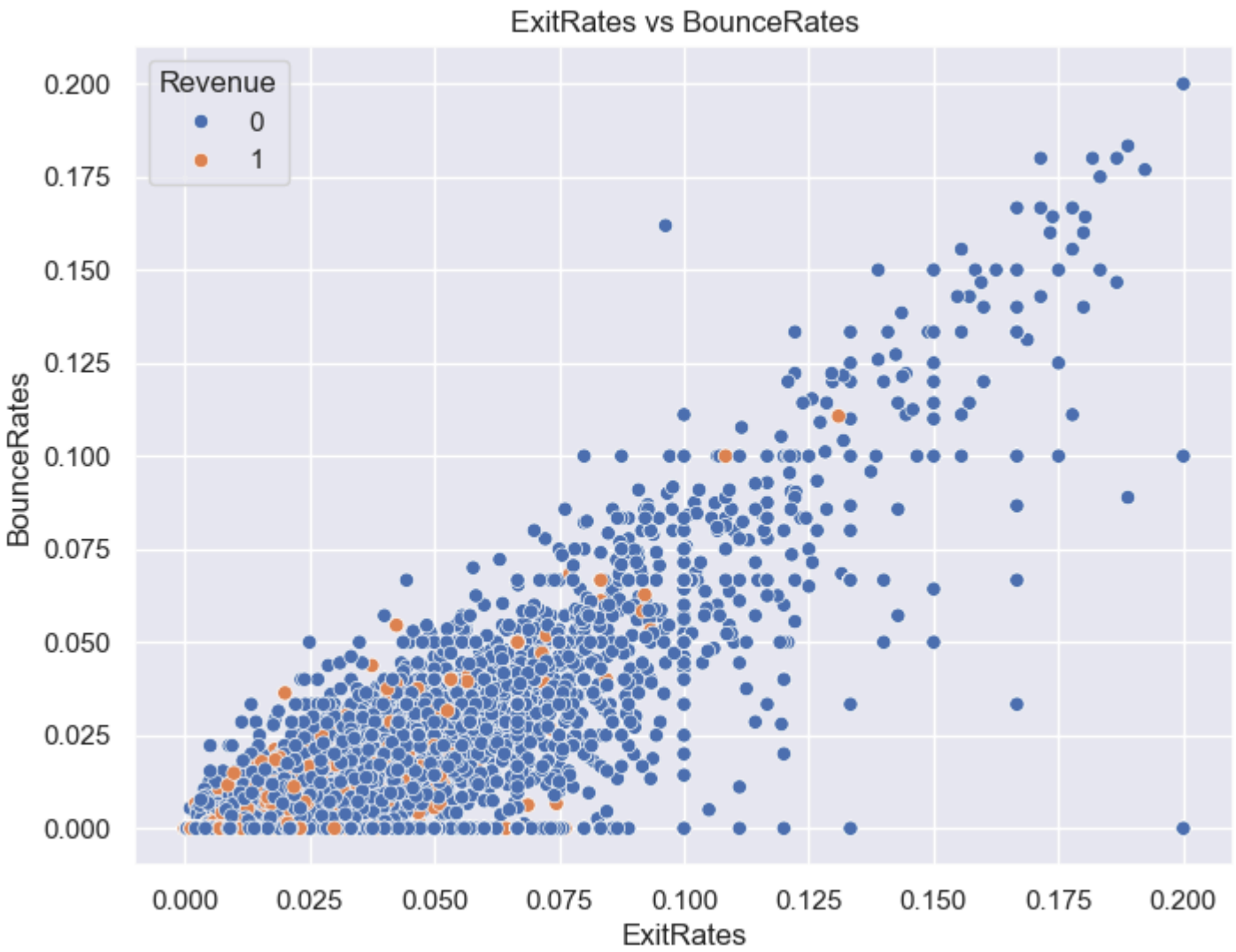


In [44]:
```
sns.pairplot(df, hue='Revenue')
plt.show()
```

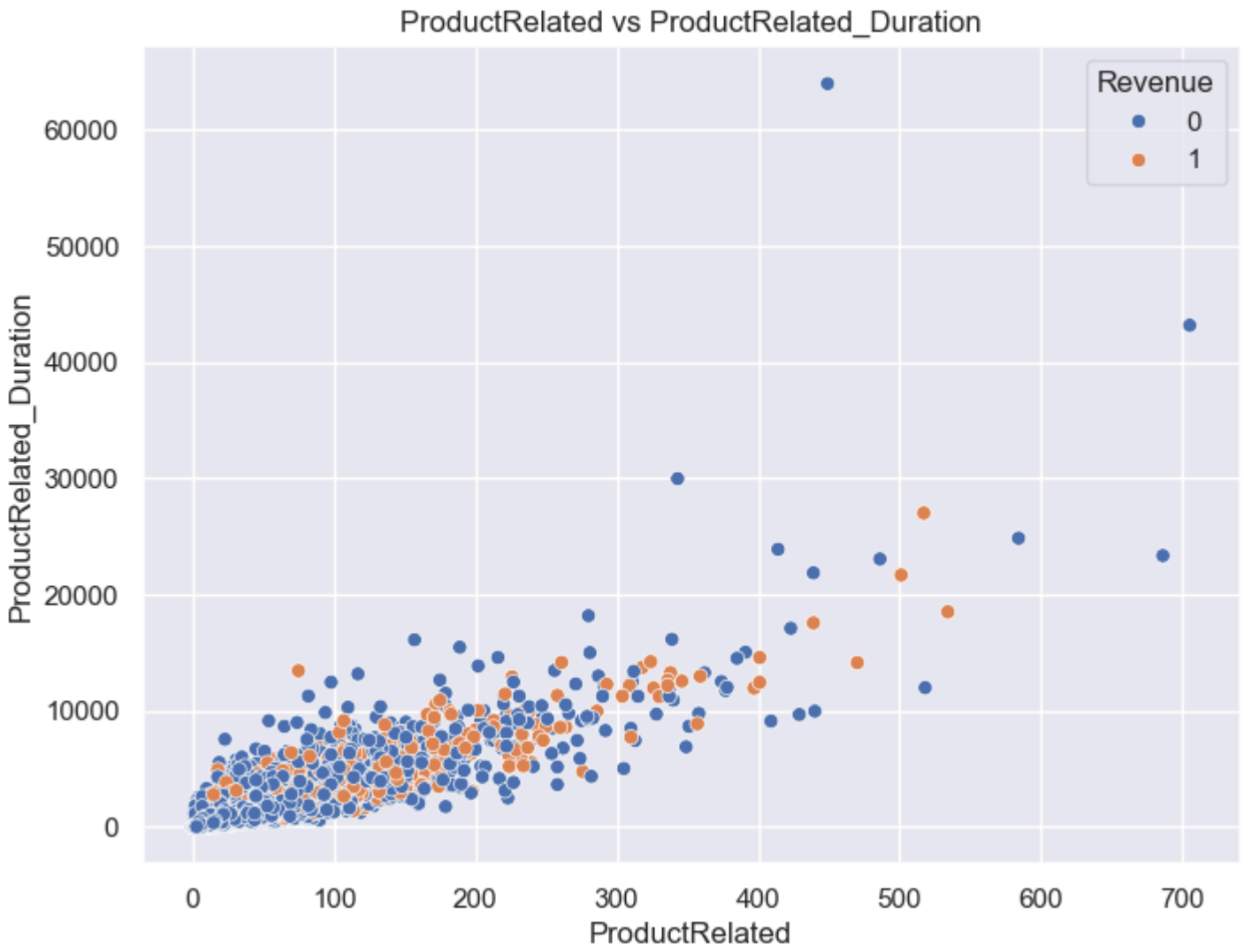

Administrative
Administrative_Duration
Informational
Informational_Duration
ProductRelated
ProductRelated_Duration
BounceRates
ExitRates
PageValues
SpecialDay
OperatingSystems
Browser
Revenue
False
True



In [72]:
```
plt.figure(figsize=(8, 6))
sns.scatterplot(data=df, x='ExitRates', y='BounceRates', hue='Revenue')
plt.title('ExitRates vs BounceRates')
plt.show()
```

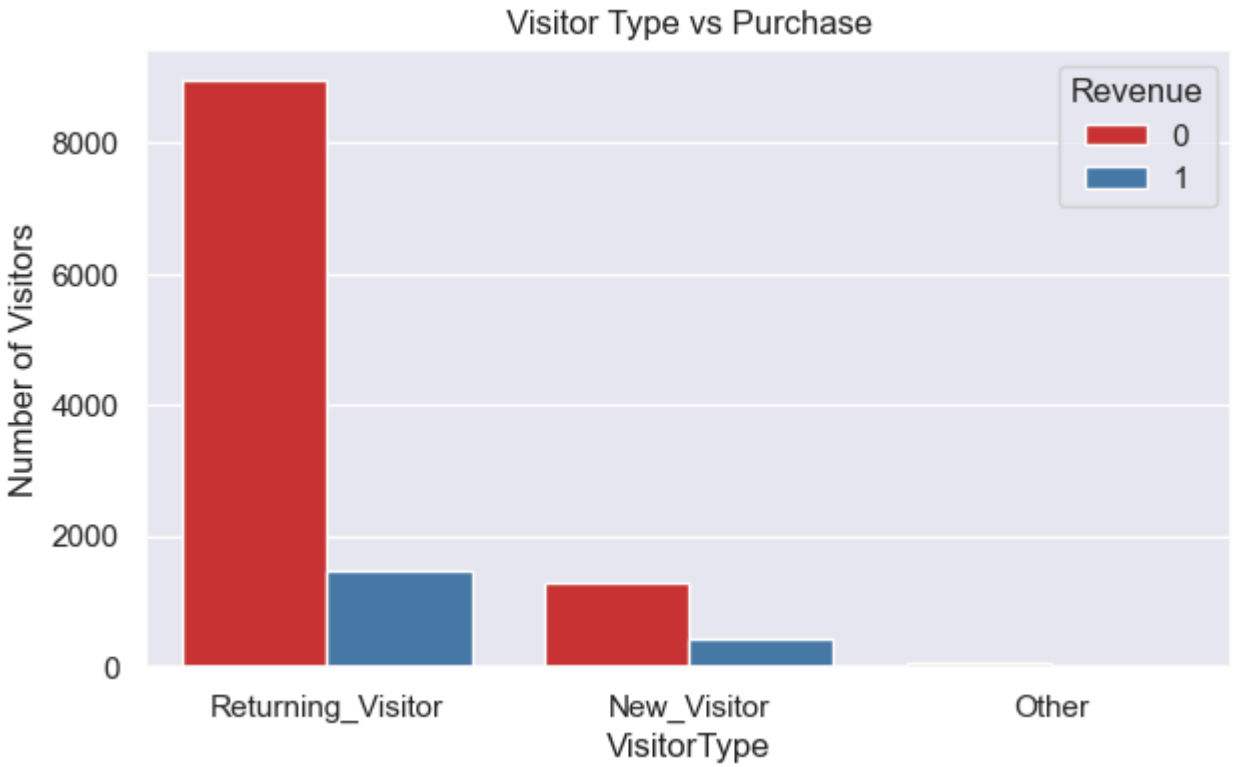


In [89]:
```
plt.figure(figsize=(8, 6))
sns.scatterplot(data=df, x='ProductRelated', y='ProductRelated_Duration', hue='Revenue')
plt.title('ProductRelated vs ProductRelated_Duration')
plt.show()
```



In [120…

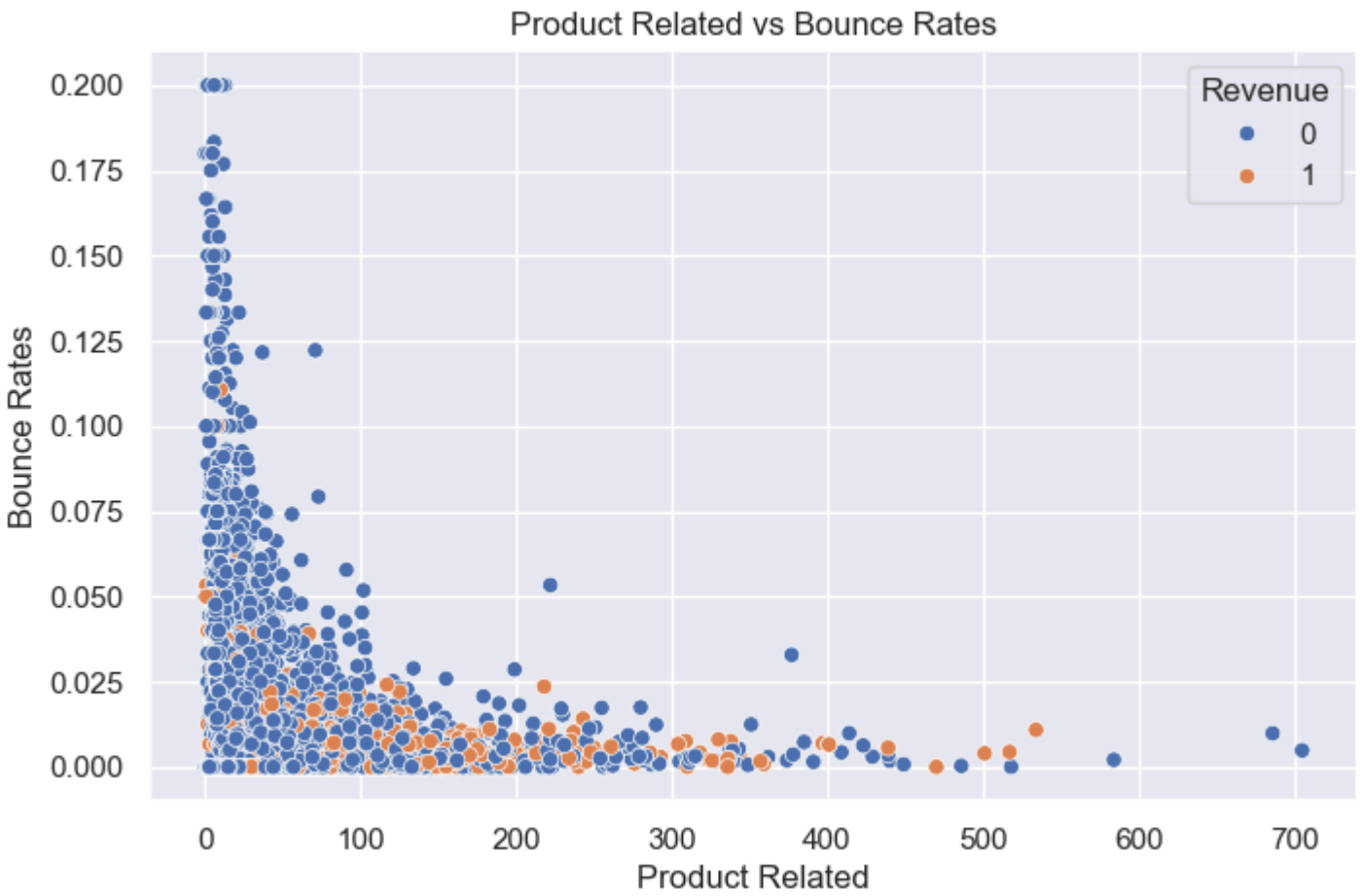
```
plt.figure(figsize=(7, 4))
sns.countplot(data=df, x='VisitorType', hue='Revenue', palette='Set1')
plt.title('Visitor Type vs Purchase')
plt.ylabel('Number of Visitors')
plt.show()
```



In [91]:
```
purchase_by_type = df.groupby('VisitorType')['Revenue'].mean().sort_values(ascending=False)
purchase_by_type.round(2)
```

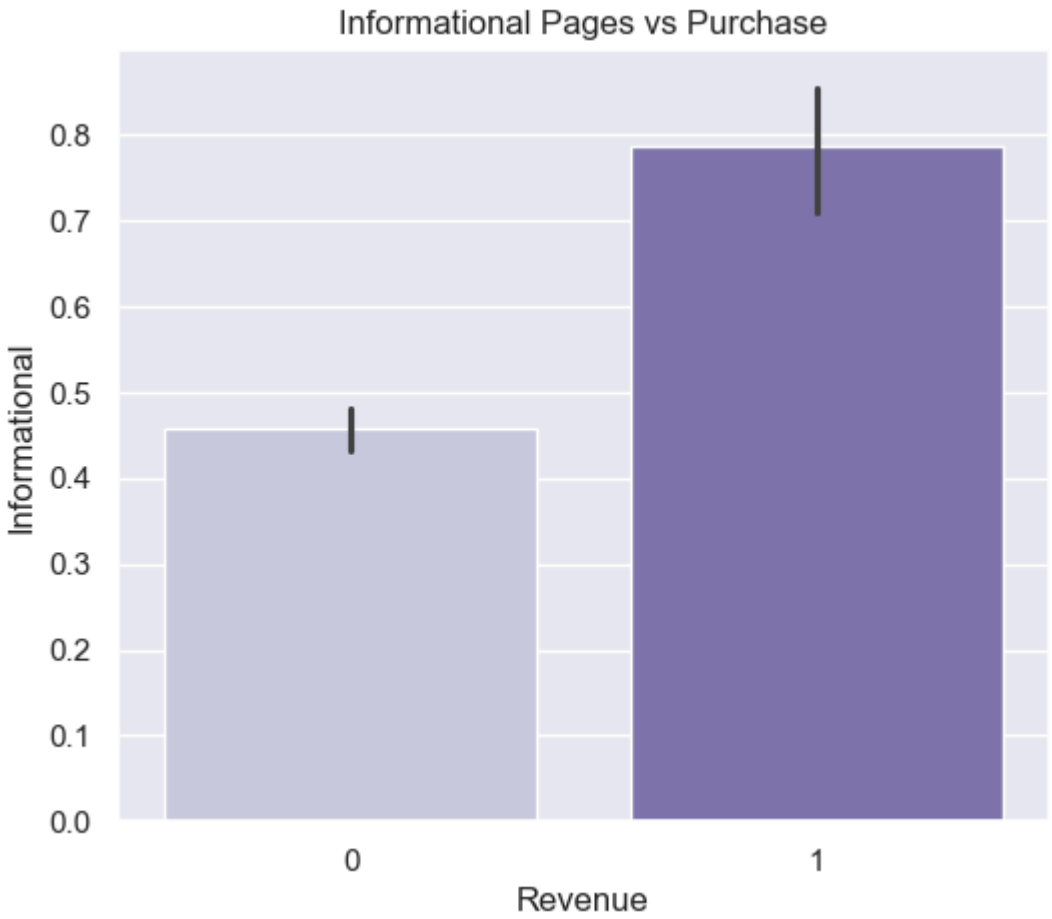
Out[91]:
```
VisitorType
New_Visitor          0.25
Other                0.20
Returning_Visitor    0.14
Name: Revenue, dtype: float64
```

In [104…
```
plt.figure(figsize=(8, 5))
sns.scatterplot(data=df, x='ProductRelated', y='BounceRates', hue='Revenue')
plt.title('Product Related vs Bounce Rates')
plt.xlabel('Product Related')
plt.ylabel('Bounce Rates')
plt.show()
```



In [117…

```
plt.figure(figsize=(6, 5))

sns.barplot(data=df, x='Revenue', y='Informational', palette='Purples')
plt.title('Informational Pages vs Purchase')
plt.xlabel('Revenue')
plt.ylabel('Informational')
plt.show()
```



In [ ]:

Preprocessing for ML Algorithms:

In [12]:
```
cols_to_scale = [
    'Administrative_Duration',
    'Informational_Duration',
    'ProductRelated_Duration',
    'BounceRates',
    'ExitRates',
```

```
    'PageValues'
]

scaler = StandardScaler()
df[cols_to_scale] = scaler.fit_transform(df[cols_to_scale])
```

In [13]:
```
df_encoded = pd.get_dummies(df, columns=['Month', 'VisitorType', 'Weekend'], drop_first=True, dtype=int)

# Now check your encoded DataFrame
df_encoded.head()
```

Out[13]:

	Administrative	Administrative_Duration	Informational	Informational_Duration	ProductRelated	ProductRelated_Duration	BounceRates
0	0	-0.460019	0	-0.246257	1	-0.628793	3.969402
1	0	-0.460019	0	-0.246257	2	-0.595451	-0.450137
2	0	-0.460019	0	-0.246257	1	-0.628793	3.969402
3	0	-0.460019	0	-0.246257	2	-0.627404	0.654748
4	0	-0.460019	0	-0.246257	10	-0.301889	-0.008183

5 rows × 27 columns

ML Algorithms:

In [14]:
```
# Splitting the Data
X = df_encoded.drop('Revenue', axis=1)
y= df_encoded['Revenue']
```

In [15]:
```
# We have Imbalanced Data --> So we will rebalance the samples using "SMOTE"
smote = SMOTE()
X_resampled, y_resampled = smote.fit_resample(X,y)
```

In [16]:
```
# Splitting the Data
```

```
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_size=0.2, random_state=42)
```

In [21]:
```
# Test the Model, is it Overfitting or not
test_overfitting = RandomForestClassifier()
test_overfitting.fit(X_train, y_train)

train_acc = accuracy_score(y_train, test_overfitting.predict(X_train))
test_acc = accuracy_score(y_test, test_overfitting.predict(X_test))

print(f"Training Accuracy: {train_acc:.2f}")
print(f"Testing Accuracy: {test_acc:.2f}")
```

```
Training Accuracy: 1.00
Testing Accuracy: 0.93
```

In [18]:
```
# Dictionary of models
models = {
    'Logistic Regression': LogisticRegression(),
    'Support Vector Classifier': SVC(),
    'Random Forest Classifier': RandomForestClassifier(),
    "KNN": KNeighborsClassifier()
}

# Create StratifiedKFold for samples are equal in each fold
skf = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)

# Loop through the models and calculate cross-validation accuracy
for model_name, model in models.items():
    scores = cross_val_score(model, X_resampled, y_resampled, cv=skf, scoring='accuracy')
    print(f'Accuracy Score with {model_name} = {scores.mean().round(2)}')
    print("-"*45)
```

```
Accuracy Score with Logistic Regression = 0.87
---------------------------------------------
Accuracy Score with Support Vector Classifier = 0.82
---------------------------------------------
Accuracy Score with Random Forest Classifier = 0.94
---------------------------------------------
Accuracy Score with KNN = 0.86
---------------------------------------------
```

In [25]:
```python
# Definition of measures
scoring = {
    'f1': 'f1',
    'recall': 'recall',
    'precision': 'precision',
    'roc_auc': 'roc_auc'
}

# Create StratifiedKFold for samples are equal in each fold
skf = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)

# Loop through the models and calculate cross-validation scores
for model_name, model in models.items():
    scores = cross_validate(model, X_resampled, y_resampled, cv=skf, scoring=scoring)

    # Calculate the Average for each measure
    f1_mean = scores['test_f1'].mean()
    recall_mean = scores['test_recall'].mean()
    precision_mean = scores['test_precision'].mean()
    roc_auc_mean = scores['test_roc_auc'].mean()

    print(f"{model_name}:")
    print(f'F1 Score = {f1_mean.round(2)}')
    print(f'Recall Score = {recall_mean.round(2)}')
    print(f'Precision Score = {precision_mean.round(2)}')
    print(f'ROC AUC Score = {roc_auc_mean.round(2)}')
    print("-" * 45)
```

```
Logistic Regression:
F1 Score = 0.87
Recall Score = 0.86
Precision Score = 0.89
ROC AUC Score = 0.94
-------------------------------------------
Support Vector Classifier:
F1 Score = 0.81
Recall Score = 0.78
Precision Score = 0.84
ROC AUC Score = 0.9
-------------------------------------------
Random Forest Classifier:
F1 Score = 0.94
Recall Score = 0.94
Precision Score = 0.93
ROC AUC Score = 0.99
-------------------------------------------
KNN:
F1 Score = 0.87
Recall Score = 0.98
Precision Score = 0.79
ROC AUC Score = 0.94
-------------------------------------------
```

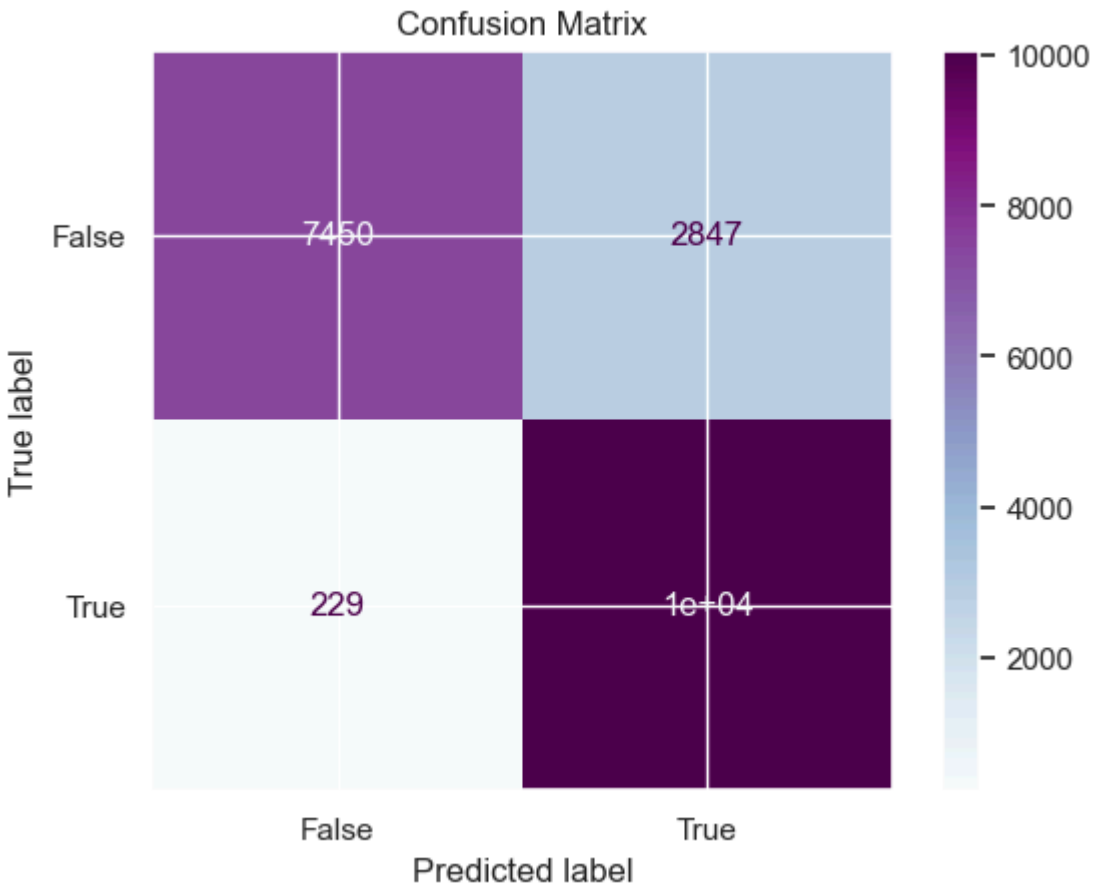
- Random Forest achieved higher performance likely due to its ensemble nature and ability to handle both categorical and numerical variables effectively. On the other hand, KNN struggled with precision, potentially due to overlapping class boundaries and sensitivity to outliers and feature scaling

In [19]:
```
# Confusion Matrix
y_pred = cross_val_predict(model, X_resampled, y_resampled, cv=10)
cm = confusion_matrix(y_resampled,y_pred)
print(cm)
```

```
[[ 7450  2847]
 [  229 10068]]
```

In [20]:
```
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['False', 'True'])
disp.plot(cmap='BuPu')
```

```
plt.title("Confusion Matrix")
plt.show()
```



In [ ]:

Deep Learning:

ANN:

In [143…

```
ann = Sequential()
```

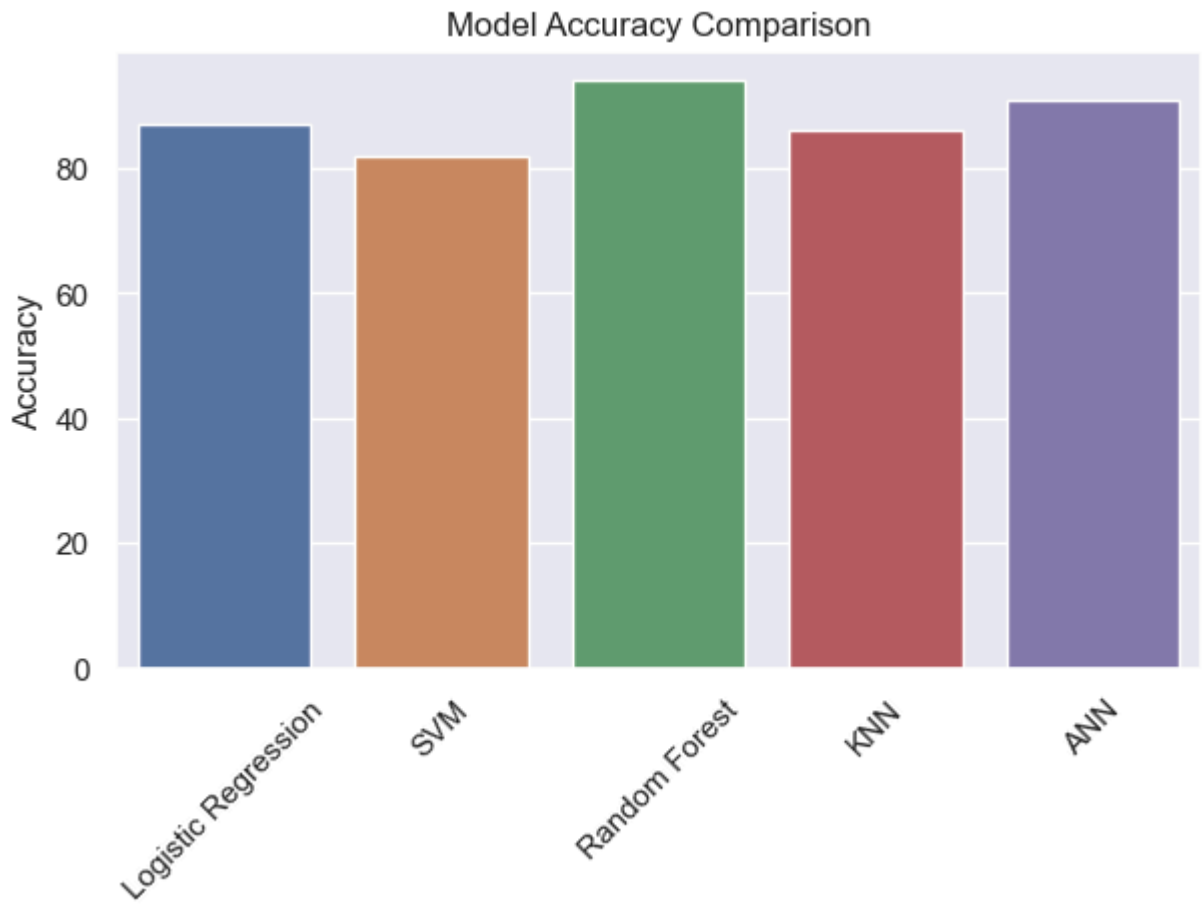
```
ann.add(Dense(units=64, activation='relu'), Dropout(0.3)) # Hidden Layer 1
ann.add(Dense(units=32, activation='relu'), Dropout(0.3)) # Hidden Layer 2
ann.add(Dense(units=1, activation='sigmoid')) # Output Layer

ann.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
ann.fit(X_train, y_train, validation_split=0.2, epochs=30, batch_size=32, verbose=1)

y_pred = ann.predict(X_test)
y_pred = (y_pred > 0.5).astype(int)  # Convert probabilities to binary class labels
score_ann= accuracy_score(y_test, y_pred)
print(f'Accuracy Score with ANN = {score_ann:.2f}')
```

```
Epoch 1/30
412/412 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.7326 - loss: 0.5720 - val_accuracy: 0.8737 - val_loss: 0.3252
Epoch 2/30
412/412 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8637 - loss: 0.3430 - val_accuracy: 0.8850 - val_loss: 0.2972
Epoch 3/30
412/412 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8797 - loss: 0.3113 - val_accuracy: 0.8938 - val_loss: 0.2762
Epoch 4/30
412/412 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8813 - loss: 0.3002 - val_accuracy: 0.8974 - val_loss: 0.2587
Epoch 5/30
412/412 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8837 - loss: 0.2823 - val_accuracy: 0.8980 - val_loss: 0.2614
Epoch 6/30
412/412 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8907 - loss: 0.2738 - val_accuracy: 0.8992 - val_loss: 0.2502
Epoch 7/30
412/412 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9074 - loss: 0.2394 - val_accuracy: 0.8868 - val_loss: 0.2641
Epoch 8/30
412/412 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9025 - loss: 0.2389 - val_accuracy: 0.9029 - val_loss: 0.2432
Epoch 9/30
412/412 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8964 - loss: 0.2532 - val_accuracy: 0.8965 - val_loss: 0.2710
Epoch 10/30
412/412 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9022 - loss: 0.2368 - val_accuracy: 0.8950 - val_loss: 0.2639
Epoch 11/30
412/412 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8975 - loss: 0.2467 - val_accuracy: 0.8977 - val_loss: 0.2570
Epoch 12/30
412/412 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9051 - loss: 0.2304 - val_accuracy: 0.8656 - val_loss: 0.3313
Epoch 13/30
412/412 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9006 - loss: 0.2350 - val_accuracy: 0.8914 - val_loss: 0.2701
Epoch 14/30
412/412 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9050 - loss: 0.2255 - val_accuracy: 0.9059 - val_loss: 0.2319
Epoch 15/30
412/412 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9120 - loss: 0.2114 - val_accuracy: 0.9047 - val_loss: 0.2278
Epoch 16/30
412/412 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9147 - loss: 0.2066 - val_accuracy: 0.9035 - val_loss: 0.2290
Epoch 17/30
412/412 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9132 - loss: 0.2110 - val_accuracy: 0.9023 - val_loss: 0.2528
Epoch 18/30
412/412 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9123 - loss: 0.2106 - val_accuracy: 0.9102 - val_loss: 0.2261
Epoch 19/30
412/412 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9126 - loss: 0.2110 - val_accuracy: 0.8822 - val_loss: 0.2623
Epoch 20/30
412/412 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9097 - loss: 0.2230 - val_accuracy: 0.9068 - val_loss: 0.2316
Epoch 21/30
```

```
412/412 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9154 - loss: 0.2013 - val_accuracy: 0.9074 - val_loss: 0.2256
Epoch 22/30
412/412 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9180 - loss: 0.2007 - val_accuracy: 0.8917 - val_loss: 0.2489
Epoch 23/30
412/412 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9150 - loss: 0.2039 - val_accuracy: 0.9077 - val_loss: 0.2238
Epoch 24/30
412/412 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9204 - loss: 0.1926 - val_accuracy: 0.9099 - val_loss: 0.2324
Epoch 25/30
412/412 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9207 - loss: 0.1921 - val_accuracy: 0.8932 - val_loss: 0.2482
Epoch 26/30
412/412 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9142 - loss: 0.2053 - val_accuracy: 0.9138 - val_loss: 0.2269
Epoch 27/30
412/412 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9194 - loss: 0.1912 - val_accuracy: 0.9062 - val_loss: 0.2347
Epoch 28/30
412/412 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9168 - loss: 0.1950 - val_accuracy: 0.9102 - val_loss: 0.2252
Epoch 29/30
412/412 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9196 - loss: 0.1932 - val_accuracy: 0.9120 - val_loss: 0.2199
Epoch 30/30
412/412 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9223 - loss: 0.1849 - val_accuracy: 0.9083 - val_loss: 0.2258
129/129 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
Accuracy Score with ANN = 0.91
```

In [ ]:

In [22]:
```python
# Finally: model accuracy comparison
plt.figure(figsize=(7,4))
model_scores = {
    'Logistic Regression': 87,
    'SVM': 82,
    'Random Forest': 94,
    'KNN': 86,
    'ANN': 91
}
sns.barplot(x= list(model_scores.keys()), y= list(model_scores.values()), palette='deep')
plt.title('Model Accuracy Comparison')
plt.ylabel('Accuracy')
plt.xticks(rotation=45)
plt.show()
```



Summary:

Project: Shoppers Behavior & Revenue Prediction

Goal: Predict whether a user will make a purchase based on session data.

Data Highlights:

- ~12,000 sessions with features like Page Duration, Bounce Rate, Visitor Type.
- Target: Revenue (True/False)

Steps Taken:

- Cleaned & encoded data, scaled numerics, handled class imbalance with SMOTE.
- EDA revealed returning visitors and high page value increase revenue chances.
- Built and compared ML models (LogReg, SVM, RF) – Random Forest performed best (94% accuracy).
- Developed an ANN achieving 91% accuracy with no overfitting.

Next Steps:

- Add StratifiedKFold and deeper error analysis.
- Tune ANN further.
- Deploy with Streamlit UI.

Thank You