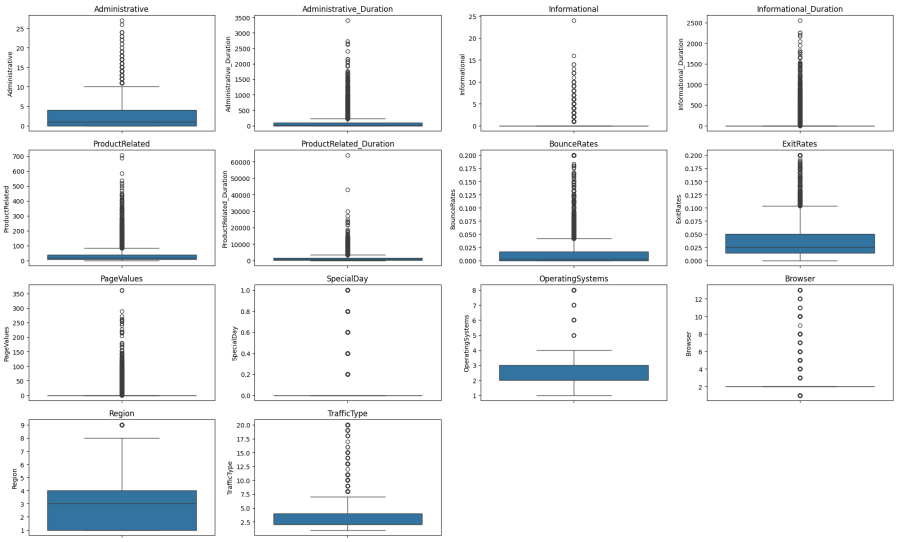
```
import pandas as pd

df = pd.read_csv('online_shoppers_intention.csv')
```

Double-click (or enter) to edit

```
print(df.head())

print(df.info())

print(df.describe())
```

```
 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
None
       Administrative  Administrative_Duration  Informational  \
count    12330.000000             12330.000000   12330.000000
mean         2.315166                80.818611       0.503569
std          3.321784               176.779107       1.270156
min          0.000000                 0.000000       0.000000
25%          0.000000                 0.000000       0.000000
50%          1.000000                 7.500000       0.000000
75%          4.000000                93.256250       0.000000
max         27.000000              3398.750000      24.000000

       Informational_Duration  ProductRelated  ProductRelated_Duration  \
count            12330.000000    12330.000000             12330.000000
mean                34.472398       31.731468              1194.746220
std                140.749294       44.475503              1913.669288
min                  0.000000        0.000000                 0.000000
25%                  0.000000        7.000000               184.137500
50%                  0.000000       18.000000               598.936905
75%                  0.000000       38.000000              1464.157214
max               2549.375000      705.000000             63973.522230

        BounceRates     ExitRates    PageValues    SpecialDay  \
count  12330.000000  12330.000000  12330.000000  12330.000000
mean       0.022191      0.043073      5.889258      0.061427
std        0.048488      0.048597     18.568437      0.198917
min        0.000000      0.000000      0.000000      0.000000
25%        0.000000      0.014286      0.000000      0.000000
50%        0.003112      0.025156      0.000000      0.000000
75%        0.016813      0.050000      0.000000      0.000000
max        0.200000      0.200000    361.763742      1.000000

       OperatingSystems       Browser        Region   TrafficType
count      12330.000000  12330.000000  12330.000000  12330.000000
mean           2.124006      2.357097      3.147364      4.069586
std            0.911325      1.717277      2.401591      4.025169
min            1.000000      1.000000      1.000000      1.000000
25%            2.000000      2.000000      1.000000      2.000000
50%            2.000000      2.000000      3.000000      2.000000
75%            3.000000      2.000000      4.000000      4.000000
max            8.000000     13.000000      9.000000     20.000000
```

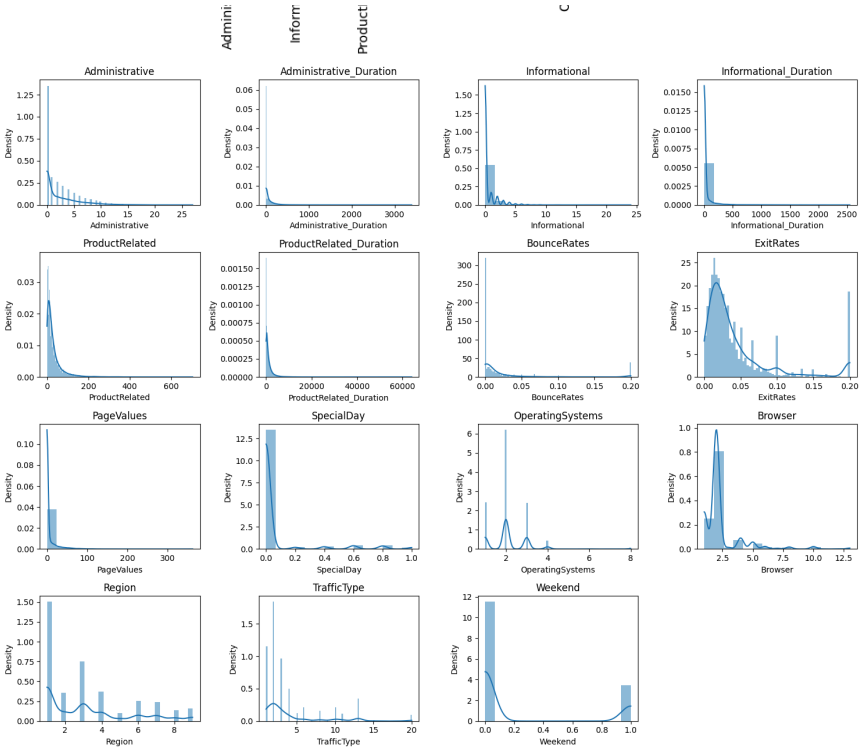
```
import matplotlib.pyplot as plt
import seaborn as sns


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

plt.figure(figsize=(20, 15))
for i, col in enumerate(numerical_cols):
    plt.subplot(5, 4, i + 1)
    sns.boxplot(y=df[col])
    plt.title(col)
plt.tight_layout()
plt.show()
```

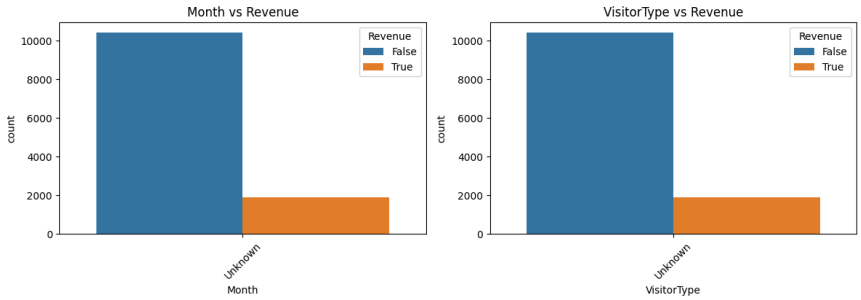



```
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import train_test_split

label_encoders = {}
for column in ['Month', 'VisitorType', 'Weekend']:
    le = LabelEncoder()
    df[column] = le.fit_transform(df[column])
    label_encoders[column] = le

X = df.drop('Revenue', axis=1)
y = df['Revenue'].astype(int)

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42, stratify=y)

print(f"Training set size: {X_train.shape[0]} samples")
print(f"Testing set size: {X_test.shape[0]} samples")
```

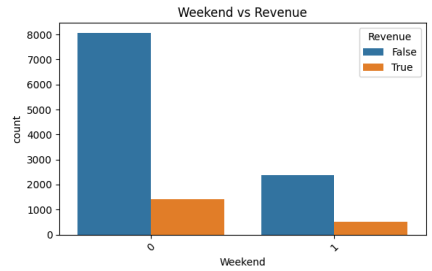
```
Training set size: 9864 samples
Testing set size: 2466 samples
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import train_test_split
import math

plt.figure(figsize=(10, 8))
sns.heatmap(X.corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix Before Inverse Transformation')
plt.show()

def safe_inverse_transform(le, series):
    inverse_mapping = {v: k for k, v in enumerate(le.classes_)}

    return series.apply(lambda x: inverse_mapping.get(x, 'Unknown'))

for column in ['Month', 'VisitorType', 'Weekend']:
    df[column] = safe_inverse_transform(label_encoders[column], df[column])

numerical_features = df.select_dtypes(include=['float64', 'int64']).columns
n_features = len(numerical_features)
cols = 4
rows = math.ceil(n_features / cols)

plt.figure(figsize=(15, 12))
for i, col in enumerate(numerical_features):
    plt.subplot(rows, cols, i + 1)
    sns.histplot(df[col], kde=True, stat="density", linewidth=0)
    plt.title(col)
plt.tight_layout()
plt.show()

categorical_features = ['Month', 'VisitorType', 'Weekend']
plt.figure(figsize=(12, 8))
for i, col in enumerate(categorical_features):
    plt.subplot(2, 2, i + 1)
    sns.countplot(x=col, hue='Revenue', data=df)
    plt.title(f'{col} vs Revenue')
    plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```


Administrative
Administrative_Duration
Informational
Informational_Duration
ProductRelated
ProductRelated_Duration
BounceRates
ExitRates
PageValues
SpecialDay
OperatingSystems
Browser
Region
TrafficType
Weekend
Density


Month vs Revenue
Revenue
False
True
count
Unknown
Month

VisitorType vs Revenue
Revenue
False
True
count
Unknown
VisitorType


Weekend vs Revenue
Revenue
False
True
count
0
1
Weekend

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_selection import SelectFromModel


rf = RandomForestClassifier(n_estimators=100, random_state=42, n_jobs=-1)

rf.fit(X_train, y_train)

selector = SelectFromModel(rf, prefit=True)
X_important_train = selector.transform(X_train)
X_important_test = selector.transform(X_test)

selected_features_mask = selector.get_support()

print("Selected features:", df.drop('Revenue', axis=1).columns[selected_features_mask])
```

```
Selected features: Index(['Administrative_Duration', 'ProductRelated', 'ProductRelated_Duration',
       'ExitRates', 'PageValues'],
      dtype='object')
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score


models = {
    "Logistic Regression": LogisticRegression(max_iter=1000),
    "Decision Tree": DecisionTreeClassifier(),
    "KNN": KNeighborsClassifier(),
    "Naive Bayes": GaussianNB()
}

def evaluate_and_display_model_performance(models, X_train, X_test, y_train, y_test):
    for name, model in models.items():
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)
        y_pred_proba = model.predict_proba(X_test)[:, 1] if hasattr(model, "predict_proba") else None


        accuracy = accuracy_score(y_test, y_pred)
        precision = precision_score(y_test, y_pred, zero_division=0)
        recall = recall_score(y_test, y_pred)
        f1 = f1_score(y_test, y_pred)
        roc_auc = roc_auc_score(y_test, y_pred_proba) if y_pred_proba is not None else "N/A"


        print(f"{name}:")
        print(f" Accuracy: {accuracy:.4f}")
        print(f" Precision: {precision:.4f}")
        print(f" Recall: {recall:.4f}")
        print(f" F1 Score: {f1:.4f}")
        print(f" ROC AUC: {roc_auc if roc_auc != 'N/A' else 'Not Applicable (Model does not support probability estimates)'}\n")


evaluate_and_display_model_performance(models, X_important_train, X_important_test, y_train, y_test)
```

```
Logistic Regression:
 Accuracy: 0.8804
 Precision: 0.7514
 Recall: 0.3403
 F1 Score: 0.4685
 ROC AUC: 0.857388253559908

Decision Tree:
 Accuracy: 0.8504
 Precision: 0.5169
 Recall: 0.5209
 F1 Score: 0.5189
 ROC AUC: 0.7048510968636632

KNN:
 Accuracy: 0.8816
 Precision: 0.6520
 Recall: 0.5052
 F1 Score: 0.5693
 ROC AUC: 0.8304722593482128

Naive Bayes:
 Accuracy: 0.8589
 Precision: 0.5489
 Recall: 0.5000
 F1 Score: 0.5233
 ROC AUC: 0.8212898574026992
```

```
from sklearn.model_selection import GridSearchCV


param_grid_lr = {'C': [0.001, 0.01, 0.1, 1, 10, 100], 'penalty': ['l1', 'l2']}
param_grid_dt = {'max_depth': [None, 10, 20, 30], 'min_samples_split': [2, 5, 10]}
param_grid_knn = {'n_neighbors': [3, 5, 7, 9], 'weights': ['uniform', 'distance']}
param_grid_nb = {}  # GaussianNB doesn't have relevant hyperparameters that are typically tuned


best_estimators = {}

grid_searches = {
    "Logistic Regression": GridSearchCV(LogisticRegression(max_iter=1000), param_grid_lr, cv=5, scoring='accuracy'),
    "Decision Tree": GridSearchCV(DecisionTreeClassifier(), param_grid_dt, cv=5, scoring='accuracy'),
    "KNN": GridSearchCV(KNeighborsClassifier(), param_grid_knn, cv=5, scoring='accuracy')
    # Naive Bayes is not included due to the lack of common hyperparameters to tune.
}


for name, grid_search in grid_searches.items():
    print(f"Running GridSearchCV for {name}...")
    grid_search.fit(X_important_train, y_train)
    best_estimators[name] = grid_search.best_estimator_
    print(f"Best parameters for {name}: {grid_search.best_params_}")
    print(f"Best score for {name}: {grid_search.best_score_}\n")

# Adding Naive Bayes to best_estimators manually
best_estimators["Naive Bayes"] = GaussianNB()
```

```
Running GridSearchCV for Logistic Regression...
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py:378: FitFailedWarning:
30 fits failed out of a total of 60.
The score on these train-test partitions for these parameters will be set to nan.
If these failures are not expected, you can try to debug them by setting error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
30 fits failed with the following error:
Traceback (most recent call last):
  File "/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py", line 1162, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File "/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py", line 54, in _check_solver
    raise ValueError(
ValueError: Solver lbfgs supports only 'l2' or 'none' penalties, got l1 penalty.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_search.py:952: UserWarning: One or more of the test scores are non-fini
        nan 0.88452918        nan 0.88452918        nan 0.8846306 ]
  warnings.warn(
Best parameters for Logistic Regression: {'C': 100, 'penalty': 'l2'}
Best score for Logistic Regression: 0.8846305962131782

Running GridSearchCV for Decision Tree...
Best parameters for Decision Tree: {'max_depth': 10, 'min_samples_split': 5}
Best score for Decision Tree: 0.8822998409563592

Running GridSearchCV for KNN...
Best parameters for KNN: {'n_neighbors': 9, 'weights': 'uniform'}
Best score for KNN: 0.8900033309721813
```

```
evaluate_and_display_model_performance(best_estimators, X_important_train, X_important_test, y_train, y_test)
```

```
Logistic Regression:
 Accuracy: 0.8804
 Precision: 0.7514
 Recall: 0.3403
 F1 Score: 0.4685
 ROC AUC: 0.8572915305845584

Decision Tree:
 Accuracy: 0.8816
 Precision: 0.6424
 Recall: 0.5314
 F1 Score: 0.5817
```