baselines_model_with_ctgan

May 18, 2025

1 Comparing performance of models using synthetic data

This notebook is used to test the performance of baseline models by applying synthetic data generated from CTGAN model

[34]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split, cross_val_score,␣
 ↪StratifiedKFold
from sklearn.preprocessing import MinMaxScaler, LabelEncoder
from sklearn.metrics import accuracy_score, confusion_matrix,␣
 ↪classification_report, roc_auc_score, roc_curve, auc

from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier,␣
 ↪GradientBoostingClassifier
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from catboost import CatBoostClassifier

# from sklearn.neural_network import MLPClassifier


import warnings
warnings.filterwarnings("ignore")
```

[35]:
```
data = pd.read_csv('/home/nhat/projectcuoiky/data/pdf_features.csv')
data.info()
data.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11101 entries, 0 to 11100
Data columns (total 25 columns):
 #   Column         Non-Null Count  Dtype
```

```
---  ------          --------------  -----
 0   Page            11101 non-null  int64
 1   Encrypt         11101 non-null  int64
 2   ObjStm          11101 non-null  int64
 3   JS              11101 non-null  int64
 4   JavaScript      11101 non-null  int64
 5   AA              11101 non-null  int64
 6   OpenAction      11101 non-null  int64
 7   AcroForm        11101 non-null  int64
 8   JBIG2Decode     11101 non-null  int64
 9   RichMedia       11101 non-null  int64
 10  Launch          11101 non-null  int64
 11  EmbeddedFile    11101 non-null  int64
 12  XFA             11101 non-null  int64
 13  Colors_gt_224   11101 non-null  int64
 14  obj             11101 non-null  int64
 15  endobj          11101 non-null  int64
 16  stream          11101 non-null  int64
 17  endstream       11101 non-null  int64
 18  xref            11101 non-null  int64
 19  trailer         11101 non-null  int64
 20  startxref       11101 non-null  int64
 21  filepath        11101 non-null  object
 22  filename        11101 non-null  object
 23  filesize_kb     11101 non-null  float64
 24  label           11101 non-null  object
dtypes: float64(1), int64(21), object(3)
memory usage: 2.1+ MB
```

[35]:

```
   Page  Encrypt  ObjStm  JS  JavaScript  AA  OpenAction  AcroForm  \
0     1        0       0   0           0   0           0         0
1     1        0       0   0           0   0           0         0
2     4        0       6   0           0   0           0         0
3     1        0       0   0           0   0           0         1
4     6        0      25   0           0   0           0         2

   JBIG2Decode  RichMedia  …  endobj  stream  endstream  xref  trailer  \
0            0          0  …      11       3          3     2        2
1            0          0  …       6       2          2     1        1
2            0          0  …      56      41         41     0        0
3            0          0  …      29      17         17     2        2
4            0          0  …     156     146        146     0        0

   startxref                                          filepath  \
0          2  /home/remnux/Desktop/extraction/data/Benign/as…
1          1  /home/remnux/Desktop/extraction/data/Benign/ar…
2          3  /home/remnux/Desktop/extraction/data/Benign/p4…
```

```
3          2  /home/remnux/Desktop/extraction/data/Benign/ar…
4          4  /home/remnux/Desktop/extraction/data/Benign/f9…

          filename  filesize_kb   label
0        assehc.pdf    23.120117  benign
1     artauthor.pdf    69.544922  benign
2      p4894_ru.pdf   180.786133  benign
3  artisticwall.pdf    85.124023  benign
4        f990sn.pdf   126.099609  benign

[5 rows x 25 columns]
```

[36]:

```python
import pandas as pd
import numpy as np

# Path to the synthetic data - Ensure this is correct
synthetic_data_path = '/home/nhat/projectcuoiky/output/
 ↪new_synthetic_malicious_data_8000_samples.csv'

# Define the target column name we want to use consistently
TARGET_COL = 'Class'

# --- 1. Prepare Original Data (loaded as 'data' in the previous cell) ---
print("--- Processing Original Data ---")
print(f"Original data shape: {data.shape}")
print(f"Original data columns: {data.columns.tolist()}")

if TARGET_COL not in data.columns:
    print(f"'{TARGET_COL}' column not found in original data.")
    if 'label' in data.columns:
        print("Found 'label' column in original data. Renaming to 'Class'.")
        data.rename(columns={'label': TARGET_COL}, inplace=True)
    elif 'Category' in data.columns: # Should not happen based on user␣
 ↪feedback, but as a fallback
        print("Found 'Category' column in original data. Renaming to 'Class'.")
        data.rename(columns={'Category': TARGET_COL}, inplace=True)
    else:
        # If you have another name for the target in original data, add its␣
 ↪renaming logic here
        print(f"ERROR: Original data must have a '{TARGET_COL}' column or a␣
 ↪known alias like 'label' to be renamed.")
        # Raising an error or stopping might be appropriate here if target is␣
 ↪missing
        # For now, we'll let it proceed and it might fail later if TARGET_COL␣
 ↪is still not there.
else:
    print(f"'{TARGET_COL}' column already exists in original data.")
```

```
# --- 2. Load and Prepare Synthetic Data ---
print("\n--- Processing Synthetic Data ---")
try:
    synthetic_df = pd.read_csv(synthetic_data_path)
    print(f"Synthetic data loaded successfully from: {synthetic_data_path}")
    print(f"Synthetic data shape: {synthetic_df.shape}")
    print(f"Synthetic data columns (raw): {synthetic_df.columns.tolist()}")

    if TARGET_COL not in synthetic_df.columns:
        if 'label' in synthetic_df.columns:
            synthetic_df.rename(columns={'label': TARGET_COL}, inplace=True)
            print(f"Renamed 'label' to '{TARGET_COL}' in synthetic data.")
        elif 'label_numeric' in synthetic_df.columns:
            # Assuming 0 for benign, 1 for malicious. Adjust if mapping is␣
↪different.
            synthetic_df[TARGET_COL] = np.where(synthetic_df['label_numeric']␣
↪== 1, 'malicious', 'benign')
            print(f"Created '{TARGET_COL}' column in synthetic_df from␣
↪'label_numeric'.")
            # Optionally drop 'label_numeric' if it's no longer needed and not␣
↪a feature
            # synthetic_df.drop(columns=['label_numeric'], inplace=True,␣
↪errors='ignore')
        else:
            raise ValueError(f"Synthetic data must have a '{TARGET_COL}',␣
↪'label', or 'label_numeric' column.")
    else:
        print(f"'{TARGET_COL}' column already exists in synthetic data.")

    # --- 3. Align Columns and Concatenate ---
    print("\n--- Aligning and Concatenating Data ---")
    original_cols = set(data.columns)
    synthetic_cols = set(synthetic_df.columns)

    common_cols = list(original_cols.intersection(synthetic_cols))

    if not common_cols:
        raise ValueError("No common columns found between original and␣
↪synthetic data. Check data preparation.")
    if TARGET_COL not in common_cols:
        # This should not happen if previous steps worked and TARGET_COL was in␣
↪both
        raise ValueError(f"Critical: '{TARGET_COL}' is not in common columns.␣
↪Original cols: {original_cols}, Synthetic cols: {synthetic_cols}")
```

```
    print(f"Common columns for alignment (including target): {common_cols}")

    data_aligned = data[common_cols]
    synthetic_df_aligned = synthetic_df[common_cols]

    augmented_data = pd.concat([data_aligned, synthetic_df_aligned],␣
↪ignore_index=True)
    print(f"\nShape of original_aligned data: {data_aligned.shape}")
    print(f"Shape of synthetic_aligned data: {synthetic_df_aligned.shape}")
    print(f"Shape of augmented data: {augmented_data.shape}")
    print("Augmented data columns:")
    print(augmented_data.columns.tolist())
    print("Augmented data head (first 2 rows of original, then first 2 of␣
↪synthetic part):")
    # Ensure there are enough rows in original data before trying to show␣
↪merged head this way
    if len(data_aligned) >= 2 and len(synthetic_df_aligned) >=2 :
        print(pd.concat([augmented_data.head(2), augmented_data.
↪iloc[data_aligned.shape[0]:data_aligned.shape[0]+2]]))
    else:
        print(augmented_data.head())

    # Replace the original 'data' DataFrame with the augmented one
    data = augmented_data
    print(f"\n'data' variable now refers to the augmented dataset. Final␣
↪columns: {data.columns.tolist()}")

except FileNotFoundError:
    print(f"ERROR: Synthetic data file not found at '{synthetic_data_path}'.")
    print("Please ensure the file exists or update the path.")
    print("Proceeding with original data only. This may cause issues in␣
↪subsequent cells if data is not as expected.")

except Exception as e:
    print(f"An error occurred during data preparation in this cell: {e}")
    import traceback
    traceback.print_exc()
    print("Proceeding with original data only. This may cause issues in␣
↪subsequent cells if data is not as expected.")
```

```
--- Processing Original Data ---
Original data shape: (11101, 25)
Original data columns: ['Page', 'Encrypt', 'ObjStm', 'JS', 'JavaScript', 'AA',
'OpenAction', 'AcroForm', 'JBIG2Decode', 'RichMedia', 'Launch', 'EmbeddedFile',
'XFA', 'Colors_gt_224', 'obj', 'endobj', 'stream', 'endstream', 'xref',
'trailer', 'startxref', 'filepath', 'filename', 'filesize_kb', 'label']
'Class' column not found in original data.
```

```
Found 'label' column in original data. Renaming to 'Class'.

--- Processing Synthetic Data ---
Synthetic data loaded successfully from:
/home/nhat/projectcuoiky/output/new_synthetic_malicious_data_8000_samples.csv
Synthetic data shape: (8000, 22)
Synthetic data columns (raw): ['Page', 'Encrypt', 'ObjStm', 'JS', 'JavaScript',
'AA', 'OpenAction', 'AcroForm', 'JBIG2Decode', 'RichMedia', 'Launch',
'EmbeddedFile', 'XFA', 'Colors_gt_224', 'obj', 'stream', 'xref', 'trailer',
'startxref', 'filesize_kb', 'label_numeric', 'label']
Renamed 'label' to 'Class' in synthetic data.

--- Aligning and Concatenating Data ---
Common columns for alignment (including target): ['JavaScript', 'OpenAction',
'stream', 'Class', 'startxref', 'Colors_gt_224', 'EmbeddedFile', 'AA',
'trailer', 'JS', 'XFA', 'xref', 'obj', 'filesize_kb', 'Launch', 'Page',
'Encrypt', 'JBIG2Decode', 'ObjStm', 'AcroForm', 'RichMedia']

Shape of original_aligned data: (11101, 21)
Shape of synthetic_aligned data: (8000, 21)
Shape of augmented data: (19101, 21)
Augmented data columns:
['JavaScript', 'OpenAction', 'stream', 'Class', 'startxref', 'Colors_gt_224',
'EmbeddedFile', 'AA', 'trailer', 'JS', 'XFA', 'xref', 'obj', 'filesize_kb',
'Launch', 'Page', 'Encrypt', 'JBIG2Decode', 'ObjStm', 'AcroForm', 'RichMedia']
Augmented data head (first 2 rows of original, then first 2 of synthetic part):
       JavaScript  OpenAction  stream      Class  startxref  Colors_gt_224  \
0               0           0       3     benign          2              0
1               0           0       2     benign          1              0
11101           1           1      27  malicious          3              0
11102           0           0       8  malicious          2              0

       EmbeddedFile  AA  trailer  JS  …  xref  obj  filesize_kb  Launch  \
0                 0   0        2   0  …     2   11    23.120117       0
1                 0   0        1   0  …     1    6    69.544922       0
11101             0   1        3   1  …     4   40   476.981959       1
11102             0   0        2   0  …     2   72    46.775094       0

       Page  Encrypt  JBIG2Decode  ObjStm  AcroForm  RichMedia
0         1        0            0       0         0          0
1         1        0            0       0         0          0
11101    17        0            0       0         0          0
11102     2        0            0       1         0          0

[4 rows x 21 columns]

'data' variable now refers to the augmented dataset. Final columns:
['JavaScript', 'OpenAction', 'stream', 'Class', 'startxref', 'Colors_gt_224',
```

```
'EmbeddedFile', 'AA', 'trailer', 'JS', 'XFA', 'xref', 'obj', 'filesize_kb',
'Launch', 'Page', 'Encrypt', 'JBIG2Decode', 'ObjStm', 'AcroForm', 'RichMedia']
```

[37]:
```
# Label Encoding on the combined data (original + synthetic)
# The 'data' DataFrame here should be the one combined in the previous cell␣
↪(cell 3)
le = LabelEncoder()

# Ensure 'Class' column exists and is ready for encoding
if 'Class' in data.columns:
    data["Class"] = le.fit_transform(data["Class"])
    print("'Class' column label encoded.")
else:
    print("ERROR: 'Class' column not found in the combined data for label␣
↪encoding.")
    # Handle error appropriately - perhaps stop execution or raise an error
    # For now, this will likely cause issues downstream.

# Separate features (X) and target (y) from the combined data
# Ensure TARGET_COL (defined as 'Class' in cell 3) is used consistently
if 'Class' in data.columns:
    X_combined = data.drop(['Class'], axis=1)
    y_combined = data['Class']
    print(f"Features (X_combined) shape: {X_combined.shape}")
    print(f"Target (y_combined) shape: {y_combined.shape}")

    # Verify no other potential label/target columns are left in X_combined
    # For example, if 'label' or 'label_numeric' from synthetic data were not␣
↪dropped and are not features.
    # This check depends on the exact output of cell 3.
    # Example check (you might need to adjust column names):
    potential_leaked_cols = [col for col in ['label', 'label_numeric',␣
↪'Category'] if col in X_combined.columns]
    if potential_leaked_cols:
        print(f"Warning: Potential target-related columns found in X_combined:␣
↪{potential_leaked_cols}. Consider dropping them.")
        # X_combined = X_combined.drop(columns=potential_leaked_cols,␣
↪errors='ignore') # Optional: auto-drop

else:
    print("ERROR: 'Class' column not found for X/y separation. Cannot proceed␣
↪with train/test split.")
    # Define X_combined, y_combined as empty or handle error to prevent␣
↪downstream crashes
    X_combined = pd.DataFrame()
    y_combined = pd.Series(dtype='int')
```

```
# Train-test split on the combined (augmented) data
# We will name these with the '_aug' suffix to make it clear for the next cell
if not X_combined.empty and not y_combined.empty:
    X_train_aug, X_test_aug, y_train_aug, y_test_aug = train_test_split(
        X_combined, y_combined, test_size=0.2, random_state=42,␣
↪stratify=y_combined # Stratify by y_combined
    )
    print("Train-test split performed on augmented data.")

    # Scaler - Apply only to feature sets
    scaler = MinMaxScaler()
    X_train_aug = scaler.fit_transform(X_train_aug)
    X_test_aug = scaler.transform(X_test_aug)
    print("Scaler applied to X_train_aug and X_test_aug.")

    print(f"X_train_aug shape: {X_train_aug.shape}")
    print(f"X_test_aug shape: {X_test_aug.shape}")
    print(f"y_train_aug shape: {y_train_aug.shape}")
    print(f"y_test_aug shape: {y_test_aug.shape}")
else:
    print("Skipping train-test split and scaling as X_combined or y_combined is␣
↪empty.")
    # Define placeholder empty arrays for downstream cells to avoid NameError,␣
↪though they won't be useful
    X_train_aug, X_test_aug, y_train_aug, y_test_aug = np.array([]), np.
↪array([]), np.array([]), np.array([])
```

```
'Class' column label encoded.
Features (X_combined) shape: (19101, 20)
Target (y_combined) shape: (19101,)
Train-test split performed on augmented data.
Scaler applied to X_train_aug and X_test_aug.
X_train_aug shape: (15280, 20)
X_test_aug shape: (3821, 20)
y_train_aug shape: (15280,)
y_test_aug shape: (3821,)
```

1.0.1 Training Models

[38]:
```
# Define models
models = {
    "Logistic Regression": LogisticRegression(random_state=42,␣
↪solver='liblinear'),
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "Random Forest": RandomForestClassifier(random_state=42),
    "AdaBoost": AdaBoostClassifier(random_state=42, algorithm='SAMME'), #␣
↪algorithm='SAMME' for discrete targets
```

```
    "Gradient Boosting": GradientBoostingClassifier(random_state=42),
    "XGBoost": XGBClassifier(random_state=42, use_label_encoder=False,␣
 ↪eval_metric='logloss'),
    "LightGBM": LGBMClassifier(random_state=42, verbosity=-1),
    "CatBoost": CatBoostClassifier(random_state=42, verbose=0)
    # "MLP Classifier": MLPClassifier(random_state=42, max_iter=500)
}

# Store results
results = {}
```

[39]:

```
%%time
from sklearn.model_selection import StratifiedKFold, cross_val_score
from sklearn.metrics import make_scorer, roc_auc_score, accuracy_score,␣
 ↪confusion_matrix, classification_report

# Assuming X_train_aug, y_train_aug, X_test_aug, y_test_aug are defined in the␣
 ↪preceding cell (cell 4 after data augmentation and split)
# If these variables are not defined due to an issue in cell 3 or 4 (e.g.␣
 ↪TARGET_COL not found),
# this cell might error or use empty arrays.

# Define Stratified K-Fold
n_splits = 5  # Or another number of folds you prefer, e.g., 10
stratified_kfold = StratifiedKFold(n_splits=n_splits, shuffle=True,␣
 ↪random_state=42)

# Define scoring metrics for cross-validation
scoring = {
    'accuracy': make_scorer(accuracy_score),
    'roc_auc': make_scorer(roc_auc_score, needs_proba=True) # Ensure model can␣
 ↪output proba
}

# Train and evaluate each model
for name, model in models.items():
    print(f"Training and evaluating {name}...")

    # Perform cross-validation on the (augmented) training set
    # Note: Some models like CatBoost might handle label encoding internally or␣
 ↪expect specific input types.
    # We are using the X_train_aug and y_train_aug which should be numerically␣
 ↪encoded.

    cv_accuracy_scores = []
    cv_roc_auc_scores = []
```

```
    # Check if training data is available
    if X_train_aug.shape[0] > 0 and y_train_aug.shape[0] > 0:
        try:
            # For ROC AUC, we need predict_proba. Some models might not have it␣
↪or need specific setup.
            # We'll try to get 'roc_auc'. If a model doesn't support␣
↪predict_proba, cross_val_score for roc_auc might fail.
            # We can catch this and report, or use a wrapper. For now, let's␣
↪assume models generally support it.

            print(f"  Performing {n_splits}-fold cross-validation on␣
↪X_train_aug...")
            cv_accuracy = cross_val_score(model, X_train_aug, y_train_aug,␣
↪cv=stratified_kfold, scoring='accuracy', error_score='raise')
            cv_accuracy_scores = cv_accuracy
            mean_cv_accuracy = np.mean(cv_accuracy)
            print(f"    Mean CV Accuracy: {mean_cv_accuracy:.4f}")

            # ROC AUC requires predict_proba
            if hasattr(model, "predict_proba"):
                cv_roc_auc = cross_val_score(model, X_train_aug, y_train_aug,␣
↪cv=stratified_kfold, scoring='roc_auc', error_score='raise')
                cv_roc_auc_scores = cv_roc_auc
                mean_cv_roc_auc = np.mean(cv_roc_auc)
                print(f"    Mean CV ROC AUC: {mean_cv_roc_auc:.4f}")
            else:
                mean_cv_roc_auc = np.nan # Not applicable
                print(f"    CV ROC AUC: Not applicable (model does not have␣
↪predict_proba or it failed).")

        except Exception as e:
            print(f"    Error during cross-validation for {name}: {e}")
            mean_cv_accuracy = np.nan
            mean_cv_roc_auc = np.nan
    else:
        print("  Skipping cross-validation due to empty training data.")
        mean_cv_accuracy = np.nan
        mean_cv_roc_auc = np.nan

    # Train the model on the full (augmented) training set
    if X_train_aug.shape[0] > 0 and y_train_aug.shape[0] > 0:
        print(f"  Training {name} on the full X_train_aug...")
        model.fit(X_train_aug, y_train_aug)
    else:
```

```
        print(f"  Skipping model training on full X_train_aug due to empty data.
↪")


    # Make predictions on the (augmented) test set
    y_pred_test = np.array([])
    y_pred_proba_test = np.array([])
    test_accuracy = np.nan
    test_roc_auc = np.nan
    test_cm = np.zeros((2,2)) # Placeholder
    test_report_dict = {} # Placeholder

    if X_test_aug.shape[0] > 0 and y_test_aug.shape[0] > 0 and hasattr(model,␣
↪'predict'): # Check if model was fitted
        print(f"  Evaluating {name} on X_test_aug...")
        y_pred_test = model.predict(X_test_aug)
        test_accuracy = accuracy_score(y_test_aug, y_pred_test)
        test_cm = confusion_matrix(y_test_aug, y_pred_test)
        test_report_dict = classification_report(y_test_aug, y_pred_test,␣
↪output_dict=True, zero_division=0)

        if hasattr(model, "predict_proba"):
            y_pred_proba_test = model.predict_proba(X_test_aug)[:, 1]
            test_roc_auc = roc_auc_score(y_test_aug, y_pred_proba_test)
        else:
            test_roc_auc = np.nan # Not applicable
            y_pred_proba_test = np.empty((X_test_aug.shape[0],0)) # ensure it's␣
↪an array for results dict

    else:
        print(f"  Skipping evaluation on X_test_aug due to empty test data or␣
↪model not fitted.")


    # Store results (including CV scores and test set scores)
    results[name] = {
        "Mean CV Accuracy": mean_cv_accuracy,
        "CV Accuracy Scores": cv_accuracy_scores.tolist(), # store individual␣
↪fold scores
        "Mean CV ROC AUC": mean_cv_roc_auc,
        "CV ROC AUC Scores": cv_roc_auc_scores.tolist(),
        "Test Accuracy": test_accuracy,
        "Test ROC AUC": test_roc_auc,
        "Test Confusion Matrix": test_cm,
        "Test Classification Report": test_report_dict,
```

```
        "y_pred_proba_on_test": y_pred_proba_test # Probabilities from the test␣
↪set
    }

    print(f"  Results for {name} on Test Set:")
    print(f"    Test Accuracy: {test_accuracy:.4f}")
    print(f"    Test ROC AUC: {test_roc_auc:.4f}")
    print("-" * 40)

# The rest of the notebook (plotting, etc.) will need to be adjusted
# to use these new keys in the 'results' dictionary, for example,
# 'Test Accuracy' instead of 'Accuracy', and 'Test ROC AUC' instead of 'ROC␣
↪AUC'.
# Also, y_pred_proba_on_test should be used for ROC curve plotting.
```

```
Training and evaluating Logistic Regression…
  Performing 5-fold cross-validation on X_train_aug…
    Mean CV Accuracy: 0.8437
    Mean CV ROC AUC: 0.8929
  Training Logistic Regression on the full X_train_aug…
  Evaluating Logistic Regression on X_test_aug…
  Results for Logistic Regression on Test Set:
    Test Accuracy: 0.8393
    Test ROC AUC: 0.8886
----------------------------------------
Training and evaluating Decision Tree…
  Performing 5-fold cross-validation on X_train_aug…
    Mean CV Accuracy: 0.9682
    Mean CV ROC AUC: 0.9682
  Training Decision Tree on the full X_train_aug…
  Evaluating Decision Tree on X_test_aug…
  Results for Decision Tree on Test Set:
    Test Accuracy: 0.9723
    Test ROC AUC: 0.9723
----------------------------------------
Training and evaluating Random Forest…
  Performing 5-fold cross-validation on X_train_aug…
    Mean CV Accuracy: 0.9822
    Mean CV ROC AUC: 0.9981
  Training Random Forest on the full X_train_aug…
  Evaluating Random Forest on X_test_aug…
  Results for Random Forest on Test Set:
    Test Accuracy: 0.9827
    Test ROC AUC: 0.9984
----------------------------------------
Training and evaluating AdaBoost…
  Performing 5-fold cross-validation on X_train_aug…
```

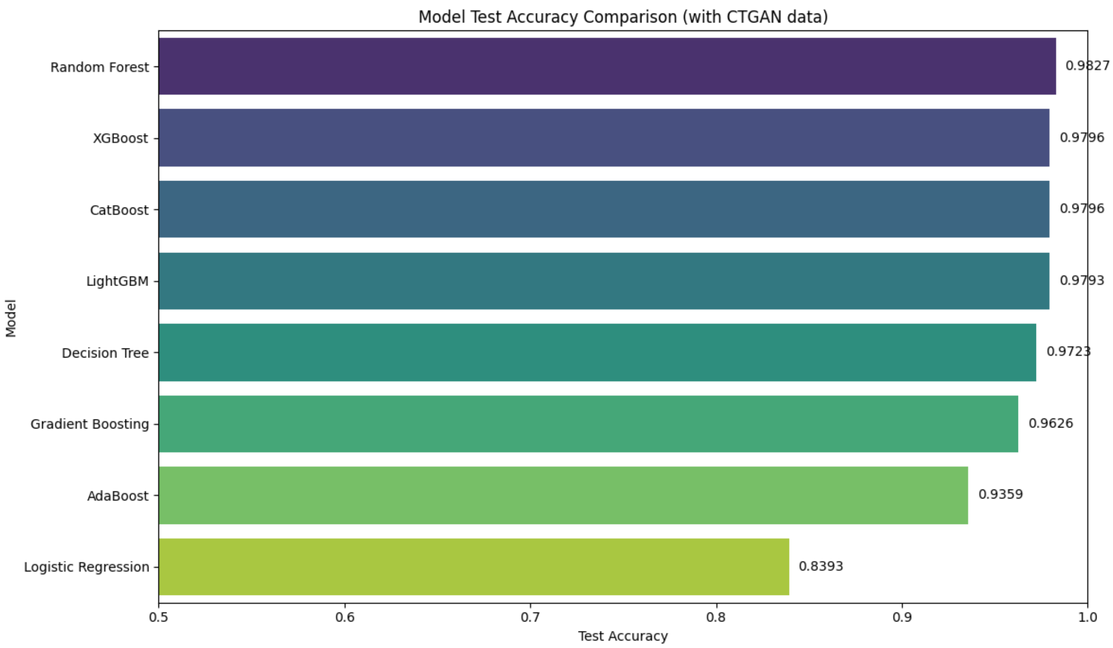
```
    Mean CV Accuracy: 0.9342
    Mean CV ROC AUC: 0.9841
  Training AdaBoost on the full X_train_aug…
  Evaluating AdaBoost on X_test_aug…
  Results for AdaBoost on Test Set:
    Test Accuracy: 0.9359
    Test ROC AUC: 0.9858
----------------------------------------
Training and evaluating Gradient Boosting…
  Performing 5-fold cross-validation on X_train_aug…
    Mean CV Accuracy: 0.9627
    Mean CV ROC AUC: 0.9946
  Training Gradient Boosting on the full X_train_aug…
  Evaluating Gradient Boosting on X_test_aug…
  Results for Gradient Boosting on Test Set:
    Test Accuracy: 0.9626
    Test ROC AUC: 0.9948
----------------------------------------
Training and evaluating XGBoost…
  Performing 5-fold cross-validation on X_train_aug…
    Mean CV Accuracy: 0.9795
    Mean CV ROC AUC: 0.9979
  Training XGBoost on the full X_train_aug…
  Evaluating XGBoost on X_test_aug…
  Results for XGBoost on Test Set:
    Test Accuracy: 0.9796
    Test ROC AUC: 0.9979
----------------------------------------
Training and evaluating LightGBM…
  Performing 5-fold cross-validation on X_train_aug…
    Mean CV Accuracy: 0.9804
    Mean CV ROC AUC: 0.9980
  Training LightGBM on the full X_train_aug…
  Evaluating LightGBM on X_test_aug…
  Results for LightGBM on Test Set:
    Test Accuracy: 0.9793
    Test ROC AUC: 0.9978
----------------------------------------
Training and evaluating CatBoost…
  Performing 5-fold cross-validation on X_train_aug…
    Mean CV Accuracy: 0.9797
    Mean CV ROC AUC: 0.9978
  Training CatBoost on the full X_train_aug…
  Evaluating CatBoost on X_test_aug…
  Results for CatBoost on Test Set:
    Test Accuracy: 0.9796
    Test ROC AUC: 0.9981
----------------------------------------
```

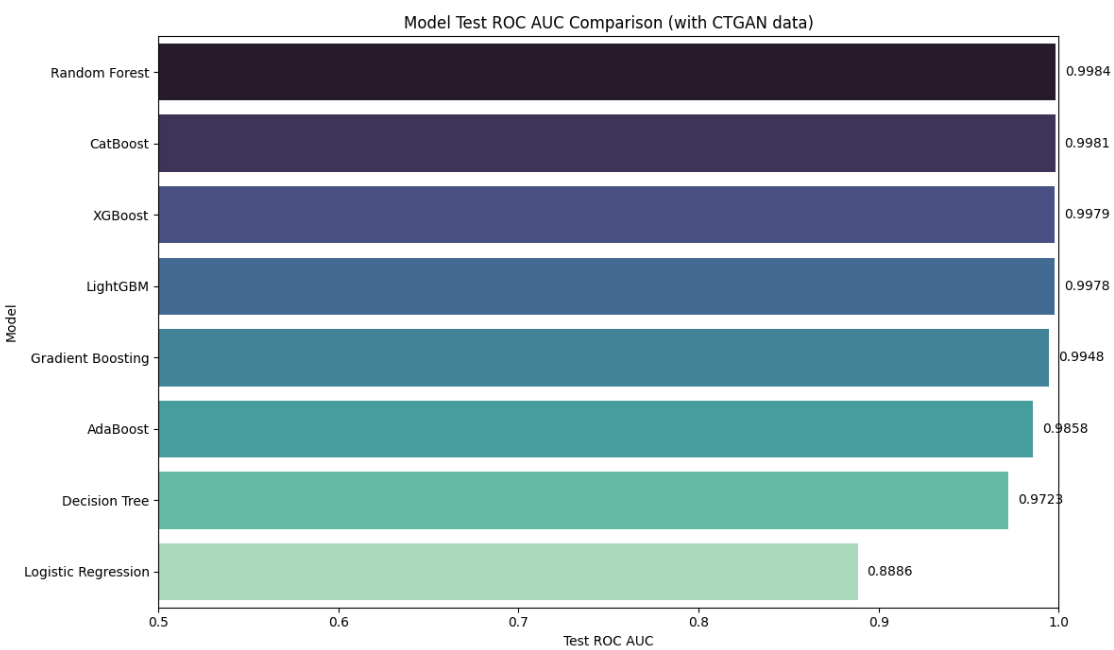
```
CPU times: user 4min 19s, sys: 2min 12s, total: 6min 32s
Wall time: 1min 6s
```

1.0.2 Compare Models

[40]:
```python
# Prepare data for plotting based on Test Set performance
# Ensure the 'results' dictionary from the previous cell (model training) is␣
↪correctly populated.

accuracy_scores = {name: res.get("Test Accuracy", float('nan')) for name, res␣
↪in results.items()}
roc_auc_scores = {name: res.get("Test ROC AUC", float('nan')) for name, res in␣
↪results.items()}

# Create a DataFrame for easy plotting
plot_df_accuracy = pd.DataFrame(list(accuracy_scores.items()),␣
↪columns=["Model", "Test Accuracy"]).sort_values(by="Test Accuracy",␣
↪ascending=False)
plot_df_roc_auc = pd.DataFrame(list(roc_auc_scores.items()), columns=["Model",␣
↪"Test ROC AUC"]).sort_values(by="Test ROC AUC", ascending=False)

# Plot Test Accuracy
plt.figure(figsize=(12, 7))
sns.barplot(x="Test Accuracy", y="Model", data=plot_df_accuracy,␣
↪palette="viridis")
plt.title("Model Test Accuracy Comparison (with CTGAN data)")
plt.xlabel("Test Accuracy")
plt.ylabel("Model")
plt.xlim(0.5, 1.0) # Adjust if accuracies are lower
for i, (model_name, acc_val) in enumerate(zip(plot_df_accuracy["Model"],␣
↪plot_df_accuracy["Test Accuracy"])):
    if pd.notna(acc_val):
        plt.text(acc_val + 0.005, i, f'{acc_val:.4f}', va='center')
plt.tight_layout()
plt.show()

# Plot Test ROC AUC
plt.figure(figsize=(12, 7))
sns.barplot(x="Test ROC AUC", y="Model", data=plot_df_roc_auc, palette="mako")
plt.title("Model Test ROC AUC Comparison (with CTGAN data)")
plt.xlabel("Test ROC AUC")
plt.ylabel("Model")
plt.xlim(0.5, 1.0) # Adjust if ROC AUCs are lower
for i, (model_name, roc_val) in enumerate(zip(plot_df_roc_auc["Model"],␣
↪plot_df_roc_auc["Test ROC AUC"])):
    if pd.notna(roc_val):
        plt.text(roc_val + 0.005, i, f'{roc_val:.4f}', va='center')
```

```
plt.tight_layout()
plt.show()

# Display results table from Test Set performance
results_summary = []
for name, res in results.items():
    # Using .get() to handle cases where a metric might be missing (e.g., if CV␣
↪failed or test data was empty)
    # The label '1' for malware might need to be confirmed if your LabelEncoder␣
↪behaves differently.
    # Check actual keys in res["Test Classification Report"] if errors occur.
    report = res.get("Test Classification Report", {})
    class_1_metrics = report.get('1', {})
    if not isinstance(class_1_metrics, dict): # Ensure it's a dictionary for .
↪get() to work
        class_1_metrics = {}

    results_summary.append({
        "Model": name,
        "Test Accuracy": res.get("Test Accuracy", float('nan')),
        "Test ROC AUC": res.get("Test ROC AUC", float('nan')),
        "Precision (Class 1)": class_1_metrics.get('precision', float('nan')),
        "Recall (Class 1)": class_1_metrics.get('recall', float('nan')),
        "F1-score (Class 1)": class_1_metrics.get('f1-score', float('nan')),
        "Mean CV Accuracy": res.get("Mean CV Accuracy", float('nan')),
        "Mean CV ROC AUC": res.get("Mean CV ROC AUC", float('nan'))
    })

results_df = pd.DataFrame(results_summary).sort_values(by="Test ROC AUC",␣
↪ascending=False)
print("\nModel Performance Summary (Based on Test Set after CTGAN augmentation):
↪")
print(results_df.to_string()) # .to_string() to print full df

# You might also want to print the CV scores per fold for a more detailed view
# for name, res in results.items():
#     print(f"\nCV Scores for {name}:")
#     print(f"  Accuracy per fold: {res.get('CV Accuracy Scores', [])}")
#     print(f"  ROC AUC per fold: {res.get('CV ROC AUC Scores', [])}")
```





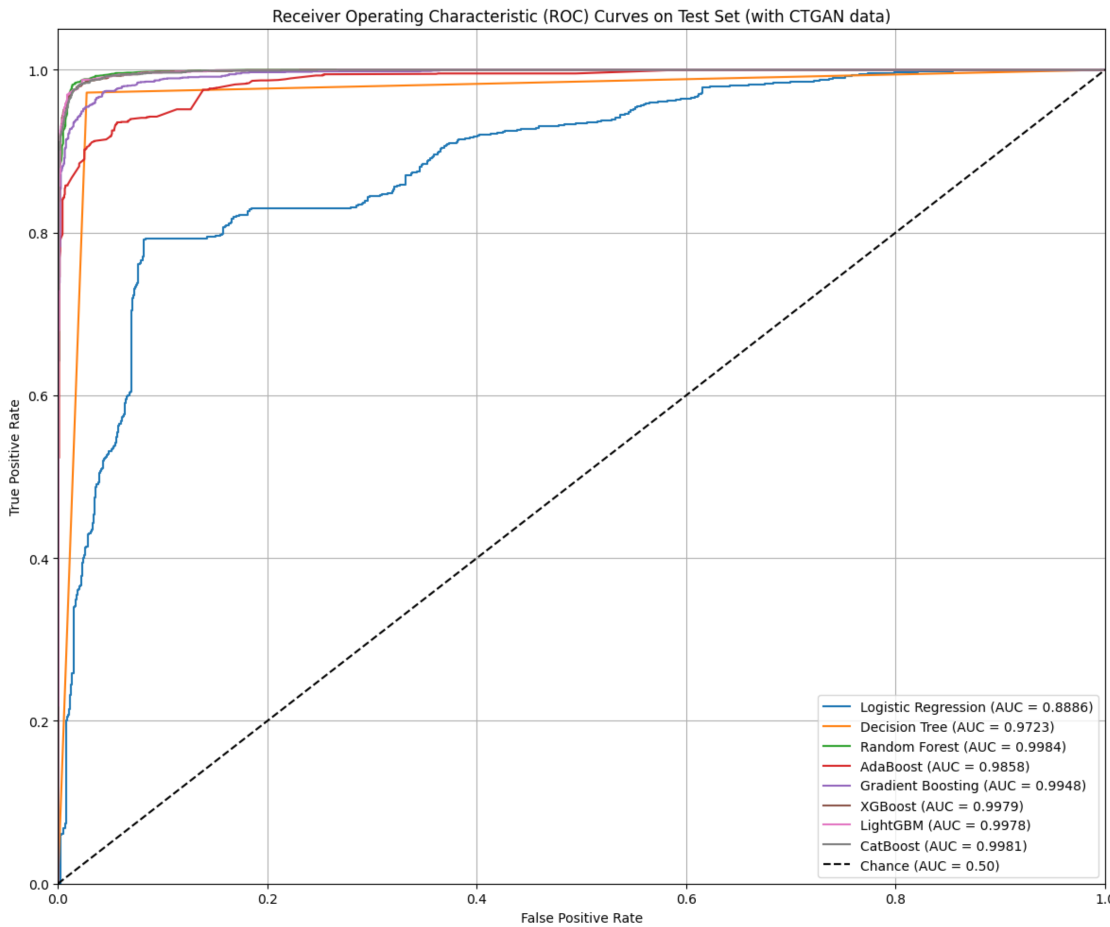

```
Model Performance Summary (Based on Test Set after CTGAN augmentation):
                 Model  Test Accuracy  Test ROC AUC  Precision (Class 1)  Recall
(Class 1)  F1-score (Class 1)  Mean CV Accuracy  Mean CV ROC AUC
2        Random Forest       0.982727      0.998354             0.987393
```

```
0.979490             0.983425          0.982199         0.998078
7              CatBoost         0.979586       0.998089                  0.984856
0.975988             0.980402          0.979712         0.997836
5               XGBoost         0.979586       0.997948                  0.983879
0.976988             0.980422          0.979450         0.997863
6              LightGBM         0.979325       0.997829                  0.984359
0.975988             0.980156          0.980432         0.997957
4     Gradient Boosting         0.962575       0.994802                  0.972987
0.954977             0.963898          0.962696         0.994564
3              AdaBoost         0.935881       0.985806                  0.949744
0.926463             0.937959          0.934228         0.984081
1         Decision Tree         0.972259       0.972272                  0.974912
0.971986             0.973447          0.968194         0.968222
0   Logistic Regression         0.839309       0.888575                  0.912940
0.765883             0.832971          0.843717         0.892888
```

[41]:

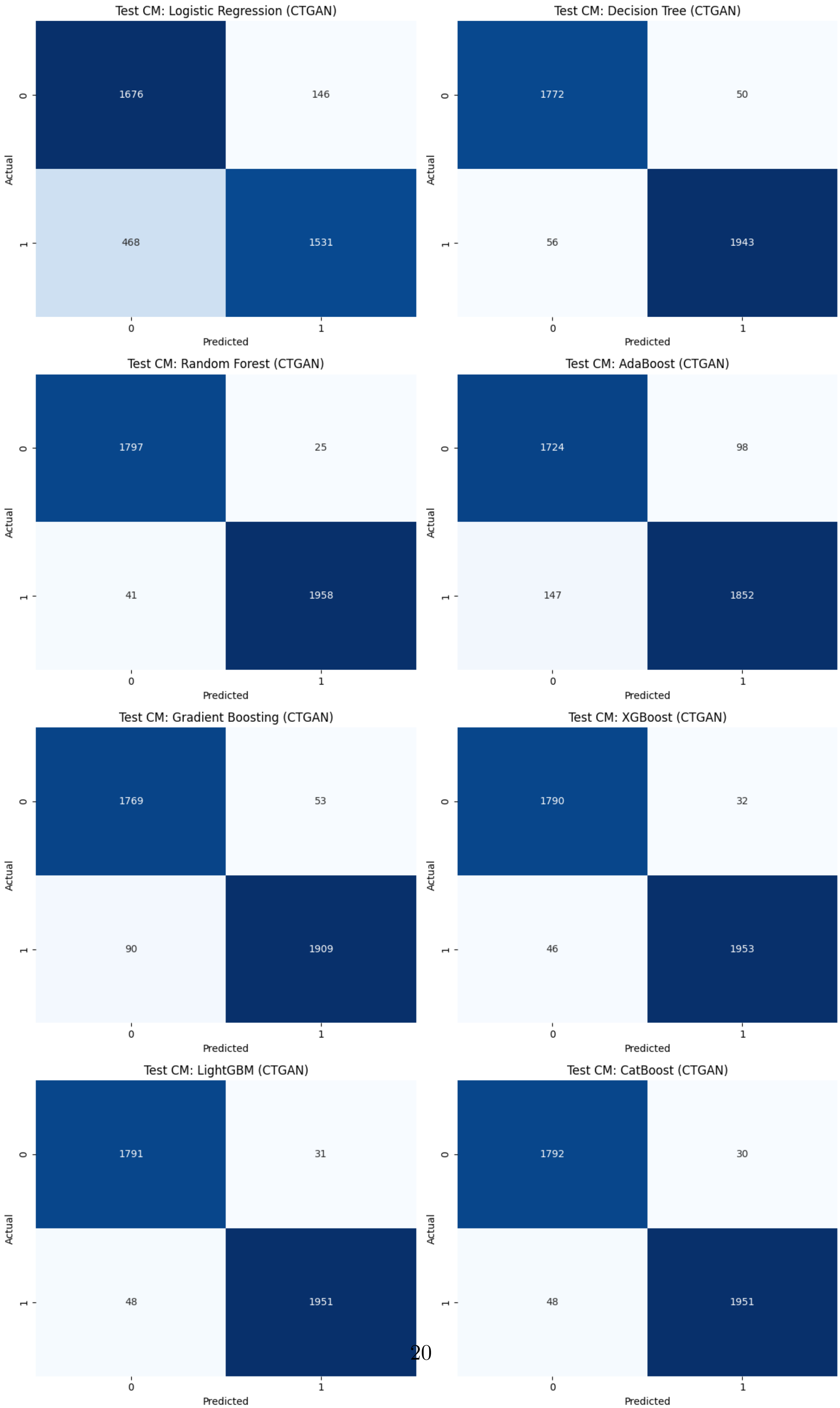
```
# Plot ROC curves for all models using predictions on the Test Set
# Assumes y_test_aug is the true labels for the test set from cell 4 (data prep/
↪split)
# and results[name]["y_pred_proba_on_test"] contains predicted probabilities␣
↪from cell 7 (model training).

plt.figure(figsize=(12, 10))

for name, res in results.items():
    # Use .get() for safety, though these should exist if training was␣
↪successful
    y_true_for_roc = y_test_aug # Ensure this is the correct variable name for␣
↪test true labels
    y_pred_proba_for_roc = res.get("y_pred_proba_on_test")
    test_roc_auc = res.get("Test ROC AUC") # Get the pre-calculated Test ROC␣
↪AUC for the label

    if y_pred_proba_for_roc is not None and len(y_pred_proba_for_roc) ==␣
↪len(y_true_for_roc) and pd.notna(test_roc_auc):
        fpr, tpr, _ = roc_curve(y_true_for_roc, y_pred_proba_for_roc)
        # Use the Test ROC AUC calculated during evaluation for consistency in␣
↪the legend
        plt.plot(fpr, tpr, label=f'{name} (AUC = {test_roc_auc:.4f})')
    elif y_pred_proba_for_roc is not None and len(y_pred_proba_for_roc) ==␣
↪len(y_true_for_roc):
        # Fallback if Test ROC AUC wasn't stored or was NaN, recalculate for␣
↪plot
        fpr, tpr, _ = roc_curve(y_true_for_roc, y_pred_proba_for_roc)
        current_auc = auc(fpr, tpr)
        plt.plot(fpr, tpr, label=f'{name} (AUC = {current_auc:.4f})')
    else:
```

```
        print(f"Skipping ROC curve for {name} due to missing/mismatched␣
 ↪probability predictions or y_test_aug.")

plt.plot([0, 1], [0, 1], 'k--', label='Chance (AUC = 0.50)') # Dashed diagonal
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curves on Test Set (with␣
 ↪CTGAN data)')
plt.legend(loc="lower right")
plt.grid(True)
plt.tight_layout()
plt.show()
```



[42]:
```
# Display confusion matrices for all models based on Test Set performance
```

```python
num_models = len(results) # Iterate over results which might be a subset of
 ↪models if some failed
if num_models == 0:
    print("No results to display confusion matrices for.")
else:
    # Determine grid size dynamically
    cols = 2
    rows = (num_models + cols - 1) // cols # Calculate rows needed

    fig, axes = plt.subplots(rows, cols, figsize=(6 * cols, 5 * rows),
 ↪squeeze=False) # squeeze=False ensures axes is always 2D
    axes = axes.flatten() # Flatten to 1D array for easy iteration

    plot_idx = 0
    for name, res in results.items():
        cm = res.get("Test Confusion Matrix")

        if cm is not None and isinstance(cm, np.ndarray) and cm.shape == (2,2):
 ↪# Basic check for a valid 2x2 CM
            sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
 ↪ax=axes[plot_idx], cbar=False)
            axes[plot_idx].set_title(f'Test CM: {name} (CTGAN)')
            axes[plot_idx].set_xlabel('Predicted')
            axes[plot_idx].set_ylabel('Actual')
            plot_idx += 1
        else:
            print(f"Skipping confusion matrix for {name} due to missing or
 ↪invalid data.")
            # Optionally, you can still use the subplot to display a message
            if plot_idx < len(axes):
                axes[plot_idx].text(0.5, 0.5, f'CM not available\nfor {name}',
 ↪ha='center', va='center')
                axes[plot_idx].axis('off') # Hide axis for blank plots
                plot_idx += 1

    # Hide any unused subplots
    for j in range(plot_idx, len(axes)):
        fig.delaxes(axes[j])

    plt.tight_layout()
    plt.show()
```


Test CM: Logistic Regression (CTGAN)
1676
146
468
1531
Test CM: Decision Tree (CTGAN)
1772
50
56
1943
Test CM: Random Forest (CTGAN)
1797
25
41
1958
Test CM: AdaBoost (CTGAN)
1724
98
147
1852
Test CM: Gradient Boosting (CTGAN)
1769
53
90
1909
Test CM: XGBoost (CTGAN)
1790
32
46
1953
Test CM: LightGBM (CTGAN)
1791
31
48
1951
Test CM: CatBoost (CTGAN)
1792
30
48
1951
Actual
Predicted
0
1

[ ]:

```

```