**Model Code and Documentation:**

1. **Introduction:**
   This project aims to develop a predictive model for forecasting Goods and Services Tax (GST) revenue using historical data and machine learning. Accurate GST predictions are crucial for effective financial planning, budgeting, and resource allocation. The model will support data-driven decision-making, helping optimize tax collection and anticipate economic trends.

2. **Key Methodology and Steps:**

   **2.1 Data Preprocessing:**

   In the preprocessing phase, the aim was to ready the anonymous dataset for model training without loss of any feature since it was not allowable. Hence, for handling the voids, we first applied mean to the relevant numerical features. Since there were no categorical variables in the dataset, there was no need for encoding. Because some models are sensitive to the scales of features, it was relevant to standardize the data to ensure that all features were on the same scale to avoid bias in learning. For instance, in the case of Logistic Regression, SVM, etc, feature magnitudes are very important. This further enhances the performance and convergence of the models. In consideration of the anonymous feature attributes no additional feature elimination or dimensionality reduction processes was performed at this stage hence assisting us to utilize the full set of attributes during model training.

   **2.1.1 Handling Missing Data:**

   Missing values are handled using techniques such as mean/mode imputation or dropping incomplete records.

   **2.1.2 Feature Scaling:**

   Features are normalized using Standard-Scaler to ensure that the model's performance is not biased by different scales of input features.

   **2.1.3 Train-Test**

   As of now we have different files for training data and test data as well. So we train our model with train data and for the testing we use test data.

   **2.1.4 Code for Data Preprocessing:**

   **#Importing Libraries**

   import pandas as pd

   import numpy as np

   **#Load Data Sets**

```
x_test = pd.read_csv('X_Test_Data_Input.csv')
y_test = pd.read_csv('Y_Test_Data_Target.csv')
x_train=pd.read_csv('X_Train_Data_Input.csv')
y_train=pd.read_csv('Y_Train_Data_Target.csv')

#Import necessary package from ski-kit Learn
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import StackingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from xgboost import XGBClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

#Dropping Unnecessary Column
x = x_train.drop(columns=['ID'])
y = y_train['target']
X = x_test.drop(columns=['ID'])
Y = y_test['target']

#Imputing Missing Values with mean and scalling using Standard Scaler
preprocessor = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='mean')),
    ('scaler', StandardScaler())
])
#Working with mix model
# Calculate the class imbalance ratio to use with XGBoost's scale_pos_weight
class_count = Counter(y)
```

```
imbalance_ratio = class_count[0] / class_count[1]


base_models = [
    ('xgb', Pipeline([('preprocessor', preprocessor), ('classifier',
XGBClassifier(scale_pos_weight=imbalance_ratio))])),
    ('rf', Pipeline([('preprocessor', preprocessor), ('classifier',
RandomForestClassifier(class_weight='balanced'))])),
    ('lr', Pipeline([('preprocessor', preprocessor), ('classifier',
LogisticRegression(class_weight='balanced'))]))
]
stacked_model = StackingClassifier(estimators=base_models,
final_estimator=meta_model)
```

**#Writing driver code for Accuracy , Confusion Matrix , AUC Score and ROC curve**

```
def predict_and_plot(inputs, targets, classifier_regressor, name=''):
    # Predict the class labels
    preds = classifier_regressor.predict(inputs)

    # Accuracy
    accuracy = accuracy_score(targets, preds)
    print("Accuracy: {:.2f}%".format(accuracy * 100))

    # Confusion Matrix
    cf = confusion_matrix(targets, preds)
    plt.figure()
    sns.heatmap(cf, annot=True, cmap='Blues', fmt='d')
    plt.xlabel('Prediction')
    plt.ylabel('Target')
    plt.title('{} Confusion Matrix'.format(name))
    plt.show()
```

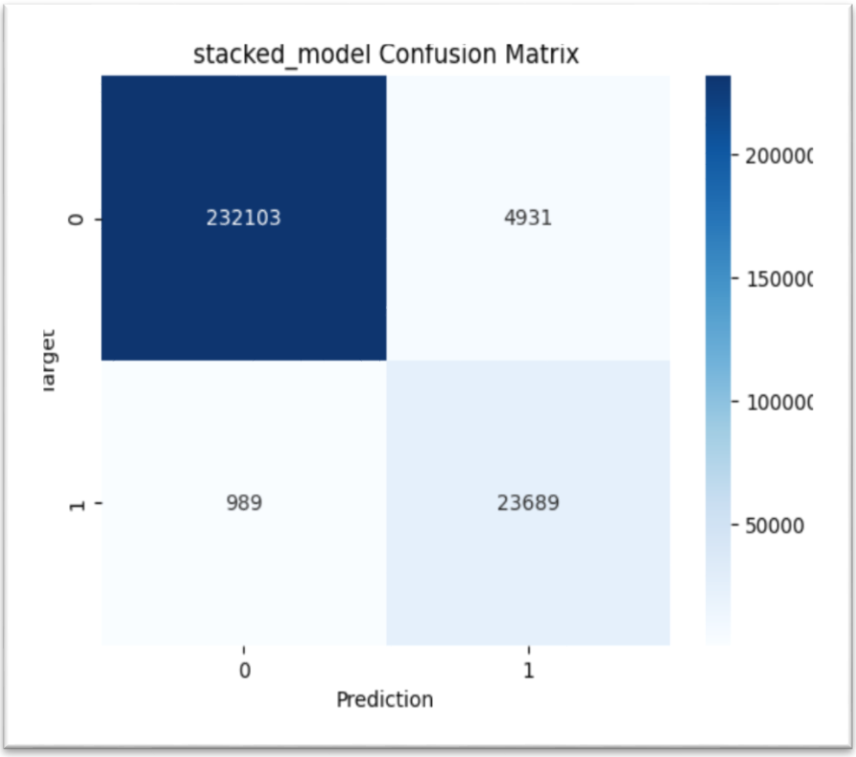
```
    # Classification Report
    print("Classification Report:")
    print(classification_report(targets, preds))

    # ROC and AUC calculation
    if hasattr(classifier_regressor, "predict_proba"):
        # Predict the probabilities of the positive class (class 1)
        probs = classifier_regressor.predict_proba(inputs)[:, 1]

        # Calculate the ROC curve and AUC score
        fpr, tpr, _ = roc_curve(targets, probs)
        auc_score = roc_auc_score(targets, probs)

        # Plot the ROC curve
        plt.figure()
        plt.plot(fpr, tpr, label=f'ROC Curve (AUC = {auc_score:.2f})')
        plt.plot([0, 1], [0, 1], 'k--')  # Diagonal line for random classifier
        plt.xlabel('False Positive Rate')
        plt.ylabel('True Positive Rate (Recall)')
        plt.title(f'{name} ROC Curve')
        plt.legend(loc="lower right")
        plt.show()

        print(f"AUC Score: {auc_score:.2f}")
    else:
        print("The classifier does not have a predict_proba method for calculating
ROC and AUC.")

    return preds
#Let's do Prediction
predict_and_plot(X, Y, stacked_model, 'stacked_model')
```

**Accuracy :** 97.74 %

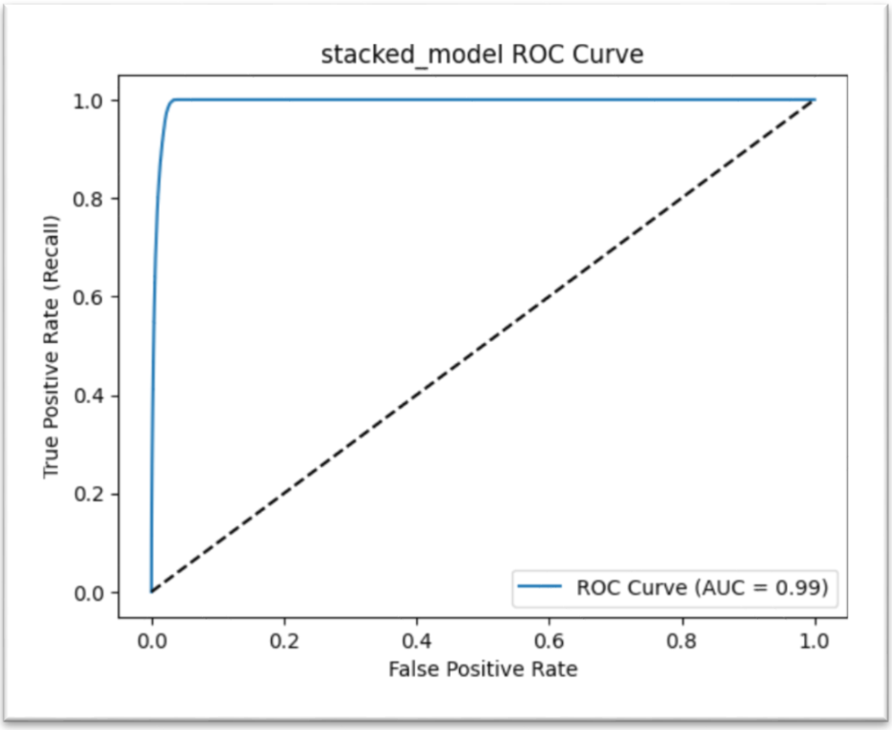
**Classification Report:**

| | PRECISION | RECALL | F1-SCORE | SUPPORT |
|---|---|---|---|---|
| 0 | 1.00 | 0.98 | 0.99 | 237034 |
| 1 | 0.83 | 0.96 | 0.89 | 24678 |
| ACCURACY | | | 0.98 | 261712 |
| MACRO AVG | 0.91 | 0.97 | 0.94 | 261712 |
| WEIGHTED AVG | 0.98 | 0.98 | 0.98 | 261712 |

**Confusion Matrix :**



**ROC Curve :**




**AUC Score :** 0.99

**ENSEMBLE MODEL**

- **Base Models - Logistic Regression, XGBoost, and Random Forest**: The ensemble model uses three diverse base models: Logistic Regression, a linear classifier; XGBoost, a powerful gradient boosting algorithm; and Random Forest, an ensemble of decision trees. This combination balances interpretability, robustness, and the ability to handle complex, non-linear relationships.
- **Ensemble Learning Approach**: By combining these three algorithms, the ensemble method leverages the strengths of each. Logistic Regression adds simplicity and interpretability, XGBoost excels in handling feature interactions and reducing bias, while Random Forest increases the robustness by averaging multiple decision trees to lower variance.
- **Stacking Model**: The stacked ensemble model was used to combine predictions from the base models. Stacking typically improves predictive performance by taking the strengths of each base model and mitigating their individual weaknesses. This hybrid approach enhances generalization.
- **AUC Score**: The stacked ensemble achieved a high AUC score of 0.99, indicating excellent discriminative ability between classes. This shows that the model performs exceptionally well in classification tasks, with high precision and recall.

- **Performance of the Ensemble**: The combination of Logistic Regression, XGBoost, and Random Forest resulted in a highly accurate model. The ensemble approach outperformed the individual base models by reducing overfitting and improving generalization.