Binary Classification of Insurance Cross Selling 💼 🔍

💡 About The data Columns :

😛 We're studying to predict which customers respond positively to an automobile insurance offer.

🥨 **Gender**: Categorical variable indicating the gender of the customer.

🌭 **Age**: Numeric variable indicating the age of the customer.

🥓 **Driving_License**: Binary variable indicating if the customer has a driving license (1 if yes, 0 if no).

🚜 **Region_Code**: Numeric variable indicating the region code of the customer.

🥡 **Previously_Insured**: Binary variable indicating if the customer was previously insured (1 if yes, 0 if no).

🖼️ **Vehicle_Age**: Categorical variable indicating the age of the vehicle.

🕸️ **Vehicle_Damage**: Categorical variable indicating if the vehicle was damaged in the past.

Annual_Premium: Numeric variable indicating the annual premium amount.

🎡 **Policy_Sales_Channel**: Numeric variable indicating the sales channel of the policy.

📶 **Vintage**: Numeric variable indicating the number of days the customer has been associated with the company.

📼 **Response**: Binary target variable indicating if the customer responded positively to the automobile insurance offer (1 if yes, 0 if no).

💡 About The Competition :

Task: The objective of this competition is to predict which customers respond positively to an automobile insurance offer..

Dataset: The dataset for this competition (both train and test) was generated from a deep learning model trained on the Health Insurance Cross Sell Prediction Data dataset. Feature distributions are close to, but not exactly the same, as the original. Feel free to use the original dataset as part of this competition, both to explore differences as well as to see whether incorporating the original in training improves model performance.

Evaluation: Submissions are evaluated using area under the ROC curve.

Submission: train.csv - the training dataset; Response is the binary target test.csv - the test dataset; your objective is to predict the probability of Response for each row sample_submission.csv - a sample submission file in the correct format

In [ ]:

```
# This Python 3 environment comes with many helpful analytics libraries insta
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will li

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that g
# You can also write temporary files to /kaggle/temp/, but they won't be save
```

```
/kaggle/input/playground-series-s4e7/sample_submission.csv
/kaggle/input/playground-series-s4e7/train.csv
/kaggle/input/playground-series-s4e7/test.csv
```

In [ ]:

```
# pip install xgboost --upgrade
```

Importing all necessary libraries

In [ ]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import random as rand
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder, OrdinalEncoder
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import RobustScaler,PowerTransformer
from sklearn.linear_model import LogisticRegression,SGDClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split,StratifiedKFold,Randomiz
from sklearn.pipeline import Pipeline
from sklearn.svm import LinearSVC
from sklearn.ensemble import RandomForestClassifier,ExtraTreesClassifier
from sklearn.metrics import *
from xgboost import XGBClassifier
import lightgbm as lgb
import warnings
warnings.filterwarnings("ignore")
```

Reading Dataset

```
In [ ]: train = pd.read_csv('/kaggle/input/playground-series-s4e7/train.csv')
        test = pd.read_csv('/kaggle/input/playground-series-s4e7/test.csv')
```

Performing Exploratory Data Analysis

```
In [ ]: print(train.shape)
        print(test.shape)
```

```
(11504798, 12)
(7669866, 11)
```

```
In [ ]: train.columns
```

```
Out[ ]: Index(['id', 'Gender', 'Age', 'Driving_License', 'Region_Code',
               'Previously_Insured', 'Vehicle_Age', 'Vehicle_Damage', 'Annual_Premiu
        m',
               'Policy_Sales_Channel', 'Vintage', 'Response'],
              dtype='object')
```

```
In [ ]: train.head()
```

Out[ ]:

	id	Gender	Age	Driving_License	Region_Code	Previously_Insured	Vehicle_Age
0	0	Male	21	1	35.0	0	1-2 Year
1	1	Male	43	1	28.0	0	> 2 Years
2	2	Female	25	1	14.0	1	< 1 Year
3	3	Female	35	1	1.0	0	1-2 Year
4	4	Female	36	1	15.0	1	1-2 Year

Deleting the Column **id** from both train and test data

```
In [ ]: train = train.drop('id',axis=1)
        test = test.drop('id',axis=1)
```

```
In [ ]: train.describe().T
```

Out[ ]:

	count	mean	std	min	25%	50
Age	11504798.0	38.383563	14.993459	20.0	24.0	36.
Driving_License	11504798.0	0.998022	0.044431	0.0	1.0	1.
Region_Code	11504798.0	26.418690	12.991590	0.0	15.0	28.
Previously_Insured	11504798.0	0.462997	0.498629	0.0	0.0	0.
Annual_Premium	11504798.0	30461.370411	16454.745205	2630.0	25277.0	31824.
Policy_Sales_Channel	11504798.0	112.425442	54.035708	1.0	29.0	151.
Vintage	11504798.0	163.897744	79.979531	10.0	99.0	166.
Response	11504798.0	0.122997	0.328434	0.0	0.0	0.

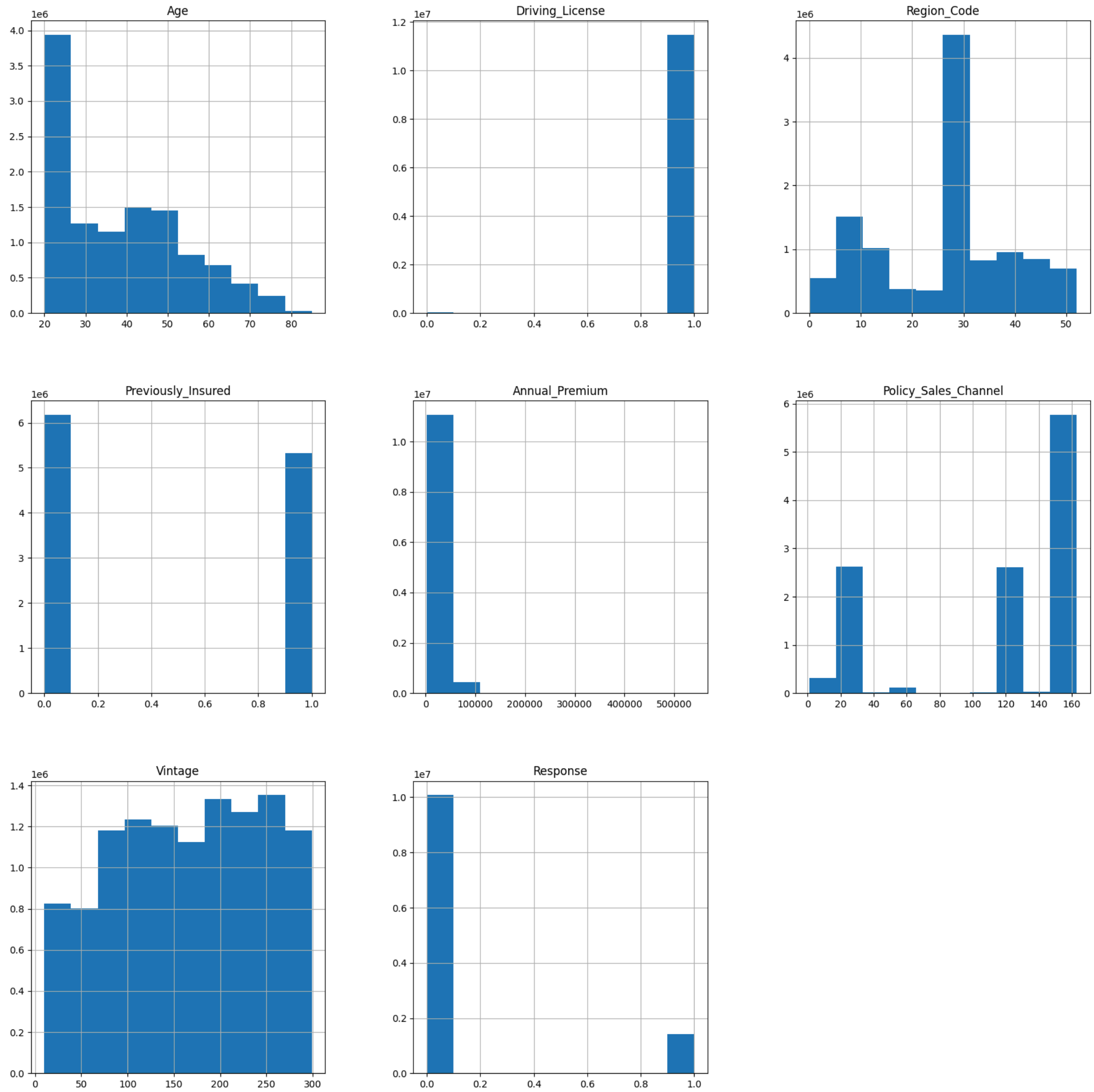
In [ ]:

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11504798 entries, 0 to 11504797
Data columns (total 11 columns):
 #   Column                Dtype  
---  ------                -----  
 0   Gender                object 
 1   Age                   int64  
 2   Driving_License       int64  
 3   Region_Code           float64
 4   Previously_Insured    int64  
 5   Vehicle_Age           object 
 6   Vehicle_Damage        object 
 7   Annual_Premium        float64
 8   Policy_Sales_Channel  float64
 9   Vintage               int64  
 10  Response              int64  
dtypes: float64(3), int64(5), object(3)
memory usage: 965.5+ MB
```

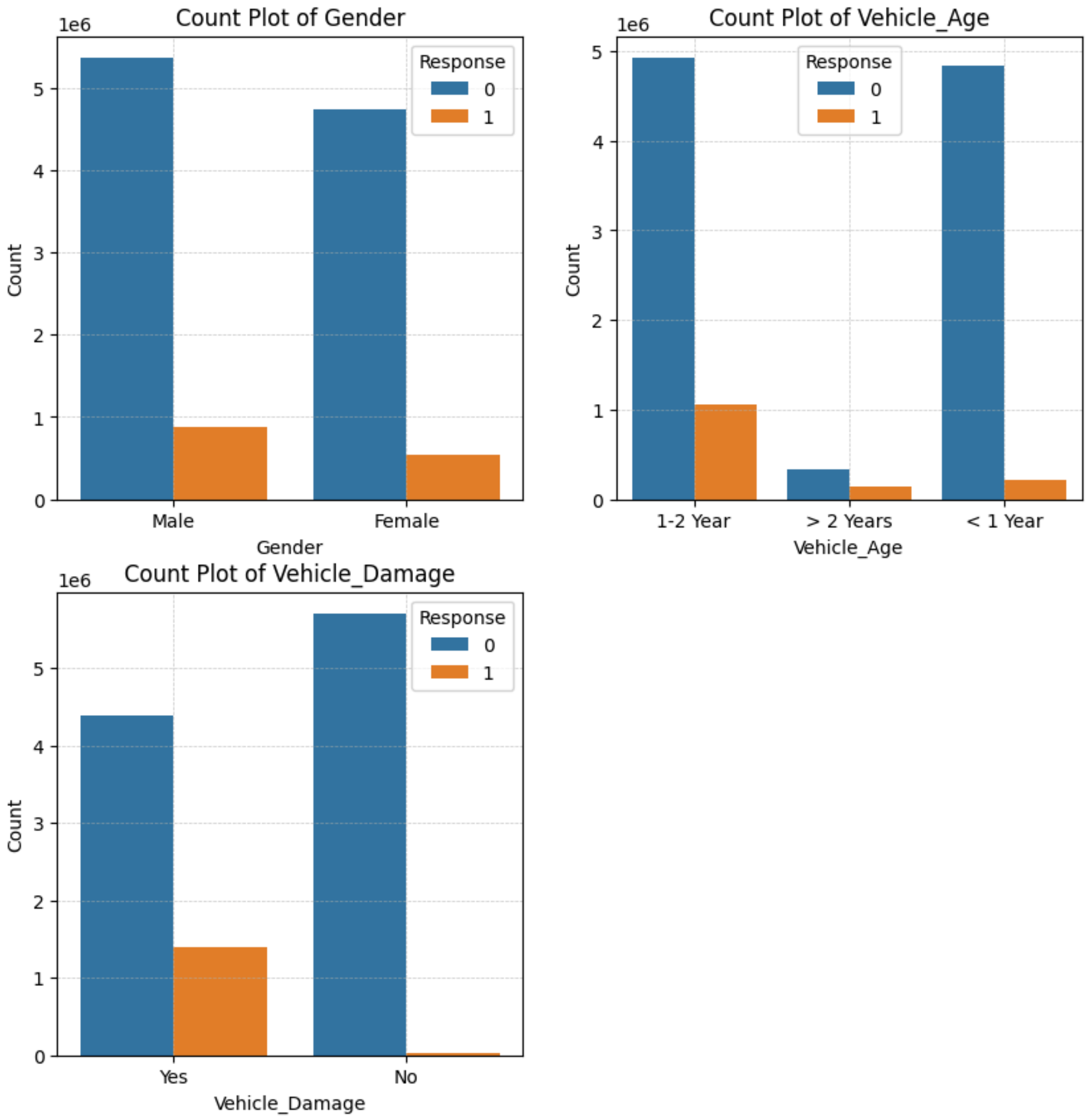
In [ ]:

```
train.hist(figsize=(20, 20));
```



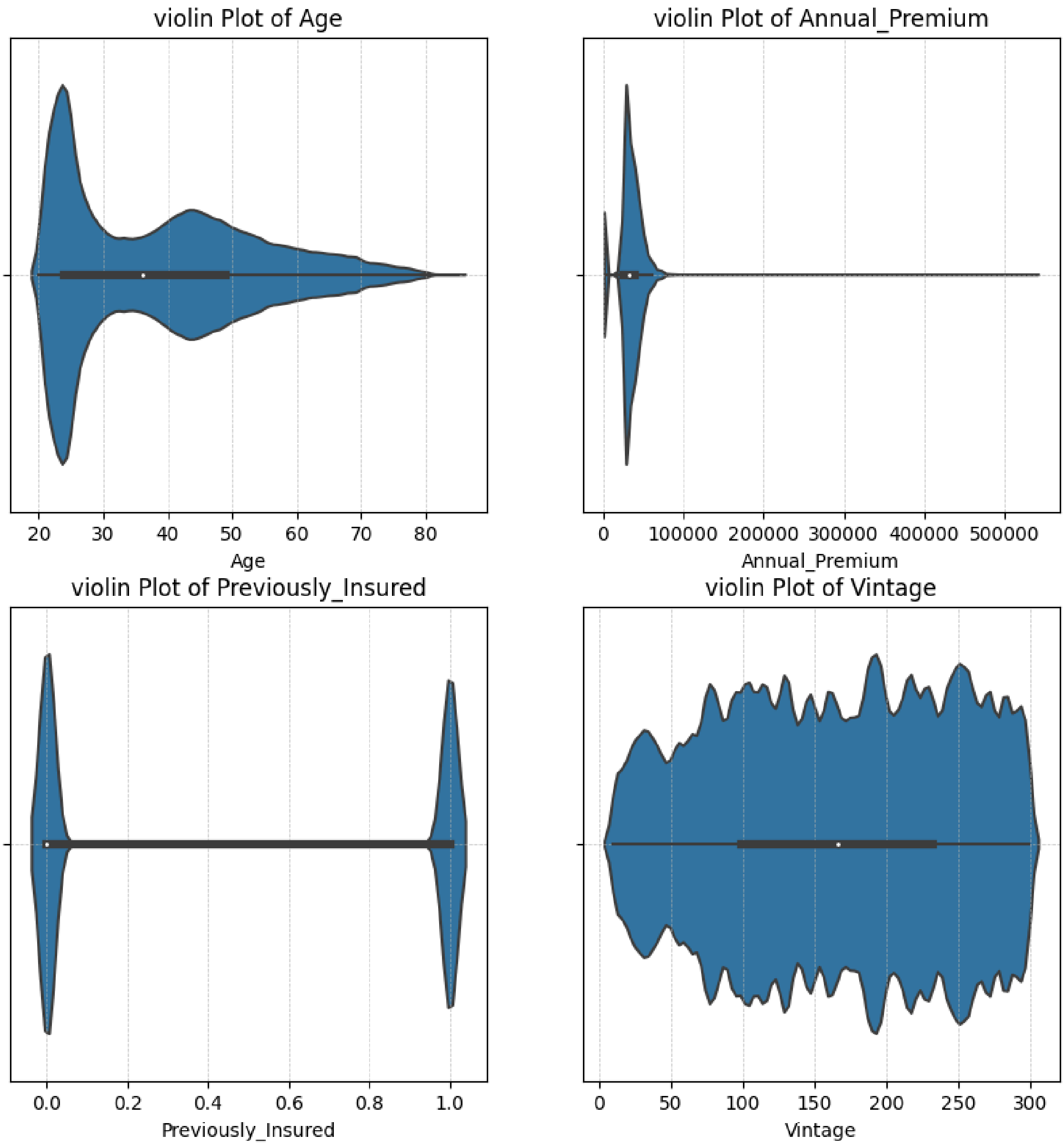
In [ ]:

```python
categorical_features = ['Gender', 'Vehicle_Age', 'Vehicle_Damage']
plt.figure(figsize=(10,10))
for i,feature in enumerate(categorical_features,1):
    plt.subplot(2, 2, i)
    sns.countplot(data=train,x = feature,hue = 'Response')
    plt.grid(True, which='both', linestyle='--', linewidth=0.5, alpha=0.7)
    plt.title(f'Count Plot of {feature}')
    plt.xlabel(feature)
    plt.ylabel('Count')
    plt.show
```



In [ ]:

```
plt.figure(figsize=(10,10))
numerical_features = ['Age', 'Annual_Premium','Previously_Insured', 'Vintage'
for i,feature in enumerate(numerical_features,1):
    plt.subplot(2, 2, i)
    sns.violinplot(data=train,x = feature,hue = 'Response')
    plt.grid(True, which='both', linestyle='--', linewidth=0.5, alpha=0.7)
    plt.title(f'violin Plot of {feature}')
    plt.xlabel(feature)
    plt.show
```



In [ ]:
```
train['Vehicle_Age'].value_counts()
```

Out[ ]:
```
Vehicle_Age
1-2 Year     5982678
< 1 Year     5044145
> 2 Years     477975
Name: count, dtype: int64
```

In [ ]:
```
correlation = train[['Driving_License', 'Response']].corr()
print(correlation)
correlation = train[['Annual_Premium', 'Response']].corr()
print(correlation)
correlation = train[['Previously_Insured', 'Response']].corr()
print(correlation)
correlation = train[['Vintage', 'Response']].corr()
print(correlation)
correlation = train[['Policy_Sales_Channel', 'Response']].corr()
print(correlation)
correlation = train[['Region_Code', 'Response']].corr()
print(correlation)
correlation = train[['Age', 'Response']].corr()
print(correlation)
```

```
                 Driving_License  Response
Driving_License         1.000000  0.009197
Response                0.009197  1.000000
                Annual_Premium  Response
Annual_Premium        1.000000  0.032261
Response              0.032261  1.000000
                    Previously_Insured  Response
Previously_Insured             1.00000  -0.34593
Response                      -0.34593   1.00000
           Vintage  Response
Vintage   1.000000 -0.015177
Response -0.015177  1.000000
                      Policy_Sales_Channel  Response
Policy_Sales_Channel              1.000000 -0.152733
Response                         -0.152733  1.000000
             Region_Code  Response
Region_Code     1.000000  0.012816
Response        0.012816  1.000000
               Age  Response
Age       1.000000  0.122134
Response  0.122134  1.000000
```

Correlation between Driving License and Response is very low.

In [ ]:
```
train['Driving_License'].value_counts()
```

Out[ ]:
```
Driving_License
1    11482041
0       22757
Name: count, dtype: int64
```

Removing Column Driving License

In [ ]:
```
train = train.drop('Driving_License',axis=1)
test = test.drop('Driving_License',axis=1)
```

Creating new features to give weightage to those values which are frequently occuring

> For Policy Sales Channel

In [ ]:
```
special_channels = train['Policy_Sales_Channel'].value_counts().nlargest(2).i

for channel in special_channels:
    new_feature = f'special_channel_{channel}'
    for df in [train, test]:
        df[new_feature] = (df["Policy_Sales_Channel"] == channel).astype("int

new_feature = 'special_channels'
for df in [train, test]:
    df[new_feature] = (
        df['Policy_Sales_Channel'].isin(special_channels)
    ).astype("int8")
```

> For Age

In [ ]:
```
new_feature = 'is_young_driver'
for df in [train, test]:
    df[new_feature] = ((df['Age'] >= 20) & (df['Age'] < 25)).astype('int8')

new_feature = 'is_old_driver'
for df in [train, test]:
    df[new_feature] = (df['Age'] > 61).astype('int8')
```

> For Region Code

In [ ]:
```
special_region = train['Region_Code'].value_counts().nlargest(2).index
new_feature = 'is_special_region'
for df in [train, test]:
    df[new_feature] = (
        df['Region_Code'].isin(special_region)
    ).astype("int8")
```

In [ ]:
```
print(train.shape)
train.head(5)
```

```
(11504798, 16)
```

Out[ ]:

	Gender	Age	Region_Code	Previously_Insured	Vehicle_Age	Vehicle_Damage	An
0	Male	21	35.0	0	1-2 Year	Yes	
1	Male	43	28.0	0	> 2 Years	Yes	
2	Female	25	14.0	1	< 1 Year	No	
3	Female	35	1.0	0	1-2 Year	Yes	
4	Female	36	15.0	1	1-2 Year	No	

In [ ]:
```
print(test.shape)
test.head(5)
```

```
(7669866, 15)
```

Out[ ]:

	Gender	Age	Region_Code	Previously_Insured	Vehicle_Age	Vehicle_Damage	An
0	Female	20	47.0	0	< 1 Year	No	
1	Male	47	28.0	0	1-2 Year	Yes	
2	Male	47	43.0	0	1-2 Year	Yes	
3	Female	22	47.0	1	< 1 Year	No	
4	Male	51	19.0	0	1-2 Year	No	

> **Preprocessing**

In [ ]:
```
categorical_features = ['Gender', 'Vehicle_Age', 'Vehicle_Damage']
numerical_features = ['Annual_Premium']
```

In [ ]:
```
X,y = train.drop('Response',axis=1), train['Response']
```

In [ ]:
```
preprocessor = ColumnTransformer(
    transformers=[('oe', OrdinalEncoder(), categorical_features),
                  ('scaler', RobustScaler(), numerical_features)],
    remainder='passthrough')

train_transformed = preprocessor.fit_transform(X)
train_transformed[0]
```

Out[ ]:
```
array([  1.        ,   0.        ,   1.        ,   2.3477494,  21.        ,
        35.        ,   0.        , 124.        , 187.        ,   0.        ,
         0.        ,   0.        ,   1.        ,   0.        ,   0.        ])
```

Applied Encoder for categorical columns and Scaler for Numerical

In [ ]:
```
test_transformed = preprocessor.transform(test)
test_transformed[0]
```

Out[ ]:
```
array([  0.        ,   1.        ,   0.        ,  -2.05968675,
        20.        ,  47.        ,   0.        , 160.        ,
       228.        ,   0.        ,   0.        ,   0.        ,
         1.        ,   0.        ,   0.        ])
```

Splitting Data into train and test which will help in validation

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(train_transformed, y, tes
```

> **Model Training**

> - Logistic Regression

In [ ]:
```
lr = LogisticRegression()
lr.fit(X_train, y_train)
y_pred = lr.predict_proba(X_test)[:,1]

score = roc_auc_score(y_test, y_pred)
print(f'Score: {score}')
```

```
Score: 0.8466604981474508
```

> - XGB Classifier

In [ ]:
```
xgb = XGBClassifier()
xgb.fit(X_train, y_train)
y_pred = xgb.predict_proba(X_test)[:,1]

score = roc_auc_score(y_test, y_pred)
print(f'Score: {score}')
```

```
Score: 0.8780805506230185
```

HyperParameter Tuning of XGB Classifier

In [ ]:
```
# xgb = XGBClassifier(
#     objective='binary:logistic',
#     eval_metric='auc',
#     device='cuda'
# )
# param_grid = {
#     'n_estimators': [1000, 1500],
#     'learning_rate': [0.01, 0.1],
#     'max_depth': [5, 10],
#     'min_child_weight': [10,20],
#     'subsample': [0.8, 0.9],
#     'colsample_bynode': [0.8, 0.9],
#     'reg_lambda': [10,20],
#     'tree_method': ['approx'],
#     'max_bin': [256, 512,1024],
# }
# random_search = RandomizedSearchCV(
#     estimator=xgb,
#     param_distributions=param_grid,

#     scoring='roc_auc',
#     cv=5,
#     verbose=1,
#     random_state=42,
#     n_jobs=-1
# )


# random_search.fit(X_train, y_train)
# print(f"Best parameters: {random_search.best_params_}")
# print(f"Best score: {random_search.best_score_}")
```

In [ ]:
```
xgb_tuned = XGBClassifier(objective='binary:logistic',
    eval_metric='auc',
    device='cuda',
    n_estimators=1500,
    learning_rate=0.1,
    max_depth=10,
    min_child_weight=25,
    subsample=0.9,
    colsample_bynode=0.9,
    reg_lambda=20,
    tree_method='approx',
    max_bin=1024

    )
xgb_tuned.fit(X_train, y_train)
y_pred = xgb_tuned.predict_proba(X_test)[:,1]

score = roc_auc_score(y_test, y_pred)
print(f'Score: {score}')
```

```
Score: 0.8838595434254661
```

LightGBM Classifier

In [ ]:
```
lgb_model = lgb.LGBMClassifier(learning_rate=0.2,metric='auc',num_leaves =  7
                               bagging_freq =10,
                               random_state=42)

lgb_model.fit(X_train, y_train)
y_pred = lgb_model.predict_proba(X_test)[:,1]

score = roc_auc_score(y_test, y_pred)
print(f'Score: {score}')
```

```
[LightGBM] [Warning] bagging_freq is set=10, subsample_freq=0 will be ignored.
Current value: bagging_freq=10
[LightGBM] [Warning] bagging_fraction is set=0.8, subsample=1.0 will be ignore
d. Current value: bagging_fraction=0.8
[LightGBM] [Warning] bagging_freq is set=10, subsample_freq=0 will be ignored.
Current value: bagging_freq=10
[LightGBM] [Warning] bagging_fraction is set=0.8, subsample=1.0 will be ignore
d. Current value: bagging_fraction=0.8
[LightGBM] [Warning] bagging_freq is set=10, subsample_freq=0 will be ignored.
Current value: bagging_freq=10
[LightGBM] [Warning] bagging_fraction is set=0.8, subsample=1.0 will be ignore
d. Current value: bagging_fraction=0.8
Score: 0.8782958954696195
```

Comparing Models

In [ ]:
```
y_probs_log_reg = lr.predict_proba(X_test)[:, 1]
y_probs_xgb = xgb.predict_proba(X_test)[:, 1]
y_probs_xg = xgb_tuned.predict_proba(X_test)[:, 1]
y_probs_lgbm = lgb_model.predict_proba(X_test)[:, 1]

# Compute ROC curve and ROC AUC score
fpr_log_reg, tpr_log_reg, _ = roc_curve(y_test, y_probs_log_reg)
fpr_xgb, tpr_xgb, _ = roc_curve(y_test, y_probs_xgb)
fpr_xg, tpr_xg, _ = roc_curve(y_test, y_probs_xg)
fpr_lgbm, tpr_lgbm, _ = roc_curve(y_test, y_probs_lgbm)

roc_auc_log_reg = auc(fpr_log_reg, tpr_log_reg)
roc_auc_xgb = auc(fpr_xgb, tpr_xgb)
roc_auc_xg = auc(fpr_xg, tpr_xg)
roc_auc_lgbm = auc(fpr_lgbm, tpr_lgbm)
```

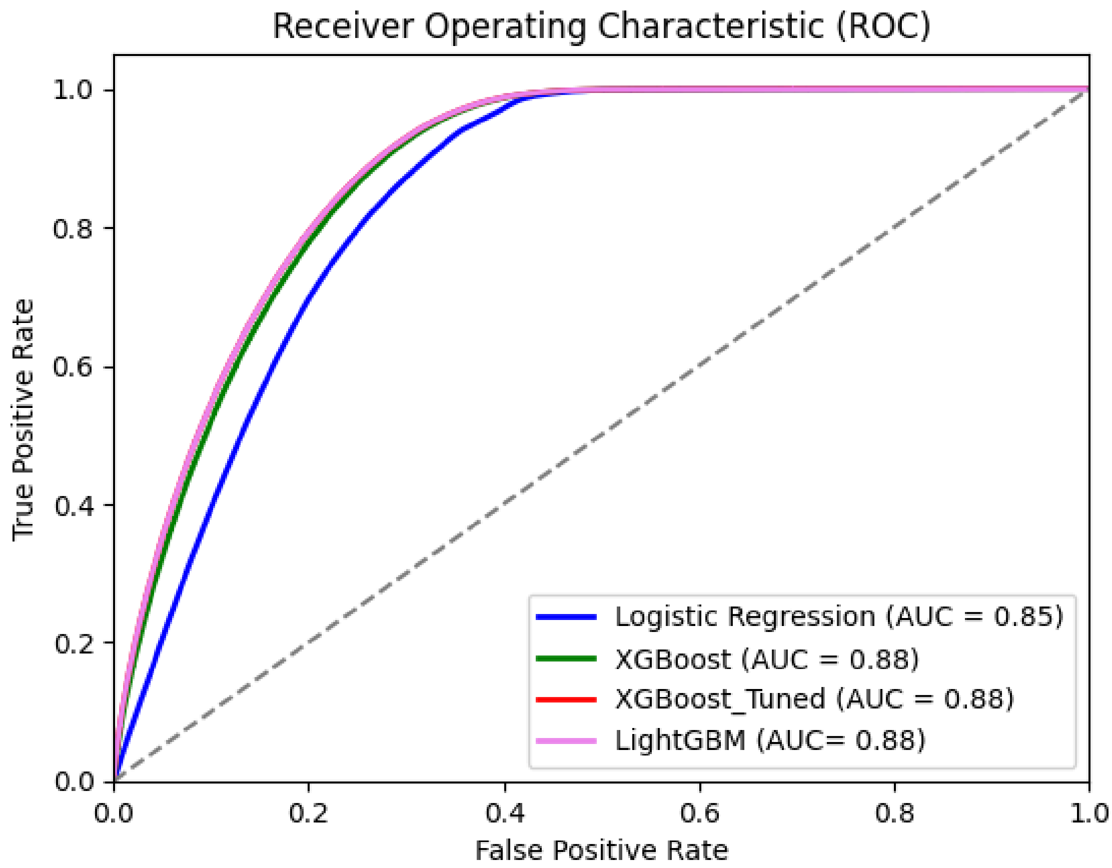
```
[LightGBM] [Warning] bagging_freq is set=10, subsample_freq=0 will be ignored.
Current value: bagging_freq=10
[LightGBM] [Warning] bagging_fraction is set=0.8, subsample=1.0 will be ignore
d. Current value: bagging_fraction=0.8
```

In [ ]:
```
plt.figure()

plt.plot(fpr_log_reg, tpr_log_reg, color='blue', lw=2, label=f'Logistic Regre
plt.plot(fpr_xgb, tpr_xgb, color='green', lw=2, label=f'XGBoost (AUC = {roc_a
plt.plot(fpr_xg, tpr_xg, color='red', lw=2, label=f'XGBoost_Tuned (AUC = {roc
plt.plot(fpr_xg, tpr_xg, color='violet', lw=2, label=f'LightGBM (AUC= {roc_au


plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
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)')
plt.legend(loc='lower right')
plt.show()
```



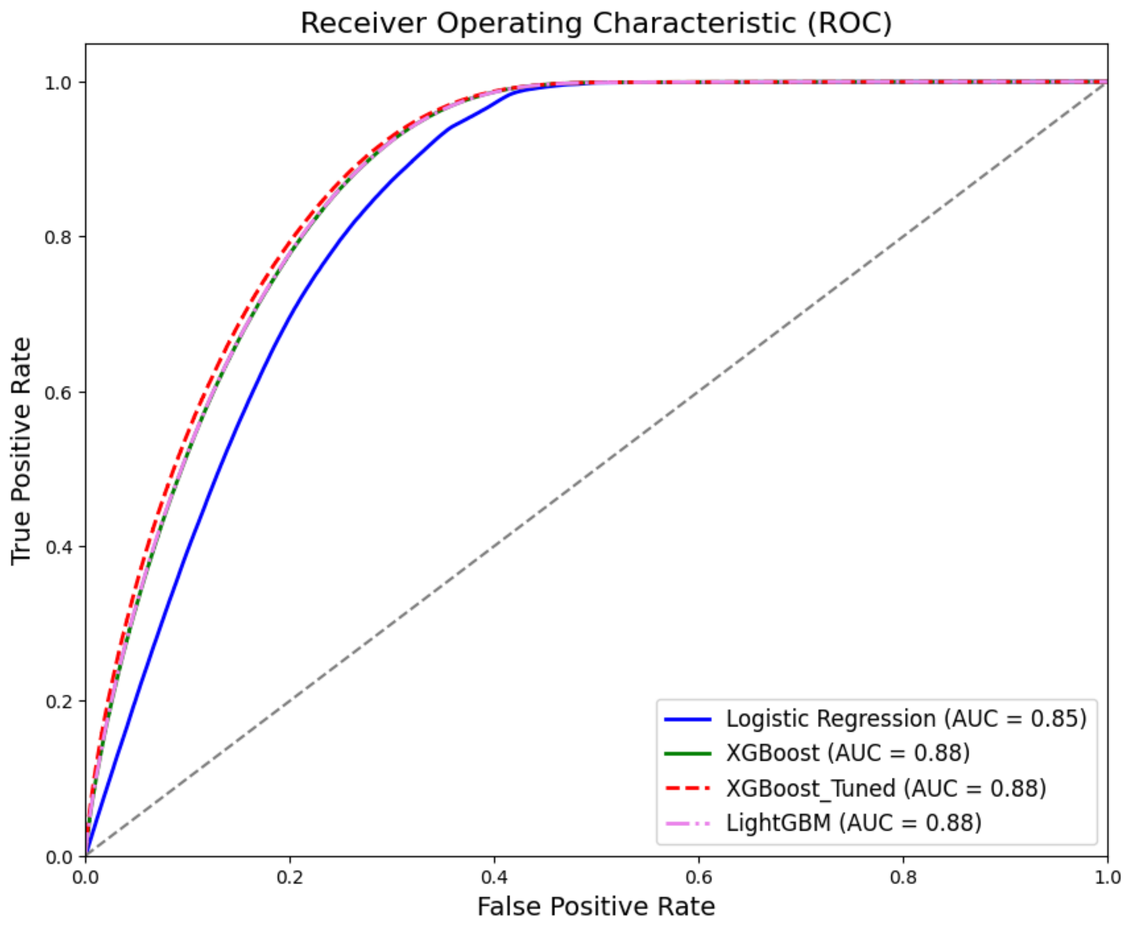
In [ ]:

```
plt.figure(figsize=(10, 8))

plt.plot(fpr_log_reg, tpr_log_reg, color='blue', lw=2, label=f'Logistic Regre
plt.plot(fpr_xgb, tpr_xgb, color='green', lw=2, label=f'XGBoost (AUC = {roc_a
plt.plot(fpr_xg, tpr_xg, color='red', lw=2, linestyle='--', label=f'XGBoost_T
plt.plot(fpr_lgbm, tpr_lgbm, color='violet', lw=2, linestyle='-.', label=f'Li

plt.plot([0, 1], [0, 1], color='gray', linestyle='--')

# Set limits and labels
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate', fontsize=14)
plt.ylabel('True Positive Rate', fontsize=14)
plt.title('Receiver Operating Characteristic (ROC)', fontsize=16)
plt.legend(loc='lower right', fontsize=12)
plt.show()
```



Creating Submission File

In [ ]:
```
test_data_1 = pd.read_csv('/kaggle/input/playground-series-s4e7/test.csv')
```

In [ ]:
```
test_predictions = xgb_tuned.predict_proba(test_transformed)
res_df = pd.DataFrame({

   'id': test_data_1['id'],
    'Response': test_predictions[:, 1]
})
```

In [ ]:
```
res_df.to_csv('XGB_tuned_sub.csv', index = False)
```

In [ ]:
```

```