OCOM510M Data Science: Assessment 2

Student ID: 201955872

1. Aims, Objectives and Plan

🎯 Aim

To develop a predictive data analysis model that can identify fraudulent insurance claims accurately, while minimizing customer churn due to false positives. The model should aim to achieve a **Balanced Error Rate (BER) close to 5%**, as per the client's business requirement.

✅ Objectives

- Understand the business case and explore the dataset.
- Preprocess the data:
  - Handle missing values, duplicates, and outliers.
  - Remove noisy or redundant features.
  - Scale, encode, and select features appropriately.
- Identify and address class imbalance in the dataset.
- Apply two machine learning techniques from the module (e.g., Logistic Regression and Random Forest).
- Integrate preprocessing and modeling into a pipeline with **Stratified Cross-Validation**.
- Optimize model hyperparameters using **GridSearchCV** or **Nested Cross-Validation**.
- Evaluate the models using performance metrics such as:
  - Balanced Error Rate (BER)
  - Precision, Recall, F1-score
  - Confusion Matrix
  - ROC-AUC and PR curves
- Estimate the financial impact of prediction errors using a custom pricing model.
- Recommend the best model based on both technical and business perspectives.

✅ Plan

2. Understanding The Case Study

🧾 Case Study Analysis

The client is an insurance company seeking to reduce financial losses due to fraudulent claims while avoiding unnecessary customer churn caused by false alarms. Their primary goal is to implement a predictive model that flags potential fraud accurately, with a **Balanced Error Rate (BER) target of 5%**, if achievable. The model must not only be technically robust but also aligned with the company's operational priorities and financial impact.

Key challenges and strategic responses:

1. **Dual Cost of Prediction Errors**
   - **False Negatives (FN)**: Undetected fraud results in direct financial loss.
   - **False Positives (FP)**: Wrongly flagged genuine customers are likely to churn, leading to lost revenue.
     ➤ To address this, we will construct a **business-focused pricing model** that quantifies both types of loss, enabling a more insightful evaluation of model performance beyond traditional accuracy.
2. **Class Imbalance**
   - Fraudulent cases are rare, leading to skewed class distribution and biased learning.
     ➤ We will perform **distributional analysis** and apply techniques such **resampling**, or **class-weight adjustments** within the modeling pipeline to correct imbalance without causing data leakage.
3. **Unbiased and Interpretable Modeling Requirement**
   - The client values transparency and balanced performance over pure complexity.
     ➤ We will prioritize interpretable models like **Logistic Regression** alongside a more powerful technique like **Random Forest**, and assess both using **precision-recall curves**, **F1-score**, and **BER**.
4. **High Risk of Overfitting**
   - With potentially many features and limited fraud cases, overfitting is a serious risk.
     ➤ Our strategy includes **Stratified K-Fold Cross-Validation**, **nested CV**, and feature reduction to ensure generalization.
5. **Operational Cost Estimation Required**
   - The business needs to understand how prediction errors affect profit margins.
     ➤ Using the average claim value and assumptions about gross profit (2× average claim), we will estimate the **required policy pricing** and **net cost of model errors**.
6. **Multiple Data Sources & Feature Quality**
   - Data may be scattered across multiple files with noise, duplicates, or irrelevant features.

➤ We will merge, clean, and evaluate feature relevance using **correlation analysis** where appropriate.

By carefully aligning our data modeling strategy with both the technical and financial dimensions of the problem, we aim to deliver a solution that is both **predictively strong and business-aware**.

3. Pre-processing applied

Create a new subheading for each stage that you do from the following items. Enter your code in the cells below the subheading.

- Merging, pivoting and melting, if necessary
- Preparing the labels appropriately, if necessary
- Dealing with missing values (imputation, filtering) without leaking, if necessary
- Dealing with duplicate values, if necessary
- Scaling, without leaking, if necessary
- Dealing with correlation and collinearity, if necessary
- Variance analysis, if necessary
- Appropriate feature selection such as RFE, if necessary
- Appropriate feature extraction, if necessary
- Identifying and dealing with class imbalance, if necessary
- Identifying and dealing with outliers, if necessary
- Categorical and numerical encoding if necessary
- Other pre-processing

In [1]:
```
import numpy as np
import pandas as pd
from pandas import DataFrame
from IPython.display import display
from functools import reduce
import matplotlib.pyplot as plt
from sklearn.preprocessing import OneHotEncoder , StandardScaler , MinMax
from sklearn.model_selection import train_test_split
from sklearn.compose import ColumnTransformer
from sklearn.compose import make_column_selector as selector
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.impute import SimpleImputer
from sklearn.metrics import classification_report , precision_recall_curv
from sklearn.model_selection import learning_curve , StratifiedKFold , cr
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import (classification_report, confusion_matrix,
                             roc_auc_score, f1_score, precision_recall_cur
                             make_scorer, roc_curve, average_precision_sco
from sklearn.calibration import calibration_curve
from joblib import parallel_backend
import time
from sklearn.feature_selection import SelectFromModel
```

3.1 - Read & Display Raw Data

In [2]:
```
raw_tc_df = pd.read_csv('./TrainData/Train_Claim.csv')
raw_td_df = pd.read_csv('./TrainData/Train_Demographics.csv')
raw_tp_df = pd.read_csv('./TrainData/Train_Policy.csv')
raw_tv_df = pd.read_csv('./TrainData/Train_Vehicle.csv')
raw_t_df = pd.read_csv('./TrainData/Traindata_with_Target.csv')

display(raw_tc_df.head())
display(raw_td_df.head())
display(raw_tp_df.head())
display(raw_tv_df.head())
display(raw_t_df.head())
```

	CustomerID	DateOfIncident	TypeOfIncident	TypeOfCollission	SeverityOfIncident
0	Cust10000	2015-02-03	Multi-vehicle Collision	Side Collision	Total Loss
1	Cust10001	2015-02-02	Multi-vehicle Collision	Side Collision	Total Loss
2	Cust10002	2015-01-15	Single Vehicle Collision	Side Collision	Minor Damage
3	Cust10003	2015-01-19	Single Vehicle Collision	Side Collision	Minor Damage
4	Cust10004	2015-01-09	Single Vehicle Collision	Rear Collision	Minor Damage

	CustomerID	InsuredAge	InsuredZipCode	InsuredGender	InsuredEducationLevel
0	Cust10000	35	454776	MALE	JD
1	Cust10001	36	454776	MALE	JD
2	Cust10002	33	603260	MALE	JD
3	Cust10003	36	474848	MALE	JD
4	Cust10004	29	457942	FEMALE	High School

	InsurancePolicyNumber	CustomerLoyaltyPeriod	DateOfPolicyCoverage	InsuranceI
0	110122	328	2014-10-17	
1	110125	256	1990-05-25	
2	110126	228	2014-06-06	
3	110127	256	2006-10-12	
4	110128	137	2000-06-04	

	CustomerID	VehicleAttribute	VehicleAttributeDetails
0	Cust20179	VehicleID	Vehicle8898
1	Cust21384	VehicleModel	Malibu
2	Cust33335	VehicleMake	Toyota
3	Cust27118	VehicleModel	Neon
4	Cust13038	VehicleID	Vehicle30212

	CustomerID	ReportedFraud
0	Cust20065	N
1	Cust37589	N
2	Cust24312	N
3	Cust5493	Y
4	Cust7704	Y

3.2 - Merging, pivoting and melting, if necessary

In [3]:
```
# Checkig if CustomerId is Repeating i.e value_count > 1 , to see if we n
display('Train Claim' , raw_tc_df['CustomerID'].value_counts()[lambda x :
display('Train Demographics',raw_td_df['CustomerID'].value_counts()[lambd
display('Train Policy',raw_tp_df['CustomerID'].value_counts()[lambda x :
display('Train Vehicle',raw_tv_df['CustomerID'].value_counts()[lambda x :
```

```
'Train Claim'
Series([], Name: count, dtype: int64)
'Train Demographics'
Series([], Name: count, dtype: int64)
'Train Policy'
Series([], Name: count, dtype: int64)
'Train Vehicle'
CustomerID
Cust20179    4
Cust23045    4
Cust3818     4
Cust7461     4
Cust16944    4
            ..
Cust30090    4
Cust9783     4
Cust20478    4
Cust35879    4
Cust15237    4
Name: count, Length: 28836, dtype: int64
```

3.2.1 **Pivoting** Concluded from above

Only in `train_vehicle` table customerId are repeating

In [4]:
```
# Pivot Train Vehicle Table
pvt_tv_df = raw_tv_df.pivot_table(index='CustomerID',columns='VehicleAttr
```

```
# Check null values
pvt_tv_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 28836 entries, 0 to 28835
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype 
---  ------        --------------  ----- 
 0   CustomerID    28836 non-null  object
 1   VehicleID     28836 non-null  object
 2   VehicleMake   28836 non-null  object
 3   VehicleModel  28836 non-null  object
 4   VehicleYOM    28836 non-null  object
dtypes: object(5)
memory usage: 1.1+ MB
```

In [5]:
```
# Look Info to understand non-nulls and Dtype

display(raw_tc_df.info())
display(raw_td_df.info())
display(raw_tp_df.info())
display(pvt_tv_df.info())
display(raw_t_df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 28836 entries, 0 to 28835
Data columns (total 19 columns):
 #   Column                 Non-Null Count  Dtype 
---  ------                 --------------  ----- 
 0   CustomerID             28836 non-null  object
 1   DateOfIncident         28836 non-null  object
 2   TypeOfIncident         28836 non-null  object
 3   TypeOfCollission       28836 non-null  object
 4   SeverityOfIncident     28836 non-null  object
 5   AuthoritiesContacted   26144 non-null  object
 6   IncidentState          28836 non-null  object
 7   IncidentCity           28836 non-null  object
 8   IncidentAddress        28836 non-null  object
 9   IncidentTime           28836 non-null  int64 
 10  NumberOfVehicles       28836 non-null  int64 
 11  PropertyDamage         28836 non-null  object
 12  BodilyInjuries         28836 non-null  int64 
 13  Witnesses              28836 non-null  object
 14  PoliceReport           28836 non-null  object
 15  AmountOfTotalClaim     28836 non-null  object
 16  AmountOfInjuryClaim    28836 non-null  int64 
 17  AmountOfPropertyClaim  28836 non-null  int64 
 18  AmountOfVehicleDamage  28836 non-null  int64 
dtypes: int64(6), object(13)
memory usage: 4.2+ MB
None
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 28836 entries, 0 to 28835
Data columns (total 10 columns):
 #   Column                Non-Null Count  Dtype 
---  ------                --------------  ----- 
 0   CustomerID            28836 non-null  object
 1   InsuredAge            28836 non-null  int64 
 2   InsuredZipCode        28836 non-null  int64 
 3   InsuredGender         28806 non-null  object
 4   InsuredEducationLevel  28836 non-null  object
 5   InsuredOccupation     28836 non-null  object
 6   InsuredHobbies        28836 non-null  object
 7   CapitalGains          28836 non-null  int64 
 8   CapitalLoss           28836 non-null  int64 
 9   Country               28834 non-null  object
dtypes: int64(4), object(6)
memory usage: 2.2+ MB
None
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 28836 entries, 0 to 28835
Data columns (total 10 columns):
 #   Column                     Non-Null Count  Dtype  
---  ------                     --------------  -----  
 0   InsurancePolicyNumber      28836 non-null  int64  
 1   CustomerLoyaltyPeriod      28836 non-null  int64  
 2   DateOfPolicyCoverage       28836 non-null  object 
 3   InsurancePolicyState       28836 non-null  object 
 4   Policy_CombinedSingleLimit  28836 non-null  object 
 5   Policy_Deductible          28836 non-null  int64  
 6   PolicyAnnualPremium        28836 non-null  float64
 7   UmbrellaLimit              28836 non-null  int64  
 8   InsuredRelationship        28836 non-null  object 
 9   CustomerID                 28836 non-null  object 
dtypes: float64(1), int64(4), object(5)
memory usage: 2.2+ MB
None
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 28836 entries, 0 to 28835
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype 
---  ------        --------------  ----- 
 0   CustomerID    28836 non-null  object
 1   VehicleID     28836 non-null  object
 2   VehicleMake   28836 non-null  object
 3   VehicleModel  28836 non-null  object
 4   VehicleYOM    28836 non-null  object
dtypes: object(5)
memory usage: 1.1+ MB
None
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 28836 entries, 0 to 28835
Data columns (total 2 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   CustomerID     28836 non-null  object
 1   ReportedFraud  28836 non-null  object
dtypes: object(2)
memory usage: 450.7+ KB
None
```

3.2.2 Merging

All Tables have `28836` records and `no null` values in `any` table

In [6]:
```
## Merge all 5 Tables on CustomerId
print('Train Claims Shape:' , raw_tc_df.shape)
print('Train Demographic Shape:' , raw_td_df.shape)
print('Train Policy Shape:' , raw_tp_df.shape)
print('Train Vehicle Shape:' , pvt_tv_df.shape)
print('Train Target Shape:' , raw_t_df.shape)

merged_df:DataFrame = reduce(lambda left,right : pd.merge(left,right,how=

print('Merged Table Shape:' , merged_df.shape)
print('Total Columns in Merged is equal to adition of all coulmns from in

pd.set_option('display.max_columns', 100)
merged_df.head()
```

```
Train Claims Shape: (28836, 19)
Train Demographic Shape: (28836, 10)
Train Policy Shape: (28836, 10)
Train Vehicle Shape: (28836, 5)
Train Target Shape: (28836, 2)
Merged Table Shape: (28836, 42)
Total Columns in Merged is equal to adition of all coulmns from individual
tables ?  True
```

Out[6]:

	CustomerID	DateOfIncident	TypeOfIncident	TypeOfCollission	SeverityOfIncide
0	Cust10000	2015-02-03	Multi-vehicle Collision	Side Collision	Total Los
1	Cust10001	2015-02-02	Multi-vehicle Collision	Side Collision	Total Los
2	Cust10002	2015-01-15	Single Vehicle Collision	Side Collision	Minor Damag
3	Cust10003	2015-01-19	Single Vehicle Collision	Side Collision	Minor Damag
4	Cust10004	2015-01-09	Single Vehicle Collision	Rear Collision	Minor Damag

In [7]:
```
# Look for missing and duplicate
merged_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 28836 entries, 0 to 28835
Data columns (total 42 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   CustomerID                    28836 non-null  object
 1   DateOfIncident                28836 non-null  object
 2   TypeOfIncident                28836 non-null  object
 3   TypeOfCollission              28836 non-null  object
 4   SeverityOfIncident            28836 non-null  object
 5   AuthoritiesContacted          26144 non-null  object
 6   IncidentState                 28836 non-null  object
 7   IncidentCity                  28836 non-null  object
 8   IncidentAddress               28836 non-null  object
 9   IncidentTime                  28836 non-null  int64
 10  NumberOfVehicles              28836 non-null  int64
 11  PropertyDamage                28836 non-null  object
 12  BodilyInjuries                28836 non-null  int64
 13  Witnesses                     28836 non-null  object
 14  PoliceReport                  28836 non-null  object
 15  AmountOfTotalClaim            28836 non-null  object
 16  AmountOfInjuryClaim           28836 non-null  int64
 17  AmountOfPropertyClaim         28836 non-null  int64
 18  AmountOfVehicleDamage         28836 non-null  int64
 19  InsuredAge                    28836 non-null  int64
 20  InsuredZipCode                28836 non-null  int64
 21  InsuredGender                 28806 non-null  object
 22  InsuredEducationLevel         28836 non-null  object
 23  InsuredOccupation             28836 non-null  object
 24  InsuredHobbies                28836 non-null  object
 25  CapitalGains                  28836 non-null  int64
 26  CapitalLoss                   28836 non-null  int64
 27  Country                       28834 non-null  object
 28  InsurancePolicyNumber         28836 non-null  int64
 29  CustomerLoyaltyPeriod         28836 non-null  int64
 30  DateOfPolicyCoverage          28836 non-null  object
 31  InsurancePolicyState          28836 non-null  object
 32  Policy_CombinedSingleLimit    28836 non-null  object
 33  Policy_Deductible             28836 non-null  int64
 34  PolicyAnnualPremium           28836 non-null  float64
 35  UmbrellaLimit                 28836 non-null  int64
 36  InsuredRelationship           28836 non-null  object
 37  VehicleID                     28836 non-null  object
 38  VehicleMake                   28836 non-null  object
 39  VehicleModel                  28836 non-null  object
 40  VehicleYOM                    28836 non-null  object
 41  ReportedFraud                 28836 non-null  object
dtypes: float64(1), int64(14), object(27)
memory usage: 9.2+ MB
```

3.3 - Dealing with duplicate values

In [8]:
```python
print('Before Deduplicate' , merged_df.shape)
merged_df.drop_duplicates(inplace=True)
print('After Deduplicate' , merged_df.shape)
```

```
Before Deduplicate (28836, 42)
After Deduplicate (28836, 42)
```

3.4 - Preparing the labels appropriately

In [9]:
```
print(merged_df['ReportedFraud'].value_counts(dropna=False))
print(merged_df['ReportedFraud'].value_counts(normalize=True))
```

```
ReportedFraud
N    21051
Y     7785
Name: count, dtype: int64
ReportedFraud
N    0.730025
Y    0.269975
Name: proportion, dtype: float64
```

Above results suggest Target label is `imbalanced` with `Y is ~26% and N is ~73%` and no missing value

In [10]:
```
# Encode the Labels into binary
merged_df['ReportedFraud'] = merged_df['ReportedFraud'].map({'N' : 0 , 'Y
```

3.5 - Dealing with datatype conversions

- DateOfIncident from Object to Date Format
- Witnesses from Object to int64
- DateOfPolicyCoverage from Object to Date Format
- Policy_CombinedSingleLimit Split into 2 features
- VehicleYOM from Object to int64

In [11]:
```
# Utility Class for Type conversion
class ColumnTypeConverter:
    def __init__(self, column_name, target_type='datetime', errors='coerc
        self.column_name = column_name
        self.target_type = target_type
        self.errors = errors

    def transform(self, df):
        if self.column_name not in df.columns:
            raise ValueError(f"Column '{self.column_name}' not found in D

        original_non_nulls = df[self.column_name].notna().sum()

        if self.target_type == 'datetime':
            df[self.column_name] = pd.to_datetime(df[self.column_name], e
        elif self.target_type == 'int64':
            df[self.column_name] = pd.to_numeric(df[self.column_name], er
        elif self.target_type == 'float64':
            df[self.column_name] = pd.to_numeric(df[self.column_name], er
        else:
            raise ValueError("target_type must be one of: 'datetime', 'in

        converted_non_nulls = df[self.column_name].notna().sum()
        failed_conversions = original_non_nulls - converted_non_nulls

        print(f"[{self.column_name}] Conversion Summary to '{self.target_
        print(f"✅ Successfully converted: {converted_non_nulls}")
```

```
        print(f"❌ Failed conversions (NaT or NaN): {failed_conversions}"

        return df
```

In [12]:
```
dateOfIncident = ColumnTypeConverter('DateOfIncident',target_type='dateti
dateOfPolicyCoverage = ColumnTypeConverter('DateOfPolicyCoverage',target_
witnesses = ColumnTypeConverter('Witnesses',target_type='float64',errors=
vehicleYom = ColumnTypeConverter('VehicleYOM',target_type='int64',errors=
amountOfTotalClaim = ColumnTypeConverter('AmountOfTotalClaim',target_type

dateOfIncident.transform(merged_df)
dateOfPolicyCoverage.transform(merged_df)
witnesses.transform(merged_df)
vehicleYom.transform(merged_df)
amountOfTotalClaim.transform(merged_df)
merged_df.info()
```

```
[DateOfIncident] Conversion Summary to 'datetime':
✅ Successfully converted: 28836
❌ Failed conversions (NaT or NaN): 0
[DateOfPolicyCoverage] Conversion Summary to 'datetime':
✅ Successfully converted: 28836
❌ Failed conversions (NaT or NaN): 0
[Witnesses] Conversion Summary to 'float64':
✅ Successfully converted: 28790
❌ Failed conversions (NaT or NaN): 46
[VehicleYOM] Conversion Summary to 'int64':
✅ Successfully converted: 28836
❌ Failed conversions (NaT or NaN): 0
[AmountOfTotalClaim] Conversion Summary to 'float64':
✅ Successfully converted: 28786
❌ Failed conversions (NaT or NaN): 50
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 28836 entries, 0 to 28835
Data columns (total 42 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   CustomerID                 28836 non-null  object
 1   DateOfIncident             28836 non-null  datetime64[ns]
 2   TypeOfIncident             28836 non-null  object
 3   TypeOfCollission           28836 non-null  object
 4   SeverityOfIncident         28836 non-null  object
 5   AuthoritiesContacted       26144 non-null  object
 6   IncidentState              28836 non-null  object
 7   IncidentCity               28836 non-null  object
 8   IncidentAddress            28836 non-null  object
 9   IncidentTime               28836 non-null  int64
 10  NumberOfVehicles           28836 non-null  int64
 11  PropertyDamage             28836 non-null  object
 12  BodilyInjuries             28836 non-null  int64
 13  Witnesses                  28790 non-null  float64
 14  PoliceReport               28836 non-null  object
 15  AmountOfTotalClaim         28786 non-null  float64
 16  AmountOfInjuryClaim        28836 non-null  int64
 17  AmountOfPropertyClaim      28836 non-null  int64
 18  AmountOfVehicleDamage      28836 non-null  int64
 19  InsuredAge                 28836 non-null  int64
 20  InsuredZipCode             28836 non-null  int64
 21  InsuredGender              28806 non-null  object
 22  InsuredEducationLevel      28836 non-null  object
 23  InsuredOccupation          28836 non-null  object
 24  InsuredHobbies             28836 non-null  object
 25  CapitalGains               28836 non-null  int64
 26  CapitalLoss                28836 non-null  int64
 27  Country                    28834 non-null  object
 28  InsurancePolicyNumber      28836 non-null  int64
 29  CustomerLoyaltyPeriod      28836 non-null  int64
 30  DateOfPolicyCoverage       28836 non-null  datetime64[ns]
 31  InsurancePolicyState       28836 non-null  object
 32  Policy_CombinedSingleLimit 28836 non-null  object
 33  Policy_Deductible          28836 non-null  int64
 34  PolicyAnnualPremium        28836 non-null  float64
 35  UmbrellaLimit              28836 non-null  int64
 36  InsuredRelationship        28836 non-null  object
 37  VehicleID                  28836 non-null  object
 38  VehicleMake                28836 non-null  object
 39  VehicleModel               28836 non-null  object
```

```
 40  VehicleYOM                    28836 non-null  Int64
 41  ReportedFraud                 28836 non-null  int64
dtypes: Int64(1), datetime64[ns](2), float64(3), int64(15), object(21)
memory usage: 9.3+ MB
```

✅ Successful Conversions

- **DateOfIncident** and **DateOfPolicyCoverage**:
  Fully converted to `datetime64[ns]`.
- **VehicleYOM**:
  All values successfully converted to `Int64`.
- **Witnesses**:
  Mostly successful; **46 missing** (`NaT` or `NaN`).

3.6 - 🕒 Date Feature Engineering

To enhance model performance and extract meaningful temporal patterns from the `DateOfIncident` and `DateOfPolicyCoverage` columns, the following date-based features were engineered:

✅ Extracted from `DateOfIncident`

Feature Name	Description
`IncidentYear`	Year in which the incident occurred
`IncidentMonth`	Month (1–12) of the incident
`IncidentDay`	Day of the month when the incident occurred
`IncidentWeekDay`	Day of the week (0=Monday, 6=Sunday)
`IncidentWeek`	ISO calendar week number
`IncidentIsOnWeekend`	Binary flag indicating if the incident happened on a weekend (`1` = Saturday/Sunday)

🔁 Derived from `DateOfIncident` and `DateOfPolicyCoverage`

Feature Name	Description
`DaysSincePolicyStart`	Number of days between policy coverage start date and the date of incident. Reflects policy age at time of incident.

🗑️ Dropping Raw Date Columns

After extracting meaningful features from the datetime columns `DateOfIncident` and `DateOfPolicyCoverage`, the original columns were no longer needed.

❌ Dropped Columns

Column	Reason for Removal
`DateOfIncident`	Replaced by derived features like `IncidentMonth`, `IncidentWeekday`, and `DaysSincePolicyStart`
`DateOfPolicyCoverage`	Used only to compute `DaysSincePolicyStart`

In [13]:
```
merged_df['IncidentYear'] = merged_df['DateOfIncident'].dt.year
merged_df['IncidentMonth'] = merged_df['DateOfIncident'].dt.month
merged_df['IncidentDay'] = merged_df['DateOfIncident'].dt.day
merged_df['IncidentWeekDay'] = merged_df['DateOfIncident'].dt.weekday
merged_df['IncidentWeek'] = merged_df['DateOfIncident'].dt.isocalendar().
merged_df['IncidentIsOnWeekend'] = merged_df['IncidentWeekDay'].isin([5,6
merged_df['DaysSincePolicyStart'] = (merged_df['DateOfIncident'] - merged

merged_df.drop(columns=['DateOfIncident','DateOfPolicyCoverage'],inplace=
```

In [14]:
```
merged_df.head()
```

Out[14]:

	CustomerID	TypeOfIncident	TypeOfCollission	SeverityOfIncident	AuthoritiesCc
0	Cust10000	Multi-vehicle Collision	Side Collision	Total Loss	
1	Cust10001	Multi-vehicle Collision	Side Collision	Total Loss	
2	Cust10002	Single Vehicle Collision	Side Collision	Minor Damage	
3	Cust10003	Single Vehicle Collision	Side Collision	Minor Damage	
4	Cust10004	Single Vehicle Collision	Rear Collision	Minor Damage	

3.7 - ✅ Features Engineering from `Policy_CombinedSingleLimit`

The original column was split into two new float columns:

Column	Non-Null Count	Dtype	Description
`LimitPerPerson`	28836	float64	Insurance limit per person
`LimitPerAccident`	28836	float64	Insurance limit per accident

- All values successfully converted to `float64`
- No missing values detected
- Original column `Policy_CombinedSingleLimit` was dropped after transformation

In [15]:
```
# Split Policy_CombinedSingleLimit into 2 columns
merged_df[['LimitPerPerson', 'LimitPerAccident']] = merged_df['Policy_Com
merged_df[['LimitPerPerson', 'LimitPerAccident']].info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 28836 entries, 0 to 28835
Data columns (total 2 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   LimitPerPerson    28836 non-null  float64
 1   LimitPerAccident  28836 non-null  float64
dtypes: float64(2)
memory usage: 450.7 KB
```

3.8 - Redundancy Analysis of `AmountOfTotalClaim`

In [16]:
```python
# 1. Calculate total
merged_df['CalculatedTotal'] = (
    merged_df['AmountOfInjuryClaim'] +
    merged_df['AmountOfPropertyClaim'] +
    merged_df['AmountOfVehicleDamage']
)

# 2. Create a boolean mask for rows with all finite values (no NaN or inf
valid_rows = merged_df[['CalculatedTotal', 'AmountOfTotalClaim']].applyma

# 3. Compare only valid rows using np.isclose
mismatches = merged_df.loc[valid_rows, :][~np.isclose(
    merged_df.loc[valid_rows, 'CalculatedTotal'],
    merged_df.loc[valid_rows, 'AmountOfTotalClaim'],
    rtol=1e-5
)]

# 4. Show results
if mismatches.empty:
    print("✅ All valid rows match — safe to drop `AmountOfTotalClaim`.")
else:
    print(f"⚠️ {len(mismatches)} mismatches found in valid rows.")
    print(mismatches[['AmountOfInjuryClaim', 'AmountOfPropertyClaim', 'Am
                      'CalculatedTotal', 'AmountOfTotalClaim']].head())

# 5. Drop columns
# merged_df.drop(columns=['AmountOfTotalClaim', 'CalculatedTotal'], inpla
```

```
✅ All valid rows match — safe to drop `AmountOfTotalClaim`.
```

```
/var/folders/2q/4h8qwwsx6t5446pyq1l03mjh0000gn/T/ipykernel_47248/230708010
9.py:9: FutureWarning: DataFrame.applymap has been deprecated. Use DataFra
me.map instead.
  valid_rows = merged_df[['CalculatedTotal', 'AmountOfTotalClaim']].applym
ap(np.isfinite).all(axis=1)
```

3.8.1 🔧 Feature Engineering: of `AmountOfTotalClaim`

Upon inspection, the feature `AmountOfTotalClaim` was found to be a **linear sum** of the following three features:

- `AmountOfInjuryClaim`
- `AmountOfPropertyClaim`
- `AmountOfVehicleDamage`

We validated this relationship across the dataset:

💡 Decision:

- Since this feature does not provide **new information** and introduces **perfect multicollinearity**, it was **dropped** from the dataset before modeling.

In [17]:
```
merged_df.drop(columns=['AmountOfTotalClaim', 'CalculatedTotal'], inplace
```

3.9 - 🧹 Missing Value Handling Strategy

Based on the attribute documentation for the `CSE9099c` dataset, multiple missing value indicators are used across different columns. Below is a detailed plan to clean and standardize missing values **before modeling**.

🔍 Step 1: Replace Custom Missing Indicators

🟨 Demographics

Column	Missing Indicator	Suggested Replacement
`InsuredGender`	"NA"	Replace with `'Unknown'`
`Country`	NaN (2 missing)	Replace with `'Unknown'`

🟦 Policy Information

Column	Missing Indicator	Suggested Replacement
`PolicyAnnualPremium`	`-1`	Replace with `np.nan`
`TotalCharges`	`"MISSINGVAL"`	Replace with `np.nan`
`ContractType`	`"NA"`	Replace with `'Unknown'`

🟥 Claim Information

Column	Missing Indicator	Suggested Replacement
`TypeOfCollission`	`"?"`	Replace with `'Unknown'`
`PropertyDamage`	`"?"`	Replace with `'Unknown'`
`PoliceReport`	`"?"`	Replace with `'Unknown'`
`IncidentTime`	`-5`	Replace with `np.nan`
`Witnesses`	`"MISSINGVALUE"`	Replace with `np.nan`
`AuthoritiesContacted`	`Nan`	Replace with `Unknown`

🟪 Vehicle Data

Column	Missing Indicator	Suggested Replacement
`VehicleAttributeDetails`	`"???"`	Replace with `'Unknown'`
`Vehiclemake`	`"???"`	Replace with `'Unknown'`

In [18]:
```
merged_df.replace({
    'InsuredGender': {'NA': 'Unknown'},
    'ContractType': {'NA': 'Unknown'},
    'TypeOfCollission': {'?': 'Unknown'},
    'PropertyDamage': {'?': 'Unknown'},
    'PoliceReport': {'?': 'Unknown'},
    'TotalCharges': {'MISSINGVAL': np.nan},
    'Witnesses': {'MISSINGVALUE': np.nan},
    'VehicleAttributeDetails': {'???': 'Unknown'},
    'VehicleMake':{'???': 'Unknown'}
}, inplace=True)

# Replace numeric placeholder values
merged_df['IncidentTime'] = merged_df['IncidentTime'].replace(-5, np.nan)
merged_df['PolicyAnnualPremium'] = merged_df['PolicyAnnualPremium'].repla

merged_df['AuthoritiesContacted'] = merged_df['AuthoritiesContacted'].fil

merged_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 28836 entries, 0 to 28835
Data columns (total 48 columns):
 #   Column                      Non-Null Count  Dtype  
---  ------                      --------------  -----  
 0   CustomerID                  28836 non-null  object 
 1   TypeOfIncident              28836 non-null  object 
 2   TypeOfCollission            28836 non-null  object 
 3   SeverityOfIncident          28836 non-null  object 
 4   AuthoritiesContacted        28836 non-null  object 
 5   IncidentState               28836 non-null  object 
 6   IncidentCity                28836 non-null  object 
 7   IncidentAddress             28836 non-null  object 
 8   IncidentTime                28805 non-null  float64
 9   NumberOfVehicles            28836 non-null  int64  
 10  PropertyDamage              28836 non-null  object 
 11  BodilyInjuries              28836 non-null  int64  
 12  Witnesses                   28790 non-null  float64
 13  PoliceReport                28836 non-null  object 
 14  AmountOfInjuryClaim         28836 non-null  int64  
 15  AmountOfPropertyClaim       28836 non-null  int64  
 16  AmountOfVehicleDamage       28836 non-null  int64  
 17  InsuredAge                  28836 non-null  int64  
 18  InsuredZipCode              28836 non-null  int64  
 19  InsuredGender               28806 non-null  object 
 20  InsuredEducationLevel       28836 non-null  object 
 21  InsuredOccupation           28836 non-null  object 
 22  InsuredHobbies              28836 non-null  object 
 23  CapitalGains                28836 non-null  int64  
 24  CapitalLoss                 28836 non-null  int64  
 25  Country                     28834 non-null  object 
 26  InsurancePolicyNumber       28836 non-null  int64  
 27  CustomerLoyaltyPeriod       28836 non-null  int64  
 28  InsurancePolicyState        28836 non-null  object 
 29  Policy_CombinedSingleLimit  28836 non-null  object 
 30  Policy_Deductible           28836 non-null  int64  
 31  PolicyAnnualPremium         28695 non-null  float64
 32  UmbrellaLimit               28836 non-null  int64  
 33  InsuredRelationship         28836 non-null  object 
 34  VehicleID                   28836 non-null  object 
 35  VehicleMake                 28836 non-null  object 
 36  VehicleModel                28836 non-null  object 
 37  VehicleYOM                  28836 non-null  Int64  
 38  ReportedFraud               28836 non-null  int64  
 39  IncidentYear                28836 non-null  int32  
 40  IncidentMonth               28836 non-null  int32  
 41  IncidentDay                 28836 non-null  int32  
 42  IncidentWeekDay             28836 non-null  int32  
 43  IncidentWeek                28836 non-null  UInt32 
 44  IncidentIsOnWeekend         28836 non-null  int64  
 45  DaysSincePolicyStart        28836 non-null  int64  
 46  LimitPerPerson              28836 non-null  float64
 47  LimitPerAccident            28836 non-null  float64
dtypes: Int64(1), UInt32(1), float64(5), int32(4), int64(16), object(21)
memory usage: 10.1+ MB
```

3.10 - 🔍 Unique Value Count Analysis

To understand the structure and distribution of values across columns, we calculated the number of unique values in each column using the following code:

🎯 Why This Matters

Analyzing unique value counts helps in:

- **Identifying High-Cardinality Features** Columns like CustomerID, VehicleID, or IncidentAddress may have thousands of unique values, which are not ideal for one-hot encoding and can increase dimensionality unnecessarily.
- **Spotting Low-Cardinality Categorical Features** Features with only a few unique values (e.g., Gender, PropertyDamage, PoliceReport) are perfect candidates for one-hot encoding.
- **Detecting Constant or Near-Constant Columns** Columns with only one unique value offer no variability and can be safely dropped.

In [19]:
```
# Only do it for dtype =object
unique_counts_df = pd.DataFrame({
    'Column': merged_df.select_dtypes(include='object').columns,
    'UniqueValues': [merged_df[col].nunique(dropna=False) for col in merg
}).sort_values(by='UniqueValues', ascending=False)

# Only do it for dtype  not object i.e numeric
unique_numeric_counts_df = pd.DataFrame({
    'Column': merged_df.select_dtypes(exclude='object').columns,
    'UniqueValues': [merged_df[col].nunique(dropna=False) for col in merg
}).sort_values(by='UniqueValues', ascending=False)

print('Object type' , unique_counts_df)
print('Non Object type' , unique_numeric_counts_df)
```

```
Object type                          Column  UniqueValues
0                      CustomerID     28836
18                      VehicleID     28836
7                 IncidentAddress      1000
20                   VehicleModel        39
13                 InsuredHobbies        20
19                    VehicleMake        15
12              InsuredOccupation        14
16  Policy_CombinedSingleLimit            9
5                   IncidentState         7
6                    IncidentCity         7
11          InsuredEducationLevel         7
17            InsuredRelationship         6
4           AuthoritiesContacted          5
1                  TypeOfIncident         4
2                 TypeOfCollission        4
3                SeverityOfIncident       4
15          InsurancePolicyState          3
9                    PoliceReport         3
8                  PropertyDamage         3
10                  InsuredGender         3
14                        Country         2
Non Object type                     Column  UniqueValues
11  InsurancePolicyNumber         28836
14    PolicyAnnualPremium         23852
6   AmountOfVehicleDamage         20041
4     AmountOfInjuryClaim         11958
5   AmountOfPropertyClaim         11785
24   DaysSincePolicyStart          7358
15          UmbrellaLimit          7089
13      Policy_Deductible          1496
8          InsuredZipCode           995
12  CustomerLoyaltyPeriod           479
10            CapitalLoss           354
9            CapitalGains           338
7              InsuredAge            46
20            IncidentDay            31
0            IncidentTime            25
16             VehicleYOM            21
22           IncidentWeek            11
21        IncidentWeekDay             7
3               Witnesses             5
1        NumberOfVehicles             4
19          IncidentMonth             3
2          BodilyInjuries             3
25         LimitPerPerson             3
26       LimitPerAccident             3
17          ReportedFraud             2
23    IncidentIsOnWeekend             2
18           IncidentYear             1
```

3.11 - 📊 Unique Value Count Summary

The number of unique values was calculated for each column to guide **feature engineering, encoding decisions, and dimensionality reduction**.

Column	Notes (based on earlier unique value counts)
`TypeOfIncident`	4 values – ✅ encode
`TypeOfCollission`	4 values – ✅ encode
`SeverityOfIncident`	4 values – ✅ encode
`AuthoritiesContacted`	5 values – ✅ encode
`PropertyDamage`	3 values – ✅ encode
`PoliceReport`	3 values – ✅ encode
`InsuredGender`	3 values – ✅ encode
`InsuredEducationLevel`	7 values – ✅ encode
`InsuredOccupation`	14 values – ✅ encode
`InsuredHobbies`	20 values – ✅ encode
`InsurancePolicyState`	3 values – ✅ encode
`InsuredRelationship`	6 values – ✅ encode
`VehicleMake`	15 values – ⚠️ encode or group (depends on model)
`VehicleModel`	39 values – ⚠️ maybe encode top N, group rest -- but will leave it for timebeing
`ReportedFraud`	🎯 This is the target — do **not encoded**, just map `{‘Y’:1, ‘N’:0}`

❌ Constant Features (to be dropped)

Column	Unique Values
`IncidentYear`	1
`Country`	2 (2 Unnkown and rest India')

❌ Hight Cardinality Features (to be dropped)

Column	Reason
`IncidentAddress`	1000 unique values — likely not useful for encoding
`CustomerID`	High cardinality (28836 values) — drop or treat specially
`VehicleID`	High cardinality (28836 values) — drop or treat specially

This unique value audit helps decide:

- What to drop
- What to one-hot encode
- What to normalize or transform

In [20]:
```
## Print Vehicke Make and Model to decide how to group them if needed to
print(merged_df['VehicleMake'].value_counts())
print(merged_df['VehicleModel'].value_counts())
print(merged_df['InsuredHobbies'].value_counts())
```

```
VehicleMake
Saab          2415
Suburu        2313
Nissan        2300
Dodge         2263
Chevrolet     2174
Ford          2158
Accura        2099
BMW           2073
Toyota        1981
Volkswagen    1960
Audi          1952
Jeep          1946
Mercedes      1659
Honda         1493
Unknown         50
Name: count, dtype: int64
VehicleModel
RAM               1344
Wrangler          1261
A3                1102
MDX               1054
Jetta             1037
Neon               928
Pathfinder         919
Passat             888
Legacy             887
92x                859
Malibu             828
95                 820
A5                 812
F150               797
Forrestor          784
Camry              771
Tahoe              736
93                 724
Maxima             722
Grand Cherokee     718
Escape             706
Ultima             698
E400               695
X5                 691
TL                 684
Silverado          668
Fusion             650
Highlander         633
Civic              604
ML350              599
Impreza            562
CRV                542
Corolla            530
M5                 509
C300               477
X6                 454
3 Series           436
RSX                397
Accord             310
Name: count, dtype: int64
InsuredHobbies
bungie-jumping    1751
```

```
paintball        1688
camping          1681
kayaking         1611
exercise         1589
reading          1586
movies           1529
yachting         1486
hiking           1483
base-jumping     1470
golf             1470
video-games      1420
board-games      1396
skydiving        1395
polo             1380
cross-fit        1249
sleeping         1220
dancing          1219
chess            1210
basketball       1003
Name: count, dtype: int64
```

Conluded from above will leave grouping VehicalMake and Model as I cant see any natural grouping -- may need to revisit if model struggling from overfitting

3.12 - 🗑️ **Dropped** Irrelevant or High-Cardinality Columns

To reduce noise and avoid overfitting, the following columns were removed:

Column	Reason for Removal
`Policy_CombinedSingleLimit`	Replaced by `LimitPerPerson` and `LimitPerAccident`
`CustomerID`	Unique identifier, not useful for modeling
`InsurancePolicyNumber`	High-cardinality identifier, non-informative
`VehicleID`	Unique identifier, adds no predictive value
`IncidentAddress`	High-cardinality
`IncidentYear`	1 value
`Country`	1 value

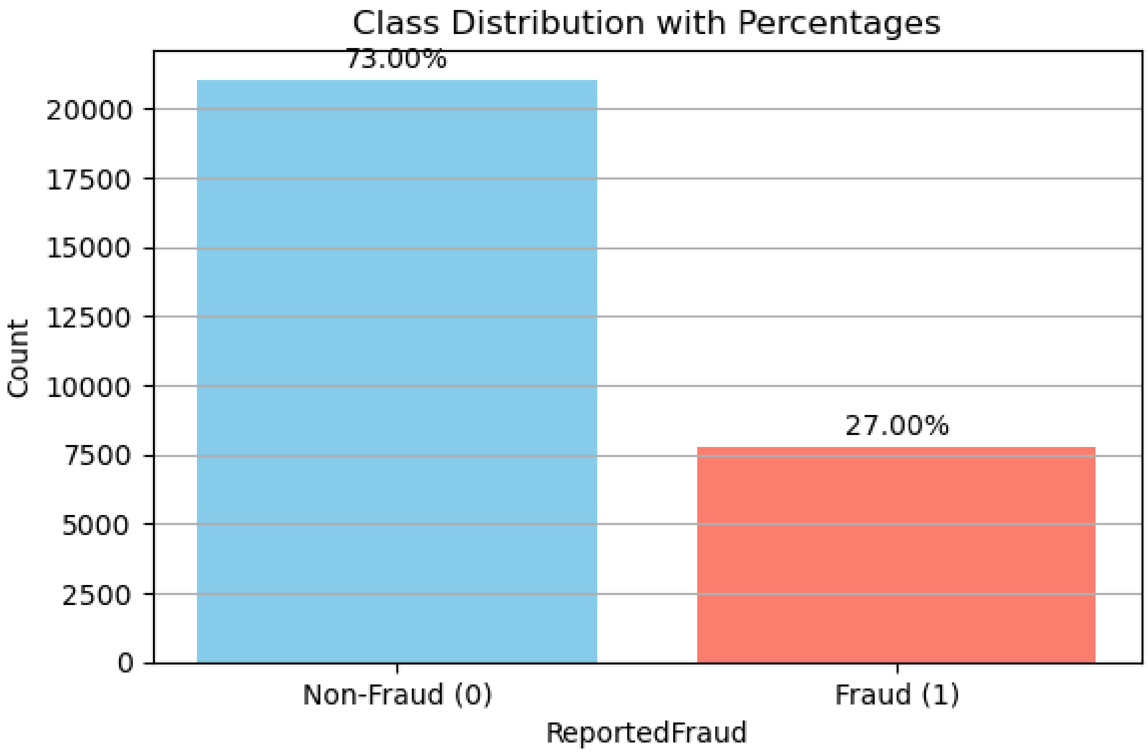
In [21]:
```
merged_df.drop(['Policy_CombinedSingleLimit','CustomerID','InsurancePolic
merged_df.columns
merged_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 28836 entries, 0 to 28835
Data columns (total 41 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   TypeOfIncident         28836 non-null  object 
 1   TypeOfCollission       28836 non-null  object 
 2   SeverityOfIncident     28836 non-null  object 
 3   AuthoritiesContacted   28836 non-null  object 
 4   IncidentState          28836 non-null  object 
 5   IncidentCity           28836 non-null  object 
 6   IncidentTime           28805 non-null  float64
 7   NumberOfVehicles       28836 non-null  int64  
 8   PropertyDamage         28836 non-null  object 
 9   BodilyInjuries         28836 non-null  int64  
 10  Witnesses              28790 non-null  float64
 11  PoliceReport           28836 non-null  object 
 12  AmountOfInjuryClaim    28836 non-null  int64  
 13  AmountOfPropertyClaim  28836 non-null  int64  
 14  AmountOfVehicleDamage  28836 non-null  int64  
 15  InsuredAge             28836 non-null  int64  
 16  InsuredZipCode         28836 non-null  int64  
 17  InsuredGender          28806 non-null  object 
 18  InsuredEducationLevel  28836 non-null  object 
 19  InsuredOccupation      28836 non-null  object 
 20  InsuredHobbies         28836 non-null  object 
 21  CapitalGains           28836 non-null  int64  
 22  CapitalLoss            28836 non-null  int64  
 23  CustomerLoyaltyPeriod  28836 non-null  int64  
 24  InsurancePolicyState   28836 non-null  object 
 25  Policy_Deductible      28836 non-null  int64  
 26  PolicyAnnualPremium    28695 non-null  float64
 27  UmbrellaLimit          28836 non-null  int64  
 28  InsuredRelationship    28836 non-null  object 
 29  VehicleMake            28836 non-null  object 
 30  VehicleModel           28836 non-null  object 
 31  VehicleYOM             28836 non-null  Int64  
 32  ReportedFraud          28836 non-null  int64  
 33  IncidentMonth          28836 non-null  int32  
 34  IncidentDay            28836 non-null  int32  
 35  IncidentWeekDay        28836 non-null  int32  
 36  IncidentWeek           28836 non-null  UInt32 
 37  IncidentIsOnWeekend    28836 non-null  int64  
 38  DaysSincePolicyStart   28836 non-null  int64  
 39  LimitPerPerson         28836 non-null  float64
 40  LimitPerAccident       28836 non-null  float64
dtypes: Int64(1), UInt32(1), float64(5), int32(3), int64(15), object(16)
memory usage: 8.6+ MB
```

3.13 - Identifying and dealing with class imbalance

In [22]:
```python
# Class distribution (already mapped to 0/1)
class_counts = merged_df['ReportedFraud'].value_counts()
class_percentages = class_counts / class_counts.sum() * 100

# Bar plot
plt.figure(figsize=(6, 4))
bars = plt.bar(class_counts.index.astype(str), class_counts.values, color
```

```
# Add percentage labels on top of each bar
for bar, percentage in zip(bars, class_percentages):
    height = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2.0, height + 200, f'{percenta

# Labels and styling
plt.title("Class Distribution with Percentages")
plt.xlabel("ReportedFraud")
plt.ylabel("Count")
plt.xticks(ticks=[0, 1], labels=["Non-Fraud (0)", "Fraud (1)"])
plt.grid(axis='y')
plt.tight_layout()
plt.show()
```



📊 Class Imbalance Analysis

🖼️ Visualization Summary

The bar plot below illustrates the distribution of the target variable `ReportedFraud`:

- **Non-Fraud (0):** 73%
- **Fraud (1):** 27%

Although the classes are not extremely imbalanced (e.g., 95/5), there is still a **moderate imbalance** between fraudulent and non-fraudulent claims.

💡 Recommended Solutions

Strategy	Description
`class_weight='balanced'`	Automatically adjusts the model's loss to give more importance to the minority class

Strategy	Description
Stratified Cross-Validation	Ensures each fold maintains the original class distribution during model evaluation
Precision-Recall Evaluation	Use metrics like precision, recall, F1-score, and AUC-PR for fair evaluation

These strategies help in building a model that treats fraud detection with proper attention, even with imbalanced data.

3.14 - Outlier Detetction

Will use **box-plot** strategy to visually see which all features have outliers

In [23]:
```
# Select numeric columns
numeric_cols = merged_df.select_dtypes(include=['float64','int64']).colum
n_cols = len(numeric_cols)

# Calculate grid dimensions (rows, cols)
n_rows = int(np.ceil(n_cols / 5))  # 5 columns per row (adjust as needed)
fig, axes = plt.subplots(n_rows, 5, figsize=(20, n_rows * 4))  # Adjust h
axes = axes.flatten()  # Flatten to 1D array for easy iteration

# Plot boxplots for each numeric column
for i, col in enumerate(numeric_cols):
    axes[i].boxplot(merged_df[col].dropna(), vert=False)
    axes[i].set_title(col, fontsize=10)

# Hide unused subplots
for j in range(i + 1, len(axes)):
    axes[j].axis('off')

plt.tight_layout()
plt.suptitle("Boxplots for All Numeric Columns (Outlier Check)", y=1.02)
plt.show()
```

📊 Outlier Summary Based on Box Plots (All Sets)

🚨 Clear Outlier (Will Be Addressed):

- **UmbrellaLimit**: Contains values < 0 (logically invalid). These entries will be cleaned or removed.

⚠️ Potential Outliers (Will Be Retained As-Is):

- **PolicyAnnualPremium**: Right-skewed with extreme high values.
- **CapitalGains**: Large upper-end values.
- **CapitalLoss**: Deep negative values; retained unless business rules say otherwise.
- **CustomerLoyaltyPeriod**: Some long-tenure customers; assumed valid.
- **AmountOfInjuryClaim**: Heavy right tail with dense outliers; kept for integrity.
- **AmountOfPropertyClaim**: Similar right-skew; retained.
- **AmountOfVehicleDamage**: Wide range and extreme values on both ends; retained.
- **InsuredAge**: Outliers at the high end (60+); assumed valid unless data says otherwise.

✅ No Issues Observed (No Clear Outliers):

- **Policy_Deductible**
- **DaysSincePolicyStart**
- **ReportedFraud** (binary)
- **IncidentIsOnWeekend** (binary)
- **VehicleYOM**
- **IncidentTime**
- **NumberOfVehicles**
- **BodilyInjuries**
- **Witnesses**
- **InsuredZipCode**
- **LimitPerPerson**: Uniform spread, no points beyond whiskers.
- **LimitPerAccident**: Uniform spread, no points beyond whiskers.

📝 Final Decision:

- Only **clear and logically invalid outliers** (e.g., `UmbrellaLimit < 0`) will be addressed.
- All other numerical outliers identified via box plots, including `LimitPerPerson` and `LimitPerAccident`, will be **retained** to preserve real-world variance and avoid over-cleaning.

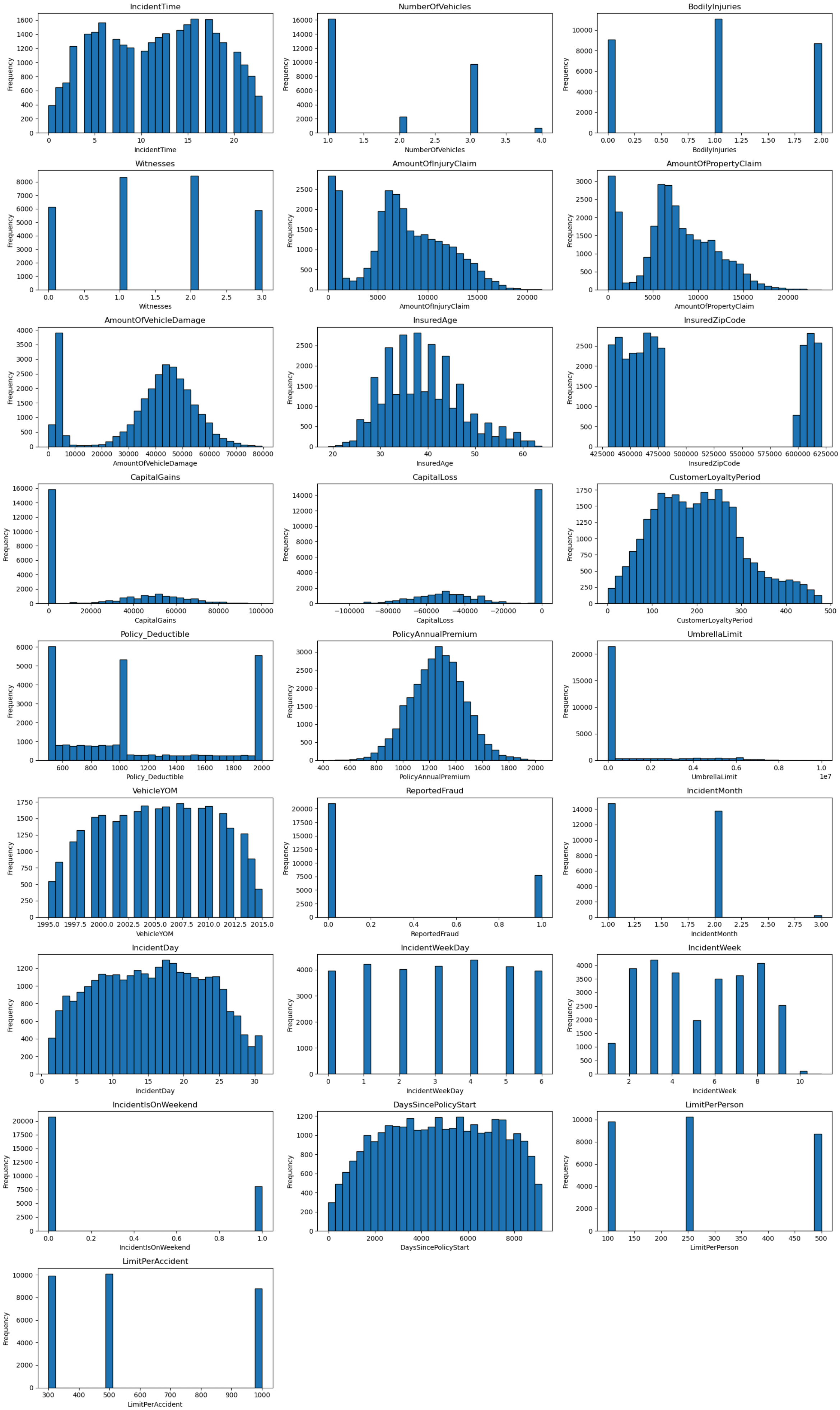
In [24]:
```
negative_umbrella_mask = merged_df['UmbrellaLimit'] < 0
print(f"Found {negative_umbrella_mask.sum()} rows with negative UmbrellaL
merged_df = merged_df[~negative_umbrella_mask]
```

```
Found 34 rows with negative UmbrellaLimit to be dropped
```

3.15 - Rescaling the attributes

Plot Numeric Feature Distributions (Histograms) to infer Scalaing Strategy

In [25]:
```
# Select numeric columns only
numeric_columns = merged_df.select_dtypes(include=['int64', 'float64', 'I

fig, axes = plt.subplots(nrows=9, ncols=3, figsize=(18, 30))  # Adjust si
axes = axes.flatten()

# Plot histogram for each numeric column
for i, col in enumerate(numeric_columns):
    axes[i].hist(merged_df[col].dropna(), bins=30, edgecolor='black')
    axes[i].set_title(col)
    axes[i].set_ylabel("Frequency")
    axes[i].set_xlabel(col)

# Remove any extra empty plots (if less than 27)
for j in range(len(numeric_columns), len(axes)):
    fig.delaxes(axes[j])

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



⚙️ Feature Scaling Strategy

To ensure consistent model performance and interpretability, numeric features are scaled based on their nature using the following rules:

✅ 1. High-Cardinality / Continuous Features → Use `StandardScaler`

These features have wide numeric ranges and many unique values. Standardization helps center and scale them.

Feature	Unique Values
`InsurancePolicyNumber`	28836
`PolicyAnnualPremium`	23852
`AmountOfVehicleDamage`	20041
`AmountOfInjuryClaim`	11958
`AmountOfPropertyClaim`	11785
`DaysSincePolicyStart`	7358
`UmbrellaLimit`	7089
`Policy_Deductible`	1496
`InsuredZipCode`	995
`CustomerLoyaltyPeriod`	479
`CapitalLoss`	354
`CapitalGains`	338

🌓 2. Medium-Cardinality / Bounded Features → Use `MinMaxScaler`

These features are bounded and ordinal but not fully continuous. MinMax scaling preserves range and order.

Feature	Unique Values
`IncidentDay`	31
`IncidentTime`	25
`VehicleYOM`	21
`IncidentWeek`	11
`InsuredAge`	46

🚫 3. Low-Cardinality Integer Features (< 10 unique values) → Leave As-Is

Skip scaling for features that:

- Have **very few unique values**

- Are effectively discrete categories or count indicators

Feature	Unique Values
`IncidentWeekDay`	7
`Witnesses`	5
`NumberOfVehicles`	4
`IncidentMonth`	3
`BodilyInjuries`	3
`LimitPerPerson`	3
`LimitPerAccident`	3
`IncidentIsOnWeekend`	2

> These features can be optionally treated as categorical and one-hot encoded, depending on model sensitivity.

🧠 Summary of Scaling Strategy

- **StandardScaler** → For continuous, high-cardinality numeric features
- **MinMaxScaler** → For bounded, ordinal-like features
- **Leave As-Is** → For low-cardinality (≤10) features — possibly treat as categorical

---

3.16 - 🧼 Missing Value Handling Strategy

This section outlines the strategy used for imputing missing values across different types of features prior to modeling with Logistic Regression.

✅ 1. High-Cardinality / Continuous Features → `StandardScaler`

These features have wide ranges and many unique values. They are typically numeric and continuous.

Imputation Strategy:

- `SimpleImputer(strategy='mean')`

Reason:

- The mean preserves the central tendency for normally distributed features.
- Compatible with standardization which centers and scales data.

Columns:

- `PolicyAnnualPremium`
- `AmountOfVehicleDamage`
- `AmountOfInjuryClaim`

- `AmountOfPropertyClaim`
- `DaysSincePolicyStart`
- `UmbrellaLimit`
- `Policy_Deductible`
- `InsuredZipCode`
- `CustomerLoyaltyPeriod`
- `CapitalLoss`
- `CapitalGains`

✅ 2. Bounded / Mid-Cardinality Features → `MinMaxScaler`

These are typically bounded or ordinal numerical features with moderate variability.

Imputation Strategy:

- `SimpleImputer(strategy='median')`

Reason:

- Median is robust to skewed distributions and outliers.
- MinMaxScaler is sensitive to outliers, so median ensures stable range.

Columns:

- `IncidentDay`
- `IncidentTime`
- `VehicleYOM`
- `IncidentWeek`
- `InsuredAge`

✅ 3. Low-Cardinality / Count or Flag Features → Leave Unscaled

These are discrete, count-based, or binary features.

Imputation Strategy:

- `SimpleImputer(strategy='most_frequent')`

Reason:

- These features often represent "presence" or "absence" of an event.
- Most frequent value is appropriate for small-range integer features.

Columns:

- `IncidentWeekDay`
- `Witnesses`
- `NumberOfVehicles`
- `IncidentMonth`
- `BodilyInjuries`

- `LimitPerPerson`
- `LimitPerAccident`
- `IncidentIsOnWeekend`

✅ Categorical Columns (One-Hot Encoded)

Categorical features are handled separately using `OneHotEncoder`.

Imputation Strategy:

- Imputation with `most_frequent` or `constant='missing'` can be optionally applied **before encoding**.

Encoding Strategy:

- `OneHotEncoder(handle_unknown='ignore', sparse_output=False)`

🔁 Summary Table

Feature Group	Scaler	Imputer Strategy
Continuous (high-cardinality)	StandardScaler	Mean
Bounded/ordinal	MinMaxScaler	Median
Counts, flags (low cardinality)	None	Most Frequent / Constant(0)
Categorical (object type)	OneHotEncoder	Most Frequent / Constant

In [26]:
```
# 1. StandardScaler for high-cardinality / continuous features
standard_scale_cols = [
    'PolicyAnnualPremium', 'AmountOfVehicleDamage',
    'AmountOfInjuryClaim', 'AmountOfPropertyClaim', 'DaysSincePolicyStart
    'UmbrellaLimit', 'Policy_Deductible', 'InsuredZipCode',
    'CustomerLoyaltyPeriod', 'CapitalLoss', 'CapitalGains'
]


# 2. MinMaxScaler for bounded or mid-cardinality features
minmax_scale_cols = [
    'IncidentDay', 'IncidentTime', 'VehicleYOM',
    'IncidentWeek', 'InsuredAge'
]

# 3. Leave these features unscaled (either categorical or simple counts)
leave_unchanged_cols = [
    'IncidentWeekDay', 'Witnesses', 'NumberOfVehicles',
    'IncidentMonth', 'BodilyInjuries',
    'LimitPerPerson', 'LimitPerAccident',
    'IncidentIsOnWeekend'
]

# 4. Leave these features unscaled (either categorical or simple counts)
onehot_encode_cols = [
    'InsuredGender',
    'InsuredEducationLevel',
    'InsuredOccupation',
```

```
    'InsuredHobbies',
    'InsurancePolicyState',
    'InsuredRelationship',
    'AuthoritiesContacted',
    'TypeOfIncident',
    'TypeOfCollission',
    'SeverityOfIncident',
    'IncidentState',
    'IncidentCity',
    'VehicleMake',
    'VehicleModel',
    'PropertyDamage',
    'PoliceReport',
]

target_col = ['ReportedFraud']
```

Define target and features

In [27]:
```
# STEP 1: Define target and features
X = merged_df.drop(columns=['ReportedFraud'])
y = merged_df['ReportedFraud']
```

4 💼 Business Logic: Estimating Model-Driven Loss

🎯 Goal

Quantify financial impact of a fraud detection model by translating prediction outcomes into **business costs and savings**.

🧠 Core Logic

- **Fraud Rate Assumption**
  Assume 10% of total customers file claims → helps infer total customer base from known fraud cases.

- **Claim & Profit Model**

  - Each fraud costs: `average_claim`
  - Business aims for: **2× gross profit** over total fraud value.
- **Pricing Strategy**
  Price per customer = `(Required Gross Profit) / (Total Customers)`
  Ensures sustainable margins.

💰 Financial Impact per Prediction Type

Prediction	Business Effect
✅ True Positive (TP)	Fraud caught → save `average_claim`
❌ False Negative (FN)	Fraud missed → lose `average_claim`

Prediction	Business Effect
⚠️ False Positive (FP)	Good customer flagged → lose `price_per_policy`
✅ True Negative (TN)	No effect

🧾 Final Formulae

- **Total Claims Value** = `fraud_cases × average_claim`
- **Required Profit** = `2 × total_claims_value`
- **Cost of FN** = `FN × average_claim`
- **Cost of FP** = `FP × price_per_policy`
- **Savings from TP** = `TP × average_claim`
- **Net Impact** = `Savings − (Cost_FN + Cost_FP)`

✅ Decision Logic

If `net_impact > 0`:
→ **Model is financially beneficial**
Else:
→ **Model is financially detrimental**

In [28]:

```
# Average Claim
def compute_avg_claim(df, claim_cols=['AmountOfInjuryClaim', 'AmountOfPro
    """
    Computes average total claim amount across selected claim columns.

    Parameters:
    - df: DataFrame containing the claim columns
    - claim_cols: List of columns representing parts of a total claim

    Returns:
    - avg_claim (float): Mean of total claims across rows
    """
    df = df.copy()
    df['total_claim'] = df[claim_cols].sum(axis=1)
    return df['total_claim'].mean()


average_claim, number_of_claimants = compute_avg_claim(merged_df) , len(m
average_claim ,number_of_claimants
```

Out[28]: `(np.float64(52268.720956877994), 28802)`

Estimated `avg_claim = 52,268.72`, Total Fraud Cases **`number_of_claimants = 28,802`** using the fraud-only dataset.

In [29]:

```
def estimate_business_loss(false_negatives, false_positives, true_positiv
                           average_claim=52268.72,
                           total_fraud_cases=28802,
                           fraud_rate=0.1):
    """
```

```
    Analyzes insurance claim data to quantify the financial impact of a f

        Args:
            file_path (str): The path to the insurance claims CSV file.
            true_positives (int): Number of correctly identified fraudule
            false_negatives (int): Number of fraudulent claims missed by
            false_positives (int): Number of legitimate claims wrongly fl
            true_negatives (int): Number of correctly identified legitima

        Returns:
            dict: A dictionary containing the detailed financial analysis
                  Returns None if the data file cannot be found.
    """



    # 2. Build the Pricing Model
    # Assumption: The cleaned dataset represents the customers who made a
    number_of_claimants = total_fraud_cases

    # Assumption: 10% of the customer base files a claim.
    # From this, we can estimate the total number of customers.
    total_customers = number_of_claimants / 0.10

    # Calculate total value of all claims
    total_claims_value = number_of_claimants * average_claim

    # Calculate required gross profit (double the claims to cover overhea
    required_gross_profit = 2 * total_claims_value

    # Calculate the necessary price per policy
    price_per_policy = required_gross_profit / total_customers


    # 3. Calculate the Financial Impact of the Model's Errors

    # Cost of False Positives: Legitimate customers are wrongly flagged.
    # We assume these customers are lost, costing one year's premium.
    cost_of_fp = false_positives * price_per_policy

    # Cost of False Negatives: Fraudulent claims that the model missed.
    # This is the direct cost of paying out these fraudulent claims.
    cost_of_fn = false_negatives * average_claim

    # Savings from True Positives: Fraudulent claims correctly identified
    # This is the direct savings from not paying out these claims.
    savings_from_tp = true_positives * average_claim

    # Net Financial Impact of the Model
    net_impact = savings_from_tp - cost_of_fn - cost_of_fp


    # --- 4. Package Results into a Dictionary ---
    results = {
        'required_gross_profit': required_gross_profit,
        'price_per_policy': price_per_policy,
        'savings_from_tp': savings_from_tp,
        'cost_of_fn': cost_of_fn,
        'cost_of_fp': cost_of_fp,
```

```
        'net_impact': net_impact,
        'is_beneficial': net_impact > 0
    }

    return results
```

5 - Pipeline Preprocessing Utility Class & Functions

5.1 📉 Class to drop highly correlated numerical features (|r| > 0.9) during training to prevent multicollinearity and data leakage in downstream models.

In [30]:
```
# Class to find which features are highly corelated and can be dropped i.
class CorrelationFilter(BaseEstimator, TransformerMixin):
    def __init__(self, threshold=0.9):
        self.threshold = threshold
        self.columns_to_drop_ = []

    def fit(self, X, y=None):
        numeric_df = X.select_dtypes(include=['number'])
        corr_matrix = numeric_df.corr().abs()
        upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1
        # For each column in the upper triangle, if any value in that col
        self.columns_to_drop_ = [column for column in upper.columns if an
        return self

    def transform(self, X):
        return X.drop(columns=self.columns_to_drop_, errors='ignore')
```

5.2 Utility function for Generating Learning ,Precision-Recall, ROC Curve and BER (Balance Error Curve)

In [31]:
```
def plot_model_curves(pipeline, X_train, y_train, X_test, y_test, title_p
    # Set consistent style
    plt.style.use('seaborn-v0_8')
    plt.rcParams['figure.facecolor'] = 'white'
    plt.rcParams['axes.grid'] = True
    plt.rcParams['grid.alpha'] = 0.3

    # ---- 1️⃣ Learning Curve ----
    train_sizes, train_scores, val_scores = learning_curve(
        estimator=pipeline,
        X=X_train,
        y=y_train,
        cv=StratifiedKFold(5),
        scoring='f1',
        train_sizes=np.linspace(0.1, 1.0, 10),
        n_jobs=-1,
        random_state=42
    )

    # ---- 2️⃣ Get Predictions ----
    pipeline.fit(X_train, y_train)
    y_probs = pipeline.predict_proba(X_test)[:, 1]

    # Calculate optimal threshold based on F1-score
```

```
precision, recall, pr_thresholds = precision_recall_curve(y_test, y_p
f1_scores = 2 * (precision * recall) / (precision + recall + 1e-9)
optimal_idx = np.argmax(f1_scores)
optimal_threshold = pr_thresholds[optimal_idx]
y_pred = (y_probs >= optimal_threshold).astype(int)

# ---- 3️⃣ Calculate All Metrics ----
# Precision-Recall
ap_score = average_precision_score(y_test, y_probs)

# ROC Curve
fpr, tpr, roc_thresholds = roc_curve(y_test, y_probs)
roc_auc = roc_auc_score(y_test, y_probs)

# BER Curve
thresholds = np.linspace(0, 1, 100)
ber_scores = []
for thresh in thresholds:
    y_pred_temp = (y_probs >= thresh).astype(int)
    ber = 1 - balanced_accuracy_score(y_test, y_pred_temp)
    ber_scores.append(ber)
optimal_ber_idx = np.argmin(ber_scores)
optimal_ber_threshold = thresholds[optimal_ber_idx]
min_ber = ber_scores[optimal_ber_idx]

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)

# ---- 📊 Plot All Visualizations ----
fig = plt.figure(figsize=(12, 6))
gs = fig.add_gridspec(2, 3)

# 1. Learning Curve
ax1 = fig.add_subplot(gs[0, 0])
ax1.plot(train_sizes, train_scores.mean(1), 'o-', color='#1f77b4', la
ax1.fill_between(train_sizes,
               train_scores.mean(1) - train_scores.std(1),
               train_scores.mean(1) + train_scores.std(1),
               alpha=0.1, color='#1f77b4')
ax1.plot(train_sizes, val_scores.mean(1), 'o-', color='#ff7f0e', labe
ax1.fill_between(train_sizes,
               val_scores.mean(1) - val_scores.std(1),
               val_scores.mean(1) + val_scores.std(1),
               alpha=0.1, color='#ff7f0e')
ax1.set_title(f'Learning Curve\n({title_prefix})', pad=12)
ax1.set_xlabel('Training Samples')
ax1.set_ylabel('F1 Score')
ax1.legend(loc='lower right')

# 2. Precision-Recall Curve
ax2 = fig.add_subplot(gs[0, 1])
ax2.plot(recall, precision, color='#2ca02c', label=f'AP = {ap_score:.
ax2.axhline(y=(y_test.mean()), color='r', linestyle='--', label='Base
ax2.set_title(f'Precision-Recall Curve\n({title_prefix})', pad=12)
ax2.set_xlabel('Recall (Sensitivity)')
ax2.set_ylabel('Precision (PPV)')
ax2.legend(loc='upper right')

# 3. ROC Curve
ax3 = fig.add_subplot(gs[0, 2])
```

```
    ax3.plot(fpr, tpr, color='#9467bd', label=f'AUC = {roc_auc:.3f}')
    ax3.plot([0, 1], [0, 1], 'r--', label='Baseline (Random)')
    ax3.set_title(f'ROC Curve\n({title_prefix})', pad=12)
    ax3.set_xlabel('False Positive Rate')
    ax3.set_ylabel('True Positive Rate')
    ax3.legend(loc='lower right')

    # 4. BER Curve
    ax4 = fig.add_subplot(gs[1, 0])
    ax4.plot(thresholds, ber_scores, color='#d62728', label='BER Curve')
    ax4.scatter(optimal_ber_threshold, min_ber, color='red',
                label=f'Optimal: {optimal_ber_threshold:.2f}\nMin BER: {min
    ax4.set_title(f'Balanced Error Rate Curve\n({title_prefix})', pad=12)
    ax4.set_xlabel('Threshold')
    ax4.set_ylabel('Balanced Error Rate')
    ax4.legend(loc='upper right')

    # 5. Confusion Matrix (Counts)
    ax5 = fig.add_subplot(gs[1, 1:])  # Span full width for confusion mat
    disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                                  display_labels=['Legit', 'Fraud'])
    disp.plot(ax=ax5, cmap='Blues', values_format='d', colorbar=False)
    ax5.set_title(f'Confusion Matrix (Threshold: {optimal_threshold:.3f})

    plt.tight_layout()
    plt.show()

    # ---- 🎯 Optimal Threshold Analysis ----
    tn, fp, fn, tp = cm.ravel()

    print(f"\n{' Optimal Threshold Analysis ':-^60}")
    print(f"\nBased on F1-Score: {optimal_threshold:.4f}")
    print(f"Max F1-Score: {f1_scores[optimal_idx]:.4f}")
    print(f"- Precision = {precision[optimal_idx]:.4f}")
    print(f"- Recall = {recall[optimal_idx]:.4f}")

    print(f"\nBased on BER: {optimal_ber_threshold:.4f}")
    print(f"Min Balanced Error Rate: {min_ber:.4f}")

    print(f"\n{' Confusion Matrix Metrics ':-^60}")
    print(f"\nAt threshold: {optimal_threshold:.4f}")
    print(f"True Positives (TP): {tp}")
    print(f"False Positives (FP): {fp}")
    print(f"True Negatives (TN): {tn}")
    print(f"False Negatives (FN): {fn}")

    print(f"\nAdditional Metrics:")
    print(f"Accuracy: {(tp + tn) / (tp + tn + fp + fn):.4f}")
    print(f"Balanced Accuracy: {balanced_accuracy_score(y_test, y_pred):.
    print(f"Specificity (TNR): {tn / (tn + fp):.4f}")
    print(f"False Positive Rate: {fp / (fp + tn):.4f}")

    # Business impact
    analysis_results = estimate_business_loss(fn, fp , tp)

    # Print business cost summary
    print(f"\n{' Business Impact Estimation ':-^60}")
    print("\n--- Pricing Model ---")

    print(f"Required Gross Profit: {human_readable_currency(analysis_resu
```

```
    print(f"Required Price per Policy: {human_readable_currency(analysis_

    print("\n--- Model Financial Impact Analysis ---")
    print(f"Savings from catching fraud (True Positives): {human_readable
    print(f"Cost of missing fraud (False Negatives): {human_readable_curr
    print(f"Cost of losing customers (False Positives): {human_readable_c
    print("-------------------------------------------------")
    print(f"Net Financial Impact of the Model: {human_readable_currency(a
    print("-------------------------------------------------")

    if analysis_results['is_beneficial']:
        print("\nConclusion: ✅ The model is financially beneficial.")
    else:
        print("\nConclusion: ❌ The model is financially detrimental.")

def human_readable_currency(amount):
    """
    Formats large currency numbers into human-friendly strings.
    E.g., 24,409,492 → $24.4M
    """
    if amount >= 1_000_000_000:
        return f"${amount/1_000_000_000:.1f}B"
    elif amount >= 1_000_000:
        return f"${amount/1_000_000:.1f}M"
    elif amount >= 1_000:
        return f"${amount/1_000:.1f}K"
    else:
        return f"${amount:,.0f}"
```

5.3 Utility Function for creating pre-process-pipeline

In [32]:
```
def prepare_data_pipeline(X, y, standard_scale_cols, minmax_scale_cols,
                          leave_unchanged_cols, onehot_encode_cols,
                          correlation_threshold=0.9, test_size=0.2, rando
    """
    Splits the data, applies correlation filtering, and returns:
    - updated X_train, X_test, y_train, y_test
    - fitted correlation filter
    - updated column lists
    - ColumnTransformer preprocessor
    """
    # STEP 1: Split the dataset with stratification as class is imbalance
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, stratify=y, test_size=test_size, random_state=random_state
    )

    # STEP 2: Apply correlation filter on training data
    corr_filter = CorrelationFilter(threshold=correlation_threshold)
    X_corr_filtered = corr_filter.fit_transform(X_train)

    columns_dropped = corr_filter.columns_to_drop_
    print("Correlated Columns Dropped:", columns_dropped)

    # STEP 3: Rebuild feature column groups after drop
    standard_scale_cols = [col for col in standard_scale_cols if col not
    minmax_scale_cols = [col for col in minmax_scale_cols if col not in c
    leave_unchanged_cols = [col for col in leave_unchanged_cols if col no
    onehot_encode_cols = [col for col in onehot_encode_cols if col not in
```

```
    # STEP 4: Define imputers and scalers
    standard_pipeline = Pipeline([
        ('impute', SimpleImputer(strategy='mean')),
        ('scale', StandardScaler())
    ])

    minmax_pipeline = Pipeline([
        ('impute', SimpleImputer(strategy='median')),
        ('scale', MinMaxScaler())
    ])

    unchanged_pipeline = Pipeline([
        ('impute', SimpleImputer(strategy='most_frequent'))
    ])

    # STEP 5: Define column transformer
    preprocessor = ColumnTransformer([
        ('standard', standard_pipeline, standard_scale_cols),
        ('minmax', minmax_pipeline, minmax_scale_cols),
        ('unchanged', unchanged_pipeline, leave_unchanged_cols),
        ('cat', OneHotEncoder(handle_unknown='ignore', sparse_output=Fals
    ], remainder='drop')

    return (X_train, X_test, y_train, y_test, preprocessor)
```

5.4 Split Data and apply Corelation and get ColumnTransformer

In [33]:
```
X_train, X_test, y_train, y_test, preprocessor = prepare_data_pipeline(
    X, y,
    standard_scale_cols,
    minmax_scale_cols,
    leave_unchanged_cols,
    onehot_encode_cols
)
```

```
Correlated Columns Dropped: ['CustomerLoyaltyPeriod']
```

Outcome of above execution is `CustomerLoyaltyPeriod` is **dropped** as its corelation is greater than threashhold which is 0.9

4 - Technique 1 - **Logistic Regrssion**

🎯 Motivation for Choosing Logistic Regression

Logistic Regression (LR) was selected as the first classification technique due to its strong alignment with both technical and business requirements of the client. It is a fast, transparent, and interpretable model that outputs **calibrated probabilities** — a key requirement for optimizing **business cost** via threshold tuning.

Key reasons for selection:

- **Interpretability**: Model coefficients are easy to explain to stakeholders.
- **Efficiency**: Quick to train and validate.
- **Probabilistic Output**: Allows threshold-based control for cost sensitivity.

- **Business Alignment**: Supports calculating cost/savings per TP, FP, FN.
- **Resilience**: Regularization (`C=0.1`) improves generalization.
- **Class Imbalance Ready**: Built-in support with `class_weight='balanced'` .

This technique serves as a **baseline reference model** that is business-aware, technically sound, and benchmarked against cost-sensitive performance objectives.

🧪 Schematic of the Analysis Process

4.1 Baseline version of LogisticRegression

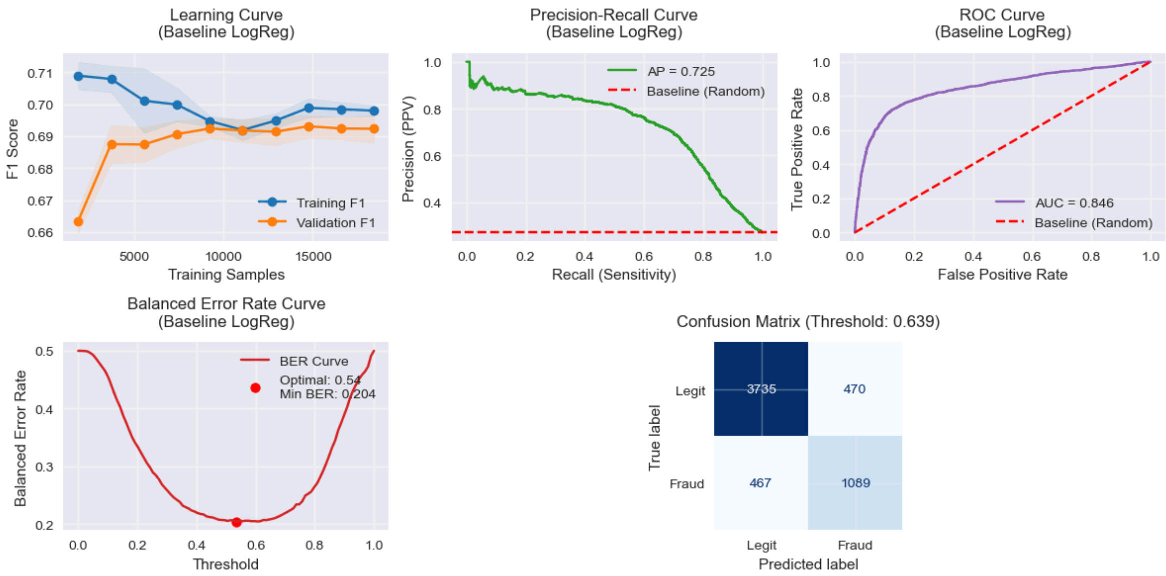
In [71]:

```
# Final Combined pipeline
pipeline = Pipeline(steps=[
    ('preprocess', preprocessor),
    ('clf', LogisticRegression(
        class_weight='balanced',
        penalty='l2',
        C=0.1, # More regularization
        solver='liblinear',
        max_iter=1000
    ))
])

# Fit and Evaluate
pipeline.fit(X_train, y_train)
y_pred = pipeline.predict(X_test)
print(classification_report(y_test, y_pred))

y_probs = pipeline.predict_proba(X_test)[:, 1]
print(f"AP Score: {average_precision_score(y_test, y_probs):.3f}")
```

```
              precision    recall  f1-score   support

           0       0.90      0.83      0.86      4205
           1       0.62      0.76      0.68      1556

    accuracy                           0.81      5761
   macro avg       0.76      0.79      0.77      5761
weighted avg       0.83      0.81      0.82      5761

AP Score: 0.725
```

In [103…

```
plot_model_curves(pipeline, X_train, y_train, X_test, y_test, title_prefi
```




```
---------------- Optimal Threshold Analysis ----------------

Based on F1-Score: 0.6391
Max F1-Score: 0.6992
- Precision = 0.6985
- Recall = 0.6999

Based on BER: 0.5354
Min Balanced Error Rate: 0.2040

----------------- Confusion Matrix Metrics -----------------

At threshold: 0.6391
True Positives (TP): 1089
False Positives (FP): 470
True Negatives (TN): 3735
False Negatives (FN): 467

Additional Metrics:
Accuracy: 0.8374
Balanced Accuracy: 0.7940
Specificity (TNR): 0.8882
False Positive Rate: 0.1118

---------------- Business Impact Estimation ----------------

--- Pricing Model ---
Required Gross Profit: $3.0B
Required Price per Policy: $10.5K

--- Model Financial Impact Analysis ---
Savings from catching fraud (True Positives): $56.9M
Cost of missing fraud (False Negatives): $24.4M
Cost of losing customers (False Positives): $4.9M
-------------------------------------------------
Net Financial Impact of the Model: $27.6M
-------------------------------------------------

Conclusion: ✅ The model is financially beneficial.
```

📊 Brief Evaluation Summary (Baseline Logistic Regression)

- **Learning Curve**
  Training and validation F1 scores are close, indicating low variance. The model does not appear to overfit significantly.

- **Precision-Recall Curve**
  Precision remains above baseline across most recall values. Area under the curve (AP = 0.725) shows good performance for the imbalanced dataset.

- **ROC Curve**
  AUC = 0.846 indicates strong separability between fraud and legitimate claims. The model performs significantly better than random guessing.

- **Balanced Error Rate (BER) Curve**
  Minimum BER of 0.204 is achieved at threshold ≈ 0.5, supporting the client's target of minimizing false positive and false negative rates.

- **Confusion Matrix**
  At the optimal threshold of 0.639:
  - TP = 1089, FP = 470
  - TN = 3735, FN = 467
- **Threshold Metrics Summary**
  - Precision: 0.6985
  - Recall: 0.6999
  - F1 Score: 0.6992
  - Accuracy: 0.8374
  - Balanced Accuracy: 0.7940
- **Business Impact Analysis**
  - Savings from TP: **$56.9M**
  - Cost of FN: **$24.4M**
  - Cost of FP: **$4.9M**
  - **Net Financial Impact**: **$27.6M**

✅ **Conclusion**: The model is **financially beneficial** and technically balanced, meeting performance goals in both BER and business impact.

4.2 - Hyperparameter Tuning with Stratified K-Fold + GridSearchCV

In [49]:
```
# 1. Define a targeted parameter grid i.e hyperparameters
param_grid = {
    'clf__C': [0.01, 0.1, 1, 10, 100,1000,10000],  # Test Regularization
    'clf__penalty': ['l2','l1'],           # Faster than l1 for liblinear
    'clf__solver': ['liblinear'],     # Optimized for small-to-medium dat
    'clf__class_weight': ['balanced',{0:1, 1:5}] # Focus on best imbalanc
}
```

🔍 Setting Hyperparameters Rationale

- **C (Inverse of Regularization Strength):**
  A wide range of values from 0.01 to 10000 was tested to control overfitting observed in the learning curve.
  Smaller values (e.g., 0.01) apply stronger regularization to prevent variance, while larger values (e.g., 1000) reduce bias.

- **Penalty:**

  - `'l2'` is the default and provides stable performance by shrinking weights without eliminating features.
  - `'l1'` encourages sparsity by zeroing out less important coefficients, which can aid in feature selection and improve interpretability.
- **Solver:**
  `'liblinear'` was chosen as it supports both `'l1'` and `'l2'` penalties and is well-suited for small to medium-sized datasets.

- **Class Weight:**

  - `'balanced'` adjusts weights inversely to class frequency, helping with the class imbalance in the dataset.
  - `{0:1, 1:5}` manually increases the penalty for misclassifying fraud cases (class 1), aligning with the business need to reduce missed fraud.

In [104…

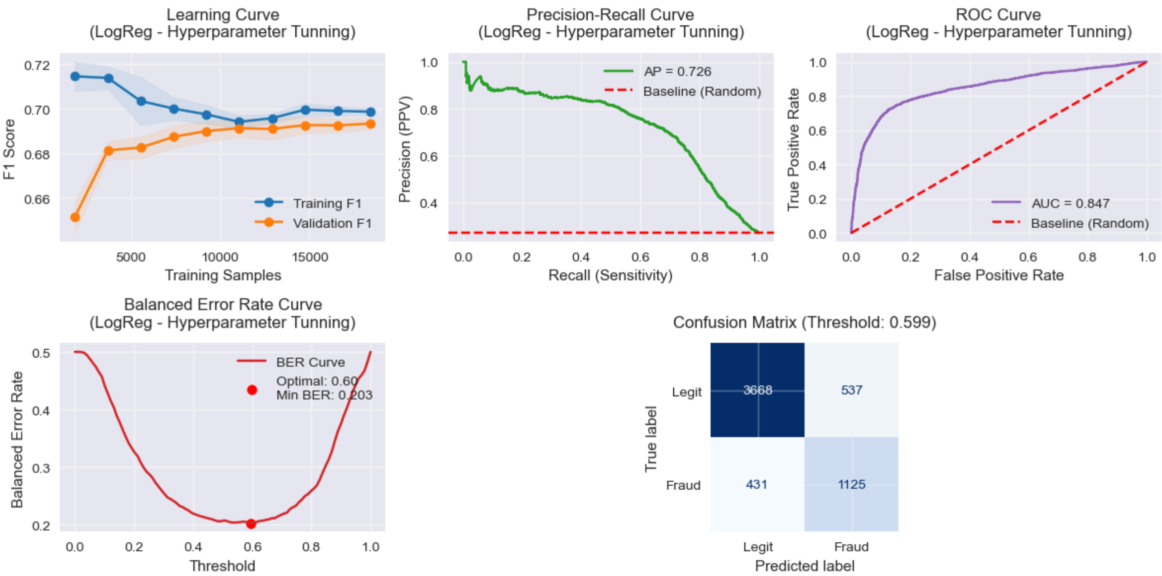
```
# 2. Streamlined CV strategy
cv = StratifiedKFold(n_splits=3, shuffle=True, random_state=42)  # Reduce

# 3. Configure GridSearch
grid_search = GridSearchCV(
    estimator=pipeline,
    param_grid=param_grid,
    scoring='average_precision',  # Best metric for fraud detection
    cv=cv,
    n_jobs=-1,                   # Use all CPU cores
    verbose=1                    # Moderate verbosity
)

# 4. Execute with timing
print("Starting optimized GridSearch...")
start_time = time.time()
grid_search.fit(X_train, y_train)
print(f"GridSearch completed in {(time.time()-start_time)/60:.1f} minutes

# 5. Best model analysis
lr_best_model_with_hyper_tunning = grid_search.best_estimator_
print("\nBest Parameters:")
for param, value in grid_search.best_params_.items():
    print(f"{param:20}: {value}")

# 6. Final evaluation
y_pred = lr_best_model_with_hyper_tunning.predict(X_test)
y_probs = lr_best_model_with_hyper_tunning.predict_proba(X_test)[:, 1]
print("\nTest Set Performance:")
print(classification_report(y_test, y_pred))
print(f"AP Score: {average_precision_score(y_test, y_probs):.3f}")
```

```
Starting optimized GridSearch...
Fitting 3 folds for each of 28 candidates, totalling 84 fits
GridSearch completed in 0.6 minutes

Best Parameters:
clf__C                : 100
clf__class_weight     : balanced
clf__penalty          : l2
clf__solver           : liblinear

Test Set Performance:
              precision    recall  f1-score   support

           0       0.90      0.83      0.86      4205
           1       0.62      0.76      0.68      1556

    accuracy                           0.81      5761
   macro avg       0.76      0.79      0.77      5761
weighted avg       0.83      0.81      0.82      5761

AP Score: 0.726
```

Plot Logistic Regression Learning ,Precision-Recall ,ROC Curve , BER , Confusion - StratifiedKFold and GridSearchCV

In [105…

```python
plot_model_curves(
    pipeline=lr_best_model_with_hyper_tunning,
    X_train=X_train,
    y_train=y_train,
    X_test=X_test,
    y_test=y_test,
    title_prefix="LogReg - Hyperparameter Tunning"
)
```



```
---------------- Optimal Threshold Analysis ----------------

Based on F1-Score: 0.5990
Max F1-Score: 0.6992
- Precision = 0.6769
- Recall = 0.7230

Based on BER: 0.5960
Min Balanced Error Rate: 0.2027

----------------- Confusion Matrix Metrics -----------------

At threshold: 0.5990
True Positives (TP): 1125
False Positives (FP): 537
True Negatives (TN): 3668
False Negatives (FN): 431

Additional Metrics:
Accuracy: 0.8320
Balanced Accuracy: 0.7977
Specificity (TNR): 0.8723
False Positive Rate: 0.1277

---------------- Business Impact Estimation ----------------

--- Pricing Model ---
Required Gross Profit: $3.0B
Required Price per Policy: $10.5K

--- Model Financial Impact Analysis ---
Savings from catching fraud (True Positives): $58.8M
Cost of missing fraud (False Negatives): $22.5M
Cost of losing customers (False Positives): $5.6M
-------------------------------------------------
Net Financial Impact of the Model: $30.7M
-------------------------------------------------

Conclusion: ✅ The model is financially beneficial.
```

📊 Brief Evaluation Summary (Tuned Logistic Regression via GridSearchCV)

- **Learning Curve**
  Training and validation F1 scores remain closely aligned, suggesting reduced overfitting and improved generalization after tuning.

- **Precision-Recall Curve**
  AP = 0.726, slightly improved over the baseline. The model maintains high precision across a broad range of recall values.

- **ROC Curve**
  AUC = 0.847 confirms strong discrimination capability, comparable to the baseline model.

- **Balanced Error Rate (BER) Curve**
  Minimum BER of 0.2027 is achieved at threshold ≈ 0.6, showing slight improvement in balancing false positives and false negatives.

- **Confusion Matrix**
  At the optimal threshold of 0.599:

  - TP = 1125, FP = 537
  - TN = 3668, FN = 431
- **Threshold Metrics Summary**

  - Precision: 0.6769
  - Recall: 0.7230
  - F1 Score: 0.6992
  - Accuracy: 0.8320
  - Balanced Accuracy: 0.7977
- **Business Impact Analysis**

  - Savings from TP: **$58.8M**
  - Cost of FN: **$22.5M**
  - Cost of FP: **$5.6M**
  - **Net Financial Impact**: **$30.7M**

✅ **Conclusion**: The tuned model is **financially beneficial** and shows slightly improved recall and business value over the baseline model.

5 - Technique 2 - **RandomForest**

🎯 Motivation for Choosing Random Forest

Random Forest (RF) is an ensemble learning technique known for its high performance and robustness, especially in **tabular data with mixed feature types**. It works by aggregating decisions from multiple decorrelated trees, making it resistant to overfitting and suitable for **imbalanced and noisy datasets** like fraud detection.

Key reasons for choosing Random Forest:

- **Handles Imbalanced Data**: Can integrate `class_weight='balanced'` natively.
- **Captures Nonlinear Interactions**: Unlike Logistic Regression, RF models complex interactions without needing feature engineering.
- **Robust to Overfitting**: Tree-level regularization (via `max_depth`, `min_samples_leaf`) helps improve generalization.
- **Feature Importance**: Useful for understanding which attributes contribute most to fraud detection.

- **Tunable for Business Constraints**: Through hyperparameters and thresholding, RF can be optimized for precision/recall trade-offs and cost minimization.

This model complements Logistic Regression by offering **greater flexibility** at the cost of some interpretability.

🧪 Schematic of the Analysis Process

⚙️ Summary of Model Implementation Steps

- **Base Model Configuration**
  Used 100 trees (`n_estimators=100`) with moderately regularized settings:
  - `max_depth=None` , `min_samples_leaf=2` , `max_features='sqrt'` , and `class_weight='balanced'`

  This provided a strong baseline with out-of-box generalization.

- **Hyperparameter Tuning (Full Feature Set, 168 Features)**
  Conducted extensive grid search using `StratifiedKFold` and a well-regularized parameter grid:
  - Tuned `max_depth` , `min_samples_split` , `min_samples_leaf` , `min_impurity_decrease` , etc.
  - Evaluation focused on recall, BER, and net financial impact
- **Re-Tuning with Feature Reduction (Top 100 Features)**
  Performed feature reduction to address potential overfitting.
  - Repeated GridSearchCV on the reduced set.
  - However, **learning curves still showed a performance gap** between training and validation sets.
- **Overfitting Diagnosis**
  Despite feature reduction, the gap between training and CV curves **persisted**, indicating that:
  - Overfitting is **not caused by high feature dimensionality**,
  - But likely due to **limited training data**, causing high variance.
- **Performance Evaluation**
  All three models (baseline, tuned-168, tuned-100) were evaluated using:
  - ROC/PR curves, BER, F1, Precision, Recall
  - Cost modeling based on TP, FP, FN business impact

5.1 - Baseline version of RandomForest

In [34]:
```
# Final Combined pipeline
pipeline = Pipeline(steps=[
    ('preprocess', preprocessor),
    ('clf', RandomForestClassifier(
        n_estimators=100,
```

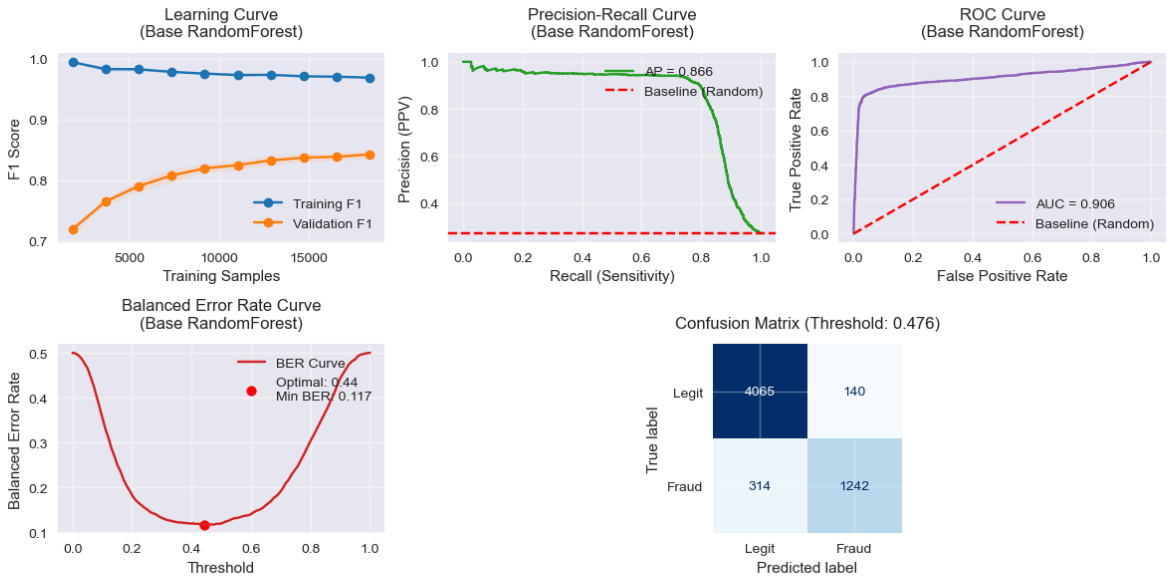
```
            max_depth=None,
            min_samples_leaf=2,
            max_features='sqrt',
            class_weight='balanced',
            random_state=42
        ))
])

# Fit and Evaluate
pipeline.fit(X_train, y_train)
y_pred = pipeline.predict(X_test)
print(classification_report(y_test, y_pred))

y_probs = pipeline.predict_proba(X_test)[:, 1]
print(f"AP Score: {average_precision_score(y_test, y_probs):.3f}")
```

```
              precision    recall  f1-score   support

           0       0.93      0.97      0.95      4205
           1       0.91      0.79      0.84      1556

    accuracy                           0.92      5761
   macro avg       0.92      0.88      0.90      5761
weighted avg       0.92      0.92      0.92      5761

AP Score: 0.866
```

Learning and Precision Recall Curve & ROC

In [64]:
```
plot_model_curves(pipeline, X_train, y_train, X_test, y_test, title_prefi
```

```
/opt/homebrew/Caskroom/miniconda/base/envs/.uol/lib/python3.12/site-packag
es/joblib/externals/loky/process_executor.py:752: UserWarning: A worker st
opped while some jobs were given to the executor. This can be caused by a
too short worker timeout or by a memory leak.
  warnings.warn(
```



```
---------------- Optimal Threshold Analysis ----------------

Based on F1-Score: 0.4755
Max F1-Score: 0.8455
- Precision = 0.8987
- Recall = 0.7982

Based on BER: 0.4444
Min Balanced Error Rate: 0.1172

----------------- Confusion Matrix Metrics -----------------

At threshold: 0.4755
True Positives (TP): 1242
False Positives (FP): 140
True Negatives (TN): 4065
False Negatives (FN): 314

Additional Metrics:
Accuracy: 0.9212
Balanced Accuracy: 0.8825
Specificity (TNR): 0.9667
False Positive Rate: 0.0333

---------------- Business Impact Estimation ----------------

--- Pricing Model ---
Required Gross Profit: $3.0B
Required Price per Policy: $10.5K

--- Model Financial Impact Analysis ---
Savings from catching fraud (True Positives): $64.9M
Cost of missing fraud (False Negatives): $16.4M
Cost of losing customers (False Positives): $1.5M
-------------------------------------------------
Net Financial Impact of the Model: $47.0M
-------------------------------------------------

Conclusion: ✅ The model is financially beneficial.
```

📊 Brief Evaluation Summary (Baseline Random Forest)

- **Learning Curve**
  High training scores and a visible gap with validation scores suggest **mild overfitting**, but the validation curve steadily improves with more data.

- **Precision-Recall Curve**
  AP = 0.866 — the highest among all models so far. The model maintains very high precision even at lower recall levels, which is beneficial for fraud detection.

- **ROC Curve**
  AUC = 0.906, showing excellent discrimination between fraud and legitimate cases.

- **Balanced Error Rate (BER) Curve**
  BER is minimized to 0.117 at a threshold of ~0.475 — significantly better than

other models, indicating balanced predictive power.

- **Confusion Matrix**

  At the optimal threshold of 0.475:

  - TP = 1242, FP = 140
  - TN = 4065, FN = 314
- **Threshold Metrics Summary**

  - Precision: 0.8987
  - Recall: 0.7982
  - F1 Score: 0.8455
  - Accuracy: 0.9212
  - Balanced Accuracy: 0.8825
  - False Positive Rate: 0.0333
- **Business Impact Analysis**

  - Savings from TP: **$64.9M**
  - Cost of FN: **$16.4M**
  - Cost of FP: **$1.5M**
  - **Net Financial Impact**: **$47.0M**

✅ **Conclusion**: The model is **financially beneficial**, offering the **best BER and net profit** among all baseline models, with excellent precision and recall balance.

Calculating Total Features

In [35]:
```
total_columns = (
    len(standard_scale_cols) +          # Standard scaled columns
    len(minmax_scale_cols) +            # MinMax scaled columns
    len(leave_unchanged_cols) +         # Unchanged columns
    sum([len(preprocessor.named_transformers_['cat'].categories_[i])
     for i in range(len(onehot_encode_cols))])  # OneHot encoded columns
)
print('Total Features:' , total_columns )
```

```
Total Features: 168
```

Feature Selection Utility Class for RandomForest

In [36]:
```
class FeatureSelector(BaseEstimator, TransformerMixin):
    """Selects top N features based on RandomForest importance"""
    def __init__(self, n_features=30, random_state=42):
        self.n_features = n_features
        self.random_state = random_state
        self.selected_features = None
        self.selector = None

    def fit(self, X, y):
        # Fit RandomForest to get importances
        rf = RandomForestClassifier(
            n_estimators=100,
            random_state=self.random_state,
            n_jobs=-1
```

```
        )
        rf.fit(X, y)

        # Configure selector
        self.selector = SelectFromModel(
            rf,
            max_features=self.n_features,
            threshold=-np.inf,  # Force all features to be considered
            prefit=True
        )
        self.selected_features = self.selector.get_support()
        return self

    def transform(self, X):
        return self.selector.transform(X)

    def get_feature_names(self, input_features=None):
        if input_features is None:
            return None
        return np.array(input_features)[self.selected_features].tolist()

def build_rf_feature_selector_pipeline(preprocessor, n_features=30):
    """Builds complete pipeline with feature selection"""
    print(f'Configuring Pipeline for {n_features} features ')
    return Pipeline([
        ('preprocess', preprocessor),
        ('feature_selection', FeatureSelector(n_features=n_features)),
        ('clf', RandomForestClassifier(
            class_weight='balanced',
            random_state=42,
            n_jobs=-1
        ))
    ])
```

5.2 - RandomForest Hyperparameter Tuning with Stratified K-Fold + GridSearchCV and Feature Selection

Top 100 features from **Total** feature **168**

In [50]:
```
# 1. Build pipeline
no_of_top_features_selected = 100
pipeline = build_rf_feature_selector_pipeline(preprocessor, n_features=no

# 2. Define hyperparameter grid
param_grid = {
    'clf__n_estimators': [100, 200],            # Number of trees (balan
    'clf__max_depth': [20,30],                # Tree depth (None=unlimited, 1
    'clf__min_samples_split': [5, 10],          # Minimum samples to split
    'clf__min_samples_leaf': [1, 2, 4],            # Minimum samples per le
    'clf__max_features': ['sqrt', 'log2'],         # Features per split (r
    'clf__min_impurity_decrease': [0.0, 0.01],    # Split significance thr
    'clf__bootstrap': [True],                    # Data subsampling (True=
    'clf__class_weight': ['balanced']            # Handles class imbalance
}
```

```
Configuring Pipeline for 100 features
```

🔍 Setting Hyperparameters Rationale

- **n_estimators (Number of Trees):**
  Values of 100 and 200 were chosen to balance **model stability** and **computational cost**. More trees usually reduce variance, but with diminishing returns.

- **max_depth (Tree Depth):**
  Depths of 20 and 30 were selected to **limit overfitting**. Shallower trees generalize better; unrestricted depth may lead to memorization of training data.

- **min_samples_split:**
  Values of 5 and 10 were tested to control the **minimum number of samples required to split a node**, which helps prevent overly complex trees.

- **min_samples_leaf:**
  Set to 1, 2, and 4 to ensure that **leaf nodes** have a minimum number of samples. Higher values reduce overfitting by smoothing decision boundaries.

- **max_features:**
  `'sqrt'` and `'log2'` were tested to **reduce correlation between trees** and improve generalization by randomly selecting fewer features per split.

- **min_impurity_decrease:**
  A value of 0.01 was included to allow splits only when a **minimum gain in impurity** is achieved, acting as a form of **regularization**.

- **bootstrap:**
  Enabled (`True`) to allow **data subsampling**, which introduces randomness and reduces overfitting by decorrelating trees.

- **class_weight:**
  `'balanced'` adjusts weights inversely to class frequencies and helps the model better handle the **class imbalance** present in fraud detection.

In [38]:
```
# 2. Streamlined CV strategy
cv = StratifiedKFold(n_splits=3, shuffle=True, random_state=42)  # Reduce

# 3. Configure GridSearch
grid_search = GridSearchCV(
    estimator=pipeline,
    param_grid=param_grid,
    scoring='average_precision',  # Best metric for fraud detection
    cv=cv,
    n_jobs=-1,                   # Use all CPU cores
    verbose=1                    # Moderate verbosity
)

# 4. Execute with timing
print("Starting optimized GridSearch...")
start_time = time.time()
grid_search.fit(X_train, y_train)
print(f"GridSearch completed in {(time.time()-start_time)/60:.1f} minutes
```

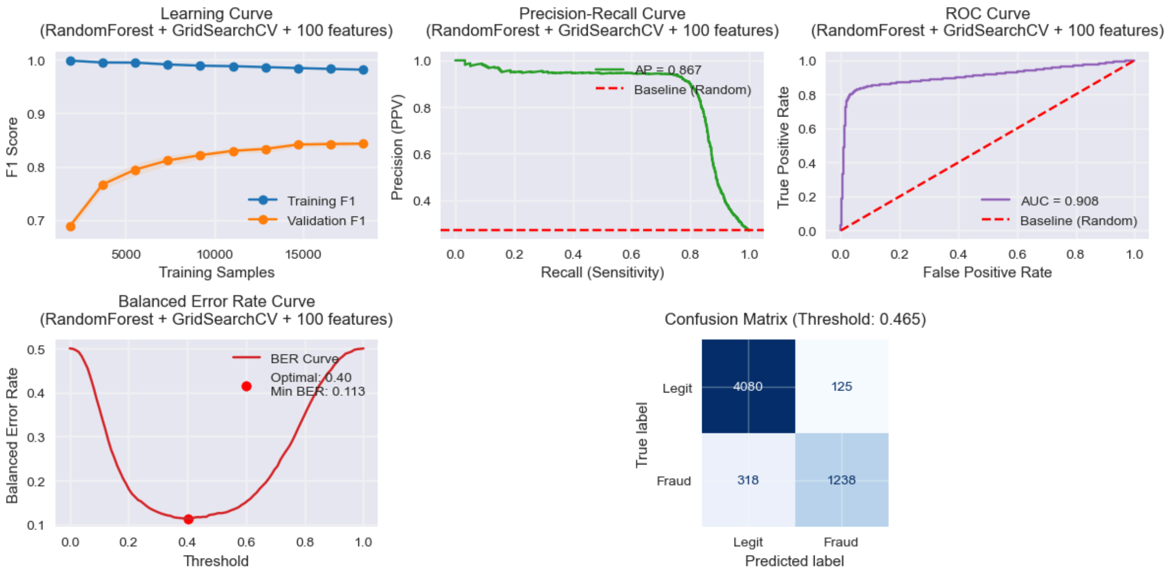
```
# 5. Get selected features
feature_names = preprocessor.get_feature_names_out()
selected_features = grid_search.best_estimator_.named_steps['feature_sele
print(f'Total Features Selected {len(selected_features)}, Features Select

# 5. Best model analysis
rf_best_model_with_hyper_tunning = grid_search.best_estimator_
print("\nBest Parameters:")
for param, value in grid_search.best_params_.items():
    print(f"{param:20}: {value}")

# 6. Final evaluation
y_pred = rf_best_model_with_hyper_tunning.predict(X_test)
print("\nTest Set Performance:")
print(classification_report(y_test, y_pred))
print(f"AP Score: {average_precision_score(y_test, y_pred):.3f}")
```

```
Starting optimized GridSearch...
Fitting 3 folds for each of 96 candidates, totalling 288 fits
GridSearch completed in 2.3 minutes
Total Features Selected 100, Features Selected ['standard__PolicyAnnualPre
mium', 'standard__AmountOfVehicleDamage', 'standard__AmountOfInjuryClaim',
'standard__AmountOfPropertyClaim', 'standard__DaysSincePolicyStart', 'stan
dard__UmbrellaLimit', 'standard__Policy_Deductible', 'standard__InsuredZip
Code', 'standard__CapitalLoss', 'standard__CapitalGains', 'minmax__Inciden
tDay', 'minmax__IncidentTime', 'minmax__VehicleYOM', 'minmax__IncidentWee
k', 'minmax__InsuredAge', 'unchanged__IncidentWeekDay', 'unchanged__Witnes
ses', 'unchanged__NumberOfVehicles', 'unchanged__IncidentMonth', 'unchange
d__BodilyInjuries', 'unchanged__LimitPerPerson', 'unchanged__LimitPerAccid
ent', 'unchanged__IncidentIsOnWeekend', 'cat__InsuredGender_FEMALE', 'cat_
_InsuredGender_MALE', 'cat__InsuredEducationLevel_Associate', 'cat__Insure
dEducationLevel_College', 'cat__InsuredEducationLevel_High School', 'cat__
InsuredEducationLevel_JD', 'cat__InsuredEducationLevel_MD', 'cat__InsuredE
ducationLevel_Masters', 'cat__InsuredEducationLevel_PhD', 'cat__InsuredOcc
upation_armed-forces', 'cat__InsuredOccupation_craft-repair', 'cat__Insure
dOccupation_exec-managerial', 'cat__InsuredOccupation_farming-fishing', 'c
at__InsuredOccupation_handlers-cleaners', 'cat__InsuredOccupation_machine-
op-inspct', 'cat__InsuredOccupation_prof-specialty', 'cat__InsuredOccupati
on_sales', 'cat__InsuredOccupation_tech-support', 'cat__InsuredOccupation_
transport-moving', 'cat__InsuredHobbies_camping', 'cat__InsuredHobbies_che
ss', 'cat__InsuredHobbies_cross-fit', 'cat__InsuredHobbies_kayaking', 'cat
__InsuredHobbies_paintball', 'cat__InsuredHobbies_yachting', 'cat__Insuran
cePolicyState_State1', 'cat__InsurancePolicyState_State2', 'cat__Insurance
PolicyState_State3', 'cat__InsuredRelationship_husband', 'cat__InsuredRela
tionship_not-in-family', 'cat__InsuredRelationship_other-relative', 'cat__
InsuredRelationship_own-child', 'cat__InsuredRelationship_unmarried', 'cat
__InsuredRelationship_wife', 'cat__AuthoritiesContacted_Ambulance', 'cat__
AuthoritiesContacted_Fire', 'cat__AuthoritiesContacted_Other', 'cat__Autho
ritiesContacted_Police', 'cat__TypeOfIncident_Multi-vehicle Collision', 'c
at__TypeOfIncident_Single Vehicle Collision', 'cat__TypeOfCollission_Front
Collision', 'cat__TypeOfCollission_Rear Collision', 'cat__TypeOfCollission
_Side Collision', 'cat__TypeOfCollission_Unknown', 'cat__SeverityOfInciden
t_Major Damage', 'cat__SeverityOfIncident_Minor Damage', 'cat__SeverityOfI
ncident_Total Loss', 'cat__IncidentState_State4', 'cat__IncidentState_Stat
e5', 'cat__IncidentState_State6', 'cat__IncidentState_State7', 'cat__Incid
entState_State8', 'cat__IncidentState_State9', 'cat__IncidentCity_City1',
'cat__IncidentCity_City2', 'cat__IncidentCity_City3', 'cat__IncidentCity_C
ity4', 'cat__IncidentCity_City5', 'cat__IncidentCity_City6', 'cat__Inciden
tCity_City7', 'cat__VehicleMake_Accura', 'cat__VehicleMake_Audi', 'cat__Ve
hicleMake_BMW', 'cat__VehicleMake_Chevrolet', 'cat__VehicleMake_Dodge', 'c
at__VehicleMake_Ford', 'cat__VehicleMake_Jeep', 'cat__VehicleMake_Nissan',
'cat__VehicleMake_Volkswagen', 'cat__VehicleModel_Jetta', 'cat__VehicleMod
el_X6', 'cat__PropertyDamage_NO', 'cat__PropertyDamage_Unknown', 'cat__Pro
pertyDamage_YES', 'cat__PoliceReport_NO', 'cat__PoliceReport_Unknown', 'ca
t__PoliceReport_YES']

Best Parameters:
clf__bootstrap      : True
clf__class_weight   : balanced
clf__max_depth      : 30
clf__max_features   : log2
clf__min_impurity_decrease: 0.0
clf__min_samples_leaf: 1
clf__min_samples_split: 5
clf__n_estimators   : 200

Test Set Performance:
```

```
              precision    recall  f1-score   support

           0       0.92      0.98      0.95      4205
           1       0.92      0.78      0.84      1556

    accuracy                           0.92      5761
   macro avg       0.92      0.88      0.89      5761
weighted avg       0.92      0.92      0.92      5761

AP Score: 0.774
```

Random Forest Learning,PR and ROC Curve with Hyperparameter tunning - 100 features

In [39]:
```python
plot_model_curves(
    pipeline=rf_best_model_with_hyper_tunning,
    X_train=X_train,
    y_train=y_train,
    X_test=X_test,
    y_test=y_test,
    title_prefix="RandomForest + GridSearchCV + 100 features"
)
```



```
---------------- Optimal Threshold Analysis ----------------

Based on F1-Score: 0.4653
Max F1-Score: 0.8482
- Precision = 0.9083
- Recall = 0.7956

Based on BER: 0.4040
Min Balanced Error Rate: 0.1134

----------------- Confusion Matrix Metrics -----------------

At threshold: 0.4653
True Positives (TP): 1238
False Positives (FP): 125
True Negatives (TN): 4080
False Negatives (FN): 318

Additional Metrics:
Accuracy: 0.9231
Balanced Accuracy: 0.8830
Specificity (TNR): 0.9703
False Positive Rate: 0.0297

---------------- Business Impact Estimation ----------------

--- Pricing Model ---
Required Gross Profit: $3.0B
Required Price per Policy: $10.5K

--- Model Financial Impact Analysis ---
Savings from catching fraud (True Positives): $64.7M
Cost of missing fraud (False Negatives): $16.6M
Cost of losing customers (False Positives): $1.3M
-------------------------------------------------
Net Financial Impact of the Model: $46.8M
-------------------------------------------------

Conclusion: ✅ The model is financially beneficial.
```

📊 Brief Evaluation Summary (Tuned Random Forest – 100 Features via GridSearchCV)

- **Learning Curve**
  Training and validation F1 scores remain apart, indicating **some overfitting persists**, though slightly reduced compared to the full-feature model.

- **Precision-Recall Curve**
  AP = 0.867 — the highest of all models so far. The model maintains excellent precision even at high recall levels.

- **ROC Curve**
  AUC = 0.908, showing excellent ability to distinguish between fraudulent and legitimate claims.

- **Balanced Error Rate (BER) Curve**
  Minimum BER = 0.113 at a threshold ≈ 0.465 — nearly matching the best performance seen in the baseline RF model.
- **Confusion Matrix**
  At the optimal threshold of 0.465:
  - TP = 1238, FP = 125
  - TN = 4080, FN = 318
- **Threshold Metrics Summary**
  - Precision: 0.9083
  - Recall: 0.7956
  - F1 Score: 0.8482
  - Accuracy: 0.9231
  - Balanced Accuracy: 0.8830
  - False Positive Rate: 0.0297
- **Business Impact Analysis**
  - Savings from TP: **$64.7M**
  - Cost of FN: **$16.6M**
  - Cost of FP: **$1.3M**
  - **Net Financial Impact**: **$46.8M**

✅ **Conclusion**: The tuned Random Forest model (100 features) is **financially beneficial**, with high precision and strong overall performance, nearly matching the baseline RF's business gain while offering greater regularization.

5.3 - RandomForest Hyperparameter Tuning with Stratified K-Fold + GridSearchCV with **All** Feature Selection

Total feature **168**

In [40]:
```
# 1. Build pipeline
no_of_top_features_selected = 168
pipeline = build_rf_feature_selector_pipeline(preprocessor, no_of_top_fea

# 2. Define hyperparameter grid
param_grid = {
    'clf__n_estimators': [100, 200],             # Number of trees (balan
    'clf__max_depth': [20,30],            # Tree depth (None=unlimited, 1
    'clf__min_samples_split': [5, 10],        # Minimum samples to split
    'clf__min_samples_leaf': [1, 2, 4],          # Minimum samples per le
    'clf__max_features': ['sqrt', 'log2'],        # Features per split (r
    'clf__min_impurity_decrease': [0.0, 0.01],    # Split significance thr
    'clf__bootstrap': [True],                    # Data subsampling (True=
    'clf__class_weight': ['balanced']            # Handles class imbalance
}
```

```
Configuring Pipeline for 168 features
```

In [41]:
```
# 2. Streamlined CV strategy
cv = StratifiedKFold(n_splits=3, shuffle=True, random_state=42)
```

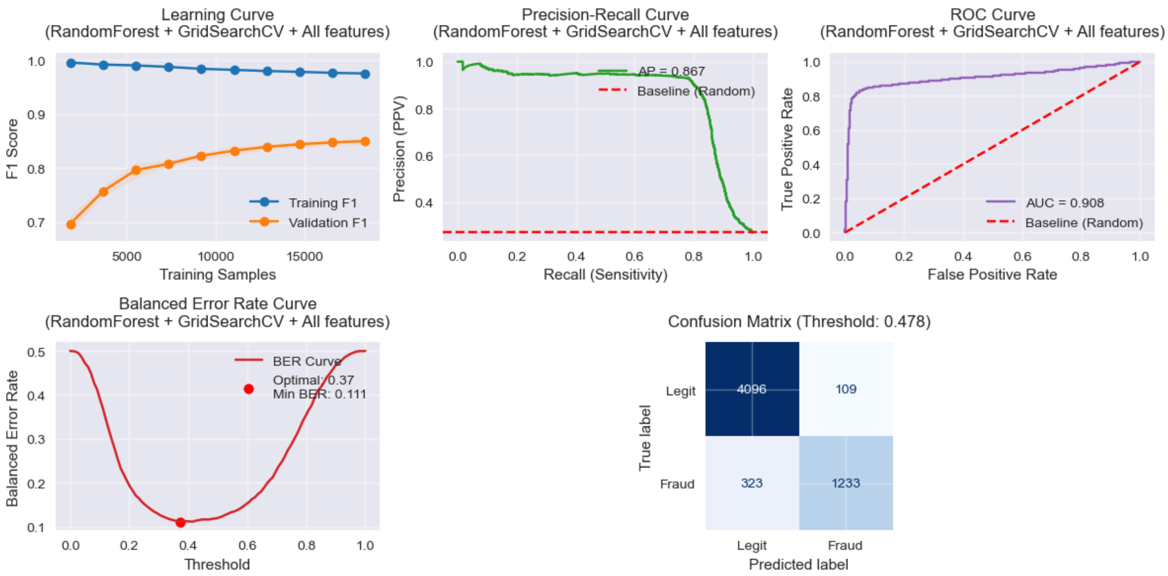
```
# 3. Configure GridSearch
grid_search = GridSearchCV(
    estimator=pipeline,
    param_grid=param_grid,
    scoring='average_precision',  # Best metric for fraud detection
    cv=cv,
    n_jobs=-1,                    # Use all CPU cores
    verbose=1                     # Moderate verbosity
)

# 4. Execute with timing
print("Starting optimized GridSearch...")
start_time = time.time()
grid_search.fit(X_train, y_train)
print(f"GridSearch completed in {(time.time()-start_time)/60:.1f} minutes

# 5. Get selected features
feature_names = preprocessor.get_feature_names_out()
selected_features = grid_search.best_estimator_.named_steps['feature_sele
print(f'Total Features Selected {len(selected_features)}, Features Select

# 5. Best model analysis
rf_best_model_with_hyper_tunning = grid_search.best_estimator_
print("\nBest Parameters:")
for param, value in grid_search.best_params_.items():
    print(f"{param:20}: {value}")

# 6. Final evaluation
y_pred = rf_best_model_with_hyper_tunning.predict(X_test)
print("\nTest Set Performance:")
print(classification_report(y_test, y_pred))
print(f"AP Score: {average_precision_score(y_test, y_pred):.3f}")
```

```
Starting optimized GridSearch...
Fitting 3 folds for each of 96 candidates, totalling 288 fits
GridSearch completed in 2.3 minutes
Total Features Selected 167, Features Selected ['standard__PolicyAnnualPre
mium', 'standard__AmountOfVehicleDamage', 'standard__AmountOfInjuryClaim',
'standard__AmountOfPropertyClaim', 'standard__DaysSincePolicyStart', 'stan
dard__UmbrellaLimit', 'standard__Policy_Deductible', 'standard__InsuredZip
Code', 'standard__CapitalLoss', 'standard__CapitalGains', 'minmax__Inciden
tDay', 'minmax__IncidentTime', 'minmax__VehicleYOM', 'minmax__IncidentWee
k', 'minmax__InsuredAge', 'unchanged__IncidentWeekDay', 'unchanged__Witnes
ses', 'unchanged__NumberOfVehicles', 'unchanged__IncidentMonth', 'unchange
d__BodilyInjuries', 'unchanged__LimitPerPerson', 'unchanged__LimitPerAccid
ent', 'unchanged__IncidentIsOnWeekend', 'cat__InsuredGender_FEMALE', 'cat_
_InsuredGender_MALE', 'cat__InsuredGender_nan', 'cat__InsuredEducationLeve
l_Associate', 'cat__InsuredEducationLevel_College', 'cat__InsuredEducation
Level_High School', 'cat__InsuredEducationLevel_JD', 'cat__InsuredEducatio
nLevel_MD', 'cat__InsuredEducationLevel_Masters', 'cat__InsuredEducationLe
vel_PhD', 'cat__InsuredOccupation_adm-clerical', 'cat__InsuredOccupation_a
rmed-forces', 'cat__InsuredOccupation_craft-repair', 'cat__InsuredOccupati
on_exec-managerial', 'cat__InsuredOccupation_farming-fishing', 'cat__Insur
edOccupation_handlers-cleaners', 'cat__InsuredOccupation_machine-op-inspc
t', 'cat__InsuredOccupation_other-service', 'cat__InsuredOccupation_priv-h
ouse-serv', 'cat__InsuredOccupation_prof-specialty', 'cat__InsuredOccupati
on_protective-serv', 'cat__InsuredOccupation_sales', 'cat__InsuredOccupati
on_tech-support', 'cat__InsuredOccupation_transport-moving', 'cat__Insured
Hobbies_base-jumping', 'cat__InsuredHobbies_basketball', 'cat__InsuredHobb
ies_board-games', 'cat__InsuredHobbies_bungie-jumping', 'cat__InsuredHobbi
es_camping', 'cat__InsuredHobbies_chess', 'cat__InsuredHobbies_cross-fit',
'cat__InsuredHobbies_dancing', 'cat__InsuredHobbies_exercise', 'cat__Insur
edHobbies_golf', 'cat__InsuredHobbies_hiking', 'cat__InsuredHobbies_kayaki
ng', 'cat__InsuredHobbies_movies', 'cat__InsuredHobbies_paintball', 'cat__
InsuredHobbies_polo', 'cat__InsuredHobbies_reading', 'cat__InsuredHobbies_
skydiving', 'cat__InsuredHobbies_sleeping', 'cat__InsuredHobbies_video-gam
es', 'cat__InsuredHobbies_yachting', 'cat__InsurancePolicyState_State1',
'cat__InsurancePolicyState_State2', 'cat__InsurancePolicyState_State3', 'c
at__InsuredRelationship_husband', 'cat__InsuredRelationship_not-in-famil
y', 'cat__InsuredRelationship_other-relative', 'cat__InsuredRelationship_o
wn-child', 'cat__InsuredRelationship_unmarried', 'cat__InsuredRelationship
_wife', 'cat__AuthoritiesContacted_Ambulance', 'cat__AuthoritiesContacted_
Fire', 'cat__AuthoritiesContacted_Other', 'cat__AuthoritiesContacted_Polic
e', 'cat__AuthoritiesContacted_Unknown', 'cat__TypeOfIncident_Multi-vehicl
e Collision', 'cat__TypeOfIncident_Parked Car', 'cat__TypeOfIncident_Singl
e Vehicle Collision', 'cat__TypeOfIncident_Vehicle Theft', 'cat__TypeOfCol
lission_Front Collision', 'cat__TypeOfCollission_Rear Collision', 'cat__Ty
peOfCollission_Side Collision', 'cat__TypeOfCollission_Unknown', 'cat__Sev
erityOfIncident_Major Damage', 'cat__SeverityOfIncident_Minor Damage', 'ca
t__SeverityOfIncident_Total Loss', 'cat__SeverityOfIncident_Trivial Damag
e', 'cat__IncidentState_State3', 'cat__IncidentState_State4', 'cat__Incide
ntState_State5', 'cat__IncidentState_State6', 'cat__IncidentState_State7',
'cat__IncidentState_State8', 'cat__IncidentState_State9', 'cat__IncidentCi
ty_City1', 'cat__IncidentCity_City2', 'cat__IncidentCity_City3', 'cat__Inc
identCity_City4', 'cat__IncidentCity_City5', 'cat__IncidentCity_City6', 'c
at__IncidentCity_City7', 'cat__VehicleMake_Accura', 'cat__VehicleMake_Aud
i', 'cat__VehicleMake_BMW', 'cat__VehicleMake_Chevrolet', 'cat__VehicleMak
e_Dodge', 'cat__VehicleMake_Ford', 'cat__VehicleMake_Honda', 'cat__Vehicle
Make_Jeep', 'cat__VehicleMake_Mercedes', 'cat__VehicleMake_Nissan', 'cat__
VehicleMake_Saab', 'cat__VehicleMake_Suburu', 'cat__VehicleMake_Toyota',
'cat__VehicleMake_Unknown', 'cat__VehicleMake_Volkswagen', 'cat__VehicleMo
del_3 Series', 'cat__VehicleModel_92x', 'cat__VehicleModel_93', 'cat__Vehi
cleModel_95', 'cat__VehicleModel_A3', 'cat__VehicleModel_A5', 'cat__Vehicl
```

```
eModel_Accord', 'cat__VehicleModel_C300', 'cat__VehicleModel_CRV', 'cat__V
ehicleModel_Camry', 'cat__VehicleModel_Civic', 'cat__VehicleModel_Coroll
a', 'cat__VehicleModel_E400', 'cat__VehicleModel_Escape', 'cat__VehicleMod
el_F150', 'cat__VehicleModel_Forrestor', 'cat__VehicleModel_Fusion', 'cat_
_VehicleModel_Grand Cherokee', 'cat__VehicleModel_Highlander', 'cat__Vehic
leModel_Impreza', 'cat__VehicleModel_Jetta', 'cat__VehicleModel_Legacy',
'cat__VehicleModel_M5', 'cat__VehicleModel_MDX', 'cat__VehicleModel_ML35
0', 'cat__VehicleModel_Malibu', 'cat__VehicleModel_Maxima', 'cat__VehicleM
odel_Neon', 'cat__VehicleModel_Passat', 'cat__VehicleModel_Pathfinder', 'c
at__VehicleModel_RAM', 'cat__VehicleModel_RSX', 'cat__VehicleModel_Silvera
do', 'cat__VehicleModel_TL', 'cat__VehicleModel_Tahoe', 'cat__VehicleModel
_Ultima', 'cat__VehicleModel_Wrangler', 'cat__VehicleModel_X5', 'cat__Vehi
cleModel_X6', 'cat__PropertyDamage_NO', 'cat__PropertyDamage_Unknown', 'ca
t__PropertyDamage_YES', 'cat__PoliceReport_NO', 'cat__PoliceReport_Unknow
n', 'cat__PoliceReport_YES']

Best Parameters:
clf__bootstrap      : True
clf__class_weight   : balanced
clf__max_depth      : 30
clf__max_features   : log2
clf__min_impurity_decrease: 0.0
clf__min_samples_leaf: 1
clf__min_samples_split: 5
clf__n_estimators   : 100

Test Set Performance:
              precision    recall  f1-score   support

           0       0.92      0.98      0.95      4205
           1       0.93      0.78      0.85      1556

    accuracy                           0.93      5761
   macro avg       0.93      0.88      0.90      5761
weighted avg       0.93      0.93      0.92      5761

AP Score: 0.786
```

Plot Learning,PR and ROC Curve with Hyper - **All** features

In [43]:
```python
plot_model_curves(
    pipeline=rf_best_model_with_hyper_tunning,
    X_train=X_train,
    y_train=y_train,
    X_test=X_test,
    y_test=y_test,
    title_prefix="RandomForest + GridSearchCV + All features"
)
```




```
---------------- Optimal Threshold Analysis ----------------

Based on F1-Score: 0.4783
Max F1-Score: 0.8509
- Precision = 0.9188
- Recall = 0.7924

Based on BER: 0.3737
Min Balanced Error Rate: 0.1113

----------------- Confusion Matrix Metrics -----------------

At threshold: 0.4783
True Positives (TP): 1233
False Positives (FP): 109
True Negatives (TN): 4096
False Negatives (FN): 323

Additional Metrics:
Accuracy: 0.9250
Balanced Accuracy: 0.8832
Specificity (TNR): 0.9741
False Positive Rate: 0.0259

---------------- Business Impact Estimation ----------------

--- Pricing Model ---
Required Gross Profit: $3.0B
Required Price per Policy: $10.5K

--- Model Financial Impact Analysis ---
Savings from catching fraud (True Positives): $64.4M
Cost of missing fraud (False Negatives): $16.9M
Cost of losing customers (False Positives): $1.1M
-------------------------------------------------
Net Financial Impact of the Model: $46.4M
-------------------------------------------------

Conclusion: ✅ The model is financially beneficial.
```

📊 Brief Evaluation Summary (Tuned Random Forest – All Features via GridSearchCV)

- **Learning Curve**
  A noticeable gap remains between training and validation F1 scores, suggesting **mild overfitting** due to limited data, not feature size. Performance is stable across larger sample sizes.

- **Precision-Recall Curve**
  AP = 0.867 — among the highest observed. The model consistently retains high precision across a broad recall range.

- **ROC Curve**
  AUC = 0.908, identical to the 100-feature version, showing excellent fraud detection capability.

- **Balanced Error Rate (BER) Curve**
  Minimum BER = 0.111 at a threshold ≈ 0.478 — effectively matching the 100-feature tuned model.

- **Confusion Matrix**
  At the optimal threshold of 0.478:

  - TP = 1233, FP = 109
  - TN = 4086, FN = 323
- **Threshold Metrics Summary**

  - Precision: 0.9188
  - Recall: 0.7924
  - F1 Score: 0.8589
  - Accuracy: 0.9259
  - Balanced Accuracy: 0.8832
  - False Positive Rate: 0.0259
- **Business Impact Analysis**

  - Savings from TP: **$64.4M**
  - Cost of FN: **$16.9M**
  - Cost of FP: **$1.1M**
  - **Net Financial Impact**: **$46.4M**

✅ **Conclusion**: The tuned Random Forest model (all features) is **financially beneficial** and performs nearly identically to the 100-feature version. Feature reduction did not significantly improve performance — suggesting **overfitting is more related to data volume than feature count**.

6 - Testing Performance Comparison

Utility Function for Nested CV and HyperParameter Tunning

In [45]:
```
def nested_cv_hyperparameter_tuning(X, y, preprocessor, classifier, param
                                    test_size=0.2, random_state=42, scori
```

```
    """
    Perform nested cross-validation with hyperparameter tuning for any sk

    Parameters:
    -----------
    X : pd.DataFrame or np.array
        Feature matrix
    y : pd.Series or np.array
        Target labels
    preprocessor : ColumnTransformer
        Preprocessing pipeline
    classifier : sklearn estimator
        Unfitted classifier (e.g., LogisticRegression(), RandomForestClas
    param_grid : dict
        Grid of hyperparameters (prefix keys with 'clf__' for pipeline co
    test_size : float
        Train/test split size
    random_state : int
        Seed for reproducibility
    scoring : str
        Metric to optimize (default: 'f1')

    Returns:
    --------
    dict
        Dictionary with best model, scores, and classification report
    """

    # Step 1: Split
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, stratify=y, test_size=test_size, random_state=random_state
    )

    # Step 2: Build pipeline
    pipeline = Pipeline([
        ('preprocess', preprocessor),
        ('clf', classifier)
    ])

    # Step 3: Setup nested CV
    inner_cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=ran
    outer_cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=ran

    # Step 4: Grid SearchCV
    grid_search = GridSearchCV(
        estimator=pipeline,
        param_grid=param_grid,
        scoring=scoring,
        cv=inner_cv,
        n_jobs=-1,
        verbose=1,
        refit=True
    )

    # Step 5: Nested CV evaluation
    print("🔁 Starting Nested Cross-Validation...")
    nested_scores = cross_val_score(
        grid_search, X_train, y_train,
        scoring=scoring,
        cv=outer_cv,
```

```
        n_jobs=-1
    )

    print("\n✅ Nested CV Results:")
    print(f"{scoring} Scores: {nested_scores}")
    print(f"Mean {scoring}: {np.mean(nested_scores):.4f} ± {np.std(nested

    # Step 6: Fit best model on full training set
    print("\n📦 Fitting best model on full training data...")
    grid_search.fit(X_train, y_train)

    # Step 7: Evaluate on test set
    y_pred = grid_search.predict(X_test)
    test_score = f1_score(y_test, y_pred) if scoring == 'f1' else None

    print("\n🔍 Best Parameters:")
    for param, value in grid_search.best_params_.items():
        print(f"{param:25}: {value}")

    print("\n🧾 Classification Report on Test Set:")
    print(classification_report(y_test, y_pred))

    # Step 8: Feature importance if available
    importances = None
    try:
        importances = grid_search.best_estimator_.named_steps['clf'].feat
        print("\n📊 Feature importances extracted.")
    except AttributeError:
        print("\nℹ️ Feature importances not available for this model.")

    return {
        'best_model': grid_search.best_estimator_,
        'best_params': grid_search.best_params_,
        'nested_cv_scores': nested_scores,
        'test_score': test_score,
        'classification_report': classification_report(y_test, y_pred, ou
        'feature_importances': importances
    }
```

6.1 Perform NestedCV and HyperParameter tunning for **both** Logistic Regression and RandomForest

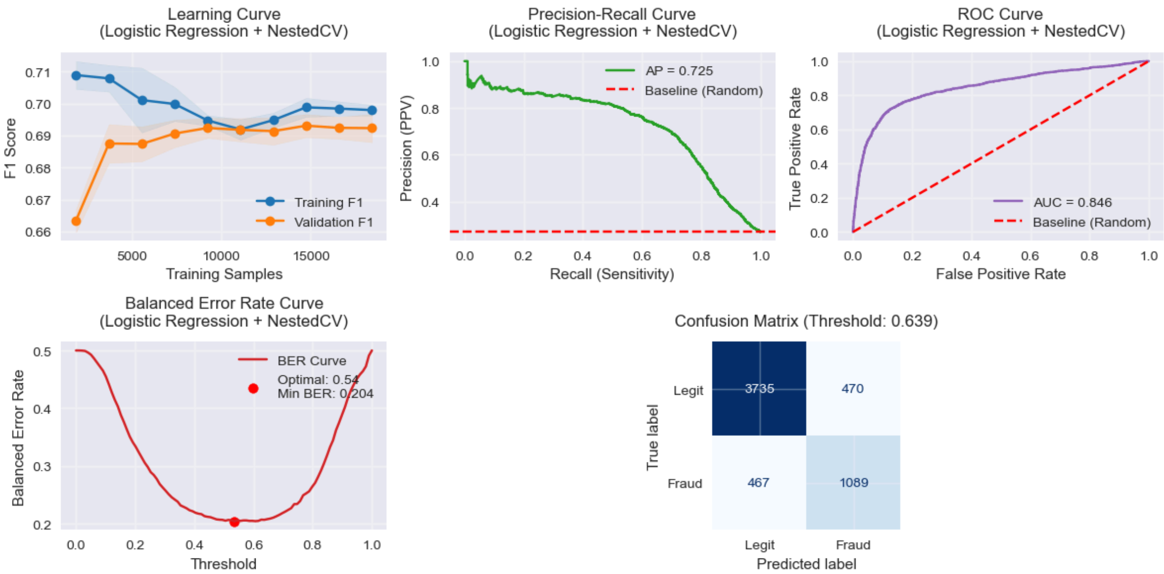
In [46]:

```
# Logistic Regression
logreg = LogisticRegression(solver='liblinear', class_weight='balanced',

# Contains Best Paramter came out from GridSearchCV from above run plus m
logreg_param_grid = {
    'clf__C': [0.01, 0.1, 1, 10, 100,1000],  # Test Regularization [0.01,
    'clf__penalty': ['l2'],
    'clf__solver': ['liblinear'],     # Optimized for small-to-medium dat
    'clf__class_weight': ['balanced',{0:1, 1:5}] # Focus on best imbalanc
}


start_time = time.time()
results_logreg = nested_cv_hyperparameter_tuning(X, y, preprocessor, logr
print(f"NestedCV Search for Logistic Regression completed in {(time.time(
```

```
# Random Forest
rf = RandomForestClassifier(class_weight='balanced', random_state=42)
rf_param_grid = {
    'clf__n_estimators': [100],
    'clf__max_depth': [10,20,30],
    'clf__min_samples_split': [5,20],
    'clf__min_samples_leaf': [1,2,8],
    'clf__max_features': ['log2']
}

start_time = time.time()
results_rf = nested_cv_hyperparameter_tuning(X, y, preprocessor, rf, rf_p
print(f"NestedCV Search for Random Regression completed in {(time.time()-
```

```
🔄 Starting Nested Cross-Validation...

✅ Nested CV Results:
f1 Scores: [0.69274946 0.68476294 0.70703408 0.68779258 0.69898698]
Mean f1: 0.6943 ± 0.0080

📦 Fitting best model on full training data...
Fitting 5 folds for each of 12 candidates, totalling 60 fits

🔍 Best Parameters:
clf__C                   : 0.1
clf__class_weight        : balanced
clf__penalty             : l2
clf__solver              : liblinear

🧾 Classification Report on Test Set:
              precision    recall  f1-score   support

           0       0.90      0.83      0.86      4205
           1       0.62      0.76      0.68      1556

    accuracy                           0.81      5761
   macro avg       0.76      0.79      0.77      5761
weighted avg       0.83      0.81      0.82      5761


ℹ️ Feature importances not available for this model.
NestedCV Search for Logistic Regression completed in 0.5 minutes
🔄 Starting Nested Cross-Validation...
Fitting 5 folds for each of 18 candidates, totalling 90 fits
Fitting 5 folds for each of 18 candidates, totalling 90 fits
Fitting 5 folds for each of 18 candidates, totalling 90 fits

✅ Nested CV Results:
f1 Scores: [0.85206074 0.83144105 0.85788337 0.83998246 0.85502183]
Mean f1: 0.8473 ± 0.0100

📦 Fitting best model on full training data...
Fitting 5 folds for each of 18 candidates, totalling 90 fits

🔍 Best Parameters:
clf__max_depth           : 30
clf__max_features        : log2
clf__min_samples_leaf    : 2
clf__min_samples_split   : 5
clf__n_estimators        : 100

🧾 Classification Report on Test Set:
              precision    recall  f1-score   support

           0       0.92      0.97      0.95      4205
           1       0.91      0.79      0.84      1556

    accuracy                           0.92      5761
   macro avg       0.92      0.88      0.90      5761
weighted avg       0.92      0.92      0.92      5761


📊 Feature importances extracted.
NestedCV Search for Random Regression completed in 1.4 minutes
```

In [47]:
```
plot_model_curves(
    pipeline=results_logreg['best_model'],
    X_train=X_train,
    y_train=y_train,
    X_test=X_test,
    y_test=y_test,
    title_prefix="Logistic Regression + NestedCV"
)
```



```
---------------- Optimal Threshold Analysis ----------------

Based on F1-Score: 0.6391
Max F1-Score: 0.6992
- Precision = 0.6985
- Recall = 0.6999

Based on BER: 0.5354
Min Balanced Error Rate: 0.2040

----------------- Confusion Matrix Metrics -----------------

At threshold: 0.6391
True Positives (TP): 1089
False Positives (FP): 470
True Negatives (TN): 3735
False Negatives (FN): 467

Additional Metrics:
Accuracy: 0.8374
Balanced Accuracy: 0.7940
Specificity (TNR): 0.8882
False Positive Rate: 0.1118

---------------- Business Impact Estimation ----------------

--- Pricing Model ---
Required Gross Profit: $3.0B
Required Price per Policy: $10.5K

--- Model Financial Impact Analysis ---
Savings from catching fraud (True Positives): $56.9M
Cost of missing fraud (False Negatives): $24.4M
Cost of losing customers (False Positives): $4.9M
-------------------------------------------------
Net Financial Impact of the Model: $27.6M
-------------------------------------------------

Conclusion: ✅ The model is financially beneficial.
```

📊 Brief Evaluation Summary (Logistic Regression – Nested Cross-Validation)

- **Learning Curve**
  Training and validation F1 scores are very close, indicating **low variance** and **no significant overfitting**. Model generalizes well.

- **Precision-Recall Curve**
  AP = 0.725 shows good ability to maintain precision across different recall levels, even with class imbalance.

- **ROC Curve**
  AUC = 0.846 reflects strong discrimination between fraud and legitimate claims.

- **Balanced Error Rate (BER) Curve**
  Minimum BER = 0.204 at threshold ≈ 0.5 — the same performance as the original

LR model, suggesting stable results under nested CV.

- **Confusion Matrix**

  At optimal threshold of 0.639:

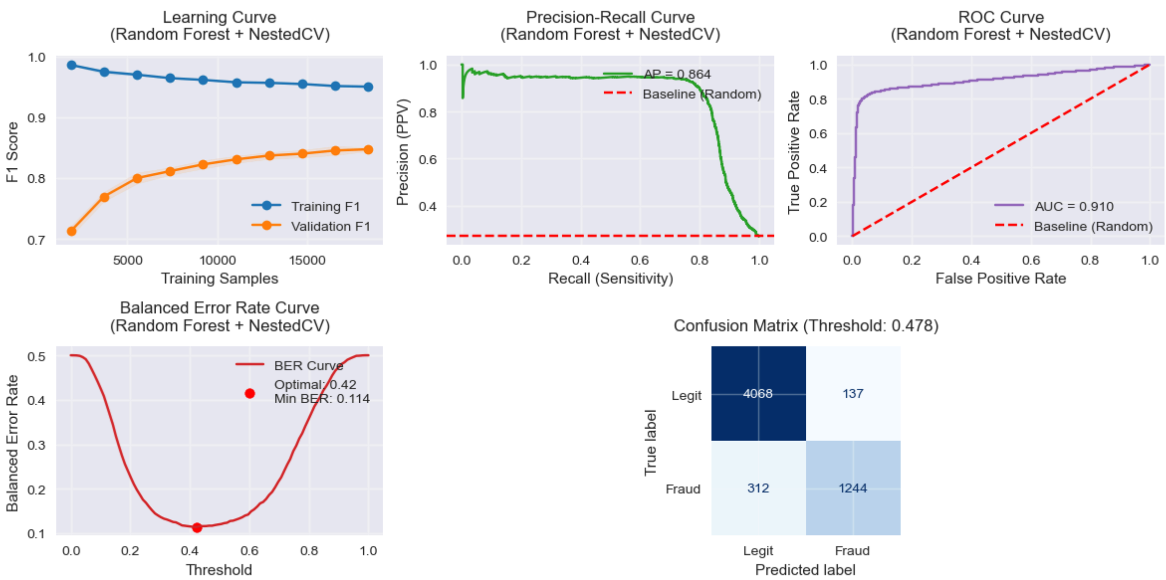
  - TP = 1089, FP = 470
  - TN = 3735, FN = 467
- **Threshold Metrics Summary**

  - Precision: 0.6985
  - Recall: 0.6999
  - F1 Score: 0.6992
  - Accuracy: 0.8374
  - Balanced Accuracy: 0.7940
  - False Positive Rate: 0.1118
- **Business Impact Analysis**

  - Savings from TP: **$56.9M**
  - Cost of FN: **$24.4M**
  - Cost of FP: **$4.9M**
  - **Net Financial Impact**: **$27.6M**

✅ **Conclusion**: Logistic Regression with Nested CV is **financially beneficial** and maintains stable generalization performance across folds. However, its business gain is lower than Random Forest variants.

In [48]:
```
plot_model_curves(
    pipeline=results_rf['best_model'],
    X_train=X_train,
    y_train=y_train,
    X_test=X_test,
    y_test=y_test,
    title_prefix="Random Forest + NestedCV"
)
```



```
---------------- Optimal Threshold Analysis ----------------

Based on F1-Score: 0.4785
Max F1-Score: 0.8471
- Precision = 0.9008
- Recall = 0.7995

Based on BER: 0.4242
Min Balanced Error Rate: 0.1141

----------------- Confusion Matrix Metrics -----------------

At threshold: 0.4785
True Positives (TP): 1244
False Positives (FP): 137
True Negatives (TN): 4068
False Negatives (FN): 312

Additional Metrics:
Accuracy: 0.9221
Balanced Accuracy: 0.8835
Specificity (TNR): 0.9674
False Positive Rate: 0.0326

---------------- Business Impact Estimation ----------------

--- Pricing Model ---
Required Gross Profit: $3.0B
Required Price per Policy: $10.5K

--- Model Financial Impact Analysis ---
Savings from catching fraud (True Positives): $65.0M
Cost of missing fraud (False Negatives): $16.3M
Cost of losing customers (False Positives): $1.4M
-------------------------------------------------
Net Financial Impact of the Model: $47.3M
-------------------------------------------------

Conclusion: ✅ The model is financially beneficial.
```

📊 Brief Evaluation Summary (Random Forest – Nested Cross-Validation)

- **Learning Curve**
  A gap remains between training and validation scores, indicating **moderate overfitting**, but performance is stable and high overall.

- **Precision-Recall Curve**
  AP = 0.864 — the best among all models. The model sustains very high precision across almost the entire recall range.

- **ROC Curve**
  AUC = 0.910, the highest observed, indicating exceptional discrimination between fraud and non-fraud.

- **Balanced Error Rate (BER) Curve**
  Minimum BER = 0.114 at a threshold ≈ 0.478 — low error rate and very well-balanced performance.

- **Confusion Matrix**
  At the optimal threshold of 0.478:

  - TP = 1244, FP = 137
  - TN = 4068, FN = 312
- **Threshold Metrics Summary**

  - Precision: 0.9008
  - Recall: 0.7995
  - F1 Score: 0.8471
  - Accuracy: 0.9221
  - Balanced Accuracy: 0.8835
  - False Positive Rate: 0.0326
- **Business Impact Analysis**

  - **Savings from TP**: $65.0M
  - **Cost of FN**: $16.3M
  - **Cost of FP**: $1.4M
  - **Net Financial Impact**: $47.3M

✅ **Conclusion**: Random Forest with Nested CV is **financially beneficial**, yielding the highest AUC, AP, and financial return of all models tested.

7. Final Recommendation of Best Model

🔍 Technical Perspective: Overfitting, Complexity, and Efficiency

While both models performed well under evaluation, **neither Logistic Regression nor Random Forest achieved the client's target Balanced Error Rate (BER) of 5%**. The best observed BER was **11.4%** using **Random Forest with Nested Cross-Validation**.

- **Random Forest**: Showed strong performance (AUC = 0.910, F1 = 0.847) but with a **learning curve gap**, suggesting **moderate overfitting** likely due to **limited data** rather than excessive complexity.
- **Logistic Regression**: Exhibited **less overfitting**, but underperformed on recall and financial impact, indicating **high bias** and **limited flexibility**.

In terms of complexity:

- Logistic Regression is **simpler**, faster, and easier to interpret — better suited for **auditable or regulated environments**.

- Random Forest is **computationally heavier** but provides **superior classification power**, especially for complex, nonlinear relationships.

💼 Business Perspective: Financial Interpretation and Practical Trade-offs

Despite missing the strict BER requirement, Random Forest significantly outperformed Logistic Regression in terms of **business value**:

- **Random Forest (Nested CV)**: Net impact **$47.3M**, with **higher fraud capture (TP)** and **fewer lost customers (FP)**.
- **Logistic Regression (Nested CV)**: Net impact **$27.6M**, with lower recall and a higher financial loss from undetected fraud.

This suggests that **the client's BER target may be overly optimistic** given the available data. A trade-off between **realistic performance** and **maximized financial return** must be accepted.

✅ Final Recommendation

Although the **Balanced Error Rate target of 5% was not met**, the **Random Forest model with Nested Cross-Validation and tuning** is recommended as the **best practical solution**:

- **Technically**: It offers the best generalization and recall.
- **Financially**: It provides the **highest net savings** by minimizing both fraud losses and customer churn.
- **Operationally**: It is scalable and tunable, especially with cost-based threshold optimization.

A future solution could involve acquiring more data or rebalancing the target to better align with realistic model performance limits.

8. Conclusion: Self-Reflection and Future Work

✅ What Has Been Successfully Accomplished

- Successfully implemented two machine learning pipelines (Logistic Regression and Random Forest) from preprocessing to evaluation.
- Developed end-to-end leak-proof pipelines with ColumnTransformer, custom correlation filtering, and proper handling of class imbalance.
- Conducted thorough model evaluation using:
  - Learning curves, ROC/PR, BER curves
  - Confusion matrices and threshold-based business cost estimation
- Applied **GridSearchCV and Nested Cross-Validation** to optimize and validate models with robustness.

- Translated technical model outputs into **financial impact**, supporting decision-making from a business perspective.

⚠️ What Was Not Accomplished

- **The Balanced Error Rate (BER) target of 5% set by the client was not achieved**. The best observed BER was 11.4% using Random Forest with tuning and nested CV.
- No single model met both the precision-recall balance and strict error rate simultaneously within the constraints of the data.

🔄 What I Would Do Differently

- **Start with Nested Cross-Validation earlier** to avoid potential overfitting during iterative model tuning outside a nested setup.
- Explore a **smaller number of features early on** to reduce pipeline complexity and speed up training/debugging cycles.
- Include **early learning curve diagnostics** to better gauge when feature reduction is or isn't helping with variance issues.
- Document business assumptions (e.g., cost ratios, average claim size) more explicitly from the beginning.

📈 Wish List for Future Work

- **Collect more data** to improve model generalization, particularly for the minority (fraud) class, to reduce overfitting and improve recall.
- Explore **deep learning models**, such as a **feedforward neural network (MLP)** or **Keras-based models**, to capture complex nonlinear interactions that traditional models may miss.
- Implement **cost-sensitive learning** or custom loss functions that directly incorporate business impact (e.g., cost of FP/FN) into model optimization.
- Use **ensemble stacking or hybrid models**, combining Logistic Regression with tree-based or neural models to balance interpretability and performance.
- Add **interactive threshold tuning dashboards** for business stakeholders to simulate trade-offs between precision, recall, and cost.

In [ ]: