✅ 1. Logistic Regression

python
```
LogisticRegression(class_weight="balanced", max_iter=1000)
```

🔍 Parameter Explanations:

Parameter	Description
`class_weight="balanced"`	This tells the model to give more weight to the **minority class** (fraud cases) to combat class imbalance. It automatically adjusts weights inversely proportional to class frequencies.
`max_iter=1000`	Sets the maximum number of iterations for the solver to converge. Logistic Regression is an iterative algorithm, and we increase this to ensure it fully converges on large datasets.

Additional Note:
In the grid search, we passed:

python
```
"model__C": [0.1, 1.0, 10]
```

- `C` is the **inverse of regularization strength** (λ).
    - Smaller `C` (e.g. 0.1) = **stronger regularization** (less overfitting).
    - Larger `C` = weaker regularization (model fits more closely to the data).
- Think of it like a bias-variance knob.

✅ 2. Random Forest Classifier

python
CopyEdit
```
RandomForestClassifier(class_weight="balanced", random_state=42)
```

🔍 Parameter Explanations:

Parameter	Description
`class_weight="balanced"`	Like Logistic Regression, this balances the fraud and non-fraud classes based on their frequencies.
`random_state=42`	Sets the seed for reproducibility — ensures consistent results every time the model is trained.

In the grid search:

python
CopyEdit
```
"model__n_estimators": [50, 100],
"model__max_depth": [10, 20]
```

Param	What it Does
`n_estimators`	Number of trees in the forest. More trees usually improve performance but increase training time.
`max_depth`	Maximum depth of each tree. Controls how deep the trees go (how complex they are). Prevents overfitting.

✅ 3. XGBoost Classifier

python
```
XGBClassifier(scale_pos_weight=100, use_label_encoder=False,
eval_metric="logloss")
```

🔍 Parameter Explanations:

Parameter	Description
`scale_pos_weight=100`	Very important for imbalanced data. This boosts the importance of the fraud class during training. Set approximately to the **ratio of non-fraud to fraud** in the training set.
`use_label_encoder=False`	Prevents a deprecation warning from older versions of XGBoost.

`eval_metric="logloss"`	Evaluation metric used during training. `logloss` is good for binary classification as it penalizes wrong confident predictions.

In the grid search:

python
CopyEdit
```
"model__n_estimators": [50, 100],
"model__max_depth": [3, 6]
```

Param	What it Does
`n_estimators`	Number of boosting rounds (trees). More = potentially better performance.
`max_depth`	Maximum depth of each tree. Lower = simpler trees = less overfitting.

🔁 Cross-Model: GridSearchCV

python
```
GridSearchCV(pipe, param_grid, scoring="f1", cv=3, n_jobs=-1)
```

Param	Description
`scoring="f1"`	F1-score is the **harmonic mean of precision and recall**. Perfect for imbalanced classification tasks.
`cv=3`	3-fold cross-validation: data is split into 3 parts to ensure robustness of results.
`n_jobs=-1`	Use all available CPU cores to train faster.

Bonus — SMOTE

python
```
SMOTE(random_state=42)
```

Param	Description

`random_state=42`	Reproducible results when generating synthetic samples of the minority class.

Summary Table

Model	Core Concept	Key Params (and Why)
Logistic Regression	Linear model, interpretable	`C`, `class_weight`
Random Forest	Ensemble of decision trees	`n_estimators`, `max_depth`, `class_weight`
XGBoost	Gradient-boosted trees, high performance	`scale_pos_weight`, `n_estimators`, `max_depth`