MODEL SELECTION AND TRIALS:

This is one of the models that I have deployed to solve the problem. The different models that I used while trying to solve the problem include:

- Random Forest Algorithm (Peak accuracy after tuning: 80.373%)
- XGBoost Algorithm (Peak accuracy after tuning: 79.127%)
- LightGBM Algorithm (Peak accuracy after tuning: 77.88%)
- CatBoost Algorithm (Peak accuracy after tuning: 80.996%)

Random Forest and CatBoost algorithms were performing almost equally well and owing to CatBoost's speciality of dealing with multiple dimensionality in data for pattern detection more effectively, it has a slight edge over Random Forest. Hence, the model was tuned accordingly, using a 5-fold CV approach and hyperparameter fine-tuning, which yielded a peak accuracy of 80.996% on 70% of the test data.

```
import pandas as pd
import numpy as np

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

from catboost import CatBoostClassifier
```

```
!unzip evacuation-readiness-who-makes-it-out.zip
```

```
Archive:  evacuation-readiness-who-makes-it-out.zip
  inflating: metadata.csv
  inflating: sample_submission.csv
  inflating: test.csv
  inflating: train.csv
```

```
train_df = pd.read_csv("train.csv")
test_df  = pd.read_csv("test.csv")
```

DATA PREPROCESSING:

The RecordID is saved for the final submission and dropped along with AccessCode from the training set. This was a basic cleaning step to prevent the model from picking up on unique identifiers that don't have any predictive value. By removing these early, I ensured the CatBoost model focused only on relevant features.

I used a regex to pull the Title out of the client name strings and then cleaned them up by grouping synonyms and rare categories. I mapped "Mlle" and "Ms" to "Miss" and grouped low-frequency titles like "Dr" and "Rev" into a "Rare" bucket. This helps CatBoost by reducing the number of categories it has to process, which prevents overfitting on small groups.

To capture the social context, I calculated FamilySize by summing the dependents and then created a binary IsAlone flag. These features give the model a more direct way to understand the passenger's situation without having to infer it from multiple raw columns. It’s a simple way to help the trees find better splits.

For the missing values in the Years column, I used a group-based median imputation. I grouped the data by ServiceTier and Gender to get a more accurate estimate for the missing entries, rather than just using a global average. I made sure to calculate these medians only on the training set to avoid data leakage before applying them to the test set.

Finally, I applied a log transformation to the TransactionValue using np.log1p. Even though CatBoost handles raw numerical data well, log-scaling helps normalize skewed financial data and keeps outliers from having a disproportionate effect on the model's loss function.

```
test_ids = test_df["RecordID"]
drop_cols = ["RecordID", "AccessCode"]

train_df = train_df.drop(columns=drop_cols)
test_df  = test_df.drop(columns=drop_cols)
for df in [train_df, test_df]:
    df["Title"] = df["ClientAlias"].str.extract(
        r",\s*([^\.]+)\.", expand=False
    )
```

```
        df["Title"] = df["Title"].replace(
            ["Mlle", "Ms"], "Miss"
        ).replace(
            ["Mme"], "Mrs"
        )

        rare_titles = [
            "Dr", "Rev", "Major", "Col",
            "Sir", "Lady", "Countess",
            "Jonkheer", "Don"
        ]

        df["Title"] = df["Title"].replace(rare_titles, "Rare")

        df.drop(columns="ClientAlias", inplace=True)
    for df in [train_df, test_df]:
        df["FamilySize"] = (
            df["DependentsOnboard"] +
            df["DependentsAtDestination"] + 1
        )

        df["IsAlone"] = (df["FamilySize"] == 1).astype(int)
    train_df["Years"] = train_df.groupby(
        ["ServiceTier", "Gender"]
    )["Years"].transform(
        lambda x: x.fillna(x.median())
    )

    years_median = train_df.groupby(
        ["ServiceTier", "Gender"]
    )["Years"].median()

    def fill_years(row):
        if pd.isna(row["Years"]):
            return years_median.loc[
                row["ServiceTier"], row["Gender"]
            ]
        return row["Years"]

    test_df["Years"] = test_df.apply(fill_years, axis=1)
    fare_median = train_df["TransactionValue"].median()

    train_df["TransactionValue"] = train_df["TransactionValue"].fillna(fare_median)
    test_df["TransactionValue"]  = test_df["TransactionValue"].fillna(fare_median)
    import numpy as np

    for df in [train_df, test_df]:
        df["LogTransactionValue"] = np.log1p(df["TransactionValue"])
```

MODEL VALIDATION STRATERGY

The training data was prepared by separating the target variable, Outcome, from the feature set. To leverage CatBoost's native handling of non-numeric data, the categorical columns Gender and Title were identified, and their integer indices were stored. This step ensures the algorithm applies its specialized ordered boosting and categorical encoding directly to these features without needing external preprocessing like one-hot encoding.

A StratifiedKFold cross-validation strategy with five splits was implemented to evaluate the model's robustness. This approach ensures that each fold maintains the same percentage of target classes as the original dataset, which is vital for preventing biased performance estimates. I shuffled the data and fixed the random state to 42 to maintain consistency and reproducibility across different experimental runs.

By utilizing this validation framework, the model is trained and validated on different subsets of the data, minimizing the risk of overfitting. This process provides a more stable and generalized accuracy metric, reflecting how the model likely performs against the hidden test set.

```
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import accuracy_score
import numpy as np
X = train_df.drop("Outcome", axis=1)
y = train_df["Outcome"]
```

```
cat_features = ["Gender", "Title"]
cat_idx = [X.columns.get_loc(c) for c in cat_features]

skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
```

MODEL TRAINING:

The training process was executed using a cross-validation loop to ensure that every sample in the training set received an unbiased prediction. Two arrays, oof_probs and test_probs, were initialized to store the probability estimates for the training and test sets, respectively. By capturing these probabilities rather than hard labels, I maintained more granular information about the model's confidence, which is beneficial for final threshold tuning.

For each fold, the data was partitioned into training and validation subsets. I initialized a CatBoostClassifier with specific hyperparameters, including a depth of 7 and a learning rate of 0.05, to balance model complexity with convergence speed. A slight variation in the random_seed for each fold was used to enhance the diversity of the ensemble, while l2_leaf_reg and subsample were set to control for potential overfitting.

The model was fitted using an early_stopping_rounds criteria of 100, which monitored the validation set performance to halt training once the log-loss stopped improving. This prevented the model from memorizing noise in the training data. The categorical features were passed directly via their indices, allowing CatBoost to perform its native transformations during the fitting process.

```
oof_probs = np.zeros(len(X))
test_probs = np.zeros(len(test_df))
models=[]
for fold, (tr_idx, val_idx) in enumerate(skf.split(X, y)):
    X_tr, X_val = X.iloc[tr_idx], X.iloc[val_idx]
    y_tr, y_val = y.iloc[tr_idx], y.iloc[val_idx]

    model = CatBoostClassifier(
        iterations=1500,
        depth=7,
        learning_rate=0.05,
        loss_function="Logloss",
        l2_leaf_reg=3,
        subsample=0.9,
        random_seed=42 + fold,
        verbose=False
    )

    model.fit(
        X_tr, y_tr,
        cat_features=cat_idx,
        eval_set=(X_val, y_val),
        early_stopping_rounds=100
    )

    oof_probs[val_idx] = model.predict_proba(X_val)[:, 1]
    test_probs += model.predict_proba(test_df)[:, 1]
    models.append(model)
test_probs /= skf.n_splits
```

```
import numpy as np
from sklearn.metrics import accuracy_score

thresholds = np.arange(0.45, 0.65, 0.01)
scores = []

for t in thresholds:
    preds = (oof_probs >= t).astype(int)
    scores.append(accuracy_score(y, preds))

best_t = thresholds[np.argmax(scores)]
best_acc = max(scores)

best_t, best_acc
```

```
(np.float64(0.6100000000000001), 0.8270588235294117)
```

```
val_preds = model.predict(X_val)
print("Validation Accuracy:", accuracy_score(y_val, val_preds))
```

```
Validation Accuracy: 0.7823529411764706
```

```
test_probs = np.zeros(len(test_df))
test_X = test_df[X.columns]

for model in models:
    test_probs += model.predict_proba(test_X)[:, 1]

test_probs /= len(models)
FINAL_THRESHOLD = 0.61
final_preds = (test_probs >= FINAL_THRESHOLD).astype(int)
```

```
submission = pd.DataFrame({
    "RecordID": test_ids,
    "Outcome": final_preds
})

submission.to_csv("submission.csv", index=False)
```