Objective

The goal of this assignment is to create a classification model to define the pricing of Airbnbs based on its characteristics. Given an Airbnb dataset containing listings from Montreal, the goal is to classify three different categories for pricing: beginner, plus and premium (0, 1 and 2 respectively).

Evaluated models and Tuning

Three different classification models were evaluated in this assignment: XGBoosting, Logistic regression and Random forest. For hyper-parameter tuning, GridSearchCV() was used to exaustively explore values from the dictionary param_grid.

XGBoosting

The initial hyper-parameters values and best score for XGBoosting were the following:

```
param_grid = {
    'preprocessor__num__imputer__strategy': ['mean'],
    'regressor__n_estimators': [50, 100],
    'regressor__max_depth':[10, 20]
}
best_score_ = 0.69454
```

In attempt to extend explored values, two more parameters were added to param_grid:

```
param_grid = {
    'preprocessor__num__imputer__strategy': ['mean'],
    'regressor__n_estimators': [50, 100],
    'regressor__max_depth':[10, 20],
    'regressor__min_child_weight':[4,5,6],
    'regressor__learning_rate': [0.1, 0.01]
}
best_score_ = 0.70069
```

So adding these parameters slightly increased the best score from 0.69454 to 0.70069.

Some relevant features were added to the training set for evaluation. The idea was that these features could give us valuable insight about the pricing of listings:

```
numeric_features = [
    'availability_30', 'number_of_reviews', 'minimum_nights',
    'review_scores_rating', 'reviews_per_month'
]
categorical_features = [
     'neighbourhood', 'host_is_superhost'
]
```

Which leaves the full list of evaluated features as the following:

```
numeric_features = [
    'bedrooms', 'accommodates', 'beds', 'availability_30',
    'number_of_reviews', 'minimum_nights', 'review_scores_rating',
    'review_scores_location', 'reviews_per_month'
]
categorical_features = [
    'property_type', 'is_business_travel_ready', 'room_type',
    'neighbourhood', 'host_is_superhost'
]
```

Using the new set of features, XGBoosting could then achieve a best score of 0.72572. So adding these feature increased the score from 0.70069 to 0.72572.

Logistic Regression

The initial hyper-parameters values and best score for Logistic Regression were the following:

```
param_grid = {
    'preprocessor__num__imputer__strategy': ['mean'],
    'regressor__max_iter': [1000],
    'regressor__penalty': ['l1','l2'],
    'regressor__C': [1.0, 3.0, 5.0, 7.0]
}
best_score_ = 0.70083
```

Aiming to increase the score, higher values for the C parameter were explored:

```
param_grid = {
    'preprocessor__num__imputer__strategy': ['mean'],
    'regressor__max_iter': [1000],
    'regressor__penalty': ['l1','l2'],
    'regressor__C': [0.001, 0.01, 0.1, 1, 10, 100, 1000]
}
best_score_ = 0.70122
```

So increasing the explored C's slightly changed the best score from 0.70083 to 0.70122.

The same set of relevant features as in XGBoosting was added to train the Logistic regression, which then achieved the best score of 0.72061. So adding these features increased the score from 0.70122 to 0.72061.

Random Forest

The initial hyper-parameters values and best score for Random Forest were the following:

```
    param_grid = {
        'preprocessor__num__imputer__strategy': ['mean'],
        'regressor__n_estimators' : [10, 50, 100, 200],
        'regressor__max_features' : [3, 6, 9, 15]
    }
    best_score_ = 0.69244
```

In attempt to extend explored values, the parameter max_depth was added to param_grid:

```
    param_grid = {
        'preprocessor__num__imputer__strategy': ['mean'],
        'regressor__n_estimators' : [10, 20, 30, 40, 50],
        'regressor__max_features' : [3, 6, 9, 15],
        'regressor__max_depth': [10, 30, 50, 70, 90, 100]
    }
    best_score_ = 0.70476
```

By analyzing the best_estimator_ after performing GridSearchCV() using the previous param_grid, the values of the parameters were max_depth = 10, max_features = 6 and n_estimators = 50. Param_grid was then changed taking into account those values:

```
    param_grid = {
        'preprocessor__num__imputer__strategy': ['mean'],
        'regressor__n_estimators' : [10, 50, 100, 200],
        'regressor__max_features' : [3, 6, 9, 15],
        'regressor__max_depth': [5, 10, 15]
    }
    best_score_ = 0.70449
```

When trying the new parameter values the score actually decreased from 0.70476 to 0.70449.

The same set of relevant features as in XGBoost was added to train the Random Forest, which then achieved the best score of 0.72952. So adding these features increased the score from 0.70449 to 0.72952.

To test whether increasing the number of estimators would increase the model score, the parameter n_estimator in param_grid was changed to 'regressor__n_estimators' : [350, 400, 450, 500]. This only slightly increased the score from 0.72952 to 0.7307.

Questions

- So here we have a public leaderboard and a private leaderboard. Each of them use a different subset of the testing set test.csv (well we can just treat them as two different testing sets). For the public leaderboard, you can try 3 times per day and observe the performance right away. Why should we limit the number of trials per day?

  R. We should limit the number of trials since the training set must always have minimal usage to avoid overfitting.

- For the private leaderboard, it will be used only after the assignment submission deadline for evaluation. Why it is designed like this?

  R. It is designed like this to test the model performance on unseen data.

- Which model did you use (just pick one of them) and how did you control its flexibility? When did you make it more flexible and when did you make it less?

  R. I used Random Forest since it achieved the best performance and it controls well the bias and variance. To control its flexibility, I changed the number of features in the training data and tuned hyper-parameters such as max_features and max_depth.