ASSIGNMENT 2

PREDICTIVE ANALYTICS

Classification and Regression

Ervin Liu (s10036078)

HR ANALYTICS & AIRBNB SINGAPORE

1. Load Dataset
2. Split Dataset into Train/Validate/Test Dataset
3. Fit Models with Cross-validation/GridSearch CV
4. Evaluate and Tune Models
5. Summary and Model Selection

LOAD

Amend and Load Clean Datasets

We load both cleaned datasets with one amendment - categorical variables are now one-hot instead of label encoded. This removes ordering and weighting of the variables due to different values

1.1.1 Load cleansed HR Analytics dataset

```
df_hr = pd.read_csv('hr_data_new.csv', header=0)
df_hr.head(10)
```

	no_of_trainings	age	previous_year_rating	length_of_service	KPIs_met >80%	awards_won?	avg_training_score	is_promoted	Analytics	Finance	...	region_5	r
0	1	0.375	5.0	0.060606	1	0	0.155172	1	0	0	...	0	
1	1	0.325	5.0	0.151515	1	0	0.172414	1	0	0	...	0	
2	1	0.750	4.0	0.484848	1	0	0.103448	1	0	0	...	0	
3	1	0.250	3.0	0.181818	1	0	0.741379	1	1	0	...	0	
4	1	0.350	5.0	0.181818	1	0	0.637931	1	0	0	...	0	

2.1.1 Load cleansed AirBnb Singapore dataset

```
df_airbnb = pd.read_csv('listings_new.csv')
df_airbnb.head(10)
```

	price	minimum_nights	number_of_reviews	reviews_per_month	availability_365	Entire home/apt	Private room	Shared room	Bedok	Bishan	...	Rochor	Serangoon	Singapor Rive
0	44	15	18	0.230000	331	0	1	0	0	0	...	0	0	
1	276	4	11	0.130000	362	1	0	0	0	0	...	0	0	
2	208	1	0	1.043669	0	0	0	1	0	0	...	0	0	
3	128	3	0	1.043669	365	0	1	0	0	0	...	0	0	
4	278	1	0	1.043669	365	0	1	0	1	0	...	0	0	

SPLIT

Split Dataset into Train/Validate/Test Dataset

Both datasets are then split into three parts - Train (60%), Validate (20%) and Test (20%) sets

In [4]:
```
# Split the data into training, validation and testing data
x_train, x_test, y_train, y_test = train_test_split(x_data, y_data, test_size=0.4, random_state=2)
x_val, x_test, y_val, y_test = train_test_split(x_test, y_test, test_size=0.5, random_state=2)
```

In [5]:
```
# Verify that data split is successful
for data in (y_train, y_val, y_test):
    print(round(len(data) / len(y_data), 2))
```

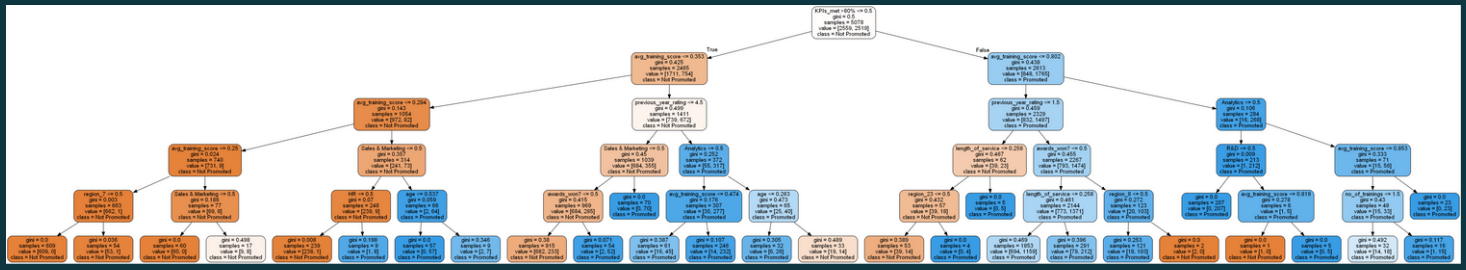
```
0.6
0.2
0.2
```

FIT - CLASSIFICATION

1.3 Use Tree Algorithms to identify features

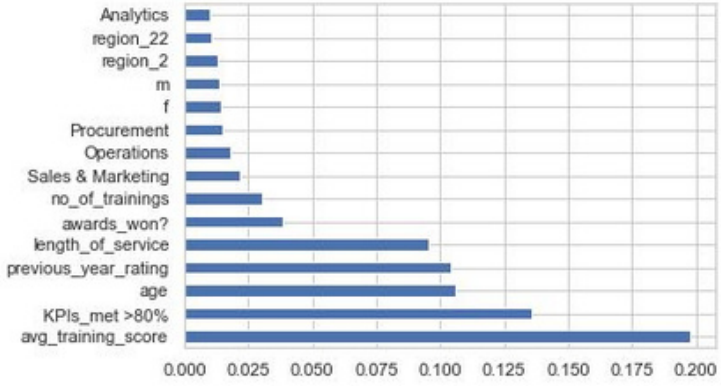
Given the large number of features due to one-hot encoding, there is a tendency for overfitting to occur when training the models. Hence, different algorithms, including Random Forest and Extra Trees, are used to identify high-importance features to streamline the dataset used for modelling

Random Forest is used to identify these top 15 features from the HR Analytics dataset



In [19]:

```
#plot graph of feature importances for better visualization
feat_importances = pd.Series(model.feature_importances_, index=tr_features.columns)
feat_importances.nlargest(15).plot(kind='barh')
plt.show()
```



EVALUATE & TUNE

The models were then re-fitted with the 15 features and evaluated using the validation set for their accuracy

1.4 Random Forest

```
tr_features = tr_features[['age', 'previous_year_rating', 'length_of_service', 'KPIs_met >80%',
                           'awards_won?', 'avg_training_score', 'no_of_trainings',
                           'Sales & Marketing','region_22','region_2','Operations','Analytics', 'Procurement',
                           'f', 'm']]

val_features = val_features[['age', 'previous_year_rating', 'length_of_service', 'KPIs_met >80%',
                           'awards_won?', 'avg_training_score', 'no_of_trainings',
                           'Sales & Marketing','region_22','region_2','Operations','Analytics', 'Procurement',
                           'f', 'm']]

test_features = test_features[['age', 'previous_year_rating', 'length_of_service', 'KPIs_met >80%',
                             'awards_won?', 'avg_training_score', 'no_of_trainings',
                             'Sales & Marketing','region_22','region_2','Operations','Analytics', 'Procurement',
                             'f', 'm']]
```

```
# Reassess the RF R^2 value/Accuracy by using the Score function, following feature selection
cv.fit(tr_features,tr_labels.values.ravel())
print('rf training accuracy is: ',cv.score(tr_features,tr_labels))
print('rf validation accuracy is: ',cv.score(val_features,val_labels))
```

```
rf training accuracy is:  0.962189838519102
rf validation accuracy is:  0.7903130537507383
```

The **min_samples_leaf** hyperparameter - the minimum number of data points allowed per leaf node - was then tuned further before a final assessment using the testing set

```
parameters = {
    'n_estimators': [500],
    'max_depth': [16],
    'min_samples_leaf': [5]

}

cv = GridSearchCV(rf, parameters, cv=5)
cv.fit(tr_features,tr_labels.values.ravel())

print('rf training accuracy is: ',cv.score(tr_features,tr_labels))
print('rf testing accuracy is: ',cv.score(test_features,test_labels))
```

```
rf training accuracy is:  0.8538794801102796
rf testing accuracy is:  0.8003544004725339
```

EVALUATE & TUNE

1.4 Support Vector Machine (SVM)

```
print('SVM training accuracy is: ',cv.score(tr_features,tr_labels))
print('SVM validation accuracy is: ',cv.score(val_features,val_labels))
```

```
SVM training accuracy is:  0.8194170933438362
SVM validation accuracy is:  0.7914943886591849
```

The **gamma** hyperparameter was then tuned further before a final assessment using the testing set. This controls the distance of influence of a single training point. High gamma values mean data points need to be very close to each other to be considered in the same group and hence often leads to overfitting.

```
parameters = {
    'kernel': ['rbf'],
    'C': [100],
    'gamma': [0.1]

}

cv = GridSearchCV(svc, parameters, cv=5)
cv.fit(tr_features,tr_labels.values.ravel())

print('SVM training accuracy is: ',cv.score(tr_features,tr_labels))
print('SVM testing accuracy is: ',cv.score(test_features,test_labels))
```

```
SVM training accuracy is:  0.82926348956282
SVM testing accuracy is:  0.8003544004725339
```

EVALUATE & TUNE

1.4 XGBoost Classifier

```
print('XGBoostClassifier training accuracy is: ',cv.score(tr_features,tr_labels))
print('XGBoostClassifier validation accuracy is: ',cv.score(val_features,val_labels))
```

```
XGBoostClassifier training accuracy is:  0.8190232374950768
XGBoostClassifier validation accuracy is:  0.8080330773774365
```

The **learning rate** (or shrinkage**)** hyperparameter was then tuned further before a final assessment using the testing set. This is a technique to slow down the learning in the model by applying a weighting factor for new CARTs when they are added to the model

```
parameters = {
    'max_depth': [5],
    'alpha': [5],
    'n_estimators': [50],
    'learning_rate': [0.17]

}

cv = GridSearchCV(xg_class, parameters, cv=5)
cv.fit(tr_features,tr_labels.values.ravel())

print('XGBoostClassifier training accuracy is: ',cv.score(tr_features,tr_labels))
print('XGBoostClassifier testing accuracy is: ',cv.score(test_features,test_labels))
```

```
XGBoostClassifier training accuracy is:  0.8355651831429697
XGBoostClassifier testing accuracy is:  0.8168930891907856
```

EVALUATE & TUNE

1.4 Ensemble Model

Finally, all models were combined into a single ensemble model to see if this can yield a more robust predictive model with a higher accuracy than any one single model.

1.3.1 Multiple Ensemble Model using Random Forest, XGBoost Classifier and Support Vector Machine for HR Analytics dataset

In [213]:
```
# Voting Classifier - Multiple Ensemble Model
rf = RandomForestClassifier(n_estimators=500, max_depth = 16, min_samples_leaf=5)
xg_class = XGBClassifier(max_depth=5, alpha=5, n_estimators=50, learning_rate=0.17)
svm = SVC(C=100, kernel='rbf', gamma=0.1)
```

In [214]:
```
evc = VotingClassifier(estimators = [('rf',rf),('xg_class',xg_class),('svm',svm)], voting='hard')
```

In [216]:
```
evc.fit(tr_features, tr_labels.values.ravel())
print('EVC training accuracy is', evc.score(tr_features, tr_labels.values.ravel()))
print('EVC testing accuracy is', evc.score(test_features, test_labels.values.ravel()))
```

```
EVC training accuracy is 0.840882237101221
EVC testing accuracy is 0.8121677495569994
```

While its accuracy is higher than the Random Forest or SVM model, it still falls short of the XGBoost Classifier model's accuracy

SUMMARY

1. The **XGBoost Classifier model** is the most accurate of the regression models with a testing accuracy of approximately **0.8169**

2. The Ensemble Variable Classifier (EVC) model failed to yield the highest testing accuracy

3. Using feature importance analysis, the dataset was reduced to 15 variables, which helped to significantly reduce overfitting

4. Model accuracy might be improved with additional details like the types of awards won

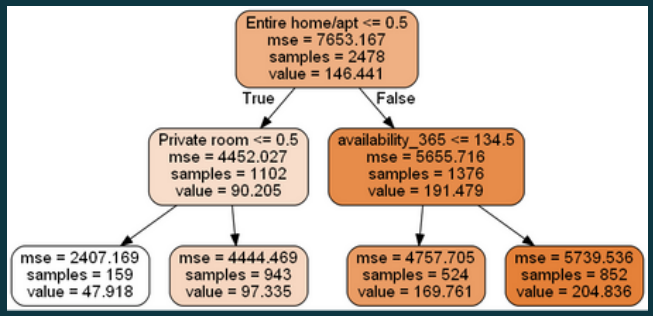
5. Model accuracy might also be improved with more data records

FIT - REGRESSION

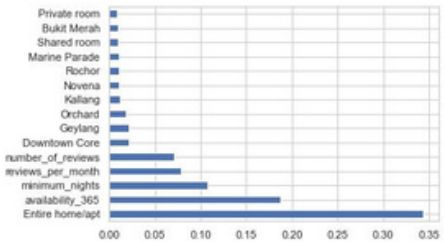
2.3 Use Tree Algorithms to identify features

Again, given the large number of features due to one-hot encoding, there is a tendency for overfitting to occur when training the models. Hence, we need to identify high-importance features to streamline and reduce dataset complexity

Random Forest was used to identify these top 10 features from the Airbnb dataset




```
#plot graph of feature importances for better visualization
feat_importances = pd.Series(model.feature_importances_, index=tr_features_airbnb.columns)
feat_importances.nlargest(15).plot(kind='barh')
plt.show()
```



Based on an ensemble of 500 independent decision trees, the ten features that achieved the highest votes for their importance are:

1. Entire home/apt
2. availability_365
3. minimum_nights
4. reviews_per_month
5. number_of_reviews
6. Downtown Core
7. Geylang
8. Orchard
9. Rochor
10. Kallang

These insights could be later used to build a more refined model, with reduced dimensionality and complexity, using only these ten variables to determine if it improves model accuracy.

EVALUATE & TUNE

As done previously, the models were then re-fitted with the 10 highest voted features identified through Random Forest and evaluated using the validation set for their R2 score (accuracy), Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE)

2.4 Random Forest

```
tr_features_airbnb = tr_features_airbnb[['Entire home/apt','availability_365','minimum_nights','reviews_per_month',
                                         'number_of_reviews','Downtown Core','Geylang','Orchard','Rochor','Kallang',
                                         'Shared room','Novena','Private room', 'Marine Parade', 'Outram']]

val_features_airbnb = val_features_airbnb[['Entire home/apt','availability_365','minimum_nights','reviews_per_month',
                                         'number_of_reviews','Downtown Core','Geylang','Orchard','Rochor','Kallang',
                                         'Shared room','Novena','Private room', 'Marine Parade', 'Outram']]

test_features_airbnb = test_features_airbnb[['Entire home/apt','availability_365','minimum_nights','reviews_per_month',
                                         'number_of_reviews','Downtown Core','Geylang','Orchard','Rochor','Kallang',
                                         'Shared room','Novena','Private room', 'Marine Parade', 'Outram']]
```

```
# Reassess the RF R^2 value/Accuracy by using the Score function, following feature selection
cv.fit(tr_features_airbnb,tr_labels_airbnb.values.ravel())
print('rf training accuracy is: ',cv.score(tr_features_airbnb,tr_labels_airbnb))
print('rf validation accuracy is: ',cv.score(val_features_airbnb,val_labels_airbnb))
```

```
rf training accuracy is:  0.6530302198244499
rf validation accuracy is:  0.5204216737137182
```

Compared to the initial run of the Random Forest model, the overfitting has been significantly reduced, while testing accuracy has only dropped by 0.05 compared to the previously overfitted model.

EVALUATE & TUNE

2.4 Random Forest

Again, the **min_samples_leaf** hyperparameter - the minimum number of data points allowed per leaf node - was then tuned further before a final assessment using the testing set

```
parameters = {
    'n_estimators': [500],
    'max_depth': [16],
    'min_samples_leaf': [7]

}

cv = GridSearchCV(rf, parameters, cv=5)
cv.fit(tr_features_airbnb,tr_labels_airbnb.values.ravel())

print('rf training accuracy is: ',cv.score(tr_features_airbnb,tr_labels_airbnb))
print('rf testing accuracy is: ',cv.score(test_features_airbnb,test_labels_airbnb))
```

```
rf training accuracy is:  0.6676214373172117
rf testing accuracy is:  0.5450496016403836
```

Tuning the min_samples_leaf hyperparameter of the Random Forest model helped boost its testing accuracy

```
print('rf training MAE is: ', mean_absolute_error(cv.predict(tr_features_airbnb), tr_labels_airbnb))
print('rf testing MAE is: ', mean_absolute_error(cv.predict(test_features_airbnb), test_labels_airbnb))
```

```
rf training MAE is:  36.15975024268349
rf testing MAE is:  42.58737282551633
```

```
print('rf training RMSE is: ', np.sqrt(mean_squared_error(cv.predict(tr_features_airbnb), tr_labels_airbnb)))
print('rf testing RMSE is: ', np.sqrt(mean_squared_error(cv.predict(test_features_airbnb), test_labels_airbnb)))
```

```
rf training RMSE is:  50.43559052105382
rf testing RMSE is:  58.79153774954758
```

EVALUATE & TUNE

2.4 Multilayer Perceptron (MLP)

```
from sklearn.neural_network import MLPRegressor

MLP = MLPRegressor(max_iter = 6000)
parameters = {
    'hidden_layer_sizes': [(5,), (10,), (50,), (100,)],
    'activation': ['relu', 'tanh', 'identity'],
    'learning_rate': ['constant','invscaling', 'adaptive']
}

cv = GridSearchCV(MLP, parameters, cv=5)
cv.fit(tr_features_airbnb,tr_labels_airbnb.values.ravel())

print_results(cv)
```

```
0.43 (+/-0.062) for {'activation': 'relu', 'hidden_layer_sizes': (10,), 'learning_rate': 'constant'}
0.425 (+/-0.062) for {'activation': 'relu', 'hidden_layer_sizes': (10,), 'learning_rate': 'invscaling'}
0.422 (+/-0.056) for {'activation': 'relu', 'hidden_layer_sizes': (10,), 'learning_rate': 'adaptive'}
0.426 (+/-0.076) for {'activation': 'relu', 'hidden_layer_sizes': (50,), 'learning_rate': 'constant'}
0.426 (+/-0.067) for {'activation': 'relu', 'hidden_layer_sizes': (50,), 'learning_rate': 'invscaling'}
0.424 (+/-0.057) for {'activation': 'relu', 'hidden_layer_sizes': (50,), 'learning_rate': 'adaptive'}
0.426 (+/-0.07) for {'activation': 'relu', 'hidden_layer_sizes': (100,), 'learning_rate': 'constant'}
0.435 (+/-0.063) for {'activation': 'relu', 'hidden_layer_sizes': (100,), 'learning_rate': 'invscaling'}
0.426 (+/-0.055) for {'activation': 'relu', 'hidden_layer_sizes': (100,), 'learning_rate': 'adaptive'}
0.403 (+/-0.031) for {'activation': 'tanh', 'hidden_layer_sizes': (5,), 'learning_rate': 'constant'}
0.413 (+/-0.031) for {'activation': 'tanh', 'hidden_layer_sizes': (5,), 'learning_rate': 'invscaling'}
0.4 (+/-0.042) for {'activation': 'tanh', 'hidden_layer_sizes': (5,), 'learning_rate': 'adaptive'}
0.425 (+/-0.038) for {'activation': 'tanh', 'hidden_layer_sizes': (10,), 'learning_rate': 'constant'}
0.417 (+/-0.031) for {'activation': 'tanh', 'hidden_layer_sizes': (10,), 'learning_rate': 'invscaling'}
0.42 (+/-0.03) for {'activation': 'tanh', 'hidden_layer_sizes': (10,), 'learning_rate': 'adaptive'}
0.45 (+/-0.038) for {'activation': 'tanh', 'hidden_layer_sizes': (50,), 'learning_rate': 'constant'}
0.443 (+/-0.031) for {'activation': 'tanh', 'hidden_layer_sizes': (50,), 'learning_rate': 'invscaling'}
0.438 (+/-0.043) for {'activation': 'tanh', 'hidden_layer_sizes': (50,), 'learning_rate': 'adaptive'}
0.452 (+/-0.052) for {'activation': 'tanh', 'hidden_layer_sizes': (100,), 'learning_rate': 'constant'}
0.453 (+/-0.032) for {'activation': 'tanh', 'hidden_layer_sizes': (100,), 'learning_rate': 'invscaling'}
```

```
print('MLP training accuracy is: ',cv.score(tr_features_airbnb,tr_labels_airbnb))
print('MLP validation accuracy is: ',cv.score(val_features_airbnb,val_labels_airbnb))
```

```
MLP training accuracy is:  0.5044537750538767
MLP validation accuracy is:  0.47108824335900923
```

EVALUATE & TUNE

2.4 Multilayer Perceptron (MLP)

The MLP's **alpha** hyperparameter was tuned before being evaluated on the final testing set. Alpha is a penalty term, that reduces overfitting by constraining the size of the weights. Increasing alpha improves data with high variance while decreasing it improves data with high bias.

```
parameters = {
    'hidden_layer_sizes': [(100,)],
    'activation': ['tanh'],
    'learning_rate': ['invscaling'],
    'alpha': [0.008]
}

cv = GridSearchCV(MLP, parameters, cv=5)
cv.fit(tr_features_airbnb,tr_labels_airbnb.values.ravel())

print('MLP training accuracy is: ',cv.score(tr_features_airbnb,tr_labels_airbnb))
print('MLP testing accuracy is: ',cv.score(test_features_airbnb,test_labels_airbnb))
```

```
MLP training accuracy is:  0.5055990619710139
MLP testing accuracy is:  0.4810923410121989
```

Tuning the alpha hyperparameter marginally improves the model accuracy but this model still remains the poores performer compared to the first two models

```
print('MLP training MAE is: ', mean_absolute_error(cv.predict(tr_features_airbnb), tr_labels_airbnb))
print('MLP testing MAE is: ', mean_absolute_error(cv.predict(test_features_airbnb), test_labels_airbnb))
```

```
MLP training MAE is:  45.72177010298978
MLP testing MAE is:  46.446853442606844
```

```
print('MLP training RMSE is: ', np.sqrt(mean_squared_error(cv.predict(tr_features_airbnb), tr_labels_airbnb)))
print('MLP testing RMSE is: ', np.sqrt(mean_squared_error(cv.predict(test_features_airbnb), test_labels_airbnb)))
```

```
MLP training RMSE is:  61.5120570096149
MLP testing RMSE is:  62.7881714277331
```

EVALUATE & TUNE

2.4 XGBoost Regressor

```python
# Using 5 K-folds cross-validation, we will use GridSearchCV to tune the max_depth and alpha parameters for the XGBoost m
from xgboost import XGBRegressor
xg_regress = XGBRegressor(default = 'reg:squarederror')
parameters = {
    'max_depth': [1, 3, 4, 5],
    'alpha': [5, 10, 15, 20],
    'n_estimators': [5, 10, 50, 100]
}

cv = GridSearchCV(xg_regress, parameters, cv=5)
cv.fit(tr_features_airbnb,tr_labels_airbnb.values.ravel())

print_results(cv)
```

```
[12:06:58] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[12:06:58] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[12:06:58] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[12:06:58] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[12:06:58] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[12:06:58] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[12:06:58] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[12:06:58] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[12:06:58] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[12:06:58] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[12:06:58] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[12:06:58] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[12:06:58] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[12:06:58] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[12:06:58] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[12:06:58] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[12:06:58] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[12:06:58] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[12:06:58] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
```

```python
print('XGBR training accuracy is: ',cv.score(tr_features_airbnb,tr_labels_airbnb))
print('XGBR validation accuracy is: ',cv.score(val_features_airbnb,val_labels_airbnb))
```

```
XGBR training accuracy is:  0.6965423930576431
XGBR validation accuracy is:  0.5047548642456285
```

EVALUATE & TUNE

2.4 XGBoost Regressor

Similar to the earlier case, the **learning rate** (or shrinkage) hyperparameter was then tuned further before a final assessment using the testing set. This is a technique to slow down the learning in the model by applying a weighting factor for new CARTs when they are added to the model

```
parameters = {
    'max_depth': [5],
    'alpha': [5],
    'n_estimators': [50],
    'learning_rate': [0.15]

}

cv = GridSearchCV(xg_regress, parameters, cv=5)
cv.fit(tr_features_airbnb,tr_labels_airbnb.values.ravel())

print('XGBoost Regressor (XGBR) training accuracy is: ',cv.score(tr_features_airbnb,tr_labels_airbnb))
print('XGBoost Regressor (XGBR) testing accuracy is: ',cv.score(test_features_airbnb,test_labels_airbnb))
```

```
[21:03:27] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[21:03:27] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[21:03:27] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[21:03:27] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[21:03:27] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[21:03:27] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
XGBoost Regressor (XGBR) training accuracy is:  0.6684989644626869
XGBoost Regressor (XGBR) testing accuracy is:  0.5309683321941236
```

Setting the learning_rate hyperparameter to 0.15 boosted accuracy while reducing overfitting.

```
print('XGBR training MAE is: ', mean_absolute_error(cv.predict(tr_features_airbnb), tr_labels_airbnb))
print('XGBR testing MAE is: ', mean_absolute_error(cv.predict(test_features_airbnb), test_labels_airbnb))
```

```
XGBR training MAE is:  36.83976841511699
XGBR testing MAE is:  43.95164989416083
```

```
print('XGBR training RMSE is: ', np.sqrt(mean_squared_error(cv.predict(tr_features_airbnb), tr_labels_airbnb)))
print('XGBR testing RMSE is: ', np.sqrt(mean_squared_error(cv.predict(test_features_airbnb), test_labels_airbnb)))
```

```
XGBR training RMSE is:  50.36896791712601
XGBR testing RMSE is:  59.69443939433386
```

EVALUATE & TUNE

2.4 Ensemble Model

Finally, all models were again combined into a single ensemble model to see if this can yield a more robust predictive model with a higher accuracy and lower MAE and RMSE than any one single model.

```
# Voting Classifier - Multiple Ensemble Model
from sklearn.ensemble import VotingRegressor

rf = RandomForestRegressor(n_estimators=500, max_depth = 16, min_samples_leaf = 7)
xg_regress = XGBRegressor(default = 'reg:squarederror', max_depth = 5, alpha = 5, n_estimators = 50, learning_rate = 0.15
MLP = MLPRegressor(max_iter = 6000, activation = 'tanh', hidden_layer_sizes = (100,), learning_rate = 'invscaling', alpha
```

```
evr = VotingRegressor([('rf',rf),('xg_regress',xg_regress),('MLP',MLP)])
```

```
evr.fit(tr_features_airbnb, tr_labels_airbnb.values.ravel())
print('EVR training accuracy is', evr.score(tr_features_airbnb, tr_labels_airbnb.values.ravel()))
print('EVR testing accuracy is', evr.score(val_features_airbnb, val_labels_airbnb.values.ravel()))
```

```
[21:13:20] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
EVR training accuracy is 0.6312969930817837
EVR testing accuracy is 0.5273523890979295
```

```
print('EVR training MAE is: ', mean_absolute_error(cv.predict(tr_features_airbnb), tr_labels_airbnb))
print('EVR testing MAE is: ', mean_absolute_error(cv.predict(test_features_airbnb), test_labels_airbnb))
```

```
EVR training MAE is:  45.72177010298978
EVR testing MAE is:  46.446853442606844
```

```
print('EVR training RMSE is: ', np.sqrt(mean_squared_error(cv.predict(tr_features_airbnb), tr_labels_airbnb)))
print('EVR testing RMSE is: ', np.sqrt(mean_squared_error(cv.predict(test_features_airbnb), test_labels_airbnb)))
```

```
EVR training RMSE is:  61.5120570096149
EVR testing RMSE is:  62.7881714277331
```

It was found that while the Ensemble model had a higher accuracy and lower MAE and RMSE than the XGBoost Regressor and MLP models, the Random Forest model produced the best results.

SUMMARY

1. The **Random Forest Regressor** model had the R^2 value at **0.545** and lowest MAE and RMSEs

2. Overall, all models posted low testing accuracies, most likely due to the very limited dataset size (>40% of data was removed during data cleanup) and messy data.

3. Model accuracy may be increased by using a larger dataset and focusing the data on one or two specific data categories such as room type or location.

4. Obtaining more features - like ratings (on a 5-point scale) or whether a host has SuperHost status (an Airbnb badge to distinguish hosts with exceptional service standards) - might also be another way to improve model accuracy.