Assignment - Decision Trees and Random Forests

In this assignment, you'll continue building on the previous assignment to predict the price of a house using information like its location, area, no. of rooms etc. You'll use the dataset from the House Prices - Advanced Regression Techniques competition on Kaggle.

We'll follow a step-by-step process:

1. Download and prepare the dataset for training
2. Train, evaluate and interpret a decision tree
3. Train, evaluate and interpret a random forest
4. Tune hyperparameters to improve the model
5. Make predictions and save the model

As you go through this notebook, you will find a **???** in certain places. Your job is to replace the **???** with appropriate code or values, to ensure that the notebook runs properly end-to-end and your machine learning model is trained properly without errors.

Guidelines

1. Make sure to run all the code cells in order. Otherwise, you may get errors like `NameError` for undefined variables.
2. Do not change variable names, delete cells, or disturb other existing code. It may cause problems during evaluation.
3. In some cases, you may need to add some code cells or new statements before or after the line of code containing the **???**.
4. Since you'll be using a temporary online service for code execution, save your work by running `jovian.commit` at regular intervals.
5. Review the "Evaluation Criteria" for the assignment carefully and make sure your submission meets all the criteria.
6. Questions marked **(Optional)** will not be considered for evaluation and can be skipped. They are for your learning.
7. It's okay to ask for help & discuss ideas on the community forum, but please don't post full working code, to give everyone an opportunity to solve the assignment on their own.

Important Links:

- Make a submission here: https://jovian.ai/learn/machine-learning-with-python-zero-to-gbms/assignment/assignment-2-decision-trees-and-random-forests

- Ask questions, discuss ideas and get help here: https://jovian.ai/forum/c/zero-to-gbms/gbms-assignment-2/99
- Review this Jupyter notebook: https://jovian.ai/aakashns/sklearn-decision-trees-random-forests

How to Run the Code and Save Your Work

Option 1: Running using free online resources (1-click, recommended): The easiest way to start executing the code is to click the **Run** button at the top of this page and select **Run on Binder**. This will set up a cloud-based Jupyter notebook server and allow you to modify/execute the code.

Option 2: Running on your computer locally: To run the code on your computer locally, you'll need to set up Python, download the notebook and install the required libraries. Click the **Run** button at the top of this page, select the **Run Locally** option, and follow the instructions.

Saving your work: You can save a snapshot of the assignment to your Jovian profile, so that you can access it later and continue your work. Keep saving your work by running `jovian.commit` from time to time.

```
!pip install jovian --upgrade --quiet
```

```
import jovian
```

```
jovian.commit(project='python-random-forests-assignment', privacy='secret')
```

```
[jovian] Updating notebook "ahmedatif655/python-random-forests-assignment" on https://jovian.ai
[jovian] Committed successfully! https://jovian.ai/ahmedatif655/python-random-forests-assignment
```

```
'https://jovian.ai/ahmedatif655/python-random-forests-assignment'
```

Let's begin by installing the required libraries.

```
!pip install opendatasets scikit-learn plotly folium --upgrade --quiet
```

```
!pip install pandas numpy matplotlib seaborn --quiet
```

Download and prepare the dataset for training

```
import os
from zipfile import ZipFile
from urllib.request import urlretrieve

dataset_url = 'https://github.com/JovianML/opendatasets/raw/master/data/house-prices-ac
urlretrieve(dataset_url, 'house-prices.zip')
with ZipFile('house-prices.zip') as f:
    f.extractall(path='house-prices')
```

```
os.listdir('house-prices')
```

```
['data_description.txt', 'sample_submission.csv', 'test.csv', 'train.csv']
```

```
import pandas as pd
pd.options.display.max_columns = 200
pd.options.display.max_rows = 200

prices_df = pd.read_csv('house-prices/train.csv')
prices_df
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfi
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	Insid
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	FR
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	Insid
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	Corne
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	FR
...	...	...	...	...	...	...	...	...	...	...	.
1455	1456	60	RL	62.0	7917	Pave	NaN	Reg	Lvl	AllPub	Insid
1456	1457	20	RL	85.0	13175	Pave	NaN	Reg	Lvl	AllPub	Insid
1457	1458	70	RL	66.0	9042	Pave	NaN	Reg	Lvl	AllPub	Insid
1458	1459	20	RL	68.0	9717	Pave	NaN	Reg	Lvl	AllPub	Insid
1459	1460	20	RL	75.0	9937	Pave	NaN	Reg	Lvl	AllPub	Insid

1460 rows × 81 columns

```
import numpy as np
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import MinMaxScaler, OneHotEncoder
from sklearn.model_selection import train_test_split

# Identify input and target columns
input_cols, target_col = prices_df.columns[1:-1], prices_df.columns[-1]
inputs_df, targets = prices_df[input_cols].copy(), prices_df[target_col].copy()

# Identify numeric and categorical columns
numeric_cols = prices_df[input_cols].select_dtypes(include=np.number).columns.tolist()
categorical_cols = prices_df[input_cols].select_dtypes(include='object').columns.tolist

# Impute and scale numeric columns
imputer = SimpleImputer().fit(inputs_df[numeric_cols])
inputs_df[numeric_cols] = imputer.transform(inputs_df[numeric_cols])
scaler = MinMaxScaler().fit(inputs_df[numeric_cols])
inputs_df[numeric_cols] = scaler.transform(inputs_df[numeric_cols])

# One-hot encode categorical columns
encoder = OneHotEncoder(sparse=False, handle_unknown='ignore').fit(inputs_df[categorica
encoded_cols = list(encoder.get_feature_names(categorical_cols))
inputs_df[encoded_cols] = encoder.transform(inputs_df[categorical_cols])

# Create training and validation sets
train_inputs, val_inputs, train_targets, val_targets = train_test_split(
    inputs_df[numeric_cols + encoded_cols], targets, test_size=0.25, random_state=42)
```

```
/opt/conda/lib/python3.9/site-packages/sklearn/utils/deprecation.py:87: FutureWarning:
Function get_feature_names is deprecated; get_feature_names is deprecated in 1.0 and
will be removed in 1.2. Please use get_feature_names_out instead.
  warnings.warn(msg, category=FutureWarning)
/opt/conda/lib/python3.9/site-packages/pandas/core/frame.py:3678: PerformanceWarning:
DataFrame is highly fragmented.  This is usually the result of calling `frame.insert`
many times, which has poor performance.  Consider joining all columns at once using
pd.concat(axis=1) instead.  To get a de-fragmented frame, use `newframe = frame.copy()`
  self[col] = igetitem(value, i)
```

Decision Tree

> **QUESTION 1**: Train a decision tree regressor using the training set.

```
from sklearn.tree import DecisionTreeRegressor
```

```
# Create the model
tree = DecisionTreeRegressor(random_state=42)
```

```
# Fit the model to the training data
tree.fit(train_inputs, train_targets)
```

```
DecisionTreeRegressor(random_state=42)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook. On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

DecisionTreeRegressor

```
DecisionTreeRegressor(random_state=42)
```

> **QUESTION 2**: Generate predictions on the training and validation sets using the trained decision tree, and compute the RMSE loss.

```
from sklearn.metrics import mean_squared_error
```

```
tree_train_preds = tree.predict(train_inputs)
```

```
tree_train_rmse = mean_squared_error(train_targets, tree_train_preds,squared=False)
```

```
tree_val_preds = tree.predict(val_inputs)
```

```
tree_val_rmse = mean_squared_error(val_targets, tree_val_preds,squared=False)
```

```
print('Train RMSE: {}, Validation RMSE: {}'.format(tree_train_rmse, tree_val_rmse))
```

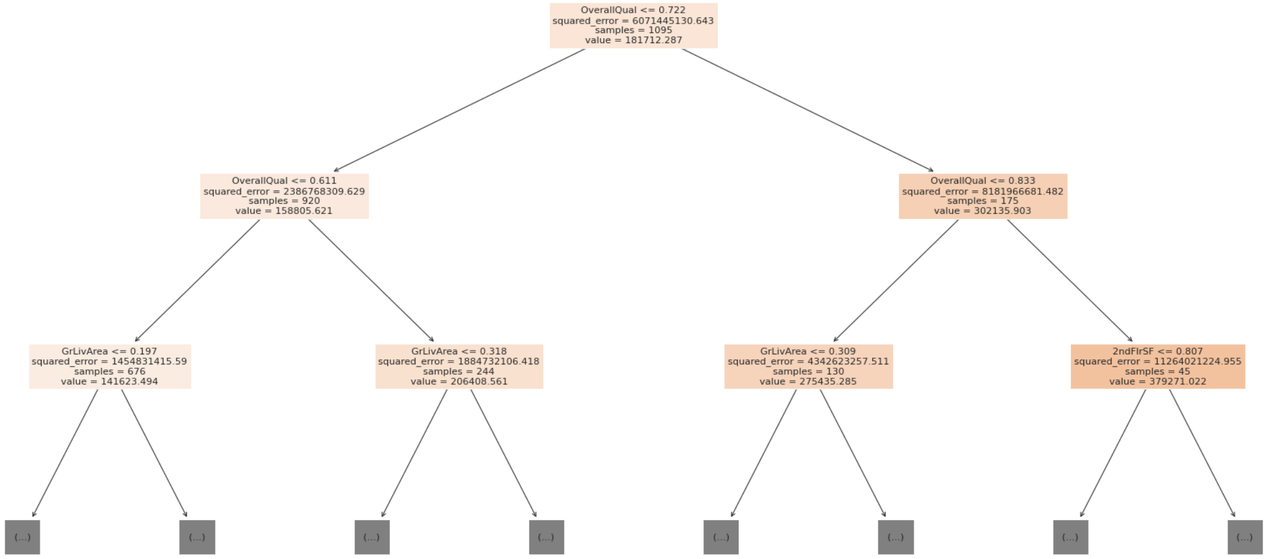
```
Train RMSE: 0.0, Validation RMSE: 37802.872039112044
```

> **QUESTION 3**: Visualize the decision tree (graphically and textually) and display feature importances as a graph. Limit the maximum depth of graphical visualization to 3 levels.

```
import matplotlib.pyplot as plt
from sklearn.tree import plot_tree, export_text
import seaborn as sns
sns.set_style('darkgrid')
%matplotlib inline
```

```
plt.figure(figsize=(30,15))

# Visualize the tree graphically using plot_tree
plot_tree(tree, feature_names=train_inputs.columns, max_depth=2, filled=True);
```



```python
# Visualize the tree textually using export_text
tree_text = export_text(tree, max_depth=10, feature_names=list(train_inputs.columns))
```

```python
# Display the first few lines
print(tree_text[:2000])
```

```
|--- OverallQual <= 0.72
|   |--- OverallQual <= 0.61
|   |   |--- GrLivArea <= 0.20
|   |   |   |--- TotalBsmtSF <= 0.16
|   |   |   |   |--- OverallQual <= 0.39
|   |   |   |   |   |--- GarageCond_TA <= 0.50
|   |   |   |   |   |   |--- LotArea <= 0.04
|   |   |   |   |   |   |   |--- CentralAir_Y <= 0.50
|   |   |   |   |   |   |   |   |--- YearBuilt <= 0.54
|   |   |   |   |   |   |   |   |   |--- SaleCondition_Abnorml <= 0.50
|   |   |   |   |   |   |   |   |   |   |--- Neighborhood_BrkSide <= 0.50
|   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of depth 3
|   |   |   |   |   |   |   |   |   |   |--- Neighborhood_BrkSide >  0.50
|   |   |   |   |   |   |   |   |   |   |   |--- value: [39300.00]
|   |   |   |   |   |   |   |   |   |--- SaleCondition_Abnorml >  0.50
|   |   |   |   |   |   |   |   |   |   |--- 2ndFlrSF <= 0.09
|   |   |   |   |   |   |   |   |   |   |   |--- value: [34900.00]
|   |   |   |   |   |   |   |   |   |   |--- 2ndFlrSF >  0.09
|   |   |   |   |   |   |   |   |   |   |   |--- value: [37900.00]
|   |   |   |   |   |   |   |   |--- YearBuilt >  0.54
|   |   |   |   |   |   |   |   |   |--- Condition1_Norm <= 0.50
|   |   |   |   |   |   |   |   |   |   |--- value: [72500.00]
|   |   |   |   |   |   |   |   |   |--- Condition1_Norm >  0.50
|   |   |   |   |   |   |   |   |   |   |--- YrSold <= 0.25
|   |   |   |   |   |   |   |   |   |   |   |--- value: [80500.00]
|   |   |   |   |   |   |   |   |   |   |--- YrSold >  0.25
|   |   |   |   |   |   |   |   |   |   |   |--- value: [82000.00]
|   |   |   |   |   |   |   |--- CentralAir_Y >  0.50
|   |   |   |   |   |   |   |   |--- Functional_Maj2 <= 0.50
|   |   |   |   |   |   |   |   |   |--- Fireplaces <= 0.17
|   |   |   |   |   |   |   |   |   |   |--- LotArea <= 0.03
|   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of depth 4
|   |   |   |   |   |   |   |   |   |   |--- LotArea >  0.03
|   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of
```

```python
# Check feature importance
tree_importances = tree.feature_importances_
```

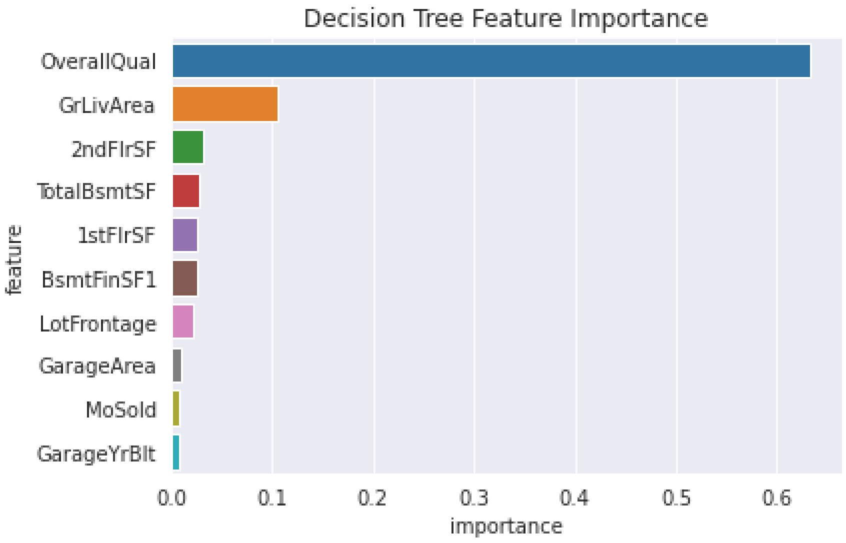
```
tree_importance_df = pd.DataFrame({
    'feature': train_inputs.columns,
    'importance': tree_importances
}).sort_values('importance', ascending=False)
```

```
tree_importance_df
```

	feature	importance
3	OverallQual	0.632537
15	GrLivArea	0.104334
13	2ndFlrSF	0.031896
11	TotalBsmtSF	0.028504
12	1stFlrSF	0.026730
...	...	...
104	Condition2_RRAn	0.000000
103	Condition2_RRAe	0.000000
102	Condition2_PosN	0.000000
212	BsmtFinType2_nan	0.000000
152	Exterior2nd_CBlock	0.000000

304 rows × 2 columns

```
plt.title('Decision Tree Feature Importance')
sns.barplot(data=tree_importance_df.head(10), x='importance', y='feature');
```



Random Forests

> **QUESTION 4**: Train a random forest regressor using the training set.

```
from sklearn.ensemble import RandomForestRegressor
```

```
# Create the model
rf1 = RandomForestRegressor(n_jobs=-1, random_state=42)
```

```
# Fit the model
rf1.fit(train_inputs, train_targets)
```

```
RandomForestRegressor(n_jobs=-1, random_state=42)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook. On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

RandomForestRegressor

```
RandomForestRegressor(n_jobs=-1, random_state=42)
```

> **QUESTION 5**: Make predictions using the random forest regressor.

```
rf1_train_preds = rf1.predict(train_inputs)
```

```
rf1_train_rmse = mean_squared_error(train_targets, rf1_train_preds,squared=False)
```

```
rf1_val_preds = rf1.predict(val_inputs)
```

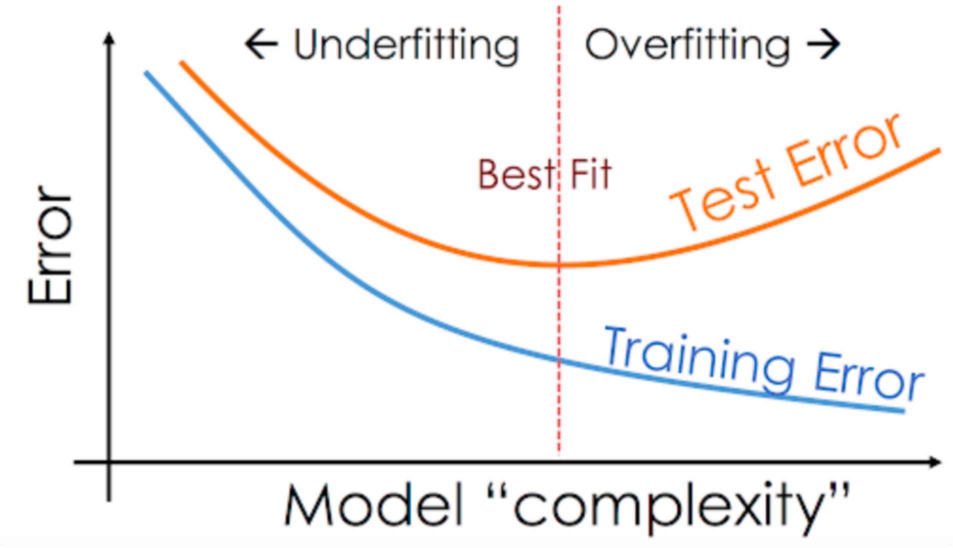
```
rf1_val_rmse = mean_squared_error(val_targets, rf1_val_preds,squared=False)
```

```
print('Train RMSE: {}, Validation RMSE: {}'.format(rf1_train_rmse, rf1_val_rmse))
```

```
Train RMSE: 11864.318299877767, Validation RMSE: 27830.03863639856
```

Hyperparameter Tuning

Let us now tune the hyperparameters of our model. You can find the hyperparameters for `RandomForestRegressor` here: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html



Hyperparameters are use

Let's define a helper function `test_params` which can test the given value of one or more hyperparameters.

```
def test_params(**params):
    model = RandomForestRegressor(random_state=42, n_jobs=-1, **params).fit(train_input
    train_rmse = mean_squared_error(model.predict(train_inputs), train_targets, squarec
    val_rmse = mean_squared_error(model.predict(val_inputs), val_targets, squared=False
    return train_rmse, val_rmse
```

It can be used as follows:

```
test_params(n_estimators=20, max_depth=20)
```

```
(13776.89957127333, 28886.033523273858)
```

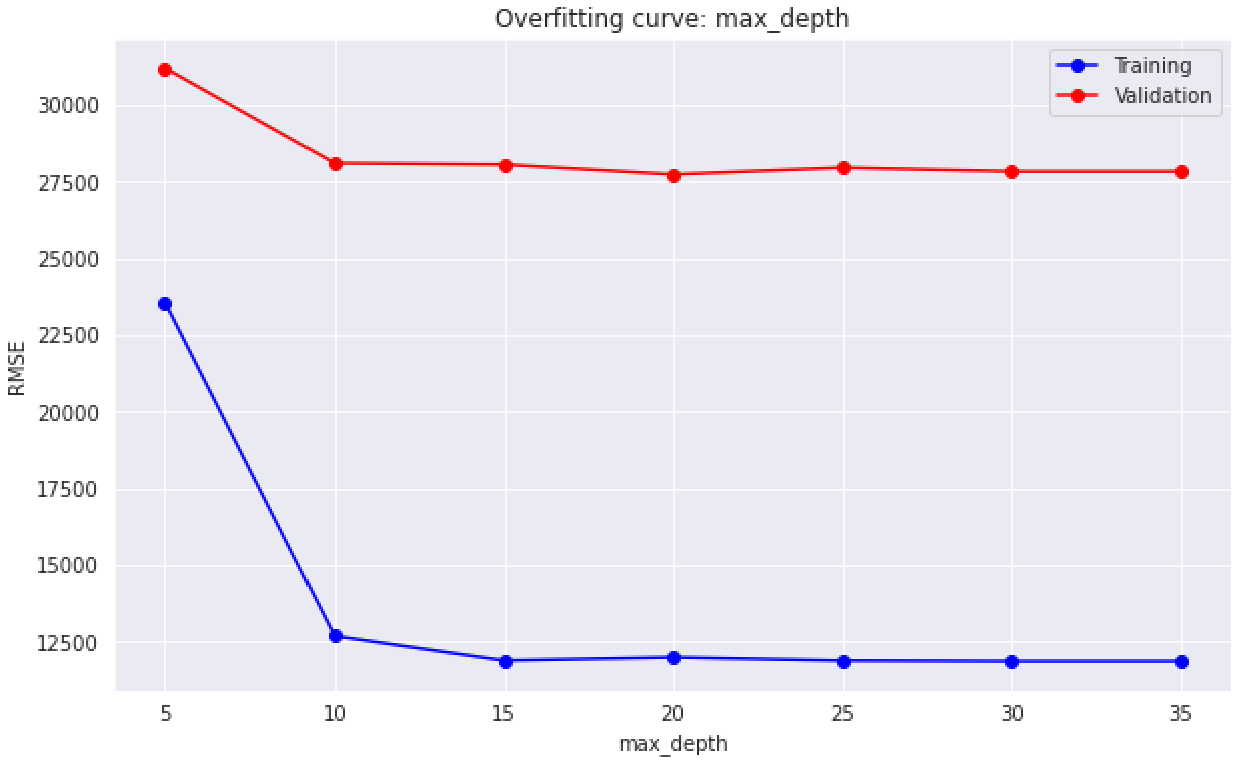
```
test_params(n_estimators=50, max_depth=10, min_samples_leaf=4, max_features=0.4)
```

```
(20490.359632429263, 29804.931642791606)
```

Let's also define a helper function to test and plot different values of a single parameter.

```
def test_param_and_plot(param_name, param_values):
    train_errors, val_errors = [], []
    for value in param_values:
        params = {param_name: value}
        train_rmse, val_rmse = test_params(**params)
        train_errors.append(train_rmse)
        val_errors.append(val_rmse)
    plt.figure(figsize=(10,6))
    plt.title('Overfitting curve: ' + param_name)
    plt.plot(param_values, train_errors, 'b-o')
    plt.plot(param_values, val_errors, 'r-o')
    plt.xlabel(param_name)
    plt.ylabel('RMSE')
    plt.legend(['Training', 'Validation'])
```

```
test_param_and_plot('max_depth', [5, 10, 15, 20, 25, 30, 35])
```

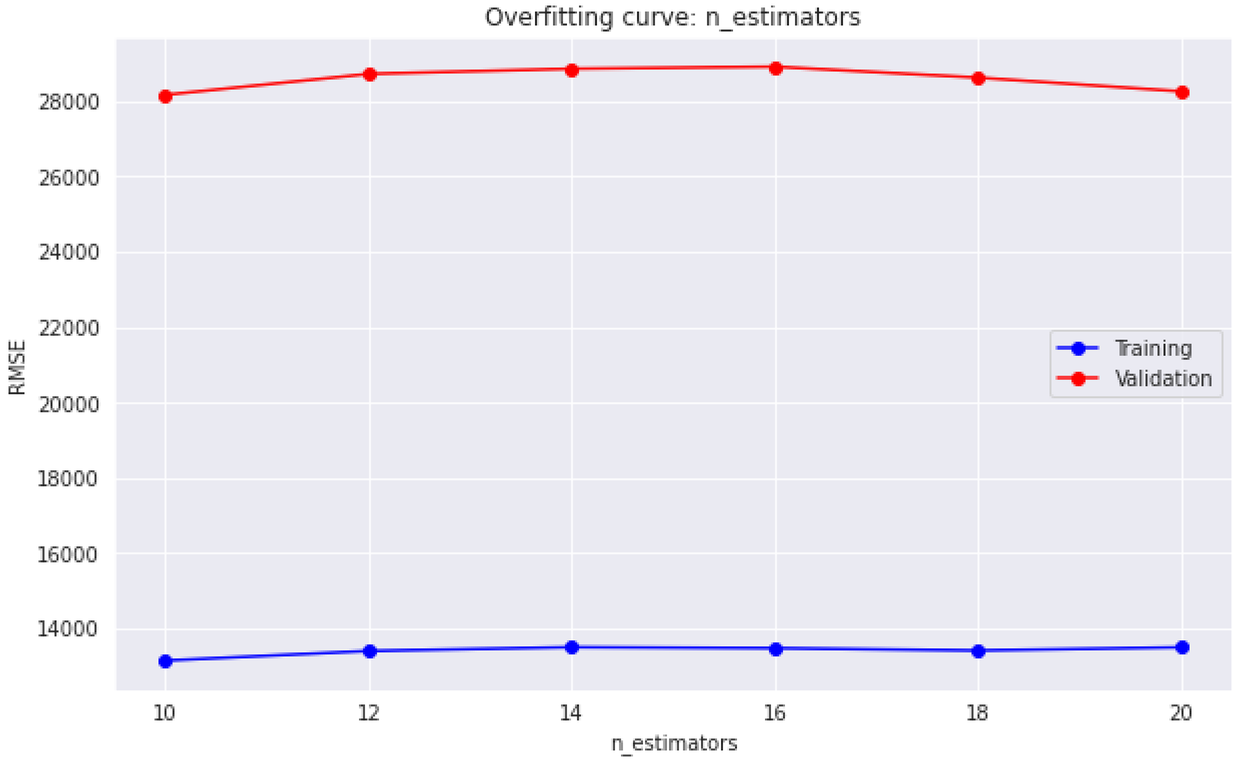


From the above graph, it appears that the best value for `max_depth` is around 20, beyond which the model starts to overfit.

> **QUESTION 6**: Use the `test_params` and `test_param_and_plot` functions to experiment with different values of the hyperparmeters like `n_estimators` , `max_depth` , `min_samples_split` , `min_samples_leaf` , `min_weight_fraction_leaf` , `max_features` , `max_leaf_nodes` , `min_impurity_decrease` , `min_impurity_split` etc. You can learn more about the hyperparameters here: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html

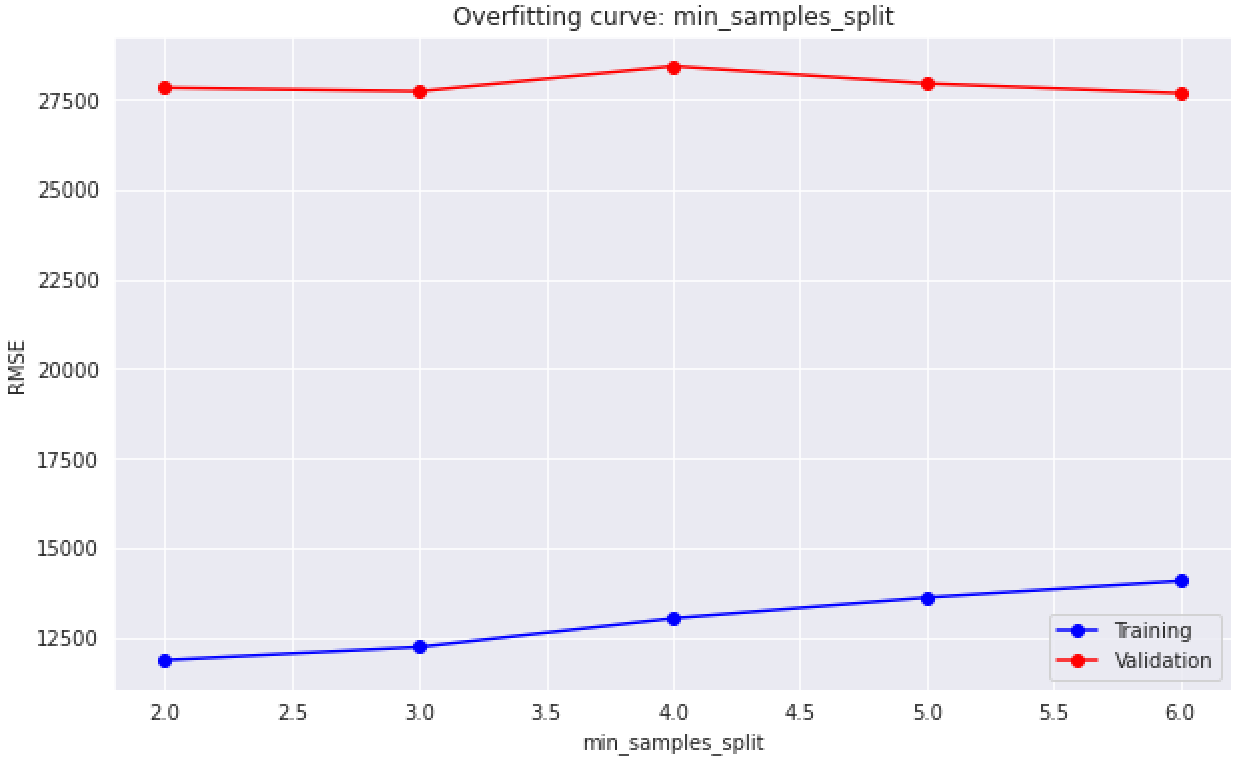
```
params_values = [10,12,14,16,18,20]
for value in params_values:
    print(test_params(n_estimators=value))
test_param_and_plot('n_estimators',params_values)
```

```
(13136.56133802938, 28167.975467898592)
(13392.64141808523, 28724.29762061054)
(13492.350084620002, 28861.751940871633)
(13466.544468027318, 28916.669289392612)
(13404.82882888439, 28625.803737112117)
(13488.72322462969, 28260.691071368405)
```



```
params_values = [2,3,4,5,6]
for value in params_values:
    print(test_params(min_samples_split=value))
test_param_and_plot('min_samples_split',params_values)
```

```
(11864.318299877767, 27830.03863639856)
(12232.781522986243, 27730.52618920368)
(13032.092652659161, 28431.98904011738)
(13614.310454297596, 27947.437052048183)
(14078.750068373442, 27676.206600642377)
```



Training the Best Model

> **QUESTION 7**: Train a random forest regressor model with your best hyperparameters to minimize the validation loss.

```
# Create the model with custom hyperparameters
rf2 = RandomForestRegressor(
        n_estimators=20,
        max_leaf_nodes=10,
        max_depth=100)
```

```
# Train the model
rf2.fit(train_inputs,train_targets)
```

```
    RandomForestRegressor(max_depth=100, max_leaf_nodes=10, n_estimators=20)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook. On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.
RandomForestRegressor

```
    RandomForestRegressor(max_depth=100, max_leaf_nodes=10, n_estimators=20)
```

Let's save our work before continuing.

> **QUESTION 8**: Make predictions and evaluate your final model. If you're unhappy with the results, modify the hyperparameters above and try again.

```
rf2_train_preds = rf2.predict(train_inputs)
```

```
rf2_train_rmse = mean_squared_error(train_targets, rf2_train_preds,squared=False)
```

```
rf2_val_preds = rf2.predict(val_inputs)
```

```
rf2_val_rmse = mean_squared_error(val_targets, rf2_val_preds,squared=False)
```

```
print('Train RMSE: {}, Validation RMSE: {}'.format(rf2_train_rmse, rf2_val_rmse))
```

```
Train RMSE: 32036.916620633863, Validation RMSE: 36207.795462386195
```

Let's also view and plot the feature importances.

```
rf2_importance_df = pd.DataFrame({
    'feature': train_inputs.columns,
    'importance': rf2.feature_importances_
}).sort_values('importance', ascending=False)
```

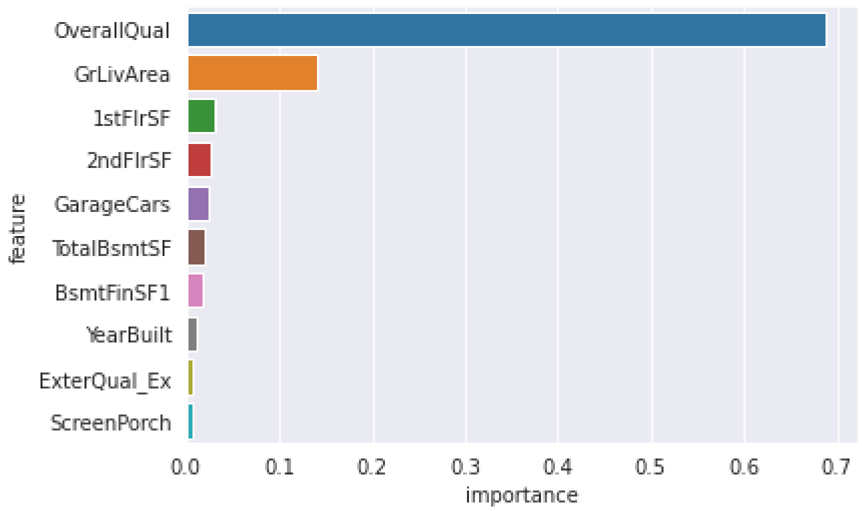
```
rf2_importance_df
```

	feature	importance
3	OverallQual	0.687120
15	GrLivArea	0.140287
12	1stFlrSF	0.031400
13	2ndFlrSF	0.025931
25	GarageCars	0.023759
...	...	...
119	RoofStyle_Flat	0.000000
120	RoofStyle_Gable	0.000000
121	RoofStyle_Gambrel	0.000000
122	RoofStyle_Hip	0.000000
303	SaleCondition_Partial	0.000000

304 rows × 2 columns

```
sns.barplot(data=rf2_importance_df.head(10), x='importance', y='feature')
```

```
<AxesSubplot:xlabel='importance', ylabel='feature'>
```



Let's save our work before continuing.

```
jovian.commit()
```

```
[jovian] Updating notebook "ahmedatif655/python-random-forests-assignment" on
https://jovian.ai
[jovian] Committed successfully! https://jovian.ai/ahmedatif655/python-random-forests-
assignment
```

```
'https://jovian.ai/ahmedatif655/python-random-forests-assignment'
```

Make a Submission

To make a submission, just execute the following cell:

```
jovian.submit('zerotogbms-a2')
```

You can also submit your Jovian notebook link on the assignment page: https://jovian.ai/learn/machine-learning-with-python-zero-to-gbms/assignment/assignment-2-decision-trees-and-random-forests

Make sure to review the evaluation criteria carefully. You can make any number of submissions, and only your final submission will be evalauted.

Ask questions, discuss ideas and get help here: https://jovian.ai/forum/c/zero-to-gbms/gbms-assignment-2/99

NOTE: **The rest of this assignment is optional.**

Making Predictions on the Test Set

Let's make predictions on the test set provided with the data.

```
test_df = pd.read_csv('house-prices/test.csv')
```

```
test_df
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfi
0	1461	20	RH	80.0	11622	Pave	NaN	Reg	Lvl	AllPub	Insid
1	1462	20	RL	81.0	14267	Pave	NaN	IR1	Lvl	AllPub	Corne
2	1463	60	RL	74.0	13830	Pave	NaN	IR1	Lvl	AllPub	Insid
3	1464	60	RL	78.0	9978	Pave	NaN	IR1	Lvl	AllPub	Insid
4	1465	120	RL	43.0	5005	Pave	NaN	IR1	HLS	AllPub	Insid
...	...	...	...	...	...	...	...	...	...	...	.
1454	2915	160	RM	21.0	1936	Pave	NaN	Reg	Lvl	AllPub	Insid
1455	2916	160	RM	21.0	1894	Pave	NaN	Reg	Lvl	AllPub	Insid
1456	2917	20	RL	160.0	20000	Pave	NaN	Reg	Lvl	AllPub	Insid
1457	2918	85	RL	62.0	10441	Pave	NaN	Reg	Lvl	AllPub	Insid
1458	2919	60	RL	74.0	9627	Pave	NaN	Reg	Lvl	AllPub	Insid

1459 rows × 80 columns

First, we need to reapply all the preprocessing steps.

```
test_df[numeric_cols] = imputer.transform(test_df[numeric_cols])
test_df[numeric_cols] = scaler.transform(test_df[numeric_cols])
test_df[encoded_cols] = encoder.transform(test_df[categorical_cols])
```

```
/opt/conda/lib/python3.9/site-packages/pandas/core/frame.py:3678: PerformanceWarning:
DataFrame is highly fragmented.  This is usually the result of calling `frame.insert`
```

```
many times, which has poor performance.  Consider joining all columns at once using
pd.concat(axis=1) instead.  To get a de-fragmented frame, use `newframe = frame.copy()`
  self[col] = igetitem(value, i)
```

```
test_inputs = test_df[numeric_cols + encoded_cols]
```

We can now make predictions using our final model.

```
test_preds = rf2.predict(test_inputs)
```

```
submission_df = pd.read_csv('house-prices/sample_submission.csv')
```

```
submission_df
```

	Id	SalePrice
0	1461	169277.052498
1	1462	187758.393989
2	1463	183583.683570
3	1464	179317.477511
4	1465	150730.079977
...	...	...
1454	2915	167081.220949
1455	2916	164788.778231
1456	2917	219222.423400
1457	2918	184924.279659
1458	2919	187741.866657

1459 rows × 2 columns

Let's replace the values of the `SalePrice` column with our predictions.

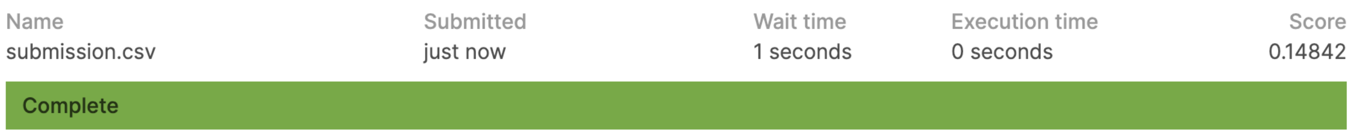
```
submission_df['SalePrice'] = test_preds
```

Let's save it as a CSV file and download it.

```
submission_df.to_csv('submission.csv', index=False)
```

```
from IPython.display import FileLink
FileLink('submission.csv') # Doesn't work on Colab, use the file browser instead to dow
```

We can now submit this file to the competition: https://www.kaggle.com/c/house-prices-advanced-regression-techniques/submissions



Name
submission.csv

Submitted
just now

Wait time
1 seconds

Execution time
0 seconds

Score
0.14842

Complete

> **(OPTIONAL) QUESTION**: Submit your predictions to the competition. Experiment with different models, feature engineering strategies and hyperparameters and try to reach the top 10% on the leaderboard.

Let's save our work before continuing.

```
jovian.commit()
```

```
[jovian] Updating notebook "aakashns/python-random-forests-assignment" on
https://jovian.ai/
[jovian] Committed successfully! https://jovian.ai/aakashns/python-random-forests-
assignment
```

```
'https://jovian.ai/aakashns/python-random-forests-assignment'
```

Making Predictions on Single Inputs

```
def predict_input(model, single_input):
    input_df = pd.DataFrame([single_input])
    input_df[numeric_cols] = imputer.transform(input_df[numeric_cols])
    input_df[numeric_cols] = scaler.transform(input_df[numeric_cols])
    input_df[encoded_cols] = encoder.transform(input_df[categorical_cols].values)
    return model.predict(input_df[numeric_cols + encoded_cols])[0]
```

```
sample_input = { 'MSSubClass': 20, 'MSZoning': 'RL', 'LotFrontage': 77.0, 'LotArea': 93
 'Street': 'Pave', 'Alley': None, 'LotShape': 'IR1', 'LandContour': 'Lvl', 'Utilities':
 'LotConfig': 'Inside', 'LandSlope': 'Gtl', 'Neighborhood': 'NAmes', 'Condition1': 'Nor
 'BldgType': '1Fam', 'HouseStyle': '1Story', 'OverallQual': 4, 'OverallCond': 5, 'YearB
 'YearRemodAdd': 1959, 'RoofStyle': 'Gable', 'RoofMatl': 'CompShg', 'Exterior1st': 'Ply
 'Exterior2nd': 'Plywood', 'MasVnrType': 'None','MasVnrArea': 0.0,'ExterQual': 'TA','Ex
 'Foundation': 'CBlock','BsmtQual': 'TA','BsmtCond': 'TA','BsmtExposure': 'No','BsmtFin
 'BsmtFinSF1': 569,'BsmtFinType2': 'Unf','BsmtFinSF2': 0,'BsmtUnfSF': 381,
 'TotalBsmtSF': 950,'Heating': 'GasA','HeatingQC': 'Fa','CentralAir': 'Y','Electrical':
 '2ndFlrSF': 0, 'LowQualFinSF': 0, 'GrLivArea': 1225, 'BsmtFullBath': 1, 'BsmtHalfBath'
 'HalfBath': 1, 'BedroomAbvGr': 3, 'KitchenAbvGr': 1,'KitchenQual': 'TA','TotRmsAbvGrd'
 'Fireplaces': 0,'FireplaceQu': np.nan,'GarageType': np.nan,'GarageYrBlt': np.nan,'Gara
 'GarageArea': 0,'GarageQual': np.nan,'GarageCond': np.nan,'PavedDrive': 'Y', 'WoodDeck
 'EnclosedPorch': 0,'3SsnPorch': 0, 'ScreenPorch': 0, 'PoolArea': 0, 'PoolQC': np.nan,
 'MiscVal': 400, 'MoSold': 1, 'YrSold': 2010, 'SaleType': 'WD', 'SaleCondition': 'Norma
```

```
predicted_price = predict_input(rf2, sample_input)
```

```
print('The predicted sale price of the house is ${}'.format(predicted_price))
```

> **EXERCISE**: Change the sample input above and make predictions. Try different examples and try to figure out which columns have a big impact on the sale price. Hint: Look at the feature importance to decide which columns to try.

Saving the Model

```
import joblib
```

```
house_prices_rf = {
    'model': rf2,
    'imputer': imputer,
    'scaler': scaler,
    'encoder': encoder,
    'input_cols': input_cols,
    'target_col': target_col,
    'numeric_cols': numeric_cols,
    'categorical_cols': categorical_cols,
    'encoded_cols': encoded_cols
}
```

```
joblib.dump(house_prices_rf, 'house_prices_rf.joblib')
```

Let's save our work before continuing.

```
jovian.commit(outputs=['house_prices_rf.joblib'])
```

Predicting the Logarithm of Sale Price

> **(OPTIONAL) QUESTION**: In the original Kaggle competition, the model is evaluated by computing the Root Mean Squared Error on the logarithm of the sale price. Try training a random forest to predict the

> logarithm of the sale price, instead of the actual sales price and see if the results you obtain are better than the models trained above.