# index

March 21, 2022

# 1 Regression Trees and Model Optimization - Lab

## 1.1 Introduction

In this lab, we'll see how to apply regression analysis using CART trees while making use of some hyperparameter tuning to improve our model.

## 1.2 Objectives

In this lab you will:

- Perform the full process of cleaning data, tuning hyperparameters, creating visualizations, and evaluating decision tree models
- Determine the optimal hyperparameters for a decision tree model and evaluate the performance of decision tree models

## 1.3 Ames Housing dataset

The dataset is available in the file `'ames.csv'`.

- Import the dataset and examine its dimensions:

[6]:
```
# Import necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
plt.style.use('ggplot')
%matplotlib inline

# Load the Ames housing dataset
data = pd.read_csv("ames.csv")

# Print the dimensions of data
data.shape

# Check out the info for the dataframe
data.info()

# Show the first 5 rows
```

```
data.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 81 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             1460 non-null   int64
 1   MSSubClass     1460 non-null   int64
 2   MSZoning       1460 non-null   object
 3   LotFrontage    1201 non-null   float64
 4   LotArea        1460 non-null   int64
 5   Street         1460 non-null   object
 6   Alley          91 non-null     object
 7   LotShape       1460 non-null   object
 8   LandContour    1460 non-null   object
 9   Utilities      1460 non-null   object
 10  LotConfig      1460 non-null   object
 11  LandSlope      1460 non-null   object
 12  Neighborhood   1460 non-null   object
 13  Condition1     1460 non-null   object
 14  Condition2     1460 non-null   object
 15  BldgType       1460 non-null   object
 16  HouseStyle     1460 non-null   object
 17  OverallQual    1460 non-null   int64
 18  OverallCond    1460 non-null   int64
 19  YearBuilt      1460 non-null   int64
 20  YearRemodAdd   1460 non-null   int64
 21  RoofStyle      1460 non-null   object
 22  RoofMatl       1460 non-null   object
 23  Exterior1st    1460 non-null   object
 24  Exterior2nd    1460 non-null   object
 25  MasVnrType     1452 non-null   object
 26  MasVnrArea     1452 non-null   float64
 27  ExterQual      1460 non-null   object
 28  ExterCond      1460 non-null   object
 29  Foundation     1460 non-null   object
 30  BsmtQual       1423 non-null   object
 31  BsmtCond       1423 non-null   object
 32  BsmtExposure   1422 non-null   object
 33  BsmtFinType1   1423 non-null   object
 34  BsmtFinSF1     1460 non-null   int64
 35  BsmtFinType2   1422 non-null   object
 36  BsmtFinSF2     1460 non-null   int64
 37  BsmtUnfSF      1460 non-null   int64
 38  TotalBsmtSF    1460 non-null   int64
 39  Heating        1460 non-null   object
```

```
 40  HeatingQC      1460 non-null   object
 41  CentralAir     1460 non-null   object
 42  Electrical     1459 non-null   object
 43  1stFlrSF       1460 non-null   int64
 44  2ndFlrSF       1460 non-null   int64
 45  LowQualFinSF   1460 non-null   int64
 46  GrLivArea      1460 non-null   int64
 47  BsmtFullBath   1460 non-null   int64
 48  BsmtHalfBath   1460 non-null   int64
 49  FullBath       1460 non-null   int64
 50  HalfBath       1460 non-null   int64
 51  BedroomAbvGr   1460 non-null   int64
 52  KitchenAbvGr   1460 non-null   int64
 53  KitchenQual    1460 non-null   object
 54  TotRmsAbvGrd   1460 non-null   int64
 55  Functional     1460 non-null   object
 56  Fireplaces     1460 non-null   int64
 57  FireplaceQu    770 non-null    object
 58  GarageType     1379 non-null   object
 59  GarageYrBlt    1379 non-null   float64
 60  GarageFinish   1379 non-null   object
 61  GarageCars     1460 non-null   int64
 62  GarageArea     1460 non-null   int64
 63  GarageQual     1379 non-null   object
 64  GarageCond     1379 non-null   object
 65  PavedDrive     1460 non-null   object
 66  WoodDeckSF     1460 non-null   int64
 67  OpenPorchSF    1460 non-null   int64
 68  EnclosedPorch  1460 non-null   int64
 69  3SsnPorch      1460 non-null   int64
 70  ScreenPorch    1460 non-null   int64
 71  PoolArea       1460 non-null   int64
 72  PoolQC         7 non-null      object
 73  Fence          281 non-null    object
 74  MiscFeature    54 non-null     object
 75  MiscVal        1460 non-null   int64
 76  MoSold         1460 non-null   int64
 77  YrSold         1460 non-null   int64
 78  SaleType       1460 non-null   object
 79  SaleCondition  1460 non-null   object
 80  SalePrice      1460 non-null   int64
dtypes: float64(3), int64(35), object(43)
memory usage: 924.0+ KB
```

[6]:

```
   Id  MSSubClass MSZoning  LotFrontage  LotArea Street Alley LotShape  \
0   1          60       RL         65.0     8450   Pave   NaN      Reg
1   2          20       RL         80.0     9600   Pave   NaN      Reg
```

```
2   3           60         RL          68.0     11250    Pave    NaN       IR1
3   4           70         RL          60.0      9550    Pave    NaN       IR1
4   5           60         RL          84.0     14260    Pave    NaN       IR1

  LandContour Utilities  … PoolArea PoolQC Fence MiscFeature MiscVal MoSold  \
0         Lvl    AllPub  …        0    NaN   NaN         NaN       0      2
1         Lvl    AllPub  …        0    NaN   NaN         NaN       0      5
2         Lvl    AllPub  …        0    NaN   NaN         NaN       0      9
3         Lvl    AllPub  …        0    NaN   NaN         NaN       0      2
4         Lvl    AllPub  …        0    NaN   NaN         NaN       0     12

  YrSold  SaleType  SaleCondition  SalePrice
0   2008        WD         Normal     208500
1   2007        WD         Normal     181500
2   2008        WD         Normal     223500
3   2006        WD        Abnorml     140000
4   2008        WD         Normal     250000

[5 rows x 81 columns]
```

## 1.4 Identify features and target data

In this lab, we will use using 3 predictive continuous features:

### Features

- `LotArea`: Lot size in square feet
- `1stFlrSF`: Size of first floor in square feet
- `GrLivArea`: Above grade (ground) living area square feet

### Target

- `SalePrice`', the sale price of the home, in dollars

- Create DataFrames for the features and the target variable as shown above
- Inspect the contents of both the features and the target variable

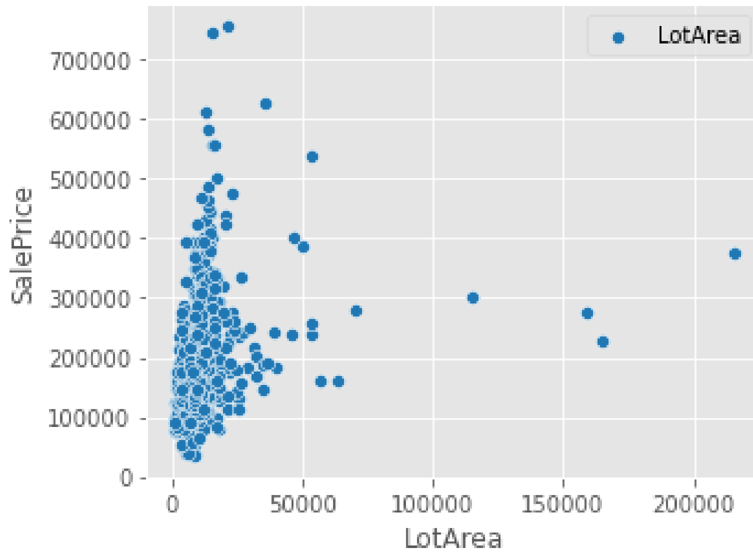
[7]:
```python
# Features and target data
target = data["SalePrice"]
features = data[["LotArea", "1stFlrSF", "GrLivArea"]]
```

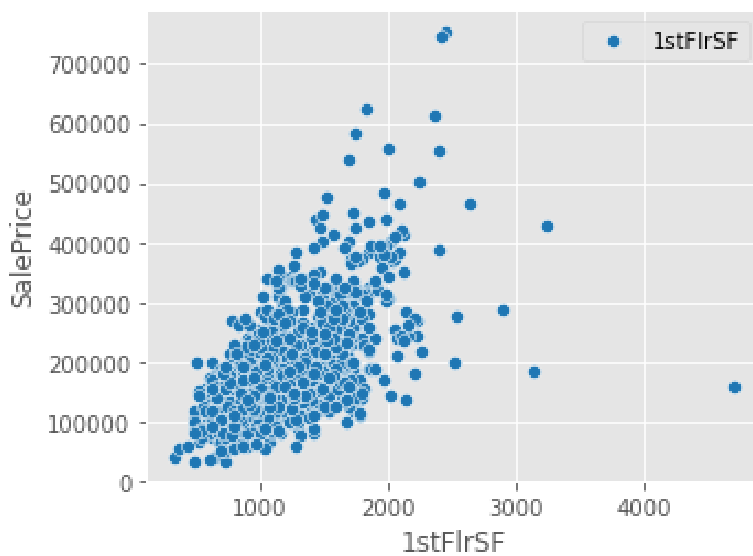
## 1.5 Inspect correlations

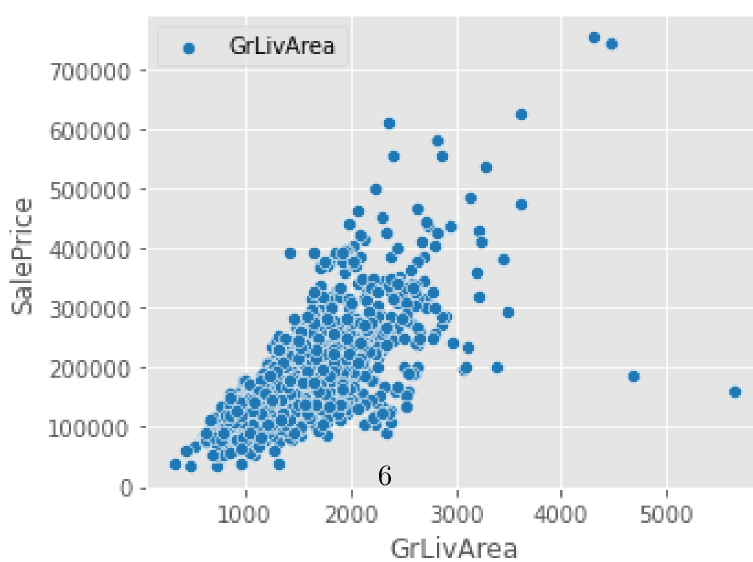
- Use scatter plots to show the correlation between the chosen features and the target variable
- Comment on each scatter plot

[12]:
```python
# Your code here

fig, axes = plt.subplots(nrows = 3, ncols = 1, figsize = (5,15))
```

```
fig.subplots_adjust(hspace=0.4, wspace=0.25)


for i,item in enumerate(features.columns):
    ax = axes[i]
    sns.scatterplot(x = features[item], y = target, ax = ax, label = item,
                color = "tab:blue")
```


LotArea
SalePrice
LotArea
0
100000
200000
300000
400000
500000
600000
700000
0
50000
100000
150000
200000


1stFlrSF
SalePrice
1stFlrSF
0
100000
200000
300000
400000
500000
600000
700000
1000
2000
3000
4000


GrLivArea
SalePrice
GrLivArea
0
100000
200000
300000
400000
500000
600000
700000
1000
2000
3000
4000
5000

### 1.6 Create evaluation metrics

- Import `r2_score` and `mean_squared_error` from `sklearn.metrics`
- Create a function `performance(true, predicted)` to calculate and return both the R-squared score and Root Mean Squared Error (RMSE) for two equal-sized arrays for the given true and predicted values
  - Depending on your version of sklearn, in order to get the RMSE score you will need to either set `squared=False` or you will need to take the square root of the output of the `mean_squared_error` function - check out the documentation or this helpful and related StackOverflow post
  - The benefit of calculating RMSE instead of the Mean Squared Error (MSE) is that RMSE is in the same units at the target - here, this means that RMSE will be in dollars, calculating how far off in dollars our predictions are away from the actual prices for homes, on average

[26]:
```python
# Import metrics
from sklearn.metrics import r2_score, mean_squared_error

# Define the function
def performance(y_true, y_predict):
    """
    Calculates and returns the two performance scores between
    true and predicted values - first R-Squared, then RMSE
    """

    # Calculate the r2 score between 'y_true' and 'y_predict'
    R2 = r2_score(y_true, y_predict)
    # Calculate the root mean squared error between 'y_true' and 'y_predict'
    RMSE = mean_squared_error(y_true, y_predict, squared=False)
    # Return the score

    return R2, RMSE


# Test the function
score = performance([3, -0.5, 2, 7, 4.2], [2.5, 0.0, 2.1, 7.8, 5.3])
score

# [0.9228556485355649, 0.6870225614927066]
```

[26]:
```
(0.9228556485355649, 0.6870225614927066)
```

### 1.7 Split the data into training and test sets

- Split `features` and `target` datasets into training/test data (80/20)

- For reproducibility, use `random_state=42`

[27]:
```
from sklearn.model_selection import train_test_split

# Split the data into training and test subsets
x_train, x_test, y_train, y_test = train_test_split(features, target,
                                                    test_size = 0.2,
                                                    random_state = 42)
```

### 1.8 Grow a vanilla regression tree

- Import the `DecisionTreeRegressor` class
- Run a baseline model for later comparison using the datasets created above
- Generate predictions for test dataset and calculate the performance measures using the function created above
- Use `random_state=45` for tree instance
- Record your observations

[28]:
```
# Import DecisionTreeRegressor

from sklearn.tree import DecisionTreeRegressor

# Instantiate DecisionTreeRegressor
# Set random_state=45
regressor = DecisionTreeRegressor(random_state = 45)

# Fit the model to training data
regressor.fit(x_train, y_train)

# Make predictions on the test data
y_pred = regressor.predict(x_test)

# Calculate performance using the performance() function
score = performance(y_test, y_pred)
score

# [0.5961521990414137, 55656.48543887347] - R2, RMSE
```

[28]:
```
(0.5961521990414137, 55656.48543887347)
```

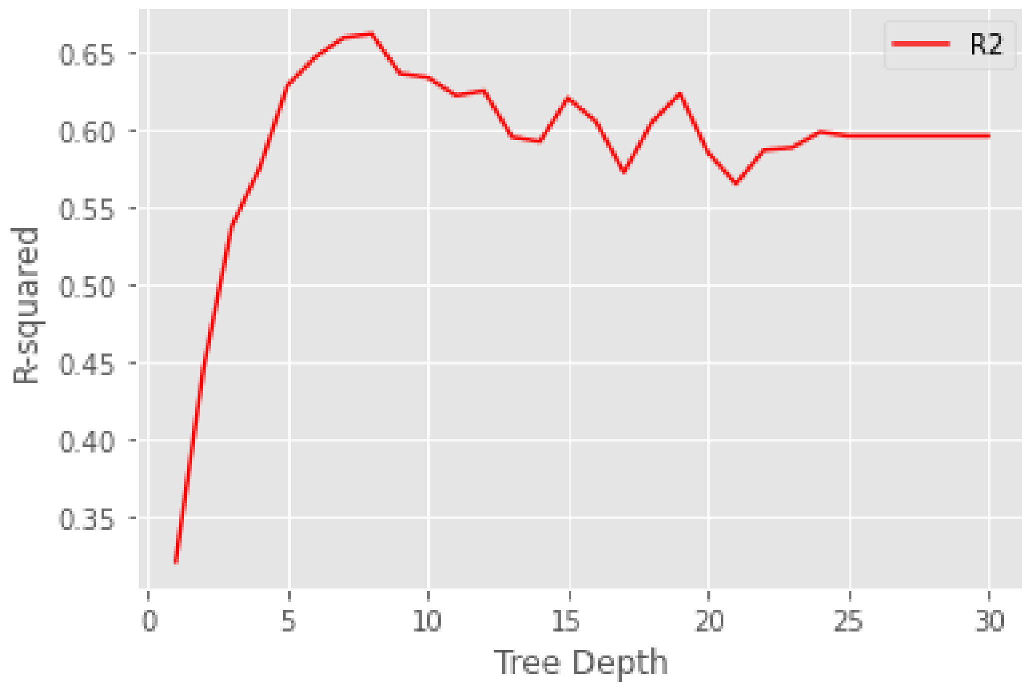
### 1.9 Hyperparameter tuning (I)

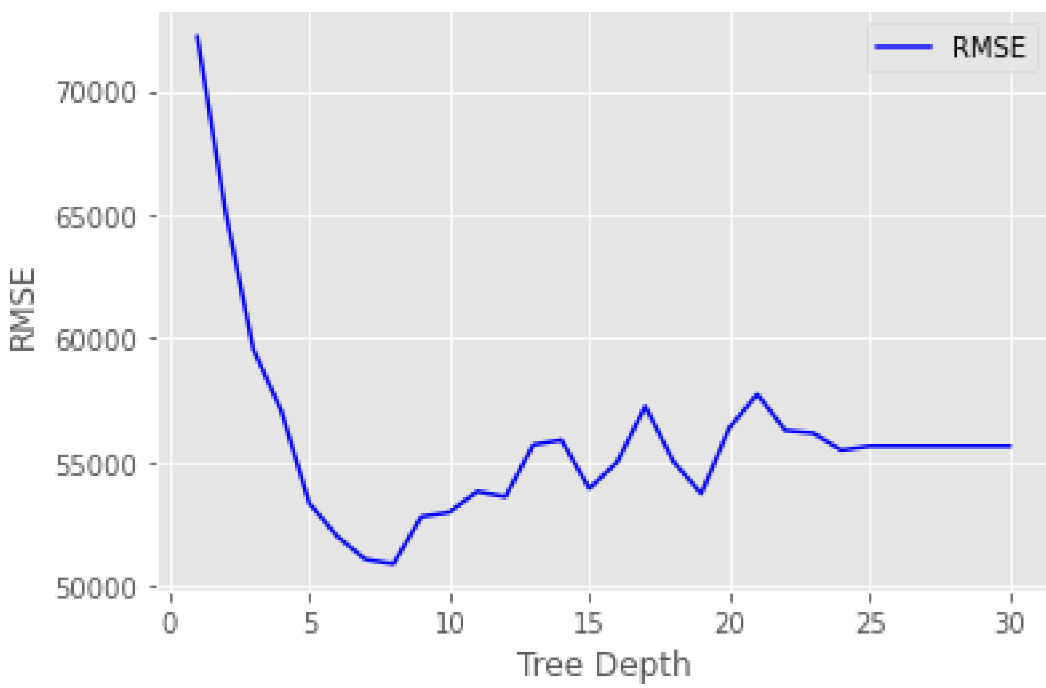
- Find the best tree depth using depth range: 1-30
- Run the regressor repeatedly in a `for` loop for each depth value

- Use `random_state=45` for reproducibility
- Calculate RMSE and r-squared for each run
- Plot both performance measures for all runs

- Comment on the output

[38]:
```
# Your code here

depth = range(1, 31)

R2 = []
RMSE = []

for i in depth:

    regressor = DecisionTreeRegressor(random_state = 45, max_depth = i)

# Fit the model to training data
    regressor.fit(x_train, y_train)

# Make predictions on the test data
    y_pred = regressor.predict(x_test)

# Calculate performance using the performance() function
    score = performance(y_test, y_pred)
    R2.append(score[0])
    RMSE.append(score[1])

sns.lineplot(x = depth, y = R2, label = "R2", color = "r");
plt.xlabel('Tree Depth')
plt.ylabel('R-squared')
plt.show()
sns.lineplot(x = depth, y = RMSE, label = "RMSE", color = "blue")
plt.xlabel('Tree Depth')
plt.ylabel('RMSE')
plt.show()
```



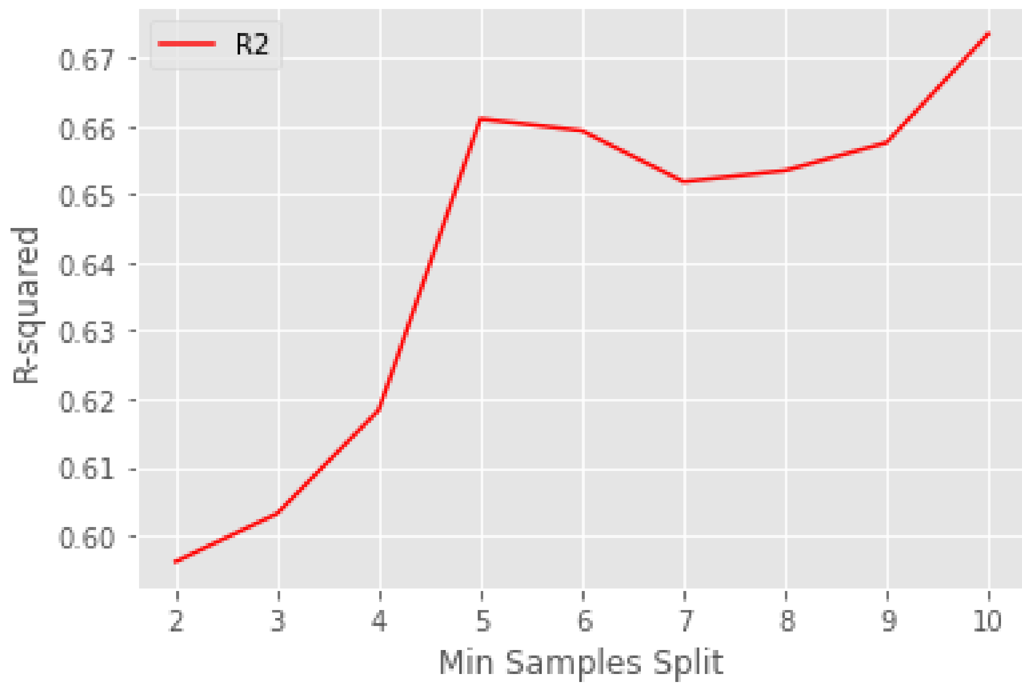



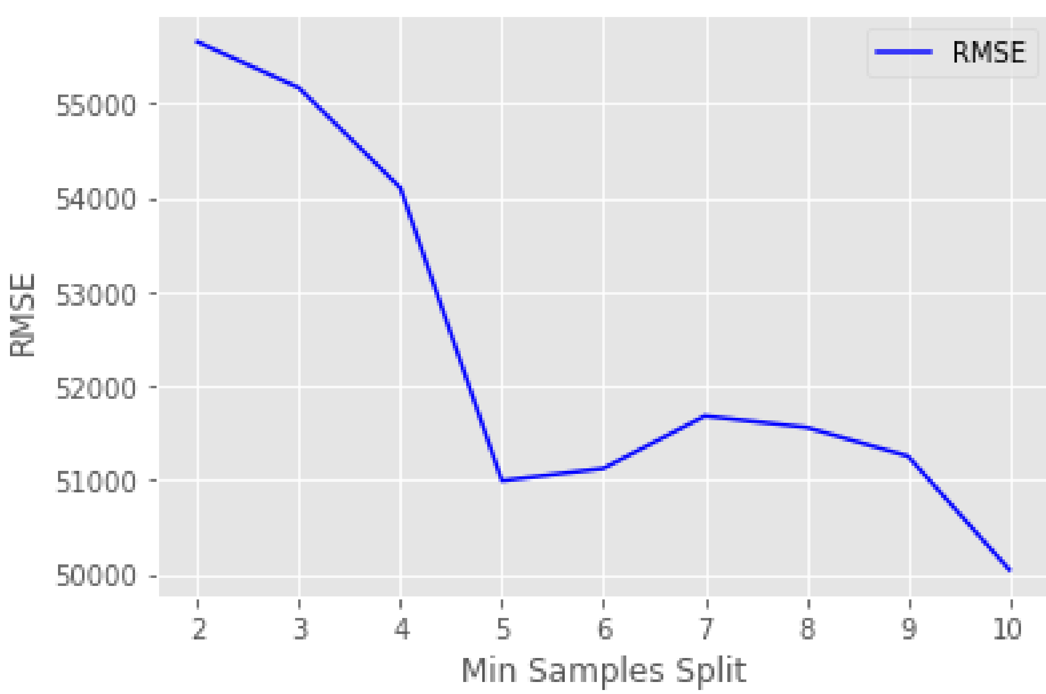

[40]:
```
# We can see that at tree depth that is equal to 7, we have minimum value for
# RMSE and maximum value for R2. So max_depth = 7 is the optimum value
```

## 1.10 Hyperparameter tuning (II)

- Repeat the above process for `min_samples_split`
- Use a range of values from 2-10 for this hyperparameter
- Use `random_state=45` for reproducibility
- Visualize the output and comment on results as above

[39]:
```
# Your code here

min_samples_split = range(2, 11)

R2 = []
RMSE = []

for i in min_samples_split:

    regressor = DecisionTreeRegressor(random_state = 45, min_samples_split=i)

# Fit the model to training data
    regressor.fit(x_train, y_train)

# Make predictions on the test data
    y_pred = regressor.predict(x_test)

# Calculate performance using the performance() function
    score = performance(y_test, y_pred)
    R2.append(score[0])
    RMSE.append(score[1])

sns.lineplot(x = min_samples_split, y = R2, label = "R2", color = "r");
plt.xlabel('Min Samples Split')
plt.ylabel('R-squared')
plt.show()
sns.lineplot(x = min_samples_split, y = RMSE, label = "RMSE", color = "blue")
plt.xlabel('Min Samples Split')
plt.ylabel('RMSE')
plt.show()
```


R2
0.67
0.66
0.65
0.64
0.63
0.62
0.61
0.60
R-squared
2
3
4
5
6
7
8
9
10
Min Samples Split


RMSE
55000
54000
53000
52000
51000
50000
RMSE
2
3
4
5
6
7
8
9
10
Min Samples Split

# 2 Run the *optimized* model

- Use the best values for `max_depth` and `min_samples_split` found in previous runs and run an optimized model with these values
- Calculate the performance and comment on the output

[44]:
```python
# Your code here

regressor = DecisionTreeRegressor(random_state = 45, min_samples_split=10,
                                  max_depth = 7)

# Fit the model to training data
regressor.fit(x_train, y_train)

# Make predictions on the test data
y_pred = regressor.predict(x_test)

# Calculate performance using the performance() function
score = performance(y_test, y_pred)
print("R2   :", score[0])
print("RMSE :", score[1])
```

```
R2   : 0.6772046046188305
RMSE : 49758.87841130982
```

## 2.1 Level up (Optional)

- How about bringing in some more features from the original dataset which may be good predictors?
- Also, try tuning more hyperparameters like `max_features` to find a more optimal version of the model

[ ]:
```python
# Your code here
```

## 2.2 Summary

In this lab, we looked at applying a decision-tree-based regression analysis on the Ames Housing dataset. We saw how to train various models to find the optimal values for hyperparameters.