# Regression Trees

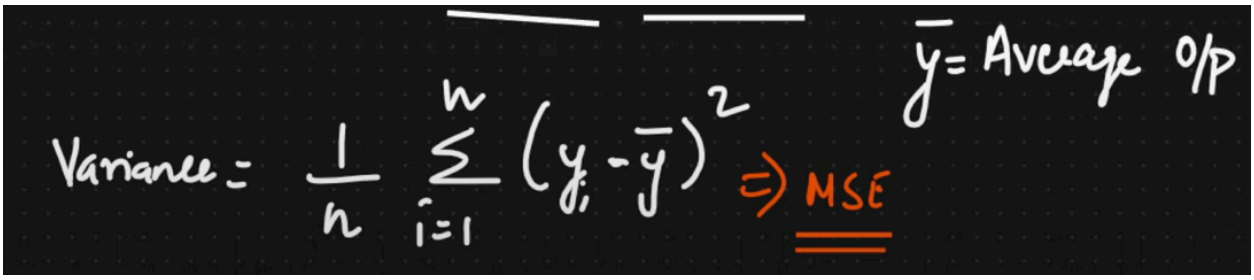
```
from sklearn.tree import DecisionTreeRegressor
```

- Useful for non-linear data

## How Do Regression Trees Work?

1. **Start at the Root**:
   - The tree starts with the entire dataset at the **root node**.
2. **Ask Questions (Split the Data)**:
   - The tree asks a question (e.g., "Is the house size greater than 1000 sq. ft.?") and splits the data into smaller groups based on the answer.
   - The goal is to split the data in a way that reduces the **variance** (or error) in the target variable.
3. **Repeat**:
   - The process repeats for each smaller group, asking more questions and splitting further.
4. **Stop When You Reach a Leaf**:
   - The splitting stops when the data in a group is homogeneous (low variance) or when a stopping condition is met (e.g., maximum depth of the tree).
   - The final groups are called **leaf nodes**, and they represent the predictions. Each leaf node predicts the **average value** of the target variable for the data points in that group.

### In Regression problem, Split is decided by →VARIANCE REDUCTION






- **We are more confident about a datapoint if it falls in low variance region as the data is less spread out.**

# Advantages of Regression Trees

1. **Easy to Understand**:
    - The tree structure is intuitive and easy to visualize.
2. **No Need for Data Scaling**:

- Regression Trees don’t require the data to be scaled or normalized.

3. **Handles Both Numerical and Categorical Data**:
    - You can use Regression Trees for datasets with both numbers and categories.
4. **Non-Parametric**:
    - Regression Trees don’t make assumptions about the data distribution.

# Disadvantages of Regression Trees

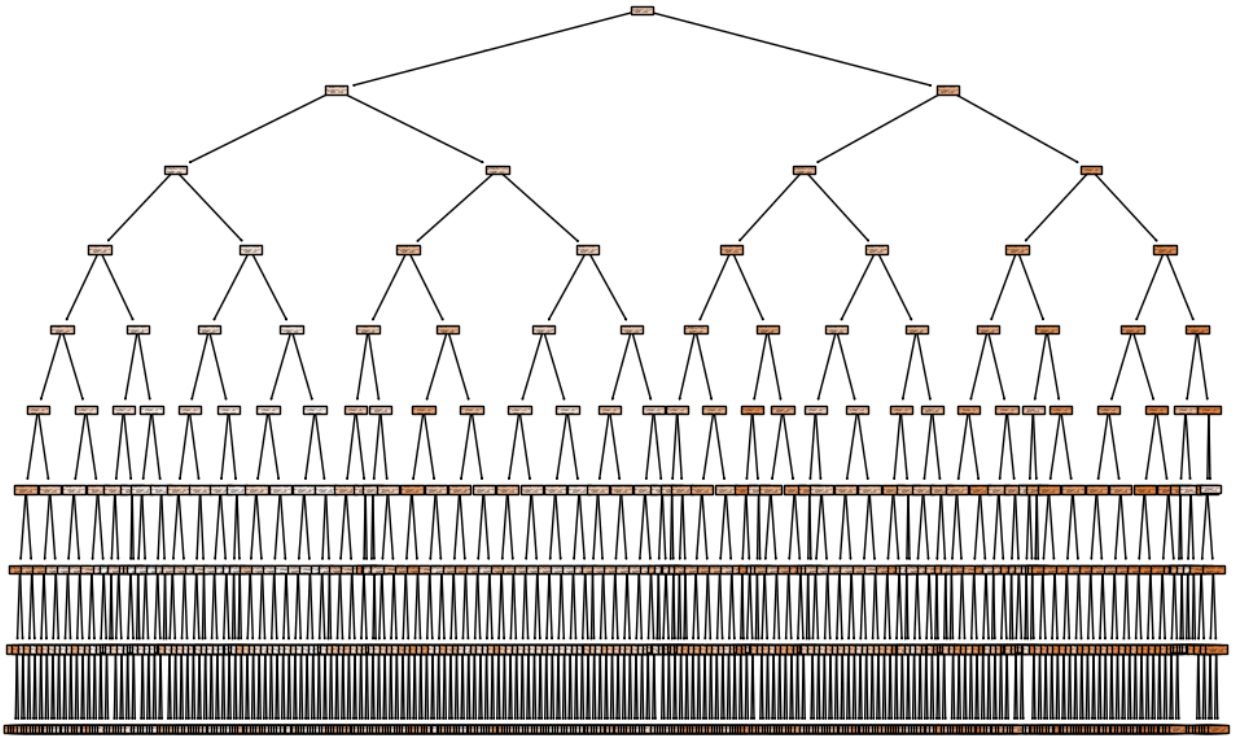
1. **Overfitting**:
    - If the tree is too deep, it can memorize the training data and perform poorly on new data.
2. **Unstable**:
    - Small changes in the data can lead to a completely different tree.
3. **Poor Performance on Linear Relationships**:
    - Regression Trees are better at capturing non-linear relationships. For linear relationships, linear regression may perform better.

## Python Code:

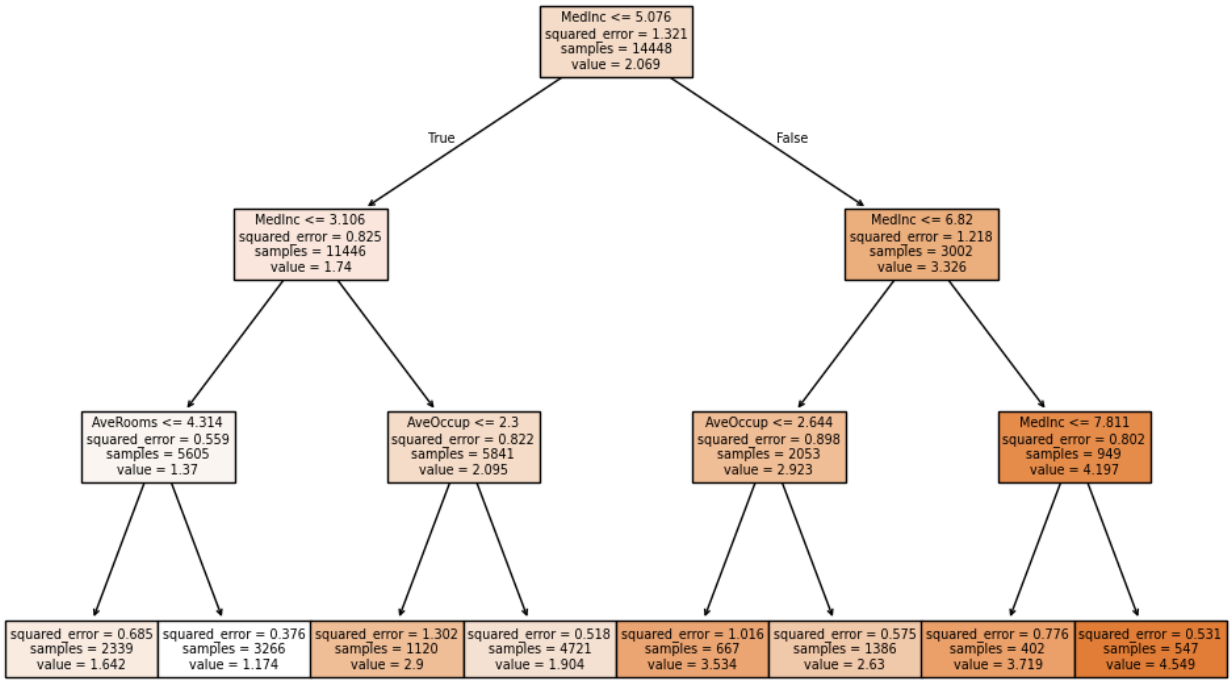
```
from sklearn.datasets import fetch_california_housing
from sklearn.tree import DecisionTreeRegressor, plot_tree
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt

# Load the California Housing dataset
data = fetch_california_housing()
X = data.data  # Features (e.g., median income, house age, etc.)
y = data.target  # Target (median house value)
```

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

# Create a Regression Tree model
tree = DecisionTreeRegressor(max_depth=9)  # Limit the depth to 3 for simplic
ity
tree.fit(X_train, y_train)

# Make predictions
y_pred = tree.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)
print("R2 Score: ", r2_score(y_test, y_pred))

# Visualize the Regression Tree
plt.figure(figsize=(12, 8))
plot_tree(tree, filled=True, feature_names=data.feature_names)
plt.show()
```

```
Mean Squared Error: 0.37868480337993443
R2 Score:  0.7159333298203459
```



- 👆This not readable. Just to make this readable, the following is result with `max_depth=3`




## Python code2

- Dataset → Boston Housing

```
import pandas as pd
from pandas_datareader import data
import numpy as np
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.metrics import r2_score
from sklearn.model_selection import GridSearchCV

df = pd.read_csv(r"https://raw.githubusercontent.com/selva86/datasets/refs/heads/master/BostonHousing.csv")
```

```
X = df.iloc[:,0:13]
y = df.iloc[:,13]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,random_stat
e=42)

rt = DecisionTreeRegressor(criterion = 'squared_error', max_depth=5)

rt.fit(X_train,y_train)

y_pred = rt.predict(X_test)

r2_score(y_test,y_pred)
```

```
0.885137272531848
```

## Hyperparameter Tuning:

```
param_grid = {
    'max_depth':[2,4,8,9,10,None],
    'criterion':['squared_error','friedman_mse','mae', 'absolute_error','poisson'],
    'max_features':[0.25,0.5,1.0],
    'min_samples_split':[0.25,0.5,1.0]
}

reg = GridSearchCV(DecisionTreeRegressor(),param_grid=param_grid, cv=5)

reg.fit(X_train,y_train)

reg.best_score_
```

```
0.7201826312636849
```

```
reg.best_params_
```

```
{'criterion': 'poisson',
 'max_depth': None,
 'max_features': 1.0,
 'min_samples_split': 0.25}
```

# Feature Importance

- You can figure out which features are more important using `feature_importances_`
- This can be used for feature selection.
- Same thing is in Random Forest.
  - **RF one is more useful than this one.**

```
rt.feature_importances_
```

```
array([0.04686682, 0.        , 0.00262747, 0.        , 0.02531597,
       0.63534866, 0.00617613, 0.06659581, 0.        , 0.        ,
       0.0043911 , 0.01723982, 0.19543823])
```

```
for importance, name in sorted(zip(rt.feature_importances_, X_train.columns),
reverse=True):
  print (name, importance)
```

```
rm 0.6353486570603067
lstat 0.19543823011083658
dis 0.0665958125859787
crim 0.046866818631323276
nox 0.025315973555601
b 0.017239815647775103
age 0.006176126174367112
ptratio 0.004391097507129
indus 0.002627468726682285
zn 0.0
tax 0.0
rad 0.0
chas 0.0
```