Contents

1 Step 4 Machine learning

1.1 Step 0: Read the dataset

Read csv-file

Look at the first row of the dataset

	areaCode	areaName	areaType	newPeopleVaccinatedFirstDoseByVaccinationDate	newPeopleVaccinatedSecondDoseByVaccinationDate	ne
2022-01-26	E12000009	South West	region	986	2520	

Drop unnecessary columns: areaCode, areaName, areaType.

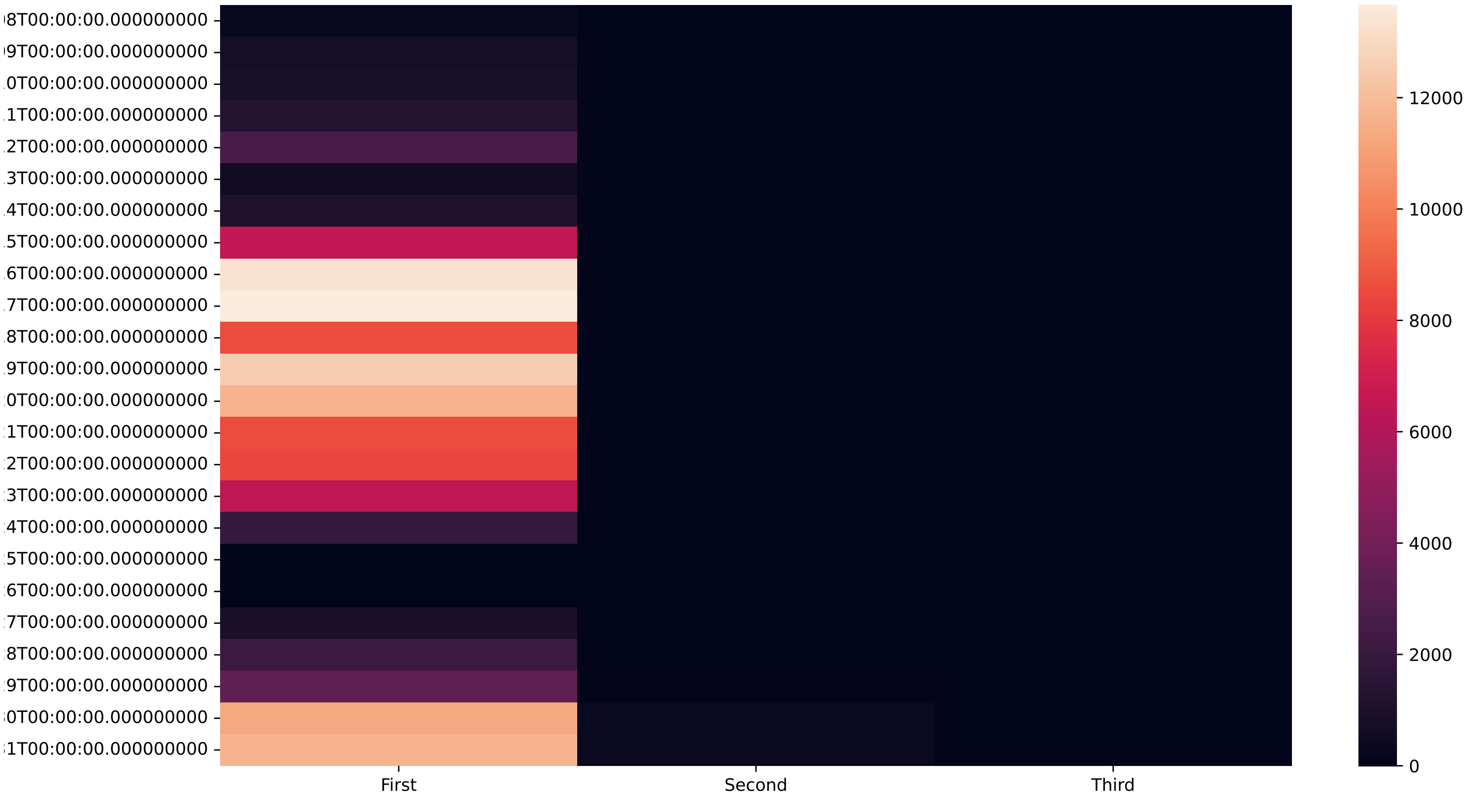
Rename columns newPeopleVaccinatedFirstDoseByVaccinationDate -> First, newPeopleVaccinatedSecondDoseByVaccinationDate -> Second, newPeopleVaccinatedThirdInjectionByVaccinationDate -> Third

Replace Na values

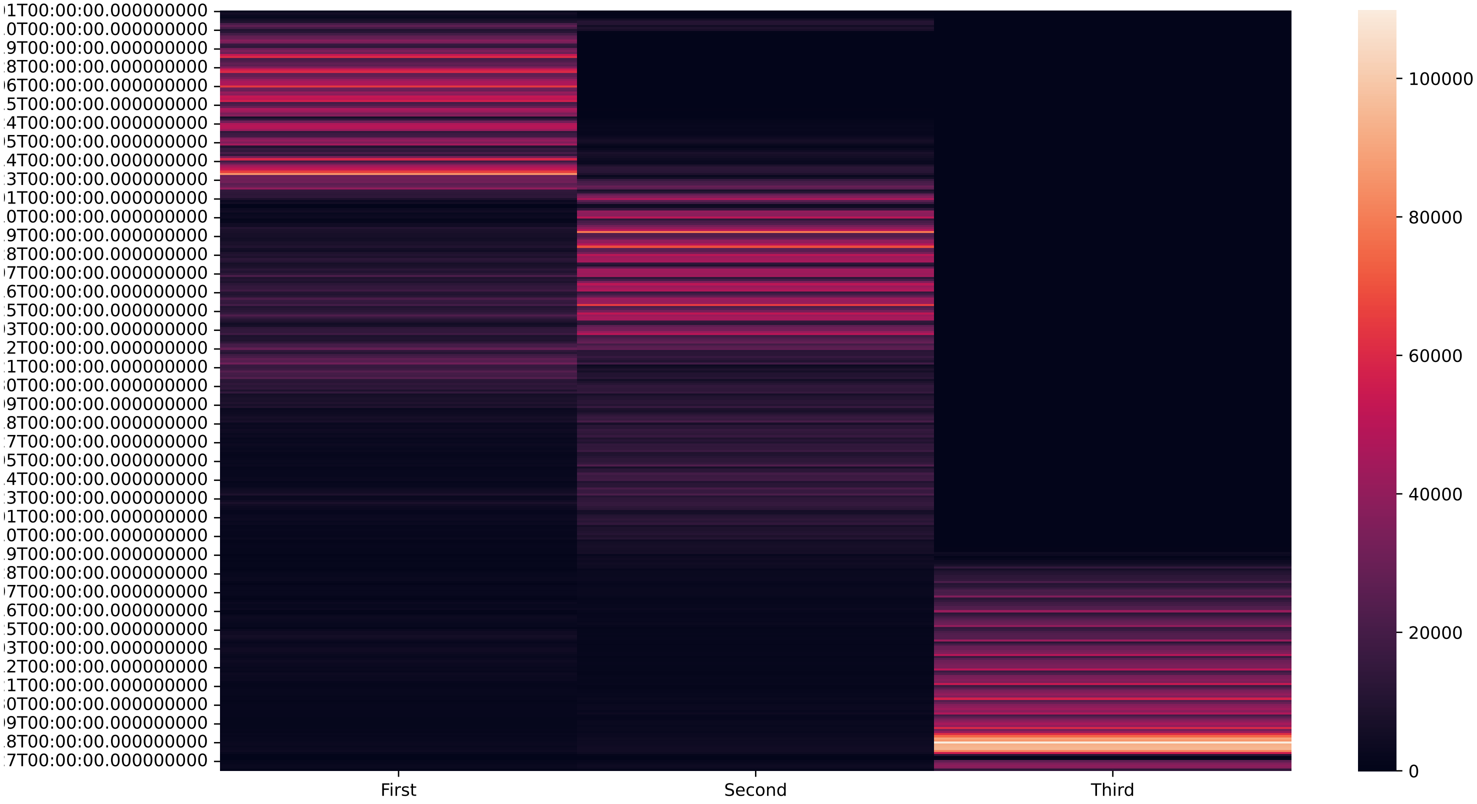
Look at the final version of the dataset

	First	Second	Third
2022-01-26	986	2520	4034

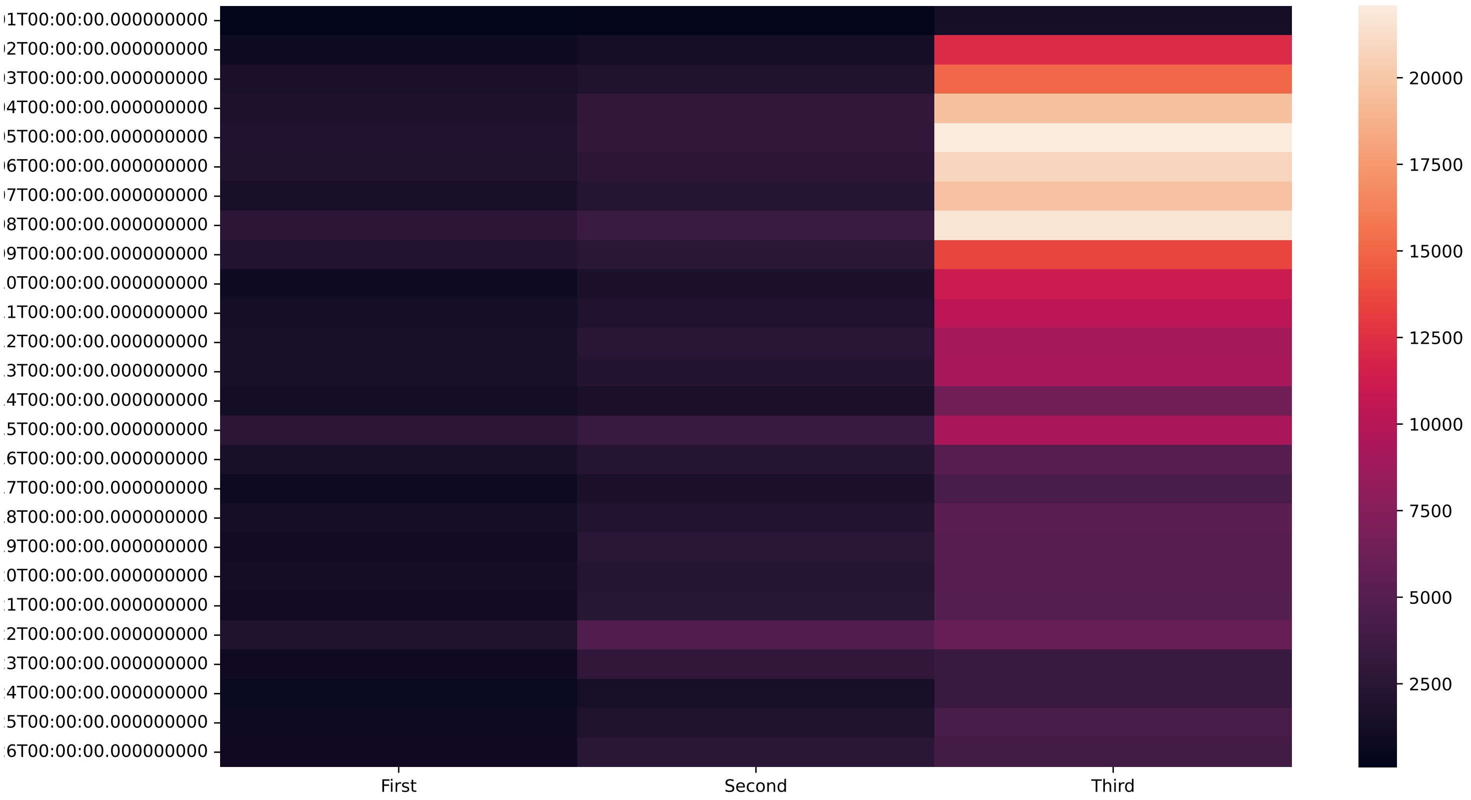
```
‘‘‘python
plt.figure(figsize=(14,8))
sns.heatmap(data=dataset.loc[[date for date in dataset.index if date.year==2020],:].sort_index())
```



```
plt.figure(figsize=(14,8))
sns.heatmap(data=dataset.loc[[date for date in dataset.index if date.year==2021],:].sort_index())
```



```
plt.figure(figsize=(14,8))
sns.heatmap(data=dataset.loc[[date for date in dataset.index if date.year==2022],:].sort_index())
```



## Step 1: Work with dates. Engineer Datatime Features

Get features: 1) Year 2) Month 3) Day etc.

```
dataset['Year'] = dataset.index.year
```

```
dataset['Month'] = dataset.index.month
```

```
dataset['Day'] = dataset.index.day
```

```
dataset['DayOfYear'] = dataset.index.dayofyear
```

```
dataset['WeekOfYear'] = dataset.index.weekofyear
```

```
## <string>:1: FutureWarning: weekofyear and week have been deprecated, please use DatetimeIndex.isocalendar().week instead, which returns
```

```
dataset['Weekday'] = dataset.index.weekday
```

```
weekdays = {0: 'Monday',
            1: 'Tuesday',
            2: 'Wednesday',
            3: 'Thursday',
            4: 'Friday',
            5: 'Saturday',
            6: 'Sunday'}




for dose in ["First", "Second", "Third"]:
    weekday_mean = dataset.groupby('Weekday')[dose].mean()
    weekday_mean = weekday_mean.rename(index=weekdays)
    plt.figure(figsize=(14,8))
    plt.title(dose)
    sns.barplot(x=weekday_mean.index, y=weekday_mean)
    plt.xticks(rotation=45)
```

```
## <Figure size 2800x1600 with 0 Axes>
## Text(0.5, 1.0, 'First')
## <AxesSubplot:title={'center':'First'}, xlabel='Weekday', ylabel='First'>
## (array([0, 1, 2, 3, 4, 5, 6]), [Text(0, 0, 'Monday'), Text(1, 0, 'Tuesday'), Text(2, 0, 'Wednesday'), Text(3, 0, 'Thursday'), Text(4, 0
## <Figure size 2800x1600 with 0 Axes>
## Text(0.5, 1.0, 'Second')
## <AxesSubplot:title={'center':'Second'}, xlabel='Weekday', ylabel='Second'>
## (array([0, 1, 2, 3, 4, 5, 6]), [Text(0, 0, 'Monday'), Text(1, 0, 'Tuesday'), Text(2, 0, 'Wednesday'), Text(3, 0, 'Thursday'), Text(4, 0
## <Figure size 2800x1600 with 0 Axes>
## Text(0.5, 1.0, 'Third')
## <AxesSubplot:title={'center':'Third'}, xlabel='Weekday', ylabel='Third'>
## (array([0, 1, 2, 3, 4, 5, 6]), [Text(0, 0, 'Monday'), Text(1, 0, 'Tuesday'), Text(2, 0, 'Wednesday'), Text(3, 0, 'Thursday'), Text(4, 0
```

```
dataset['Quarter'] = dataset.index.quarter
```

```
dataset['IsMonthStart'] = dataset.index.is_month_start
```

```
dataset['IsMonthEnd'] = dataset.index.is_month_end
```

Look at the dataset with new features

```
dataset.head()
```

```
##             First  Second   Third  ...  Quarter  IsMonthStart  IsMonthEnd
## date                               ...
## 2022-01-26    986    2520  4034.0  ...        1         False       False
## 2022-01-25    899    1845  4283.0  ...        1         False       False
## 2022-01-24    723    1445  3441.0  ...        1         False       False
## 2022-01-23   1035    3007  3439.0  ...        1         False       False
## 2022-01-22   1822    4709  5896.0  ...        1         False       False
##
## [5 rows x 12 columns]
```

```
dataset.info()
```

```
## <class 'pandas.core.frame.DataFrame'>
## DatetimeIndex: 415 entries, 2022-01-26 to 2020-12-08
## Data columns (total 12 columns):
##  #   Column        Non-Null Count  Dtype
## ---  ------        --------------  -----
##  0   First         415 non-null    int64
##  1   Second        415 non-null    int64
##  2   Third         415 non-null    float64
##  3   Year          415 non-null    int64
##  4   Month         415 non-null    int64
##  5   Day           415 non-null    int64
##  6   DayOfYear     415 non-null    int64
##  7   WeekOfYear    415 non-null    int64
##  8   Weekday       415 non-null    int64
##  9   Quarter       415 non-null    int64
##  10  IsMonthStart  415 non-null    bool
##  11  IsMonthEnd    415 non-null    bool
## dtypes: bool(2), float64(1), int64(9)
## memory usage: 52.6 KB
```

What is about Missing values? For example, there may be only one dose per day.

1.2 Step 2: Explore the dataset

1.3 Step 3: Split sets, train a Machine Learning Model and Evaluate performance

Define necessary variables

```
feature_columns = ["Year", "Month", "Day", "Weekday", "IsMonthStart", "IsMonthEnd"]
y_list = ["First", "Second"]
model_list = ["DecisionTree", "RandomForest"]
estimators_list = [100,200,300,400,500]
results = {}
val_sets = {}
```

Prepare sets

```
source_python('prepare_sets.py')
```

Train and evaluate the model

```
source_python('train_model.py')
```

Train models using parameters

```
for y in y_list:
    train_X, val_X, train_y, val_y = prepare_sets(dataset, feature_columns, y)
    val_sets[(y, "val_X")] = val_X
    val_sets[(y, "val_y")] = val_y
    for model in model_list:
        if model != "RandomForest":
            results[(y,model,"mae", 0)], results[(y,model,"predictions", 0)], results[(y,model,"model", 0)] = train_model(train_X, val_X,
        else:
            for n in estimators_list:
                results[(y,model,"mae", n)], results[(y,model,"predictions", n)], results[(y,model,"model", n)] = train_model(train_X, val
```

Compare the score with the mean value of the column that we predicted.

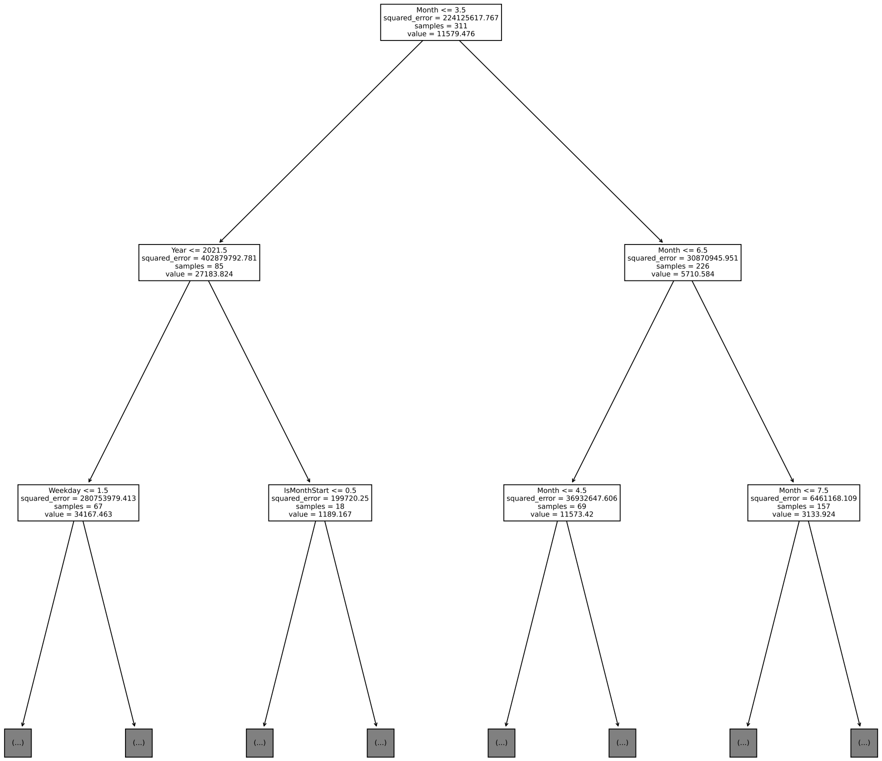
```
for res in results.keys():
    column, model, measure, treecount = res
    if measure == "mae":
        print(res, "Result: ", 1 - results[res]/dataset[column].mean())
```

```
## ('First', 'DecisionTree', 'mae', 0) Result:  0.6437087212771292
## ('First', 'RandomForest', 'mae', 100) Result:  0.751618587539483
## ('First', 'RandomForest', 'mae', 200) Result:  0.75634355378726
## ('First', 'RandomForest', 'mae', 300) Result:  0.7542008733902813
## ('First', 'RandomForest', 'mae', 400) Result:  0.7550162219639709
## ('First', 'RandomForest', 'mae', 500) Result:  0.754843479613817
## ('Second', 'DecisionTree', 'mae', 0) Result:  0.706121636338128
## ('Second', 'RandomForest', 'mae', 100) Result:  0.7715985048182643
## ('Second', 'RandomForest', 'mae', 200) Result:  0.776153470596447
## ('Second', 'RandomForest', 'mae', 300) Result:  0.7765567227604535
## ('Second', 'RandomForest', 'mae', 400) Result:  0.7751089448055306
## ('Second', 'RandomForest', 'mae', 500) Result:  0.7738541991708733
```

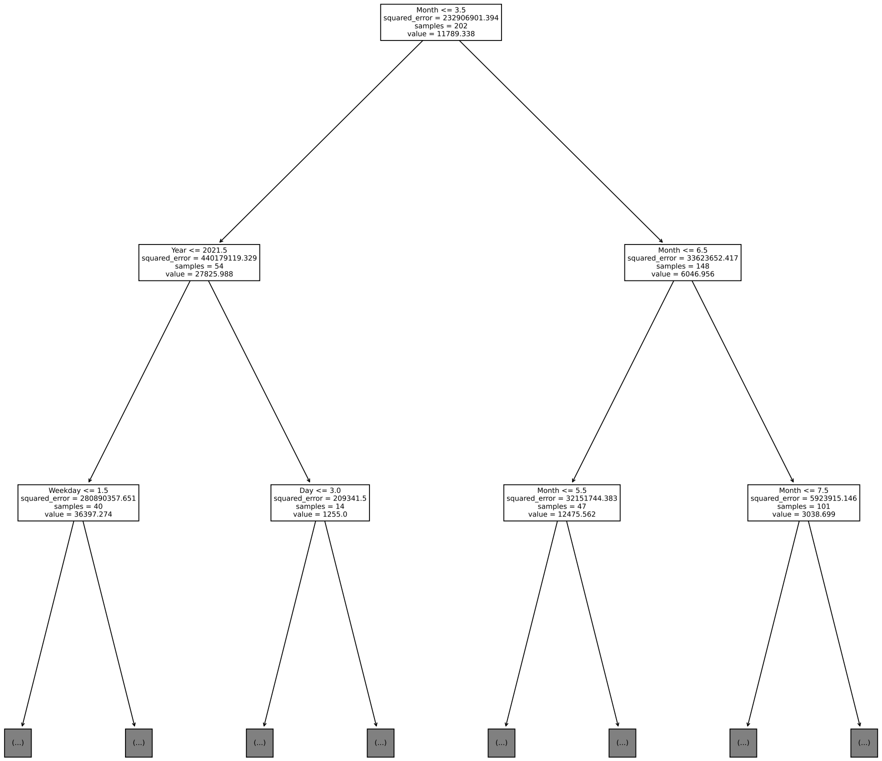
Look at the tree

```
# feature_columns is defined above
for y in y_list:
    for model in model_list:
        if model == "RandomForest":
            # one of the tree
            print('\n Dose:{}, Model:{} \n'.format(y, model))
            r = export_text(results[("First", "RandomForest", "model", 500)].estimators_[0], feature_names=feature_columns)
            print(r)
        else:
            print('\n Dose:{}, Model:{} \n'.format(y, model))
            r = export_text(results[(y, model, "model", 0)], feature_names=feature_columns)
            print(r)
```

```
plt.figure(figsize=(20,20))
# feature_columns is defined above
tree.plot_tree(results[("First", "DecisionTree", "model", 0)], max_depth=2, feature_names=feature_columns)
```

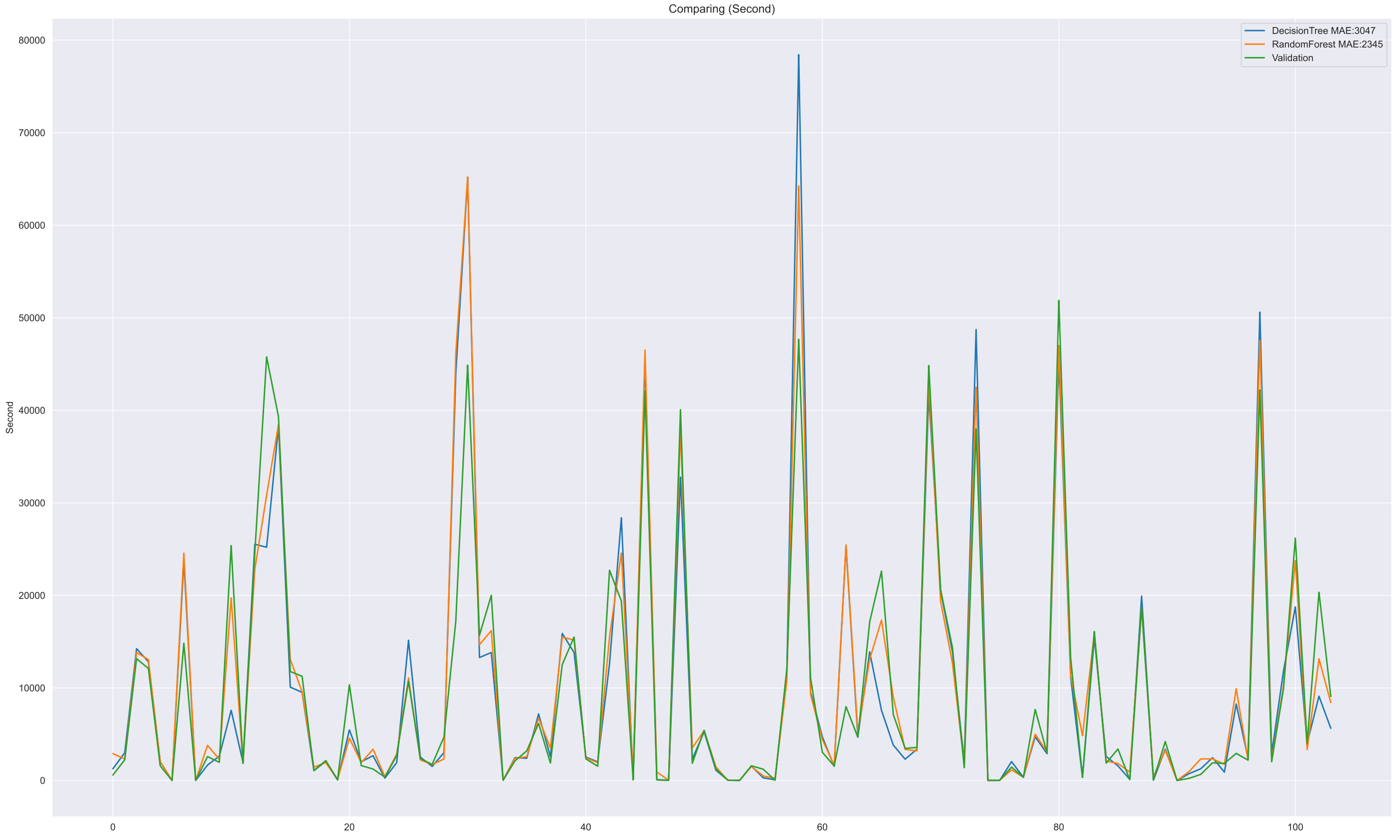

Month <= 3.5
squared_error = 224125617.767
samples = 311
value = 11579.476
Year <= 2021.5
squared_error = 402879792.781
samples = 85
value = 27183.824
Month <= 6.5
squared_error = 30870945.951
samples = 226
value = 5710.584
Weekday <= 1.5
squared_error = 280753979.413
samples = 67
value = 34167.463
IsMonthStart <= 0.5
squared_error = 199720.25
samples = 18
value = 1189.167
Month <= 4.5
squared_error = 36932647.606
samples = 69
value = 11573.42
Month <= 7.5
squared_error = 6461168.109
samples = 157
value = 3133.924
(...)
(...)
(...)
(...)
(...)
(...)
(...)
(...)

```python
plt.figure(figsize=(20,20))
# feature_columns is defined above
tree.plot_tree(results[("First", "RandomForest", "model", 500)].estimators_[0], max_depth=2, feature_names=feature_columns)
```


Month <= 3.5
squared_error = 232906901.394
samples = 202
value = 11789.338
Year <= 2021.5
squared_error = 440179119.329
samples = 54
value = 27825.988
Month <= 6.5
squared_error = 33623652.417
samples = 148
value = 6046.956
Weekday <= 1.5
squared_error = 280890357.651
samples = 40
value = 36397.274
Day <= 3.0
squared_error = 209341.5
samples = 14
value = 1255.0
Month <= 5.5
squared_error = 32151744.383
samples = 47
value = 12475.562
Month <= 7.5
squared_error = 5923915.146
samples = 101
value = 3038.699
(...)
(...)
(...)
(...)
(...)
(...)
(...)
(...)

1.4 Step 4: Plot results

```
for y in y_list:
    # set size, style and title
    plt.figure(figsize=(25,15))
    sns.set_style("darkgrid")
    plt.title('{} ({})'.format("Comparing", y))
    # plot predictions
    for model in model_list:
        if model == "RandomForest":
            sns.lineplot(data=results[(y, model, "predictions", 500)], label='{} MAE:{}'.format(model, round(results[(y, model, "mae", 500
        else:
            sns.lineplot(data=results[(y, model, "predictions", 0)], label='{} MAE:{}'.format(model, round(results[(y, model, "mae", 0)]),
    # plot validation set
    val_sets[(y, "val_y")].index=range(0,len(val_sets[(y, "val_y")]))
    sns.lineplot(data=val_sets[(y, "val_y")], label="Validation")
    # add legend
    plt.legend()
```


Comparing (Second)
Second
80000
70000
60000
50000
40000
30000
20000
10000
0
0
20
40
60
80
100
DecisionTree MAE:3047
RandomForest MAE:2345
Validation

1.5 Step 5: Improve models by changing the dataset

1.5.1 Work with features

```
dataset.info()
```

```
## <class 'pandas.core.frame.DataFrame'>
## DatetimeIndex: 415 entries, 2022-01-26 to 2020-12-08
## Data columns (total 12 columns):
##  #   Column        Non-Null Count  Dtype  
## ---  ------        --------------  -----  
##  0   First         415 non-null    int64  
##  1   Second        415 non-null    int64  
##  2   Third         415 non-null    float64
##  3   Year          415 non-null    int64  
##  4   Month         415 non-null    int64  
##  5   Day           415 non-null    int64  
##  6   DayOfYear     415 non-null    int64  
##  7   WeekOfYear    415 non-null    int64  
##  8   Weekday       415 non-null    int64  
##  9   Quarter       415 non-null    int64  
##  10  IsMonthStart  415 non-null    bool   
##  11  IsMonthEnd    415 non-null    bool   
## dtypes: bool(2), float64(1), int64(9)
## memory usage: 52.6 KB
```

Define necessary variables

```
feature_columns = ["Weekday", "Year", "DayOfYear"]
y_list = ["First", "Second"]
model_list = ["DecisionTree", "RandomForest"]
estimators_list = [100,200,300,400,500]
results = {}
val_sets = {}
```

Prepare sets and Train models

```
for y in y_list:
    train_X, val_X, train_y, val_y = prepare_sets(dataset, feature_columns, y)
    val_sets[(y, "val_X")] = val_X
    val_sets[(y, "val_y")] = val_y
    for model in model_list:
        if model != "RandomForest":
            results[(y,model,"mae", 0)], results[(y,model,"predictions", 0)], results[(y,model,"model", 0)] = train_model(train_X, val_X,
        else:
            for n in estimators_list:
                results[(y,model,"mae", n)], results[(y,model,"predictions", n)], results[(y,model,"model", n)] = train_model(train_X, val
```

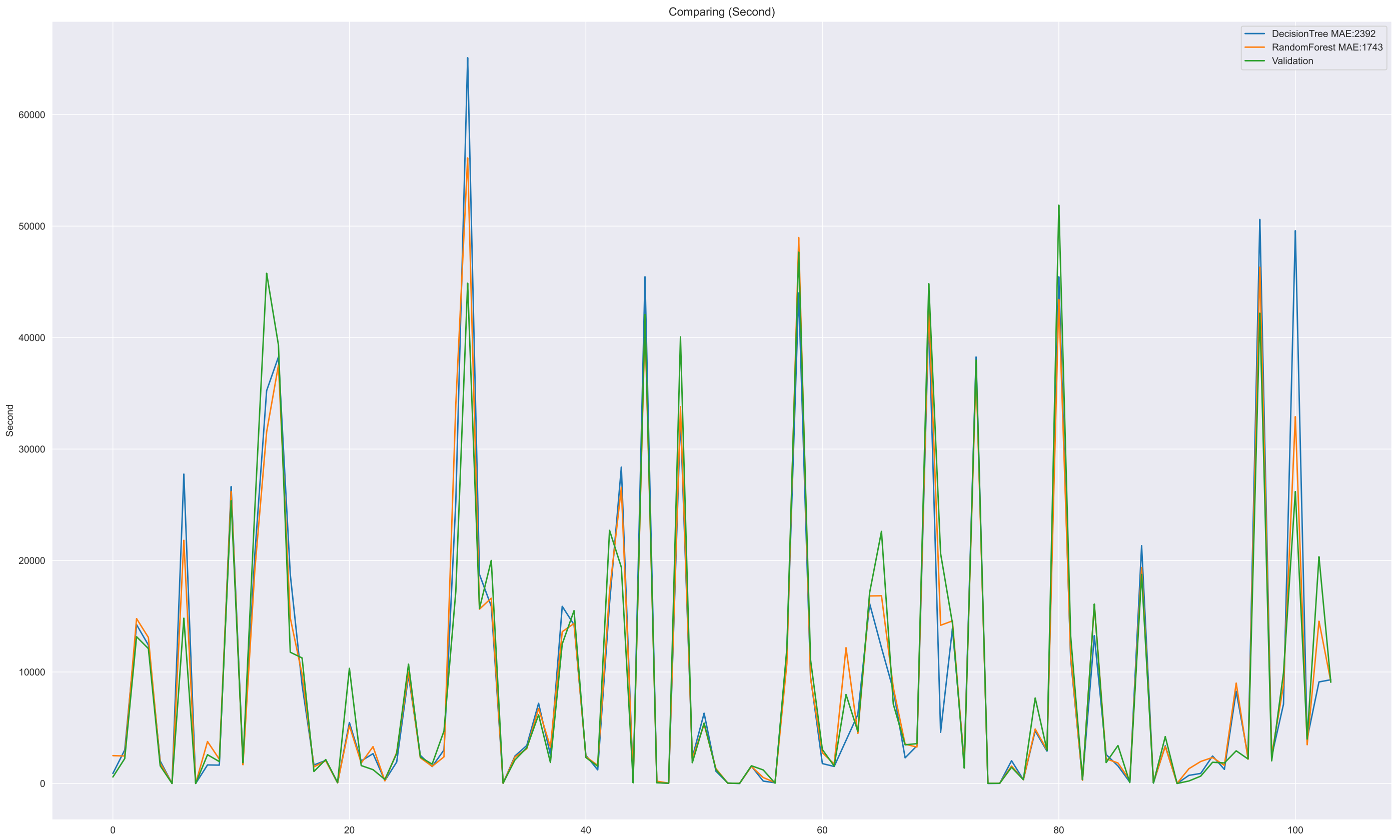
Compare the score with the mean value of the column that we predicted.

```
for res in results.keys():
    column, model, measure, treecount = res
    if measure == "mae":
        print(res, "Result: ", 1 - results[res]/dataset[column].mean())
```

```
## ('First', 'DecisionTree', 'mae', 0) Result:  0.724863032602476‌8
## ('First', 'RandomForest', 'mae', 100) Result:  0.7662734316499331
## ('First', 'RandomForest', 'mae', 200) Result:  0.7831927047569076
## ('First', 'RandomForest', 'mae', 300) Result:  0.782419191565517‌1
## ('First', 'RandomForest', 'mae', 400) Result:  0.783001374743421
## ('First', 'RandomForest', 'mae', 500) Result:  0.7837038702898657
## ('Second', 'DecisionTree', 'mae', 0) Result:  0.7692636418171874
## ('Second', 'RandomForest', 'mae', 100) Result:  0.8341621784974336
## ('Second', 'RandomForest', 'mae', 200) Result:  0.8361849996061279
## ('Second', 'RandomForest', 'mae', 300) Result:  0.8365889071806888
## ('Second', 'RandomForest', 'mae', 400) Result:  0.8333133970971072
## ('Second', 'RandomForest', 'mae', 500) Result:  0.8318561653086721
```

A combination of the following features give us the best result: * Weekday, * Year, * DayOfYear

```
for y in y_list:
    # set size, style and title
    plt.figure(figsize=(25,15))
    sns.set_style("darkgrid")
    plt.title('{} ({})'.format("Comparing", y))
    # plot predictions
    for model in model_list:
        if model == "RandomForest":
            sns.lineplot(data=results[(y, model, "predictions", 500)], label='{} MAE:{}'.format(model, round(results[(y, model, "mae", 500
        else:
            sns.lineplot(data=results[(y, model, "predictions", 0)], label='{} MAE:{}'.format(model, round(results[(y, model, "mae", 0)]),
    # plot validation set
    val_sets[(y, "val_y")].index=range(0,len(val_sets[(y, "val_y")]))
    sns.lineplot(data=val_sets[(y, "val_y")], label="Validation")
    # add legend
    plt.legend()
```



1.5.2 Work with missing values

```
dataset.index.min()
```

```
## Timestamp('2020-12-08 00:00:00')
```

```
dataset.index.max()
```

```
## Timestamp('2022-01-26 00:00:00')
```

```
dates = pd.date_range(dataset.index.min(),dataset.index.max(),freq='d')
```

```
dates
```

```
## DatetimeIndex(['2020-12-08', '2020-12-09', '2020-12-10', '2020-12-11',
##                '2020-12-12', '2020-12-13', '2020-12-14', '2020-12-15',
##                '2020-12-16', '2020-12-17',
##                ...
##                '2022-01-17', '2022-01-18', '2022-01-19', '2022-01-20',
##                '2022-01-21', '2022-01-22', '2022-01-23', '2022-01-24',
##                '2022-01-25', '2022-01-26'],
##               dtype='datetime64[ns]', length=415, freq='D')
```

```
len(dataset.index)
```

```
## 415
```

```
len(dates)
```

```
## 415
```

There are no missing dates.