Regression analysis for air quality prediction

1 Introduction

During the modern age of transportation and climate change there has been a lot of talk of air quality in big cities due to congestion and how it is way worse than in the countryside. Air quality control and prediction will be important in the future due to the Earth's climate change. Machine learning is going to be important in the predictions and will help analysts to make meaningful decisions on how to change the traffic structure to be more efficient and make the air quality better in Helsinki.

In this report we will be formulating a solution to the aforementioned regressional problem of predicting the average air quality index of Helsinki with the help of weather and traffic data. The report is split into four main parts: problem formulation where we will be discussing how we approach our problem, methods in which we will be discussing our two chosen methods for training the model, results where we discuss the results of the methods and their effectiveness and conclusions to summarize our findings.

2 Problem formulation

The application of our machine learning project is to predict the air quality index of Helsinki using various weather readings and traffic numbers from multiple years as our data. The datasets that are used in this project have been sourced from three different sources Digitraffic (Fintraffic), Helsingin seudun ympäristöpalvelut (HSY) and Ilmatieteenlaitos (Ilmatieteenlaitos), so we have to merge the three different datasets for this project. Some of the data has been collected from various locations across Helsinki so for these, we are going to use their averages. All of our data has been collected hourly which means that we have a lot of data points crossing from several years and all of the used data is also numerical and continuous.

From the weather data we have taken the multiple features such as precipitation(amount and intensity), air temperature and wind speed. The weather data has been collected from the Kumpula measuring station. We are also using average traffic numbers from various measuring stations around Helsinki as one of our features and this data is from Digitraffic. Our final data is HSY:s air quality index which has also been collected from various locations around Helsinki and the average air quality index will be our label. Date is also a feature in our data, due to the time of year having an effect on the air quality. Thus one data point represents weather data and average traffic data from a specific hour with the average air quality index as the label consequently making this a supervised learning problem.

3 Methods

3.1 Dataset

As we will be using three different datasets for this project, we have to merge them together on coinciding dates and times. Since the datasets are from different sources, quite a bit of preprocessing was needed to merge them together. The preprocessing and feature engineering process involved trimming some of the datasets, reorganizing them to be of the same form, reformatting the dates and times to be uniform so that we can easily merge them and taking averages of the traffic and air quality data so that we can use those as a feature and our label. The final dataset consists of our label, average air quality index of Helsinki, and our features such as average traffic numbers, various weather data and dates with measurements every hour from a four year period (2019-2022) meaning we have around 35064 data points total.

Features were selected based on their applicability and relevance to the problem and their compatibility with machine learning algorithms. Any non numerical features or features which could not be easily converted into numerical data were excluded from our final data. Some weather features such as ”cloud amount” and ”wind direction” were ultimately also excluded in the selection process since they showed no significant correlation with the label.

3.2 Linear regression

As a baseline, we will be using linear regression to predict the air quality index with the help of the features. This method was selected for its easy applicability to the problem and because it is sensible enough to assume linear relationships between the features and the air quality index, also some linear relationships could be seen when plotting some of the features with the average air quality index. A linear approach to this problem most likely loses some meaningful relationship between the features and the label, for example the relationship between the day of year and hour of day with the air quality index is almost certainly not linear, but it is still a good starting point for this project. As for the loss function, we will be using mean squared error as it is the standard for linear regression problems and Python libraries use it.

3.3 Random forest regression

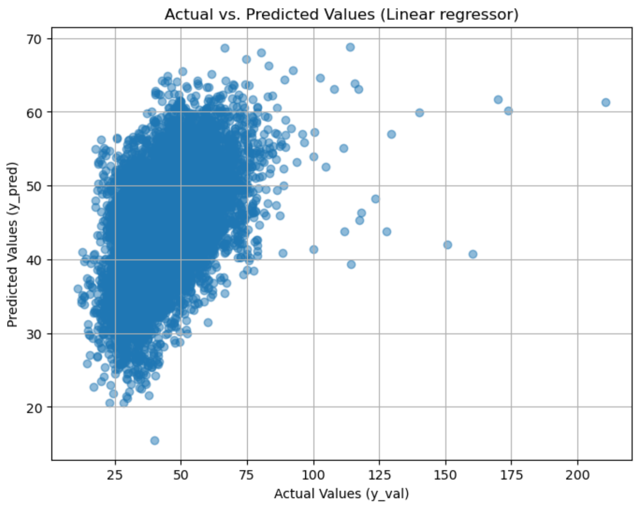
Since a lot of the features such as ”day of year” exhibited non-linear relationships with the target variable, a non-linear model is needed to properly evaluate the problem. Thus, we chose the Random forest regression model to capture these more complex non-linear relationships between the features and the label. Random forest was ultimately chosen over different options such as Decision tree and neural networks, for its applicability to the problem, its effectiveness for machine learning problems with large and complex datasets (Singh) and because it showed most promise for predicting the labels when fitted on the data with default hyperparameters. As a loss function we used mean squared error since it is standard practice to use it with Random forests, much like with linear regression and also because Python libraries use it as the default loss function for Random forest regression.

3.4 Splitting the data

We will be splitting the data into training, validation and testing sets with 50% going into training, 25% into validation and 25% into testing. The reason for this decision was that it is roughly the standard for splitting a dataset for machine learning problems (Acharya) and also because we want to split the data into year long periods for the algorithms to use. As we have data from four years, splitting the data like this sensibly gives us two years of training data, one year of validation data and one year of testing data. We would have used a more standard 60/20/20 split if data from five years was available, but hourly traffic data could not be downloaded from a five year period.

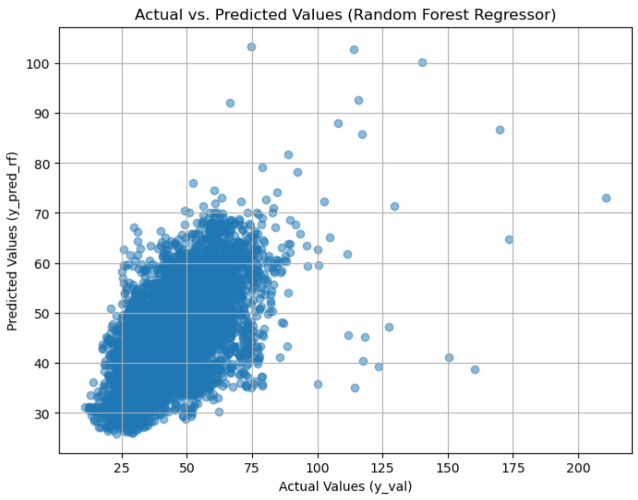
4 Results

4.1 Linear regression results



After training the linear regression model we got a training error of 95,12 and a validation error of 137,81 using mean squared error. Looking at the above plot of the actual values versus the predicted ones, we can see that the linear model is reluctant to predict very large values for the air quality index, since it is trying to minimize the sum of squared differences. Nevertheless, most of the data points are predicted within reason and only some large outliers are significantly wrongly predicted.

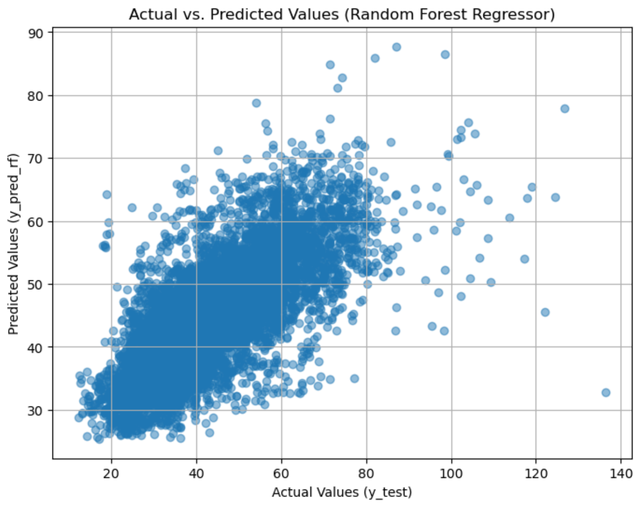
4.2 Random forest model



Unlike the linear regression model, the Random forest model has a lot of different hyperparameters which can be tuned (scikit-learn). Determining the best hyperparameters for the model is crucial so that we can predict the labels as well as possible. A lot of these hyperparameters proved to be best when left as the default parameters, but tuning the max_depth parameter proved to be beneficial. The max_depth parameter determines the maximum depth of the trees, which can be tuned to control overfitting. Even with a very large dataset, overfitting can sometimes be a problem, so we calculated the validation errors with depths from 1 to 50. Close to 50 the validation error started to converge towards the default hyperparameter error value, but a depth of 12 resulted in the best validation error of 114,47 with a significantly better training error of 20,19, which might indicate overfitting. Nevertheless, this depth was chosen for the final model.

4.3 Comparing the models

Comparing the two models on the acquired training and validations errors, the Random forest model has lower errors meaning it fares better in predicting the air quality index. The linear model was expected to be flawed for this problem especially because it cannot capture more complex relationships between the features and the label. Comparing the above images, we can also see that while both models are adequate in predicting the most common labels, the Random forest fares slightly better while also doing better in predicting the outliers.



Thus, for the final model a Random forest with maximum depth of 12 and mean squared error as the loss function is chosen. With this model, evaluating on the testing set, which is from the year 2022, we get a testing error of 96,77. In the above graph we can see the predicted final values versus the actual values of the testing set. The aforementioned error is significantly better than the previous validation error, which is due to the fact that the testing set has fewer large outliers.

5 Conclusion

As discussed above in the results section the better model was achieved with the Random forest model where the validation and training error were smaller, but our model capability to predict the air quality in Helsinki is still a bit flawed as can bee seen from the figure above and from the errors. The square root of our testing error is around 9,8 so our model's prediction differs around this amount from the real values of the index as can bee seen from the figure above. This value is relatively accurate considering the scope of this project but could still be improved upon.

To make our model better we should have more data points in our training set as the training error is way smaller then our models validation error. This is also backed up by how we had trouble predicting the outlier cases. These would also be easier to predict if the training data set was from a longer time period, thus giving us more data and outliers. Hopefully in the future we could get more data than just the last four years that we were limited by the provider of the datasets, as our data set is only based on the last four years of weather and traffic data. The validity of our model could be also affected by the Covid pandemic as it affected the amount of traffic to Helsinki for a couple of years as our validation and some of the training data being from this period and our testing data from after the pandemic.

6 References

Fintraffic. “LAM-tilastohaku.” *Digitraffic*, https://tie.digitraffic.fi/ui/tms/history/. Accessed 21. September 2023.

HSY. “Pääkaupunkiseudun ja muun Uudenmaan ilmanlaatuindeksit.” *HSY*, https://www.hsy.fi/ymparistotieto/avoindata/avoin-data---sivut/paakaupunkiseudun-ja-muun-uudenmaan-ilmanlaatuindeksit/. Accessed 21. September 2023.

Ilmatieteenlaitos. “Download observations.” *Ilmatieteenlaitos*, https://en.ilmatieteenlaitos.fi/download-observations. Accessed 21. September 2023.

Singh, Ambarish. “Random Forests: The Go-To Algorithm for Complex Data Sets.” *Medium*, 2023, https://medium.com/@Ambarish_224/random-forests-the-go-to-algorithm-for-complex-data-sets-aacbfaef57a2. Accessed 9. October 2023.

Acharya, Akruti. “Training, Validation, Test Split for Machine Learning Datasets.” *encord*, 2023, https://encord.com/blog/train-val-test-split/. Accessed 11. October 2023.

scikit-learn. “sklearn.ensemble.RandomForestRegressor.” *scikit-learn*, https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html. Accessed 9. October 2023.

7 Appendix

Air quality index project

October 11, 2023

[17]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.ensemble import RandomForestRegressor
```

[2]:
```
# Reading the datasets (weather and air quality yearly data had to be␣
 ↪downloaded separately)
weather19 = pd.read_csv('data/weather_2019.csv')
weather20 = pd.read_csv('data/weather_2020.csv')
weather21 = pd.read_csv('data/weather_2021.csv')
weather22 = pd.read_csv('data/weather_2022.csv')
air_quality19 = pd.read_excel('data/air_quality_2019.xlsx')
air_quality20 = pd.read_excel('data/air_quality_2020.xlsx')
air_quality21 = pd.read_excel('data/air_quality_2021.xlsx')
air_quality22 = pd.read_excel('data/air_quality_2022.xlsx')
traffic = pd.read_csv("data/traffic_2019-2022.csv", sep=";")
```

[3]:
```
# First we preprocess the "traffic" dataframe
# Using melt to reshape the dataframe so that hours are on one column with the␣
 ↪corresponding amounts
traffic = traffic.melt(id_vars=['sijainti', 'pvm'], var_name='datetime',␣
 ↪value_name='amount')

# Adjust the hour format and convert to datetime
traffic['datetime'] = traffic['datetime'].str.replace(r'^(\d{2})_(\d{2})$',␣
 ↪r'\1', regex=True)
traffic['datetime'] = pd.to_datetime(traffic['pvm'].astype(str) +␣
 ↪traffic['datetime'], format='%Y%m%d%H')

# Dropping the original 'pvm' column
traffic.drop(columns=['pvm'], inplace=True)

# Pivoting the measuring stations to columns
traffic = traffic.pivot_table(index=['datetime'], columns='sijainti',␣
 ↪values='amount', aggfunc='first')
```

```
# Resetting index so that datetime becomes a column
traffic.reset_index(inplace=True)
```

[4]:
```
weather = pd.concat([weather19, weather20, weather21, weather22], ignore_index␣
↪=True)

# Next the weather dataframe
# First we conver columns "Year", "m", "d" and "Time" to one column "datetime"
weather['datetime'] = pd.to_datetime(weather['Year'].astype(str) + ' ' +
                                     weather['m'].astype(str) + ' ' +
                                     weather['d'].astype(str) + ' ' +
                                     weather['Time'], format='%Y %m %d %H:00')

# Dropping the old columns
weather.drop(columns=['Year', 'm', 'd', 'Time'], inplace=True)
```

[5]:
```
# Finally we process the air qualities
air_quality = pd.concat([air_quality19, air_quality20, air_quality21,␣
↪air_quality22], ignore_index = True)
# This dataset had 01:00 to 24:00, so we need to change this to the same format␣
↪as the other datasets (00:00 to 23:00)
air_quality['Aikaleima envista'] = air_quality['Aikaleima envista'].str.
↪replace('24:00', '00:00')
# Next we create the datetime column. There are multiple different formats so␣
↪we need to handle each one
air_quality['Aikaleima envista'] = air_quality['Aikaleima envista'].str.strip()
air_quality['datetime'] = pd.to_datetime(air_quality['Aikaleima envista'],␣
↪format='%d/%m/%Y %H:%M', errors="coerce")
air_quality['datetime'] = air_quality['datetime'].fillna(pd.
↪to_datetime(air_quality['Aikaleima envista'], format='%d.%m.%Y %H:%M',␣
↪errors="coerce"))
air_quality['datetime'] = air_quality['datetime'].fillna(pd.
↪to_datetime(air_quality['Aikaleima envista'], format='%H:%M %d.%m.%Y',␣
↪errors="coerce"))

# Subtract one hour from each datetime value and if hours are 0, add a day (we␣
↪need to do this because substracting an hour puts us on the previous day in␣
↪this case)
air_quality['datetime'] = air_quality.apply(lambda row: row['datetime'] - pd.
↪Timedelta(hours=1) if row['datetime'].hour != 0 else row['datetime']-pd.
↪Timedelta(hours=1)+pd.Timedelta(days=1), axis=1)

# Drop old time column
air_quality = air_quality.drop(columns="Aikaleima envista")
```

```
# Next we find the average air_quality
air_quality = air_quality.replace("NoData", np.nan)
# Convert columns except datetime to float
for column in air_quality.columns:
    if column != 'datetime':
        air_quality[column] = air_quality[column].astype(float)
mean_columns = air_quality.drop(columns=['datetime']) # Columns to calculate␣
 ↪the mean from

# Creating average AQ column
air_quality['average AQ'] = mean_columns.mean(axis=1)

# Calculating average traffic with the same principle
mean_columns = traffic.drop(columns=['datetime'])
traffic['average traffic'] = mean_columns.mean(axis=1)

# Taking only the averages and datetime from both
traffic = traffic[['datetime', 'average traffic']]
air_quality = air_quality[['datetime', 'average AQ']]
```

[6]:
```
# Merging the dataframes
traffic_and_weather = pd.merge(traffic, weather, on='datetime')
merged = pd.merge(air_quality, traffic_and_weather, on='datetime')
# Adding day of year and hour columns
merged['day of year'] = merged['datetime'].dt.dayofyear
merged['hour'] = merged['datetime'].dt.hour
# Dropping unneccessary columns
merged = merged.drop(columns=["Time zone", "Cloud amount (1/8)", "Wind␣
 ↪direction (deg)"])
# Replacing - with NaN and converting data types to float
merged = merged.replace('-',np.nan)
for column in merged.columns:
    if column != 'datetime':
        merged[column] = merged[column].astype(float)
# Now we have one dataframe with one datetime, average air quality, average␣
 ↪traffic and weather
merged = merged.dropna()
```

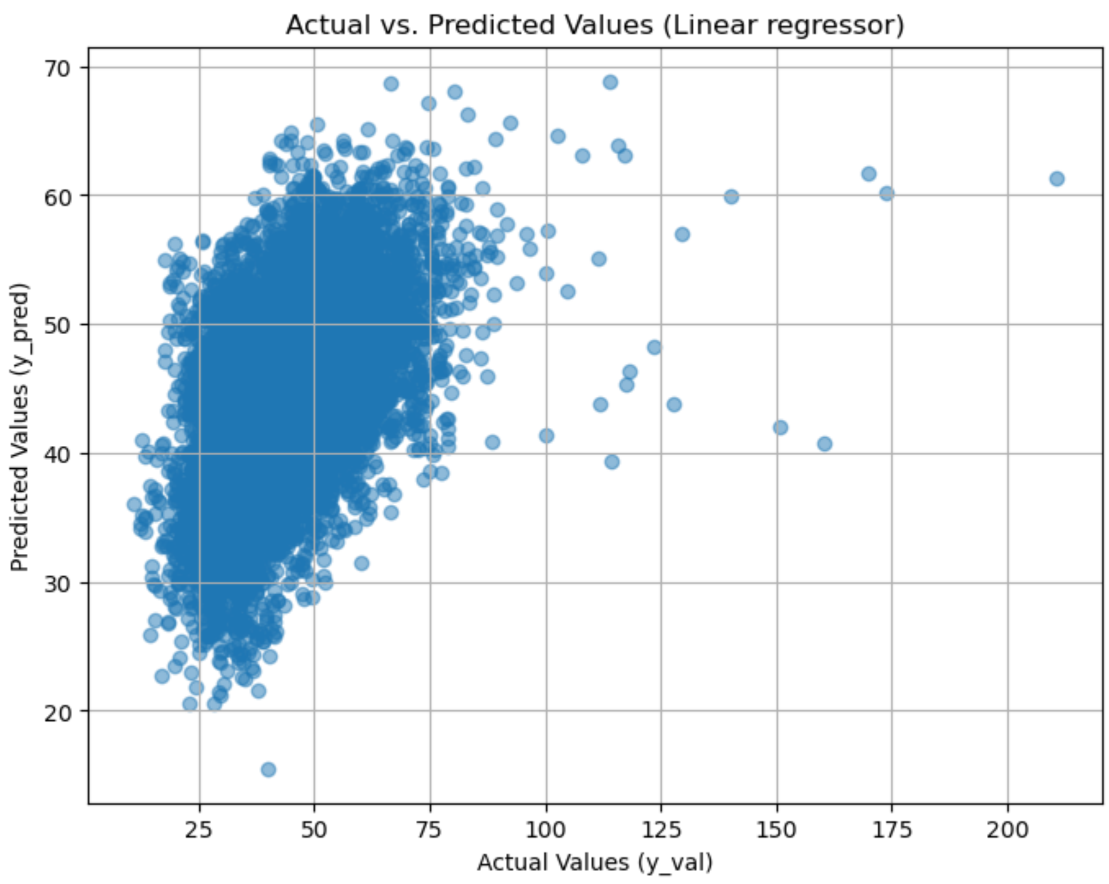
[7]:
```
# We want to use the first two years for training, one year for validation and␣
 ↪the last year for testing
training_start = '2019-1-1 00:00:00'
training_end = '2020-12-31 23:00:00'
validation_start = '2021-1-1 00:00:00'
validation_end = '2021-12-31 23:00:00'
testing_start = '2022-1-1 00:00:00'
testing_end = '2022-12-31 23:00:00'
```

```python
# Filter the data based on date ranges
train_data = merged[(merged['datetime'] >= training_start) &␣
  ↪(merged['datetime'] <= training_end)]
validation_data = merged[(merged['datetime'] >= validation_start) &␣
  ↪(merged['datetime'] <= validation_end)]
test_data = merged[(merged['datetime'] >= testing_start) & (merged['datetime']␣
  ↪<= testing_end)]

# Getting labels and features
y_train = train_data['average AQ']
y_val = validation_data['average AQ']
y_test = test_data['average AQ']

X_train = train_data.drop(columns=["datetime", "average AQ"])
X_val = validation_data.drop(columns=["datetime", "average AQ"])
X_test = test_data.drop(columns=["datetime", "average AQ"])
```

[8]:
```python
model = LinearRegression()

# Fit the model on the training data
model.fit(X_train, y_train)

# Make predictions on the testing data
y_pred = model.predict(X_train)
training_error = mean_squared_error(y_train, y_pred)
y_pred_val = model.predict(X_val)
validation_error = mean_squared_error(y_val, y_pred_val)

# Finding mean squared error of the model

print(f"Training error: {training_error:.2f}")
print(f"Validation error: {validation_error:.2f}")
```

```
Training error: 95.12
Validation error: 137.81
```

[15]:
```python
# Making a scatter plot for the linear regression model with actual values on x␣
  ↪axis and predicted on y axis
plt.figure(figsize=(8, 6))
plt.scatter(y_val, y_pred_val, alpha=0.5)
plt.xlabel('Actual Values (y_val)')
plt.ylabel('Predicted Values (y_pred)')
plt.title('Actual vs. Predicted Values (Linear regressor)')
plt.grid(True)
plt.show()
```



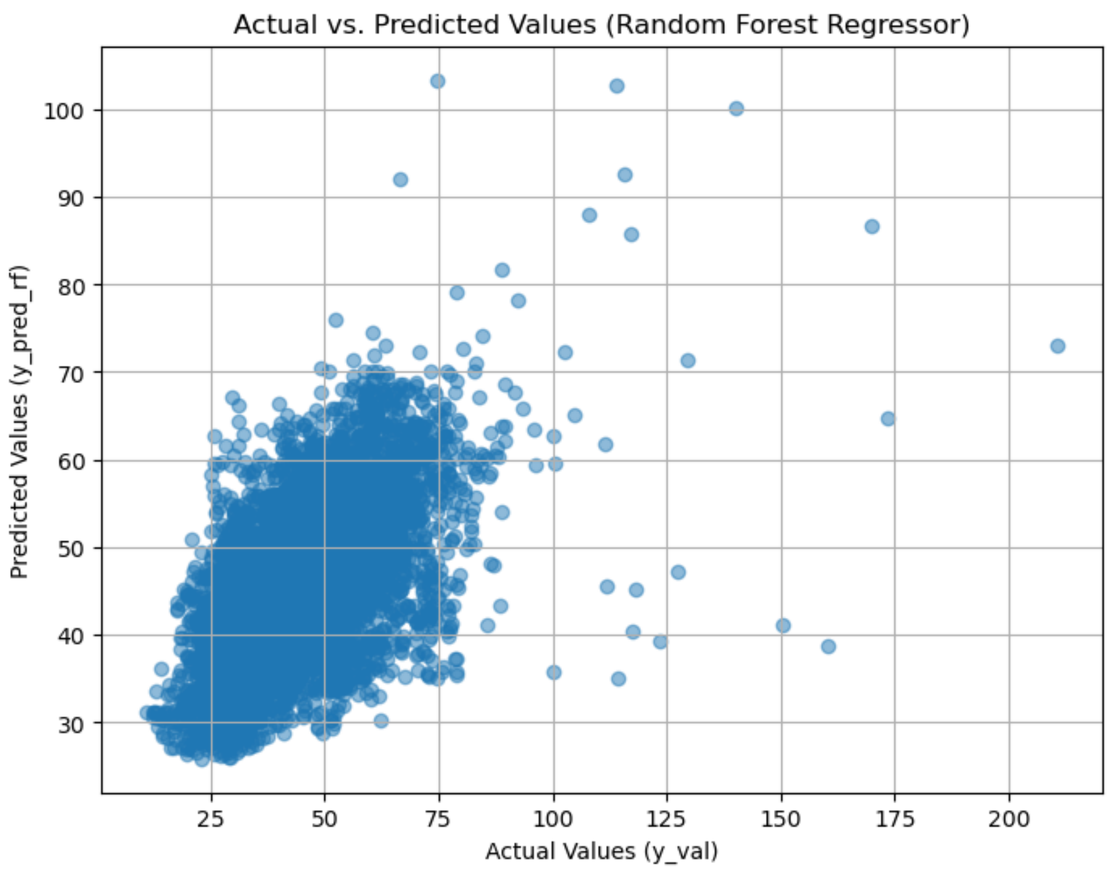
[11]:
```
training_errors = []
validation_errors = []
depths = []
# Finding the optimal depth with the help of a for-loop:
for depth in range(1,51):
    clf = RandomForestRegressor(max_depth=depth,random_state=0)

    # Fit the model to the training data
    clf.fit(X_train, y_train)

    # Predict using the trained model and get errors
    y_pred_rf = clf.predict(X_train)
    training_error = mean_squared_error(y_train, y_pred_rf)
    y_pred_rf_val = clf.predict(X_val)
    validation_error = mean_squared_error(y_val, y_pred_rf_val)

    # Append errors
    training_errors.append(training_error)
    validation_errors.append(validation_error)
    depths.append(depth)
```

[12]:
```
# Finding the best depth and printing errors
best_index = validation_errors.index(min(validation_errors))
print(f"Training error: {training_errors[best_index]:.2f}")
print(f"Validation error: {validation_errors[best_index]:.2f}")
print(f"Depth: {depths[best_index]}")
```

```
Training error: 20.19
Validation error: 114.47
Depth: 12
```

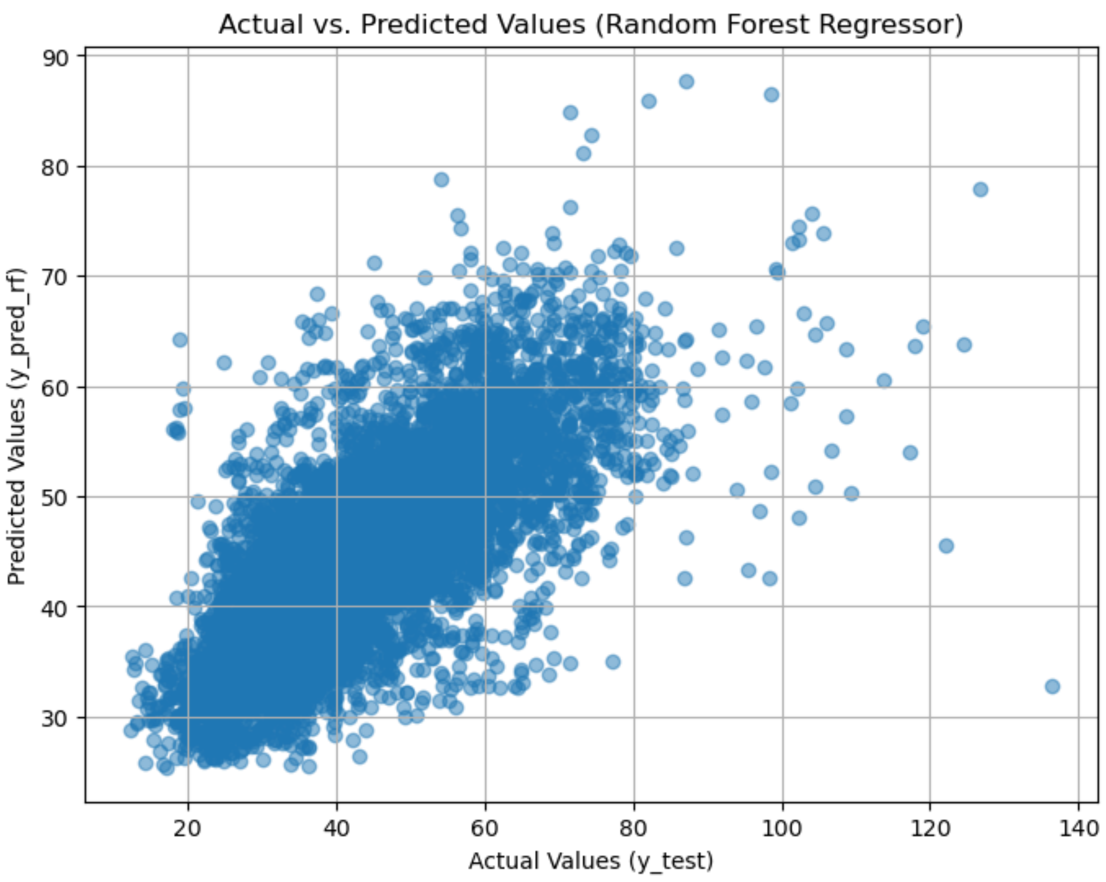
[13]:
```
# Using the depth with the model
clf = RandomForestRegressor(max_depth=12,random_state=0)

# Fit the model to the training data
clf.fit(X_train, y_train)

# Predict using the trained model
y_pred_rf = clf.predict(X_train)
y_pred_rf_val = clf.predict(X_val)

# Calculate Mean Squared Errors
training_error = mean_squared_error(y_train, y_pred_rf)
validation_error = mean_squared_error(y_val, y_pred_rf_val)
print(f"Training error: {training_error:.2f}")
print(f"Validation error: {validation_error:.2f}")

# Plot actual vs. predicted values
plt.figure(figsize=(8, 6))
plt.scatter(y_val, y_pred_rf_val, alpha=0.5)
plt.xlabel('Actual Values (y_val)')
plt.ylabel('Predicted Values (y_pred_rf)')
plt.title('Actual vs. Predicted Values (Random Forest Regressor)')
plt.grid(True)
plt.show()
```

```
Training error: 20.19
Validation error: 114.47
```



```
[16]: # Predict the test set values and calculate the testing error
      y_pred_rf_test = clf.predict(X_test)
      testing_error = mean_squared_error(y_test, y_pred_rf_test)
      print(f"Testing error: {testing_error:.2f}")
      # Plot actual vs. predicted values
      plt.figure(figsize=(8, 6))
      plt.scatter(y_test, y_pred_rf_test, alpha=0.5)
      plt.xlabel('Actual Values (y_test)')
      plt.ylabel('Predicted Values (y_pred_rf)')
      plt.title('Actual vs. Predicted Values (Random Forest Regressor)')
      plt.grid(True)
      plt.show()
```

```
Testing error: 96.77
```


Actual vs. Predicted Values (Random Forest Regressor)
Predicted Values (y_pred_rf)
Actual Values (y_test)
90
80
70
60
50
40
30
20
40
60
80
100
120
140

```
[ ]:
```