Aalto University

CS-C3240 - Machine Learning

Predicting Invariant Mass of Dielectron Events

with Random Forest

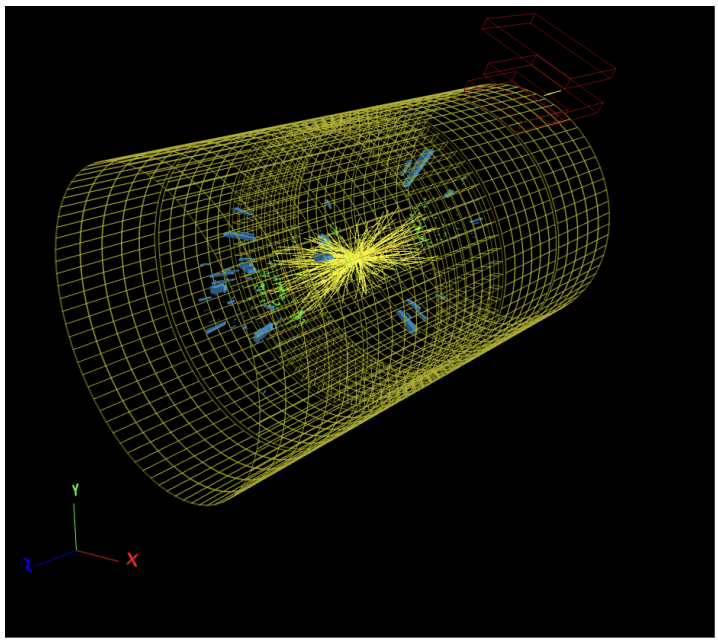


Figure 1: McCauley, Thomas; (2014). Events with two electrons from 2010. CERN Open Data Portal. DOI:10.7483/OPENDATA.CMS.PCSW.AHVG

1. Introduction 1
1.1 Background 1
1.2 Structure 1
2. Problem formulation 1
3. Methods 2
3.1 The Data 2
3.2 Choice of Model 2
3.3 Model Validation 3
4. Results 3
5. Conclusion 4
6. References 5
7. Appendix 5

1. Introduction

1.1 Background

Colliding particles, in our case, electrons, are a crucial step in comprehending the behavior of subatomic particles. Since everything is composed of subatomic particles, understanding their behavior aids in addressing numerous physics questions. An essential aspect of dielectron collisions involves predicting the invariant mass after the collision, enabling us to identify and study new particles. For instance, the discovery of the Higgs boson resulted from the collision of subatomic particles, subsequently unraveling the mystery of why some particles possess mass while others do not.

1.2 Structure

In this report, we will formulate the problem as a machine learning task. Subsequently, we will define the data points, features, and our label, and elaborate on the data source within the problem formulation section. After formalizing the problem, we will proceed to data preparation and introduce a machine learning models aimed at predicting the invariant mass of electrons. We will then discuss the results obtained from both methods and explain our choice between the two. In the conclusion section, we will summarize the report's key findings and observations, and we will conclude the report with references and appendices containing the code used for data preprocessing and model training.

2. Problem formulation

The data points contain information about the collision of two electrons. By carefully analyzing the dataset we can predict the invariant mass of the two electrons. Predicting the mass of electrons after collision helps us understand more about the behavior of the fundamental particle. It helps us with testing our theoretical models, and we can detect anomalies in the collision which may lead us to finding new particles.

Our dataset contains the following features. The run number indicates the event's specific run number. In one run there are arbitrary amounts of events. One event contains all the information about one collision. E1 and E2 are the total energy of the electrons in GeV for electrons 1 and 2, px1, py1, pz1, px2, py2 and pz2 are the components of the momentum of the electron in directions x, y and z. The momentum is measured in GeV. Pt1 and Pt2 are the

inverse of the total momentum of the electrons, Eta1 and eta2 are the pseudorapidity of the electron and it describes the motion of the particles and it is dimensionless, phi1 and phi2 are the angle between the electrons in radians. Q1 and Q2 are the charge of the electron, 1 meaning positive charge and –1 negative charge. M is the invariant mass of the two electrons. As we can see the data is a mix of categorical and continuous variables.

The dataset is from CERN own research where they collided electrons in the Large Hadron Collider. The CMS collaboration which stands for The Compact Muon Solenoid has approved the release of 100k dielectron events in the invariant mass range 2-110 GeV for use in education. It contains the subset of the total event information. [1]

In this case we are going to use supervised learning since in our data the value of M is known for the given features, with this we can use regression models to build a predictive model that we are going to explore in the next paragraph.

3. Methods

3.1 The Data

The data points represent 100k diaelectron events in the invariant mass range 2-110 GeV, the way it was explained in the first chapter. The dataset includes several features, namely Run, Event, E1 and E2, px1, py1, pz1, px2, py2, and pz2, pt1, pt2, eta1, eta2, phi1, phi2, Q1, Q2, and M, as described in the previous chapter as well. We will focus on the following features for our analysis: the total energies of the two electrons, E1 and E2; the components of the momentum of both electrons, px1, py1, pz1, px2, py2, and pz2; the transverse momenta, pt1 and pt2; pseudorapidities, eta1 and eta2; the phi angle of the electrons, phi1 and phi2; and, finally, the charge of the electrons, Q1 and Q2. Each row in the dataset that we feed into our model will contain these selected features, along with the invariant mass of the electrons (M), which serves as our label. *16 features and 1 label.*

These features were chosen because they tell us the physical properties and characteristics of the electrons and their interactions, and the Run and the Event numbers do not provide any physical or relevant information for our predictions. With these features we ensure that we have the relevant information for making our predictions.

3.2 Choice of Model

The complexity of our dataset, involving multiple nonlinear variables, is what led us to choose the Random Forest Regression model. When employing Linear Regression as the primary model, the complex and nonlinear interactions between the features may produce

less-than-ideal results. However, Linear Regression is still a useful backup option because it enables us to get a quick overview of our data, including information on the possibility of presence and frequency of outliers. It is also easy to understand and implement.

The Random Forest model's robustness to noise and outliers is a key factor in our selection, like we covered in paragraph 1. *Problem Formulation* there can be anomalies and with this regression model we can do stable prediction even with these anomalies.

The choice of the loss function is Mean Squared Error, since it quantifies the average squared between predicted and actual values. With this we can minimize our predictions error and make the predictions closer to the true invariant masses. It is also commonly used in physics because it aligns with the concept of minimizing error when making measurements or predictions. We also noticed that regression error (R^2) is used in this [2] code, so we implemented it as well.

3.3 Model Validation

We picked our training set to be 70% of the whole data. We use this set to train our Random Forest Regression model. The model has enough events to learn the relationship between the variables and the invariant mass. The validation set is 15% of the data. We use this for tuning and we assess the performance of the model selection. The rest 15% of the data we use for the test set. It is important to keep this set out of the model selection so our model can provide an unbiased estimate of how our model performs.

We went with this split because we have enough data and because we are trying to have a model that predicts a complicated physical phenomenon, with a lot of dependencies, so the training set is big, more than half of the set.

4. Results

Like we discussed in the previous chapter we will use validation errors to check which of the models are more accurate. 15% of the dataset was used as our testing set, 15% as our validation set and 70% as our training set. We made sure the test set remained untouched during model training. We had some datapoints where the values for the label ‘M’ were missing, and we modified the dataset by removing them.

We compared our Linear Regression model to Random Forest Regression. When we set the validation features and label to our predicted Linear Regression, we calculate our mean squared error (MSE) as 385.09 and our R^2 as 0.3966. This MSE is quite large, and our R^2 is

small. This can indicate two things, our training model is not accurate, and our data is non-linear, or the data we provided is flawed. We can test these two theories if we check the dataset using a model that trains nonlinear data.

Next, we use Random Forest Regression to predict our label 'M'. Once we predict the data and check the errors between the validation and predicted data's we get a lower MSE of 33.64 and for our R^2 we get 0.9473. Based on these results we can conclude that the Random Forest model is a lot more accurate than the Linear Regression model, since we scored a lower error on the MSE and R^2 is a lot bigger. Like we discussed previously the reason that Random Forest Regression scores a better error is the fact that it handles nonlinear patterns and takes account of outliers.

We further evaluated the model using the test set, obtaining an MSE of 32.72 and an R^2 of 0.9487. These results align closely with the validation set, indicating that overfitting is unlikely, as the validation and test errors are of similar magnitude.

5. Conclusion

We used two different machine learning methods, Linear Regression and Random Forest Regression to predict the invariant mass of dielectron collisions. We used the physical attributes from the data as our features, and the invariant mass as the label we want to predict. As seen in the previous chapter we chose Random Forest Regression as our final model since it scored lower MSE and higher R^2. In quantitative terms, Random Forest Regression achieved a lower MSE of 32.72 and a higher $R^2$of 0.9487, outperforming Linear Regression, which had an MSE of 385.09 and an R^2 of 0.3966.

We also attempted to identify correlations between the features and our dataset and observed that there were significant connections between the transverse momentums of the electrons (pt2 and pt1) as well as the total energies of the electrons (E2 and E1) and the variations in our invariant mass. *See the chart 'Correlation of Features with Target Variable (M)' in the appendix.*

We are pleased with the model's performance. It has learned quite effectively, but the mean square error is still relatively high. Although the model seemed promising, we don't think it is suitable for predicting the invariant mass of dielectron collisions (or this data).

The data can be improved in a way that the events in the run are done more frequently, since right now there are rapid changes between the datapoints that could affect our predictions. Another thing that we could improve is the feature selection, since we found correlations between pt2, pt1, E1 and E2 features and our label we could've only chosen those features

for the problem. These are things that possibly limit our model, and doing these changes might provide us with more accurate results. Also if we use a better model, such as a model that uses clustering, we could possibly unveil larger between validation and testing errors thus revealing overfitting in our data.

6. References

[1]https://opendata.cern.ch/record/304
[2]https://www.kaggle.com/code/pavanpadubidri/cern-electron-mass-prediction (we have the same methods as this, and we got the regression error from here.)

https://en.wikipedia.org/wiki/Pseudorapidity

https://www.kaggle.com/datasets/fedesoriano/cern-electron-collision-data?resource=download

https://www.kaggle.com/code/tiznadoeth/electron-invariant-mass-prediction

https://www.kaggle.com/code/amarshah1999/cern-electron-collision-mass-prediction

https://en.wikipedia.org/wiki/Invariant_mass#:~:text=The%20invariant%20mass%2C%20rest%20mass,overall%20motion%20of%20the%20system.

7. Appendix

Appendix

October 11, 2023

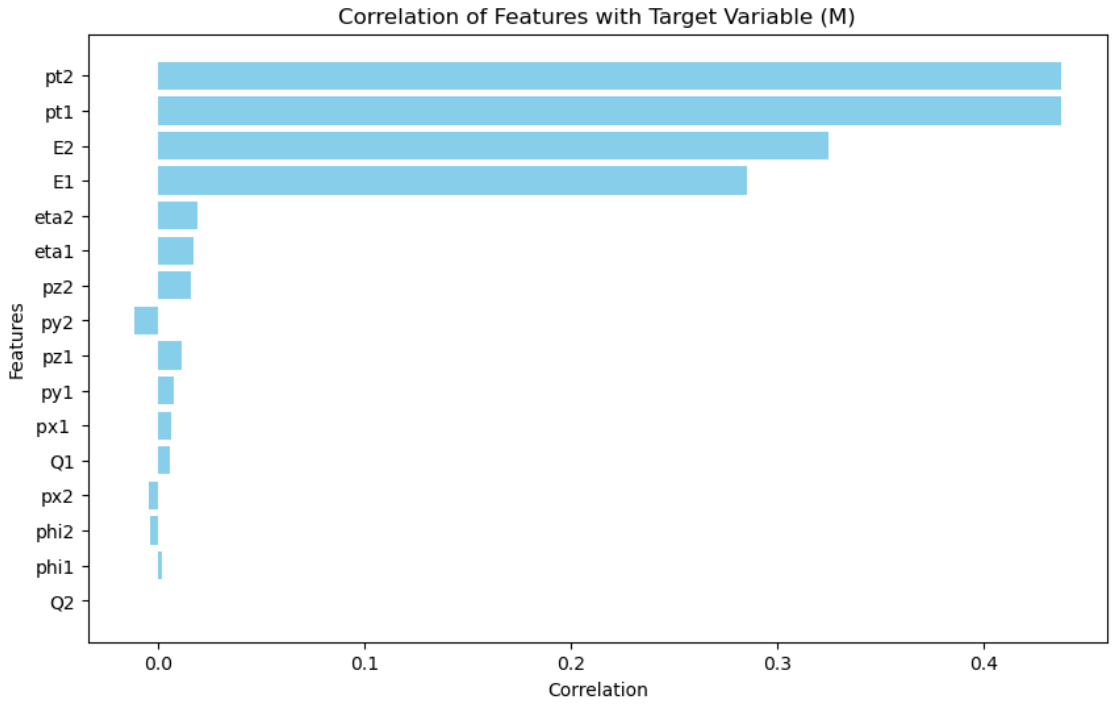
[16]:
```
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import r2_score,mean_squared_error
from sklearn.preprocessing import MinMaxScaler
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt


data = pd.read_csv('dielectron.csv')
print(data.head())

selected_column_indices = [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,␣
 ↪16,17, 18]
filtered_data = data.iloc[:, selected_column_indices]

#we check what data  is missing so we can clear them up. If we don't we will␣
 ↪face
#errors when we set our models.
filtered_data.isnull().sum()

filtered_data.dropna(inplace=True)
X = filtered_data.drop('M', axis=1) # Features
y = filtered_data['M'] # Our label 'M' invariant mass


#next we check what features affect our target  variable 'M'
# Calculate Pearson's correlation coefficients
correlation_matrix = X.corrwith(y)
# Create a DataFrame to store the results
correlation_df = pd.DataFrame({'Feature': X.columns, 'Correlation':␣
 ↪correlation_matrix})
# Sort the features by correlation magnitude (absolute value)
correlation_df = correlation_df.reindex(correlation_df['Correlation'].abs().
 ↪sort_values(ascending=False).index)
```

```
# Print the sorted DataFrame
print(correlation_df)
correlation_df = correlation_df.reindex(correlation_df['Correlation'].abs().
 ↪sort_values(ascending=False).index)

# Create a bar plot to visualize the correlations
plt.figure(figsize=(10, 6))
plt.barh(correlation_df['Feature'], correlation_df['Correlation'],␣
 ↪color='skyblue')
plt.xlabel('Correlation')
plt.ylabel('Features')
plt.title('Correlation of Features with Target Variable (M)')
plt.gca().invert_yaxis()  # Invert y-axis to display the highest correlation at␣
 ↪the top
plt.show()
```

```
      Run      Event         E1      px1         py1       pz1       pt1  \
0  147115  366639895  58.71410  -7.31132  10.531000 -57.29740  12.82020
1  147115  366704169   6.61188  -4.15213  -0.579855  -5.11278   4.19242
2  147115  367112316  25.54190 -11.48090   2.041680  22.72460  11.66100
3  147115  366952149  65.39590   7.51214  11.887100  63.86620  14.06190
4  147115  366523212  61.45040   2.95284 -14.622700 -59.61210  14.91790

      eta1     phi1  Q1       E2        px2      py2      pz2       pt2  \
0 -2.20267  2.17766   1  11.2836  -1.032340 -1.88066 -11.0778   2.14537
1 -1.02842 -3.00284  -1  17.1492 -11.713500  5.04474  11.4647  12.75360
2  1.42048  2.96560   1  15.8203  -1.472800  2.25895 -15.5888   2.69667
3  2.21838  1.00721   1  25.1273   4.087860  2.59641  24.6563   4.84272
4 -2.09375 -1.37154  -1  13.8871  -0.277757 -2.42560 -13.6708   2.44145

       eta2      phi2  Q2         M
0 -2.344030 -2.072810  -1   8.94841
1  0.808077  2.734920   1  15.89300
2 -2.455080  2.148570   1  38.38770
3  2.330210  0.565865  -1   3.72862
4 -2.423700 -1.684810  -1   2.74718
     Feature  Correlation
pt2      pt2     0.437158
pt1      pt1     0.437140
E2        E2     0.324660
E1        E1     0.285231
eta2    eta2     0.018772
eta1    eta1     0.017102
pz2      pz2     0.015702
py2      py2    -0.011725
pz1      pz1     0.011526
py1      py1     0.007538
```

```
px1     px1      0.006112
Q1        Q1     0.005624
px2      px2    -0.004419
phi2    phi2    -0.004145
phi1    phi1     0.001833
Q2        Q2    -0.000444
```

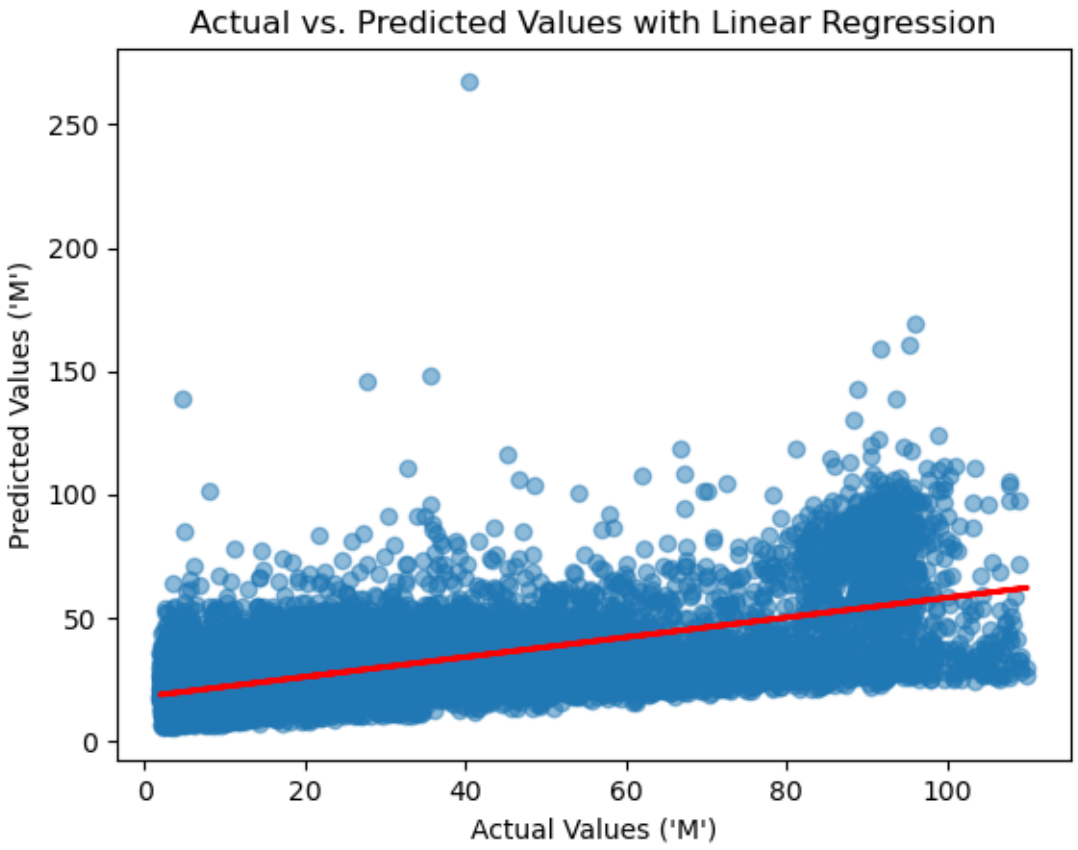
```
/tmp/ipykernel_449/3199294568.py:22: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  filtered_data.dropna(inplace=True)
```



[17]:
```
#we train 70% of the data, use 15% for testing and 15% for validation
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5,␣
 ↪random_state=42)

# Perform feature scaling
minMax = MinMaxScaler()
X_train = minMax.fit_transform(X_train)
X_val = minMax.transform(X_val)
X_test = minMax.transform(X_test)
```

```
# Train a Linear Regression model on the training set
regression = LinearRegression()
regression.fit(X_train, y_train)

# Make predictions on the validation set
y_pred = regression.predict(X_val)

# Calculate Mean Squared Error and R-squared on the validation set
mse = mean_squared_error(y_val, y_pred)
r2 = r2_score(y_val, y_pred)

print(f'R2: {r2:.4f}, MSE: {mse:.2f}')

# Scatter plot of actual vs. predicted values
plt.scatter(y_val, y_pred, alpha=0.5)
plt.xlabel("Actual Values ('M')")
plt.ylabel("Predicted Values ('M')")
plt.title("Actual vs. Predicted Values with Linear Regression")

# Fit a linear regression line
linear_reg = LinearRegression()
linear_reg.fit(y_val.values.reshape(-1, 1), y_pred.reshape(-1, 1))
plt.plot(y_val, linear_reg.predict(y_val.values.reshape(-1, 1)), color='red',␣
 ↪linewidth=2)

plt.show()
```

```
R2: 0.3966, MSE: 385.09
```

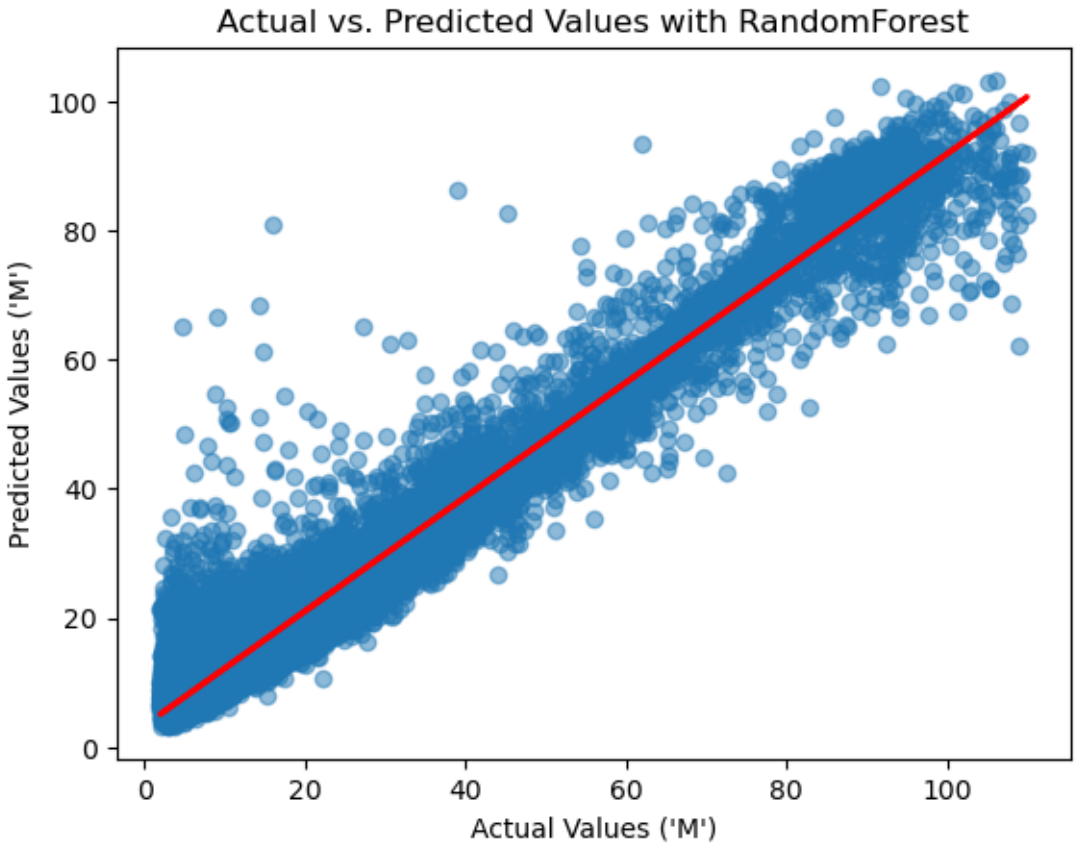


[18]:
```
#we notice that our R2 is way too small, and our MSE is a large number.
#This indicates that the data is non linear, on the label 'M'
```

[24]:
```
regression = RandomForestRegressor()
regression.fit(X_train, y_train)
y_pred = regression.predict(X_val)

# Calculate Mean Squared Error and R-squared
mse = mean_squared_error(y_val, y_pred)
r2 = r2_score(y_val, y_pred)

print(f'R2:{r2:0.4f}.MSE: {mse:0.2f}')
#no let's plot it
plt.scatter(y_val, y_pred, alpha=0.5)
plt.xlabel("Actual Values ('M')")
plt.ylabel("Predicted Values ('M')")
plt.title("Actual vs. Predicted Values with RandomForest")

# Fit a linear regression line
linear_reg = LinearRegression()
```

```
linear_reg.fit(y_val.values.reshape(-1, 1), y_pred.reshape(-1, 1))
plt.plot(y_val, linear_reg.predict(y_val.values.reshape(-1, 1)), color='red',␣
 ↪linewidth=2)

plt.show()
```

```
R2:0.9474.MSE: 33.54
```

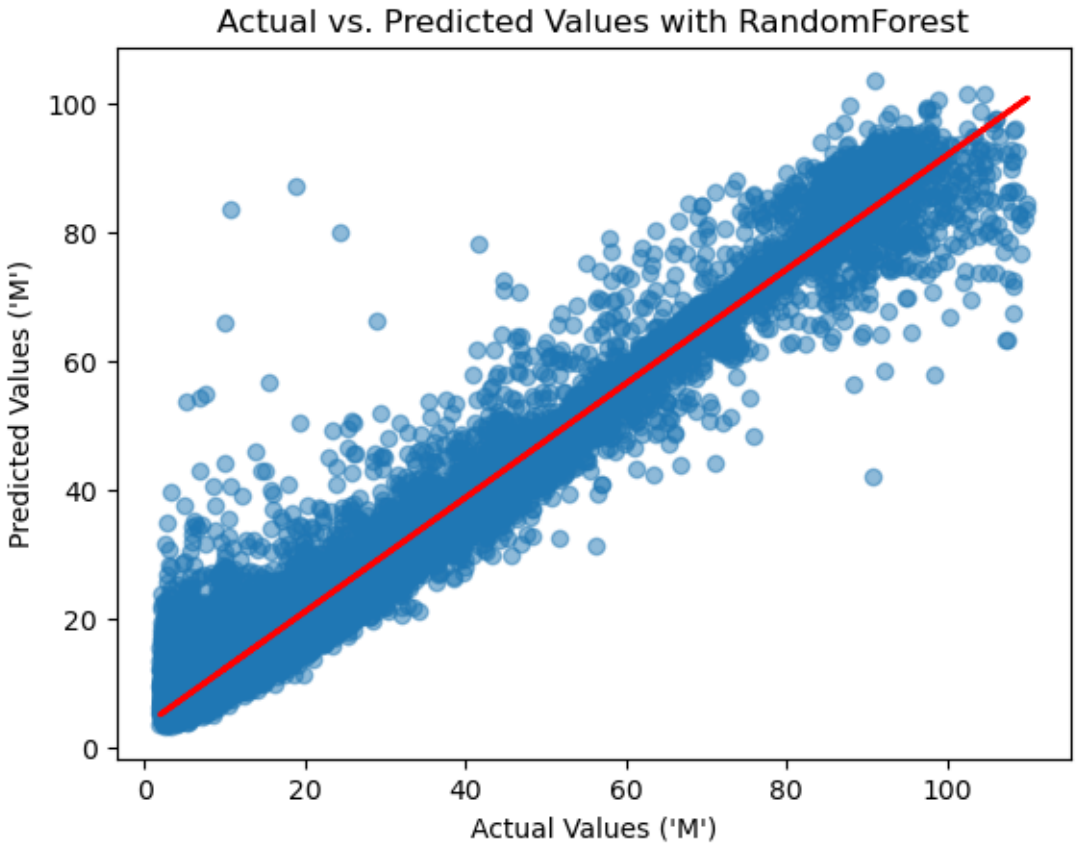


[22]:
```
#okay, now we got a relatively smasller MSE which is what we're looking for and␣
 ↪ o higher R^2
#using our validation set now we train the set again and we test it using the␣
 ↪validation set, if the results
#are low once more we've found a suitable model.

y_pred = regression.predict(X_test)

# Calculate Mean Squared Error and R-squared
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
```

```
print(f'R2:{r2:0.4f}.MSE: {mse:0.2f}')
```

```
R2:0.9487.MSE: 32.72
```

[23]:
```
#no let's plot it
plt.scatter(y_test, y_pred, alpha=0.5)
plt.xlabel("Actual Values ('M')")
plt.ylabel("Predicted Values ('M')")
plt.title("Actual vs. Predicted Values with RandomForest")

# Fit a linear regression line
linear_reg = LinearRegression()
linear_reg.fit(y_test.values.reshape(-1, 1), y_pred.reshape(-1, 1))
plt.plot(y_test, linear_reg.predict(y_test.values.reshape(-1, 1)), color='red',␣
 ↪linewidth=2)

plt.show()
```



[22]:
```
#We also notice that our plot looks much better.
```