Comparing Regression Models for Electron Mass Prediction

October 11, 2023

1 Introduction

In this project, we are looking into the topic of particle physics, specifically focusing on electron events conducted by CERN in 2010. These events involve collisions of particles, resulting in a large amount of data of such collisions involving dielectrons. Our primary objective is to predict the mass of electrons based on the available dataset for each event. To achieve this, we empoly Machine Learning by utilizing the labeled data provided in the dataset to train a predictive model. This report is structured as follows: first, we introduce the problem formulation and feature selection, outlining the dataset and data preprocessing steps. Next, we move into our models and feature selection process, discussing the relevant attributes and the reasoning behind choosing our prospective regression models for our specific case. Our goal is to take advantage of the benefits of machine learning to gain insights into the world of particle physics and make accurate predictions about electron masses in these collision events.

2 Problem Formulation and Feature selection

In This project we are dealing with a series of *events* of electrons accomplished by CERN in 2010. Events are thereby defined as a collision of particles. During this series data of 100.000 collisions between electrons were collected. The electrons in this events had an invariant mass in a range of 2 and 110 GeV.[1]
Following, we want to predict the mass of the electrons based on the given data of each event. We opt for supervised learning in our project because we aim to predict the mass of electrons based on the provided dataset of events, where the actual mass values are available (labeled data). Supervised learning allows us to train a predictive model using the labeled data, enabling it to make mass predictions for new unseen events.

2.1 Data Description and Data Prepossessing

We are using the dataset of this series provided by the "Open Data Portal" from CERN. The dataset includes 19 features, as shown in the tables below.

Feature	Description	Type
Run	run number	Int
Event	event number	Int
E1	total energy	Float
E2	total energy	Float
px1	x component	Float
py1	y component	Float
pz1	z component	Float
px2	x component	Float
py2	y component	Float
pz2	z component	Float

Feature	Description	Type
pt1	transverse momentum	Float
pt2	transverse momentum	Float
eta1	pseudorapidity	Float
eta2	pseudorapidity	Float
phi1	phi angle	Float
phi2	phi angle	Float
Q1	charge	{-1, 1}
Q2	charge	{-1, 1}
M	Invariant Mass	Float

The dataset consists of 100.000 entries thus making it a large dataset. In order to correctly apply machine learning methods to our dataset we have to apply some prepossessing techniques. In this way we are ensuring data quality, noise reduction and making it more suitable for model training.[2]

We begin with examining the dataset and looking for missing or duplicate values. By dropping the duplicate values we ensure data integrity and prevent redundancy, while removing null values helps maintain data consistency and prevents potential issues in model training. While the number of data dropped in total is relatively small to the total size of features, this common practise technique is applied in order to achieve the best possible results.

3 Models and Feature Selection

3.1 Feature Selection and Analysis

For the feature selection stage we went through a process of determining the features that are going to end up being the important ones for our model. In order to do that we:

1. Generated a heatmap that's showing the correlation of each feature with the target.
2. Generated useful scatter plots between Mass and each other feature.

After examining the above graphs, we noticed et1, et2 as well as Q1 and Q2 and both phi angles do not offer much of help on determining the Mass as there absolutely no geometrical correlation between the 2 factors. By further examining the heatmap we noticed a high correlation between E1,E2 as well as pt1 and pt2. When we are dealing with supervised model, highly correlated values will not improve the results. For linear models large correlation can lead to unstable results and can cause large variations. Moreover, although random forests model works very well on distinguishing those features, keeping them can mask this ability.
For the above mentioned reasons, for our case study and experiment we ended up using the components of the electron 1 and 2, that is px, py and pz. Finally, we used boxplot graphs to analyze our selected features and identified the presence of outliers. However, since the number of instances with outliers was small, we decided to retain them in our analysis.

3.2 Models

The are many machine learning models that are used for accurate target predictions in supervised learning but not all of them work best for all cases. For this reason, we had to select our model based on our case study and our dataset. The two models that we are going to investigate in our report are **Random Forest Regression**, **Decision tree regression** and **MLP Regression**. In more detail we will look through the thought process lead to the choosing model for our case:

Random Forest Regression (RFG)
By utilizing this model, we can take advantage of its robustness to outliers, which is highly beneficial since that during the prepossessing phase, we did not perform any outlier removal. Moreover, it uses multiple decision trees, each trained for a subset of the data, thus being more likely to find a more optimal solution than a local optimal. It's also worth noting that by drooping features on the feature selection part we are more likely to overfit since our model is getting trained on a fewer specimens which are less diverse, so Random Forests offers an

advantage to that since it is robust to overfit. Lastly, having already notices the non-linear nature of our dataset, we are considering this model for its good performance with such datasets.[3]

Decision Tree Regression (DTR)
Decision trees are renowned for their robust performance with large datasets, making them an ideal choice for our needs given the scale of our dataset. Furthermore, their versatility extends beyond standard predictions, making them readily adaptable to our specific case. Another notable advantage of this method lies in its intuitive and easily comprehensible nature, making it straightforward to apply in addressing our problem.[4]

Multi-layer Perceptron Regressor (MLP)
The Multi-layer perceptron regression model is using an artificial neural network of multiple layers of perceptrons. This regression model is good at capturing complex non-linear relationships within our dataset. In order to prevent overfitting we carefully fine-tune its hyperparameters in the following sections. Its adaptability to diverse attributes is particularly advantageous for our dataset, making the MLP Regressor a robust choice.[5]

3.3 Loss Function

Mean Squared Error

One compelling reason to use this loss function is its ability to decrease the impact of outliers. Outliers can lead to significantly larger error values because of the squaring operation in the Mean Squared Error (MSE) formula:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

As a result, when encountering outliers, the MSE increase their influence on the overall error, making it easier to identify and disregard them in the modeling process. Moreover, the mean squared error function is chosen as it allowed us to use the already made library from *sklearn*, as it is the default function of our models and the most widely used one.[6]

4 Model Validation

To validate the result of our model we splitted our data with *train_test_split* from *sklearn* into training, validation and test-set with a ratio of 0.8/0.1/0.1. We were able to afford such a big training-set because of our large dataset, by which we can ensure enough data for our validation and test-set.
After we trained our model with our training-set we tuned the hyperparameters of our models with the validation-set to improve the performance of the chosen models. At the end we compare the models by predicting labels over the test-set and comparing these with the labels of our test-set by the MSE.

5 Results

After prepossessing the data we initially trained our models with the training-set and the default parameters provided by *sklearn*. Moving on, we are predicting with each model on the training-set and validation-set and computing the training and test-error. Obtaining these, allows us to compare the two models for the first time. It's clearly visible, DTR model is overfitting a lot compared to our RFG model. More specifically training-error of the DTR model

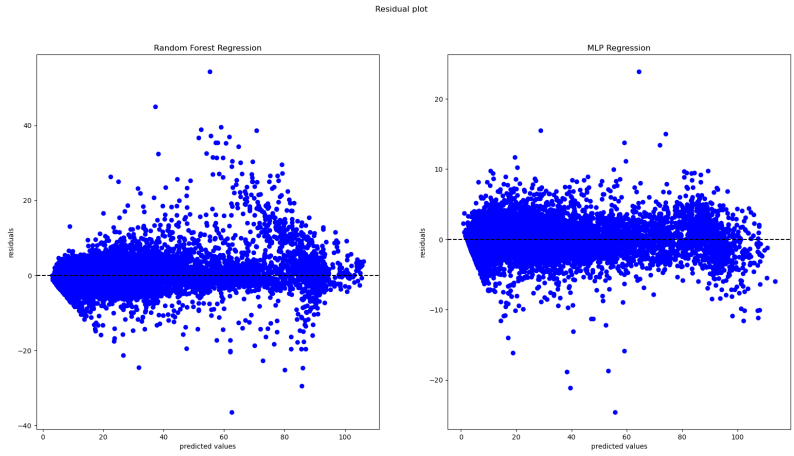
equals to zero and the validation-error is ~ 47.854. On the other hand the training-error of the RFG model is ~ 2.535 and validation-error is ~ 18.045, indicating less overfitting. Introducing MLP model to the comparison, it's training-error during the initial prediction is ~ 10.375 and the validation-error is ~ 12.716. Based on these results, we are going to focus on RFG and MLP as they deliver better validation-error performance and less overfitting.
For both models, we used the *GridSearchCV* from *sklearn* to tune our hyper-parameter on the validation-set.[7]. This enables us to find better fitting parameters for our dataset and improve the performance. As a result we improve the validation-error of RFG model to ~ 5.116 and ~ 2.288 respectively.

Continuing our comparison, we are also going through the test-error results. As aforementioned, we randomly split the training-set from the original dataset with a ratio of 10%. It's important that we never used it to train our models to provide a good comparison.

The final results are very impressive supported by the fact the MLP model is showing a very precise prediction on the test-set. The RFG model obtained a final test-error of ~ 4.498 and the MLP model a test-error of ~ 2.056.

6 Conclusion

All in all, we can see that the MLP model reaches the best performance in the end. The test-error doesn't differ from the training-error on such a large scale the RFG model does. This indicates that our RFG model still suffers from overfitting. On the other hand, RFG model is much faster. During the final run, it needed approximately 1.5 minutes to train and predict, whereas MLP model needed over 15 minutes.
In the following graph, we compare the two models by their residual plot. It's getting clear, that the outliers of MLP model are not as distant to real labels as is the case for the RFG model.



Equally noteworthy, is the performance increase of the MLP model by the higher number of maximum iterations. Should one train it for longer the result would still improve, contrary to the fact that RFG model wouldn't improve on such a scale by increasing the number of estimators.

To draw the discussion to a close, for the current case study the MLP model showed the best performance, indicating further improvements by training it for a bigger span of time and therefore stands out as the best fitting model to predict the invariant mass in our dataset.

References

[1] T. McCauley, "Dielectron events for use in education and outreach," 2016. Publisher: CERN Open Data Portal.

[2] scikit-learn contributors, "Preprocessing." https://scikit-learn.org/stable/modules/preprocessing.html, 2023.

[3] scikit-learn contributors, "Randomforestregressor." https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html, 2023.

[4] scikit-learn contributors, "Decisiontreeregressor." https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html, 2023.

[5] scikit-learn contributors, "Mlpregressor." https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html, 2023.

[6] T. Chai and R. R. Draxler, "Root mean square error (RMSE) or mean absolute error (MAE)?," preprint, Numerical Methods, Feb. 2014.

[7] scikit-learn contributors, "Gridsearchcv." https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html, 2023.

[8] F. Soriaño, "Cern electron collision data." https://www.kaggle.com/datasets/fedesoriano/cern-electron-collision-data, 2010.

Appendix

October 10, 2023

[141]:
```
import time
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import PolynomialFeatures, StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.neural_network import MLPRegressor
from sklearn import tree
from sklearn.ensemble import RandomForestRegressor

from warnings import simplefilter
from sklearn.exceptions import ConvergenceWarning
simplefilter("ignore", category=ConvergenceWarning)
```

[142]:
```
# read data
df = pd.read_csv('dielectron.csv')
df.head(5)
```

[142]:

	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
```

[143]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 19 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   Run     100000 non-null  int64  
 1   Event   100000 non-null  int64  
 2   E1      100000 non-null  float64
 3   px1     100000 non-null  float64
 4   py1     100000 non-null  float64
 5   pz1     100000 non-null  float64
 6   pt1     100000 non-null  float64
 7   eta1    100000 non-null  float64
 8   phi1    100000 non-null  float64
 9   Q1      100000 non-null  int64  
 10  E2      100000 non-null  float64
 11  px2     100000 non-null  float64
 12  py2     100000 non-null  float64
 13  pz2     100000 non-null  float64
 14  pt2     100000 non-null  float64
 15  eta2    100000 non-null  float64
 16  phi2    100000 non-null  float64
 17  Q2      100000 non-null  int64  
 18  M       99915 non-null   float64
dtypes: float64(15), int64(4)
memory usage: 14.5 MB
```

[144]:
```
# check null values
df.isnull().sum()
```

[144]:
```
Run       0
Event     0
E1        0
px1       0
py1       0
pz1       0
pt1       0
eta1      0
phi1      0
Q1        0
E2        0
```

```
px2       0
py2       0
pz2       0
pt2       0
eta2      0
phi2      0
Q2        0
M        85
dtype: int64
```

[145]:
```
# delete null values
df = df.dropna(axis=0)
df.isnull().sum()
```

[145]:
```
Run      0
Event    0
E1       0
px1      0
py1      0
pz1      0
pt1      0
eta1     0
phi1     0
Q1       0
E2       0
px2      0
py2      0
pz2      0
pt2      0
eta2     0
phi2     0
Q2       0
M        0
dtype: int64
```

[146]:
```
# check for duplicates
df.duplicated().sum()
```

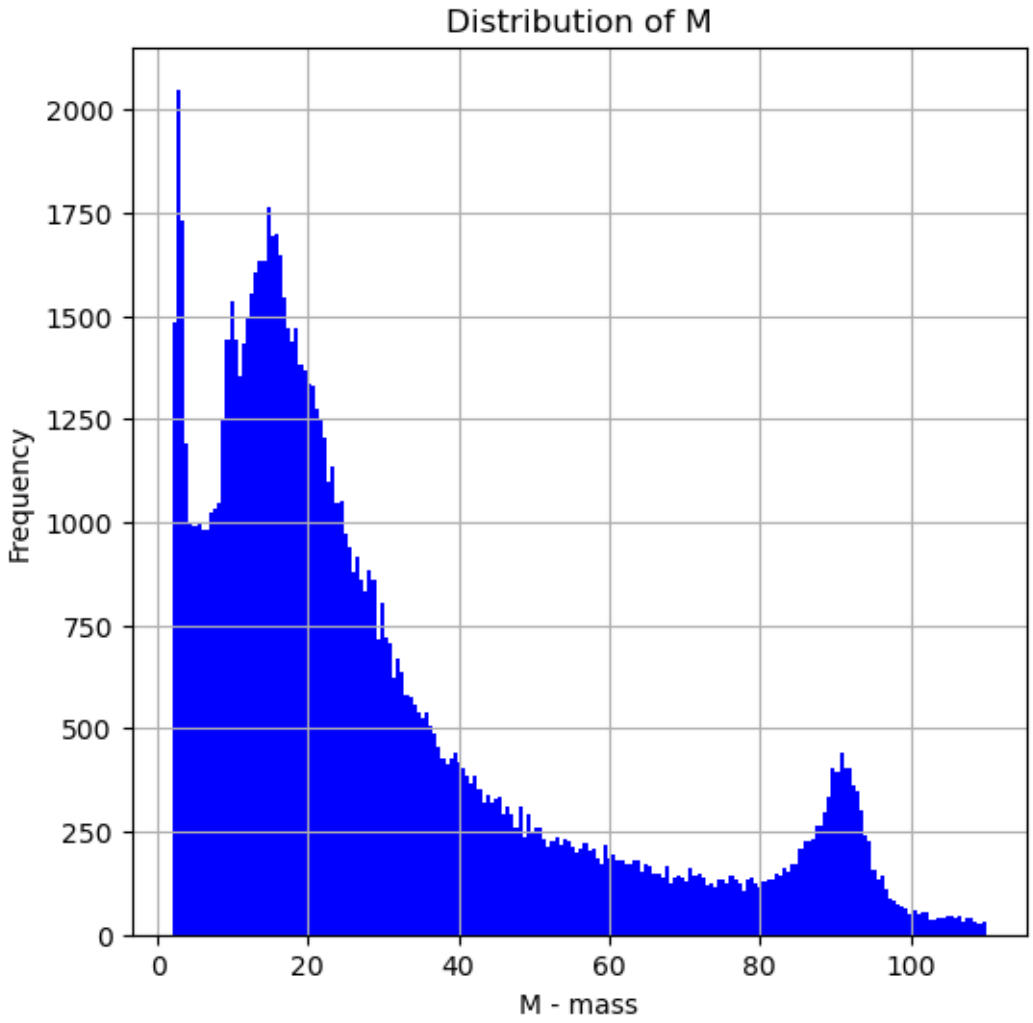
[146]:
```
23
```

[147]:
```
# delete duplicates
df = df.drop_duplicates()
df.duplicated().sum()
```

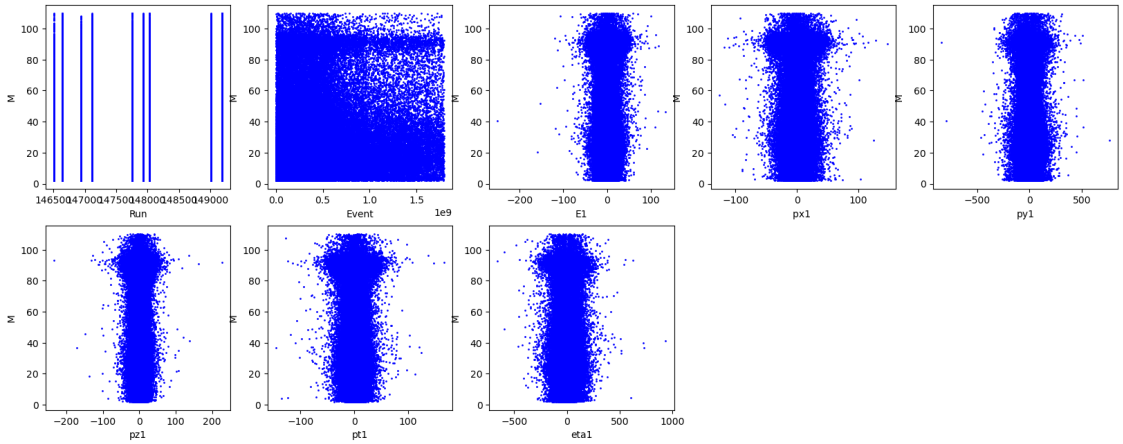
[147]:
```
0
```

[148]:
```
# generate data
features = df.drop(['M', 'pt1', 'pt2', 'E1', 'E2','Q1', 'Q2', 'phi1', 'phi2',␣
 ↪'eta1', 'eta2'], axis=1).to_numpy()
labels = df.M.to_numpy()
```

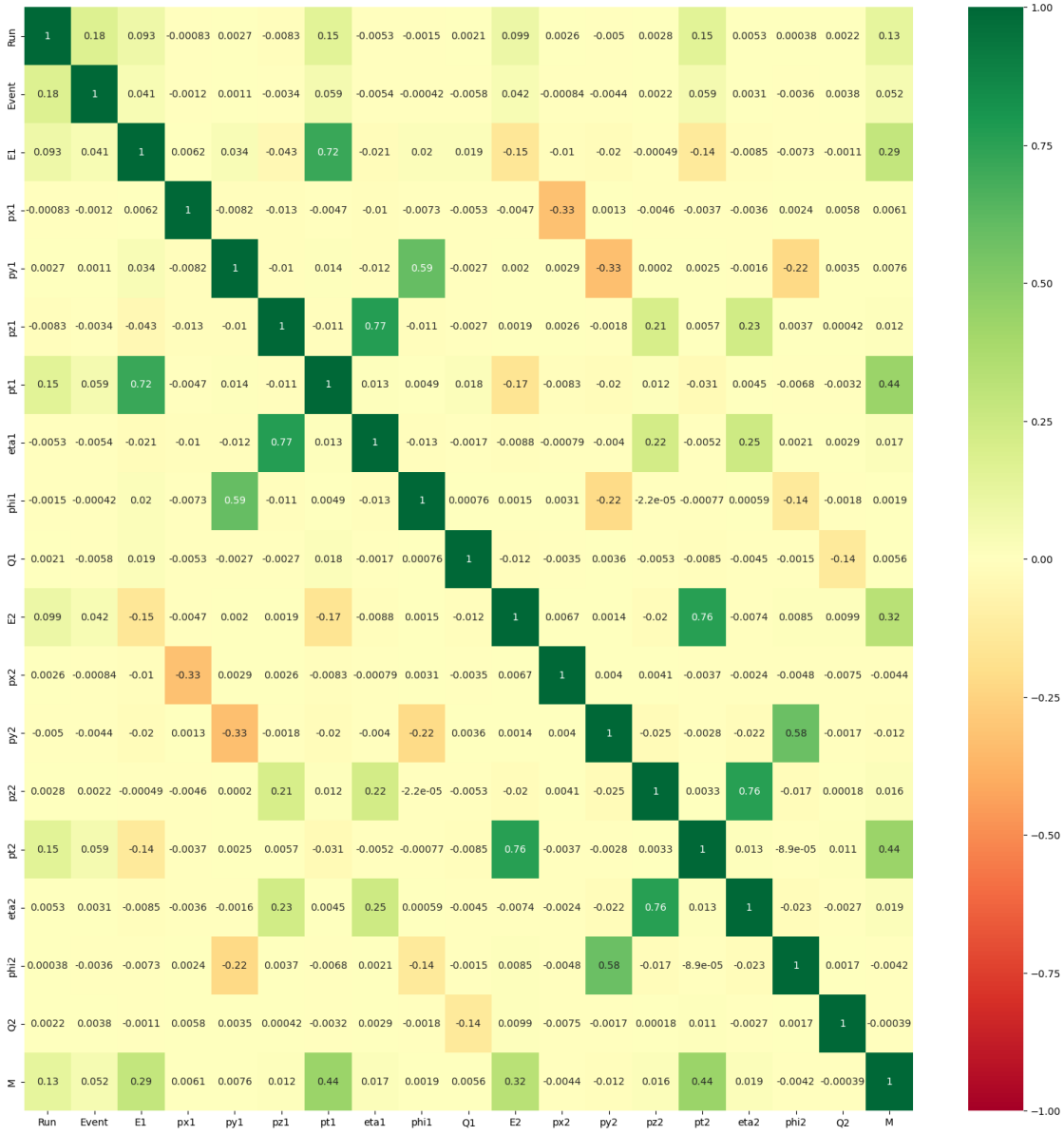
[149]:
```
# plot distribution of mass
plt.figure(figsize=(6,6))
df.M.plot(kind='hist', bins=200, color='blue')
plt.title('Distribution of M')
plt.xlabel('M - mass')
plt.grid()
plt.show()
```



[150]:
```
#create a scatte plot between M and each feature
plt.figure(figsize=(20,20))
for i in range(0, features.shape[1]):
    plt.subplot(5, 5, i+1)
    plt.scatter(features[:,i], labels, color='blue', s=1)
    plt.xlabel(df.columns[i])
    plt.ylabel('M')
plt.show()
```

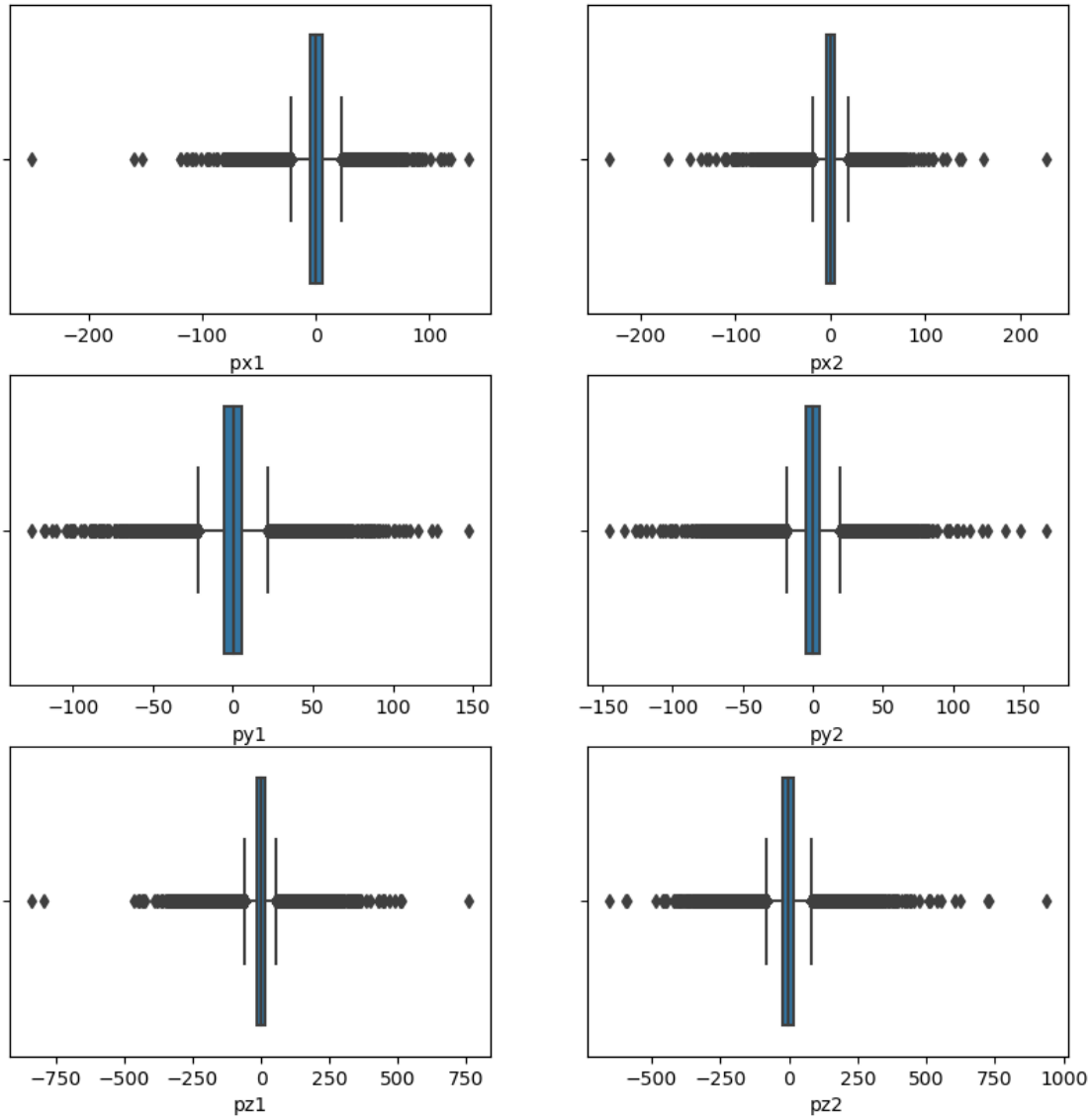


[151]:
```
# plot a correlation matrix
corr = df.corr()
fig = plt.figure(figsize = (20,20))
sns.heatmap(corr, annot=True, cmap='RdYlGn',vmin=-1, vmax=1)
plt.show()
```



[152]:
```
#detect outliers for px1,px2,py1,py2,pz1,pz2 us boxplot
plt.figure(figsize=(10,10))
plt.subplot(3,2,1)
sns.boxplot(x=df['px1 '])
plt.subplot(3,2,2)
sns.boxplot(x=df['px2'])
plt.subplot(3,2,3)
sns.boxplot(x=df['py1'])
plt.subplot(3,2,4)
sns.boxplot(x=df['py2'])
plt.subplot(3,2,5)
```

```
sns.boxplot(x=df['pz1'])
plt.subplot(3,2,6)
sns.boxplot(x=df['pz2'])
plt.show()
```



[153]:
```
# drop features
features = df.drop(['M', 'pt1', 'pt2', 'E1', 'E2','Q1', 'Q2', 'phi1', 'phi2',␣
↪'eta1', 'eta2'], axis=1).to_numpy()
```

[154]:
```
# split data into train, test and validation sets
X_train, X, y_train, y = train_test_split(features, labels, test_size=0.
↪2,train_size=0.8)
```

```
X_val, X_test, y_val, y_test = train_test_split(X,y,test_size = 0.5,train_size␣
 ↪=0.5)
X_train.shape, y_train.shape, X_test.shape, y_test.shape, X_val.shape, y_val.
 ↪shape
```

[154]:
```
((79913, 8), (79913,), (9990, 8), (9990,), (9989, 8), (9989,))
```

[155]:
```
# standardize data
X_test_std = StandardScaler().fit_transform(X_test)
X_train_std = StandardScaler().fit_transform(X_train)
X_val_std = StandardScaler().fit_transform(X_val)

np.allclose(X_train.mean(axis=0), np.zeros(X_train.shape[1])),\
    np.allclose(X_train_std.mean(axis=0),np.zeros(X_train_std.shape[1])),\
        np.allclose(X_train.std(axis=0), np.ones(X_train.shape[1])),\
            np.allclose(X_train_std.std(axis=0), np.ones(X_train_std.shape[1]))
```

[155]:
```
(False, True, False, True)
```

[176]:
```
# compare performance of DTR on validation set
DTR = tree.DecisionTreeRegressor()
DTR.fit(X_train_std, y_train)
y_pred = DTR.predict(X_train_std)
rmse = mean_squared_error(y_train, y_pred)
print("Training Error", rmse)
y_pred = DTR.predict(X_val_std)
rmse = mean_squared_error(y_val, y_pred)
print("Validation Error", rmse)
```

```
Training Error 0.0
Validation Error 47.85420034548418
```

[177]:
```
# compare performance of RFG on validation set
RFG = RandomForestRegressor()
RFG.fit(X_train_std,y_train)
y_pred = RFG.predict(X_train_std)
rmse = mean_squared_error(y_train, y_pred)
print("Training Error", rmse)
y_pred = RFG.predict(X_val_std)
print("Validation Error: ", mean_squared_error(y_val, y_pred))
```

```
Training Error 2.5351071174792317
Validation Error:  18.046412052326872
```

[178]:
```
# compare performance of MLP on validation set
MLP = MLPRegressor()
MLP.fit(X_train_std,y_train)
y_pred = MLP.predict(X_train_std)
```

```
rmse = mean_squared_error(y_train, y_pred)
print("Training Error", rmse)
y_pred = MLP.predict(X_val_std)
print("Validation Error: ", mean_squared_error(y_val, y_pred))
```

```
Training Error 10.375157787843769
Validation Error:  12.715976521072303
```

[157]:
```
# hyperparameter tuning random forest regression
params = {'n_estimators': [80, 100, 120], 'max_features': ['sqrt', 'log2', None]}

grid = GridSearchCV(RandomForestRegressor(), params, verbose=3, n_jobs=4).
 ↪fit(X_val_std, y_val)
grid.best_params_
```

```
Fitting 5 folds for each of 9 candidates, totalling 45 fits
[CV 4/5] END max_features=sqrt, n_estimators=80;, score=0.894 total time=   1.4s
[CV 1/5] END max_features=sqrt, n_estimators=80;, score=0.903 total time=   1.4s
[CV 3/5] END max_features=sqrt, n_estimators=80;, score=0.895 total time=   1.4s
[CV 2/5] END max_features=sqrt, n_estimators=80;, score=0.901 total time=   1.5s
[CV 5/5] END max_features=sqrt, n_estimators=80;, score=0.887 total time=   1.5s
[CV 1/5] END max_features=sqrt, n_estimators=100;, score=0.906 total time=
1.9s
[CV 2/5] END max_features=sqrt, n_estimators=100;, score=0.902 total time=
1.9s
[CV 3/5] END max_features=sqrt, n_estimators=100;, score=0.898 total time=
1.9s
[CV 4/5] END max_features=sqrt, n_estimators=100;, score=0.893 total time=
1.8s
[CV 5/5] END max_features=sqrt, n_estimators=100;, score=0.896 total time=
1.8s
[CV 1/5] END max_features=sqrt, n_estimators=120;, score=0.900 total time=
2.2s
[CV 2/5] END max_features=sqrt, n_estimators=120;, score=0.899 total time=
2.2s
[CV 3/5] END max_features=sqrt, n_estimators=120;, score=0.897 total time=
2.2s
[CV 4/5] END max_features=sqrt, n_estimators=120;, score=0.900 total time=
2.2s
[CV 1/5] END max_features=log2, n_estimators=80;, score=0.925 total time=   2.1s
[CV 5/5] END max_features=sqrt, n_estimators=120;, score=0.887 total time=
2.3s
[CV 2/5] END max_features=log2, n_estimators=80;, score=0.927 total time=   2.1s
[CV 3/5] END max_features=log2, n_estimators=80;, score=0.925 total time=   2.1s
[CV 4/5] END max_features=log2, n_estimators=80;, score=0.925 total time=   2.0s
[CV 5/5] END max_features=log2, n_estimators=80;, score=0.921 total time=   2.0s
[CV 1/5] END max_features=log2, n_estimators=100;, score=0.929 total time=
2.6s
```

```
[CV 2/5] END max_features=log2, n_estimators=100;, score=0.929 total time=
2.5s
[CV 3/5] END max_features=log2, n_estimators=100;, score=0.922 total time=
2.6s
[CV 4/5] END max_features=log2, n_estimators=100;, score=0.928 total time=
2.5s
[CV 5/5] END max_features=log2, n_estimators=100;, score=0.920 total time=
2.5s
[CV 1/5] END max_features=log2, n_estimators=120;, score=0.928 total time=
3.0s
[CV 2/5] END max_features=log2, n_estimators=120;, score=0.927 total time=
3.0s
[CV 3/5] END max_features=log2, n_estimators=120;, score=0.923 total time=
3.0s
[CV 4/5] END max_features=log2, n_estimators=120;, score=0.926 total time=
3.0s
[CV 5/5] END max_features=log2, n_estimators=120;, score=0.919 total time=
3.0s
[CV 1/5] END max_features=None, n_estimators=80;, score=0.935 total time=   4.8s
[CV 2/5] END max_features=None, n_estimators=80;, score=0.944 total time=   4.8s
[CV 3/5] END max_features=None, n_estimators=80;, score=0.928 total time=   4.8s
[CV 4/5] END max_features=None, n_estimators=80;, score=0.937 total time=   4.8s
[CV 5/5] END max_features=None, n_estimators=80;, score=0.931 total time=   5.0s
[CV 1/5] END max_features=None, n_estimators=100;, score=0.935 total time=
6.2s
[CV 2/5] END max_features=None, n_estimators=100;, score=0.942 total time=
6.3s
[CV 3/5] END max_features=None, n_estimators=100;, score=0.935 total time=
6.5s
[CV 4/5] END max_features=None, n_estimators=100;, score=0.934 total time=
6.6s
[CV 5/5] END max_features=None, n_estimators=100;, score=0.931 total time=
6.6s
[CV 1/5] END max_features=None, n_estimators=120;, score=0.934 total time=
7.8s
[CV 2/5] END max_features=None, n_estimators=120;, score=0.943 total time=
7.8s
[CV 3/5] END max_features=None, n_estimators=120;, score=0.933 total time=
7.6s
[CV 4/5] END max_features=None, n_estimators=120;, score=0.936 total time=
7.5s
[CV 5/5] END max_features=None, n_estimators=120;, score=0.935 total time=
7.1s
```

[157]: {'max_features': None, 'n_estimators': 120}

[158]:
```
# compare performance of random forest regression on validation set
y_pred_val = grid.predict(X_val_std)
"Validation Error RFG: ", mean_squared_error(y_pred_val, y_val)
```

[158]:
```
('Validation Error RFG: ', 5.11564480498555)
```

[160]:
```
# hyperparameter tuning MLP
params = {'hidden_layer_sizes': [50, 100, 200], 'activation': ['identity',␣
 ↪'relu'], 'solver': ['lbfgs','sgd','adam'], 'max_iter': [100, 200,500]}

grid = GridSearchCV(MLPRegressor(), params, verbose=3, n_jobs=4).fit(X_val_std,␣
 ↪y_val)
grid.best_params_
```

```
Fitting 5 folds for each of 54 candidates, totalling 270 fits
[CV 1/5] END activation=identity, hidden_layer_sizes=50, max_iter=100,
solver=lbfgs;, score=0.013 total time=   0.1s
[CV 2/5] END activation=identity, hidden_layer_sizes=50, max_iter=100,
solver=lbfgs;, score=0.018 total time=   0.1s
[CV 3/5] END activation=identity, hidden_layer_sizes=50, max_iter=100,
solver=lbfgs;, score=0.022 total time=   0.1s
[CV 4/5] END activation=identity, hidden_layer_sizes=50, max_iter=100,
solver=lbfgs;, score=0.010 total time=   0.1s
[CV 5/5] END activation=identity, hidden_layer_sizes=50, max_iter=100,
solver=lbfgs;, score=0.020 total time=   0.0s
[CV 3/5] END activation=identity, hidden_layer_sizes=50, max_iter=100,
solver=sgd;, score=0.011 total time=   0.2s
[CV 1/5] END activation=identity, hidden_layer_sizes=50, max_iter=100,
solver=sgd;, score=0.009 total time=   0.4s
[CV 4/5] END activation=identity, hidden_layer_sizes=50, max_iter=100,
solver=sgd;, score=0.003 total time=   0.5s
[CV 2/5] END activation=identity, hidden_layer_sizes=50, max_iter=100,
solver=sgd;, score=0.016 total time=   0.5s
[CV 1/5] END activation=identity, hidden_layer_sizes=50, max_iter=100,
solver=adam;, score=0.014 total time=   0.5s
[CV 5/5] END activation=identity, hidden_layer_sizes=50, max_iter=100,
solver=sgd;, score=0.018 total time=   0.5s
[CV 2/5] END activation=identity, hidden_layer_sizes=50, max_iter=200,
solver=lbfgs;, score=0.018 total time=   0.1s
[CV 5/5] END activation=identity, hidden_layer_sizes=50, max_iter=100,
solver=adam;, score=0.020 total time=   0.5s
[CV 3/5] END activation=identity, hidden_layer_sizes=50, max_iter=200,
solver=lbfgs;, score=0.022 total time=   0.0s
[CV 3/5] END activation=identity, hidden_layer_sizes=50, max_iter=100,
solver=adam;, score=0.022 total time=   0.6s
[CV 1/5] END activation=identity, hidden_layer_sizes=50, max_iter=200,
solver=lbfgs;, score=0.013 total time=   0.1s
[CV 4/5] END activation=identity, hidden_layer_sizes=50, max_iter=200,
```

```
[CV 4/5] END activation=relu, hidden_layer_sizes=200, max_iter=500,
solver=adam;, score=0.958 total time=  18.6s

/Users/clemensbandrock/opt/anaconda3/envs/py310/lib/python3.10/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:686:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (500) reached and
the optimization hasn't converged yet.
  warnings.warn(

[CV 5/5] END activation=relu, hidden_layer_sizes=200, max_iter=500,
solver=adam;, score=0.963 total time=  11.6s
```

[160]:
```
{'activation': 'relu',
 'hidden_layer_sizes': 200,
 'max_iter': 500,
 'solver': 'lbfgs'}
```

[161]:
```
# tuning hidden layers size
params = {'hidden_layer_sizes': [200, 250, 300], 'activation': ['relu'],␣
 ↪'solver': ['lbfgs'], 'max_iter': [700]}

grid = GridSearchCV(MLPRegressor(), params, verbose=3, n_jobs=4).fit(X_val_std,␣
 ↪y_val)
grid.best_params_
```

```
Fitting 5 folds for each of 3 candidates, totalling 15 fits

/Users/clemensbandrock/opt/anaconda3/envs/py310/lib/python3.10/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:541:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
/Users/clemensbandrock/opt/anaconda3/envs/py310/lib/python3.10/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:541:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)

[CV 2/5] END activation=relu, hidden_layer_sizes=200, max_iter=700,
solver=lbfgs;, score=0.990 total time=  20.6s
[CV 3/5] END activation=relu, hidden_layer_sizes=200, max_iter=700,
solver=lbfgs;, score=0.989 total time=  20.6s

/Users/clemensbandrock/opt/anaconda3/envs/py310/lib/python3.10/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:541:
```

```
/Users/clemensbandrock/opt/anaconda3/envs/py310/lib/python3.10/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:541:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)

[CV 2/5] END activation=relu, hidden_layer_sizes=300, max_iter=700,
solver=lbfgs;, score=0.992 total time=  37.6s

/Users/clemensbandrock/opt/anaconda3/envs/py310/lib/python3.10/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:541:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)

[CV 3/5] END activation=relu, hidden_layer_sizes=300, max_iter=700,
solver=lbfgs;, score=0.991 total time=  30.6s

/Users/clemensbandrock/opt/anaconda3/envs/py310/lib/python3.10/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:541:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)

[CV 4/5] END activation=relu, hidden_layer_sizes=300, max_iter=700,
solver=lbfgs;, score=0.991 total time=  27.6s

/Users/clemensbandrock/opt/anaconda3/envs/py310/lib/python3.10/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:541:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)

[CV 5/5] END activation=relu, hidden_layer_sizes=300, max_iter=700,
solver=lbfgs;, score=0.993 total time=  25.7s
```

[161]:
```
{'activation': 'relu',
 'hidden_layer_sizes': 300,
 'max_iter': 700,
```

```
 'solver': 'lbfgs'}
```

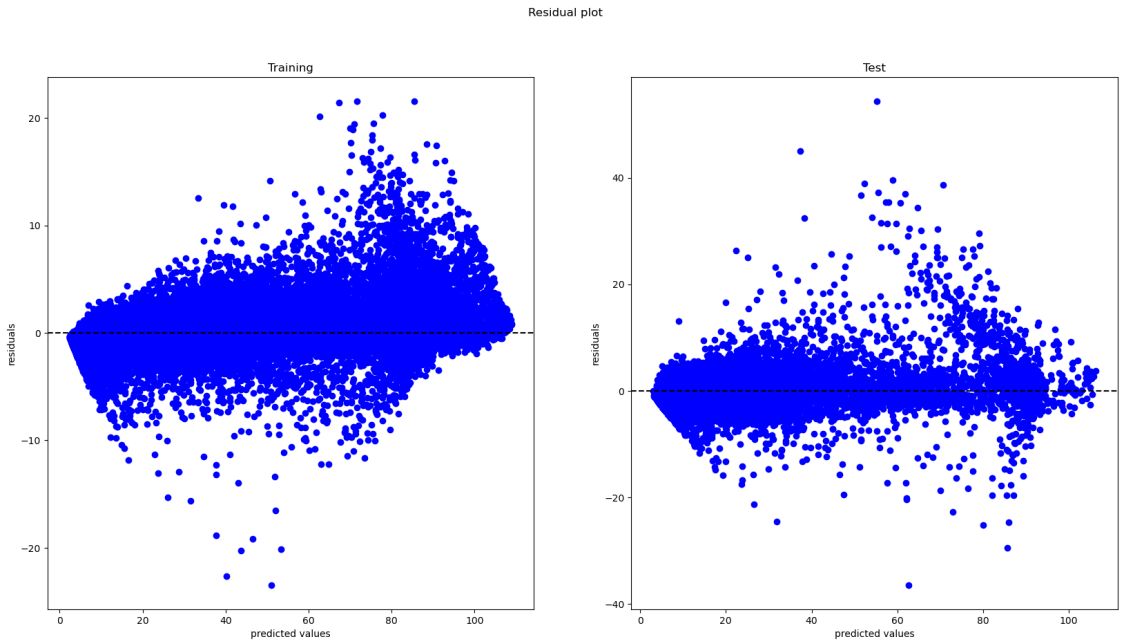
[162]:
```
# compare performance of MLP on validation set
y_pred_val = grid.predict(X_val_std)
"Validation Error: ", mean_squared_error(y_pred_val, y_val)
```

[162]:
```
('Validation Error: ', 2.2881756197888516)
```

[163]:
```
# Random Forest Regression
RFR = RandomForestRegressor(n_estimators=100)
RFR.fit(X_train_std, y_train)
print("feature importancy", RFR.feature_importances_)
y_pred_rfr_train = RFR.predict(X_train_std)
rmse_rfr = np.sqrt(mean_squared_error(y_train, y_pred_rfr_train))
print("training error", rmse_rfr)
y_pred_rfr_test = RFR.predict(X_test_std)
rmse_rfr = np.sqrt(mean_squared_error(y_test, y_pred_rfr_test))
print("test error", rmse_rfr)
```

```
feature importancy [0.00238105 0.00402386 0.09832419 0.11131413 0.29705319
0.09870191
 0.13788157 0.25032011]
training error 1.587278648980474
test error 4.497626325015393
```

[164]:
```
# residual plot RFG
residuals_train = y_train - y_pred_rfr_train
residuals_test = y_test - y_pred_rfr_test
fig, axs = plt.subplots(1,2,figsize=(20,10))
fig.suptitle('Residual plot')
axs[0].scatter(y_pred_rfr_train, residuals_train, color='blue')
axs[0].set_title("Training")
axs[1].scatter(y_pred_rfr_test, residuals_test, color='blue')
axs[1].set_title("Test")
for i in range(2):
    axs[i].axhline(y=0, color='black', linestyle='--')
    axs[i].set_xlabel('predicted values')
    axs[i].set_ylabel('residuals')
plt.show()
```

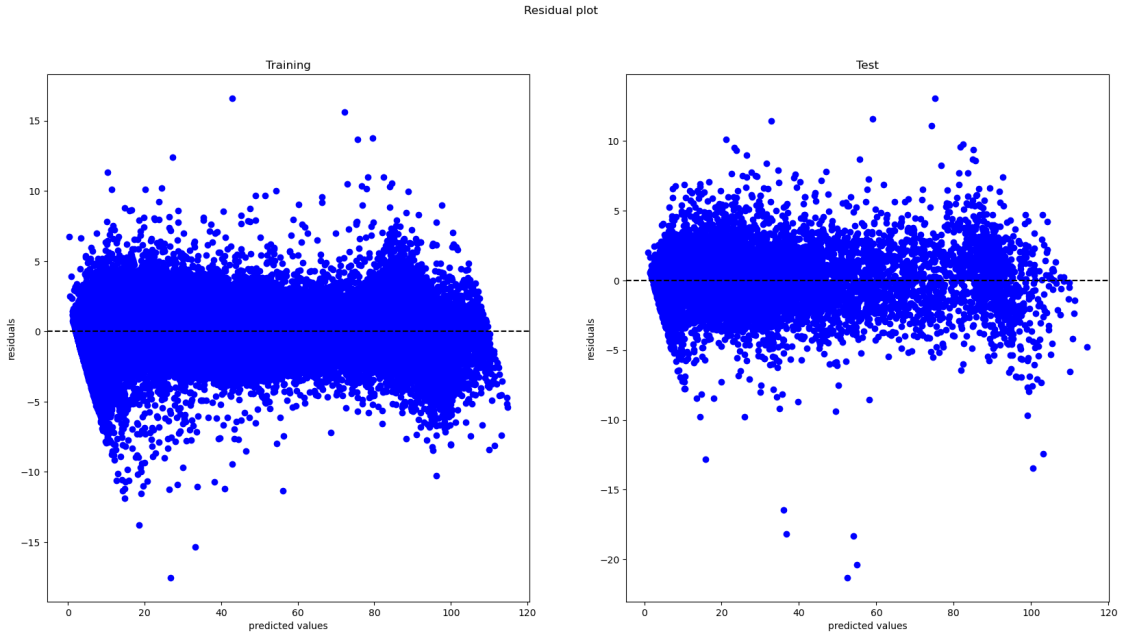


[179]:
```
# MLP Regression
MLP = MLPRegressor(activation='relu', hidden_layer_sizes=300, max_iter=2000,␣
 ↪solver='lbfgs' )
MLP.fit(X_train_std, y_train)
y_pred_mlp_train = MLP.predict(X_train_std)
rmse_mlp = np.sqrt(mean_squared_error(y_train, y_pred_mlp_train))
print("training error", rmse_mlp)
y_pred_mlp_test = MLP.predict(X_test_std)
rmse_mlp = np.sqrt(mean_squared_error(y_test, y_pred_mlp_test))
print("test error", rmse_mlp)
```

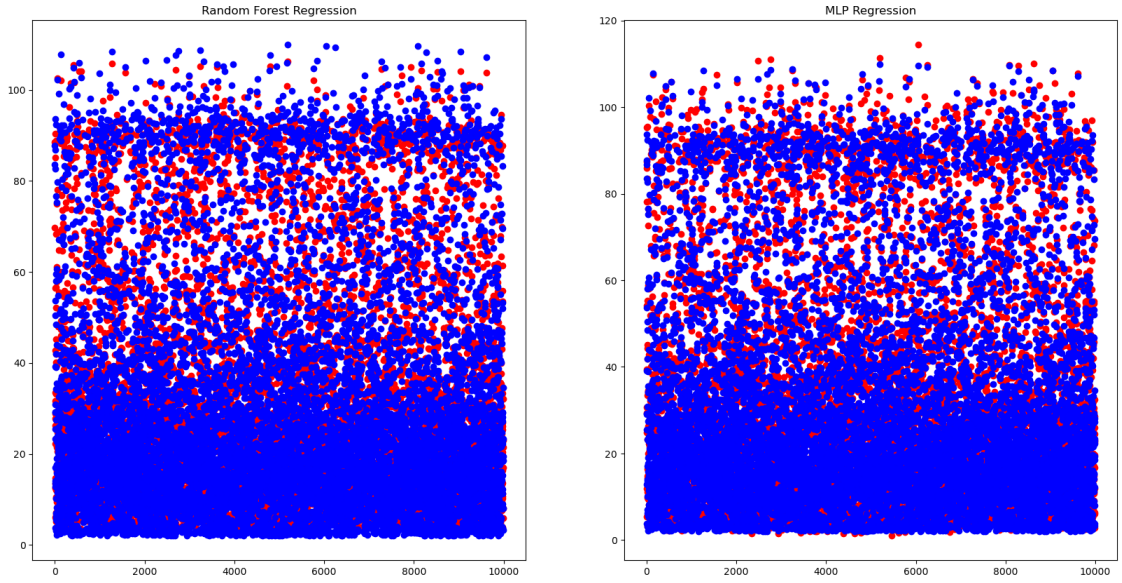
```
training error 1.4362927039275835
test error 2.0560823126426824
```

[180]:
```
# residual plot MLP
residuals_train = y_train - y_pred_mlp_train
residuals_test = y_test - y_pred_mlp_test
fig, axs = plt.subplots(1,2,figsize=(20,10))
fig.suptitle('Residual plot')
axs[0].scatter(y_pred_mlp_train, residuals_train, color='blue')
axs[0].set_title("Training")
axs[1].scatter(y_pred_mlp_test, residuals_test, color='blue')
axs[1].set_title("Test")
for i in range(2):
    axs[i].axhline(y=0, color='black', linestyle='--')
    axs[i].set_xlabel('predicted values')
```

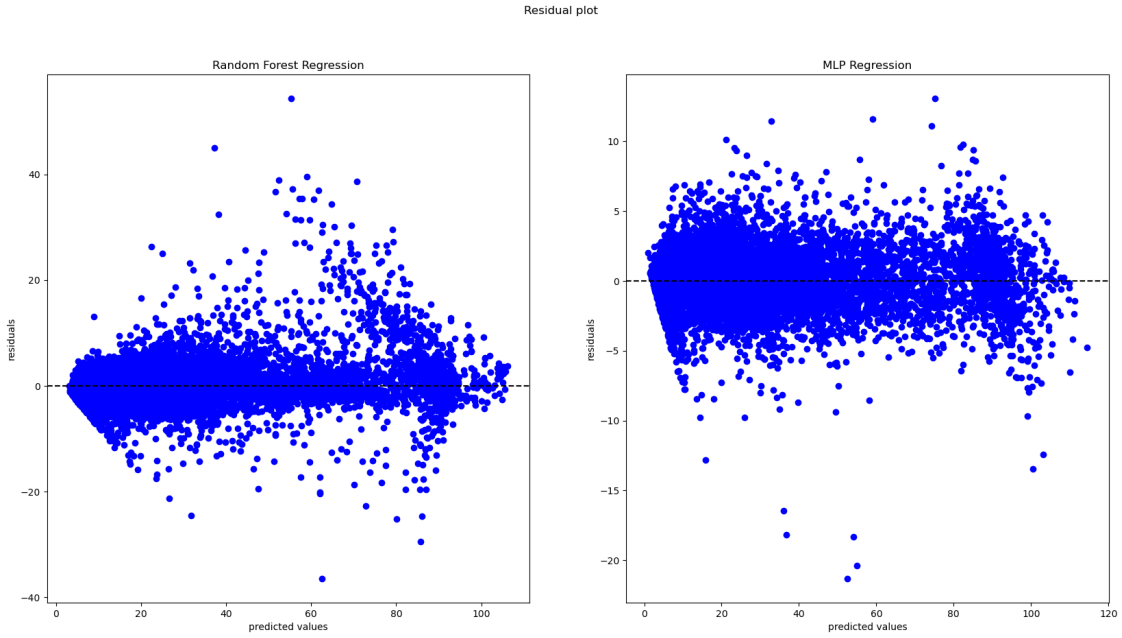
```
    axs[i].set_ylabel('residuals')
plt.show()
```



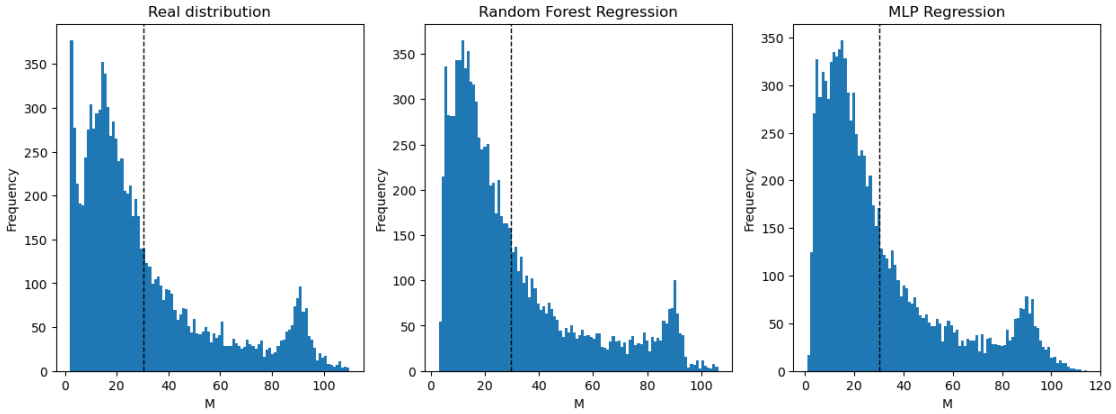
[181]:
```
# compare predicted and actual values
fig, axs = plt.subplots(1,2,figsize=(20,10))
axs[0].scatter(range(y_pred_rfr_test.shape[0]), y_pred_rfr_test, c="red")
axs[0].scatter(range(y_pred_rfr_test.shape[0]), y_test, c="blue")
axs[0].set_title("Random Forest Regression")
axs[1].scatter(range(y_pred_mlp_test.shape[0]), y_pred_mlp_test, c="red")
axs[1].scatter(range(y_pred_mlp_test.shape[0]), y_test, c="blue")
axs[1].set_title("MLP Regression")
plt.show()
```



[182]:
```
# residual plot
residuals_rfr = y_test - y_pred_rfr_test
residuals_mlp = y_test - y_pred_mlp_test
fig, axs = plt.subplots(1,2,figsize=(20,10))
fig.suptitle('Residual plot')
axs[0].scatter(y_pred_rfr_test, residuals_rfr, color='blue')
axs[0].set_title("Random Forest Regression")
axs[1].scatter(y_pred_mlp_test, residuals_mlp, color='blue')
axs[1].set_title("MLP Regression")
for i in range(2):
    axs[i].axhline(y=0, color='black', linestyle='--')
    axs[i].set_xlabel('predicted values')
    axs[i].set_ylabel('residuals')
plt.show()
```



[185]:
```
# show distribution of predicted and actual values
fig, axs = plt.subplots(1, 3, figsize = (15,5))
axs[0].hist(y_test, bins=100)
axs[0].axvline(y_test.mean(), color='k', linestyle='dashed', linewidth=1)
axs[0].set_title('Real distribution')
axs[1].hist(y_pred_rfr_test, bins=100)
axs[1].axvline(y_pred_rfr_test.mean(), color='k', linestyle='dashed',␣
 ↪linewidth=1)
axs[1].set_title('Random Forest Regression')
axs[2].hist(y_pred_mlp_test, bins=100)
axs[2].axvline(y_pred_mlp_test.mean(), color='k', linestyle='dashed',␣
 ↪linewidth=1)
axs[2].set_title('MLP Regression')
for i in range(3):
    axs[i].set_xlabel('M')
    axs[i].set_ylabel('Frequency')
plt.show()
```


Real distribution
Random Forest Regression
MLP Regression
Frequency
M