RHI Magnesita Data Science Test

In [1]:
```
# Import libraries
# Data wrangling
import pandas as pd
import numpy  as np

# Data visualisation
import seaborn           as sns
import matplotlib.pyplot as plt

# Machine learning
from sklearn.preprocessing   import StandardScaler
from sklearn.decomposition   import PCA
from sklearn.compose         import make_column_transformer
from sklearn.pipeline        import make_pipeline
from sklearn.model_selection import train_test_split
from sklearn.linear_model    import LinearRegression
from sklearn.ensemble        import GradientBoostingRegressor
from sklearn.ensemble        import RandomForestRegressor
from sklearn.tree            import DecisionTreeRegressor
from sklearn.impute          import KNNImputer
from sklearn.metrics         import mean_squared_error

# Remove warnings
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
# Data Engineering / ETL

# pipeline.py
import pandas as pd

def process_data(data1: pd.DataFrame, data2: pd.DataFrame) -> pd.DataFrame:
    '''
    Function to read and merge the datasets
    '''

    ## Merge dataframes. For that we must define two keys, with unique ident
    # 1. 'Unamed: 0' column (from 'md_raw_dataset.csv' dataset), renamed as
    #    the 'md_target_dataset.csv' data set;
    # 2. 'groups' column is the second key, wich apears in both datasets.
    data1['index'] = data1['Unnamed: 0']

    data = pd.merge(data1, data2, on=['index', 'groups'])

    ## Droping unwanted columns
    data = data.drop(columns=['Unnamed: 0', 'index'])

    return data

def run_etl_pipeline(raw_data_input_path: str, target_data_input_path: str,
    '''
    Function to run Data Engineering / ETL and write final data
    '''

    # Import and read data
    data1 = pd.read_csv(raw_data_input_path,    sep=';')
    data2 = pd.read_csv(target_data_input_path, sep=';')
```

```
    data = process_data(data1, data2)

    if save:
        data.to_csv(output_path, index = False)

    return data
```

In [84]:
```
def preprocessing_data_pipeline(data, test_size=0.2):
    '''
    A wrapping funtion to evaluate the whole step of data preprocessing, tha

    1. Spliting the data in Train and Test Datasets
    2. Dropping unwanted columns
    3. Data imputation
    4. Data normalization
    5. Data dimensionality reduction
    6. Creating column transformer
    7. Creating machine learning pipeline, given by the set of algorithms:
       - Linear Regression
       - Gradient Boosting Regression
       - Decision Tree Regression
       - Random Forest Regression

    Inputs: - data: output from function 'run_etl_pipeline'
            - test_size: a percentage to split dataset into train and test
              test_size must be in the range [0,1] (default = 0.2)

    Outputs: - X_train
             - X_test
             - y_train
             - y_test
             - pipelines_list

    '''

    # Train test split

    X = data.iloc[:, :-1]
    y = data.iloc[:, -1]
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = te

    # Considering numerical features only

    numerical_features = [c for c, dtype in zip(X.columns, X.dtypes) if dtyp

    ## Creating cloumn transformer for data preprocessing step
    # 1. Data imputation via KNN
    # 2. Standard scaler
    # 3. Reducing dimensionality via PCA

    preprocessor = make_column_transformer(
                                           (make_pipeline(KNNImputer(n_neigh
                                                          , StandardScaler()
                                                          , PCA())
                                           , numerical_features)
                                           )

    ## Create pipeline for machine learning models
    # 1. Preprocessing
    # 2. Training respective models

    linearregression = make_pipeline(preprocessor
                                     , LinearRegression()
                                      )
```

```
    gradientboostingregression = make_pipeline(preprocessor
                                               , GradientBoostingRegressor()
                                                )

    decisiontreeregression = make_pipeline(preprocessor
                                           , DecisionTreeRegressor()
                                            )

    randomforestregression = make_pipeline(preprocessor
                                           , RandomForestRegressor(max_depth=1
                                            )

    # Defining the pipelines in a list
    pipelines_list = [linearregression
                    , gradientboostingregression
                    , decisiontreeregression
                    , randomforestregression]

    return X_train, X_test, y_train, y_test, pipelines_list
```

In [85]:

```
def run_machine_learning_pipeline(X_train, X_test, y_train, y_test, pipeline
    '''
    A wrapping function for training and validating models and allowing
    the choice of the best algorithm to perform the prediction

    Inputs: - X_train
            - X_test
            - y_train
            - y_test
            - pipelines_list

    Output: - best_model: the name of the best model chosen by the
                          algortihm by the minimum rmse criteria
            - idx_best_model: index of the list 'pipelines_list'
                              that contains the best model

    '''


    ## Models training and validation

    # Creating dictionary of pipelines and training models
    PipelineDict = {0: 'Linear Regression'
                  , 1: 'Gradient Boosting Regression'
                  , 2: 'Decision Tree Regression'
                  , 3: 'Random Forest Regression'
                   }

    # Defining variable to choose the best model
    rmse = np.zeros(4)

    # Fit the pipelines and make predictions (over the train datset) to eval
    # root mean squared error in order to choose the best model
    for i, model in enumerate(pipelines_list):
        fittedmodel = model.fit(X_train, y_train)
        y_test_pred = fittedmodel.predict(X_test)
        rmse[i] = np.sqrt(mean_squared_error(y_test, y_test_pred))
        print("{} RMSE: {:.3f}".format(PipelineDict[i], rmse[i]))
    rmse = rmse.tolist()
    idx_best_model = rmse.index(min(rmse))
    print('\nModel with minimum RMSE: {}'.format(PipelineDict[idx_best_model

    best_model = PipelineDict[idx_best_model]
```

```
    return best_model, idx_best_model
```

In [86]:
```
# Run the Data Engineering / ETL pipeline

raw_data_input_path    = '/Users/lucasmassaroppe/Documents/rhi_magnesita/md_
target_data_input_path = '/Users/lucasmassaroppe/Documents/rhi_magnesita/md_
output_path            = '/Users/lucasmassaroppe/Documents/rhi_magnesita/md_

data = run_etl_pipeline(raw_data_input_path, target_data_input_path, output_
```

In [87]:
```
# Run the Preprocessing Data pipeline

X_train, X_test, y_train, y_test, pipelines_list = preprocessing_data_pipeli
```

In [88]:
```
# Run the Machine Learning pipeline

best_model, idx_best_model = run_machine_learning_pipeline(X_train, X_test,
```

```
Linear Regression RMSE: 55.809
Gradient Boosting Regression RMSE: 55.774
Decision Tree Regression RMSE: 76.286
Random Forest Regression RMSE: 55.304

Model with minimum RMSE: Random Forest Regression
```

As we can see, method Decision Tree Regression has the lowest RMSE and, therefore, is chosen among the four tested to perform the regression.

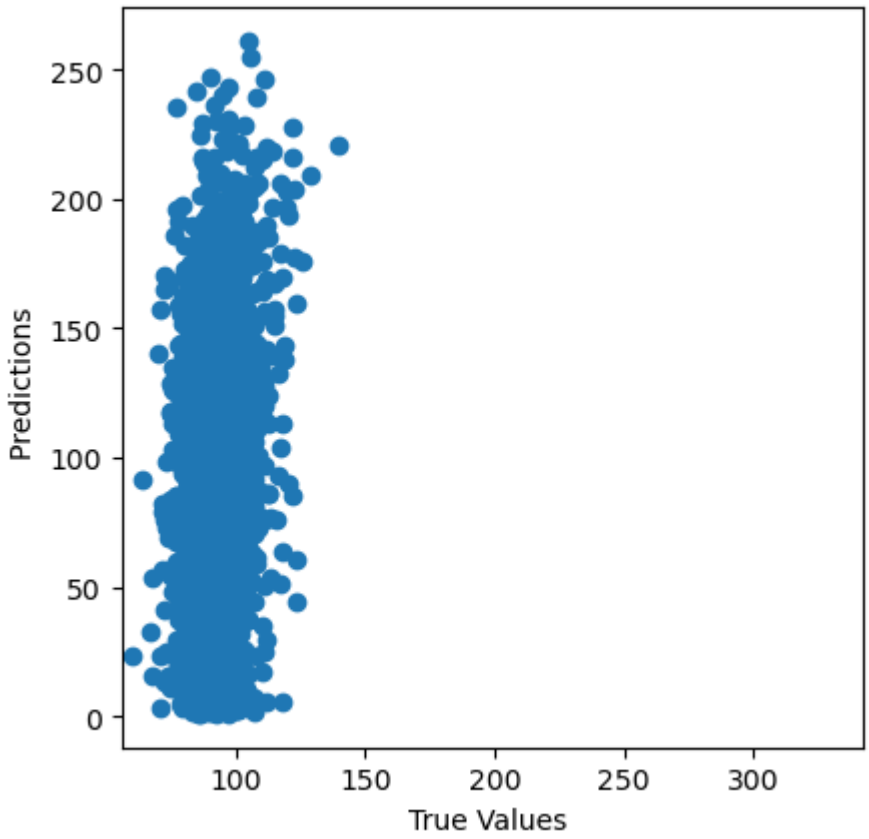
In [91]:
```
# Fitting the model
fitted_model = pipelines_list[idx_best_model].fit(X_train, y_train)

# Predicting the fitted model
y_test_predicted = fitted_model.predict(X_test)

# Evaluating the RMSE of the training set
rmse_random_forest_regression = np.sqrt(mean_squared_error(y_test, y_test_pr
```

In [94]:
```
# Plotting the predictions

plt.scatter(y_test_predicted, y_test)
plt.xlabel('True Values ')
plt.ylabel('Predictions ')
plt.axis('equal')
plt.axis('square')
plt.show()
```



In order to improve the predictions of the model, it is proposed, in a future and more detailed work, the use of categorical features and, also, the use of the optimization of the algorithms' hyperparameters through grid search or even Bayesian optimization.

In [ ]: