In [1]:
```
#Imports all needed libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import statistics
import seaborn as sns
import category_encoders as ce
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.pipeline import Pipeline
from sklearn import linear_model
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import statsmodels.api as sm
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
```

In [2]:
```
#Imports ASM and purchase order datasets
omaha_df = pd.read_csv(
    "C:/Users/predi/Documents/GitHub/DSC680\Assignments/Project 2/Datasets/Omaha_regression model data for ba
vernon_df = pd.read_csv(
    "C:/Users/predi/Documents/GitHub/DSC680/Assignments/Project 2/Datasets/Vernon_regression model data for b
purchase_order_df = pd.read_csv(
    "C:/Users/predi/Documents/GitHub/DSC680/Assignments/Project 2/Datasets/Regression model data for bacon-Q1
```

In [3]:
```
#Lowers and strips header informations for the dataframes. Also repalces spaces with underscores
omaha_df.columns = omaha_df.columns.str.lower().str.strip()
omaha_df.columns = omaha_df.columns.str.replace(' ', '_')
vernon_df.columns = vernon_df.columns.str.lower().str.strip()
vernon_df.columns = vernon_df.columns.str.replace(' ', '_')
purchase_order_df.columns = purchase_order_df.columns.str.lower().str.strip()
purchase_order_df.columns = purchase_order_df.columns.str.replace(' ', '_')
```

In [4]:
```
#drops columns that aren't needed in Omaha df and renames columns as needed to match Vernon df
omaha_df.drop (columns =
               ['line(s)', 'product_type', 'gross_lbs/hr', 'slices_per_package_(avg)',
                'cases_per_pallet', 'slices_per_package_+_settling_slices', 'idle_cuts',
                'slice_efficiency_factor'], axis=1, inplace=True)
omaha_df = omaha_df.rename(columns={'new_product_code': 's4_product_code'})
omaha_df = omaha_df.rename(columns={'#1_yield_%': '#1_yield'})
omaha_df = omaha_df.rename(columns={'slices_per_lb_(avg)': 'slices_per_lb'})
```

In [5]:
```
#Splits columns to match Omaha df columns and drops columns that aren't needed.
vernon_df[['slice_length_lower_(in)', 'slice_length_upper_(in)']] = vernon_df[
    'slice_length_(in)'].str.split('-', 1, expand=True)
vernon_df[['slice_width_lower_(in)', 'slice_width_upper_(in)']] = vernon_df[
    'slice_width_(in)'].str.split('-', 1, expand=True)
vernon_df.drop (columns =
                ['line', 'slice_length_(in)', 'slice_width_(in)', 'gross_tp_lbs/hr',
                 'maximum_rpm', 'slicer_yield', 'slices_per_inch'], axis=1, inplace=True)
```

```
C:\Users\predi\AppData\Local\Temp\ipykernel_18628\3926068560.py:2: FutureWarning: In a future version of pand
as all arguments of StringMethods.split except for the argument 'pat' will be keyword-only.
  vernon_df[['slice_length_lower_(in)', 'slice_length_upper_(in)']] = vernon_df[
C:\Users\predi\AppData\Local\Temp\ipykernel_18628\3926068560.py:4: FutureWarning: In a future version of pand
as all arguments of StringMethods.split except for the argument 'pat' will be keyword-only.
  vernon_df[['slice_width_lower_(in)', 'slice_width_upper_(in)']] = vernon_df[
```

In [6]:
```
#Drops all columns that will not be used, and renames columns to match ASM df's to be joined later.
purchase_order_df.drop (columns =
                        ['cost_center', 'order', 'week_ending_date', 'actual_process_yield',
                         'target_process_yield', 'actual_by_product_lbs', 'target_by_product_lbs',
                         'actual_seconds_lbs', 'target_seconds_lbs', 'actual_production_quantity'],
                        axis=1, inplace=True)
purchase_order_df = purchase_order_df.rename(columns={'material_number': 's4_product_code'})
```

In [7]:
```
#creates two new columns that show difference between reported and targets
purchase_order_df['diff_yield'] = purchase_order_df[
```

```
    'target_yield'] - purchase_order_df['actual_yield']
purchase_order_df['diff_primary_yield'] = purchase_order_df[
    'target_primary_yield'] - purchase_order_df['actual_primary_yield']

#Calculates the standard diviation and means of the yields.
sd_yield = statistics.stdev(purchase_order_df['diff_yield'])
mean_yield = statistics.mean(purchase_order_df['diff_yield'])
sd_primary = statistics.stdev(purchase_order_df['diff_primary_yield'])
mean_primary = statistics.mean(purchase_order_df['diff_primary_yield'])

#creates a new data frame filtering out all recorded yields greater than 2std from the mean.
filtered_po_df = purchase_order_df[(purchase_order_df['diff_yield'] <
                                    (mean_yield + 2*sd_yield)) & (purchase_order_df['diff_yield'] >
                                                                  (mean_yield - 2*sd_yield))]
filtered_po_df = filtered_po_df[(filtered_po_df['diff_primary_yield'] <
                                 (mean_primary + 2*sd_primary)) & (filtered_po_df['diff_primary_yield'] >
                                                                   (mean_primary - 2*sd_primary))]

#creates a data frame with all of the product yields by yield mean
primary_yields = filtered_po_df.groupby(['s4_product_code'])['actual_primary_yield'].mean()
```

In [8]:
```
# combines omaha and vernon data frames. Then inner joins by product code to their histocal primary yields
ASM_df = pd.concat([omaha_df, vernon_df], axis=0)
ASM_df= pd.merge(ASM_df,primary_yields, on='s4_product_code', how='inner')

#Replaces fuzzy categorical matches due to the different dataframes with exact matches
ASM_df['slicer_type'] = ASM_df['slicer_type'].str.replace('Cashin Edge', 'Cashin')
ASM_df['minimum_length_secondary_lean_(in)'] = ASM_df[
    'minimum_length_secondary_lean_(in)'].str.replace('-', '0.01')

#Drops all duplicate product codes
ASM_df = ASM_df.drop_duplicates('s4_product_code')
```

In [9]:
```
#Creates a dataframe to be used for testing current method befor changes are made to ASM_DF
ASMtest=ASM_df[['#1_yield', 'actual_primary_yield']]
```

In [10]:
```
#turns categorical variables into numeric variable
ASM_df = pd.get_dummies(ASM_df, columns = ['plant','slicer_type', 'package_type'])

#converts data types to numeric as needed
ASM_df['slice_length_lower_(in)'] = pd.to_numeric(ASM_df['slice_length_lower_(in)'])
##ASM_df['slice_length_upper_(in)'] = pd.to_numeric(ASM_df['slice_length_upper_(in)'])
##ASM_df['slice_width_lower_(in)'] = pd.to_numeric(ASM_df['slice_width_lower_(in)'])
ASM_df['slice_width_upper_(in)'] = pd.to_numeric(ASM_df['slice_width_upper_(in)'])
ASM_df['minimum_length_secondary_lean_(in)'] = pd.to_numeric(ASM_df['minimum_length_secondary_lean_(in)'])
```

In [11]:
```
#drops columns that will not be used in the model
ASM_df.drop (columns =
             ['s4_product_code', 'description','resource','packaging_style',
              '#1_yield', 'slices_per_lb','slice_width_lower_(in)', 'finished_piece_wt_(lbs)',
              'packages_per_case', 'case_weight_(lbs)', 'package_weight_(lbs)',
              'plant_Vernon', 'slicer_type_Weber 702', 'plant_Omaha','slice_length_upper_(in)'],
             axis=1, inplace=True)

#Drops rows that have null values causing issues
ASM_df = ASM_df.drop(labels = [479, 738, 815, 842], axis =0)
```

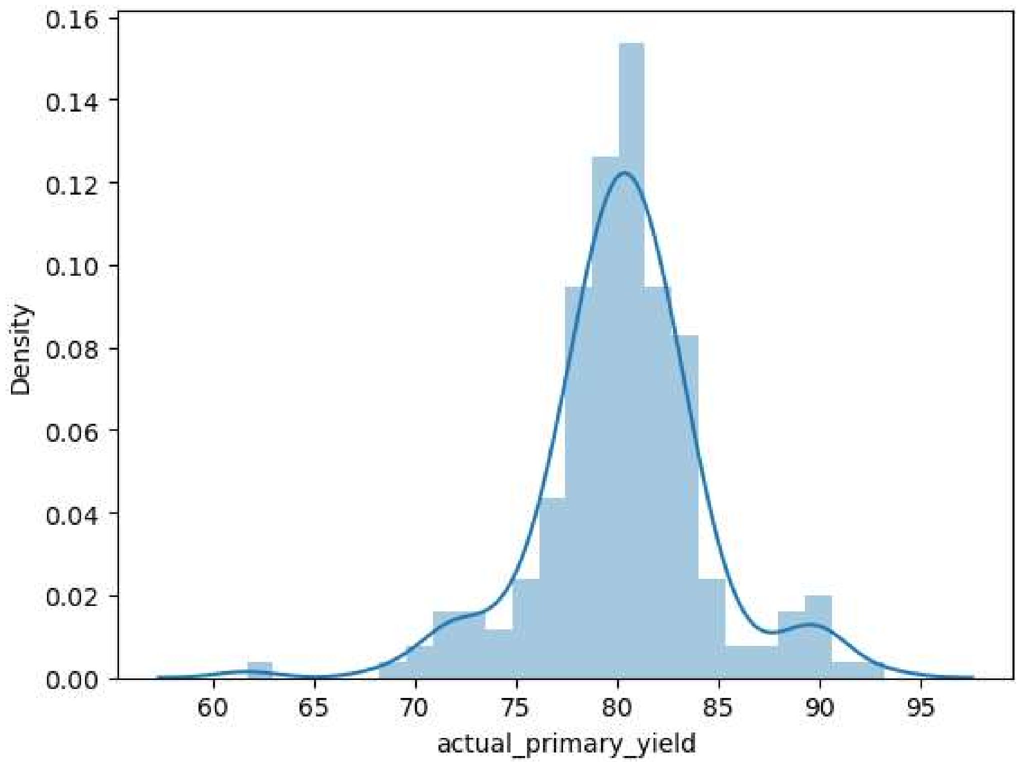
In [12]:
```
#splits the data into training and testing sets
x_data = ASM_df.drop(['actual_primary_yield'], axis = 1)
y_data = ASM_df['actual_primary_yield']
x_train, x_test, y_train, y_test = train_test_split(
    x_data, y_data, test_size=0.30, random_state=10, shuffle=True)
```

In [13]:
```
sns.distplot(ASM_df['actual_primary_yield'])
```

```
C:\Users\predi\AppData\Local\Temp\ipykernel_18628\3593745333.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(ASM_df['actual_primary_yield'])
```

Out[13]:

```
<Axes: xlabel='actual_primary_yield', ylabel='Density'>
```

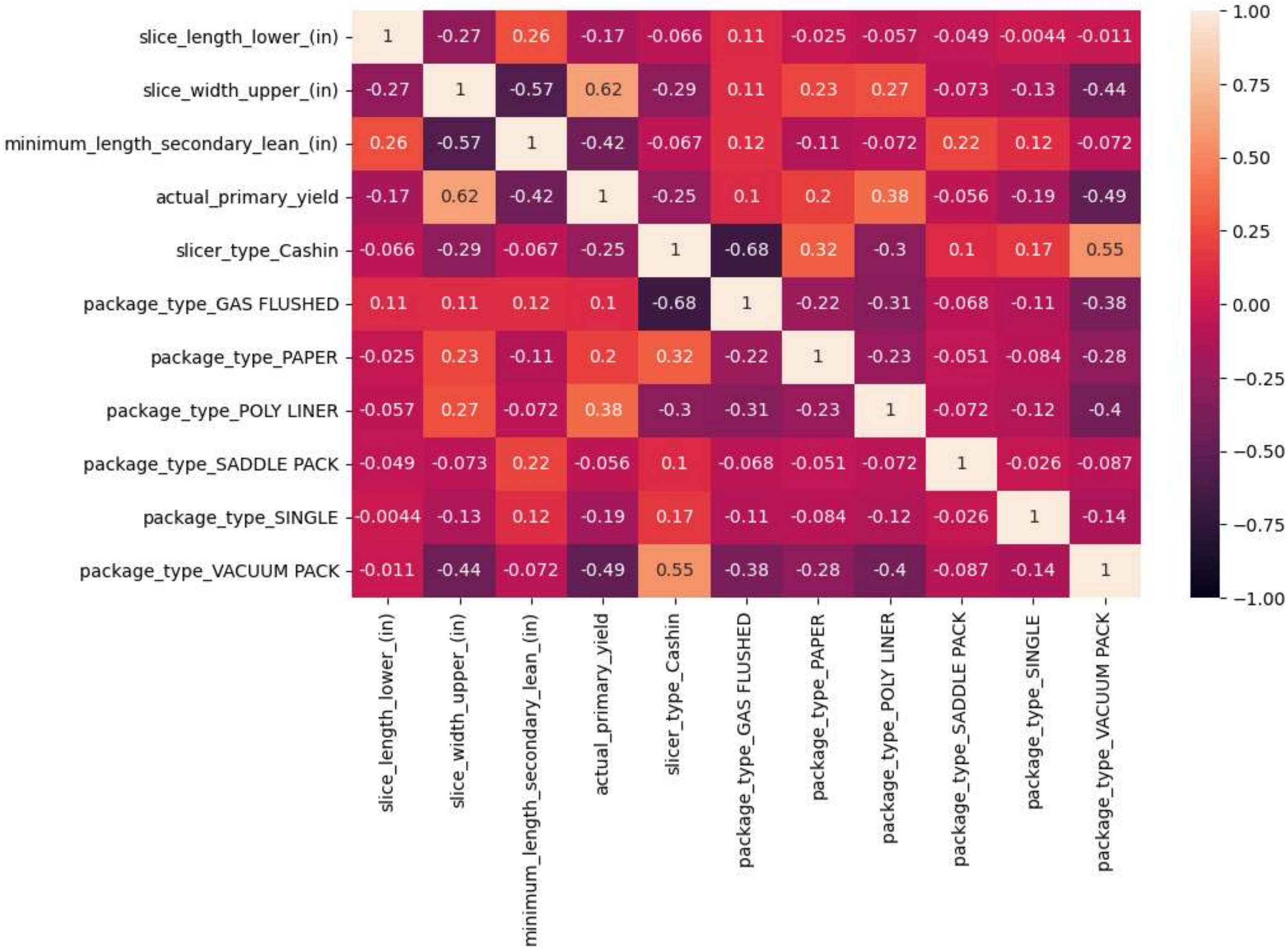


In [14]:

```python
plt.figure(figsize=(10, 6))
sns.heatmap(ASM_df.corr(), vmin=-1, vmax=1, annot=True)
```

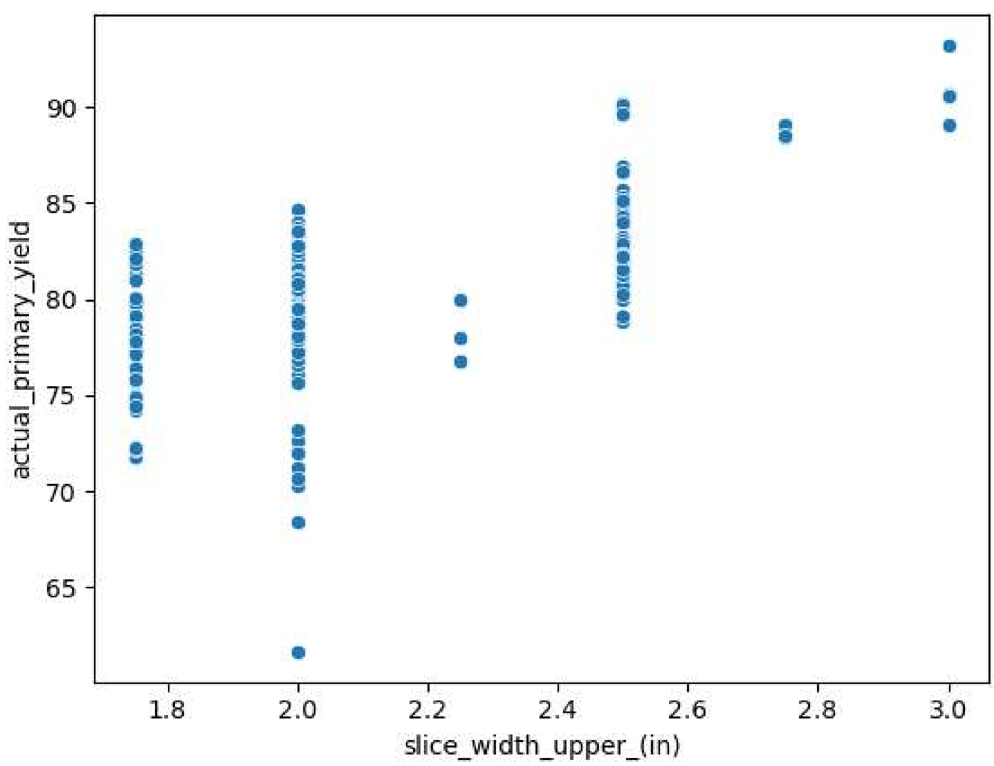
Out[14]:

```
<Axes: >
```



In [15]:
```python
sns.scatterplot(data=ASM_df, y='actual_primary_yield', x='slice_width_upper_(in)',)
```

Out[15]: <Axes: xlabel='slice_width_upper_(in)', ylabel='actual_primary_yield'>



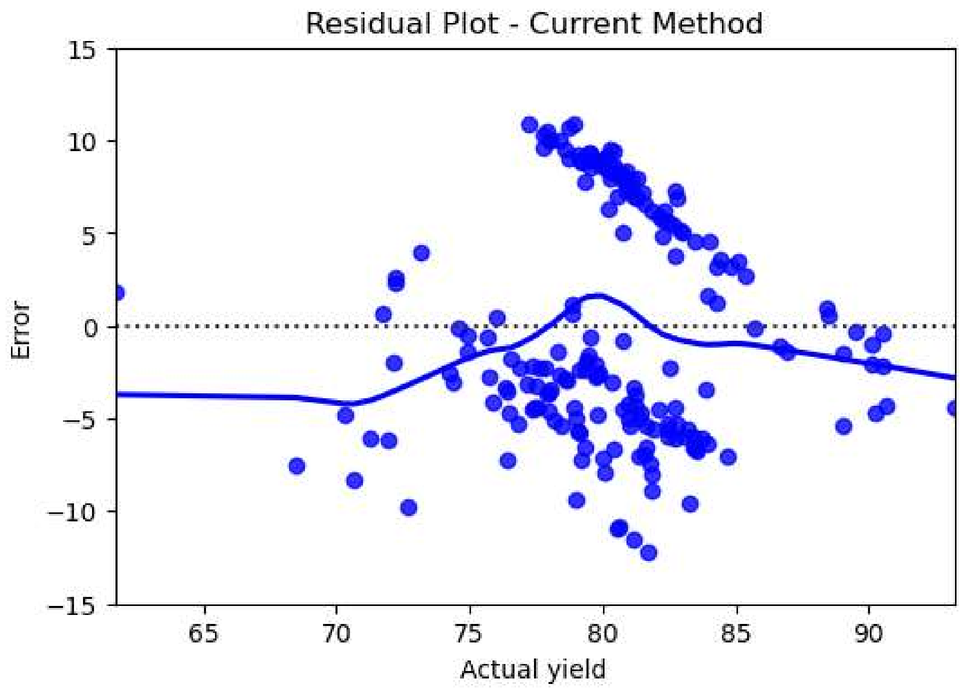
In [16]:
```python
#test current method
ASMtest['#1_yield']=ASMtest['#1_yield']*100
ASMtest = ASMtest.drop(labels = [805,831], axis =0)
```

```
original_x_data = ASMtest['#1_yield']
original_y_data = ASMtest['actual_primary_yield']
print("MAE:",  mean_absolute_error(original_y_data, original_x_data))
print('RMSE: ', np.sqrt(mean_squared_error(original_y_data, original_x_data)))
```

```
MAE: 5.181937075759555
RMSE:  6.4682202609977795
```

In [17]:
```
#Creates resid plot of current method
plt.figure(figsize=(6,4))
sns.residplot(x = original_y_data, y = original_x_data,lowess=True, color="b")
plt.title('Residual Plot - Current Method')
plt.xlabel('Actual yield')
plt.ylabel('Error')
plt.ylim(-15, 15)
```

Out[17]:
```
(-15.0, 15.0)
```



In [18]:
```
# Define the pipeline with a placeholder for the model
pipeline = Pipeline([('model', None)])
```

In [19]:
```
# Define hyperparameters to search for each model
param_grid_lr = {'model': [LinearRegression()],
                 'model__fit_intercept': [True, False],
                 'model__copy_X': [True, False]}
param_grid_rf = {'model': [RandomForestRegressor()],
                 'model__n_estimators': [100, 200, 300, 500],
                 'model__max_depth': [4, 6, 8, 10],
                 'model__min_samples_split': [2, 5, 10],
                 'model__min_samples_leaf': [1, 2, 4],
                 'model__max_features': ['auto', 'sqrt', 'log2']}
param_grid_xg = {'model': [XGBRegressor()],
                 'model__n_estimators': [100, 250, 500],
                 'model__learning_rate': [0.01, 0.1, 0.2],
                 'model__max_depth': [4, 6, 8, 10],
                 'model__subsample': [0.8, 1.0],
                 'model__colsample_bytree': [0.8, 1.0],
                 'model__gamma': [0, 0.1, 0.2]}
#combines all of the parameters
param_grids = [param_grid_lr, param_grid_xg, param_grid_rf]
```

In [20]:
```
#Iterates through each model and performs the grid search
for param_grid in param_grids:
    grid_search = GridSearchCV(pipeline, param_grid, cv=2,
                               scoring='neg_root_mean_squared_error', n_jobs=-1)
    grid_search.fit(x_train, y_train)
```

In [21]:
```
#Gets the best model from the grid search
best_model = grid_search.best_estimator_

# Fit the best model on the full training set
best_model.fit(x_train, y_train)
```

Out[21]:

```
▸ Pipeline
▸ RandomForestRegressor
```

In [22]:
```
#Prints the best model and its hyperparmeters
print(grid_search.best_params_)
```

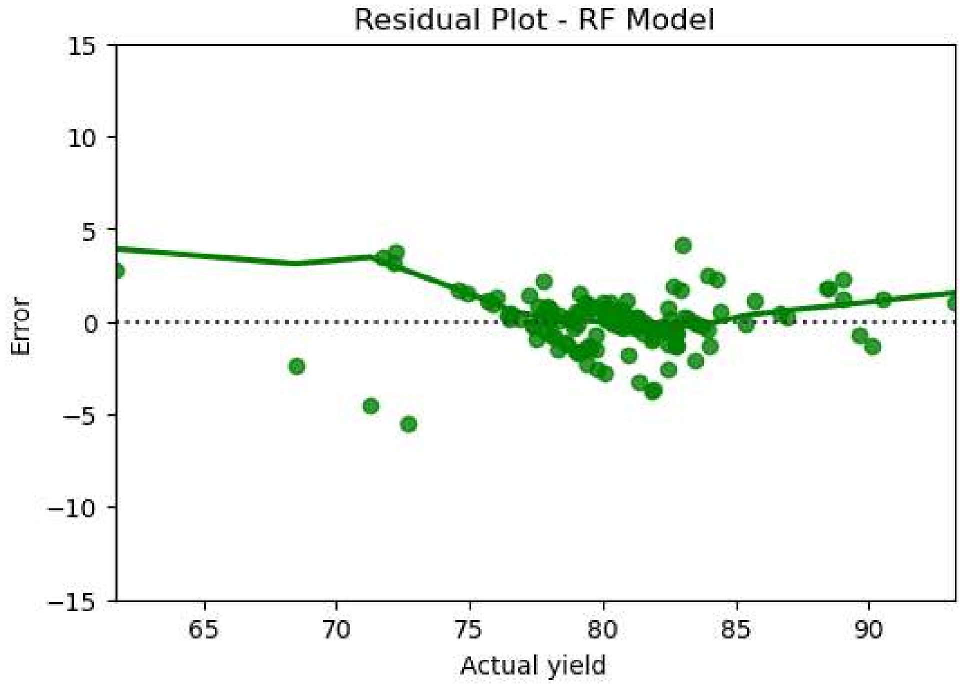
```
{'model': RandomForestRegressor(max_depth=6, max_features='log2', n_estimators=300), 'model__max_depth': 6, 'model__max_features': 'log2', 'model__min_samples_leaf': 1, 'model__min_samples_split': 2, 'model__n_estimat
ors': 300}
```

In [23]:
```
#uses the model to predict the test set
ytrain_pred = best_model.predict(x_train)
ytest_pred = best_model.predict(x_test)

#calculates/prints R2 score, MAE, and RMSE
print("---------------Training Data----------------------")
print("MAE:",  mean_absolute_error(y_train, ytrain_pred))
print('RMSE: ', np.sqrt(mean_squared_error(y_train, ytrain_pred)))
print("----------------Testing Data-----------------------")
print('R-Squared:', r2_score(y_test, ytest_pred))
print("MAE:",  mean_absolute_error(y_test, ytest_pred))
print('RMSE: ', np.sqrt(mean_squared_error(y_test, ytest_pred)))
```

```
---------------Training Data----------------------
MAE: 1.287383960456411
RMSE:  1.777884157180191
----------------Testing Data-----------------------
R-Squared: 0.7046126162199418
MAE: 1.7017796397840315
RMSE:  2.2980504902714554
```

In [24]:
```
plt.figure(figsize=(6,4))
sns.residplot(x = y_train, y = ytrain_pred,lowess=True, color="g")
plt.title('Residual Plot - RF Model')
plt.xlabel('Actual yield')
plt.ylabel('Error')
plt.ylim(-15, 15)
```

Out[24]:
```
(-15.0, 15.0)
```



In [25]:

```
test = pd.DataFrame({'Predicted value':ytest_pred, 'Actual value':y_test})
fig= plt.figure(figsize=(16,8))
test = test.reset_index()
test = test.drop(['index'],axis=1)
plt.plot(test[:150])
plt.legend(['Actual value','Predicted value'])
```

Out[25]: <matplotlib.legend.Legend at 0x2451f947520>