MECE 6397: Project 5

Syam Evani
08/12/2025

Introduction

Table of Contents

My code for this project is included in three ways:

- Embedded in portions with relevant discussion
- Uploaded as part of this deliverable on Canvas
- Available on a public github here: https://github.com/Sam-v6/mece-6397-doe/tree/main/project5

Part 1: Factor Design

I created a three factor design with 10 levels each to describe various combinations of a possible gaseous oxygen and gaseous methane rocket engine. This rocket engine is defined with:

- Chamber pressure in psia (Pc)
- The mixture ratio defined as ratio of oxygen mass to fuel mass into the thruster combustion chamber (MR)
- Area ratio defined as the ratio of the nozzle exit area to the throat area of the engine (e)

I varied the factors across the following ranges:

- Chamber pressure (50 - 500 psia with 10 values each)
- Mixture ratio (1 - 10 with 10 values each)
- Area ratio (10 - 100 with 10 values each)

Creating the factorial design for these factors and levels sweeps out 1000 combinations, enumerating the many possiblities of a gaseous oxygen and gasesous methane thruster.

In [ ]:
```
"""
Purpose: Project 5
Details: Conducts design of experiments with Pc, MR, and area ratio for GOX/GCH4 th
Author: Syam Evani
"""

# Standard imports
import os
import random

# General imports
import numpy as np
import itertools
import pandas as pd
from rocketcea.cea_obj import CEA_Obj

# Plotting imports
import seaborn as sns
import matplotlib.pyplot as plt

# ML utils
from scipy.interpolate import griddata
from scipy.stats import f_oneway
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler

# Regressors imports
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import GradientBoostingRegressor, AdaBoostRegressor
from xgboost import XGBRegressor
from catboost import CatBoostRegressor
from sklearn.linear_model import ElasticNet, Lasso, Ridge, BayesianRidge

# NN imports
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping

# Local imports
# None

#-----------------------------------------------------------
# Factor design
#-----------------------------------------------------------
# Three factors
factors = ["pc", "mr", "e"]
```

```
pc = np.linspace(50, 500, 10)   # [psia] Chamber pressure
mr = np.linspace(1, 10, 10)     # [--]   Ratio of ox/fuel mass
e = np.linspace(10, 100,10)     # [--]   Ratio of exit area to throat area

# Generate all possible combinations
combinations = list(itertools.product(pc, mr, e))
```

Part 2: Generating the Dataset

Because I wanted to evaluate a full factorial design in the context of a rocket engine's design parameters, thruster test data as one might expect is not readily available online. To substitute for this I used a NASA developed tool called Chemical Equilibrium Analysis (CEA) which will calculate chemical equilibrium product concentrations from any set of reactants and determine thermodynamic and transport properties for the product mixture. A specific built in application calculates theoretical rocket performance and is a common tool used in industry to gauge the theoretical performance of a rocket engine.

A great group of open source collaborators have developed a python wrapper for CEA (and many other useful rocketry calculations) with `CoolProp`, which I conveniently use in this project. Providing the chamber pressure, mixture ratio, and area ratio from each design combination I can use the CEA wrapper to calculate the output parameter of interest, the specific impulse (Isp). Specific impulse can be thought of as the fuel efficiency of the rocket as it is defined as the thrust of the engine divided by the product of the rocket's mass flow rate and Earth's gravity constant. Note the units of Isp are in seconds.

I provide each combination's Pc, MR, and e to calculate the specific impulse. To somewhat simulate if this data was instead collected from testing rather than analytically calculated, I apply a random offset to each calculated value (between -10 and 10) to add some noise to this measurement. While the offset value may not be precise, this noise is meant to capture the error that would accumulate from measuring the thrust of an engine from a load cell, the mass flow rate from a mass flow meter, and then calculating the Isp from these two measured parameters.

I then save this measured output as an addition to the full factorial design in a .txt file. As an example I also print the first 5 entries of this text file so readers can get a feel for the design factors with the output parameter.

In [ ]:
```
#-------------------------------------------------------
# Generate data and visuals to show the data
#-------------------------------------------------------
results = []
random.seed(42)
C = CEA_Obj( oxName='O2', fuelName='CH4')
for i, design in enumerate(combinations):
    isp = C.get_Isp(Pc=design[0], MR=design[1], eps=design[2])   # [s] Specific imp
```

```
    isp = isp + random.randint(-10,10)                          # Add some randomn
    results.append((design[0], design[1], design[2], isp))

# Convert results to a pandas df and output to text file
isp_df = pd.DataFrame(results, columns=['pc', 'mr', 'e', 'isp'])
txt_file_path = os.path.join(os.getenv('USERPROFILE'), 'repos', 'mece-6397-doe', 'p
with open(txt_file_path, 'w') as file:
    file.write(isp_df.to_string(index=False))

for i in range(0,5):
    print(results[i])
```

```
(50.0, 1.0, 10.0, 251.94362793060537)
(50.0, 1.0, 20.0, 248.4097570193321)
(50.0, 1.0, 30.0, 252.36748459023272)
(50.0, 1.0, 40.0, 264.95475194792647)
(50.0, 1.0, 50.0, 267.3315786173212)
```

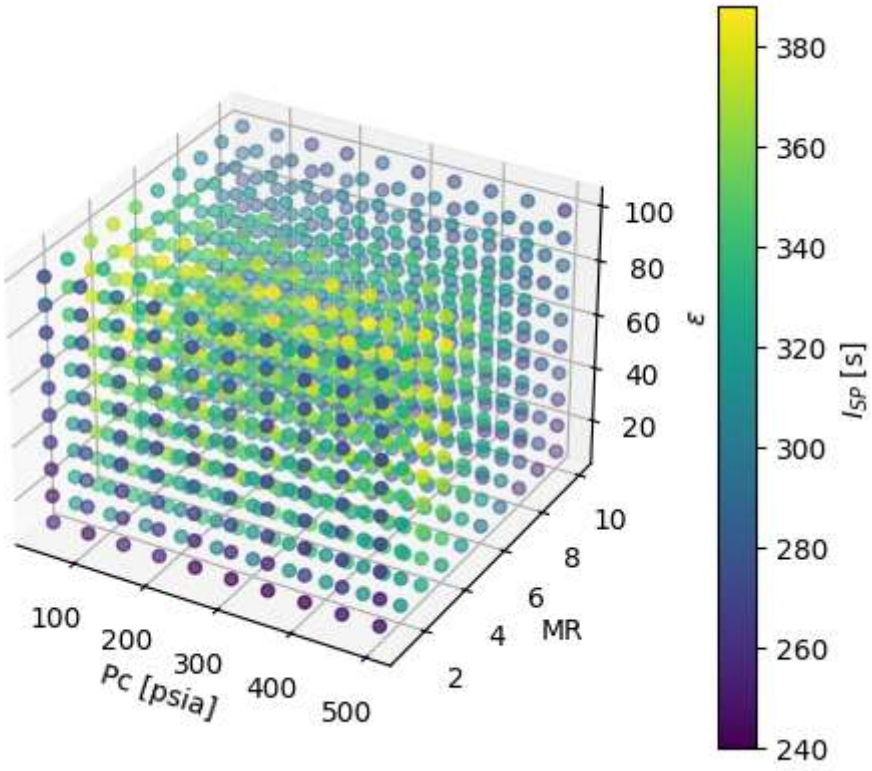
Additionally, I then create some plots to visualize the data including:

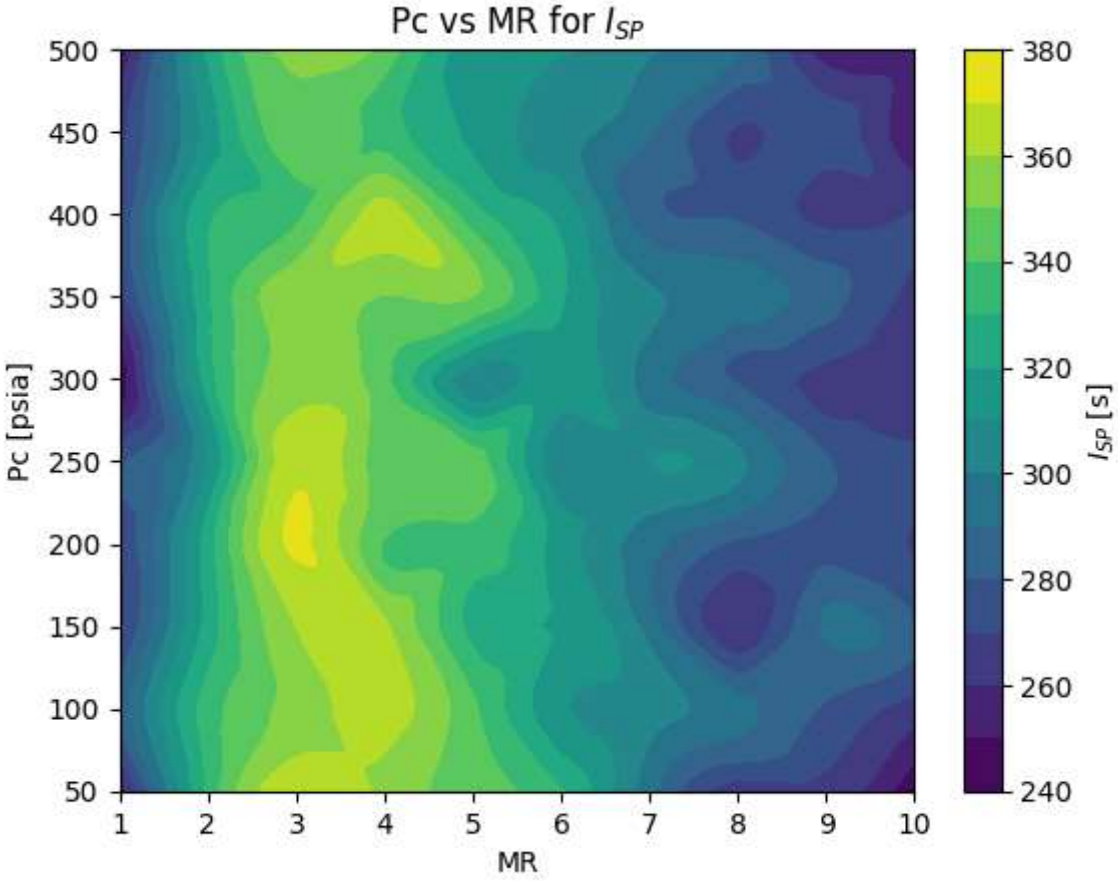
- A 3D plot showing Isp in colors against chamber pressure, mixture ratio, and area ratio
- A contour plot of chamber pressure vs mixture ratio showing specific impulse
- A contour plot of area ratio vs mixture ratio showing specific impulse

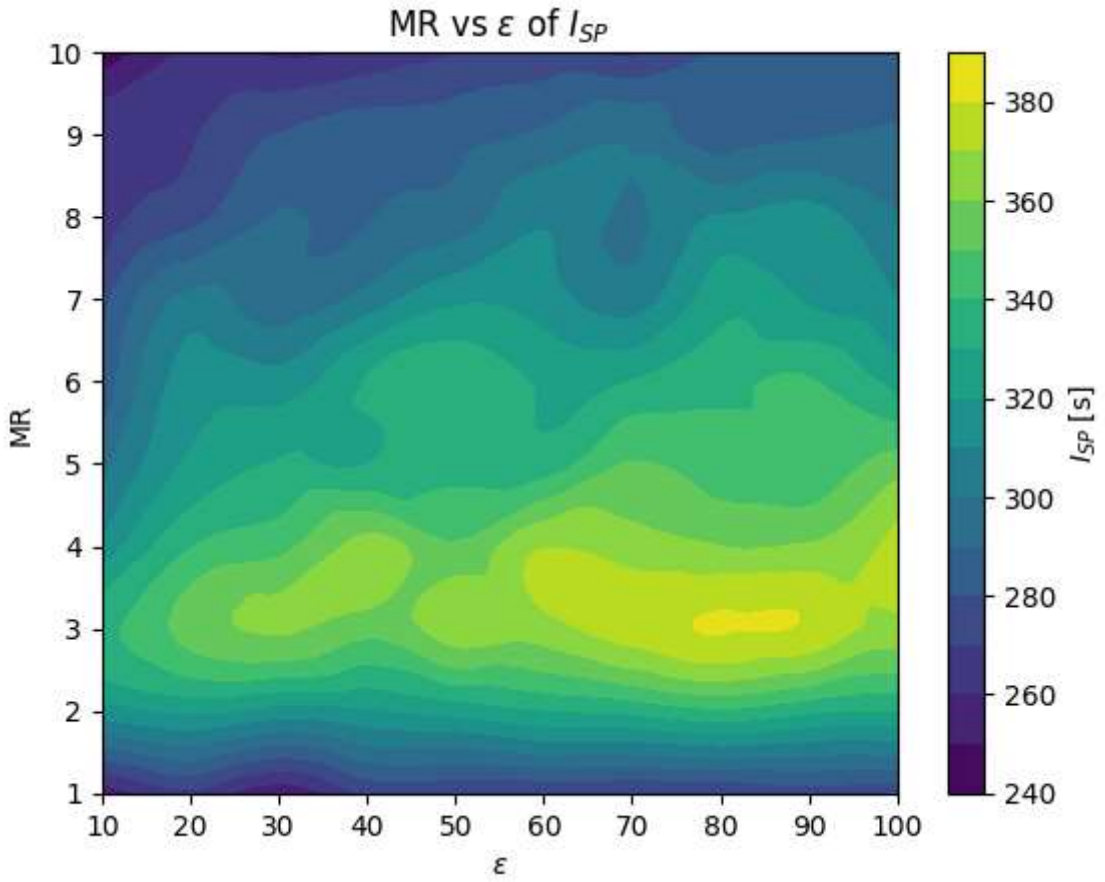
I also print some summary statistics now on our complete dataset.

In [ ]:
```
# Make scatter plot of data
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
sc = ax.scatter(isp_df['pc'], isp_df['mr'], isp_df['e'], c=isp_df['isp'], cmap='vir
plt.colorbar(sc, label=r'$I_{SP}$ [s]')
ax.set_xlabel('Pc [psia]')
ax.set_ylabel('MR')
ax.set_zlabel(r'$\epsilon$')
ax.set_box_aspect(aspect=None, zoom=0.9)
plt.savefig(os.path.join(os.getenv('USERPROFILE'),'repos', 'mece-6397-doe', 'projec
plt.show()
plt.close()

# Contour plot: Pc vs MR
xi = np.linspace(isp_df['mr'].min(), isp_df['mr'].max(), 100)
yi = np.linspace(isp_df['pc'].min(), isp_df['pc'].max(), 100)
zi = griddata((isp_df['mr'], isp_df['pc']), isp_df['isp'], (xi[None, :], yi[:, None
plt.contourf(xi, yi, zi, levels=14, cmap='viridis')
plt.colorbar(label=r'$I_{SP}$ [s]')
plt.xlabel('MR')
plt.ylabel('Pc [psia]')
plt.title(r'Pc vs MR for $I_{SP}$')
plt.savefig(os.path.join(os.getenv('USERPROFILE'),'repos', 'mece-6397-doe', 'projec
plt.show()
plt.close()

# Contour plot: e vs MR
xi = np.linspace(isp_df['e'].min(), isp_df['e'].max(), 100)
```

```
yi = np.linspace(isp_df['mr'].min(), isp_df['mr'].max(), 100)
zi = griddata((isp_df['e'], isp_df['mr']), isp_df['isp'], (xi[None, :], yi[:, None]
plt.contourf(xi, yi, zi, levels=14, cmap='viridis')
plt.colorbar(label=r'$I_{SP}$ [s]')
plt.xlabel(r'$\epsilon$')
plt.ylabel('MR')
plt.title(r'MR vs $\epsilon$ of $I_{SP}$')
plt.savefig(os.path.join(os.getenv('USERPROFILE'),'repos', 'mece-6397-doe', 'projec
plt.show()
plt.close()

# Generate summary statistics
summary_stats = isp_df.describe(include='all')
print(summary_stats)
```




Pc vs MR for I_{SP}
Pc [psia]
MR
I_{SP} [s]
500
450
400
350
300
250
200
150
100
50
1
2
3
4
5
6
7
8
9
10
380
360
340
320
300
280
260
240


MR vs ε of I_{SP}
MR
ε
I_{SP} [s]
10
9
8
7
6
5
4
3
2
1
10
20
30
40
50
60
70
80
90
100
380
360
340
320
300
280
260
240

	pc	mr	e	isp
count	1000.000000	1000.000000	1000.000000	1000.000000
mean	275.000000	5.500000	55.000000	315.038473
std	143.685927	2.873719	28.737185	35.109296
min	50.000000	1.000000	10.000000	239.888619
25%	150.000000	3.000000	30.000000	285.065014
50%	275.000000	5.500000	55.000000	314.541415
75%	400.000000	8.000000	80.000000	342.640304
max	500.000000	10.000000	100.000000	387.897207

Part 3: Factor Analysis

For each main factor (Pc, MR, e) I perform a traditional analysis where I group the factor and specific impulse and take the mean. This provides numerous means for each 10 levels which isn't very revealing on the actual main effect factors.

As an alternate approach, I use scipy's stats module that has ANOVA capability similar to JPM and can assess the main factors. Those results are shown below:

Variable	F-value	p-value
pc	0.17657876054005522	0.9963914948007738
mr	663.3631865619716	0.0
e	13.46003934613545	1.6324343067921846e-20

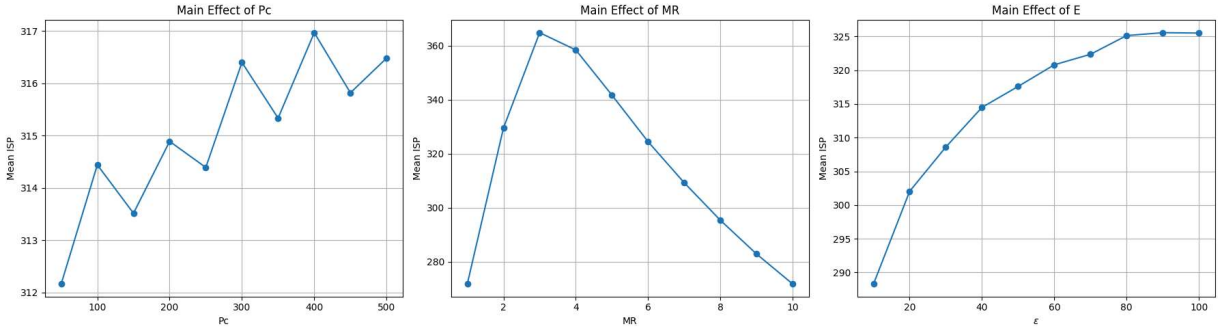
As seen above, chamber pressure does not have a statistically significant impact on specific impulse. However, both mixture ratio and area ratio are statistically significant (p values less than 0.05), with mixture ratio having a strong impact on the specific impulse of the thruster as values change. From prior knowledge this is true given that as the chemical ratio of the oxygen and methane reaction changes, the temperature of combustion will change along with the mass of the exhaust gasses, with both parameters ultimately comprising the specific impulse.

I also created a subplot to better visualize the main factors impacting the specific impulse, highlighting the general trend of area ratio impacting specific impulse and the strong impact of mixture ratio on specific impulse (especially at ideal mixture ratios).

In [ ]:
```
#----------------------------------------------------------------------
# Assess the impact of main factors
#----------------------------------------------------------------------
main_effects = {}
for factor in factors:
    main_effects[factor] = isp_df.groupby(factor)['isp'].mean()

# One-Way ANOVA for each factor
factors = ['pc', 'mr', 'e']
```

```
for factor in factors:
    groups = [isp_df[isp_df[factor] == level]['isp'] for level in isp_df[factor].un
    f_val, p_val = f_oneway(*groups)
    print(f"ANOVA for {factor}: F-value = {f_val}, p-value = {p_val}")

# Plotting the main effects
fig, axs = plt.subplots(1, 3, figsize=(18, 5))
# Plot for pc
axs[0].plot(main_effects['pc'].index, main_effects['pc'].values, marker='o')
axs[0].set_title('Main Effect of Pc')
axs[0].set_xlabel('Pc')
axs[0].set_ylabel('Mean ISP')
axs[0].grid()
# Plot for mr
axs[1].plot(main_effects['mr'].index, main_effects['mr'].values, marker='o')
axs[1].set_title('Main Effect of MR')
axs[1].set_xlabel('MR')
axs[1].set_ylabel('Mean ISP')
axs[1].grid()
# Plot for e
axs[2].plot(main_effects['e'].index, main_effects['e'].values, marker='o')
axs[2].set_title('Main Effect of E')
axs[2].set_xlabel(r'$\epsilon$')
axs[2].set_ylabel('Mean ISP')
axs[2].grid()
# Layout
plt.tight_layout()
plt.savefig(os.path.join(os.getenv('USERPROFILE'),'repos', 'mece-6397-doe', 'projec
plt.show()
plt.close()
```

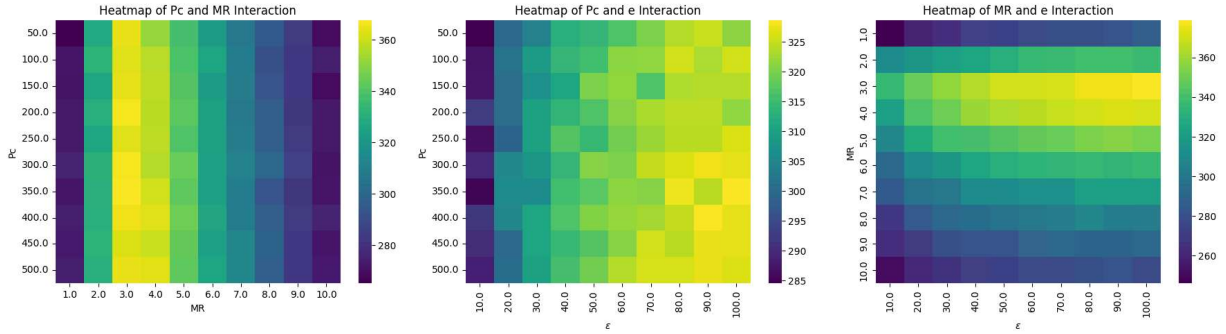
```
ANOVA for pc: F-value = 0.17657876054005522, p-value = 0.9963914948007738
ANOVA for mr: F-value = 663.3631865619716, p-value = 0.0
ANOVA for e: F-value = 13.46003934613545, p-value = 1.6324343067921846e-20
```



To analyze the impact of interactions, I perform a similar approach I used in prior assignments where I iterate across the main factors to form combinations where then I group by the combination to assess the mean specific impulse. I print some of these results and then visualize their combined approach with heat maps (showing the impact of two factors combined on specific impulse). The lighter colors on these heat maps indicate the more favorable specific impulse (higher values) and clearly start to illustrate trends of the combined impacts of two factors together.

In [ ]:
```
#----------------------------------------------------------------------
# Assess the impact of interactions
#----------------------------------------------------------------------
# Calculate interaction effects
interaction_effects = {}
for r in range(2, len(factors) + 1):
    for combo in itertools.combinations(factors, r):
        interaction_term = ' x '.join(combo)
        interaction_effects[interaction_term] = isp_df.groupby(list(combo))['isp'].

# Print interaction effects
for key, value in interaction_effects.items():
    print(f"Interaction: {key}")
    print(value)

# Plot heat maps of mean interactions
fig, axs = plt.subplots(1, 3, figsize=(18, 5))
# Heatmap for pc x mr
interaction = 'pc x mr'
if interaction in interaction_effects:
    heatmap_data = isp_df.pivot_table(values='isp', index='pc', columns='mr', aggfu
    sns.heatmap(heatmap_data, ax=axs[0], cmap='viridis')
    axs[0].set_title('Heatmap of Pc and MR Interaction')
    axs[0].set_xlabel('MR')
    axs[0].set_ylabel('Pc')

# Heatmap for pc x e
interaction = 'pc x e'
if interaction in interaction_effects:
    heatmap_data = isp_df.pivot_table(values='isp', index='pc', columns='e', aggfun
    sns.heatmap(heatmap_data, ax=axs[1], cmap='viridis')
    axs[1].set_title('Heatmap of Pc and e Interaction')
    axs[1].set_xlabel(r'$\epsilon$')
    axs[1].set_ylabel('Pc')

# Heatmap for mr x e
interaction = 'mr x e'
if interaction in interaction_effects:
    heatmap_data = isp_df.pivot_table(values='isp', index='mr', columns='e', aggfun
    sns.heatmap(heatmap_data, ax=axs[2], cmap='viridis')
    axs[2].set_title('Heatmap of MR and e Interaction')
    axs[2].set_xlabel(r'$\epsilon$')
    axs[2].set_ylabel('MR')

# Adjust layout
plt.tight_layout()
plt.savefig(os.path.join(os.getenv('USERPROFILE'), 'repos', 'mece-6397-doe', 'proje
plt.show()
plt.close()
```

```
Interaction: pc x mr
pc     mr
50.0   1.0     265.157621
       2.0     327.454867
       3.0     364.351821
       4.0     351.544488
       5.0     337.169403
                  ...
500.0  6.0     326.650601
       7.0     312.614619
       8.0     295.705764
       9.0     282.357956
       10.0    273.390172
Name: isp, Length: 100, dtype: float64
Interaction: pc x e
pc     e
50.0   10.0     284.520309
       20.0     299.400533
       30.0     304.068429
       40.0     312.734474
       50.0     314.093674
                   ...
500.0  60.0     323.963769
       70.0     325.975311
       80.0     326.038469
       90.0     327.244889
       100.0    326.255065
Name: isp, Length: 100, dtype: float64
Interaction: mr x e
mr    e
1.0   10.0     245.978808
      20.0     259.786709
      30.0     264.355361
      40.0     272.746277
      50.0     273.424086
                  ...
10.0  60.0     277.661364
      70.0     277.817082
      80.0     280.119257
      90.0     281.836289
      100.0    277.812916
Name: isp, Length: 100, dtype: float64
Interaction: pc x mr x e
pc     mr    e
50.0   1.0   10.0     251.943628
             20.0     248.409757
             30.0     252.367485
             40.0     264.954752
             50.0     267.331579
                         ...
500.0  10.0  60.0     284.558198
             70.0     282.004102
             80.0     288.198171
             90.0     283.208340
             100.0    271.079044
Name: isp, Length: 1000, dtype: float64
```



Part 4: Preparing data for training

Very simply I prepare the dataset for training regression models by splitting the dataset into 20% for testing and 80% for training. Additionally, I scale the data to the same ranges in order to make the data more digestible to some regression methods that would be sensitive to different magnitudes of values (i.e. small mixture ratios with large chamber pressures).

In [ ]:
```
#-----------------------------------------------------------------------
# Slice data into training and test
#-----------------------------------------------------------------------
X = isp_df[['pc', 'mr', 'e']].values.tolist()

# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, isp_df['isp'], test_size=0.2

# Scaling the data for some delicate regressors
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Part 5: Creating regression models

I train several regression models and evaluate these model's predicted specific impulse vs the actual specific impulse values (along with the MSE). The selected regression methods with a short description for new methods (relative to our first assignment) are:

- Linear
- Decision Tree
- Random Forest
- Support Vector
- K-Nearest Neighbors
- Gradient Boosting: Builds sequential decision trees (like random forest) but each tree corrects error of prior trees
- XGBoost: Optimized implementation of gradient boosting
- CatBoost: Gradient boost library that excels in categorical features

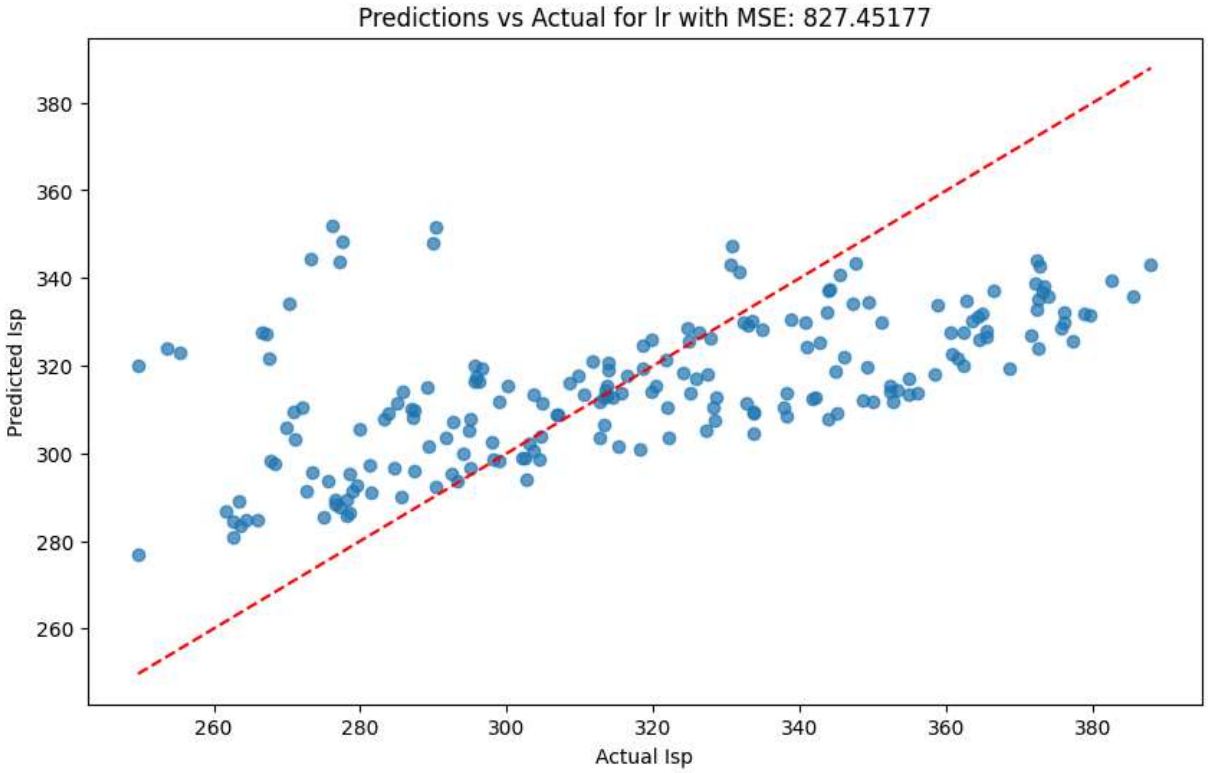
- ElasticNet: Combines L1 and L2 regularization
- Lasso: Uses L1 regularization to force selection of most relevant model features
- Ridge: Reduces model complexity to prevent overfitting via L2 regularization
- Bayesian Ridge: Estimates cfs of a linear model and incorporates prior distributions
- AdaBoost: Combines multiple learning methods and weights their predictions

Each model is plotted with its predicted vs actual values, where more discussion on the all the models' accuracy is provided at the end of this markdown.

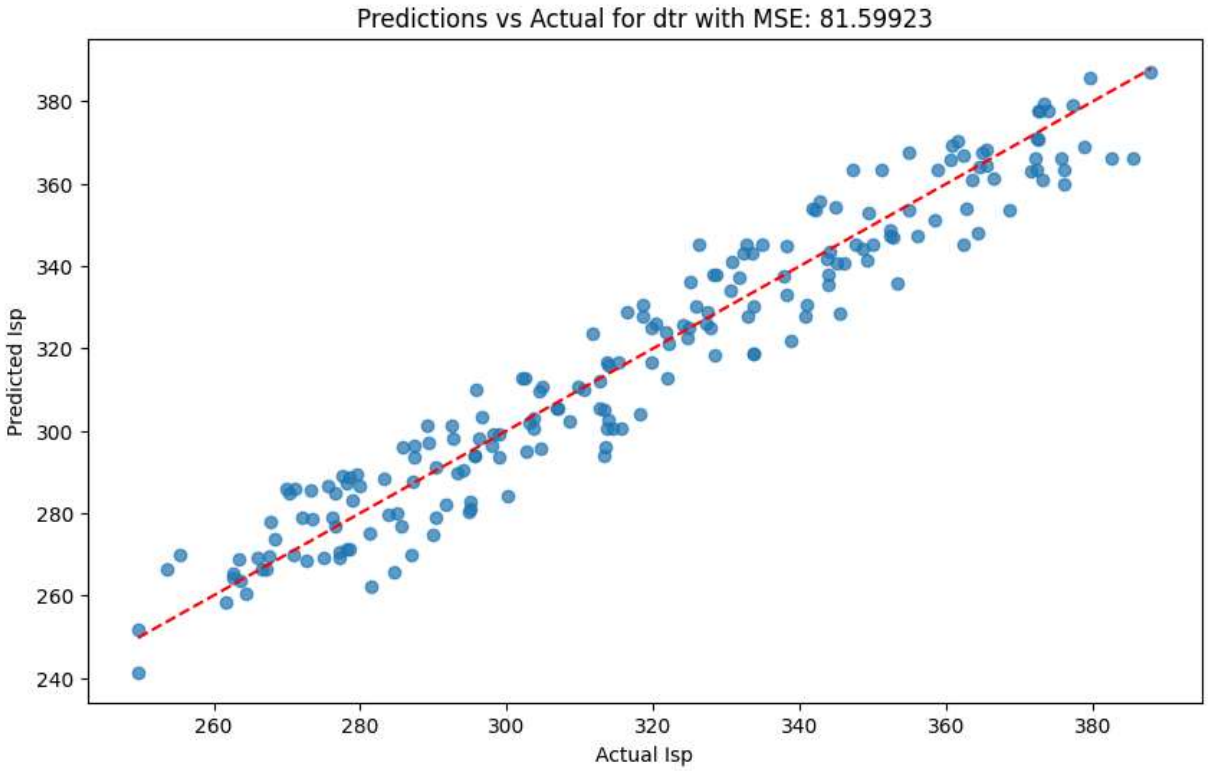
In [ ]:
```
#--------------------------------------------------------------------
# Apply different regression approaches
#--------------------------------------------------------------------
# Init results dict
results = {
    "lr": [],            # Linear
    "dtr": [],           # Decision tree
    "rfr": [],           # Random forest
    "svr": [],           # Support vector
    "knn": [],           # K-nearest neighbors
    "gbr": [],           # Gradient boosting
    "xgbr": [],          # XGBoost
    "catbr": [],         # CatBoost
    "en": [],            # ElasticNet
    "lasso": [],         # Lasso
    "ridge": [],         # Ridge
    "bayesianridge": [], # Bayesian ridge
    "adaboost": []       # AdaBoost
}

models = {}
predictions = {}
table = []
best_transformed_features = None

# Train different regression models
models["lr"] = LinearRegression().fit(X_train, y_train)
models["dtr"] = DecisionTreeRegressor().fit(X_train, y_train)
models["rfr"] = RandomForestRegressor().fit(X_train, y_train)
models["svr"] = SVR().fit(X_train, y_train)
models["knn"] = KNeighborsRegressor().fit(X_train, y_train)
models["gbr"] = GradientBoostingRegressor().fit(X_train, y_train)
models["xgbr"] = XGBRegressor().fit(X_train, y_train)
models["catbr"] = CatBoostRegressor(silent=True).fit(X_train, y_train)
models["en"] = ElasticNet().fit(X_train, y_train)
models["lasso"] = Lasso().fit(X_train, y_train)
models["ridge"] = Ridge().fit(X_train, y_train)
models["bayesianridge"] = BayesianRidge().fit(X_train, y_train)
models["adaboost"] = AdaBoostRegressor().fit(X_train, y_train)

# Predict and calculate MSE
for regressor in models:
    predictions[regressor] = models[regressor].predict(X_test)
    mse = mean_squared_error(y_test, predictions[regressor])
```

```
    # Store the results
    results[regressor].append((mse, predictions[regressor]))

#----------------------------------------------------------------------
# Post process and plot different regressors for comparison
#----------------------------------------------------------------------
# Init table for txt output
table = []

# Post-process different regression approaches
for regressor in results:
    # Extract the MSE and best predictions
    min_mse, best_predictions = results[regressor][0]

    # Update table
    table.append([regressor, min_mse])

    # Print the MSE
    print(f"Regressor: {regressor}")
    print(f"MSE: {min_mse}")

    # Plotting predictions against actual values
    plt.figure(figsize=(10, 6))
    plt.scatter(y_test, best_predictions, alpha=0.7)
    plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', l
    plt.xlabel("Actual Isp")
    plt.ylabel("Predicted Isp")
    plt.title(f"Predictions vs Actual for {regressor} with MSE: {"{:.5f}".format(mi
    plt.savefig(os.path.join(os.getenv('USERPROFILE'), 'repos', 'mece-6397-doe', 'p
    plt.show()
    plt.close()
```

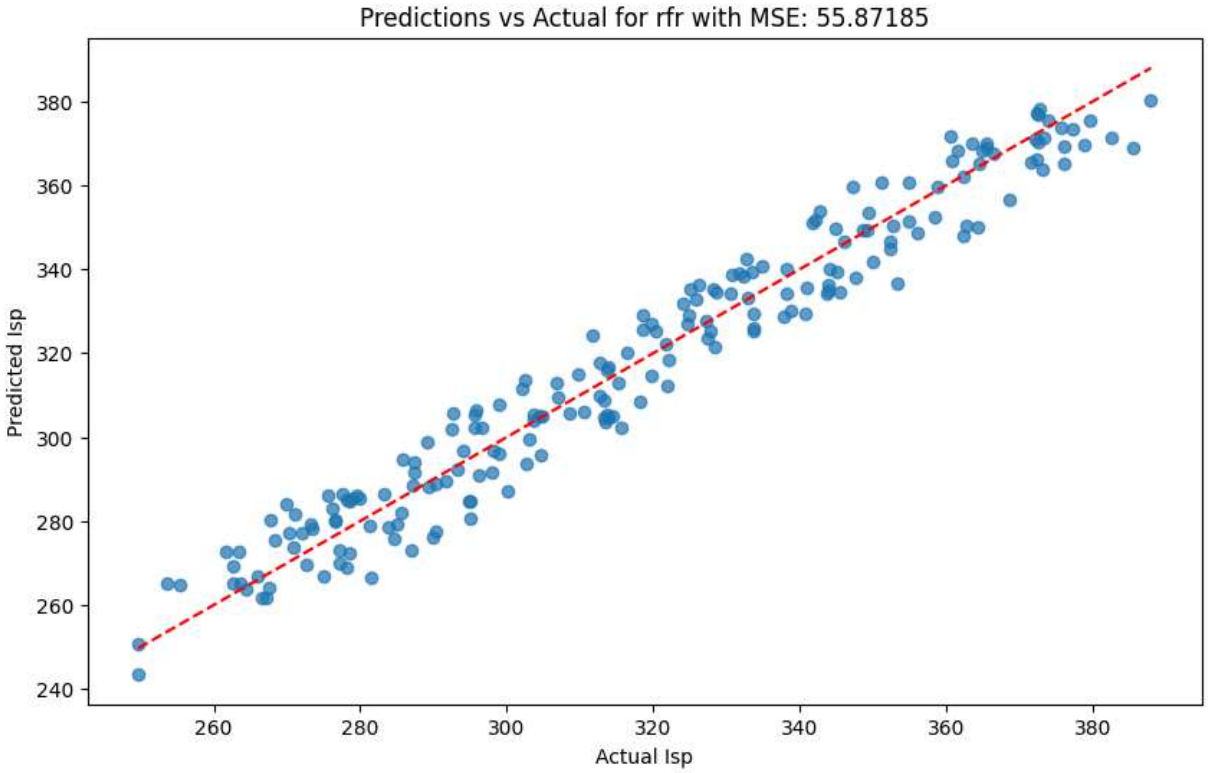
```
Regressor: lr
MSE: 827.4517743731698
```



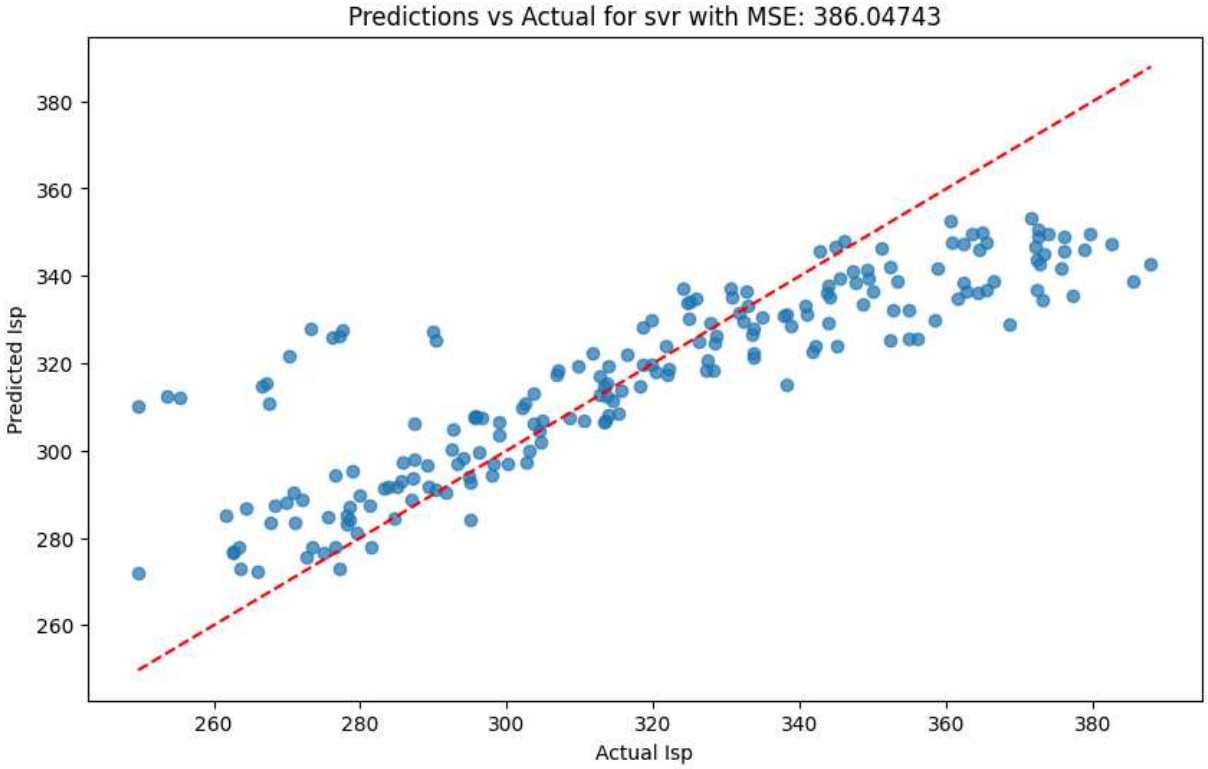

```
Regressor: dtr
MSE: 81.59922806968746
```



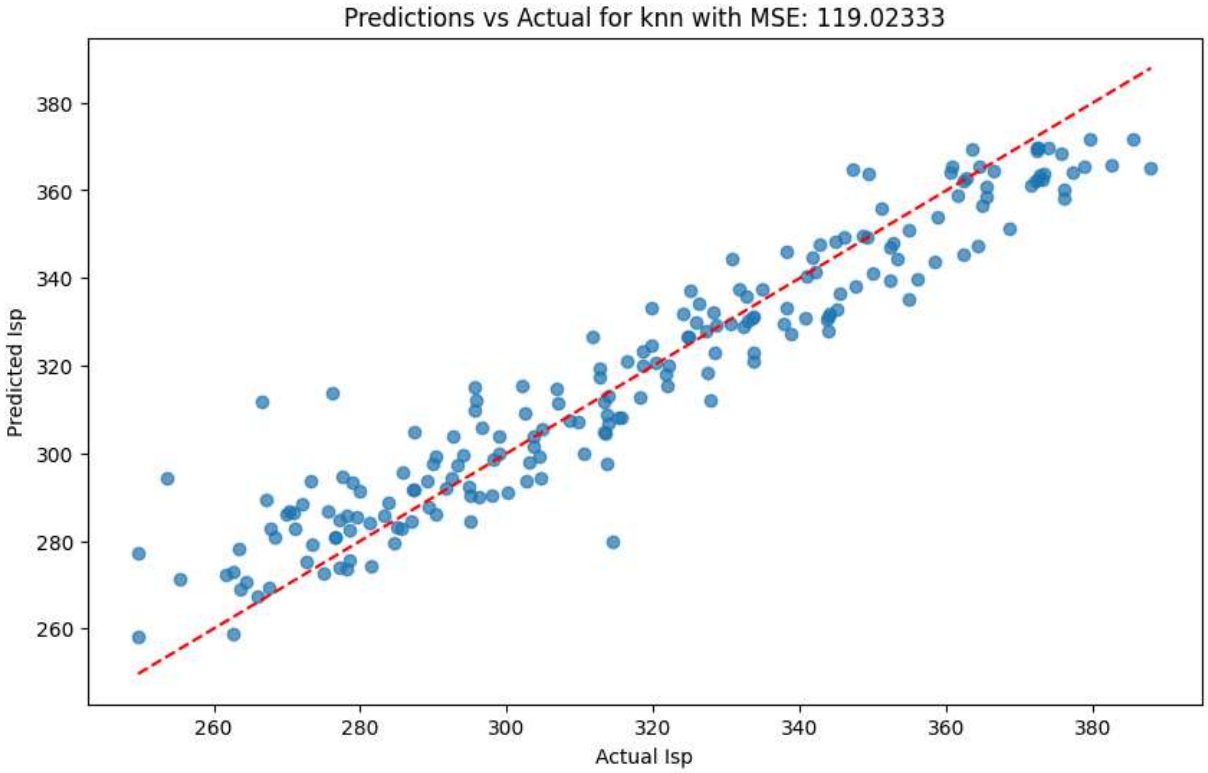

```
Regressor: rfr
MSE: 55.87184904247397
```



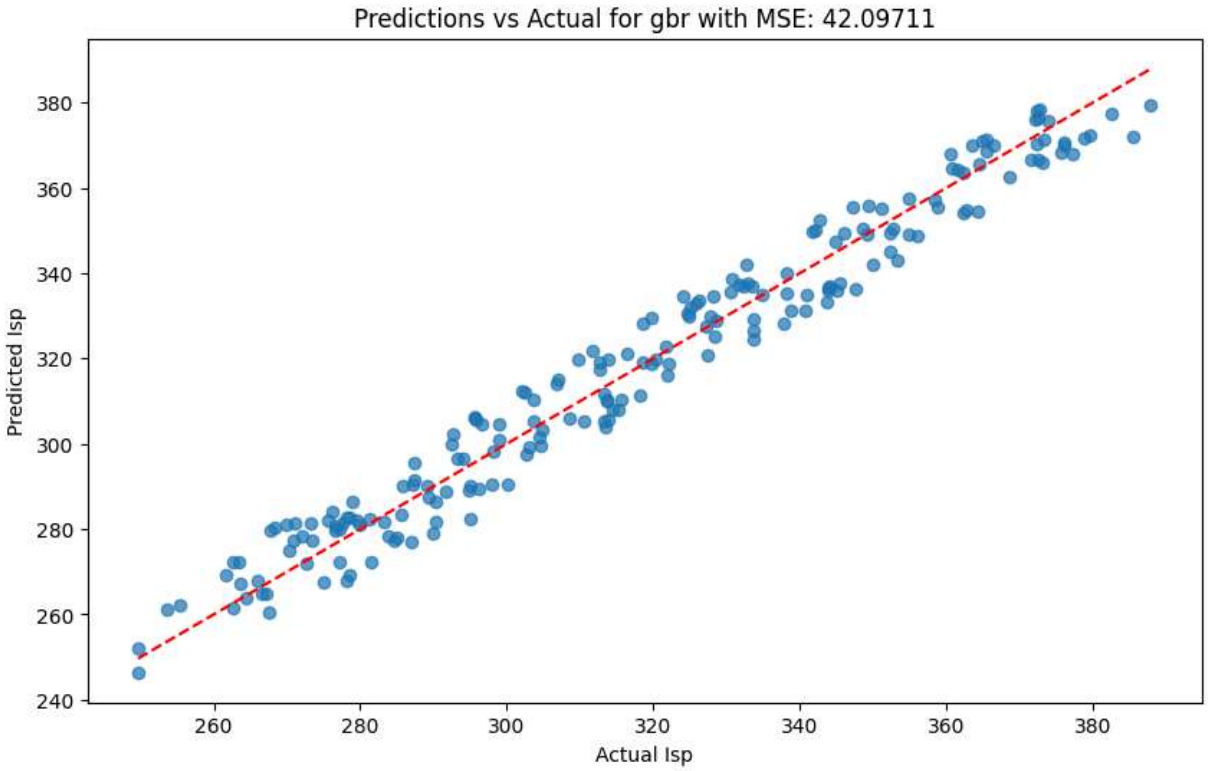

```
Regressor: svr
MSE: 386.04742678309225
```



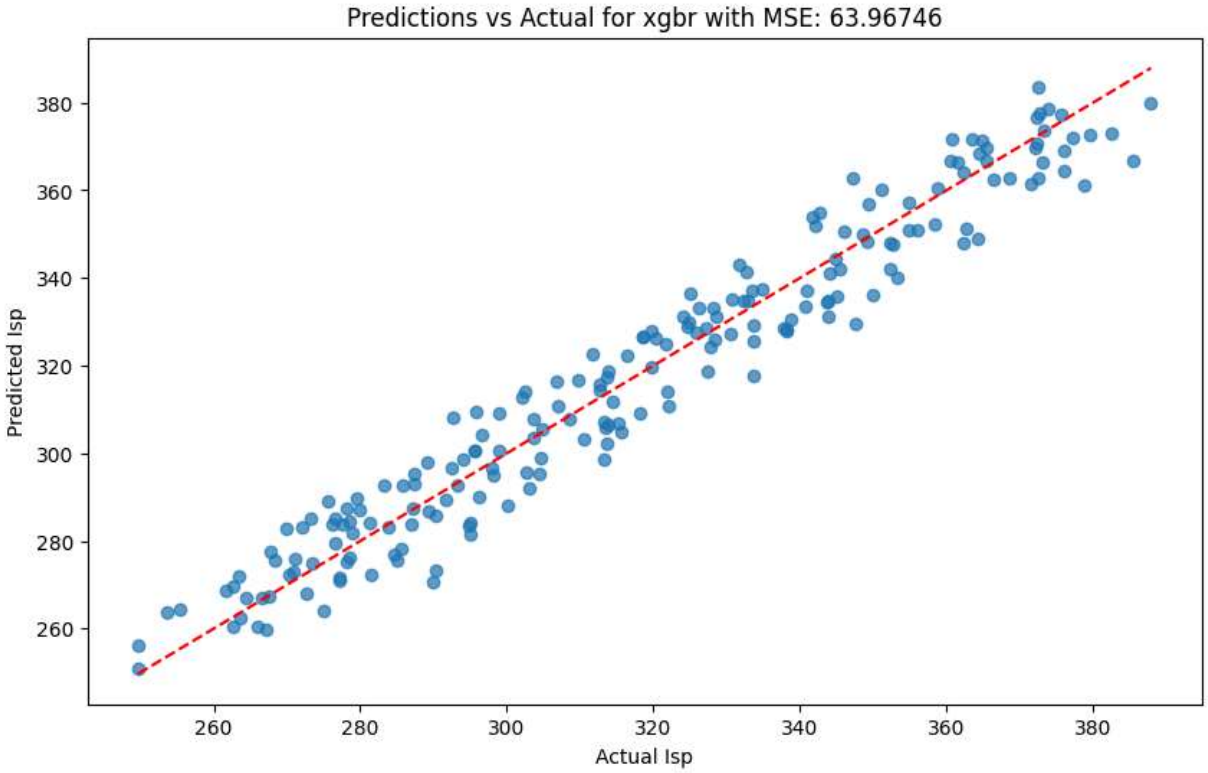

```
Regressor: knn
MSE: 119.02333465992814
```



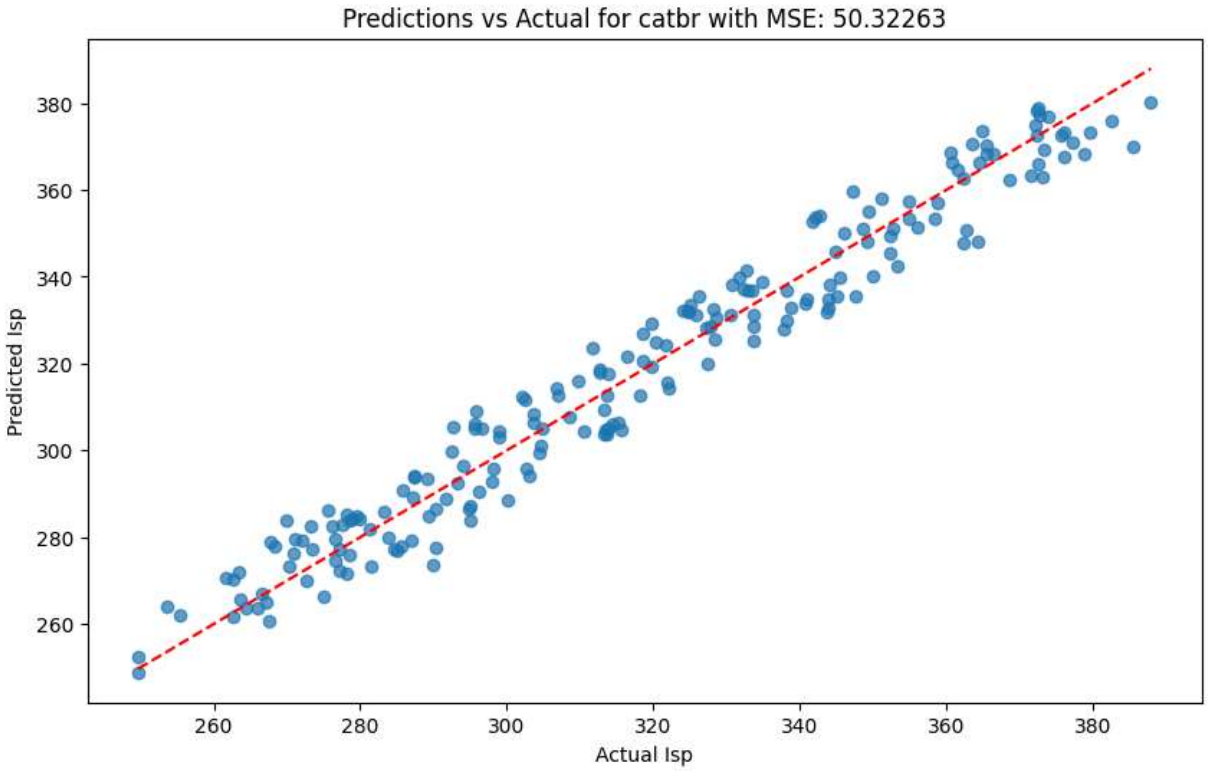

```
Regressor: gbr
MSE: 42.09710727056905
```



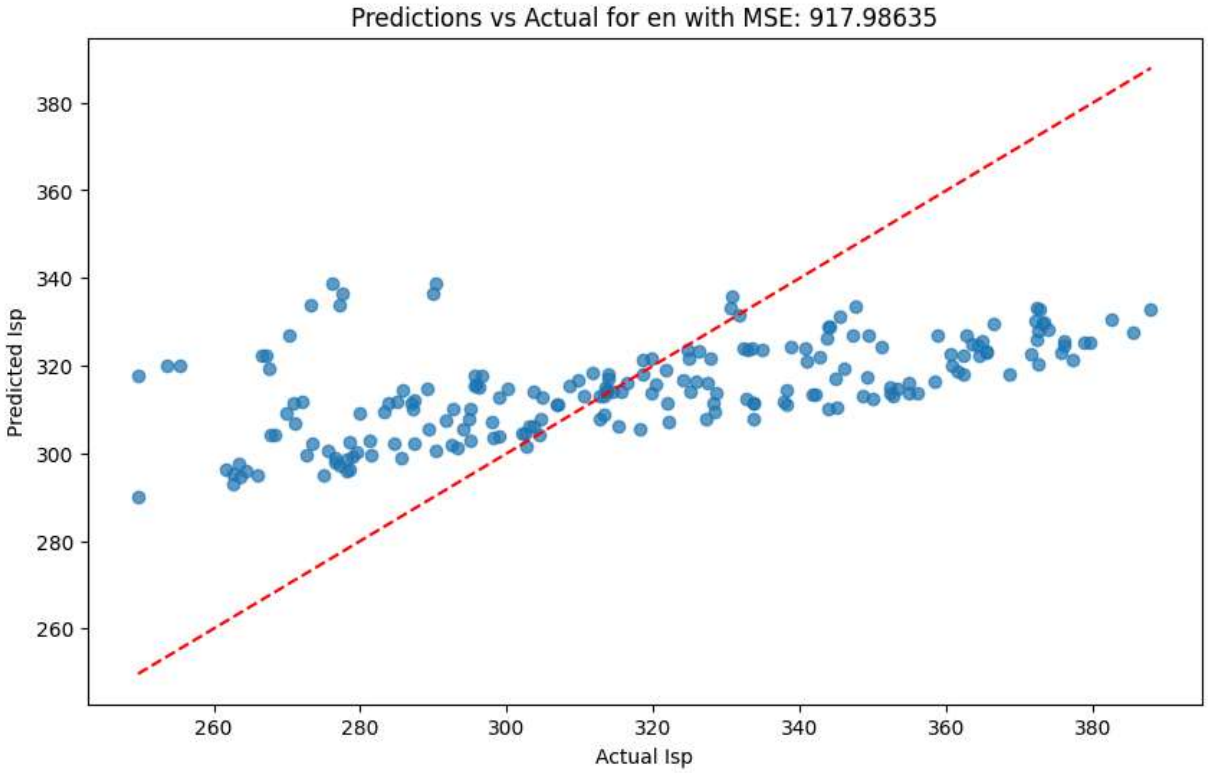

```
Regressor: xgbr
MSE: 63.967455848826575
```



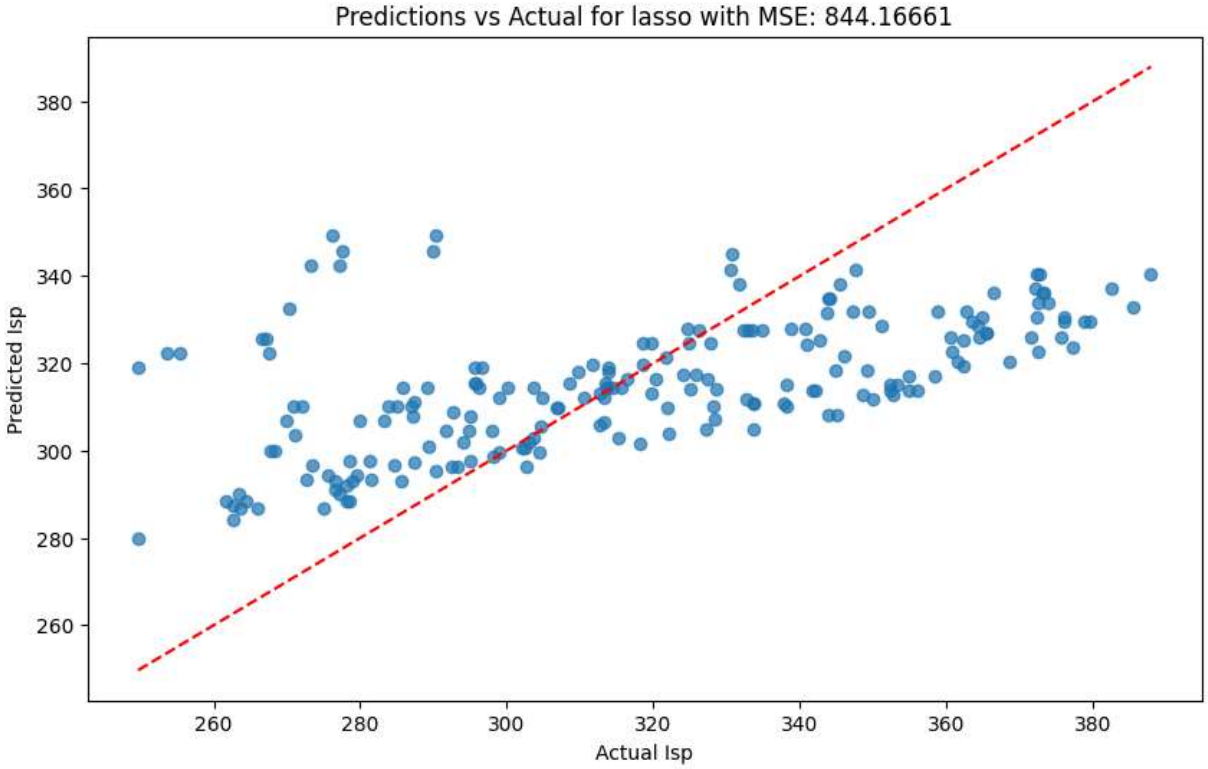

```
Regressor: catbr
MSE: 50.32263282998452
```



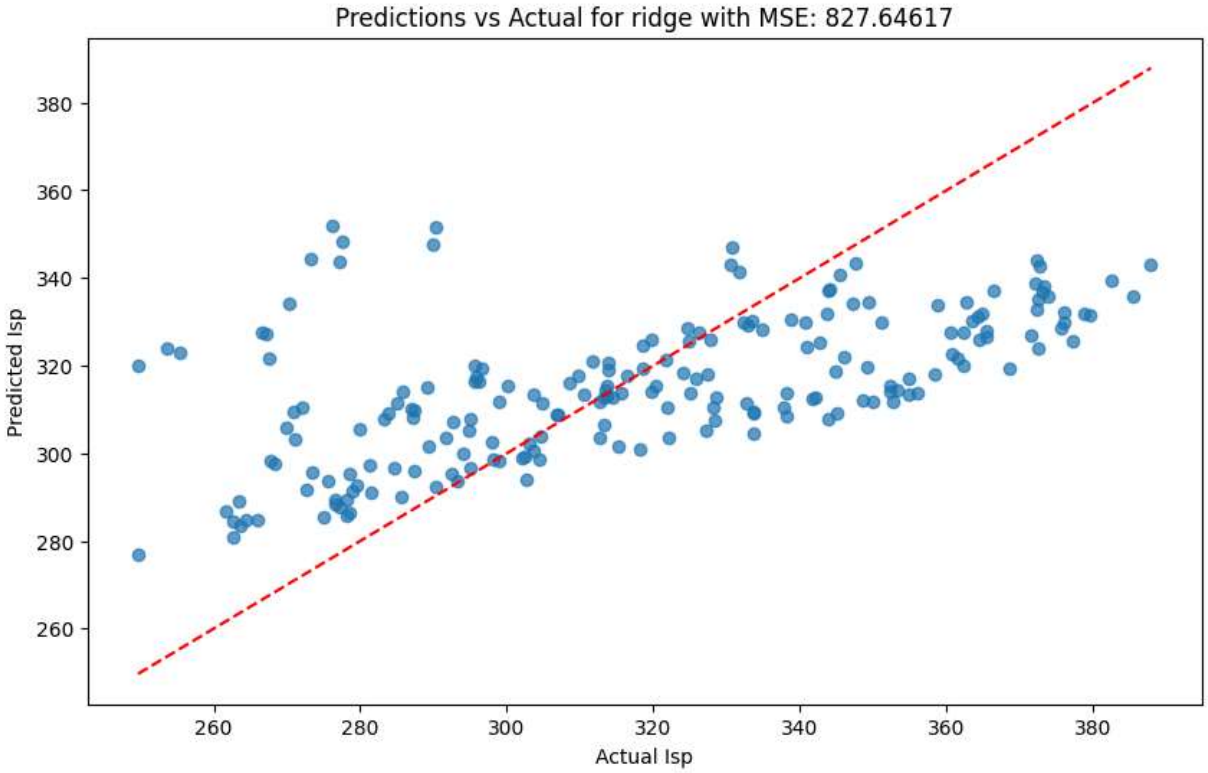

```
Regressor: en
MSE: 917.9863541681989
```



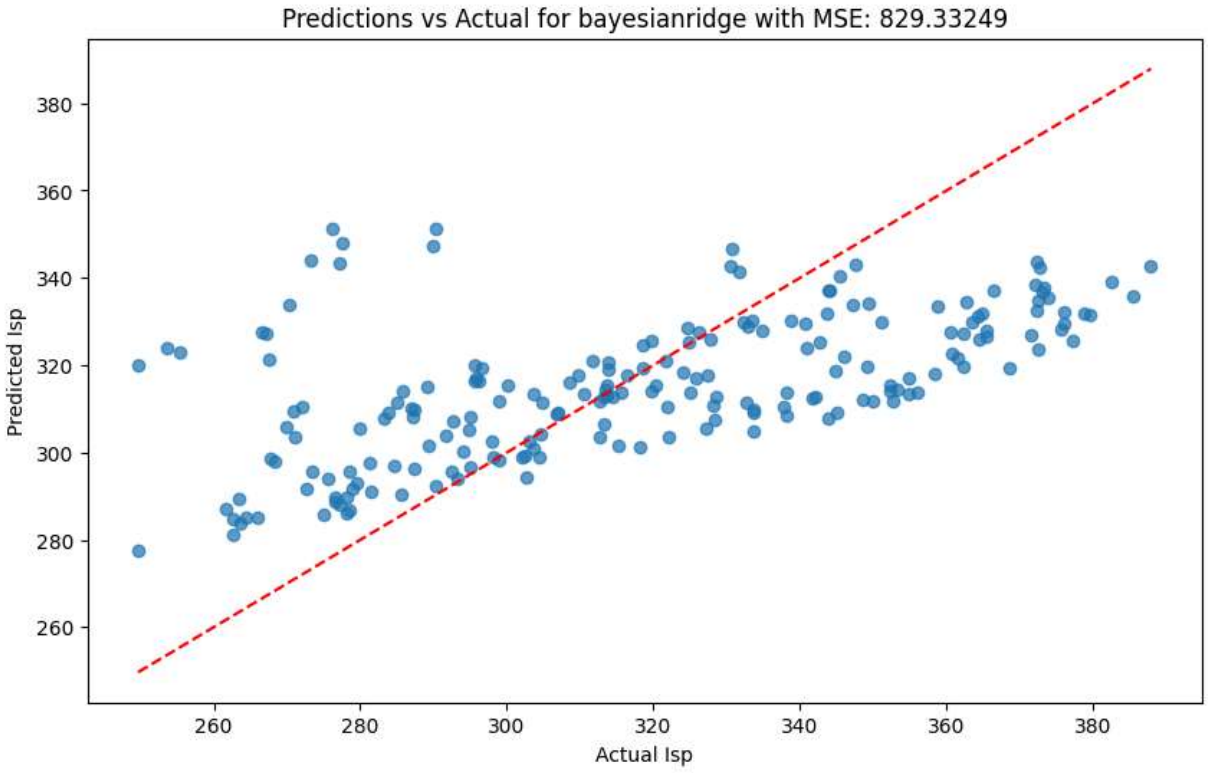

```
Regressor: lasso
MSE: 844.1666055464696
```



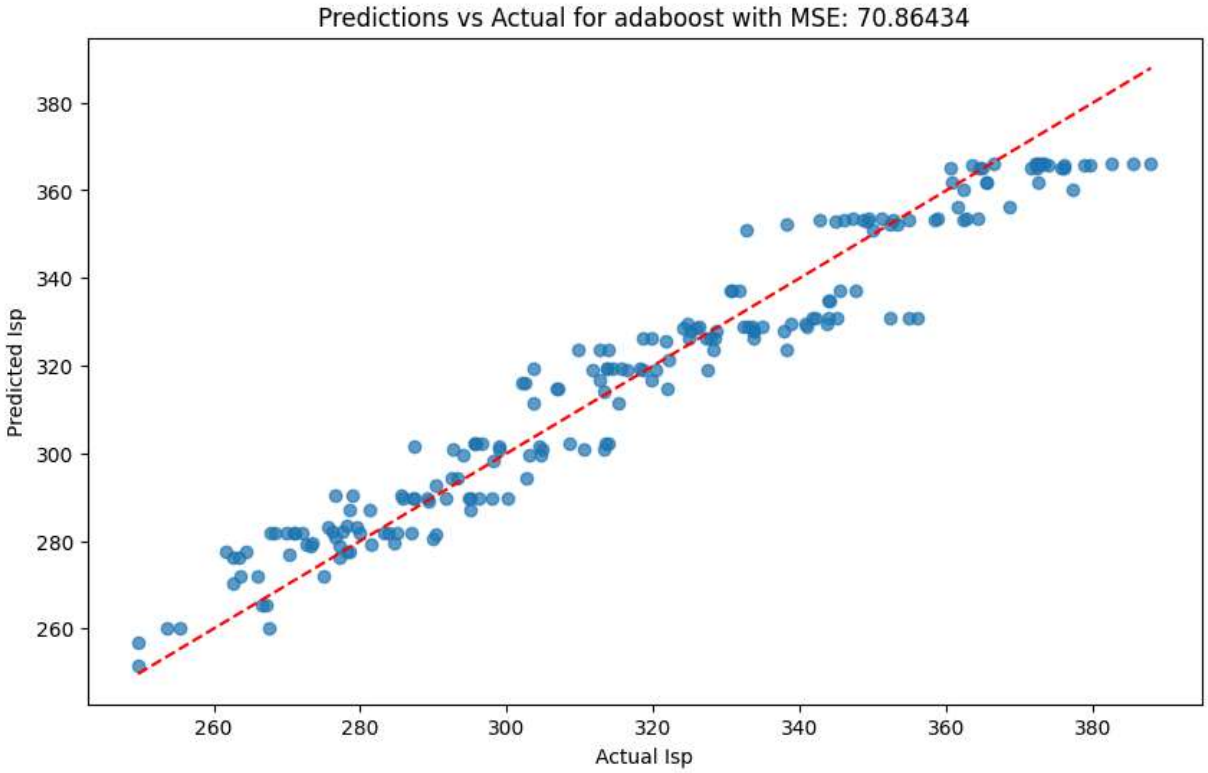

```
Regressor: ridge
MSE: 827.6461692281248
```




```
Regressor: bayesianridge
MSE: 829.3324902185856
```




```
Regressor: adaboost
MSE: 70.86434416664179
```



Part 6: Creating neural network

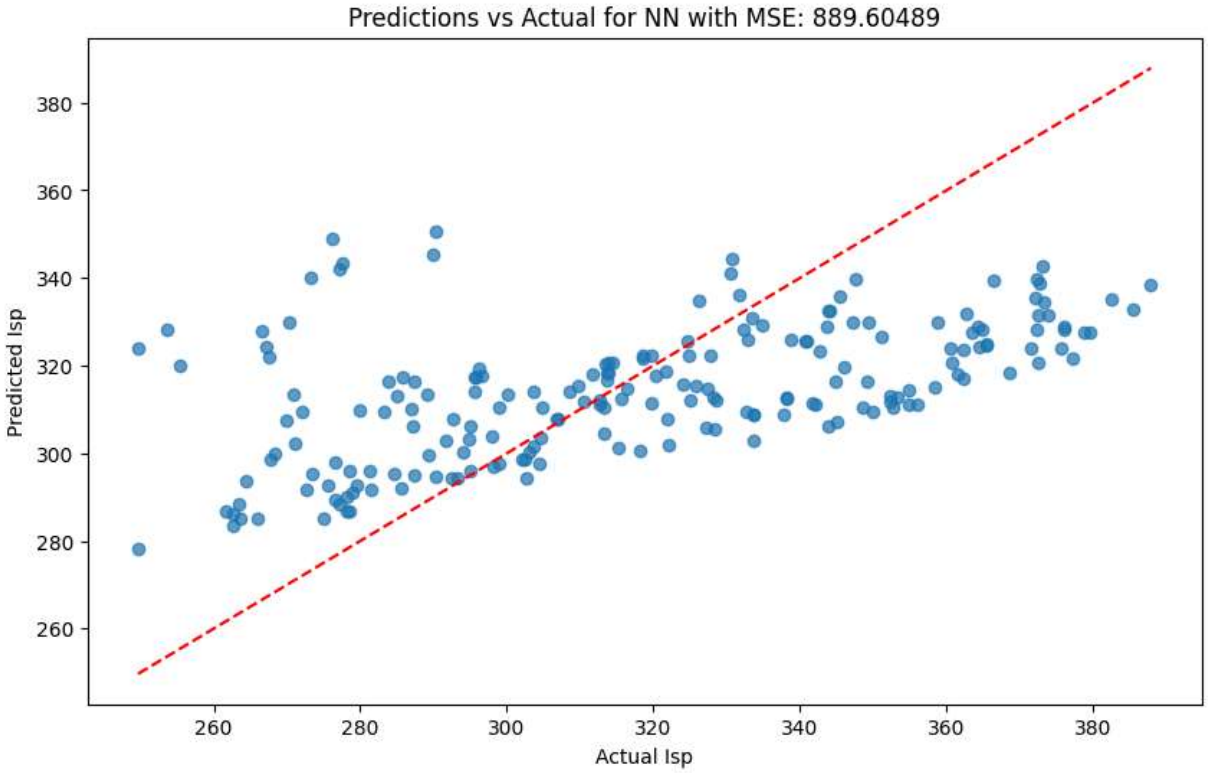
As a final implementation, I create a simple feed forward neural network with a linear stack of layers with a larger input layer and single output layer. I also use `ReLu` which from observing other's implementation of neural networks seemed to be a common approach for mitigating prolonged training.

The model is configured with a common optimizer (Adam) and set to compute the loss function of mean squared error which was a prior accuracy measurement for above regression models. I train the model for 1000 epochs/episodes with a larger batch size to stabilize training. I also add early stopping, which is a feature that stops training of the neural network when the validiation loss stops improving across 10 epochs (to prevent running too many epochs with no reward). I then predict values and plot the neural network's predictions vs actual values similar to how above regression results were shown.

In [ ]:
```
#-----------------------------------------------------------------
# NN implementation
#-----------------------------------------------------------------
# Define the neural network model
nn_model = Sequential()
nn_model.add(Dense(64, input_dim=X_train.shape[1], activation='relu'))
nn_model.add(Dense(32, activation='relu'))
nn_model.add(Dense(1))  # Output layer for regression
nn_model.compile(optimizer=Adam(learning_rate=0.01), loss='mean_squared_error')

# Define early stopping callback
```

```
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weight

# Train the neural network with early stopping
nn_model.fit(X_train, y_train, epochs=1000, batch_size=64, validation_split=0.2, ca

# Predict and calculate MSE
nn_predictions = nn_model.predict(X_test)
nn_mse = mean_squared_error(y_test, nn_predictions)
results["nn"] = [(nn_mse, nn_predictions)]

# Print the design, MSE, and feature values with the lowest error
print(f"Regressor: nn")
print(f"MSE: {nn_mse}")

# Update table
table.append(['NN', nn_mse])

# Plotting predictions against actual values
plt.figure(figsize=(10, 6))
plt.scatter(y_test, nn_predictions, alpha=0.7)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', lines
plt.xlabel("Actual Isp")
plt.ylabel("Predicted Isp")
plt.title(f"Predictions vs Actual for NN with MSE: {"{:.5f}".format(nn_mse)}")
plt.savefig(os.path.join(os.getenv('USERPROFILE'), 'repos', 'mece-6397-doe', 'proje
plt.show()
plt.close()
```

```
c:\Users\Sam\AppData\Local\Programs\Python\Python312\Lib\site-packages\keras\src\lay
ers\core\dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argument
to a layer. When using Sequential models, prefer using an `Input(shape)` object as t
he first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
Regressor: nn
MSE: 889.6048893584143
```



Part 7: Discussion

Finally I collect all MSE values and output a txt file of tabular results. Looking at all the predictions, we can observe in the table below that several models performed well given the non-linear data with 3 factors for specific impulse. Notably the following regression models performed well (defining as below 60 MSE) including:

- Random forest
- Gradient boost
- Catboost (offshoot of gradient boost)

This signals the strong performance of tree based methods whereas some of the linear models or models that heavily used regularization performed poorly with this non-linear dataset.

Interestingly, the neural network was the worst performing model, performing even worse than simple linear regression. I anticipate this is due to poor configuration of the model and better hyperparameter tuning could improve this model. Additionally, better model layout (more layers) could improve the performance of the neural network and would be considered forward work if this project were to continue.

Regressor	MSE
lr	827.45177

Regressor	MSE
dtr	81.59923
rfr	55.87185
svr	386.04743
knn	119.02333
gbr	42.09711
xgbr	63.96746
catbr	50.32263
en	917.98635
lasso	844.16661
ridge	827.64617
bayesianridge	829.33249
adaboost	70.86434
NN	889.60489

In [ ]:
```
#-----------------------------------------------------------------------
# Output summary table for various techniques
#-----------------------------------------------------------------------
# Write the table to a text file
output_file = os.path.join(os.getenv('USERPROFILE'), 'repos', 'mece-6397-doe', 'pro
with open(output_file, 'w') as f:
    f.write(f"{'Regressor':<20} {'MSE':<20}\n")
    f.write("="*60 + "\n")
    for row in table:
        f.write(f"{row[0]:<20} {row[1]:<20.5f}\n")
```