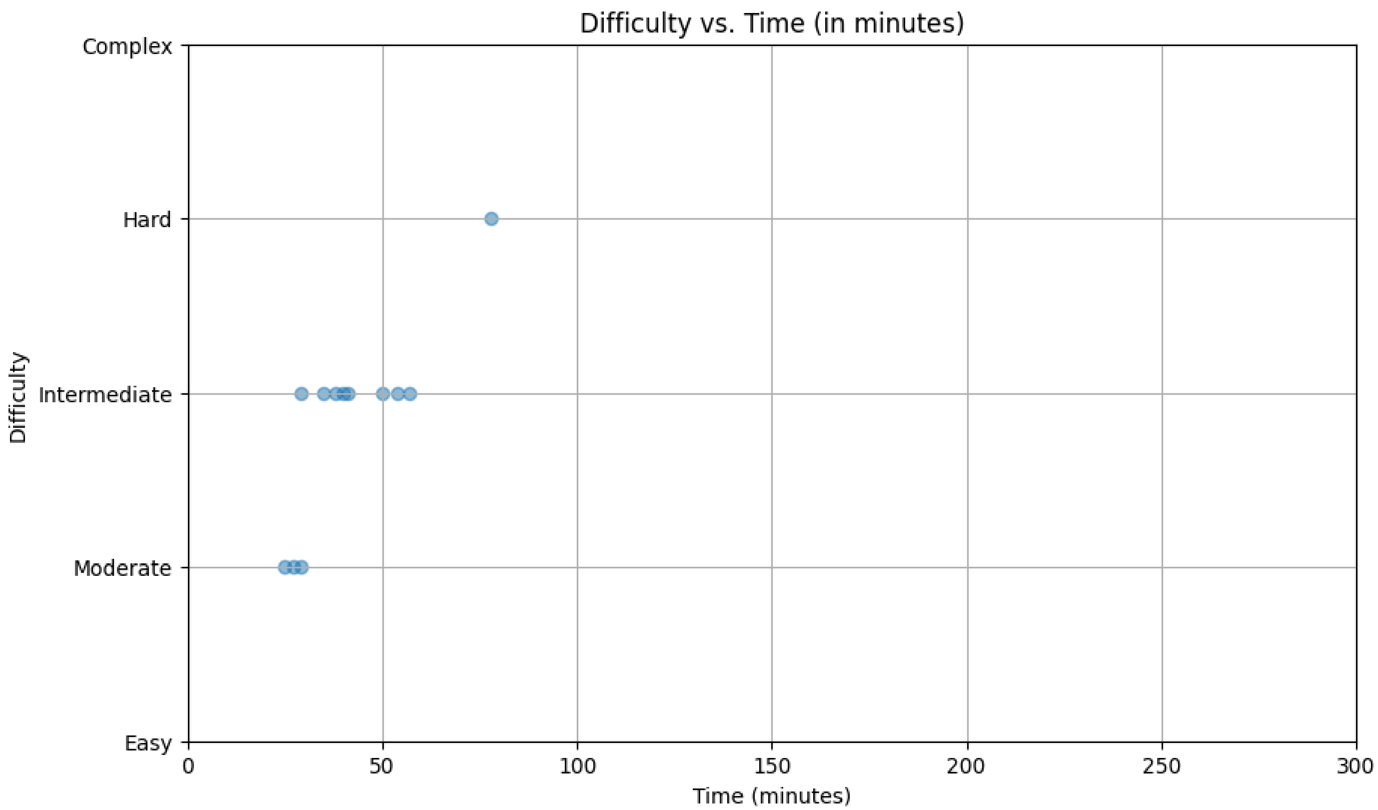
Utilizing small amounts of data to ensure data collection occured properly.

DATA IS FROM INITIALLY SCRAPED ORIGAMI MAIN PAGE IN R

```
import pandas as pd
import matplotlib.pyplot as plt
import re

# Load the data
df = pd.read_csv('origami_data_model.csv')

# Function to convert time to total minutes
def convert_to_minutes(time_str):
    hours = 0
    minutes = 0
    # Extract hours and minutes using regex
    hour_match = re.search(r'(\d+)\s*hr', time_str)
    if hour_match:
        hours = int(hour_match.group(1))
    minute_match = re.search(r'(\d+)\s*min', time_str)
    if minute_match:
        minutes = int(minute_match.group(1))
    return hours * 60 + minutes

# Apply the function to the Time column
df['time_minutes'] = df['Time'].apply(convert_to_minutes)

# Clean up the Difficulty values
df['Difficulty'] = df['Difficulty'].str.strip().str.lower()  # Remove extra spaces and convert to lower case

# Create the mapping
difficulty_mapping = {
    'easy': 1,
    'moderate': 2,
    'intermediate': 3,
    'hard': 4,
    'complex': 5
}

# Map the Difficulty column to the new numeric column
df['Difficulty_Numeric'] = df['Difficulty'].map(difficulty_mapping)

# Fill NA values in Difficulty_Numeric if any exist
df['Difficulty_Numeric'] = df['Difficulty_Numeric'].fillna(1)  # Fill NAs with 1 (or another appropriate value)

# Ensure Difficulty_Numeric is treated as an ordered categorical
df['Difficulty_Numeric'] = pd.Categorical(df['Difficulty_Numeric'], categories=[1, 2, 3, 4, 5], ordered=True)

# Create the scatter plot
```

```
# Create the scatter plot
plt.figure(figsize=(10, 6))
plt.scatter(df['time_minutes'], df['Difficulty_Numeric'], alpha=0.5)
plt.title('Difficulty vs. Time (in minutes)')
plt.xlabel('Time (minutes)')
plt.ylabel('Difficulty')
plt.yticks([1, 2, 3, 4, 5], ['Easy', 'Moderate', 'Intermediate', 'Hard', 'Complex'])  # Set y-ticks to the specified order
plt.xlim(0, 300)  # Set x-axis limits from 0 to 300
plt.grid(True)
plt.show()

# Check for unique values in the relevant columns
print("Unique time_minutes values:", df['time_minutes'].unique())
print("Unique Difficulty_Numeric values:", df['Difficulty_Numeric'].unique())
```




```
Unique time_minutes values: [29 25 41 27 50 35 40 38 54 57 78]
Unique Difficulty_Numeric values: [3, 2, 4]
Categories (5, int64): [1 < 2 < 3 < 4 < 5]
```

Attempting regression model to assess correlation between variables

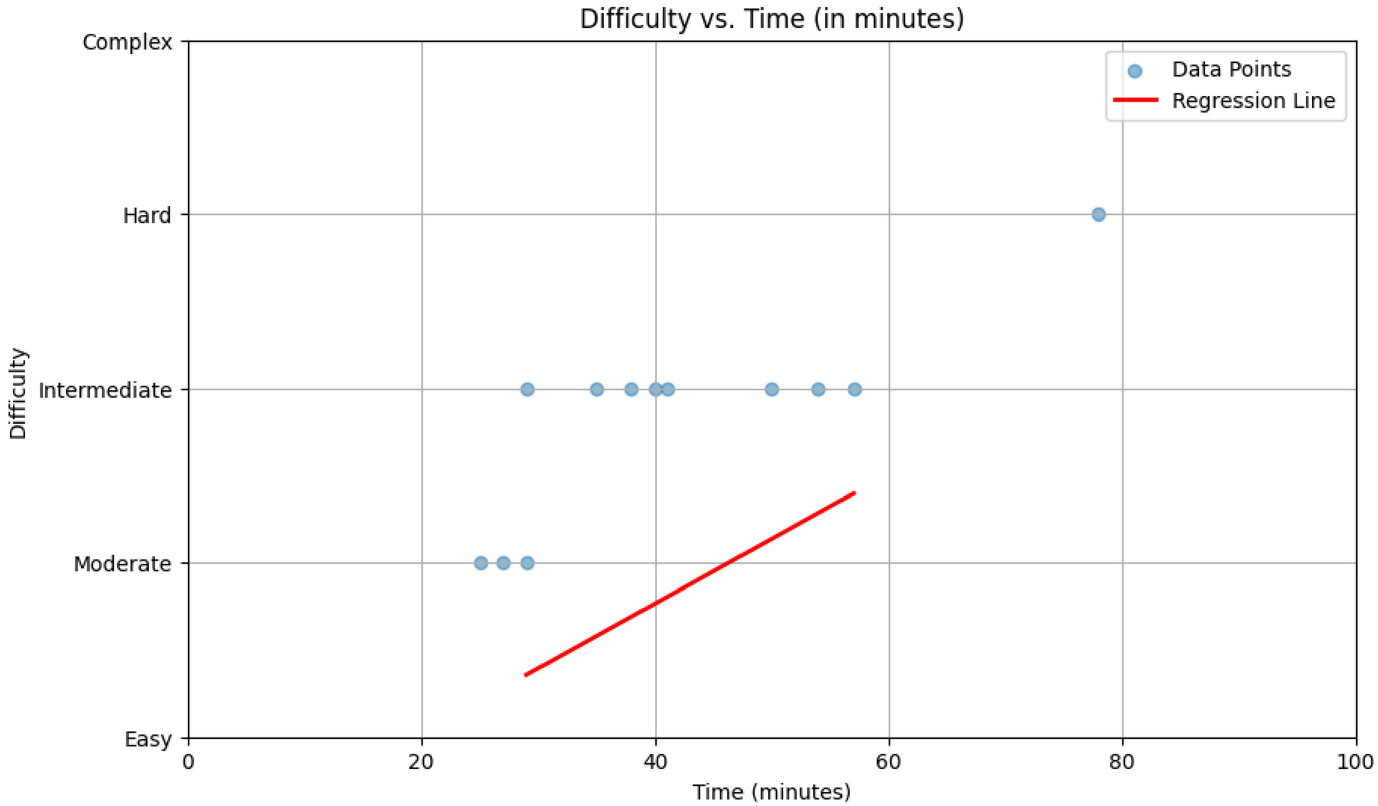
Attempting to create regression model - R-squared and MSE not reliable

```
import pandas as pd
import matplotlib.pyplot as plt
import re
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import cross_val_score
import seaborn as sns

# Load the data
df = pd.read_csv('origami_data_model.csv')

# Function to convert time to total minutes
def convert_to_minutes(time_str):
    try:
        hours = 0
        minutes = 0
        hour_match = re.search(r'(\d+)\s*hr', time_str)
        if hour_match:
            hours = int(hour_match.group(1))
        minute_match = re.search(r'(\d+)\s*min', time_str)
        if minute_match:
            minutes = int(minute_match.group(1))
        return hours * 60 + minutes
    except Exception as e:
        print(f"Error parsing time '{time_str}': {e}")
        return 0  # Default value

# Apply the function to the Time column
df['time_minutes'] = df['Time'].apply(convert_to_minutes)

# Clean up the Difficulty values
df['Difficulty'] = df['Difficulty'].str.strip().str.lower()  # Remove extra spaces and convert to lower case

# Create the mapping
difficulty_mapping = {
    'easy': 1,
    'moderate': 2,
    'intermediate': 3,
    'hard': 4,
    'complex': 5
}

# Map the Difficulty column to the new numeric column
df['Difficulty_Numeric'] = df['Difficulty'].map(difficulty_mapping)

# Fill NA values in Difficulty_Numeric if any exist
df['Difficulty_Numeric'] = df['Difficulty_Numeric'].fillna(1)  # Fill NAs with 1 (or another appropriate value)
```

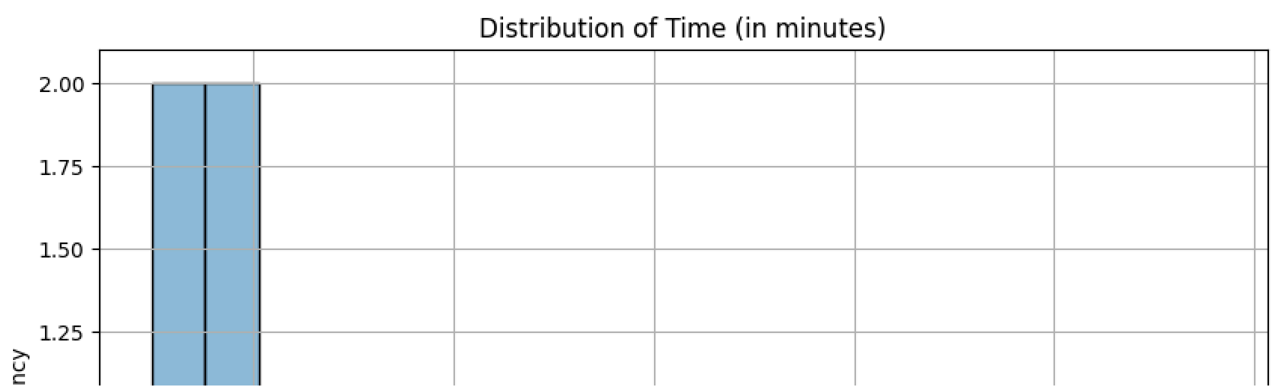
```
# Ensure Difficulty_Numeric is treated as an ordered categorical
df['Difficulty_Numeric'] = pd.Categorical(df['Difficulty_Numeric'], categories=[1, 2, 3, 4, 5], ordered=True)

# Prepare the features and target variable
X = df[['time_minutes']]  # Features
y = df['Difficulty_Numeric'].cat.codes  # Target variable as numeric codes

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create and fit the linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions
predictions = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, predictions)
r2 = r2_score(y_test, predictions)
print(f'Mean Squared Error: {mse}')
print(f'R-squared: {r2}')

# Cross-validation
scores = cross_val_score(model, X, y, cv=5, scoring='neg_mean_squared_error')
print("Cross-validated MSE:", -scores.mean())

# Create the scatter plot
plt.figure(figsize=(10, 6))
plt.scatter(df['time_minutes'], df['Difficulty_Numeric'], alpha=0.5, label='Data Points')
plt.plot(X_test, predictions, color='red', linewidth=2, label='Regression Line')
plt.title('Difficulty vs. Time (in minutes)')
plt.xlabel('Time (minutes)')
plt.ylabel('Difficulty')
plt.yticks([1, 2, 3, 4, 5], ['Easy', 'Moderate', 'Intermediate', 'Hard', 'Complex'])
plt.xlim(0, 100)
plt.grid(True)
plt.legend()
plt.show()

# Output predictions alongside actual values
predictions_df = pd.DataFrame({'Actual': y_test, 'Predicted': predictions})
print(predictions_df)

# Check for unique values in the relevant columns
print("Unique time_minutes values:", df['time_minutes'].unique())
print("Unique Difficulty_Numeric values:", df['Difficulty_Numeric'].unique())

# Optional: Visualize the distribution of time with a KDE overlay
plt.figure(figsize=(10, 6))
```

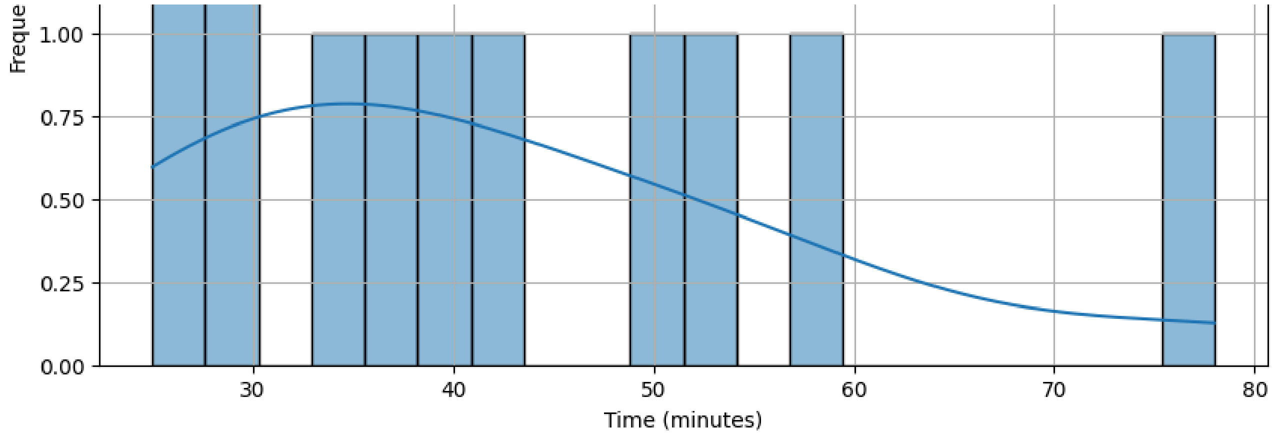
```
sns.histplot(df['time_minutes'], bins=20, kde=True)  # KDE overlay on histogram
plt.title('Distribution of Time (in minutes)')
plt.xlabel('Time (minutes)')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()
```

```
Mean Squared Error: 0.21817116037258333
R-squared: 0.0
Cross-validated MSE: 0.11745908708845616
```




```
    Actual  Predicted
10       2   2.397722
9        2   2.286132
0        2   1.356215
Unique time_minutes values: [29 25 41 27 50 35 40 38 54 57 78]
Unique Difficulty_Numeric values: [3, 2, 4]
Categories (5, int64): [1 < 2 < 3 < 4 < 5]
```






Produced closest regression model (R-squared and residual models were hard to create due to the discrete scale)

CALCULATED R-SQUARED TO BE

Coefficient of Determination (R-Squared): 0.6722

Adjusted R-Squared: 0.6394

Based on [https://exploringfinance.com/coefficient-of-determination-r-squared-calculator/](https://exploringfinance.com/coefficient-of-determination-r-squared-calculator/) Calculator

THERE ARE VERY FEW POINTS SO I DON'T EXPECT A HIGH CORRELATION BUT IN HTIS CODE I PROVIDE A POTENTIAL REGRESSION MODEL however the r-squared and MSE are not reliable here

I did calculate R-squared based on point locations as seen above

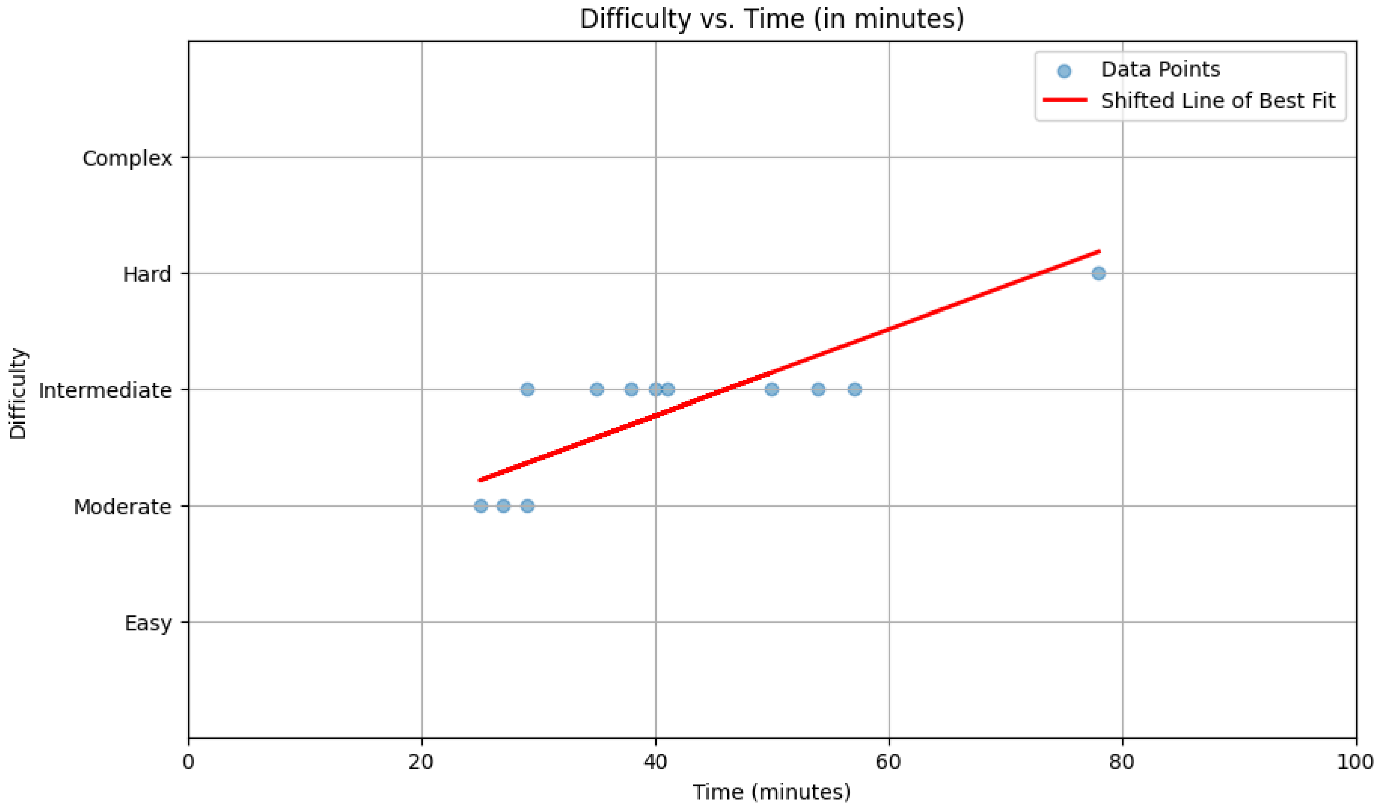
Start coding or generate with AI.

```
import pandas as pd
import matplotlib.pyplot as plt
import re
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import cross_val_score
import seaborn as sns

# Load the data
df = pd.read_csv('origami_data_model.csv')

# Function to convert time to total minutes
def convert_to_minutes(time_str):
```

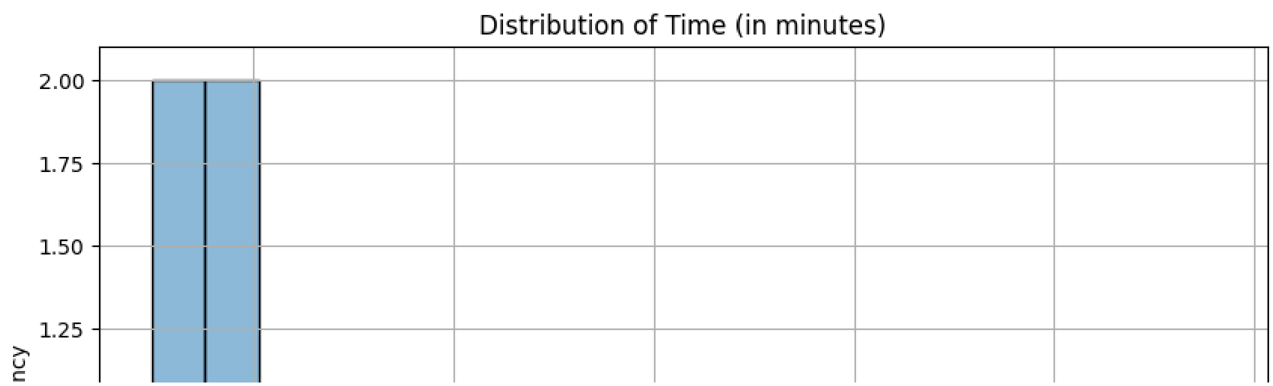
```
    try:
        hours = 0
        minutes = 0
        hour_match = re.search(r'(\d+)\s*hr', time_str)
        if hour_match:
            hours = int(hour_match.group(1))
        minute_match = re.search(r'(\d+)\s*min', time_str)
        if minute_match:
            minutes = int(minute_match.group(1))
        return hours * 60 + minutes
    except Exception as e:
        print(f"Error parsing time '{time_str}': {e}")
        return 0  # Default value

# Apply the function to the Time column
df['time_minutes'] = df['Time'].apply(convert_to_minutes)

# Clean up the Difficulty values
df['Difficulty'] = df['Difficulty'].str.strip().str.lower()  # Remove extra spaces and convert to lower case

# Create the mapping
difficulty_mapping = {
    'easy': 1,
    'moderate': 2,
    'intermediate': 3,
    'hard': 4,
    'complex': 5
}

# Map the Difficulty column to the new numeric column
df['Difficulty_Numeric'] = df['Difficulty'].map(difficulty_mapping)

# Fill NA values in Difficulty_Numeric if any exist
df['Difficulty_Numeric'] = df['Difficulty_Numeric'].fillna(1)  # Fill NAs with 1 (or another appropriate value)

# Ensure Difficulty_Numeric is treated as an ordered categorical
df['Difficulty_Numeric'] = pd.Categorical(df['Difficulty_Numeric'], categories=[1, 2, 3, 4, 5], ordered=True)

# Prepare the features and target variable
X = df[['time_minutes']]  # Features
y = df['Difficulty_Numeric'].cat.codes  # Target variable as numeric codes

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create and fit the linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions for the entire dataset for the line of best fit
predictions_full = model.predict(X)
```

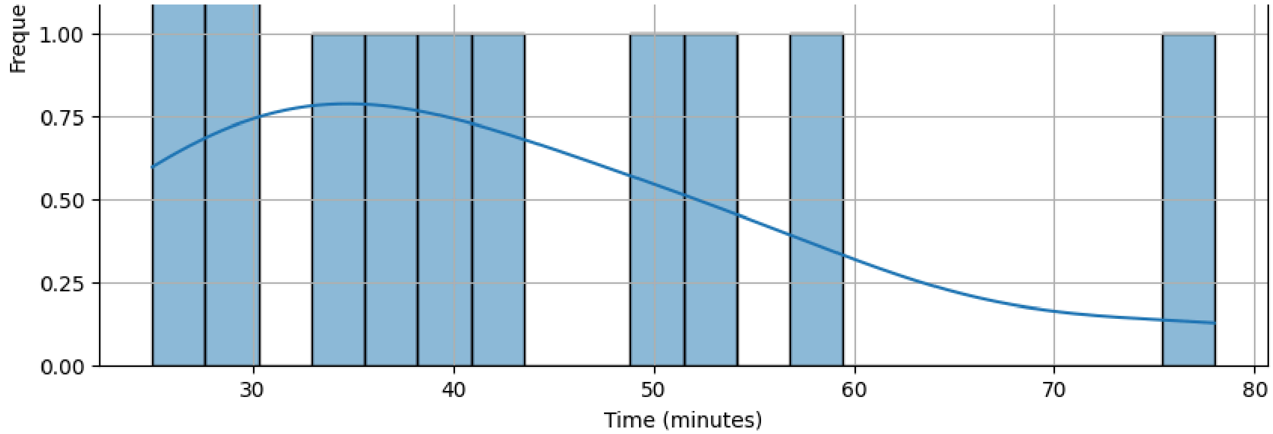
```
# Raise the line on the y-axis by adding a constant (e.g., 1)
y_shift = 1  # Adjust this value as needed
predictions_shifted = predictions_full + y_shift

# Evaluate the model
mse = mean_squared_error(y_test, predictions)
r2 = r2_score(y_test, predictions)
print(f'Mean Squared Error: {mse}')
print(f'R-squared: {r2}')

# Cross-validation
scores = cross_val_score(model, X, y, cv=5, scoring='neg_mean_squared_error')
print("Cross-validated MSE:", -scores.mean())

# Create the scatter plot
plt.figure(figsize=(10, 6))
plt.scatter(df['time_minutes'], df['Difficulty_Numeric'], alpha=0.5, label='Data Points')
plt.plot(df['time_minutes'], predictions_shifted, color='red', linewidth=2, label='Shifted Line of Best Fit')
plt.title('Difficulty vs. Time (in minutes)')
plt.xlabel('Time (minutes)')
plt.ylabel('Difficulty')
plt.yticks([1, 2, 3, 4, 5], ['Easy', 'Moderate', 'Intermediate', 'Hard', 'Complex'])
plt.xlim(0, 100)
plt.ylim(0, 6)  # Adjust y-axis limits if necessary
plt.grid(True)
plt.legend()
plt.show()

# Output predictions alongside actual values
predictions_df = pd.DataFrame({'Actual': y_test, 'Predicted': predictions})
print(predictions_df)

# Check for unique values in the relevant columns
print("Unique time_minutes values:", df['time_minutes'].unique())
print("Unique Difficulty_Numeric values:", df['Difficulty_Numeric'].unique())

# Optional: Visualize the distribution of time with a KDE overlay
plt.figure(figsize=(10, 6))
sns.histplot(df['time_minutes'], bins=20, kde=True)  # KDE overlay on histogram
plt.title('Distribution of Time (in minutes)')
plt.xlabel('Time (minutes)')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()
```

```
Mean Squared Error: 0.21817116037258333
R-squared: 0.0
Cross-validated MSE: 0.11745908708845616
```




```
    Actual  Predicted
10       2   2.397722
9        2   2.286132
0        2   1.356215
Unique time_minutes values: [29 25 41 27 50 35 40 38 54 57 78]
Unique Difficulty_Numeric values: [3, 2, 4]
Categories (5, int64): [1 < 2 < 3 < 4 < 5]
```

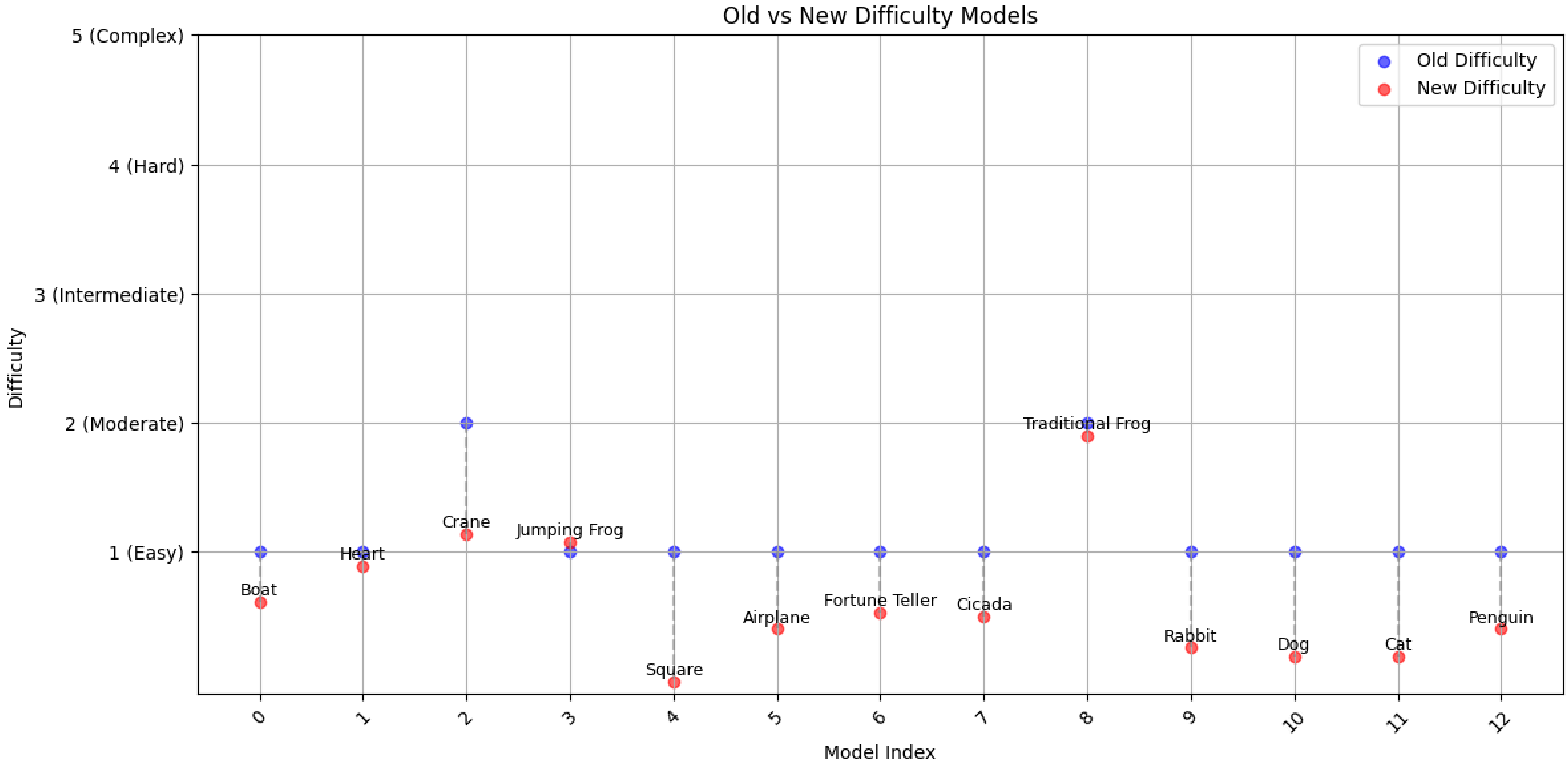





Creating adjusted difficulty scale based on number of folds and time taken. This could only be done with the models whose number of folds were counted. THIS INVOLVES THE DATASET IN OVERLEAF WEHERE I COUNTED THE NUMBER OF FOLDS FOR EACH MODEL. THE DIFFICULTY LEANS TOWARDS EASY AND I DIDNT HAVE TIME TO COUNT OR PREDICT NUMBER OF FOLDS FOR HARDER MODELS. WILL DO THIS IN THE FUTURE

```
import pandas as pd
import matplotlib.pyplot as plt

# Create DataFrame with the provided dataset
data = {
    'model': [
        'Boat', 'Heart', 'Crane', 'Jumping Frog', 'Square',
        'Airplane', 'Fortune Teller', 'Cicada', 'Traditional Frog',
        'Rabbit', 'Dog', 'Cat', 'Penguin'
    ],
    'number_of_folds': [10, 13, 16, 16, 2, 7, 8, 8, 23, 5, 4, 4, 7],
    'time_minutes': [2, 4, 5, 3, 0.5, 2, 4, 3, 14, 2, 2, 2, 2],
    'difficulty': ['Easy', 'Easy', 'Moderate', 'Easy', 'Easy',
                  'Easy', 'Easy', 'Easy', 'Moderate', 'Easy',
                  'Easy', 'Easy', 'Easy']
}

#NOTE: boat, fortune teller, and square can be recursively designed (making a smaller boat out of a larger one) which would increase the time and difficulty in some s

df = pd.DataFrame(data)

# Map difficulties to numeric values
difficulty_mapping = {
    'Easy': 1,
    'Moderate': 2,
    'Intermediate': 3,
    'Hard': 4,
```

```
        'Complex': 5  # Placeholder for future use
    }
    df['difficulty_numeric'] = df['difficulty'].map(difficulty_mapping)

    # Assign weights for Time and Number of Folds
    w1 = 0.4  # Weight for Time
    w2 = 1.5 # Weight for Number of Folds

    # Normalize Time and Number of Folds
    df['time_normalized'] = (df['time_minutes'] - df['time_minutes'].min()) / (df['time_minutes'].max() - df['time_minutes'].min())
    df['folds_normalized'] = (df['number_of_folds'] - df['number_of_folds'].min()) / (df['number_of_folds'].max() - df['number_of_folds'].min())

    # Calculate New Continuous Difficulty
    df['new_difficulty'] = w1 * df['time_normalized'] + w2 * df['folds_normalized']

    # Plotting
    plt.figure(figsize=(12, 6))

    # Scatter plot for old difficulty
    plt.scatter(range(len(df)), df['difficulty_numeric'], color='blue', label='Old Difficulty', alpha=0.6)

    # Scatter plot for new difficulty
    plt.scatter(range(len(df)), df['new_difficulty'], color='red', label='New Difficulty', alpha=0.6)

    # Adding annotations for each point
    for i, row in df.iterrows():
        plt.annotate(row['model'], (i, row['new_difficulty'] + 0.05), fontsize=9, ha='center')

    # Adding lines to show new difficulty more clearly
    for i, row in df.iterrows():
        plt.plot([i, i], [row['difficulty_numeric'], row['new_difficulty']], color='gray', linestyle='--', alpha=0.5)

    # Adding titles and labels
    plt.title('Old vs New Difficulty Models')
    plt.xlabel('Model Index')
    plt.ylabel('Difficulty')
    plt.xticks(range(len(df)), rotation=45)
    plt.yticks([1, 2, 3, 4, 5], ['1 (Easy)', '2 (Moderate)', '3 (Intermediate)', '4 (Hard)', '5 (Complex)'])
    plt.grid(True)
    plt.legend()
    plt.tight_layout()
    plt.show()

    # Display the DataFrame for reference
    print(df[['model', 'number_of_folds', 'time_minutes', 'difficulty', 'difficulty_numeric', 'new_difficulty']])
```



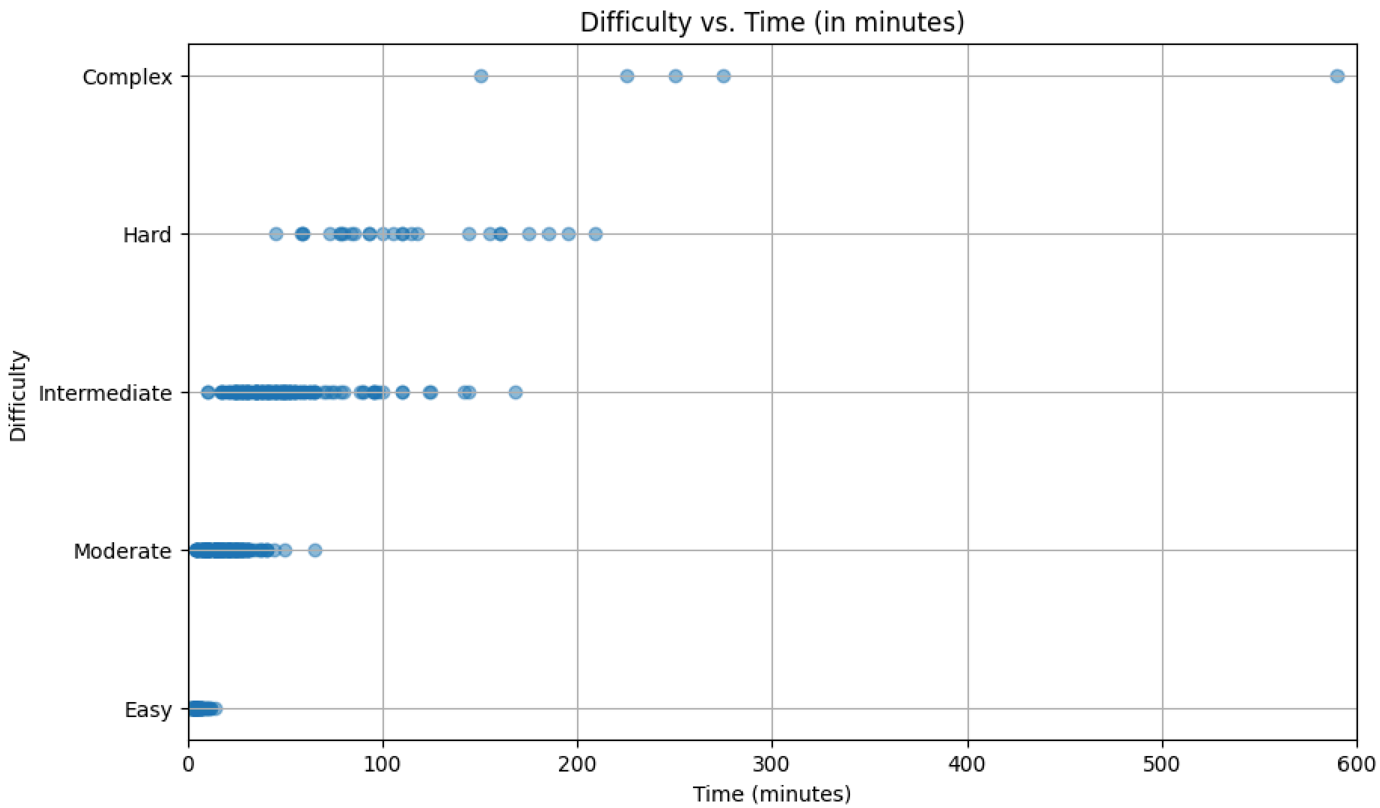

```
              model  number_of_folds  time_minutes difficulty  \
0              Boat               10           2.0       Easy
1             Heart               13           4.0       Easy
2             Crane               16           5.0   Moderate
3      Jumping Frog               16           3.0       Easy
4            Square                2           0.5       Easy
5          Airplane                7           2.0       Easy
6    Fortune Teller                8           4.0       Easy
7            Cicada                8           3.0       Easy
8  Traditional Frog               23          14.0   Moderate
9            Rabbit                5           2.0       Easy
10              Dog                4           2.0       Easy
11              Cat                4           2.0       Easy
12          Penguin                7           2.0       Easy

    difficulty_numeric  new_difficulty
0                    1        0.615873
1                    1        0.889418
2                    2        1.133333
3                    1        1.074074
4                    1        0.000000
5                    1        0.401587
```

```
6                    1       0.532275
7                    1       0.502646
8                    2       1.900000
9                    1       0.258730
10                   1       0.187302
11                   1       0.187302
12                   1       0.401587
```

Plotting fully scraped data (in R) of all 452 models

```
import pandas as pd
import matplotlib.pyplot as plt
import re

# Load the data
df = pd.read_csv('Origo_Database.csv')

# Function to convert time to total minutes
def convert_to_minutes(time_str):
    hours = 0
    minutes = 0
    # Extract hours and minutes using regex
    hour_match = re.search(r'(\d+)\s*hr', time_str)
    if hour_match:
        hours = int(hour_match.group(1))
    minute_match = re.search(r'(\d+)\s*min', time_str)
    if minute_match:
        minutes = int(minute_match.group(1))
    return hours * 60 + minutes

# Apply the function to the Time column
df['time_minutes'] = df['Time'].apply(convert_to_minutes)

# Clean up the Difficulty values
df['Difficulty'] = df['Difficulty'].str.strip().str.lower()  # Remove extra spaces and convert to lower case

# Create the mapping
difficulty_mapping = {
    'easy': 1,
    'moderate': 2,
    'intermediate': 3,
    'hard': 4,
    'complex': 5
}

# Map the Difficulty column to the new numeric column
df['Difficulty_Numeric'] = df['Difficulty'].map(difficulty_mapping)

# Fill NA values in Difficulty_Numeric if any exist
df['Difficulty_Numeric'] = df['Difficulty_Numeric'].fillna(1)  # Fill NAs with 1 (or another appropriate value)
```

```
# Ensure Difficulty_Numeric is treated as an ordered categorical
df['Difficulty_Numeric'] = pd.Categorical(df['Difficulty_Numeric'], categories=[1, 2, 3, 4, 5], ordered=True)

# Create the scatter plot
plt.figure(figsize=(10, 6))
plt.scatter(df['time_minutes'], df['Difficulty_Numeric'], alpha=0.5)
plt.title('Difficulty vs. Time (in minutes)')
plt.xlabel('Time (minutes)')
plt.ylabel('Difficulty')
plt.yticks([1, 2, 3, 4, 5], ['Easy', 'Moderate', 'Intermediate', 'Hard', 'Complex'])  # Set y-ticks to the specified order
plt.xlim(0, 600)  # Set x-axis limits from 0 to 300
plt.grid(True)
plt.show()

# Check for unique values in the relevant columns
print("Unique time_minutes values:", df['time_minutes'].unique())
print("Unique Difficulty_Numeric values:", df['Difficulty_Numeric'].unique())
```




```
Unique time_minutes values: [  3 142 175  48  30 110   4  52   2 225   6   8  20 250  14  78 118  22
  38  37  71 160   9  13   7  11  51  27 275  65  44  18  40  63  24  55
 144  29  25  41  50  35  54  57 590  33  98  88  28 209  74  90  45  93
  26  84  43  17  62  10  16  19  21 125  47  42 168 100  95  60  80  53
 124  15   5  23 185  64 150  46  12  32  36  31  39 155 105  75 195  59
  58  96  70  85 115  73]
Unique Difficulty_Numeric values: [1, 3, 4, 5, 2]
Categories (5, int64): [1 < 2 < 3 < 4 < 5]
```

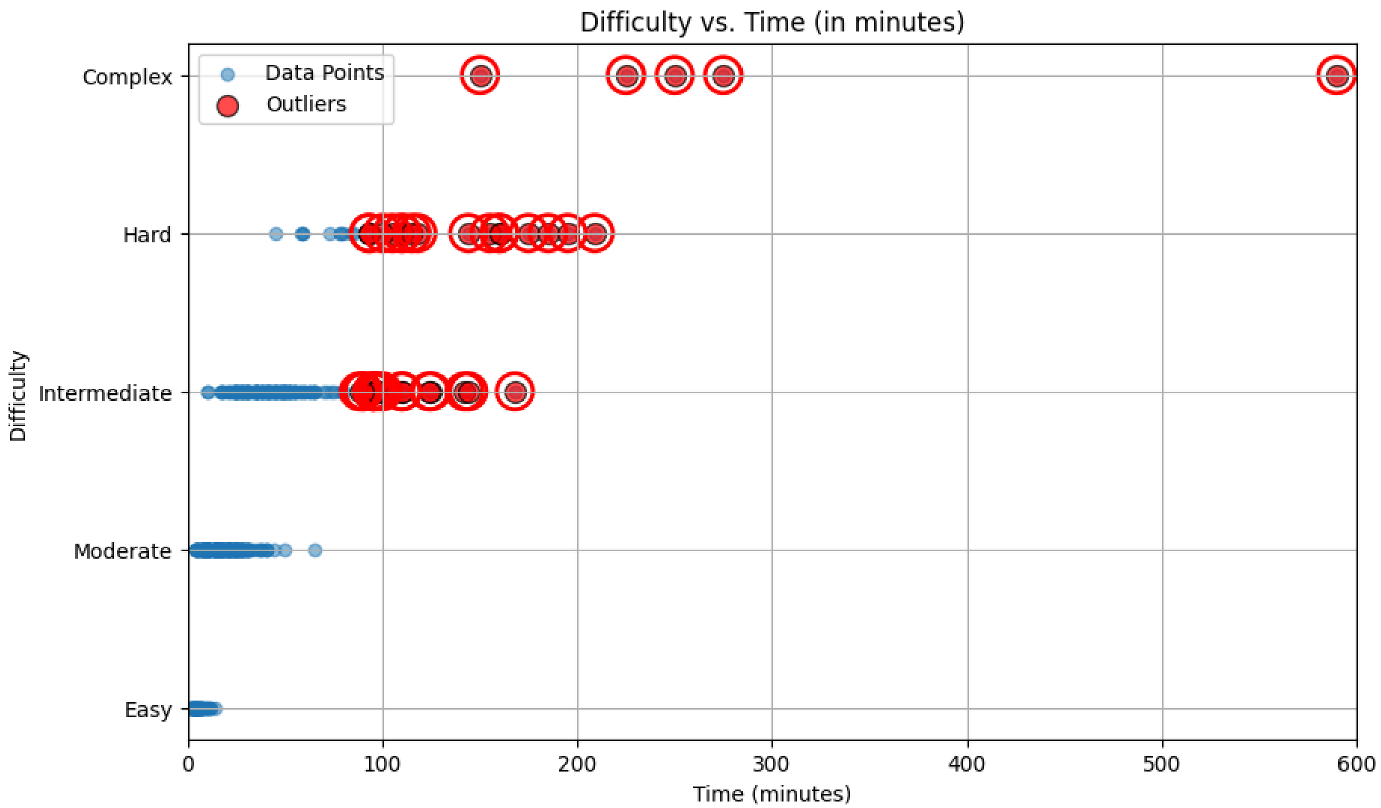
```
#OUTLIER CHART VISUALIZATION
```

In this case I removed outliers based on the time and the IQR of the time (this removed all instances in complex difficulty category)

```
import pandas as pd
import matplotlib.pyplot as plt
import re

# Load the data
df = pd.read_csv('Origo_Database.csv')

# Function to convert time to total minutes
```

```
# Function to convert time to total minutes
def convert_to_minutes(time_str):
    hours = 0
    minutes = 0
    # Extract hours and minutes using regex
    hour_match = re.search(r'(\d+)\s*hr', time_str)
    if hour_match:
        hours = int(hour_match.group(1))
    minute_match = re.search(r'(\d+)\s*min', time_str)
    if minute_match:
        minutes = int(minute_match.group(1))
    return hours * 60 + minutes

# Apply the function to the Time column
df['time_minutes'] = df['Time'].apply(convert_to_minutes)

# Clean up the Difficulty values
df['Difficulty'] = df['Difficulty'].str.strip().str.lower()

# Create the mapping
difficulty_mapping = {
    'easy': 1,
    'moderate': 2,
    'intermediate': 3,
    'hard': 4,
    'complex': 5
}

# Map the Difficulty column to the new numeric column
df['Difficulty_Numeric'] = df['Difficulty'].map(difficulty_mapping)

# Fill NA values in Difficulty_Numeric if any exist
df['Difficulty_Numeric'] = df['Difficulty_Numeric'].fillna(1)

# Ensure Difficulty_Numeric is treated as an ordered categorical
df['Difficulty_Numeric'] = pd.Categorical(df['Difficulty_Numeric'], categories=[1, 2, 3, 4, 5], ordered=True)

# Calculate IQR for time_minutes
Q1 = df['time_minutes'].quantile(0.25)
Q3 = df['time_minutes'].quantile(0.75)
IQR = Q3 - Q1

# Define outliers
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Identify outliers
df['Outlier'] = (df['time_minutes'] < lower_bound) | (df['time_minutes'] > upper_bound)

# Create the scatter plot
plt.figure(figsize=(10, 6))
plt.scatter(df['time_minutes'], df['Difficulty_Numeric'], alpha=0.5, label='Data Points')
```

```
# Highlight outliers in the plot
outliers = df[df['Outlier']]
plt.scatter(outliers['time_minutes'], outliers['Difficulty_Numeric'],
            color='red', label='Outliers', alpha=0.7, edgecolor='black', s=100)  # Circling outliers

# Circle outliers
for index, row in outliers.iterrows():
    plt.scatter(row['time_minutes'], row['Difficulty_Numeric'],
                color='none', edgecolor='red', s=300, linewidth=2)  # Draw circles around outliers

plt.title('Difficulty vs. Time (in minutes)')
plt.xlabel('Time (minutes)')
plt.ylabel('Difficulty')
plt.yticks([1, 2, 3, 4, 5], ['Easy', 'Moderate', 'Intermediate', 'Hard', 'Complex'])  # Set y-ticks to the specified order
plt.xlim(0, 600)  # Set x-axis limits from 0 to 600
plt.grid(True)
plt.legend()
plt.show()

# Check for unique values in the relevant columns
print("Unique time_minutes values:", df['time_minutes'].unique())
print("Unique Difficulty_Numeric values:", df['Difficulty_Numeric'].unique())
print("Outliers identified:\n", df[df['Outlier']])
```




```
Unique time_minutes values: [  3 142 175  48  30 110   4  52   2 225   6   8  20 250  14  78 118  22
  38  37  71 160   9  13   7  11  51  27 275  65  44  18  40  63  24  55
 144  29  25  41  50  35  54  57 590  33  98  88  28 209  74  90  45  93
  26  84  43  17  62  10  16  19  21 125  47  42 168 100  95  60  80  53
 124  15   5  23 185  64 150  46  12  32  36  31  39 155 105  75 195  59
  58  96  70  85 115  73]
Unique Difficulty_Numeric values: [1, 3, 4, 5, 2]
Categories (5, int64): [1 < 2 < 3 < 4 < 5]
Outliers identified:
                                                Image  \
1    https://origami-database.com/wp-content/upload...
2    https://origami-database.com/wp-content/upload...
5    https://origami-database.com/wp-content/upload...
11   https://origami-database.com/wp-content/upload...
18   https://origami-database.com/wp-content/upload...
21   https://origami-database.com/wp-content/upload...
27   https://origami-database.com/wp-content/upload...
39   https://origami-database.com/wp-content/upload...
48   https://origami-database.com/wp-content/upload...
61   https://origami-database.com/wp-content/upload...
65   https://origami-database.com/wp-content/upload...
68   https://origami-database.com/wp-content/upload...
74   https://origami-database.com/wp-content/upload...
76   https://origami-database.com/wp-content/upload...
78   https://origami-database.com/wp-content/upload...
```

```
103  https://origami-database.com/wp-content/upload...
104  https://origami-database.com/wp-content/upload...
108  https://origami-database.com/wp-content/upload...
109  https://origami-database.com/wp-content/upload...
125  https://origami-database.com/wp-content/upload...
137  https://origami-database.com/wp-content/upload...
142  https://origami-database.com/wp-content/upload...
145  https://origami-database.com/wp-content/upload...
160  https://origami-database.com/wp-content/upload...
172  https://origami-database.com/wp-content/upload...
179  https://origami-database.com/wp-content/upload...
215  https://origami-database.com/wp-content/upload...
223  https://origami-database.com/wp-content/upload...
231  https://origami-database.com/wp-content/upload...
237  https://origami-database.com/wp-content/upload...
240  https://origami-database.com/wp-content/upload...
255  https://origami-database.com/wp-content/upload...
268  https://origami-database.com/wp-content/upload...
288  https://origami-database.com/wp-content/upload...
312  https://origami-database.com/wp-content/upload...
331  https://origami-database.com/wp-content/upload...
378  https://origami-database.com/wp-content/upload...

                             Name  \
1             Three-Headed Dragon
2                       Charizard
5                         Pegasus
11                      Tarrasque
18   Greater Death’s Head Hawkmoth
21                        Bulette
27                 Displacer beast
39                        Phantom
48                           Lady
61                 Ancient Dragon
65                    Giant Squid
68                         Sphinx
74                 Mephistopheles
76                        Centaur
78                         Wizard
103         Five-Banded Armadillo
104               Oriental Dragon
108                     Ouroboros
109                        Medusa
125                    Red Wyvern
137                   Stegosaurus
142                    Dimetrodon
145     Winged Sonobe (icosahedron)
160                       Bicycle
172                      Dinosaur
179                          Crow
215           Three-headed Dragon
223               Snapping turtle
231                       Pegasus
237               American Turkey
240             Dilophosaurus 1.8
255                   Stegosaurus
```

```
268                  Orb Weaver
288            Senbazuru (square)
312           Darkness Dragon 2.0
331            Fiery Dragon ver.2
378                      Cerberus

                                           Description    Difficulty  \
1    Bipedal three-headed creature with small wings...  intermediate   
2    Standing three-dimentional dragon from Pokémon...          hard   
5                        Horse with medium-sized wings.          hard   
11   Quadruped armored beast with long tail, scales...       complex   
18   Flying insect with reverse-colored wings, stri...       complex   
21          Armored quadruped creature with large maw.          hard   
27   Six-legged catlike creature with two large ten...          hard   
39   Large conch shell with reverse-colored wave co...       complex   
48               Winged female figure with long skirt.          hard   
61   Dragon with eight large spikes on its head, la...       complex   
65   Cephalopod with two large and eight smaller te...  intermediate   
68      Sitting lion-like creature with abstract face.  intermediate   
74   Standing humanoid devil with horned head, larg...          hard   
76   Creature with muscled, reverse-colored upper b...  intermediate   
78   Humanoid figure with arms stretched, wearing a...          hard   
103  Armadillo with large segmented shell and feet ...  intermediate   
104  Wingless, flying dragon with hollow body and p...  intermediate   
108  Three-dimentional winged snake with large drag...  intermediate   
109  Woman with serpentine lower body and hair made...  intermediate   
125  Wyvern with a horned head and talons with thre...  intermediate   
137  Standing dinosaur with large body, 8 plates on...  intermediate   
142  Quadruped model with reverse-colored sail and ...  intermediate   
145  Modular, winged polyhedron model folded from 3...  intermediate   
160  Bicycle with distinctive frame and reverse-col...          hard   
172  Standing dinosaur with claws and reverse-color...       complex   
179      Standing crow with four digits on each talon.          hard   
215  Three-headed bipedal dragon with horned heads ...  intermediate   
223  3D turtle model with segmented shell and claws...  intermediate   
231  Detailed pegasus model with reverse-colored wi...          hard   
237  Bird model with reverse-colored elements and d...          hard   
240  Standing dinosaur model with open mouth, large...          hard   
255  Quadruped dinosaur with reverse-colored eye, f...          hard   
268        Eight-legged spider with spherical abdomen.          hard   
288  A pattern of crane models, joined together by ...  intermediate   
312  Quadruped dragon model with segmented tail, ta...          hard   
331  Dragon model with a tail ending in three point...          hard   
378  The three-headed hound of Hades from Greek myt...  intermediate   

              Time  time_minutes Difficulty_Numeric  Outlier  
1    2 hr. 22 min.           142                  3     True  
2    2 hr. 55 min.           175                  4     True  
5    1 hr. 50 min.           110                  4     True  
11   3 hr. 45 min.           225                  5     True  
18   4 hr. 10 min.           250                  5     True  
21   1 hr. 58 min.           118                  4     True  
27   2 hr. 40 min.           160                  4     True  
39   4 hr. 35 min.           275                  5     True  
48   2 hr. 24 min.           144                  4     True  
61   9 hr. 50 min.           590                  5     True  
65   1 hr. 38 min.            98                  3     True  
```

```
68   1 hr. 28 min.           88                  3     True
74   3 hr. 29 min.          209                  4     True
76   1 hr. 30 min.           90                  3     True
78   1 hr. 33 min.           93                  4     True
103  1 hr. 50 min.          110                  3     True
104   2 hr. 5 min.          125                  3     True
108  2 hr. 24 min.          144                  3     True
109  2 hr. 48 min.          168                  3     True
125  1 hr. 40 min.          100                  3     True
137  1 hr. 35 min.           95                  3     True
142  1 hr. 35 min.           95                  3     True
145   2 hr. 4 min.          124                  3     True
160   3 hr. 5 min.          185                  4     True
172  2 hr. 30 min.          150                  5     True
179  1 hr. 50 min.          110                  4     True
215  1 hr. 35 min.           95                  3     True
223  1 hr. 50 min.          110                  3     True
231  2 hr. 35 min.          155                  4     True
237  1 hr. 45 min.          105                  4     True
240  3 hr. 15 min.          195                  4     True
255  1 hr. 33 min.           93                  4     True
268  1 hr. 40 min.          100                  4     True
288  1 hr. 36 min.           96                  3     True
312  2 hr. 40 min.          160                  4     True
331  1 hr. 55 min.          115                  4     True
378  1 hr. 30 min.           90                  3     True
```

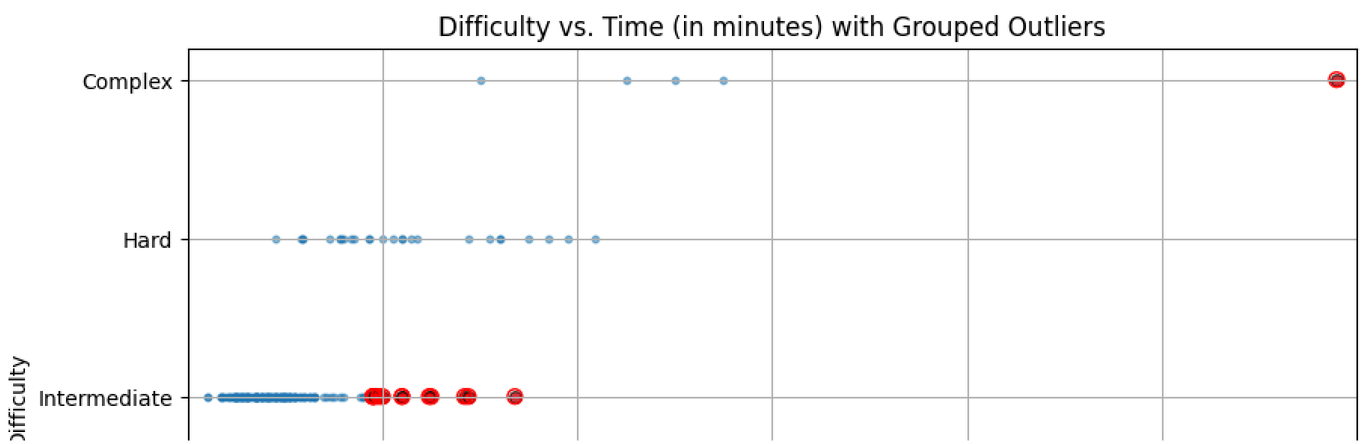
highlighted outliers per difficulty group. Allows all difficulties to be present

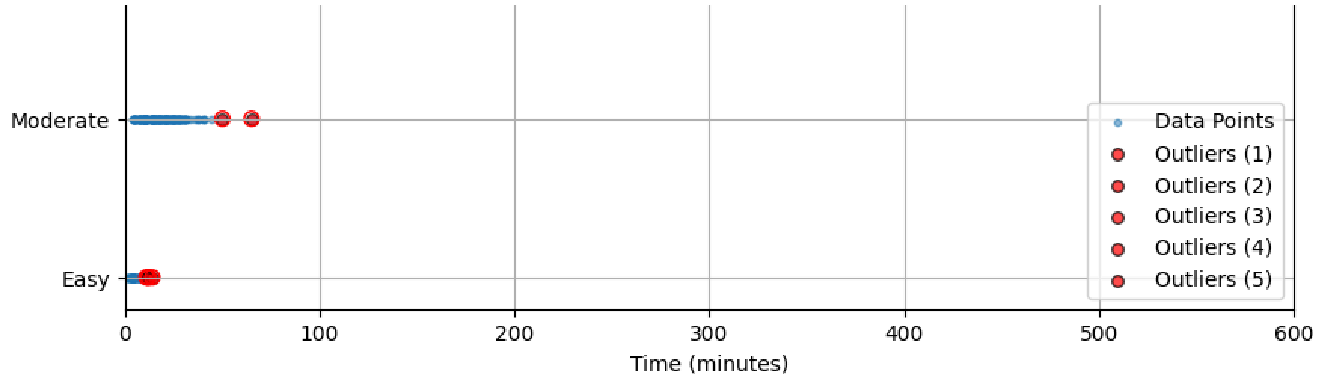
```
import pandas as pd
import matplotlib.pyplot as plt
import re

# Load the data
df = pd.read_csv('Origo_Database.csv')

# Function to convert time to total minutes
def convert_to_minutes(time_str):
    hours = 0
    minutes = 0
    # Extract hours and minutes using regex
    hour_match = re.search(r'(\d+)\s*hr', time_str)
    if hour_match:
        hours = int(hour_match.group(1))
    minute_match = re.search(r'(\d+)\s*min', time_str)
    if minute_match:
        minutes = int(minute_match.group(1))
    return hours * 60 + minutes

# Apply the function to the Time column
df['time_minutes'] = df['Time'].apply(convert_to_minutes)

# Clean up the Difficulty values
df['Difficulty'] = df['Difficulty'].str.strip().str.lower()

# Create the mapping
difficulty_mapping = {
    'easy': 1,
    'moderate': 2,
    'intermediate': 3,
    'hard': 4,
    'complex': 5
}

# Map the Difficulty column to the new numeric column
df['Difficulty_Numeric'] = df['Difficulty'].map(difficulty_mapping)

# Fill NA values in Difficulty_Numeric if any exist
df['Difficulty_Numeric'] = df['Difficulty_Numeric'].fillna(1)

# Ensure Difficulty_Numeric is treated as an ordered categorical
df['Difficulty_Numeric'] = pd.Categorical(df['Difficulty_Numeric'], categories=[1, 2, 3, 4, 5], ordered=True)

# Create a function to identify outliers within each group
def find_outliers(group):
    Q1 = group['time_minutes'].quantile(0.25)
    Q3 = group['time_minutes'].quantile(0.75)
```

```
        IQR = Q3 - Q1
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR
        return group[(group['time_minutes'] < lower_bound) | (group['time_minutes'] > upper_bound)]

    # Identify outliers for each difficulty group
    outliers_per_group = df.groupby('Difficulty_Numeric').apply(find_outliers).reset_index(drop=True)

    # Print outliers for each group
    print("Outliers for each difficulty group:\n", outliers_per_group)

    # Create the scatter plot
    plt.figure(figsize=(10, 6))
    plt.scatter(df['time_minutes'], df['Difficulty_Numeric'], alpha=0.5, label='Data Points', s=10)  # Smaller points

    # Highlight outliers in the plot
    for difficulty in df['Difficulty_Numeric'].cat.categories:
        outliers = outliers_per_group[outliers_per_group['Difficulty_Numeric'] == difficulty]
        plt.scatter(outliers['time_minutes'], outliers['Difficulty_Numeric'],
                    color='red', label=f'Outliers ({difficulty})', alpha=0.7, edgecolor='black', s=30)  # Smaller outlier points

        # Circle outliers
        for index, row in outliers.iterrows():
            plt.scatter(row['time_minutes'], row['Difficulty_Numeric'],
                        color='none', edgecolor='red', s=50, linewidth=1)  # Draw circles around outliers

    plt.title('Difficulty vs. Time (in minutes) with Grouped Outliers')
    plt.xlabel('Time (minutes)')
    plt.ylabel('Difficulty')
    plt.yticks([1, 2, 3, 4, 5], ['Easy', 'Moderate', 'Intermediate', 'Hard', 'Complex'])  # Set y-ticks to the specified order
    plt.xlim(0, 600)  # Set x-axis limits from 0 to 600
    plt.grid(True)
    plt.legend()
    plt.show()
```

```
<ipython-input-181-ca935ad89670>:55: FutureWarning:

The default of observed=False is deprecated and will be changed to True in a future version of pandas. Pass observed=False to retain current behavior or observec

<ipython-input-181-ca935ad89670>:55: DeprecationWarning:

DataFrameGroupBy.apply operated on the grouping columns. This behavior is deprecated, and in a future version of pandas the grouping columns will be excluded frc

Outliers for each difficulty group:
                                                Image  \
0   https://origami-database.com/wp-content/upload...
1   https://origami-database.com/wp-content/upload...
2   https://origami-database.com/wp-content/upload...
3   https://origami-database.com/wp-content/upload...
4   https://origami-database.com/wp-content/upload...
5   https://origami-database.com/wp-content/upload...
6   https://origami-database.com/wp-content/upload...
7   https://origami-database.com/wp-content/upload...
8   https://origami-database.com/wp-content/upload...
9   https://origami-database.com/wp-content/upload...
10  https://origami-database.com/wp-content/upload...
11  https://origami-database.com/wp-content/upload...
12  https://origami-database.com/wp-content/upload...
13  https://origami-database.com/wp-content/upload...
14  https://origami-database.com/wp-content/upload...
15  https://origami-database.com/wp-content/upload...
16  https://origami-database.com/wp-content/upload...
17  https://origami-database.com/wp-content/upload...
18  https://origami-database.com/wp-content/upload...
19  https://origami-database.com/wp-content/upload...

                          Name  \
0                          Bat
1                  Mount Fuji
2               Star box masu
3                    Seahorse
4       Fire breathing dragon
5              Christmas tree
6          Three-Headed Dragon
7                 Giant Squid
8       Five-Banded Armadillo
9             Oriental Dragon
10                  Ouroboros
11                     Medusa
12                 Red Wyvern
13                Stegosaurus
14                 Dimetrodon
15  Winged Sonobe (icosahedron)
16         Three-headed Dragon
17            Snapping turtle
18           Senbazuru (square)
19             Ancient Dragon

                                          Description    Difficulty  \
0   Abstract bat model with feet and distinctive e...          easy
1                  Volcano with reverse-colored peak.          easy
```

```
2   Square box with a lid, with a star-shaped symbol.         easy
3                 Seahorse model with segmented tail.         easy
4   Standing, bulky dragon with small wings that b...     moderate
5   Multi-sheet christmas tree with segmented foli...     moderate
6   Bipedal three-headed creature with small wings...  intermediate
7   Cephalopod with two large and eight smaller te...  intermediate
8   Armadillo with large segmented shell and feet ...  intermediate
9   Wingless, flying dragon with hollow body and p...  intermediate
10  Three-dimentional winged snake with large drag...  intermediate
11  Woman with serpentine lower body and hair made...  intermediate
12  Wyvern with a horned head and talons with thre...  intermediate
13  Standing dinosaur with large body, 8 plates on...  intermediate
14  Quadruped model with reverse-colored sail and ...  intermediate
15  Modular, winged polyhedron model folded from 3...  intermediate
16  Three-headed bipedal dragon with horned heads ...  intermediate
17  3D turtle model with segmented shell and claws...  intermediate
18  A pattern of crane models, joined together by ...  intermediate
19  Dragon with eight large spikes on its head, la...      complex

             Time  time_minutes Difficulty_Numeric
0         11 min.            11                  1
1         12 min.            12                  1
2         14 min.            14                  1
3         12 min.            12                  1
4         50 min.            50                  2
5    1 hr. 5 min.            65                  2
6   2 hr. 22 min.           142                  3
7   1 hr. 38 min.            98                  3
8   1 hr. 50 min.           110                  3
9    2 hr. 5 min.           125                  3
10  2 hr. 24 min.           144                  3
11  2 hr. 48 min.           168                  3
12  1 hr. 40 min.           100                  3
13  1 hr. 35 min.            95                  3
14  1 hr. 35 min.            95                  3
15   2 hr. 4 min.           124                  3
16  1 hr. 35 min.            95                  3
17  1 hr. 50 min.           110                  3
18  1 hr. 36 min.            96                  3
19  9 hr. 50 min.           590                  5
```




Moderate
Easy
Data Points
Outliers (1)
Outliers (2)
Outliers (3)
Outliers (4)
Outliers (5)
0
100
200
300
400
500
600
Time (minutes)

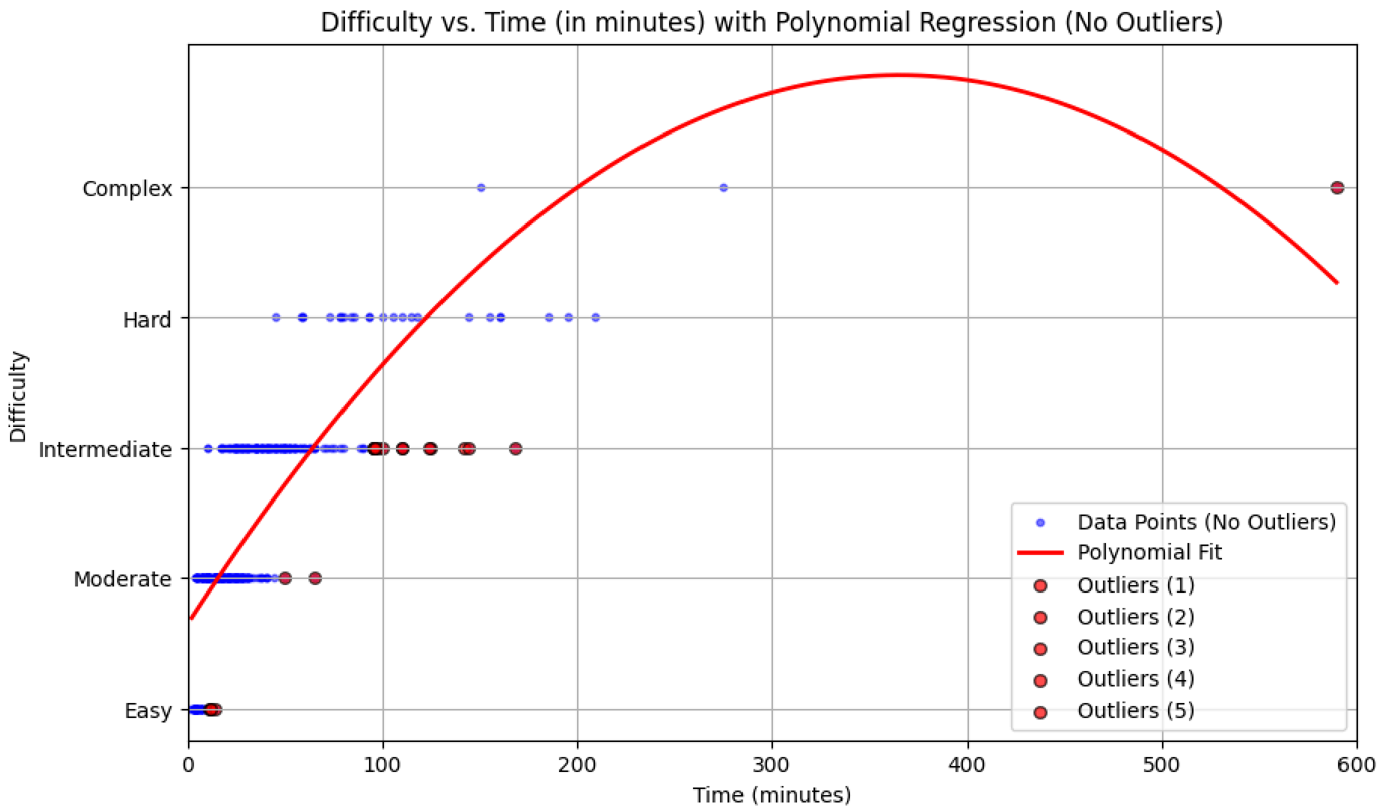
Plots polynomial model without outliers

```
import pandas as pd
import matplotlib.pyplot as plt
import re
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score

# Load the data
df = pd.read_csv('Origo_Database.csv')

# Function to convert time to total minutes
def convert_to_minutes(time_str):
    hours = 0
    minutes = 0
    # Extract hours and minutes using regex
    hour_match = re.search(r'(\d+)\s*hr', time_str)
    if hour_match:
        hours = int(hour_match.group(1))
    minute_match = re.search(r'(\d+)\s*min', time_str)
    if minute_match:
        minutes = int(minute_match.group(1))
    return hours * 60 + minutes

# Apply the function to the Time column
df['time_minutes'] = df['Time'].apply(convert_to_minutes)

# Clean up the Difficulty values
df['Difficulty'] = df['Difficulty'].str.strip().str.lower()

# Create the mapping
difficulty_mapping = {
    'easy': 1,
    'moderate': 2,
    'intermediate': 3,
    'hard': 4,
    'complex': 5
}

# Map the Difficulty column to the new numeric column
df['Difficulty_Numeric'] = df['Difficulty'].map(difficulty_mapping)

# Fill NA values in Difficulty_Numeric if any exist
df['Difficulty_Numeric'] = df['Difficulty_Numeric'].fillna(1)

# Ensure Difficulty_Numeric is treated as an ordered categorical
df['Difficulty_Numeric'] = pd.Categorical(df['Difficulty_Numeric'], categories=[1, 2, 3, 4, 5], ordered=True)

# Create a function to identify outliers within each group
```

```
def find_outliers(group):
    Q1 = group['time_minutes'].quantile(0.25)
    Q3 = group['time_minutes'].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    return group[(group['time_minutes'] < lower_bound) | (group['time_minutes'] > upper_bound)]

# Identify outliers for each difficulty group
outliers_per_group = df.groupby('Difficulty_Numeric').apply(find_outliers).reset_index(drop=True)

# Filter out the outliers from the original DataFrame
df_no_outliers = df[~df.index.isin(outliers_per_group.index)]

# Prepare data for polynomial regression
X = df_no_outliers['time_minutes'].values.reshape(-1, 1)  # Reshape for sklearn
y = df_no_outliers['Difficulty_Numeric'].values

# Create polynomial features
poly = PolynomialFeatures(degree=2)
X_poly = poly.fit_transform(X)

# Fit the polynomial regression model
model = LinearRegression()
model.fit(X_poly, y)

# Create predictions for the polynomial regression line
y_pred = model.predict(X_poly)

# Calculate R-squared
r_squared = r2_score(y, y_pred)

# Output the results
print(f"R-squared: {r_squared:.3f}")
print(f"Coefficients: {model.coef_}")
print(f"Intercept: {model.intercept_:.3f}")

# Create a scatter plot with the polynomial regression line
plt.figure(figsize=(10, 6))
plt.scatter(df_no_outliers['time_minutes'], df_no_outliers['Difficulty_Numeric'],
            color='blue', alpha=0.5, label='Data Points (No Outliers)', s=10)  # Smaller points

# Plot polynomial regression line
x_range = np.linspace(df_no_outliers['time_minutes'].min(), df_no_outliers['time_minutes'].max(), 100).reshape(-1, 1)
y_range_poly = model.predict(poly.transform(x_range))
plt.plot(x_range, y_range_poly, color='red', label='Polynomial Fit', linewidth=2)

# Highlight outliers in the plot
for difficulty in df['Difficulty_Numeric'].cat.categories:
    outliers = outliers_per_group[outliers_per_group['Difficulty_Numeric'] == difficulty]
    plt.scatter(outliers['time_minutes'], outliers['Difficulty_Numeric'],
                color='red', label=f'Outliers ({difficulty})', alpha=0.7, edgecolor='black', s=30)  # Smaller outlier points
```

```
plt.title('Difficulty vs. Time (in minutes) with Polynomial Regression (No Outliers)')
plt.xlabel('Time (minutes)')
plt.ylabel('Difficulty')
plt.yticks([1, 2, 3, 4, 5], ['Easy', 'Moderate', 'Intermediate', 'Hard', 'Complex'])  # Set y-ticks to the specified order
plt.xlim(0, 600)  # Set x-axis limits from 0 to 600
plt.grid(True)
plt.legend()
plt.show()
```

```
<ipython-input-182-82986f919fc7>:58: FutureWarning:

The default of observed=False is deprecated and will be changed to True in a future version of pandas. Pass observed=False to retain current behavior or observec

<ipython-input-182-82986f919fc7>:58: DeprecationWarning:

DataFrameGroupBy.apply operated on the grouping columns. This behavior is deprecated, and in a future version of pandas the grouping columns will be excluded frc

R-squared: 0.612
Coefficients: [ 0.00000000e+00  2.30397037e-02 -3.15266266e-05]
Intercept: 1.650
```




```
!pip install dash
```

```
Collecting dash
  Downloading dash-2.18.1-py3-none-any.whl.metadata (10 kB)
Requirement already satisfied: Flask<3.1,>=1.0.4 in /usr/local/lib/python3.10/dist-packages (from dash) (2.2.5)
Requirement already satisfied: Werkzeug<3.1 in /usr/local/lib/python3.10/dist-packages (from dash) (3.0.6)
Requirement already satisfied: plotly>=5.0.0 in /usr/local/lib/python3.10/dist-packages (from dash) (5.24.1)
Collecting dash-html-components==2.0.0 (from dash)
  Downloading dash_html_components-2.0.0-py3-none-any.whl.metadata (3.8 kB)
Collecting dash-core-components==2.0.0 (from dash)
```

```
  Downloading dash_core_components-2.0.0-py3-none-any.whl.metadata (2.9 kB)
Collecting dash-table==5.0.0 (from dash)
  Downloading dash_table-5.0.0-py3-none-any.whl.metadata (2.4 kB)
Requirement already satisfied: importlib-metadata in /usr/local/lib/python3.10/dist-packages (from dash) (8.5.0)
Requirement already satisfied: typing-extensions>=4.1.1 in /usr/local/lib/python3.10/dist-packages (from dash) (4.12.2)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from dash) (2.32.3)
Collecting retrying (from dash)
  Downloading retrying-1.3.4-py3-none-any.whl.metadata (6.9 kB)
Requirement already satisfied: nest-asyncio in /usr/local/lib/python3.10/dist-packages (from dash) (1.6.0)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from dash) (75.1.0)
Requirement already satisfied: Jinja2>=3.0 in /usr/local/lib/python3.10/dist-packages (from Flask<3.1,>=1.0.4->dash) (3.1.4)
Requirement already satisfied: itsdangerous>=2.0 in /usr/local/lib/python3.10/dist-packages (from Flask<3.1,>=1.0.4->dash) (2.2.0)
Requirement already satisfied: click>=8.0 in /usr/local/lib/python3.10/dist-packages (from Flask<3.1,>=1.0.4->dash) (8.1.7)
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from plotly>=5.0.0->dash) (9.0.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from plotly>=5.0.0->dash) (24.1)
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.10/dist-packages (from Werkzeug<3.1->dash) (3.0.2)
Requirement already satisfied: zipp>=3.20 in /usr/local/lib/python3.10/dist-packages (from importlib-metadata->dash) (3.20.2)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->dash) (3.4.0)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->dash) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->dash) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->dash) (2024.8.30)
Requirement already satisfied: six>=1.7.0 in /usr/local/lib/python3.10/dist-packages (from retrying->dash) (1.16.0)
Downloading dash-2.18.1-py3-none-any.whl (7.5 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7.5/7.5 MB 57.0 MB/s eta 0:00:00
Downloading dash_core_components-2.0.0-py3-none-any.whl (3.8 kB)
Downloading dash_html_components-2.0.0-py3-none-any.whl (4.1 kB)
Downloading dash_table-5.0.0-py3-none-any.whl (3.9 kB)
Downloading retrying-1.3.4-py3-none-any.whl (11 kB)
Installing collected packages: dash-table, dash-html-components, dash-core-components, retrying, dash
Successfully installed dash-2.18.1 dash-core-components-2.0.0 dash-html-components-2.0.0 dash-table-5.0.0 retrying-1.3.4
```

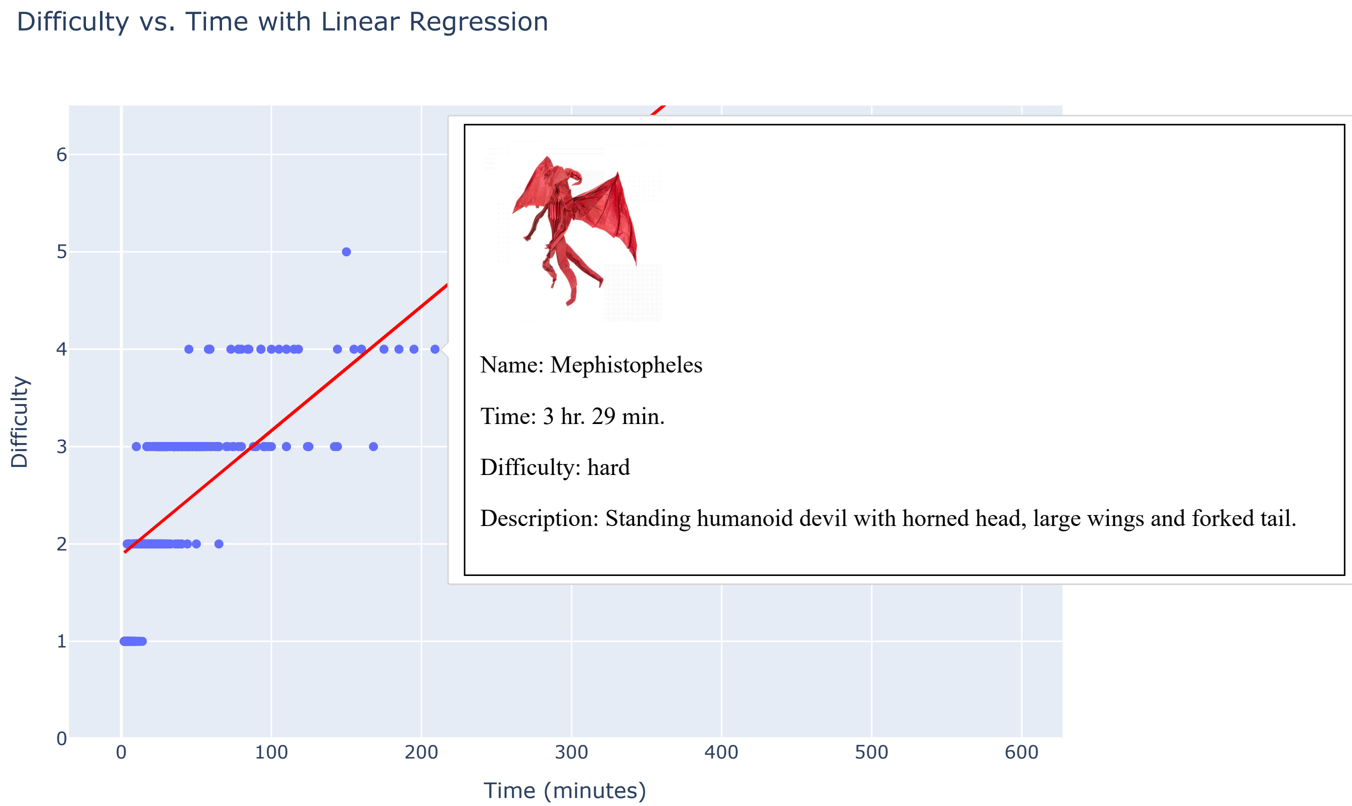
Plotting interactive scatter plot with descriptions and images of each origami model

```
import pandas as pd
import numpy as np
import re
from dash import Dash, dcc, html, Input, Output
import plotly.graph_objects as go
from sklearn.linear_model import LinearRegression

# Load the data
df = pd.read_csv('Origo_Database.csv')

# Function to convert time to total minutes
def convert_to_minutes(time_str):
    hours = 0
    minutes = 0
    hour_match = re.search(r'(\d+)\s*hr', time_str)
    if hour_match:
        hours = int(hour_match.group(1))
    minute_match = re.search(r'(\d+)\s*min', time_str)
    if minute_match:
        minutes = int(minute_match.group(1))
```

```
        return hours * 60 + minutes

    # Apply the function to the Time column
    df['time_minutes'] = df['Time'].apply(convert_to_minutes)

    # Clean up the Difficulty values
    df['Difficulty'] = df['Difficulty'].str.strip().str.lower()

    # Create the mapping
    difficulty_mapping = {
        'easy': 1,
        'moderate': 2,
        'intermediate': 3,
        'hard': 4,
        'complex': 5
    }

    # Map the Difficulty column to the new numeric column
    df['Difficulty_Numeric'] = df['Difficulty'].map(difficulty_mapping)

    # Fill NA values in Difficulty_Numeric if any exist
    df['Difficulty_Numeric'] = df['Difficulty_Numeric'].fillna(1)

    # Prepare data for linear regression
    X = df['time_minutes'].values.reshape(-1, 1)
    y = df['Difficulty_Numeric'].values

    # Fit a linear regression model
    model = LinearRegression()
    model.fit(X, y)

    # Create predictions for the regression line
    y_pred = model.predict(X)

    # Create the Dash app
    app = Dash(__name__)

    app.layout = html.Div([
        dcc.Graph(id='scatter-plot', style={'height': '80vh'}),
        html.Div(id='button-container', style={'display': 'flex', 'justify-content': 'center', 'margin-top': '10px'}),
        html.Button("Reset Graph", id='reset-button', n_clicks=0),
        dcc.Tooltip(id='graph-tooltip',
                    show=False,
                    children=[],
                    style={"pointer-events": "none", 'max-width': '800px', 'background-color': 'lightblue'}),
    ])

    @app.callback(
        Output('scatter-plot', 'figure'),
        Input('scatter-plot', 'hoverData'),
        Input('reset-button', 'n_clicks'),
```

```
        Input('scatter-plot', 'clickData')
    )
    def update_graph(hoverData, n_clicks, clickData):
        # Create the figure
        fig = go.Figure()

        # Add scatter plot for points
        fig.add_trace(go.Scatter(
            x=df['time_minutes'],
            y=df['Difficulty_Numeric'],
            mode='markers',
            text=df['Name'],
            hoverinfo='none',
            customdata=df[['Description', 'Image']],
            name='Points',
        ))

        # Add regression line
        fig.add_trace(go.Scatter(
            x=df['time_minutes'],
            y=y_pred,
            mode='lines',
            name='Line of Best Fit',
            line=dict(color='red')
        ))

        # Update layout with y-axis limit
        fig.update_layout(
            title='Difficulty vs. Time with Linear Regression',
            xaxis_title='Time (minutes)',
            yaxis_title='Difficulty',
            height=600,
            width=900
        )

        # Set y-axis range
        fig.update_yaxes(range=[0, 6.5])  # Limit y-axis from 0 to 6.5

        # If the reset button is clicked or empty space is clicked, return the initial state
        if n_clicks > 0 or clickData is None:
            return fig

        return fig

    @app.callback(
        Output('graph-tooltip', 'show'),
        Output('graph-tooltip', 'bbox'),
        Output('graph-tooltip', 'children'),
        Input('scatter-plot', 'hoverData'),
    )
    def display_hover(hoverData):
        if hoverData:
```

```
        pt = hoverData['points'][0]
        bbox = pt['bbox']  # Get bounding box coordinates
        index = pt['pointIndex']  # Get index of the point hovered
        img_src = df['Image'][index]  # Get the image URL

        # Create tooltip content with image
        children = [
            html.Div([
                html.Img(src=img_src, style={"width": "120px", "height": "auto"}),
                html.P(f"Name: {df['Name'][index]}"),
                html.P(f"Time: {df['Time'][index]}"),
                html.P(f"Difficulty: {df['Difficulty'][index]}"),
                html.P(f"Description: {df['Description'][index]}"),
            ], style={
                'width': '585px',
                'height': '300px',
                'overflow': 'auto',
                'border': '1px solid black',
                'padding': '10px',
                'box-sizing': 'border-box',
                'background-color': 'lightblue'
            })
        ]
        return True, bbox, children  # Show tooltip with content

    return False, None, []  # No tooltip if not hovering

if __name__ == '__main__':
    app.run_server(debug=True)
```

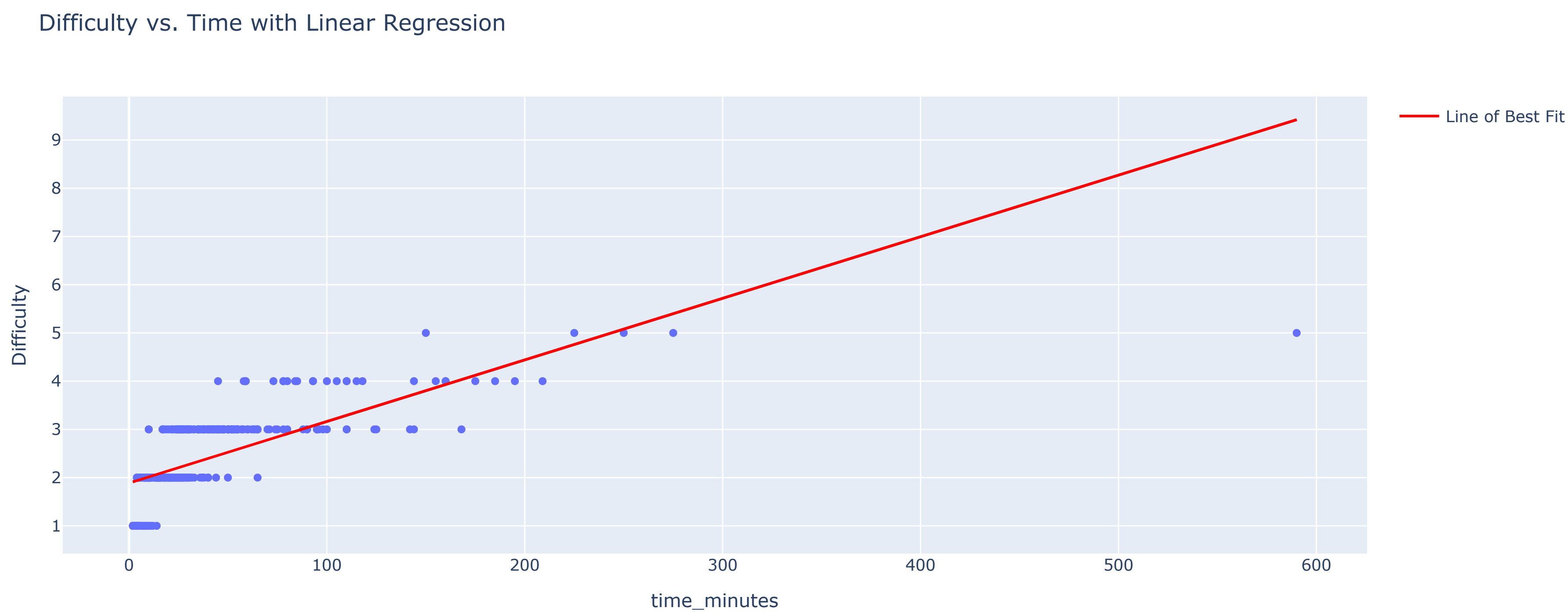



```
import pandas as pd
import numpy as np
import re
import plotly.express as px
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score

# Load the data
df = pd.read_csv('Origo_Database.csv')

# Function to convert time to total minutes
def convert_to_minutes(time_str):
    hours = 0
    minutes = 0
    hour_match = re.search(r'(\d+)\s*hr', time_str)
```

```
        if hour_match:
            hours = int(hour_match.group(1))
        minute_match = re.search(r'(\d+)\s*min', time_str)
        if minute_match:
            minutes = int(minute_match.group(1))
        return hours * 60 + minutes

    # Apply the function to the Time column
    df['time_minutes'] = df['Time'].apply(convert_to_minutes)

    # Clean up the Difficulty values
    df['Difficulty'] = df['Difficulty'].str.strip().str.lower()

    # Create the mapping
    difficulty_mapping = {
        'easy': 1,
        'moderate': 2,
        'intermediate': 3,
        'hard': 4,
        'complex': 5
    }

    # Map the Difficulty column to the new numeric column
    df['Difficulty_Numeric'] = df['Difficulty'].map(difficulty_mapping)

    # Fill NA values in Difficulty_Numeric if any exist
    df['Difficulty_Numeric'] = df['Difficulty_Numeric'].fillna(1)

    # Prepare data for linear regression
    X = df['time_minutes'].values.reshape(-1, 1)  # Reshape for sklearn
    y = df['Difficulty_Numeric'].values

    # Fit a linear regression model
    model = LinearRegression()
    model.fit(X, y)

    # Create predictions for the regression line
    y_pred = model.predict(X)

    # Calculate R-squared
    r_squared = r2_score(y, y_pred)

    # Get coefficients
    slope = model.coef_[0]
    intercept = model.intercept_

    # Output the results
    print(f"R-squared: {r_squared:.3f}")
    print(f"Slope (Coefficient of Time): {slope:.3f}")
    print(f"Intercept: {intercept:.3f}")
```

```
# Create a scatter plot with the regression line
fig = px.scatter(df, x='time_minutes', y='Difficulty_Numeric',
                 title='Difficulty vs. Time with Linear Regression',
                 labels={'Difficulty_Numeric': 'Difficulty'},
                 hover_data={'Time': True, 'Difficulty': True, 'Name': True, 'Description': True})

# Add the regression line to the plot
fig.add_scatter(x=df['time_minutes'], y=y_pred, mode='lines', name='Line of Best Fit', line=dict(color='red'))

# Show the figure
fig.show()
```

```
R-squared: 0.489
Slope (Coefficient of Time): 0.013
Intercept: 1.886
```



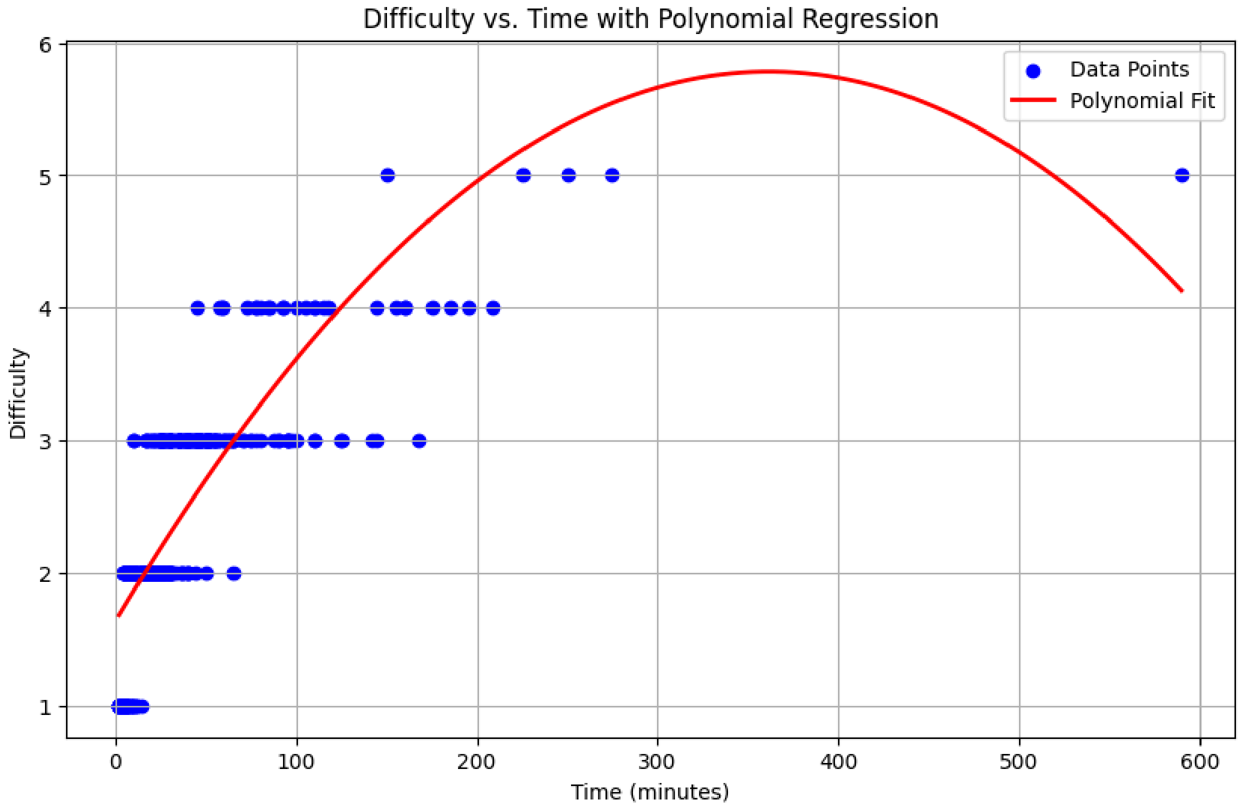

Polynomial regression (worse than regression excluding outliers)

```
import pandas as pd
import numpy as np
import re
```

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
import matplotlib.pyplot as plt

# Load the data
df = pd.read_csv('Origo_Database.csv')  # Ensure the file path is correct

# Function to convert time to total minutes
def convert_to_minutes(time_str):
    hours = 0
    minutes = 0
    hour_match = re.search(r'(\d+)\s*hr', time_str)
    if hour_match:
        hours = int(hour_match.group(1))
    minute_match = re.search(r'(\d+)\s*min', time_str)
    if minute_match:
        minutes = int(minute_match.group(1))
    return hours * 60 + minutes

# Apply the function to the Time column
df['time_minutes'] = df['Time'].apply(convert_to_minutes)

# Clean up the Difficulty values
df['Difficulty'] = df['Difficulty'].str.strip().str.lower()

# Create the mapping
difficulty_mapping = {
    'easy': 1,
    'moderate': 2,
    'intermediate': 3,
    'hard': 4,
    'complex': 5
}

# Map the Difficulty column to the new numeric column
df['Difficulty_Numeric'] = df['Difficulty'].map(difficulty_mapping)

# Fill NA values in Difficulty_Numeric if any exist
df['Difficulty_Numeric'] = df['Difficulty_Numeric'].fillna(1)

# Prepare data for polynomial regression
X = df['time_minutes'].values.reshape(-1, 1)
y = df['Difficulty_Numeric'].values

# Create polynomial features
poly = PolynomialFeatures(degree=2)
X_poly = poly.fit_transform(X)

# Fit the polynomial regression model
model = LinearRegression()
```

```
    model.fit(X_poly, y)

    # Create predictions for the polynomial regression line
    y_pred = model.predict(X_poly)

    # Calculate R-squared
    r_squared = r2_score(y, y_pred)

    # Get coefficients and intercept
    coefficients = model.coef_
    intercept = model.intercept_

    # Output the results
    print(f"R-squared: {r_squared:.3f}")
    print(f"Coefficients: {coefficients}")
    print(f"Intercept: {intercept:.3f}")

    # Create a scatter plot with the regression line
    plt.figure(figsize=(10, 6))
    plt.scatter(df['time_minutes'], df['Difficulty_Numeric'], color='blue', label='Data Points')

    # Create a smooth line for the polynomial fit
    x_range = np.linspace(df['time_minutes'].min(), df['time_minutes'].max(), 100).reshape(-1, 1)
    x_range_poly = poly.transform(x_range)
    y_range_pred = model.predict(x_range_poly)

    plt.plot(x_range, y_range_pred, color='red', label='Polynomial Fit', linewidth=2)
    plt.title('Difficulty vs. Time with Polynomial Regression')
    plt.xlabel('Time (minutes)')
    plt.ylabel('Difficulty')
    plt.legend()
    plt.grid()
    plt.show()
```

```
R-squared: 0.634
Coefficients: [ 0.00000000e+00  2.29284418e-02 -3.16995584e-05]
Intercept: 1.637
```



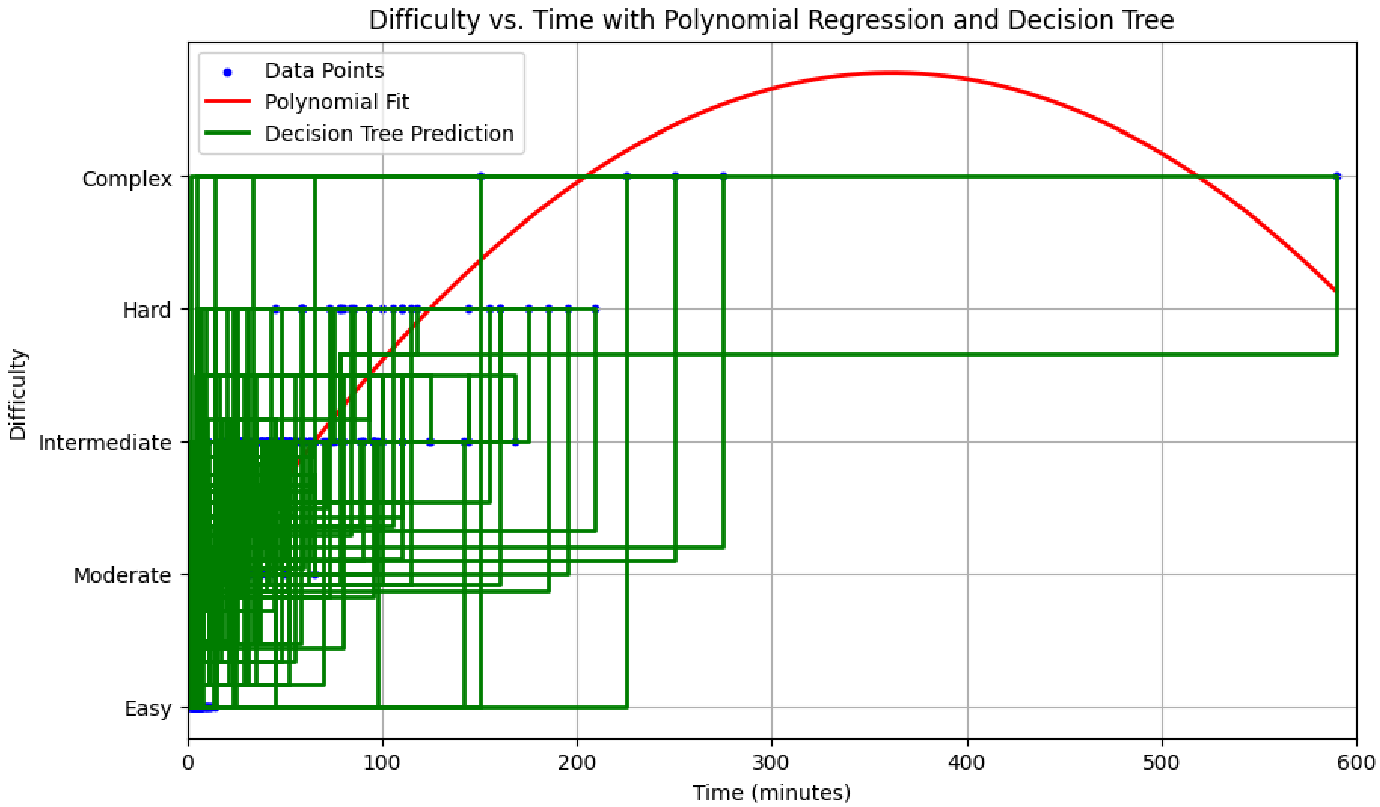

Plot of decision tree overlayed with polynomial regression

```
import pandas as pd
import numpy as np
import re
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import r2_score

# Load the data
df = pd.read_csv('Origo_Database.csv')  # Ensure the file path is correct

# Function to convert time to total minutes
def convert_to_minutes(time_str):
    hours = 0
```

```
        minutes = 0
        hour_match = re.search(r'(\d+)\s*hr', time_str)
        if hour_match:
            hours = int(hour_match.group(1))
        minute_match = re.search(r'(\d+)\s*min', time_str)
        if minute_match:
            minutes = int(minute_match.group(1))
        return hours * 60 + minutes

    # Apply the function to the Time column
    df['time_minutes'] = df['Time'].apply(convert_to_minutes)

    # Clean up the Difficulty values
    df['Difficulty'] = df['Difficulty'].str.strip().str.lower()

    # Create the mapping
    difficulty_mapping = {
        'easy': 1,
        'moderate': 2,
        'intermediate': 3,
        'hard': 4,
        'complex': 5
    }

    # Map the Difficulty column to the new numeric column
    df['Difficulty_Numeric'] = df['Difficulty'].map(difficulty_mapping)

    # Fill NA values in Difficulty_Numeric if any exist
    df['Difficulty_Numeric'] = df['Difficulty_Numeric'].fillna(1)

    # Prepare data for polynomial regression
    X = df['time_minutes'].values.reshape(-1, 1)
    y = df['Difficulty_Numeric'].values

    # Create polynomial features
    poly = PolynomialFeatures(degree=2)
    X_poly = poly.fit_transform(X)

    # Fit the polynomial regression model
    linear_model = LinearRegression()
    linear_model.fit(X_poly, y)

    # Create predictions for the polynomial regression line
    y_pred_poly = linear_model.predict(X_poly)

    # Fit a decision tree regressor
    tree_model = DecisionTreeRegressor(max_depth=9)
    tree_model.fit(X, y)

    # Create predictions for the decision tree
    y_pred_tree = tree_model.predict(X)
```

```
# Calculate R-squared values
r_squared_poly = r2_score(y, y_pred_poly)
r_squared_tree = r2_score(y, y_pred_tree)

# Output the results
print(f"Polynomial R-squared: {r_squared_poly:.3f}")
print(f"Decision Tree R-squared: {r_squared_tree:.3f}")

# Create a scatter plot with the regression lines
plt.figure(figsize=(10, 6))
plt.scatter(df['time_minutes'], df['Difficulty_Numeric'], color='blue', label='Data Points', s=10)

# Create a smooth line for the polynomial fit
x_range = np.linspace(df['time_minutes'].min(), df['time_minutes'].max(), 100).reshape(-1, 1)
x_range_poly = poly.transform(x_range)
y_range_pred_poly = linear_model.predict(x_range_poly)

plt.plot(x_range, y_range_pred_poly, color='red', label='Polynomial Fit', linewidth=2)

# Plot the decision tree predictions as a step function
plt.step(X.flatten(), y_pred_tree, color='green', label='Decision Tree Prediction', where='post', linewidth=2)

plt.title('Difficulty vs. Time with Polynomial Regression and Decision Tree')
plt.xlabel('Time (minutes)')
plt.ylabel('Difficulty')
plt.yticks([1, 2, 3, 4, 5], ['Easy', 'Moderate', 'Intermediate', 'Hard', 'Complex'])  # Set y-ticks to the specified order
plt.xlim(0, 600)  # Set x-axis limits from 0 to 600
plt.grid()
plt.legend()
plt.show()
```

```
Polynomial R-squared: 0.634
Decision Tree R-squared: 0.817
```




```
!pip install catboost
```

```
Collecting catboost
  Downloading catboost-1.2.7-cp310-cp310-manylinux2014_x86_64.whl.metadata (1.2 kB)
Requirement already satisfied: graphviz in /usr/local/lib/python3.10/dist-packages (from catboost) (0.20.3)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (from catboost) (3.8.0)
Requirement already satisfied: numpy<2.0,>=1.16.0 in /usr/local/lib/python3.10/dist-packages (from catboost) (1.26.4)
Requirement already satisfied: pandas>=0.24 in /usr/local/lib/python3.10/dist-packages (from catboost) (2.2.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from catboost) (1.13.1)
Requirement already satisfied: plotly in /usr/local/lib/python3.10/dist-packages (from catboost) (5.24.1)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from catboost) (1.16.0)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.24->catboost) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.24->catboost) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.24->catboost) (2024.2)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->catboost) (1.3.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib->catboost) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->catboost) (4.54.1)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->catboost) (1.4.7)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->catboost) (24.1)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->catboost) (10.4.0)
```

```
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->catboost) (3.2.0)
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from plotly->catboost) (9.0.0)
Downloading catboost-1.2.7-cp310-cp310-manylinux2014_x86_64.whl (98.7 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 98.7/98.7 MB 3.0 MB/s eta 0:00:00
Installing collected packages: catboost
Successfully installed catboost-1.2.7
```

```
!pip install catboost
```

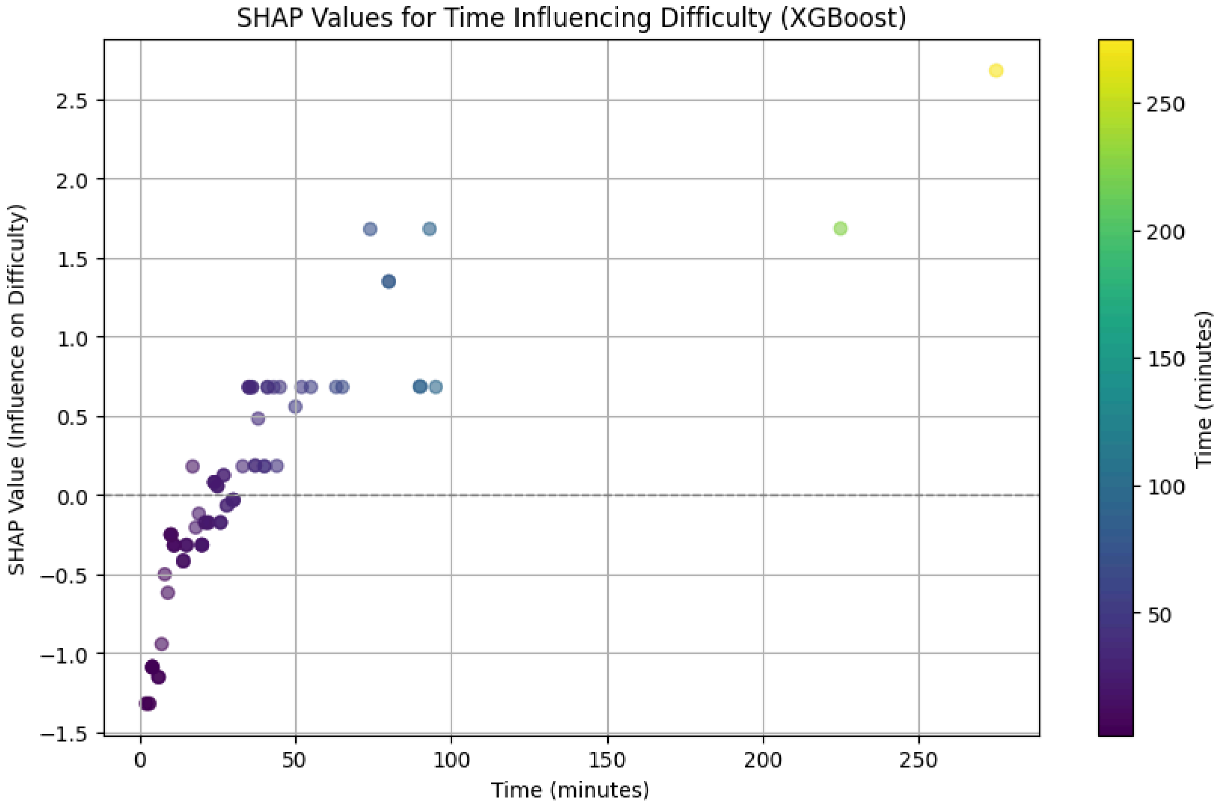
```
Collecting catboost
  Downloading catboost-1.2.7-cp310-cp310-manylinux2014_x86_64.whl.metadata (1.2 kB)
Requirement already satisfied: graphviz in /usr/local/lib/python3.10/dist-packages (from catboost) (0.20.3)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (from catboost) (3.8.0)
Requirement already satisfied: numpy<2.0,>=1.16.0 in /usr/local/lib/python3.10/dist-packages (from catboost) (1.26.4)
Requirement already satisfied: pandas>=0.24 in /usr/local/lib/python3.10/dist-packages (from catboost) (2.2.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from catboost) (1.13.1)
Requirement already satisfied: plotly in /usr/local/lib/python3.10/dist-packages (from catboost) (5.24.1)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from catboost) (1.16.0)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.24->catboost) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.24->catboost) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.24->catboost) (2024.2)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->catboost) (1.3.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib->catboost) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->catboost) (4.54.1)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->catboost) (1.4.7)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->catboost) (24.1)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->catboost) (10.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->catboost) (3.2.0)
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from plotly->catboost) (9.0.0)
Downloading catboost-1.2.7-cp310-cp310-manylinux2014_x86_64.whl (98.7 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 98.7/98.7 MB 8.6 MB/s eta 0:00:00
Installing collected packages: catboost
Successfully installed catboost-1.2.7
```

Attempting to create SHAP values

```
import pandas as pd
import numpy as np
import re
from sklearn.model_selection import train_test_split
import xgboost as xgb
import lightgbm as lgb
import catboost as cb
import shap
import matplotlib.pyplot as plt
from matplotlib import cm
from sklearn.metrics import r2_score

# Load the data
df = pd.read_csv('Origo_Database.csv')

# Function to convert time to total minutes
```

```
def convert_to_minutes(time_str):
    hours = 0
    minutes = 0
    hour_match = re.search(r'(\d+)\s*hr', time_str)
    if hour_match:
        hours = int(hour_match.group(1))
    minute_match = re.search(r'(\d+)\s*min', time_str)
    if minute_match:
        minutes = int(minute_match.group(1))
    return hours * 60 + minutes

# Apply the function to the Time column
df['time_minutes'] = df['Time'].apply(convert_to_minutes)

# Clean up the Difficulty values
df['Difficulty'] = df['Difficulty'].str.strip().str.lower()

# Map the Difficulty column to the new numeric column
difficulty_mapping = {
    'easy': 1,
    'moderate': 2,
    'intermediate': 3,
    'hard': 4,
    'complex': 5
}
df['Difficulty_Numeric'] = df['Difficulty'].map(difficulty_mapping)

# Fill NA values in Difficulty_Numeric if any exist
df['Difficulty_Numeric'] = df['Difficulty_Numeric'].fillna(1)

# Prepare data for regression
X = df[['time_minutes']]
y = df['Difficulty_Numeric']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Function to create SHAP plots for a given model
def plot_shap_values(model, X_test, model_name):
    explainer = shap.Explainer(model)
    shap_values = explainer(X_test)

    # Extract SHAP values and corresponding time values
    shap_values_array = shap_values.values
    time_values = X_test['time_minutes'].values

    # Normalize time values for color mapping
    norm = plt.Normalize(time_values.min(), time_values.max())
    colors = cm.viridis(norm(time_values))

    # Create a scatter plot for SHAP values
    plt.figure(figsize=(10, 6))
```

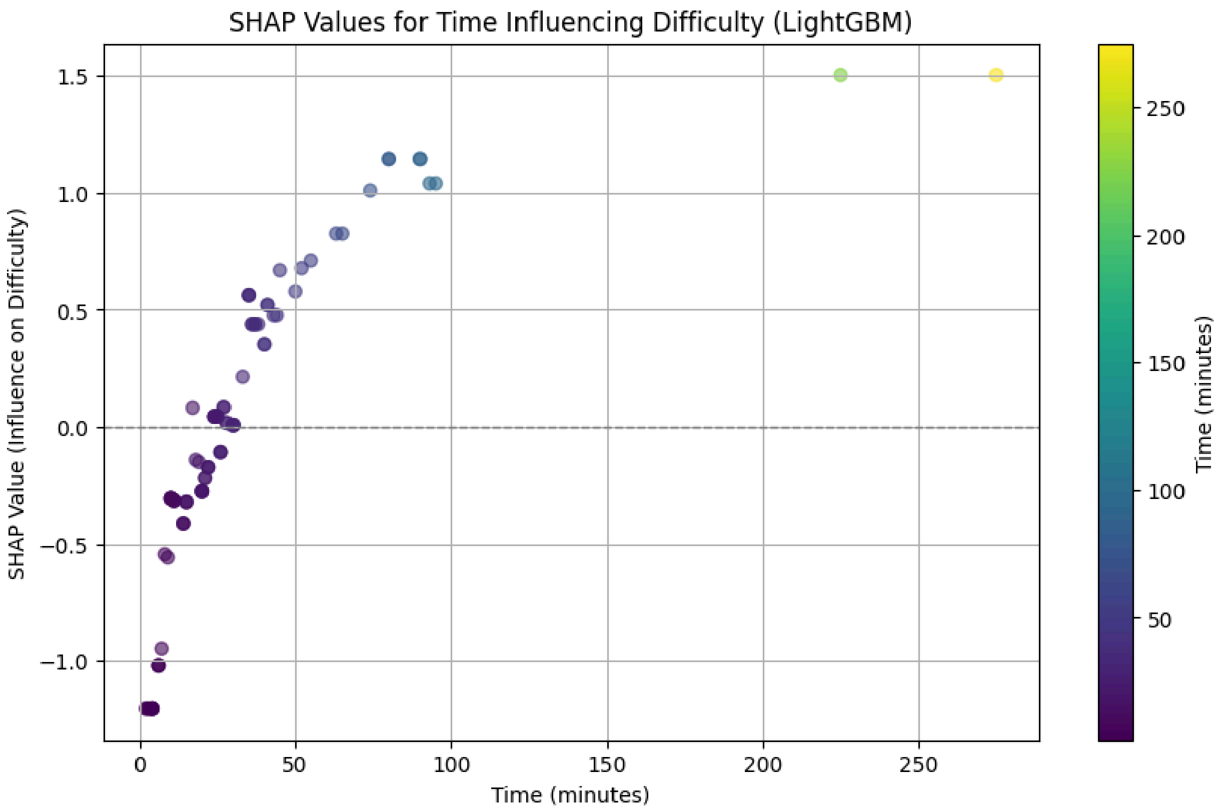
```
        scatter = plt.scatter(time_values, shap_values_array, c=colors, alpha=0.6)
        plt.title(f'SHAP Values for Time Influencing Difficulty ({model_name})')
        plt.xlabel('Time (minutes)')
        plt.ylabel('SHAP Value (Influence on Difficulty)')
        plt.axhline(0, color='gray', linestyle='--', linewidth=1)
        plt.grid(True)

        # Add a colorbar
        cbar = plt.colorbar(cm.ScalarMappable(norm=norm, cmap='viridis'), ax=plt.gca())
        cbar.set_label('Time (minutes)')
        plt.show()

    # Train the XGBoost model
    xgb_model = xgb.XGBRegressor(objective='reg:squarederror')
    xgb_model.fit(X_train, y_train)
    plot_shap_values(xgb_model, X_test, "XGBoost")

    # Train the LightGBM model
    lgb_model = lgb.LGBMRegressor()
    lgb_model.fit(X_train, y_train)
    plot_shap_values(lgb_model, X_test, "LightGBM")

    # Train the CatBoost model
    cat_model = cb.CatBoostRegressor(silent=True)
    cat_model.fit(X_train, y_train)
    plot_shap_values(cat_model, X_test, "CatBoost")
```



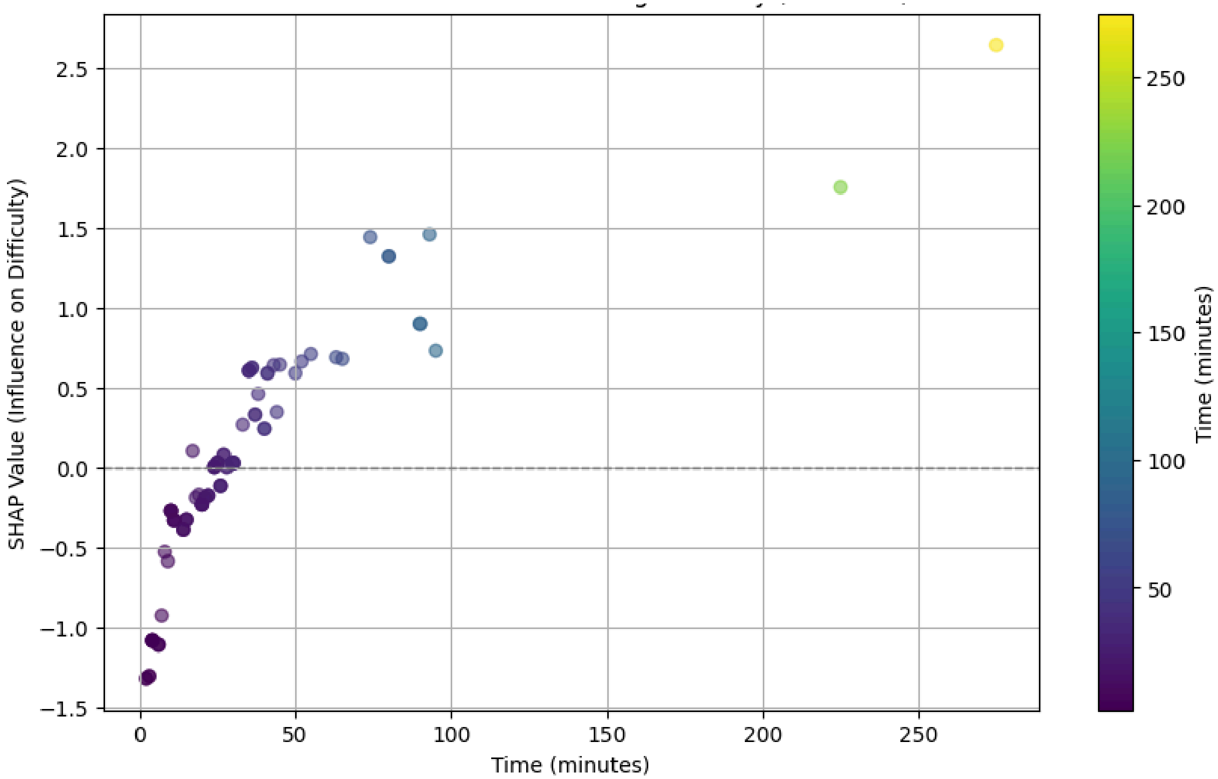

```
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.000069 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 61
[LightGBM] [Info] Number of data points in the train set: 360, number of used features: 1
[LightGBM] [Info] Start training from score 2.316667
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```




SHAP Values for Time Influencing Difficulty (CatBoost)


SHAP Value (Influence on Difficulty)
Time (minutes)
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
0
50
100
150
200
250
Time (minutes)
250
200
150
100
50

SHAP VALUES FOR EACH BOOSING METHOD. WOULD BE MORE USEFUL I F I HAD OTHER VARIABLES TO ASSESS IN RELATION TO CONTRIBUTING TO THE DIFFICULTY SCORE

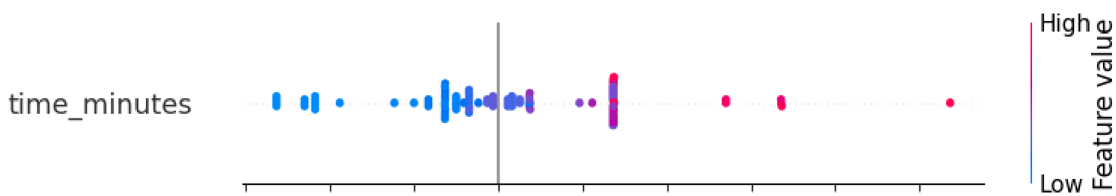
```
import pandas as pd
import numpy as np
import re
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import xgboost as xgb
import lightgbm as lgb
import catboost as cb
import shap
import plotly.express as px
from sklearn.metrics import r2_score

# Load the data
df = pd.read_csv('Origo_Database.csv')

# Function to convert time to total minutes
def convert_to_minutes(time_str):
    hours = 0
    minutes = 0
    hour_match = re.search(r'(\d+)\s*hr', time_str)
    if hour_match:
        hours = int(hour_match.group(1))
    minute_match = re.search(r'(\d+)\s*min', time_str)
    if minute_match:
        minutes = int(minute_match.group(1))
    return hours * 60 + minutes

# Apply the function to the Time column
df['time_minutes'] = df['Time'].apply(convert_to_minutes)

# Clean up the Difficulty values
df['Difficulty'] = df['Difficulty'].str.strip().str.lower()

# Map the Difficulty column to the new numeric column
difficulty_mapping = {
    'easy': 1,
    'moderate': 2,
    'intermediate': 3,
    'hard': 4,
    'complex': 5
}
df['Difficulty_Numeric'] = df['Difficulty'].map(difficulty_mapping)

# Fill NA values in Difficulty_Numeric if any exist
df['Difficulty_Numeric'] = df['Difficulty_Numeric'].fillna(1)

# Prepare data for regression
```

```
X = df[['time_minutes']]
y = df['Difficulty_Numeric']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train models
ols_model = LinearRegression()
ols_model.fit(X_train, y_train)

xgb_model = xgb.XGBRegressor(objective='reg:squarederror')
xgb_model.fit(X_train, y_train)

lgb_model = lgb.LGBMRegressor()
lgb_model.fit(X_train, y_train)

cat_model = cb.CatBoostRegressor(silent=True)
cat_model.fit(X_train, y_train)

# Predictions for SHAP values
xgb_pred = xgb_model.predict(X_test)
lgb_pred = lgb_model.predict(X_test)
cat_pred = cat_model.predict(X_test)

# Create a function to plot using Plotly
def plot_difficulty_vs_time(df, model_name, y_pred):
    df_plot = df.copy()
    df_plot['Predicted_Difficulty'] = y_pred
    fig = px.scatter(df_plot,
                     x='time_minutes',
                     y='Difficulty_Numeric',
                     title=f'Time vs Difficulty ({model_name})',
                     color_discrete_sequence=['blue'],
                     hover_data={'time_minutes': True, 'Difficulty_Numeric': True},
                     labels={'time_minutes': 'Time (minutes)', 'Difficulty_Numeric': 'Difficulty'})
    fig.add_scatter(x=df_plot['time_minutes'], y=df_plot['Predicted_Difficulty'], mode='lines', name='Predicted Difficulty', line=dict(color='red', width=2))
    fig.show()

# SHAP Values for XGBoost
explainer_xgb = shap.Explainer(xgb_model)
shap_values_xgb = explainer_xgb(X_test)

# SHAP Summary Plot for XGBoost with title
shap.summary_plot(shap_values_xgb, X_test, feature_names=['time_minutes'], title='SHAP Values for XGBoost')

# SHAP Values for LightGBM
explainer_lgb = shap.Explainer(lgb_model)
shap_values_lgb = explainer_lgb(X_test)

# SHAP Summary Plot for LightGBM with title
shap.summary_plot(shap_values_lgb, X_test, feature_names=['time_minutes'], title='SHAP Values for LightGBM')
```

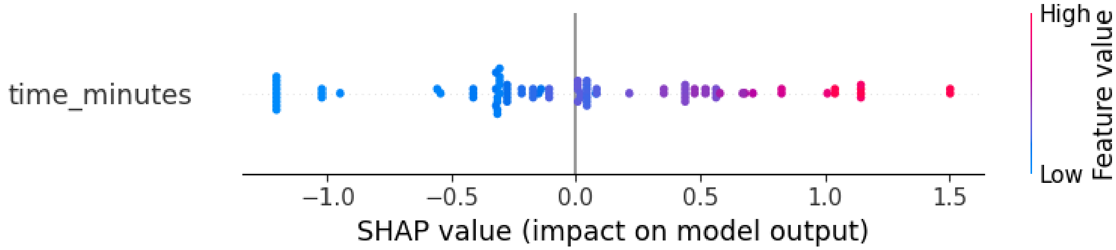
```
# SHAP Values for CatBoost
explainer_cat = shap.Explainer(cat_model)
shap_values_cat = explainer_cat(X_test)

# SHAP Summary Plot for CatBoost with title
shap.summary_plot(shap_values_cat, X_test, feature_names=['time_minutes'], title='SHAP Values for CatBoost')
```

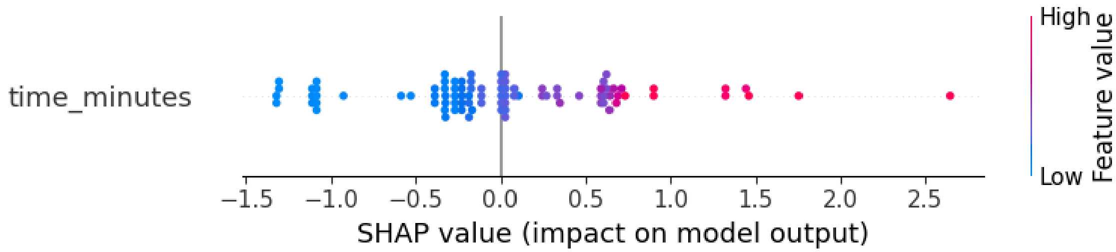
```
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.000040 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 61
[LightGBM] [Info] Number of data points in the train set: 360, number of used features: 1
[LightGBM] [Info] Start training from score 2.316667
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```




SHAP value (impact on model output)


time_minutes
−1.0
−0.5
0.0
0.5
1.0
1.5
SHAP value (impact on model output)
High
Low
Feature value

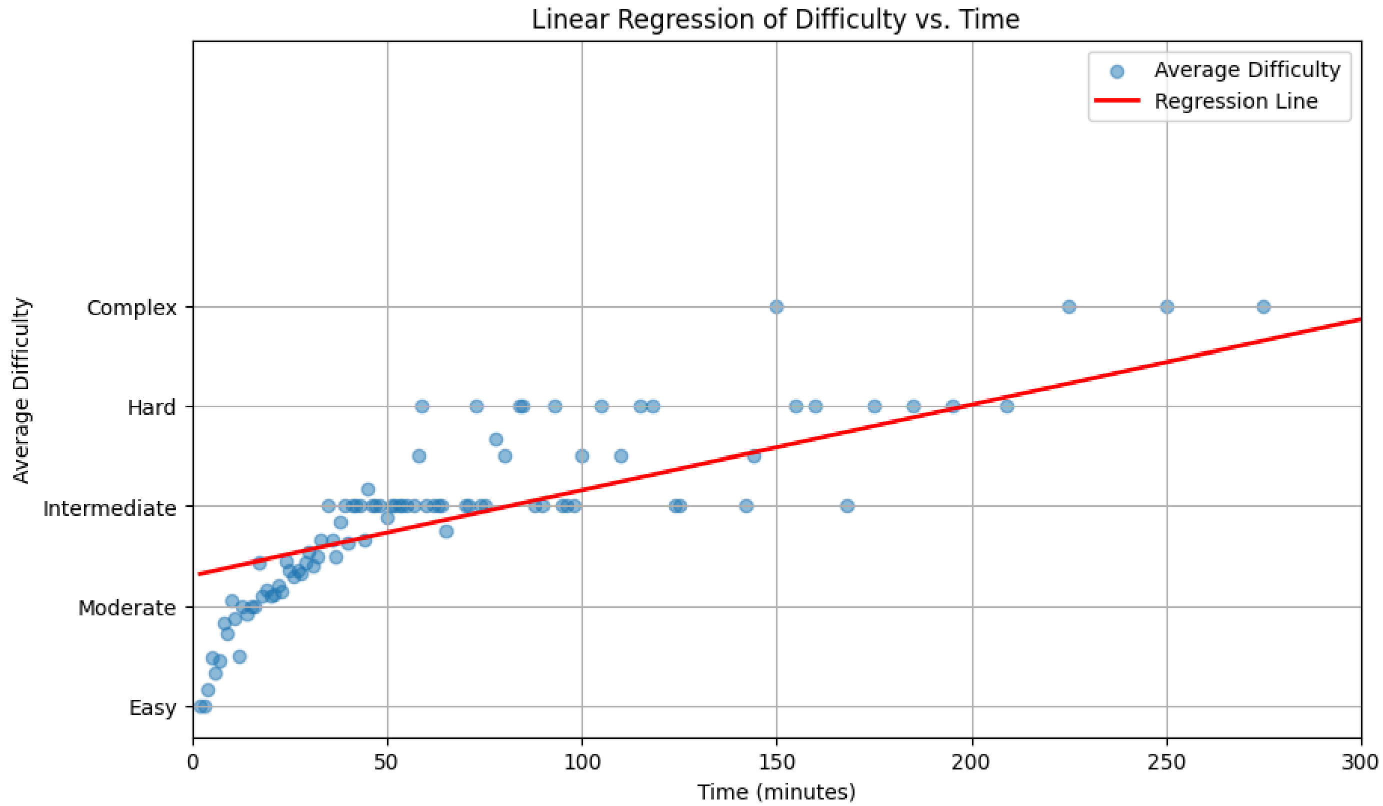

time_minutes
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
SHAP value (impact on model output)
High
Low
Feature value

lINEAR REGRESSION USING AVERAGES OF DIFFICULTY (AVERAGING DIFFICULTY WITH SAME SCORES)

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import re
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score  # Import r2_score
import numpy as np

# Load the data
df = pd.read_csv('Origo_Database.csv')

# Function to convert time to total minutes
def convert_to_minutes(time_str):
    hours = 0
    minutes = 0
    # Extract hours and minutes using regex
    hour_match = re.search(r'(\d+)\s*hr', time_str)
    if hour_match:
        hours = int(hour_match.group(1))
    minute_match = re.search(r'(\d+)\s*min', time_str)
    if minute_match:
        minutes = int(minute_match.group(1))
    return hours * 60 + minutes

# Apply the function to the Time column
df['time_minutes'] = df['Time'].apply(convert_to_minutes)

# Clean up the Difficulty values
df['Difficulty'] = df['Difficulty'].str.strip().str.lower()  # Remove extra spaces and convert to lower case

# Create the mapping
difficulty_mapping = {
    'easy': 1,
    'moderate': 2,
    'intermediate': 3,
    'hard': 4,
    'complex': 5
}

# Map the Difficulty column to the new numeric column
df['Difficulty_Numeric'] = df['Difficulty'].map(difficulty_mapping)

# Fill NA values in Difficulty_Numeric if any exist
df['Difficulty_Numeric'] = df['Difficulty_Numeric'].fillna(1)  # Fill NAs with 1 (or another appropriate value)

# Calculate the average difficulty for each unique time
avg_difficulty = df.groupby('time_minutes')['Difficulty_Numeric'].mean().reset_index()
```

```
# Fit a linear regression model
X = avg_difficulty['time_minutes'].values.reshape(-1, 1)  # Reshape for sklearn
y = avg_difficulty['Difficulty_Numeric'].values
model = LinearRegression()
model.fit(X, y)

# Create predictions for the regression line
y_pred = model.predict(X)

# Calculate R-squared value
r_squared = r2_score(y, y_pred)
print(f"R-squared: {r_squared:.3f}")

# Plotting the scatter and regression line
plt.figure(figsize=(10, 6))
plt.scatter(avg_difficulty['time_minutes'], avg_difficulty['Difficulty_Numeric'], alpha=0.5, label='Average Difficulty')
plt.plot(avg_difficulty['time_minutes'], y_pred, color='red', linewidth=2, label='Regression Line')
plt.title('Linear Regression of Difficulty vs. Time')
plt.xlabel('Time (minutes)')
plt.ylabel('Average Difficulty')
plt.yticks([1, 2, 3, 4, 5], ['Easy', 'Moderate', 'Intermediate', 'Hard', 'Complex'])  # Set y-ticks to the specified order
plt.xlim(0, 300)  # Set x-axis limits from 0 to 300
plt.grid(True)
plt.legend()
plt.show()

# Display unique values for reference
print("Unique time_minutes values:", df['time_minutes'].unique())
print("Unique Difficulty_Numeric values:", df['Difficulty_Numeric'].unique())
```

```
R-squared: 0.582
```




```
Unique time_minutes values: [  3 142 175  48  30 110   4  52   2 225   6   8  20 250  14  78 118  22
  38  37  71 160   9  13   7  11  51  27 275  65  44  18  40  63  24  55
 144  29  25  41  50  35  54  57 590  33  98  88  28 209  74  90  45  93
  26  84  43  17  62  10  16  19  21 125  47  42 168 100  95  60  80  53
 124  15   5  23 185  64 150  46  12  32  36  31  39 155 105  75 195  59
  58  96  70  85 115  73]
Unique Difficulty_Numeric values: [1 3 4 5 2]
```

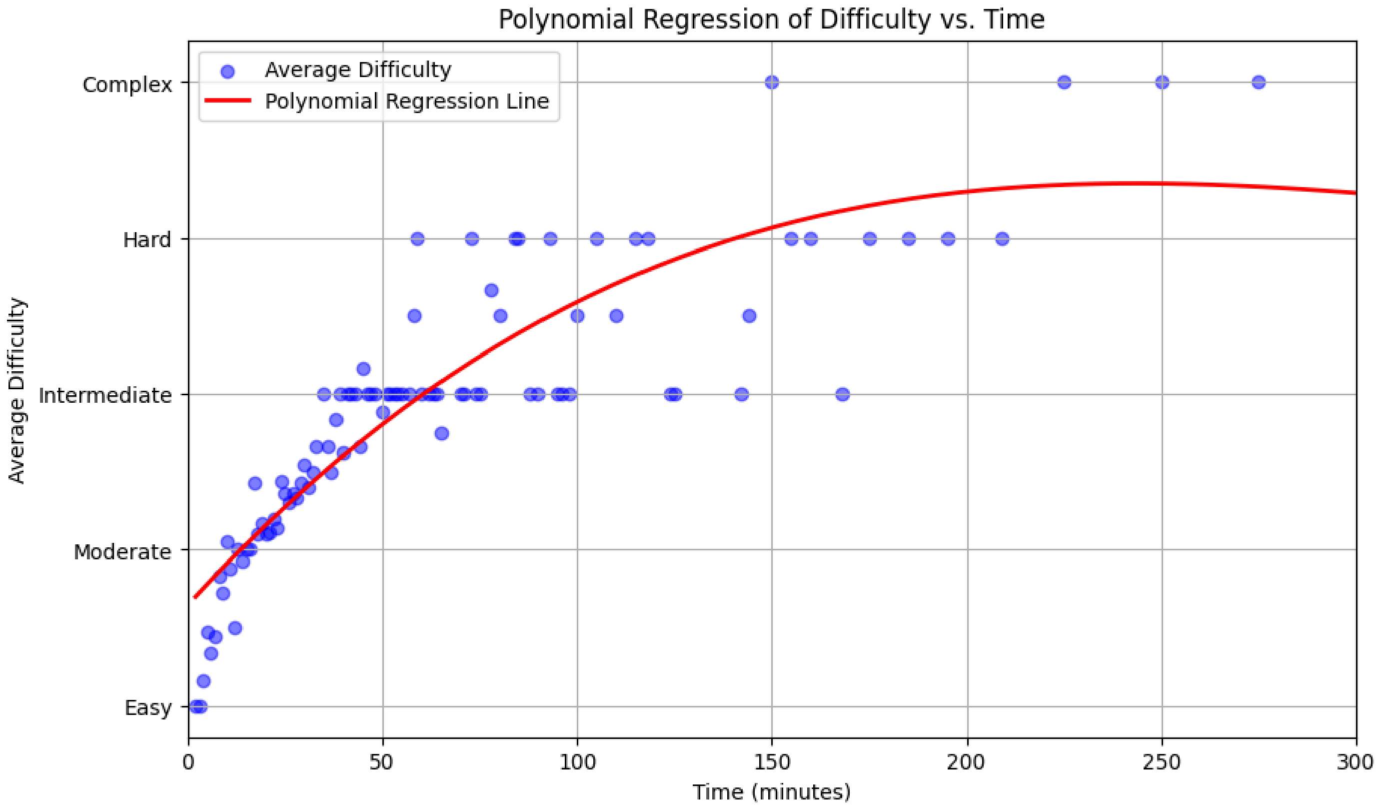
POLYNOMIAL REGRESSION

```
import pandas as pd
import matplotlib.pyplot as plt
import re
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import r2_score
import numpy as np

# Load the data
df = pd.read_csv('Origo_Database.csv')
```

```
# Function to convert time to total minutes
def convert_to_minutes(time_str):
    hours = 0
    minutes = 0
    hour_match = re.search(r'(\d+)\s*hr', time_str)
    if hour_match:
        hours = int(hour_match.group(1))
    minute_match = re.search(r'(\d+)\s*min', time_str)
    if minute_match:
        minutes = int(minute_match.group(1))
    return hours * 60 + minutes

# Apply the function to the Time column
df['time_minutes'] = df['Time'].apply(convert_to_minutes)

# Clean up the Difficulty values
df['Difficulty'] = df['Difficulty'].str.strip().str.lower()

# Create the mapping
difficulty_mapping = {
    'easy': 1,
    'moderate': 2,
    'intermediate': 3,
    'hard': 4,
    'complex': 5
}

# Map the Difficulty column to the new numeric column
df['Difficulty_Numeric'] = df['Difficulty'].map(difficulty_mapping)

# Fill NA values in Difficulty_Numeric if any exist
df['Difficulty_Numeric'] = df['Difficulty_Numeric'].fillna(1)

# Calculate the average difficulty for each unique time
avg_difficulty = df.groupby('time_minutes')['Difficulty_Numeric'].mean().reset_index()

# Fit a polynomial regression model
X = avg_difficulty['time_minutes'].values.reshape(-1, 1)
y = avg_difficulty['Difficulty_Numeric'].values

# Transform the features to polynomial features
poly = PolynomialFeatures(degree=3)  # Change degree as needed for better fit
X_poly = poly.fit_transform(X)

# Fit the linear regression model on the polynomial features
model = LinearRegression()
model.fit(X_poly, y)

# Create predictions for the regression curve
y_pred = model.predict(X_poly)
```

```
    # Calculate R-squared value
    r_squared = r2_score(y, y_pred)
    print(f"R-squared: {r_squared:.3f}")

    # Create a smooth line for the polynomial regression
    X_range = np.linspace(X.min(), X.max(), 100).reshape(-1, 1)  # Create range for x-axis
    X_range_poly = poly.transform(X_range)  # Transform to polynomial features
    y_range_pred = model.predict(X_range_poly)  # Get predictions for the range

    # Plotting the original data points and polynomial regression curve
    plt.figure(figsize=(10, 6))
    plt.scatter(avg_difficulty['time_minutes'], avg_difficulty['Difficulty_Numeric'], alpha=0.5, label='Average Difficulty', color='blue')
    plt.plot(X_range, y_range_pred, color='red', linewidth=2, label='Polynomial Regression Line')
    plt.title('Polynomial Regression of Difficulty vs. Time')
    plt.xlabel('Time (minutes)')
    plt.ylabel('Average Difficulty')
    plt.yticks([1, 2, 3, 4, 5], ['Easy', 'Moderate', 'Intermediate', 'Hard', 'Complex'])
    plt.xlim(0, 300)
    plt.grid(True)
    plt.legend()
    plt.show()

    # Display unique values for reference
    print("Unique time_minutes values:", df['time_minutes'].unique())
    print("Unique Difficulty_Numeric values:", df['Difficulty_Numeric'].unique())
```

```
R-squared: 0.793
```



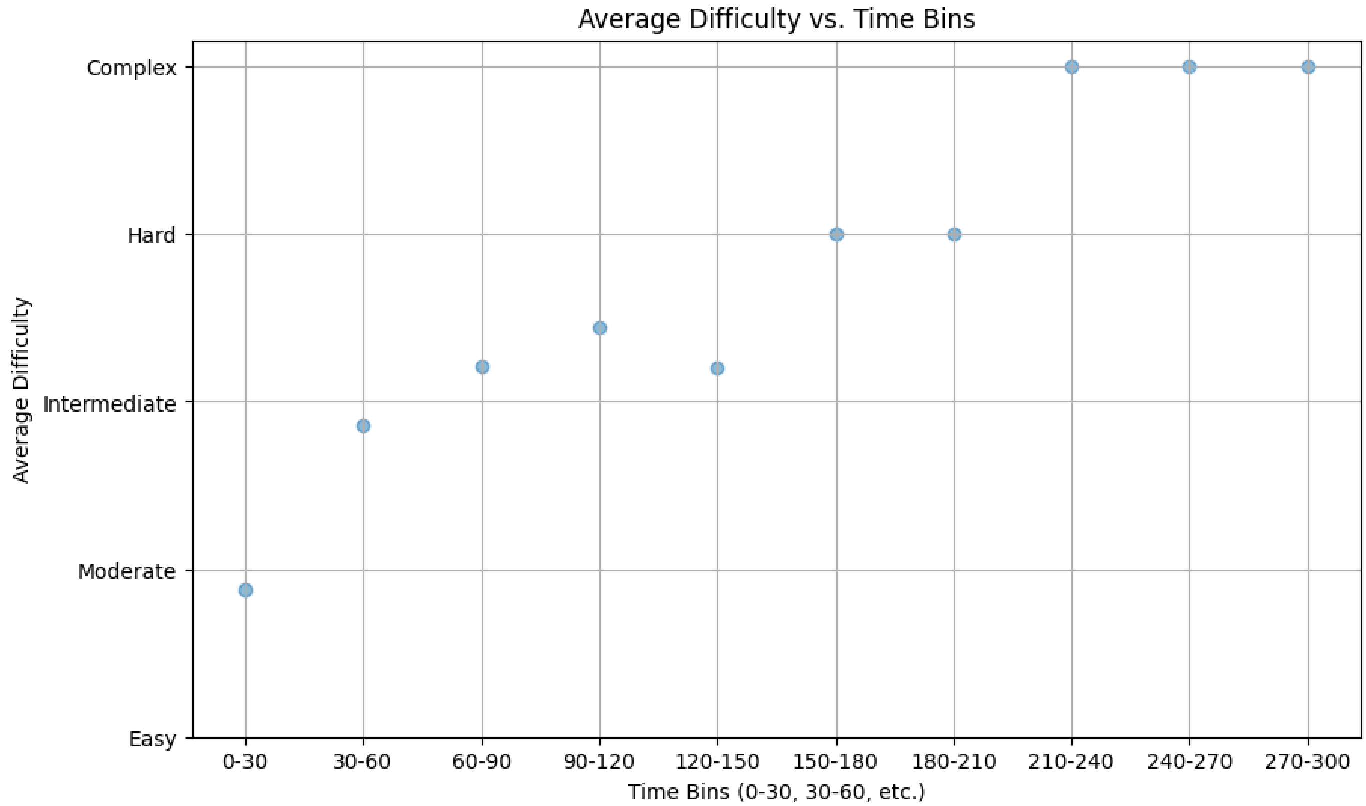

```
Unique time_minutes values: [  3 142 175  48  30 110   4  52   2 225   6   8  20 250  14  78 118  22
  38  37  71 160   9  13   7  11  51  27 275  65  44  18  40  63  24  55
 144  29  25  41  50  35  54  57 590  33  98  88  28 209  74  90  45  93
  26  84  43  17  62  10  16  19  21 125  47  42 168 100  95  60  80  53
 124  15   5  23 185  64 150  46  12  32  36  31  39 155 105  75 195  59
  58  96  70  85 115  73]
Unique Difficulty_Numeric values: [1 3 4 5 2]
```

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import re

# Load the data
df = pd.read_csv('Origo_Database.csv')

# Function to convert time to total minutes
def convert_to_minutes(time_str):
    hours = 0
    minutes = 0
    # Extract hours and minutes using regex
```

```
        hour_match = re.search(r'(\d+)\s*hr', time_str)
        if hour_match:
            hours = int(hour_match.group(1))
        minute_match = re.search(r'(\d+)\s*min', time_str)
        if minute_match:
            minutes = int(minute_match.group(1))
        return hours * 60 + minutes

    # Apply the function to the Time column
    df['time_minutes'] = df['Time'].apply(convert_to_minutes)

    # Clean up the Difficulty values
    df['Difficulty'] = df['Difficulty'].str.strip().str.lower()

    # Create the mapping
    difficulty_mapping = {
        'easy': 1,
        'moderate': 2,
        'intermediate': 3,
        'hard': 4,
        'complex': 5
    }

    # Map the Difficulty column to the new numeric column
    df['Difficulty_Numeric'] = df['Difficulty'].map(difficulty_mapping)

    # Fill NA values in Difficulty_Numeric if any exist
    df['Difficulty_Numeric'] = df['Difficulty_Numeric'].fillna(1)

    # Binning the time into intervals
    bins = np.arange(0, 301, 30)  # Create bins from 0 to 300 minutes, in 30-minute intervals
    labels = range(len(bins) - 1)  # Create labels for the bins

    # Cut the time into bins
    df['time_bins'] = pd.cut(df['time_minutes'], bins=bins, labels=labels, right=False)

    # Calculate the average difficulty for each time bin
    avg_difficulty = df.groupby('time_bins')['Difficulty_Numeric'].mean().reset_index()

    # Plotting the results
    plt.figure(figsize=(10, 6))
    plt.scatter(avg_difficulty['time_bins'], avg_difficulty['Difficulty_Numeric'], alpha=0.5)
    plt.title('Average Difficulty vs. Time Bins')
    plt.xlabel('Time Bins (0-30, 30-60, etc.)')
    plt.ylabel('Average Difficulty')
    plt.yticks([1, 2, 3, 4, 5], ['Easy', 'Moderate', 'Intermediate', 'Hard', 'Complex'])  # Set y-ticks to the specified order
    plt.xticks(avg_difficulty['time_bins'], [f"{bins[i]}-{bins[i + 1]}" for i in range(len(bins) - 1)])  # Set x-ticks to show time ranges
    plt.grid(True)
    plt.show()

    # Display unique values for reference
```

```
print("Unique time_minutes values:", df['time_minutes'].unique())
print("Unique Difficulty_Numeric values:", df['Difficulty_Numeric'].unique())
```

```
<ipython-input-53-6c51c2a4483a>:51: FutureWarning: The default of observed=False is deprecated and will be changed to True in a future version of pandas. Pass ob
  avg_difficulty = df.groupby('time_bins')['Difficulty_Numeric'].mean().reset_index()
```



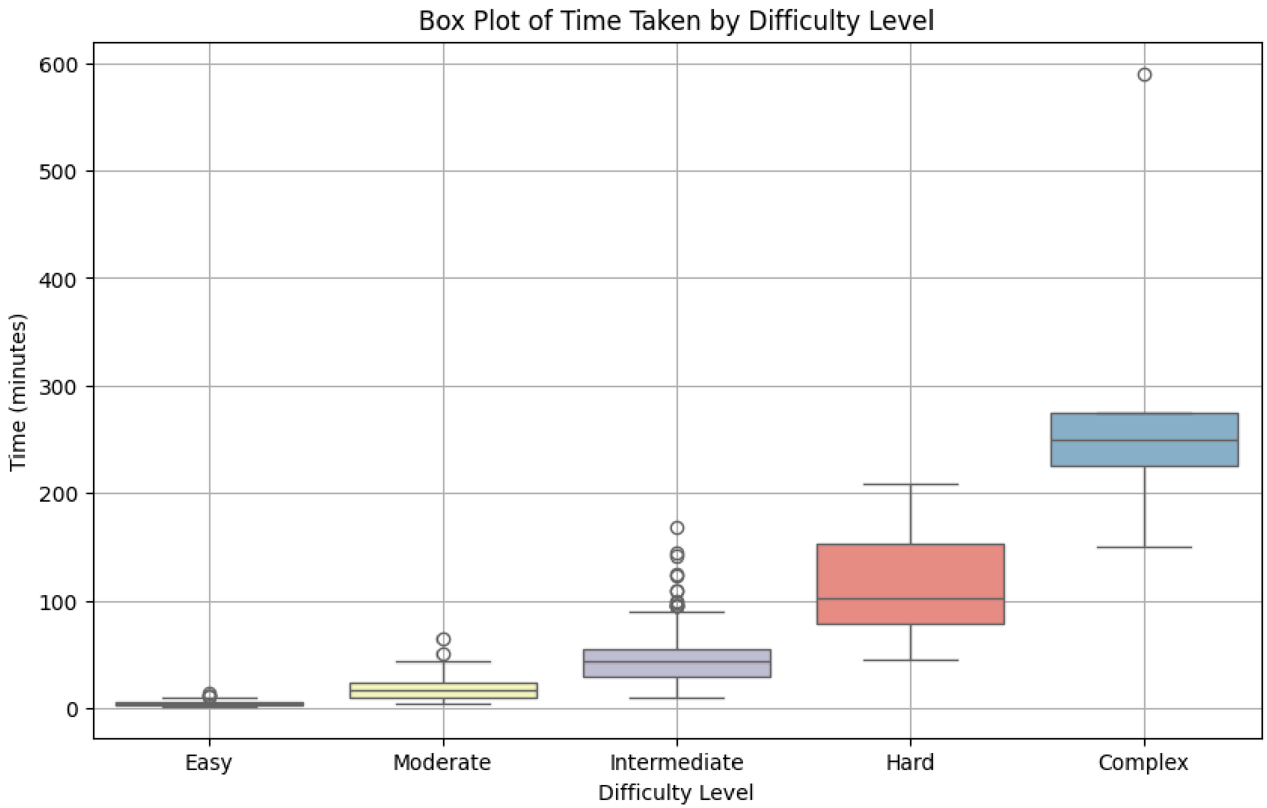

```
Unique time_minutes values: [  3 142 175  48  30 110   4  52   2 225   6   8  20 250  14  78 118  22
  38  37  71 160   9  13   7  11  51  27 275  65  44  18  40  63  24  55
 144  29  25  41  50  35  54  57 590  33  98  88  28 209  74  90  45  93
  26  84  43  17  62  10  16  19  21 125  47  42 168 100  95  60  80  53
 124  15   5  23 185  64 150  46  12  32  36  31  39 155 105  75 195  59
  58  96  70  85 115  73]
Unique Difficulty_Numeric values: [1 3 4 5 2]
```

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import re

# Load the data
df = pd.read_csv('Origo_Database.csv')
```

```
# Function to convert time to total minutes
def convert_to_minutes(time_str):
    hours = 0
    minutes = 0
    # Extract hours and minutes using regex
    hour_match = re.search(r'(\d+)\s*hr', time_str)
    if hour_match:
        hours = int(hour_match.group(1))
    minute_match = re.search(r'(\d+)\s*min', time_str)
    if minute_match:
        minutes = int(minute_match.group(1))
    return hours * 60 + minutes

# Apply the function to the Time column
df['time_minutes'] = df['Time'].apply(convert_to_minutes)

# Clean up the Difficulty values
df['Difficulty'] = df['Difficulty'].str.strip().str.lower()  # Remove extra spaces and convert to lower case

# Create the mapping
difficulty_mapping = {
    'easy': 1,
    'moderate': 2,
    'intermediate': 3,
    'hard': 4,
    'complex': 5
}

# Map the Difficulty column to the new numeric column
df['Difficulty_Numeric'] = df['Difficulty'].map(difficulty_mapping)

# Fill NA values in Difficulty_Numeric if any exist
df['Difficulty_Numeric'] = df['Difficulty_Numeric'].fillna(1)  # Fill NAs with 1 (or another appropriate value)

# Ensure Difficulty_Numeric is treated as an ordered categorical
df['Difficulty_Numeric'] = pd.Categorical(df['Difficulty_Numeric'], categories=[1, 2, 3, 4, 5], ordered=True)

# Create a box plot for Difficulty vs Time
plt.figure(figsize=(10, 6))
sns.boxplot(x='Difficulty_Numeric', y='time_minutes', data=df, palette='Set3')

# Customize the plot
plt.title('Box Plot of Time Taken by Difficulty Level')
plt.xlabel('Difficulty Level')
plt.ylabel('Time (minutes)')
plt.xticks(ticks=[0, 1, 2, 3, 4], labels=['Easy', 'Moderate', 'Intermediate', 'Hard', 'Complex'])  # Set x-ticks
plt.grid(True)
plt.show()

# Display unique values for reference
```

```
print("Unique time_minutes values:", df['time_minutes'].unique())
print("Unique Difficulty_Numeric values:", df['Difficulty_Numeric'].unique())
```

```
<ipython-input-37-81218bbd1bb3>:48: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effe

  sns.boxplot(x='Difficulty_Numeric', y='time_minutes', data=df, palette='Set3')
```




```
Unique time_minutes values: [  3 142 175  48  30 110   4  52   2 225   6   8  20 250  14  78 118  22
  38  37  71 160   9  13   7  11  51  27 275  65  44  18  40  63  24  55
 144  29  25  41  50  35  54  57 590  33  98  88  28 209  74  90  45  93
  26  84  43  17  62  10  16  19  21 125  47  42 168 100  95  60  80  53
 124  15   5  23 185  64 150  46  12  32  36  31  39 155 105  75 195  59
  58  96  70  85 115  73]
Unique Difficulty_Numeric values: [1, 3, 4, 5, 2]
Categories (5, int64): [1 < 2 < 3 < 4 < 5]
```

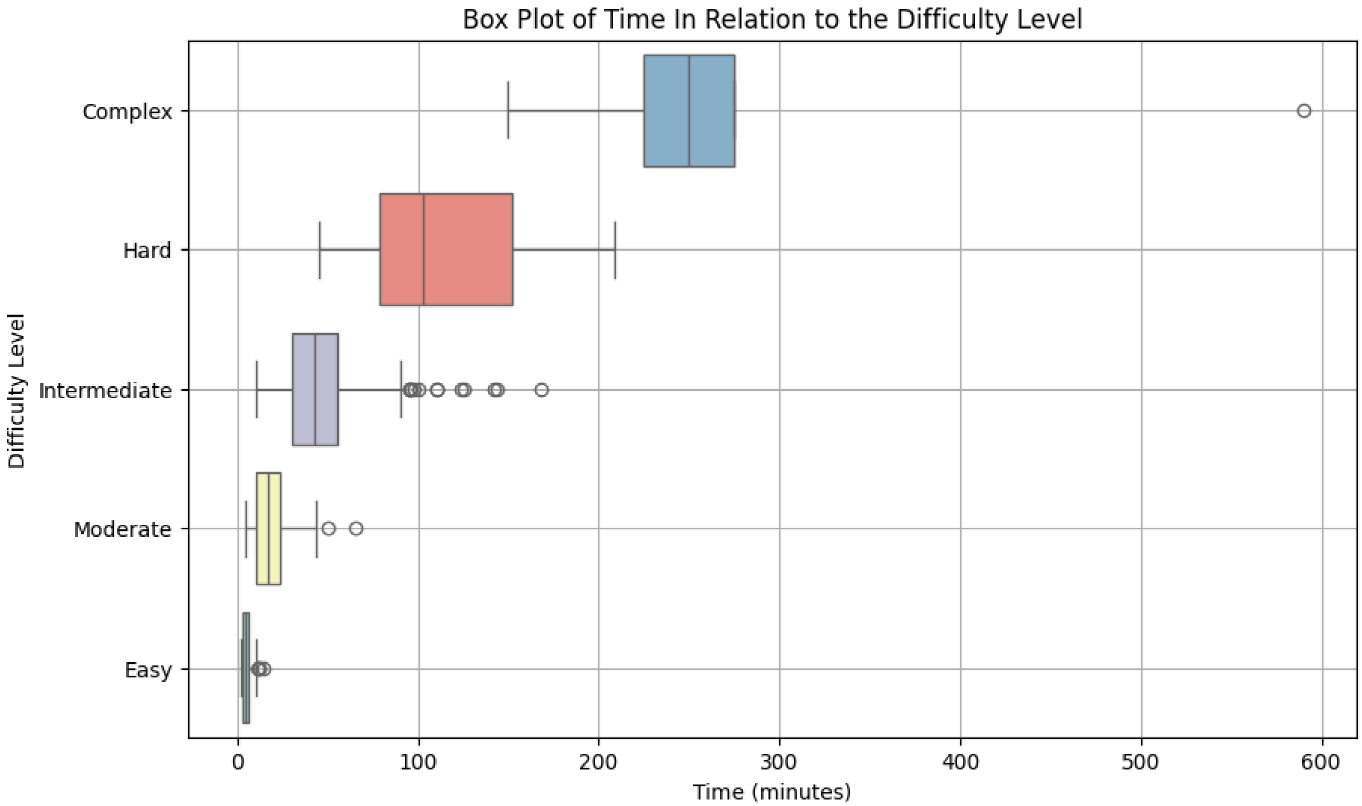
BOX PLOT DISTRIBUTION OF DATA

Start coding or generate with AI.

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import re

# Load the data
df = pd.read_csv('Origo_Database.csv')

# Function to convert time to total minutes
def convert_to_minutes(time_str):
    hours = 0
    minutes = 0
    # Extract hours and minutes using regex
    hour_match = re.search(r'(\d+)\s*hr', time_str)
    if hour_match:
        hours = int(hour_match.group(1))
    minute_match = re.search(r'(\d+)\s*min', time_str)
    if minute_match:
        minutes = int(minute_match.group(1))
    return hours * 60 + minutes

# Apply the function to the Time column
df['time_minutes'] = df['Time'].apply(convert_to_minutes)

# Clean up the Difficulty values
df['Difficulty'] = df['Difficulty'].str.strip().str.lower()  # Remove extra spaces and convert to lower case

# Create the mapping
difficulty_mapping = {
    'complex': 1,
    'hard': 2,
    'intermediate': 3,
    'moderate': 4,
    'easy': 5
}

# Map the Difficulty column to the new numeric column
df['Difficulty_Numeric'] = df['Difficulty'].map(difficulty_mapping)

# Fill NA values in Difficulty_Numeric if any exist
df['Difficulty_Numeric'] = df['Difficulty_Numeric'].fillna(1)  # Fill NAs with 1 (or another appropriate value)

# Ensure Difficulty_Numeric is treated as an ordered categorical
df['Difficulty_Numeric'] = pd.Categorical(df['Difficulty_Numeric'], categories=[5, 4, 3, 2, 1], ordered=True)

# Create a box plot for Time vs Difficulty
plt.figure(figsize=(10, 6))
sns.boxplot(y='Difficulty_Numeric', x='time_minutes', data=df, palette='Set3')

# Customize the plot
plt.title('Box Plot of Time In Relation to the Difficulty Level')
```

```
    plt.ylabel('Difficulty Level')
    plt.xlabel('Time (minutes)')
    plt.yticks(ticks=[0, 1, 2, 3, 4], labels=['Easy', 'Moderate', 'Intermediate', 'Hard', 'Complex'])  # Set y-ticks
    plt.grid(True)

    # Invert the y-axis
    plt.gca().invert_yaxis()

    plt.show()

    # Display unique values for reference
    print("Unique time_minutes values:", df['time_minutes'].unique())
    print("Unique Difficulty_Numeric values:", df['Difficulty_Numeric'].unique())
```

```
<ipython-input-55-3015d02b4df8>:48: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effe

  sns.boxplot(y='Difficulty_Numeric', x='time_minutes', data=df, palette='Set3')
```



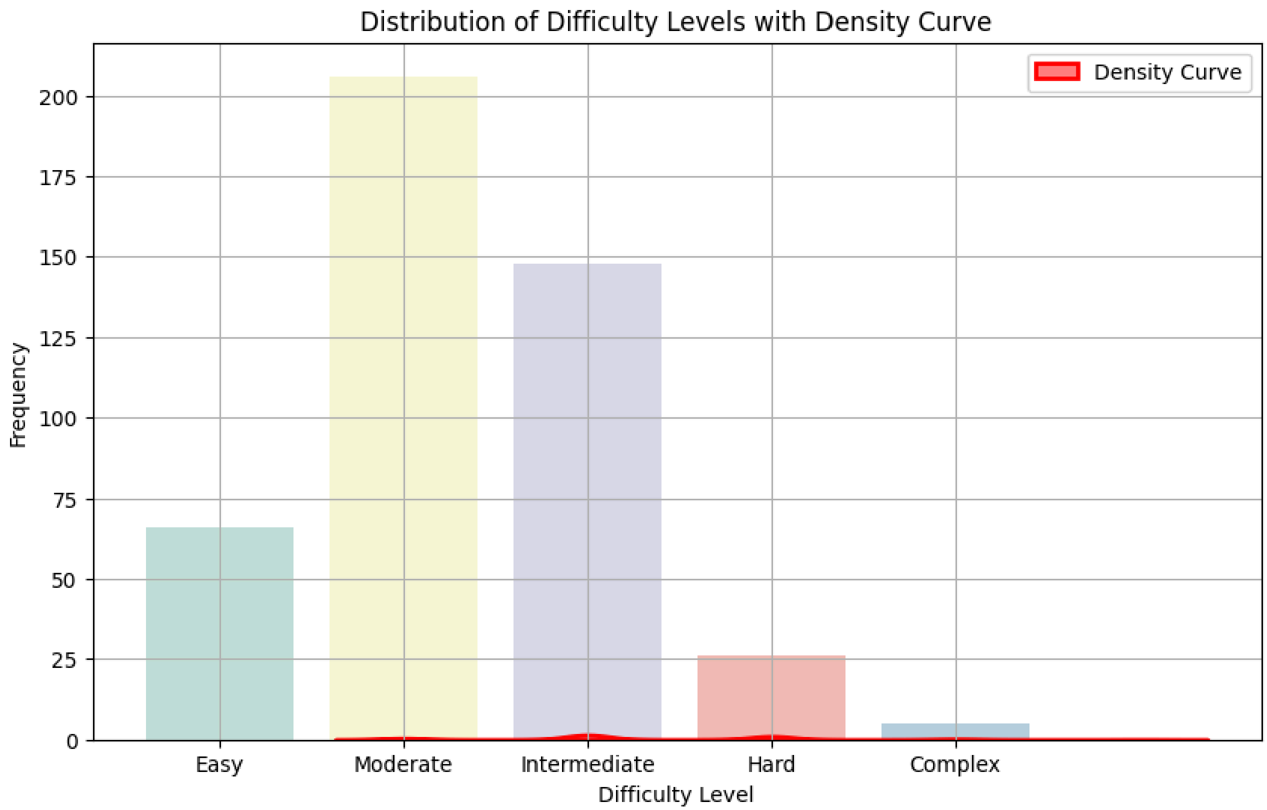

```
Unique time_minutes values: [  3 142 175  48  30 110   4  52   2 225   6   8  20 250  14  78 118  22
  38  37  71 160   9  13   7  11  51  27 275  65  44  18  40  63  24  55
 144  29  25  41  50  35  54  57 590  33  98  88  28 209  74  90  45  93
  26  84  43  17  62  10  16  19  21 125  47  42 168 100  95  60  80  53
 124  15   5  23 185  64 150  46  12  32  36  31  39 155 105  75 195  59
  58  96  70  85 115  73]
Unique Difficulty_Numeric values: [5, 3, 2, 1, 4]
Categories (5, int64): [5 < 4 < 3 < 2 < 1]
```

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import re

# Load the data
df = pd.read_csv('Origo_Database.csv')
```

```
# Function to convert time to total minutes
def convert_to_minutes(time_str):
    hours = 0
    minutes = 0
    # Extract hours and minutes using regex
    hour_match = re.search(r'(\d+)\s*hr', time_str)
    if hour_match:
        hours = int(hour_match.group(1))
    minute_match = re.search(r'(\d+)\s*min', time_str)
    if minute_match:
        minutes = int(minute_match.group(1))
    return hours * 60 + minutes

# Apply the function to the Time column
df['time_minutes'] = df['Time'].apply(convert_to_minutes)

# Clean up the Difficulty values
df['Difficulty'] = df['Difficulty'].str.strip().str.lower()  # Remove extra spaces and convert to lower case

# Create the mapping
difficulty_mapping = {
    'easy': 1,
    'moderate': 2,
    'intermediate': 3,
    'hard': 4,
    'complex': 5
}

# Map the Difficulty column to the new numeric column
df['Difficulty_Numeric'] = df['Difficulty'].map(difficulty_mapping)

# Fill NA values in Difficulty_Numeric if any exist
df['Difficulty_Numeric'] = df['Difficulty_Numeric'].fillna(1)  # Fill NAs with 1 (or another appropriate value)

# Set up the figure and axes
plt.figure(figsize=(10, 6))

# Create a count plot for the frequency of Difficulty levels
sns.countplot(x='Difficulty', data=df, palette='Set3', order=difficulty_mapping.keys(), alpha=0.6)

# Create a KDE plot overlay for the difficulty distribution
sns.kdeplot(df['Difficulty_Numeric'].dropna(),
             bw_adjust=0.5,
             fill=True,
             color='red',
             alpha=0.5,
             linewidth=2,
             label='Density Curve')

# Customize the plot
plt.title('Distribution of Difficulty Levels with Density Curve')
```

```
plt.xlabel('Difficulty Level')
plt.ylabel('Frequency')
plt.xticks(ticks=[0, 1, 2, 3, 4], labels=['Easy', 'Moderate', 'Intermediate', 'Hard', 'Complex'])  # Set x-ticks
plt.grid(True)
plt.legend()
plt.show()

# Display unique values for reference
print("Unique Difficulty values:", df['Difficulty'].unique())
print("Unique Difficulty_Numeric values:", df['Difficulty_Numeric'].unique())
```

```
<ipython-input-42-000460e70322>:47: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effe

  sns.countplot(x='Difficulty', data=df, palette='Set3', order=difficulty_mapping.keys(), alpha=0.6)
```



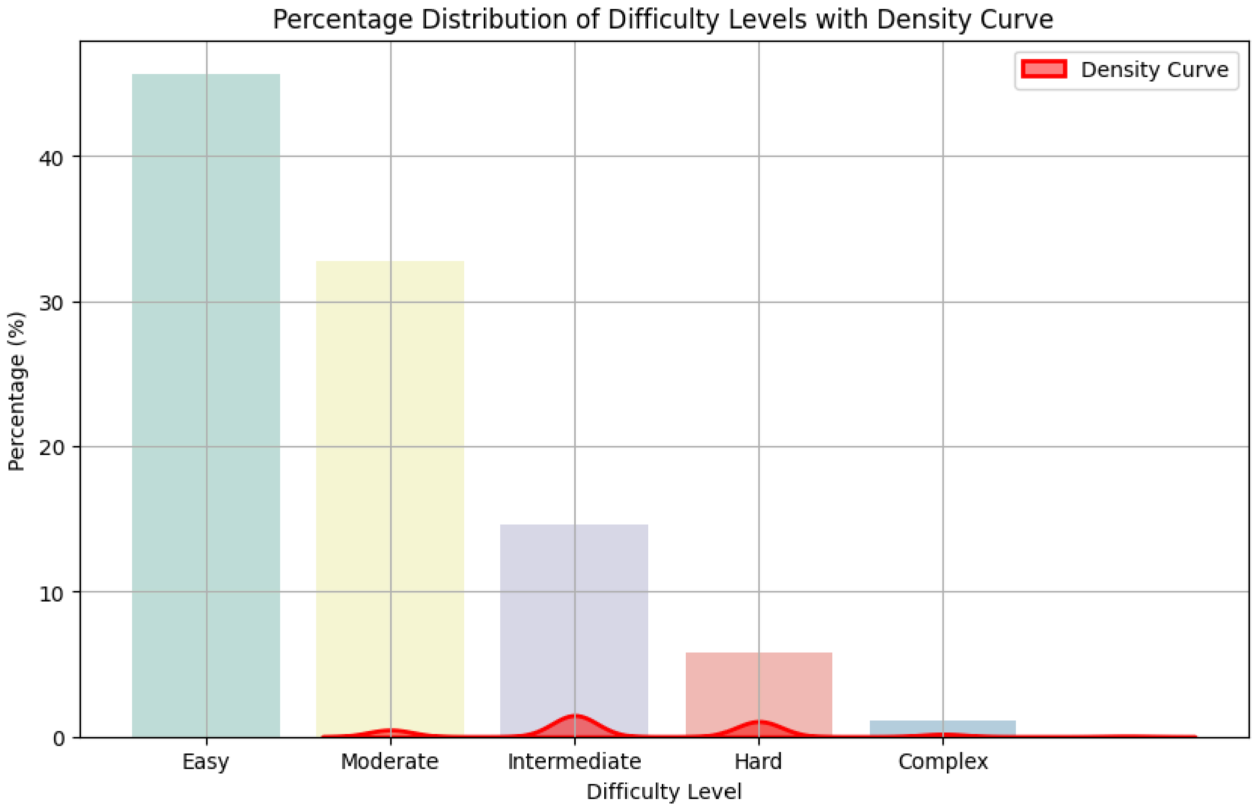

```
Unique Difficulty values: ['easy' 'intermediate' 'hard' 'complex' 'moderate']
```

```
import pandas as pd
import matplotlib.pyplot as plt
```

```
import seaborn as sns
import re

# Load the data
df = pd.read_csv('Origo_Database.csv')

# Function to convert time to total minutes
def convert_to_minutes(time_str):
    hours = 0
    minutes = 0
    # Extract hours and minutes using regex
    hour_match = re.search(r'(\d+)\s*hr', time_str)
    if hour_match:
        hours = int(hour_match.group(1))
    minute_match = re.search(r'(\d+)\s*min', time_str)
    if minute_match:
        minutes = int(minute_match.group(1))
    return hours * 60 + minutes

# Apply the function to the Time column
df['time_minutes'] = df['Time'].apply(convert_to_minutes)

# Clean up the Difficulty values
df['Difficulty'] = df['Difficulty'].str.strip().str.lower()  # Remove extra spaces and convert to lower case

# Create the mapping
difficulty_mapping = {
    'easy': 1,
    'moderate': 2,
    'intermediate': 3,
    'hard': 4,
    'complex': 5
}

# Map the Difficulty column to the new numeric column
df['Difficulty_Numeric'] = df['Difficulty'].map(difficulty_mapping)

# Fill NA values in Difficulty_Numeric if any exist
df['Difficulty_Numeric'] = df['Difficulty_Numeric'].fillna(1)  # Fill NAs with 1 (or another appropriate value)

# Calculate percentages for the difficulty levels
difficulty_counts = df['Difficulty'].value_counts(normalize=True) * 100

# Set up the figure and axes
plt.figure(figsize=(10, 6))

# Create a bar plot for the percentage of Difficulty levels
sns.barplot(x=difficulty_counts.index, y=difficulty_counts.values, palette='Set3', alpha=0.6)

# Create a KDE plot overlay for the difficulty distribution
sns.kdeplot(df['Difficulty_Numeric'].dropna(),
```

```
            bw_adjust=0.5,
            fill=True,
            color='red',
            alpha=0.5,
            linewidth=2,
            label='Density Curve')

# Customize the plot
plt.title('Percentage Distribution of Difficulty Levels with Density Curve')
plt.xlabel('Difficulty Level')
plt.ylabel('Percentage (%)')
plt.xticks(ticks=[0, 1, 2, 3, 4], labels=['Easy', 'Moderate', 'Intermediate', 'Hard', 'Complex'])  # Set x-ticks
plt.grid(True)
plt.legend()
plt.show()

# Display unique values for reference
print("Unique Difficulty values:", df['Difficulty'].unique())
print("Unique Difficulty_Numeric values:", df['Difficulty_Numeric'].unique())
```

```
<ipython-input-43-8dc188efb503>:50: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effe

  sns.barplot(x=difficulty_counts.index, y=difficulty_counts.values, palette='Set3', alpha=0.6)
```



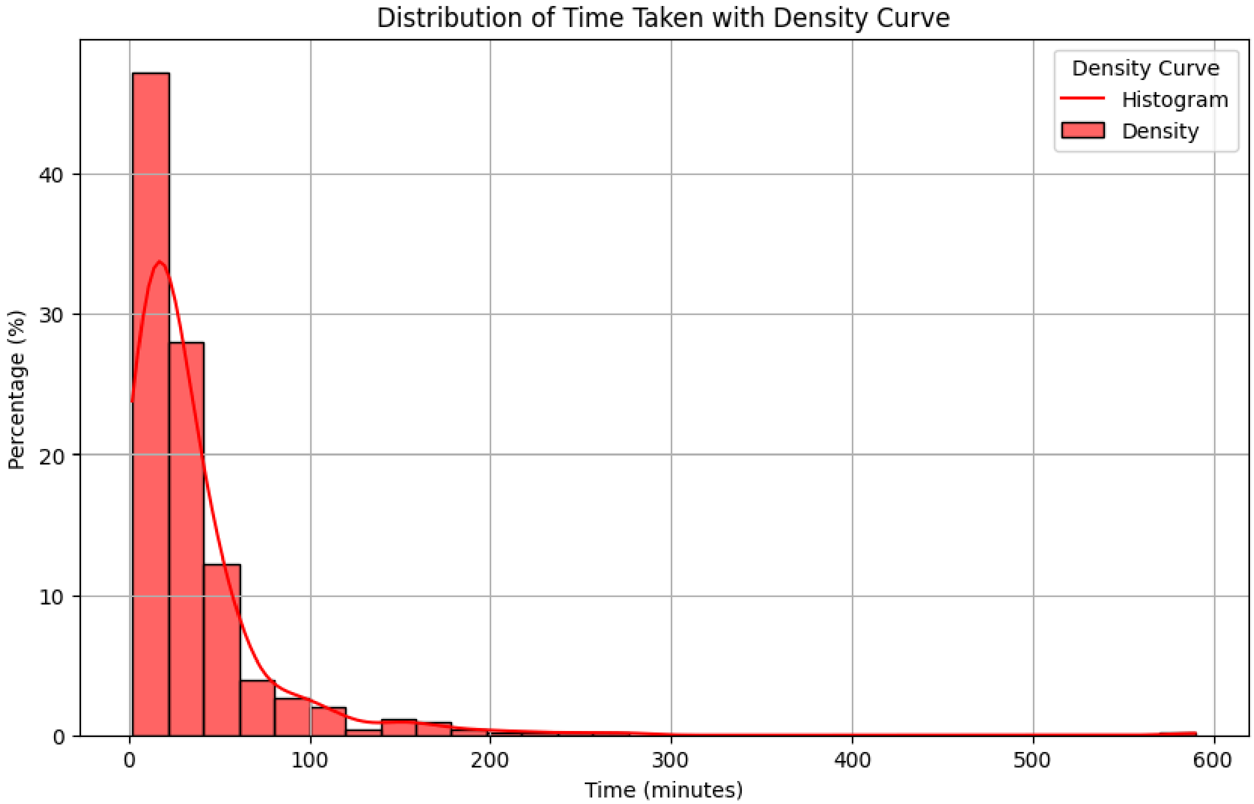

```
Unique Difficulty values: ['easy' 'intermediate' 'hard' 'complex' 'moderate']
Unique Difficulty Numeric values: [1 3 4 5 2]
```

DATA PRESENTS ITSELF AS A RIGHT TAILED DISTRIBUTION

DISTRIBUTION OF TIME AND ITS OCCURENCE IN DATA

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import re

# Load the data
df = pd.read_csv('Origo_Database.csv')
```

```
# Function to convert time to total minutes
def convert_to_minutes(time_str):
    hours = 0
    minutes = 0
    # Extract hours and minutes using regex
    hour_match = re.search(r'(\d+)\s*hr', time_str)
    if hour_match:
        hours = int(hour_match.group(1))
    minute_match = re.search(r'(\d+)\s*min', time_str)
    if minute_match:
        minutes = int(minute_match.group(1))
    return hours * 60 + minutes

# Apply the function to the Time column
df['time_minutes'] = df['Time'].apply(convert_to_minutes)

# Clean up the Difficulty values
df['Difficulty'] = df['Difficulty'].str.strip().str.lower()  # Remove extra spaces and convert to lower case

# Create the mapping
difficulty_mapping = {
    'easy': 1,
    'moderate': 2,
    'intermediate': 3,
    'hard': 4,
    'complex': 5
}

# Map the Difficulty column to the new numeric column
df['Difficulty_Numeric'] = df['Difficulty'].map(difficulty_mapping)

# Fill NA values in Difficulty_Numeric if any exist
df['Difficulty_Numeric'] = df['Difficulty_Numeric'].fillna(1)  # Fill NAs with 1 (or another appropriate value)

# Set up the figure and axes
plt.figure(figsize=(10, 6))

# Create a histogram with a KDE overlay
sns.histplot(df['time_minutes'], bins=30, kde=True, color='red', alpha=0.6, stat='percent')

# Customize the plot
plt.title('Distribution of Time Taken with Density Curve')
plt.xlabel('Time (minutes)')
plt.ylabel('Percentage (%)')
plt.grid(True)
plt.legend(title='Density Curve', labels=['Histogram', 'Density'])
plt.show()

# Display unique values for reference
print("Unique time_minutes values:", df['time_minutes'].unique())
```




```
Unique time_minutes values: [  3 142 175  48  30 110   4  52   2 225   6   8  20 250  14  78 118  22
  38  37  71 160   9  13   7  11  51  27 275  65  44  18  40  63  24  55
 144  29  25  41  50  35  54  57 590  33  98  88  28 209  74  90  45  93
  26  84  43  17  62  10  16  19  21 125  47  42 168 100  95  60  80  53
 124  15   5  23 185  64 150  46  12  32  36  31  39 155 105  75 195  59
  58  96  70  85 115  73]
```

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

---

Start coding or generate with AI.

CLASSIFICATION MODEL FOR NEXT STEPS (EXTRA CREDIT)

Predicting name of origami model based on images alone. Classification step

```
# Assuming you have already mounted Google Drive and imported necessary libraries
from google.colab import drive
drive.mount('/content/drive')

import os
import torch
from glob import glob
from PIL import Image
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms as T

# Define the CustomDataset class here
class CustomDataset(Dataset):
    def __init__(self, root, transformations=None):
        self.transformations = transformations
        self.im_paths = [im_path for im_path in sorted(glob(f"{root}/*/*/*.jpg"))]

        # Print the found image paths for debugging
        print(f"Image paths found: {self.im_paths}")

        self.cls_names, self.cls_counts, count, data_count = {}, {}, 0, 0
        for idx, im_path in enumerate(self.im_paths):
            class_name = self.get_class(im_path)
            if class_name not in self.cls_names:
                self.cls_names[class_name] = count
                self.cls_counts[class_name] = 1
                count += 1
            else:
                self.cls_counts[class_name] += 1

    def get_class(self, path): return os.path.dirname(path).split("/")[-1]

    def __len__(self): return len(self.im_paths)

    def __getitem__(self, idx):
        im_path = self.im_paths[idx]
        im = Image.open(im_path).convert("RGB")
        gt = self.cls_names[self.get_class(im_path)]

        if self.transformations is not None:
            im = self.transformations(im)

        return im, gt

# Set the root directory to your Google Drive path
```

```
root = "/content/drive/My Drive/dataset"  # Adjust as necessary

# Optionally define transformations
mean, std, im_size = [0.485, 0.456, 0.406], [0.229, 0.224, 0.225], 224
tfs = T.Compose([T.Resize((im_size, im_size)), T.ToTensor(), T.Normalize(mean=mean, std=std)])

# Create an instance of the dataset
dataset = CustomDataset(root=root, transformations=tfs)

# Check the length of the dataset
print(f"Number of images in dataset: {len(dataset)}")

# Optionally, view the first item
if len(dataset) > 0:
    img, label = dataset[0]
    print(f"Image size: {img.size()}, Label: {label}")
else:
    print("No images found in the dataset.")
```

```
Mounted at /content/drive
Image paths found: ['/content/drive/My Drive/dataset/animals/armadillo/2133.jpg', '/content/drive/My Drive/dataset/animals/armadillo/2147.jpg', '/content/drive/M
Number of images in dataset: 2561
Image size: torch.Size([3, 224, 224]), Label: 0
```

DISCOVERING CLASSES

```
# Assuming you have already mounted Google Drive and imported necessary libraries
from google.colab import drive
drive.mount('/content/drive')

import os
import torch
from glob import glob
from PIL import Image
from torch.utils.data import Dataset, DataLoader, random_split
from torchvision import transforms as T

# Define the CustomDataset class here
class CustomDataset(Dataset):
    def __init__(self, root, transformations=None):
        self.transformations = transformations
        self.im_paths = [im_path for im_path in sorted(glob(f"{root}/*/*/*.jpg"))]

        # Initialize class names and counts
        self.cls_names, self.cls_counts, count = {}, {}, 0
        for idx, im_path in enumerate(self.im_paths):
            class_name = self.get_class(im_path)
            if class_name not in self.cls_names:
```

```
                self.cls_names[class_name] = count
                self.cls_counts[class_name] = 1
                count += 1
            else:
                self.cls_counts[class_name] += 1

        # Print the found classes for debugging
        print(f"Classes found: {self.cls_names}")

    def get_class(self, path): return os.path.dirname(path).split("/")[-1]

    def __len__(self): return len(self.im_paths)

    def __getitem__(self, idx):
        im_path = self.im_paths[idx]
        im = Image.open(im_path).convert("RGB")
        gt = self.cls_names[self.get_class(im_path)]

        if self.transformations is not None:
            im = self.transformations(im)

        return im, gt

# Function to get DataLoaders
def get_dls(root, transformations, bs, split=[0.9, 0.05, 0.05], ns=4):
    ds = CustomDataset(root=root, transformations=transformations)

    total_len = len(ds)
    tr_len = int(total_len * split[0])
    vl_len = int(total_len * split[1])
    ts_len = total_len - (tr_len + vl_len)

    tr_ds, vl_ds, ts_ds = random_split(dataset=ds, lengths=[tr_len, vl_len, ts_len])

    tr_dl = DataLoader(tr_ds, batch_size=bs, shuffle=True, num_workers=ns)
    val_dl = DataLoader(vl_ds, batch_size=bs, shuffle=False, num_workers=ns)
    ts_dl = DataLoader(ts_ds, batch_size=1, shuffle=False, num_workers=ns)

    return tr_dl, val_dl, ts_dl, ds.cls_names

# Set the root directory to your Google Drive path
root = "/content/drive/My Drive/dataset"  # Adjust as necessary

# Optionally define transformations
mean, std, im_size = [0.485, 0.456, 0.406], [0.229, 0.224, 0.225], 224
tfs = T.Compose([T.Resize((im_size, im_size)), T.ToTensor(), T.Normalize(mean=mean, std=std)])

# Get the DataLoaders and classes
tr_dl, val_dl, ts_dl, classes = get_dls(root=root, transformations=tfs, bs=16)

# Print the lengths of the DataLoaders and the classes
print(f"Training DataLoader length: {len(tr_dl)}")
```

```
print(f"Validation DataLoader length: {len(val_dl)}")
print(f"Test DataLoader length: {len(ts_dl)}")
print("Classes:", classes)
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
Classes found: {'armadillo': 0, 'bear': 1, 'camel': 2, 'cat': 3, 'chameleon': 4, 'cow': 5, 'crab': 6, 'crocodile': 7, 'deer': 8, 'dog': 9, 'elephant': 10, 'fish
Training DataLoader length: 144
Validation DataLoader length: 8
Test DataLoader length: 129
Classes: {'armadillo': 0, 'bear': 1, 'camel': 2, 'cat': 3, 'chameleon': 4, 'cow': 5, 'crab': 6, 'crocodile': 7, 'deer': 8, 'dog': 9, 'elephant': 10, 'fish': 11,
/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py:617: UserWarning:

This DataLoader will create 4 worker processes in total. Our suggested max number of worker in current system is 2, which is smaller than what this DataLoader is
```

```
from google.colab import drive
drive.mount('/content/drive')
```

VISUALIZING DATASET AND KEYS

```
import random
from matplotlib import pyplot as plt

def tensor_2_im(t, t_type = "rgb"):

    gray_tfs = T.Compose([T.Normalize(mean = [ 0.], std = [1/0.5]), T.Normalize(mean = [-0.5], std = [1])])
    rgb_tfs = T.Compose([T.Normalize(mean = [ 0., 0., 0. ], std = [ 1/0.229, 1/0.224, 1/0.225 ]), T.Normalize(mean = [ -0.485, -0.456, -0.406 ], std = [ 1., 1., 1. ])

    invTrans = gray_tfs if t_type == "gray" else rgb_tfs

    return (invTrans(t) * 255).detach().squeeze().cpu().permute(1,2,0).numpy().astype(np.uint8) if t_type == "gray" else (invTrans(t) * 255).detach().cpu().permute(1,

def visualize(data, n_ims, rows, cmap = None, cls_names = None):

    assert cmap in ["rgb", "gray"], "Rasmni oq-qora yoki rangli ekanini aniqlashtirib bering!"
    if cmap == "rgb": cmap = "viridis"

    plt.figure(figsize = (20, 10))
    indekslar = [random.randint(0, len(data) - 1) for _ in range(n_ims)]
    for idx, indeks in enumerate(indekslar):

        im, gt = data[indeks]
        # Start plot
        plt.subplot(rows, n_ims // rows, idx + 1)
        if cmap: plt.imshow(tensor_2_im(im, cmap), cmap=cmap)
        else: plt.imshow(tensor_2_im(im))
        plt.axis('off')
        if cls_names is not None: plt.title(f"GT -> {cls_names[int(gt)]}")
```

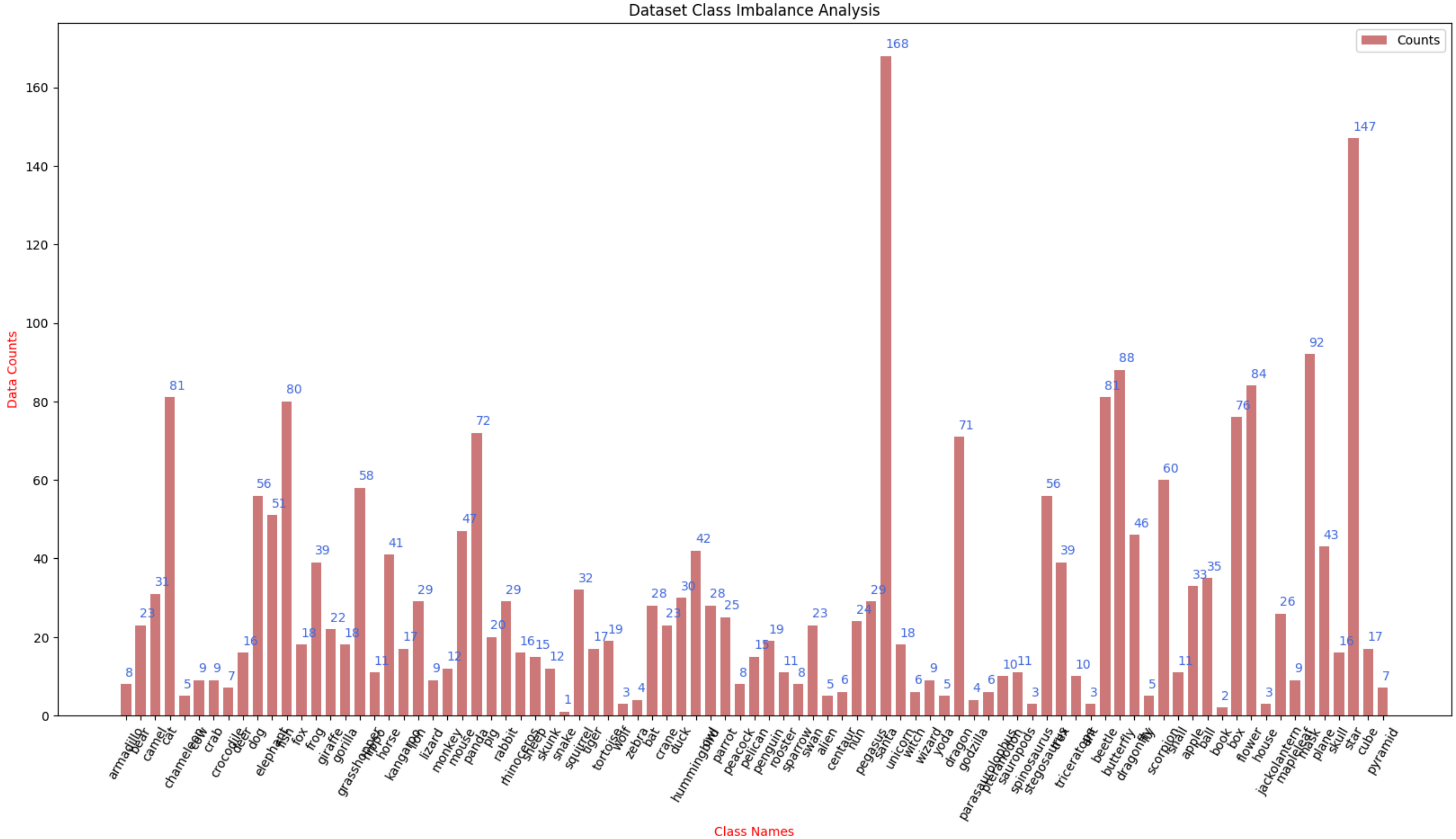
```
        else: plt.title(f"GT -> {gt}")

visualize(tr_dl.dataset, 20, 4, "rgb", list(classes.keys()))
```


CREATING DISTRIBUTION FOR NUMBER OF MODELS PER CLASS

```
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns

def data_analysis(root, transformations):
```

```
    # Create the dataset
    ds = CustomDataset(root=root, transformations=transformations)
    cls_counts = ds.cls_counts
    cls_names = list(cls_counts.keys())
    counts = np.array(list(cls_counts.values()))

    # Set up the figure and axis
    fig, ax = plt.subplots(figsize=(20, 10))
    indices = np.arange(len(counts))

    # Create bar plot
    ax.bar(indices, counts, width=0.7, color="firebrick", alpha=0.6, label='Counts')
    ax.set_xlabel("Class Names", color="red")
    ax.set_ylabel("Data Counts", color="red")
    ax.set_title("Dataset Class Imbalance Analysis")

    # Add counts on top of bars
    for i, v in enumerate(counts):
        ax.text(i - 0.05, v + 2, str(v), color="royalblue")

    # Create a density curve
    # To plot the density curve, we need to normalize the counts
    density = counts / counts.sum()  # Normalize to get a density-like representation



    # Set x-ticks
    ax.set_xticks(indices)
    ax.set_xticklabels(cls_names, rotation=60)

    # Show legend
    ax.legend()

    # Show the plot
    plt.show()

# Call the function
data_analysis(root=root, transformations=tfs)
```

```
Classes found: {'armadillo': 0, 'bear': 1, 'camel': 2, 'cat': 3, 'chameleon': 4, 'cow': 5, 'crab': 6, 'crocodile': 7, 'deer': 8, 'dog': 9, 'elephant': 10, 'fish
```




CREATING NORMAL DISTRIBUTION

```
import matplotlib.pyplot as plt
import numpy as np

def data_analysis(root, transformations):
    # Create the dataset
```