In [18]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.metrics import make_scorer

from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.svm import SVR
from xgboost import XGBRegressor
from catboost import CatBoostRegressor
from lightgbm import LGBMRegressor

import warnings
warnings.filterwarnings('ignore')

import pickle
```

In [2]:
```
dataset = pd.read_csv('train.csv')
```

In [4]:
```
dataset.head()
```

Out[4]:

	id	Sex	Length	Diameter	Height	Whole weight	Whole weight.1	Whole weight.2	Shell weight	Rings
0	0	F	0.550	0.430	0.150	0.7715	0.3285	0.1465	0.2400	11
1	1	F	0.630	0.490	0.145	1.1300	0.4580	0.2765	0.3200	11
2	2	I	0.160	0.110	0.025	0.0210	0.0055	0.0030	0.0050	6
3	3	M	0.595	0.475	0.150	0.9145	0.3755	0.2055	0.2500	10
4	4	I	0.555	0.425	0.130	0.7820	0.3695	0.1600	0.1975	9

In [5]:
```
dataset.tail()
```

Out[5]:

	id	Sex	Length	Diameter	Height	Whole weight	Whole weight.1	Whole weight.2	Shell weight	Rings
90610	90610	M	0.335	0.235	0.075	0.1585	0.0685	0.0370	0.0450	6
90611	90611	M	0.555	0.425	0.150	0.8790	0.3865	0.1815	0.2400	9
90612	90612	I	0.435	0.330	0.095	0.3215	0.1510	0.0785	0.0815	6
90613	90613	I	0.345	0.270	0.075	0.2000	0.0980	0.0490	0.0700	6
90614	90614	I	0.425	0.325	0.100	0.3455	0.1525	0.0785	0.1050	8

In [7]:

```
print('Number of Rows:',dataset.shape[0])
print('Number of Columns:',dataset.shape[1])
```

```
Number of Rows: 90615
Number of Columns: 10
```

In [8]:

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 90615 entries, 0 to 90614
Data columns (total 10 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   id              90615 non-null  int64  
 1   Sex             90615 non-null  object 
 2   Length          90615 non-null  float64
 3   Diameter        90615 non-null  float64
 4   Height          90615 non-null  float64
 5   Whole weight    90615 non-null  float64
 6   Whole weight.1  90615 non-null  float64
 7   Whole weight.2  90615 non-null  float64
 8   Shell weight    90615 non-null  float64
 9   Rings           90615 non-null  int64  
dtypes: float64(7), int64(2), object(1)
memory usage: 6.9+ MB
```

In [9]:

```
dataset.isnull().sum()
```

Out[9]:

```
id                0
Sex               0
Length            0
Diameter          0
Height            0
Whole weight      0
Whole weight.1    0
Whole weight.2    0
Shell weight      0
Rings             0
dtype: int64
```

In [10]:

```
dataset.duplicated().sum()
```

Out[10]:

```
0
```

In [11]:

```
dataset.describe()
```

Out[11]:

	id	Length	Diameter	Height	Whole weight	Whole weight.1	Whole weight.2	Shell weight	
count	90615.000000	90615.000000	90615.000000	90615.000000	90615.000000	90615.000000	90615.000000	90615.000000	906
mean	45307.000000	0.517098	0.401679	0.135464	0.789035	0.340778	0.169422	0.225898	
std	26158.441658	0.118217	0.098026	0.038008	0.457671	0.204428	0.100909	0.130203	
min	0.000000	0.075000	0.055000	0.000000	0.002000	0.001000	0.000500	0.001500	
25%	22653.500000	0.445000	0.345000	0.110000	0.419000	0.177500	0.086500	0.120000	
50%	45307.000000	0.545000	0.425000	0.140000	0.799500	0.330000	0.166000	0.225000	
75%	67960.500000	0.600000	0.470000	0.160000	1.067500	0.463000	0.232500	0.305000	
max	90614.000000	0.815000	0.650000	1.130000	2.825500	1.488000	0.760000	1.005000	2

In [12]:

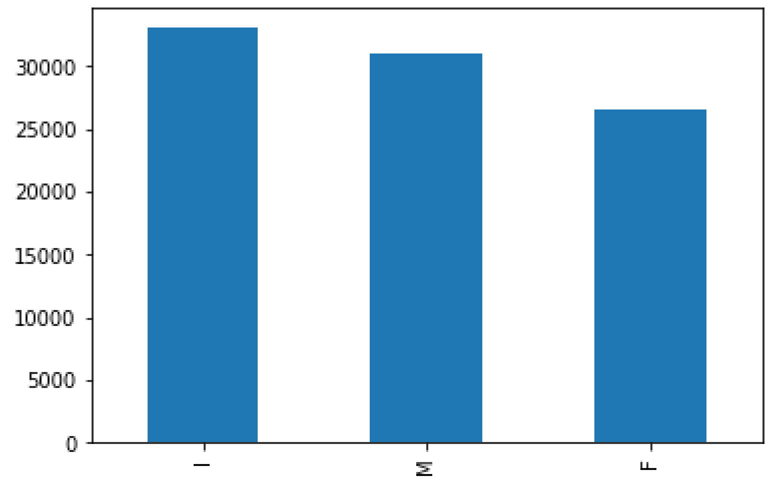
```
dataset.corr()
```

Out[12]:

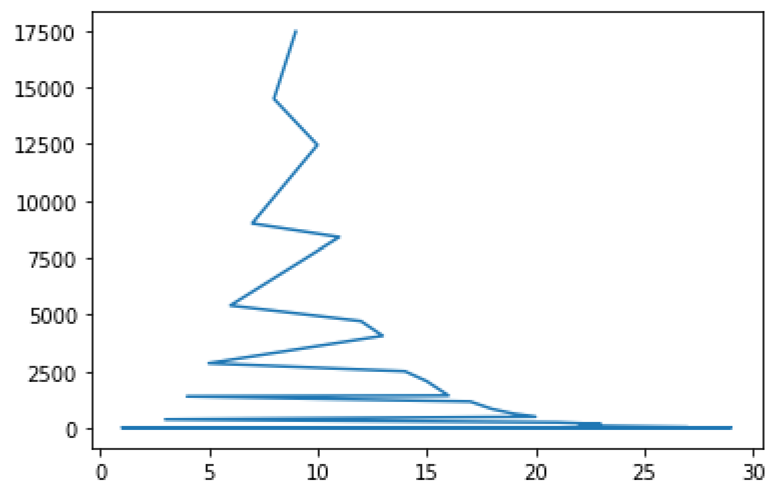
	id	Length	Diameter	Height	Whole weight	Whole weight.1	Whole weight.2	Shell weight	Rings
id	1.000000	0.004724	0.004290	0.005916	0.005228	0.004203	0.004878	0.005887	0.000938
Length	0.004724	1.000000	0.989732	0.916094	0.931449	0.909609	0.913134	0.911073	0.623786
Diameter	0.004290	0.989732	1.000000	0.919618	0.933848	0.908466	0.914668	0.917871	0.636832
Height	0.005916	0.916094	0.919618	1.000000	0.902344	0.861769	0.886132	0.904019	0.665772
Whole weight	0.005228	0.931449	0.933848	0.902344	1.000000	0.971249	0.974319	0.964201	0.617274
Whole weight.1	0.004203	0.909609	0.908466	0.861769	0.971249	1.000000	0.949227	0.911800	0.515067
Whole weight.2	0.004878	0.913134	0.914668	0.886132	0.974319	0.949227	1.000000	0.937069	0.588954
Shell weight	0.005887	0.911073	0.917871	0.904019	0.964201	0.911800	0.937069	1.000000	0.694766
Rings	0.000938	0.623786	0.636832	0.665772	0.617274	0.515067	0.588954	0.694766	1.000000

In [19]:

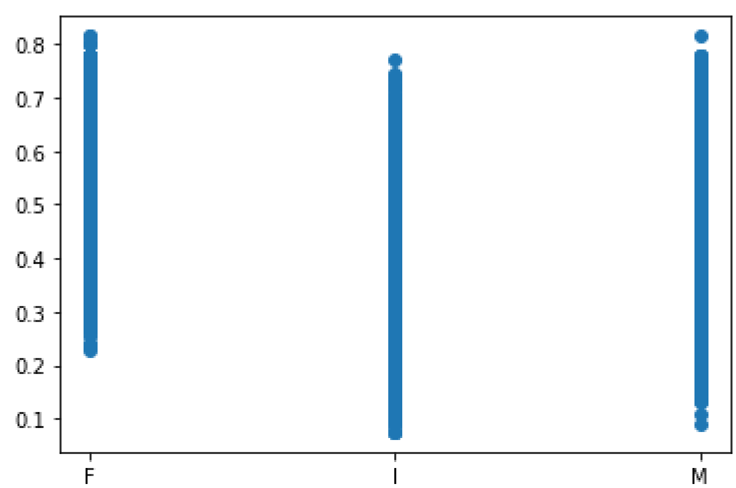
```
dataset.Sex.value_counts().plot(kind='bar')
plt.show()
```



In [21]:
```
dataset.Rings.value_counts().plot(kind='line')
plt.show()
```

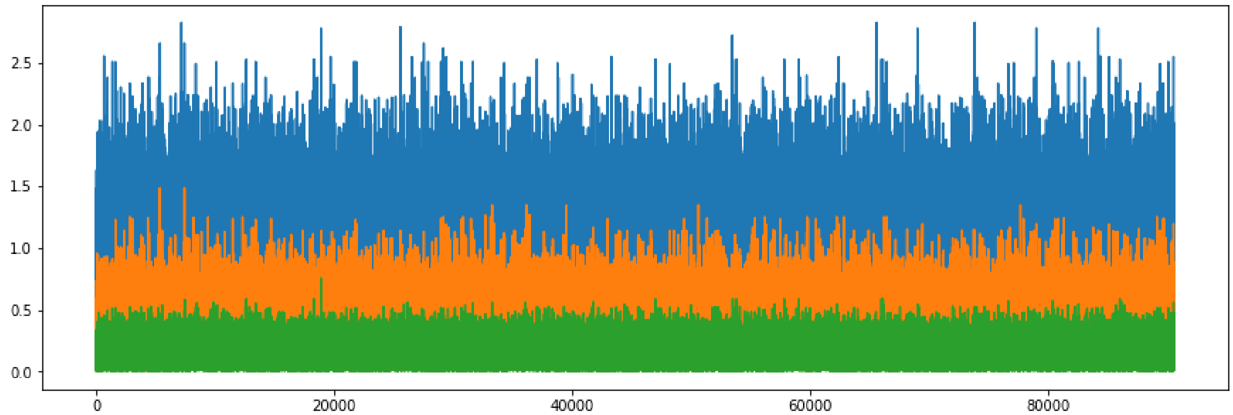


In [22]:
```
plt.scatter(dataset['Sex'], dataset['Length'])
plt.show()
```



In [35]:
```
plt.figure(figsize=(15,5))

dataset['Whole weight'].plot(kind='line')
dataset['Whole weight.1'].plot(kind='line')
dataset['Whole weight.2'].plot(kind='line')
```

```python
plt.show()
```



In [37]:
```python
le = LabelEncoder()
dataset['Sex'] = le.fit_transform(dataset['Sex'])
```

In [39]:
```python
X = dataset.drop(['id', 'Rings'], axis=1)
y = dataset['Rings']
```

In [40]:
```python
X_train,X_test, y_train,y_test = train_test_split(X,y,test_size=0.20,random_sta
```

In [41]:
```python
def rmsle_score(y_true, y_pred):
    y_true = np.maximum(y_true, 0)
    y_pred = np.maximum(y_pred, 0)
    squared_log_errors = (np.log1p(y_pred) - np.log1p(y_true)) ** 2
    mean_squared_log_error = np.mean(squared_log_errors)
    rmsle = np.sqrt(mean_squared_log_error)
    return rmsle
```

In [42]:
```python
rmsle_scorer = make_scorer(rmsle_score, greater_is_better=False)
```

In [43]:
```python
models = [
    LinearRegression(),
    DecisionTreeRegressor(),
    RandomForestRegressor(n_estimators=100, random_state=42),
    GradientBoostingRegressor(),
    SVR(),
    XGBRegressor(),
    CatBoostRegressor(verbose=0),
    LGBMRegressor()
]
```

In [52]:
```python
# Track best model and its RMSLE score

best_model = None
best_model_name = ''
best_rmsle = float('inf')

# Evaluate models
for model in models:
    # Train the model
    model.fit(X_train, y_train)
```

```
    # Predictions
    y_pred = model.predict(X_test)

    # Calculate RMSLE score
    rmsle = rmsle_score(y_test, y_pred)

    # Print RMSLE score
    print(f'{model.__class__.__name__}: RMSLE: {rmsle}')

    # Update best model if current model has lower RMSLE
    if rmsle < best_rmsle:
        best_model = model
        best_model_name = model.__class__.__name__
        best_rmsle = rmsle

# Print name of the best model
print(f'Best Model: {best_model_name}')
```

```
LinearRegression: RMSLE: 0.16788369061299582
DecisionTreeRegressor: RMSLE: 0.21672407015215117
RandomForestRegressor: RMSLE: 0.1559158972567478
GradientBoostingRegressor: RMSLE: 0.15567507932553024
SVR: RMSLE: 0.15893590278683453
XGBRegressor: RMSLE: 0.15192754235811698
CatBoostRegressor: RMSLE: 0.15128924362291235
LGBMRegressor: RMSLE: 0.15277213467670908
Best Model: CatBoostRegressor
```

In [54]:

```
pickle.dump(best_model,open('abalone_age_prediction.pkl','wb'))
```

In [ ]:

```
# model = pickle.load(open('abalone_age_prediction.pkl','rb'))
```

In [55]:

```
best_model = pickle.load(open('abalone_age_prediction.pkl','rb'))

data = {'Sex': ['Female'],
        'Length': [0.455],
        'Diameter': [0.365],
        'Height': [0.095],
        'Whole weight': [0.514],
        'Whole weight.1': [0.2245],
        'Whole weight.2': [0.101],
        'Shell weight': [0.15]}
predict_df = pd.DataFrame(data)

le = LabelEncoder()
predict_df['Sex'] = le.fit_transform(predict_df['Sex'])

prediction = best_model.predict(predict_df)

print('Predicted Rings:', prediction)
```

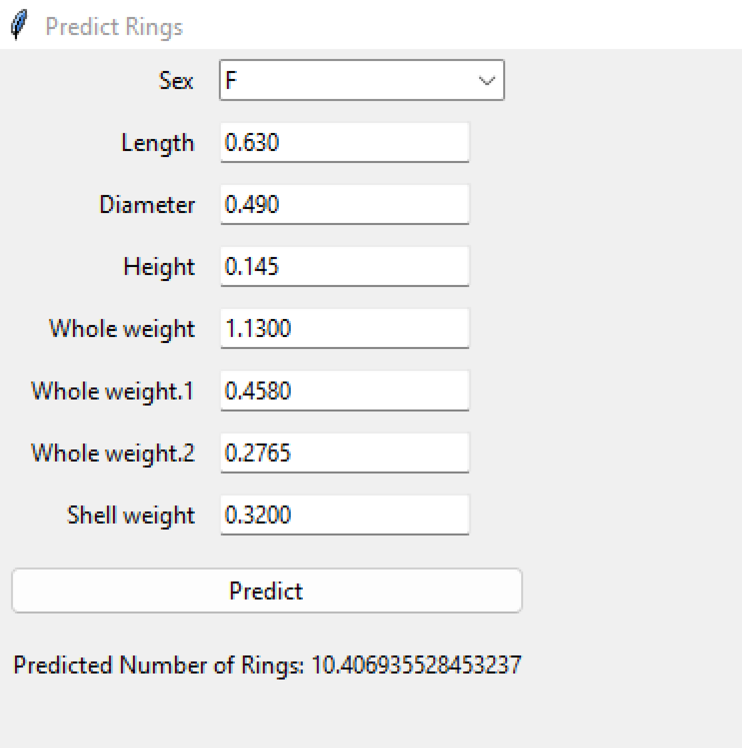
```
Predicted Rings: [8.66855482]
```

In [60]:

```
import tkinter as tk
from tkinter import ttk, messagebox
import pickle
import pandas as pd
from sklearn.preprocessing import LabelEncoder

# Load the best model
```

```
best_model = pickle.load(open('abalone_age_prediction.pkl','rb'))

# Create a dictionary to store feature names and their respective input widgets
feature_widgets = {}

# Load train data to get feature names for input
train_df = pd.read_csv('train.csv')
feature_names = train_df.columns.drop(['id', 'Rings'])

# Encode Sex feature
le = LabelEncoder()
train_df['Sex'] = le.fit_transform(train_df['Sex'])
sex_encode = {val: key for key, val in enumerate(le.classes_)}

# Create GUI window
root = tk.Tk()
root.title('Predict Rings')

# Function to predict the rings
def predict_rings():
    try:
        # Get input values from the user
        input_data = []
        for feature, widget in feature_widgets.items():
            if isinstance(widget, tk.StringVar):
                input_data.append(sex_encode[widget.get()])
            else:
                if widget.get().strip() == '':
                    raise ValueError(f'{feature} cannot be empty.')
                input_data.append(float(widget.get()))

        # Predict rings using the best model
        prediction = best_model.predict([input_data])[0]

        # Show prediction in the GUI window
        prediction_label.config(text=f'Predicted Number of Rings: {prediction}'
    except ValueError as ve:
        messagebox.showerror('Error', str(ve))
    except Exception as e:
        messagebox.showerror('Error', f'Error Occurred: {e}')

# Function to create input widgets for features
def create_input_widgets():
    for idx, feature in enumerate(feature_names):
        if feature == 'Sex':
            label = ttk.Label(root, text=feature)
            label.grid(row=idx, column=0, padx=5, pady=5, sticky='e')
            sex_options = list(sex_encode.keys())
            sex_var = tk.StringVar(root)
            sex_dropdown = ttk.Combobox(root, textvariable=sex_var, values=sex_
            sex_dropdown.grid(row=idx, column=1, padx=5, pady=5, sticky='w')
            feature_widgets[feature] = sex_var
        else:
            label = ttk.Label(root, text=feature)
            label.grid(row=idx, column=0, padx=5, pady=5, sticky='e')
            entry = ttk.Entry(root)
            entry.grid(row=idx, column=1, padx=5, pady=5, sticky='w')
            feature_widgets[feature] = entry

# Create input widgets for features
```

```python
create_input_widgets()

# Create Predict button
predict_button = ttk.Button(root, text='Predict', command=predict_rings)
predict_button.grid(row=len(feature_names), column=0, columnspan=2, padx=5, pad

# Label to display prediction result
prediction_label = ttk.Label(root, text='')
prediction_label.grid(row=len(feature_names)+1, column=0, columnspan=2, padx=5,

# Run the GUI
root.mainloop()
```



CONNECT WITH ME :

LinkedIn GitHub kaggle Medium

PRASADMJADHAV2