SUPERMARKET SALES PREDICTION

In [65]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import os
import scipy as sp
import warnings
import datetime
import pickle
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_sco
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
from sklearn.linear_model import Ridge, Lasso
from xgboost import XGBRegressor
from lightgbm import LGBMRegressor
from catboost import CatBoostRegressor
from sklearn.neural_network import MLPRegressor
warnings.filterwarnings("ignore")
%matplotlib inline
```

READING THE DATASET

In [2]:
```
data = pd.read_csv("supermarket_sales - Sheet1.csv")
```

In [3]:
```
data
```

Out[3]:

	Invoice ID	Branch	City	Customer type	Gender	Product line	Unit price	Quantity	Tax 5%	
0	750-67-8428	A	Yangon	Member	Female	Health and beauty	74.69	7	26.1415	5
1	226-31-3081	C	Naypyitaw	Normal	Female	Electronic accessories	15.28	5	3.8200	
2	631-41-3108	A	Yangon	Normal	Male	Home and lifestyle	46.33	7	16.2155	3
3	123-19-1176	A	Yangon	Member	Male	Health and beauty	58.22	8	23.2880	4
4	373-73-7910	A	Yangon	Normal	Male	Sports and travel	86.31	7	30.2085	6
...	...	...	...	...	...	...	...	...	...	
995	233-67-5758	C	Naypyitaw	Normal	Male	Health and beauty	40.35	1	2.0175	
996	303-96-2227	B	Mandalay	Normal	Female	Home and lifestyle	97.38	10	48.6900	10
997	727-02-1313	A	Yangon	Member	Male	Food and beverages	31.84	1	1.5920	
998	347-56-2442	A	Yangon	Normal	Male	Home and lifestyle	65.82	1	3.2910	
999	849-09-3807	A	Yangon	Member	Female	Fashion accessories	88.34	7	30.9190	6

1000 rows × 17 columns

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

Out[4]:

	Invoice ID	Branch	City	Customer type	Gender	Product line	Unit price	Quantity	Tax 5%	
0	750-67-8428	A	Yangon	Member	Female	Health and beauty	74.69	7	26.1415	548.
1	226-31-3081	C	Naypyitaw	Normal	Female	Electronic accessories	15.28	5	3.8200	80.
2	631-41-3108	A	Yangon	Normal	Male	Home and lifestyle	46.33	7	16.2155	340.
3	123-19-1176	A	Yangon	Member	Male	Health and beauty	58.22	8	23.2880	489.
4	373-73-7910	A	Yangon	Normal	Male	Sports and travel	86.31	7	30.2085	634.

In [5]:
```
data.describe()
```

Out[5]:

	Unit price	Quantity	Tax 5%	Total	cogs	gross margin percentage	i
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.00000	1.000000e+03	1000.(
mean	55.672130	5.510000	15.379369	322.966749	307.58738	4.761905e+00	15.3
std	26.494628	2.923431	11.708825	245.885335	234.17651	6.131498e-14	11.7
min	10.080000	1.000000	0.508500	10.678500	10.17000	4.761905e+00	0.5
25%	32.875000	3.000000	5.924875	124.422375	118.49750	4.761905e+00	5.9
50%	55.230000	5.000000	12.088000	253.848000	241.76000	4.761905e+00	12.0
75%	77.935000	8.000000	22.445250	471.350250	448.90500	4.761905e+00	22.4
max	99.960000	10.000000	49.650000	1042.650000	993.00000	4.761905e+00	49.6

In [6]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 17 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   Invoice ID               1000 non-null   object 
 1   Branch                   1000 non-null   object 
 2   City                     1000 non-null   object 
 3   Customer type            1000 non-null   object 
 4   Gender                   1000 non-null   object 
 5   Product line             1000 non-null   object 
 6   Unit price               1000 non-null   float64
 7   Quantity                 1000 non-null   int64  
 8   Tax 5%                   1000 non-null   float64
 9   Total                    1000 non-null   float64
 10  Date                     1000 non-null   object 
 11  Time                     1000 non-null   object 
 12  Payment                  1000 non-null   object 
 13  cogs                     1000 non-null   float64
 14  gross margin percentage  1000 non-null   float64
 15  gross income             1000 non-null   float64
 16  Rating                   1000 non-null   float64
dtypes: float64(7), int64(1), object(9)
memory usage: 132.9+ KB
```

DATA PREPROCESSING AND VISUALISATION

In [9]:
```
data.shape
```

Out[9]:
```
(1000, 17)
```

In [10]:
```
data.dtypes
```

Out[10]:
```
Invoice ID                  object
Branch                      object
City                        object
Customer type               object
Gender                      object
Product line                object
Unit price                 float64
Quantity                     int64
Tax 5%                     float64
Total                      float64
Date                        object
Time                        object
Payment                     object
cogs                       float64
gross margin percentage    float64
gross income               float64
Rating                     float64
dtype: object
```

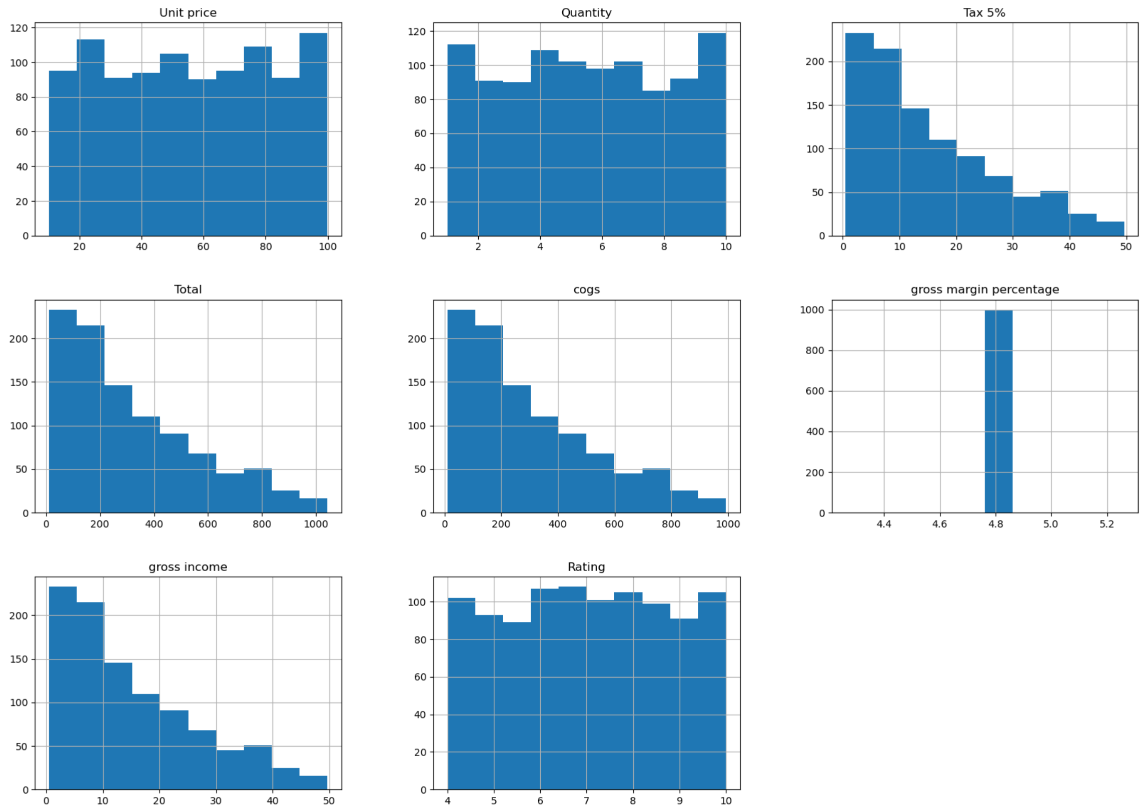
In [12]:
```
data.columns
```

Out[12]:
```
Index(['Invoice ID', 'Branch', 'City', 'Customer type', 'Gender',
       'Product line', 'Unit price', 'Quantity', 'Tax 5%', 'Total', 'Dat
e',
       'Time', 'Payment', 'cogs', 'gross margin percentage', 'gross incom
e',
       'Rating'],
      dtype='object')
```

In [13]:
```
data.isnull().sum()
```

Out[13]:
```
Invoice ID                 0
Branch                     0
City                       0
Customer type              0
Gender                     0
Product line               0
Unit price                 0
Quantity                   0
Tax 5%                     0
Total                      0
Date                       0
Time                       0
Payment                    0
cogs                       0
gross margin percentage    0
gross income               0
Rating                     0
dtype: int64
```

In [14]:
```
data.hist(figsize=(20,14))
plt.show()
```



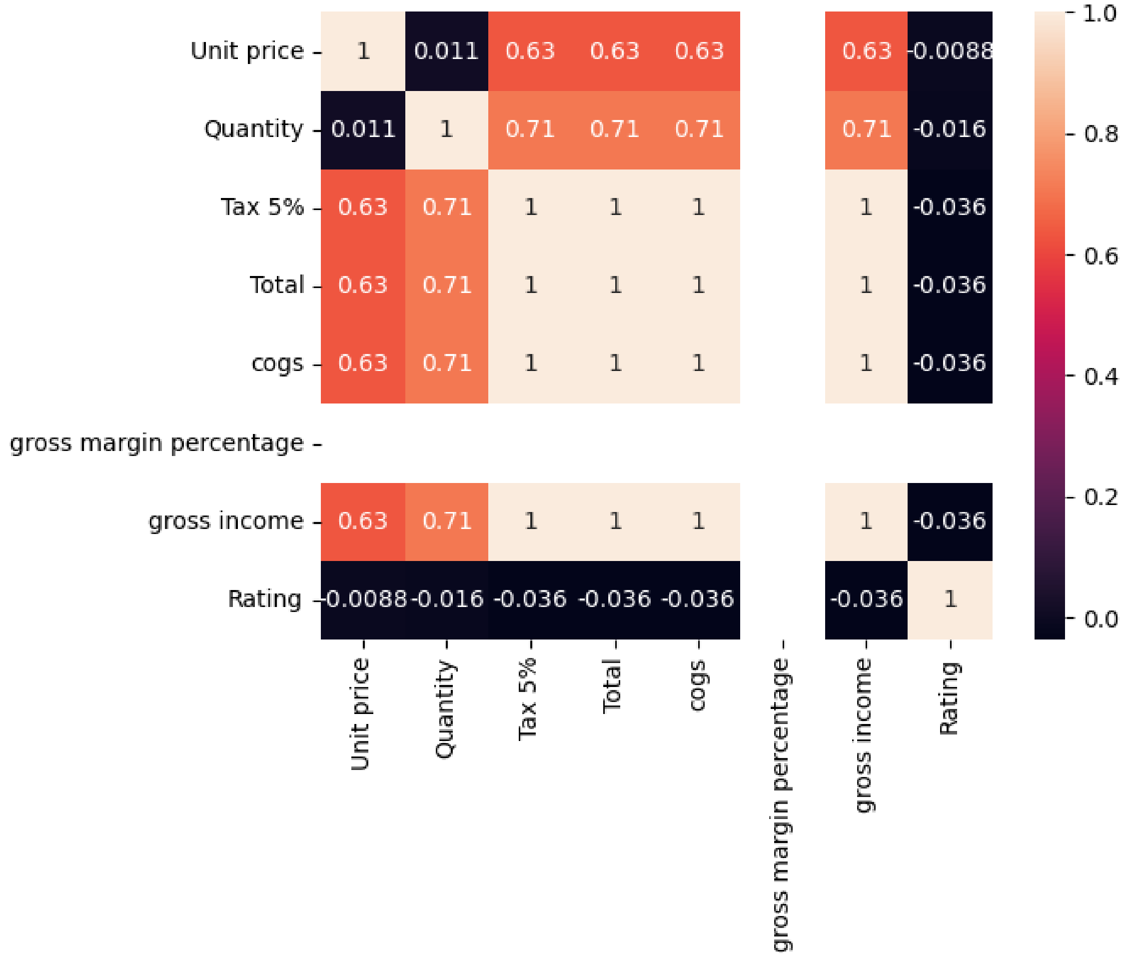
In [15]:
```
data.corr()
```

Out[15]:

	Unit price	Quantity	Tax 5%	Total	cogs	gross margin percentage	gross income	R
Unit price	1.000000	0.010778	0.633962	0.633962	0.633962	NaN	0.633962	-0.00
Quantity	0.010778	1.000000	0.705510	0.705510	0.705510	NaN	0.705510	-0.01
Tax 5%	0.633962	0.705510	1.000000	1.000000	1.000000	NaN	1.000000	-0.03
Total	0.633962	0.705510	1.000000	1.000000	1.000000	NaN	1.000000	-0.03
cogs	0.633962	0.705510	1.000000	1.000000	1.000000	NaN	1.000000	-0.03
gross margin percentage	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
gross income	0.633962	0.705510	1.000000	1.000000	1.000000	NaN	1.000000	-0.03
Rating	-0.008778	-0.015815	-0.036442	-0.036442	-0.036442	NaN	-0.036442	1.00

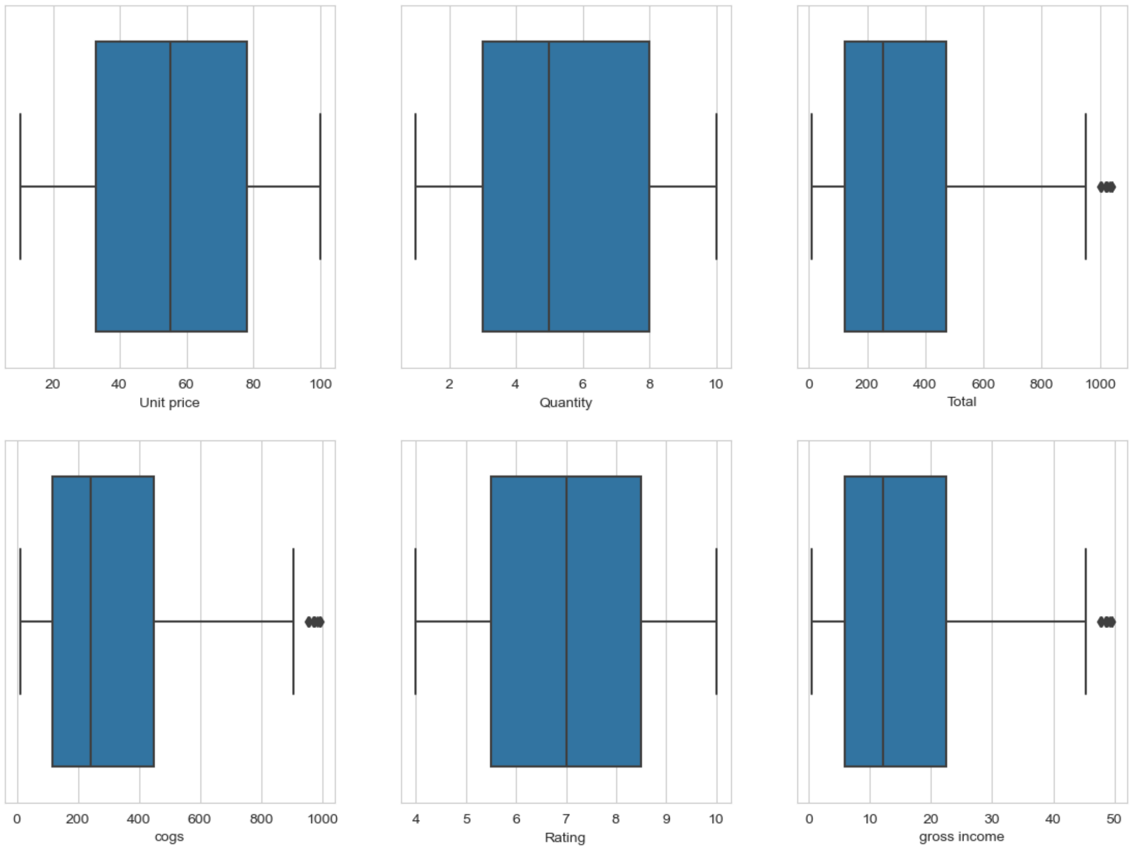
In [17]:
```
#plt.figure(figsize = (12,10))

sns.heatmap(data.corr(), annot =True)
```

Out[17]: `<Axes: >`



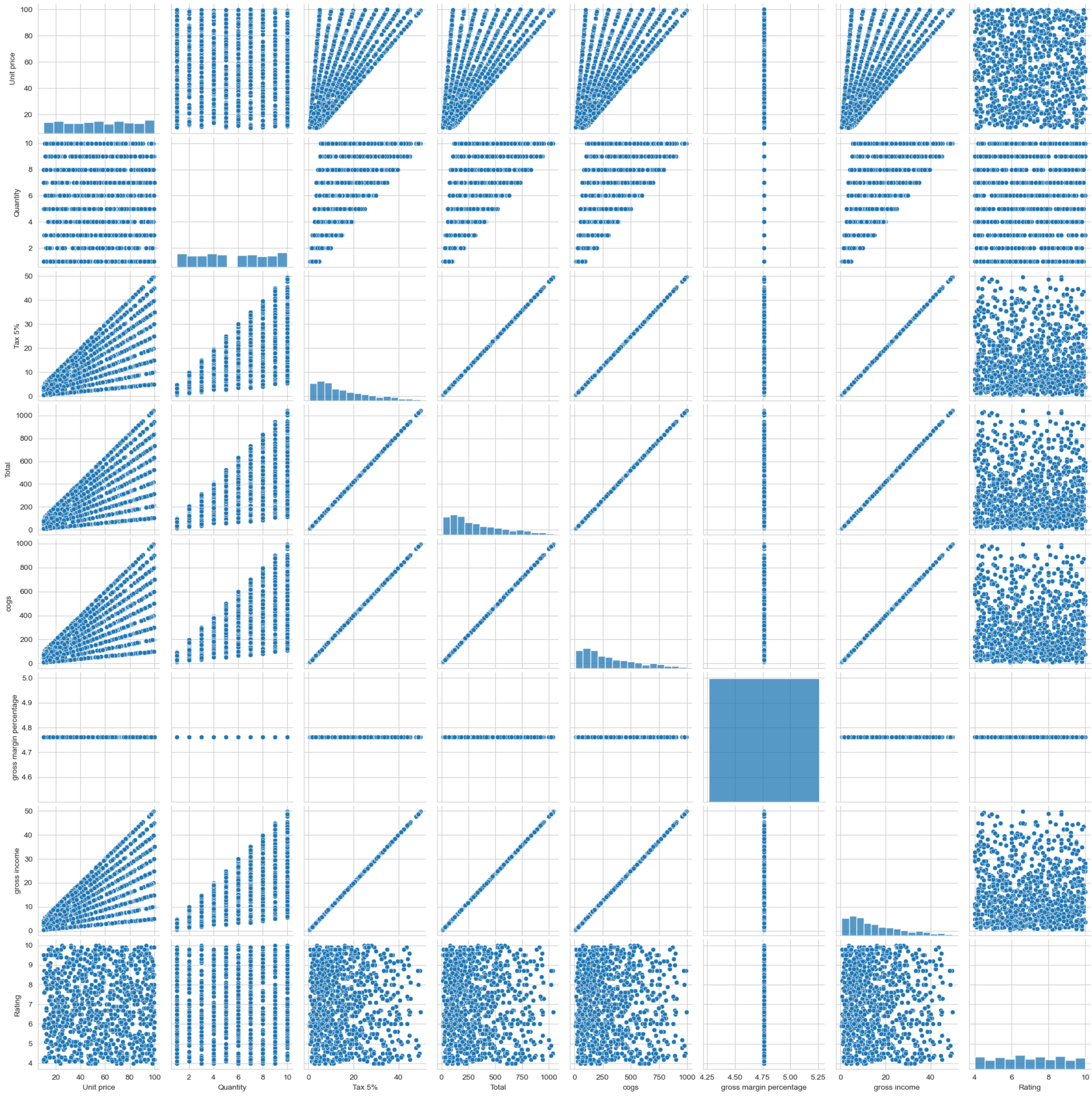
In [18]:
```
plt.figure(figsize=(14,10))
sns.set_style(style='whitegrid')
plt.subplot(2,3,1)
sns.boxplot(x='Unit price',data=data)
plt.subplot(2,3,2)
sns.boxplot(x='Quantity',data=data)
plt.subplot(2,3,3)
sns.boxplot(x='Total',data=data)
plt.subplot(2,3,4)
sns.boxplot(x='cogs',data=data)
plt.subplot(2,3,5)
sns.boxplot(x='Rating',data=data)
plt.subplot(2,3,6)
sns.boxplot(x='gross income',data=data)
```

Out[18]:
```
<Axes: xlabel='gross income'>
```



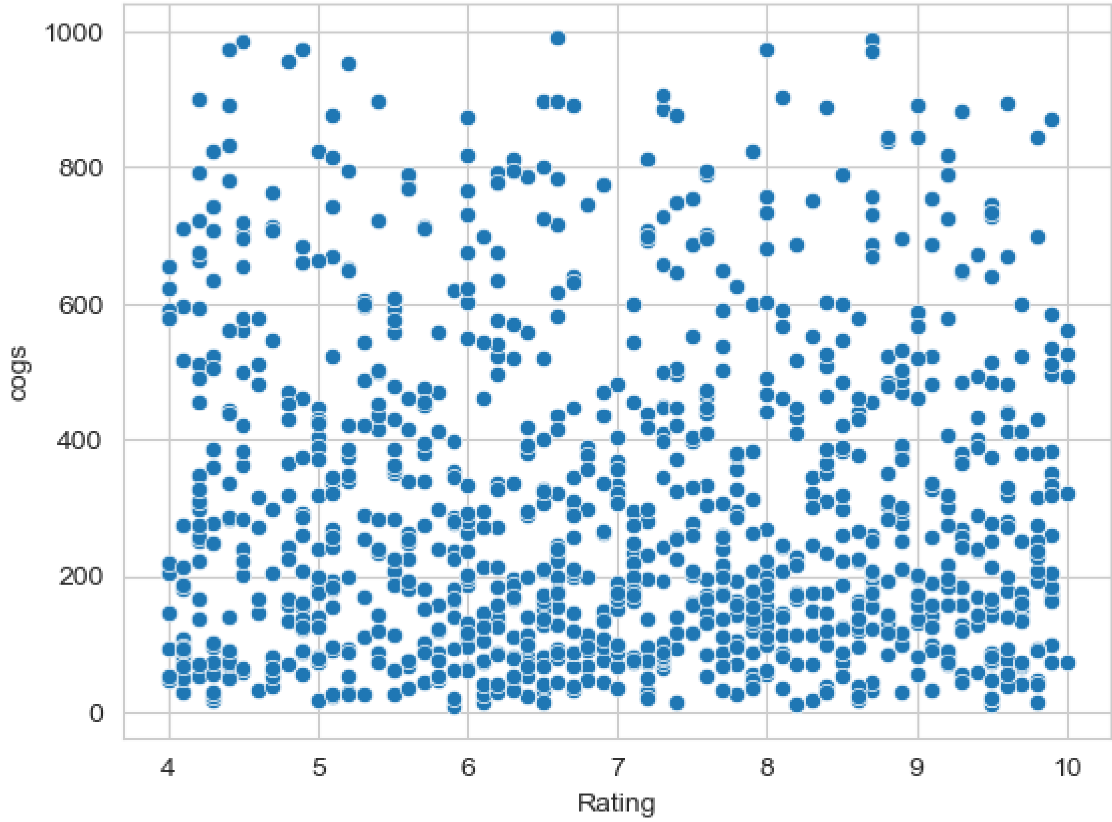
In [19]:
```
sns.pairplot(data=data)
```

Out[19]: <seaborn.axisgrid.PairGrid at 0x2b2ca829e40>



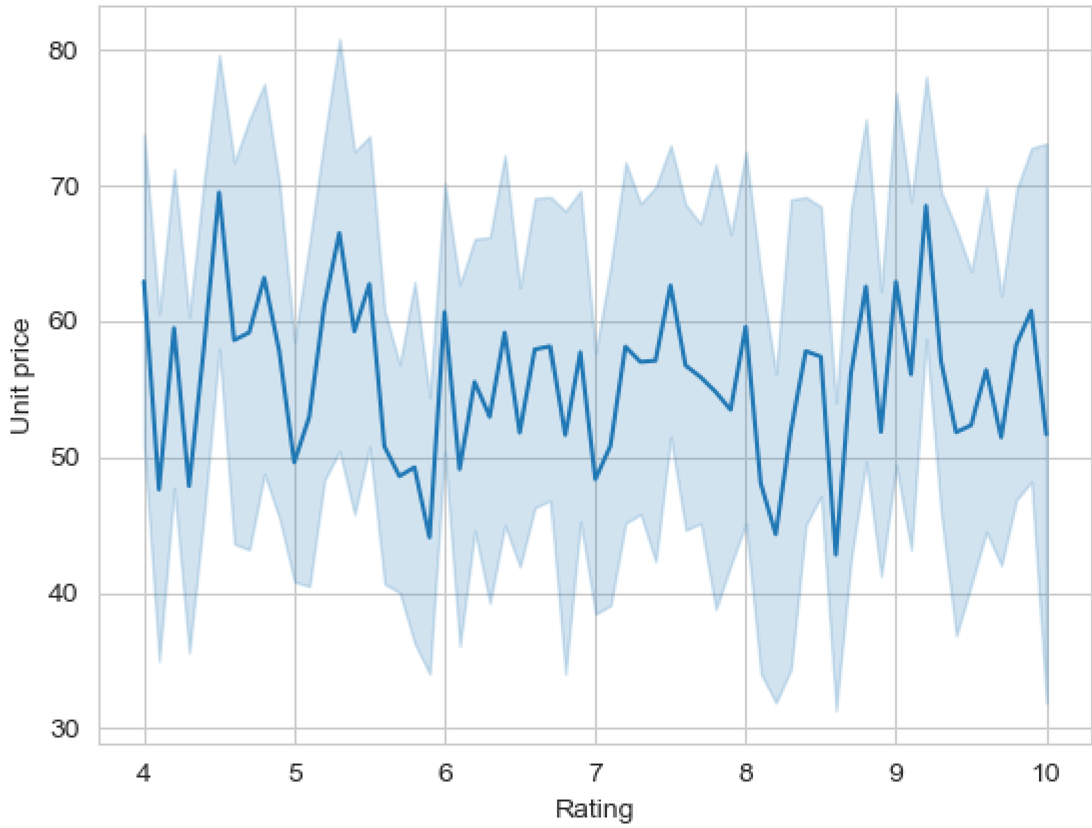
In [20]:
```
sns.scatterplot(x='Rating', y= 'cogs', data=data)
```

Out[20]: <Axes: xlabel='Rating', ylabel='cogs'>

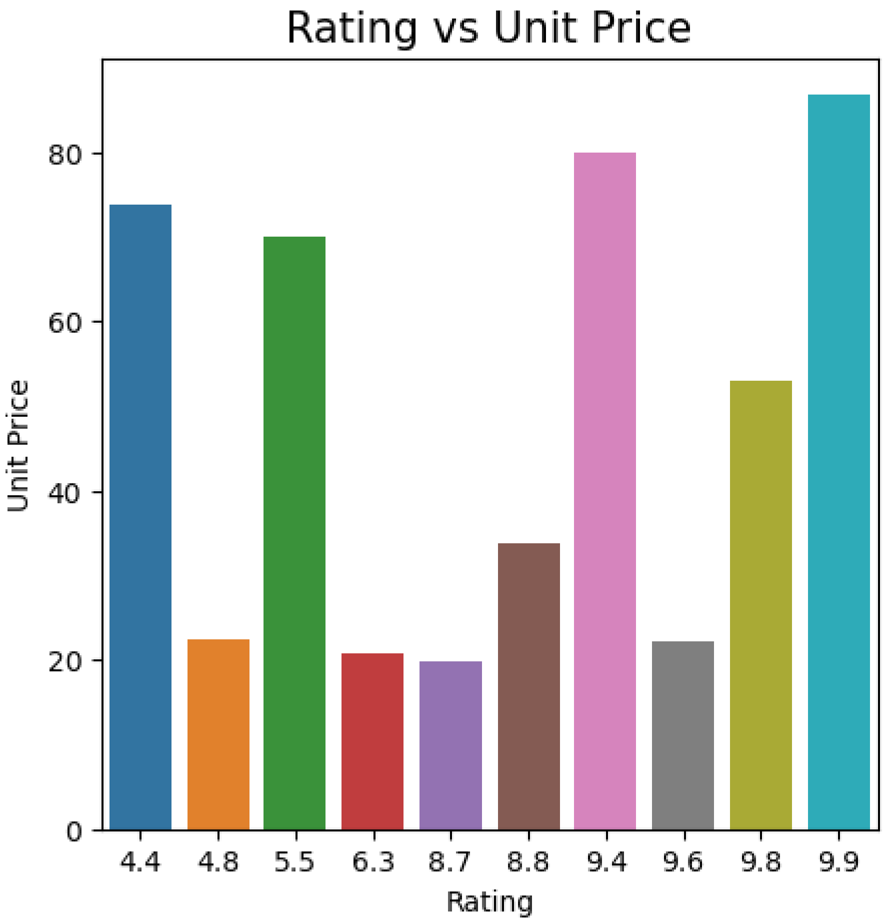


In [23]:
```
sns.lineplot(x='Rating', y= 'Unit price', data=data)
```

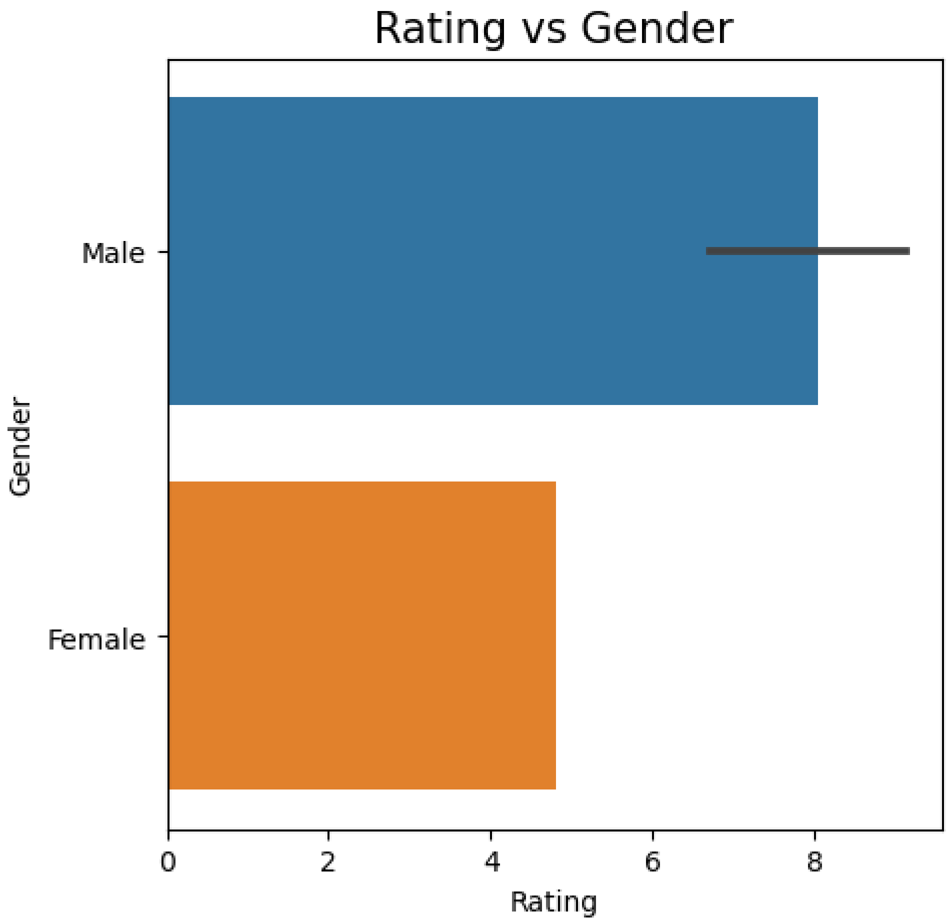
Out[23]: <Axes: xlabel='Rating', ylabel='Unit price'>



In [24]:
```
plt.style.use("default")
plt.figure(figsize=(5,5))
sns.barplot(x="Rating", y="Unit price", data=data[170:180])
plt.title("Rating vs Unit Price",fontsize=15)
plt.xlabel("Rating")
plt.ylabel("Unit Price")
plt.show()
```



In [25]:
```
plt.style.use("default")
plt.figure(figsize=(5,5))
sns.barplot(x="Rating", y="Gender", data=data[170:180])
plt.title("Rating vs Gender",fontsize=15)
plt.xlabel("Rating")
plt.ylabel("Gender")
plt.show()
```



In [26]:
```
#=========================finding categorical features===================
list_1=list(data.columns)
list_cate=[]
for i in list_1:
    if data[i].dtype=='object':
        list_cate.append(i)
```

In [27]:
```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
for i in list_cate:
    data[i]=le.fit_transform(data[i])
```

In [28]:
```python
data
```

Out[28]:

	Invoice ID	Branch	City	Customer type	Gender	Product line	Unit price	Quantity	Tax 5%	Total
0	814	0	2	0	0	3	74.69	7	26.1415	548.9715
1	142	2	1	1	0	0	15.28	5	3.8200	80.2200
2	653	0	2	1	1	4	46.33	7	16.2155	340.5255
3	18	0	2	0	1	3	58.22	8	23.2880	489.0480
4	339	0	2	1	1	5	86.31	7	30.2085	634.3785
...	...	...	...	...	...	...	...	...	...	...
995	153	2	1	1	1	3	40.35	1	2.0175	42.3675
996	250	1	0	1	0	4	97.38	10	48.6900	1022.4900
997	767	0	2	0	1	2	31.84	1	1.5920	33.4320
998	308	0	2	1	1	4	65.82	1	3.2910	69.1110
999	935	0	2	0	0	1	88.34	7	30.9190	649.2990

1000 rows × 17 columns

In [41]:
```python
data.drop(['Invoice ID', 'Date', 'Time', 'gross margin percentage'], axis=1
```

In [42]:
```python
data.columns
```

Out[42]:
```
Index(['Branch', 'City', 'Customer type', 'Gender', 'Product line',
       'Unit price', 'Quantity', 'Tax 5%', 'Total', 'Payment', 'cogs',
       'gross income', 'Rating'],
      dtype='object')
```

In [29]:
```python
y=data['Total']
x=data.drop('Total',axis=1)
```

In [30]:
```
x
```

Out[30]:

	Invoice ID	Branch	City	Customer type	Gender	Product line	Unit price	Quantity	Tax 5%	Date	Tim
0	814	0	2	0	0	3	74.69	7	26.1415	26	14
1	142	2	1	1	0	0	15.28	5	3.8200	87	2
2	653	0	2	1	1	4	46.33	7	16.2155	81	15
3	18	0	2	0	1	3	58.22	8	23.2880	19	48
4	339	0	2	1	1	5	86.31	7	30.2085	57	2
...	...	...	...	...	...	...	...	...	...	...	..
995	153	2	1	1	1	3	40.35	1	2.0175	21	17
996	250	1	0	1	0	4	97.38	10	48.6900	70	34
997	767	0	2	0	1	2	31.84	1	1.5920	58	15
998	308	0	2	1	1	4	65.82	1	3.2910	45	26
999	935	0	2	0	0	1	88.34	7	30.9190	40	16

1000 rows × 16 columns

In [31]:
```
y
```

Out[31]:
```
0       548.9715
1        80.2200
2       340.5255
3       489.0480
4       634.3785
          ...
995      42.3675
996    1022.4900
997      33.4320
998      69.1110
999     649.2990
Name: Total, Length: 1000, dtype: float64
```

In [44]:
```
# Standardize numerical features to have mean=0 and standard deviation=1
scaler = StandardScaler()
x_scaled = scaler.fit_transform(x)
with open('scaler.pkl', 'wb') as scaler_file:
    pickle.dump(scaler, scaler_file)
```

TRAINING AND TESTING OF DATA

In [47]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x_scaled,y,random_state=0,te
```

In [48]:
```
x_train
```

Out[48]:
```
array([[ 0.62527065, -1.20897001,  1.21017372, ...,  0.        ,
         1.40140764, -1.55595718],
       [-0.9855374 ,  0.01468385, -1.22969265, ...,  0.        ,
        -0.99811154, -0.91557371],
       [ 0.24768339, -1.20897001,  1.21017372, ...,  0.        ,
         0.33116646, -0.85735703],
       ...,
       [-0.83658096, -1.20897001,  1.21017372, ...,  0.        ,
        -1.26248888,  0.7144933 ],
       [-1.26959388, -1.20897001,  1.21017372, ...,  0.        ,
        -0.38591649,  0.7144933 ],
       [-0.00173205,  0.01468385, -1.22969265, ...,  0.        ,
        -0.72249773, -1.20665711]])
```

In [49]:
```
x_train.shape
```

Out[49]:
```
(800, 16)
```

In [50]:
```
y_train.shape
```

Out[50]:
```
(800,)
```

APPLYING DIFFERENT REGRESSION MODELS

In [52]:
```
#RANDOM FOREST
rf_reg = RandomForestRegressor(random_state=42)
rf_reg.fit(x_train, y_train)
y_pred_rf = rf_reg.predict(x_test)
rmse_rf = np.sqrt(mean_squared_error(y_test, y_pred_rf))
mae_rf = mean_absolute_error(y_test, y_pred_rf)
with open('rf_model.pkl', 'wb') as file:
    pickle.dump(rf_reg, file)
```

In [54]:
```
#SVM (SUPPORT VECTOR MACHINE)
svm_reg = SVR()
svm_reg.fit(x_train, y_train)
y_pred_svm = svm_reg.predict(x_test)
rmse_svm = np.sqrt(mean_squared_error(y_test, y_pred_svm))
mae_svm = mean_absolute_error(y_test, y_pred_svm)
with open('svm_model.pkl', 'wb') as file:
    pickle.dump(svm_reg, file)
```

In [55]:
```
#K NEAREST NEIGHBORS
knn_reg = KNeighborsRegressor()
knn_reg.fit(x_train, y_train)
y_pred_knn = knn_reg.predict(x_test)
rmse_knn = np.sqrt(mean_squared_error(y_test, y_pred_knn))
mae_knn = mean_absolute_error(y_test, y_pred_knn)
with open('knn_model.pkl', 'wb') as file:
    pickle.dump(knn_reg, file)
```

In [57]:
```
#RIDGE REGRESSOR
ridge_reg = Ridge()
ridge_reg.fit(x_train, y_train)
y_pred_ridge = ridge_reg.predict(x_test)
rmse_ridge = np.sqrt(mean_squared_error(y_test, y_pred_ridge))
mae_ridge = mean_absolute_error(y_test, y_pred_ridge)
with open('ridge_model.pkl', 'wb') as file:
    pickle.dump(ridge_reg, file)
```

In [58]:
```
#LASSO REGRESSOR
lasso_reg = Lasso()
lasso_reg.fit(x_train, y_train)
y_pred_lasso = lasso_reg.predict(x_test)
rmse_lasso = np.sqrt(mean_squared_error(y_test, y_pred_lasso))
mae_lasso = mean_absolute_error(y_test, y_pred_lasso)
with open('lasso_model.pkl', 'wb') as file:
    pickle.dump(lasso_reg, file)
```

In [59]:
```
#XGBoost REGRESSOR
import xgboost as xgb
dtrain = xgb.DMatrix(x_train, label=y_train)
dtest = xgb.DMatrix(x_test)

params = {
    'objective': 'reg:squarederror',
    'eval_metric': 'rmse',
    'eta': 0.1,
    'max_depth': 6,
    'subsample': 0.8,
    'colsample_bytree': 0.8
}

xgb_reg = xgb.train(params, dtrain, num_boost_round=100)
y_pred_xgb = xgb_reg.predict(dtest)
rmse_xgb = np.sqrt(mean_squared_error(y_test, y_pred_xgb))
mae_xgb = mean_absolute_error(y_test, y_pred_xgb)

with open('xgb_model.pkl', 'wb') as file:
    pickle.dump(xgb_reg, file)
```

In [60]:
```
# LGBMRegressor
lgbm_reg = LGBMRegressor(
    boosting_type='gbdt',
    num_leaves=31,
    learning_rate=0.05,
    n_estimators=100,
    reg_alpha=0.1,
    reg_lambda=0.1,
    min_child_samples=20,
    force_row_wise=True,
    random_state=42
)


lgbm_reg.fit(x_train, y_train)


y_pred_lgbm = lgbm_reg.predict(x_test)


rmse_lgbm = np.sqrt(mean_squared_error(y_test, y_pred_lgbm))
mae_lgbm = mean_absolute_error(y_test, y_pred_lgbm)
with open('lgbm_model.pkl', 'wb') as file:
    pickle.dump(lgbm_reg, file)
```

...

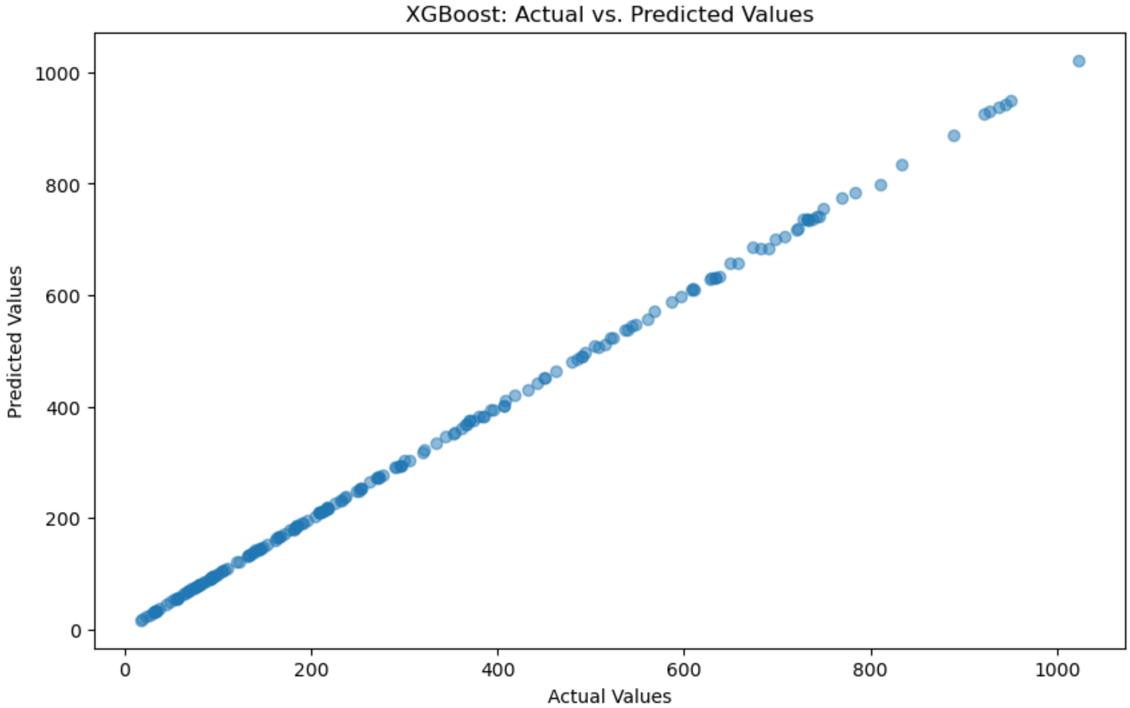
In [61]:
```
#CATBOOST REGRESSOR
catboost_reg = CatBoostRegressor(silent=True)
catboost_reg.fit(x_train, y_train)
y_pred_catboost = catboost_reg.predict(x_test)
rmse_catboost = np.sqrt(mean_squared_error(y_test, y_pred_catboost))
mae_catboost = mean_absolute_error(y_test, y_pred_catboost)
with open('catboost_model.pkl', 'wb') as file:
    pickle.dump(catboost_reg, file)
```

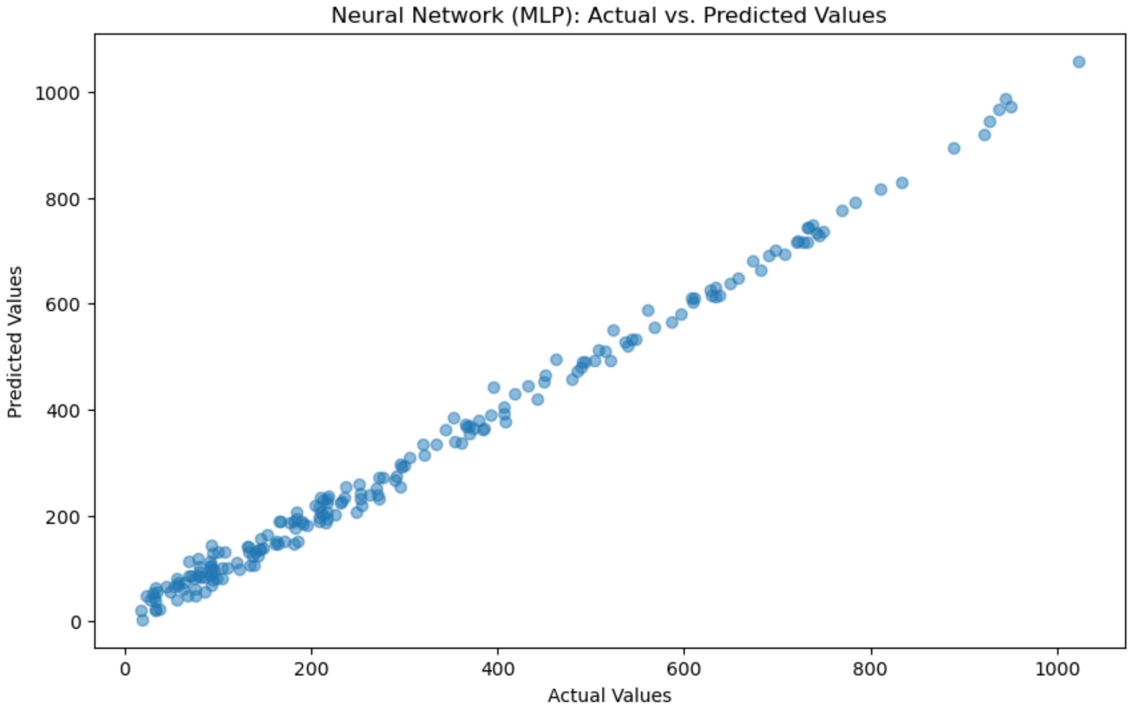
In [62]:
```
# MLPRegressor
mlp_reg = MLPRegressor(random_state=42, max_iter=1000)  # Increase max_iter

mlp_reg.fit(x_train, y_train)

y_pred_mlp = mlp_reg.predict(x_test)

rmse_mlp = np.sqrt(mean_squared_error(y_test, y_pred_mlp))
mae_mlp = mean_absolute_error(y_test, y_pred_mlp)

with open('mlp_model.pkl', 'wb') as file:
    pickle.dump(mlp_reg, file)
```

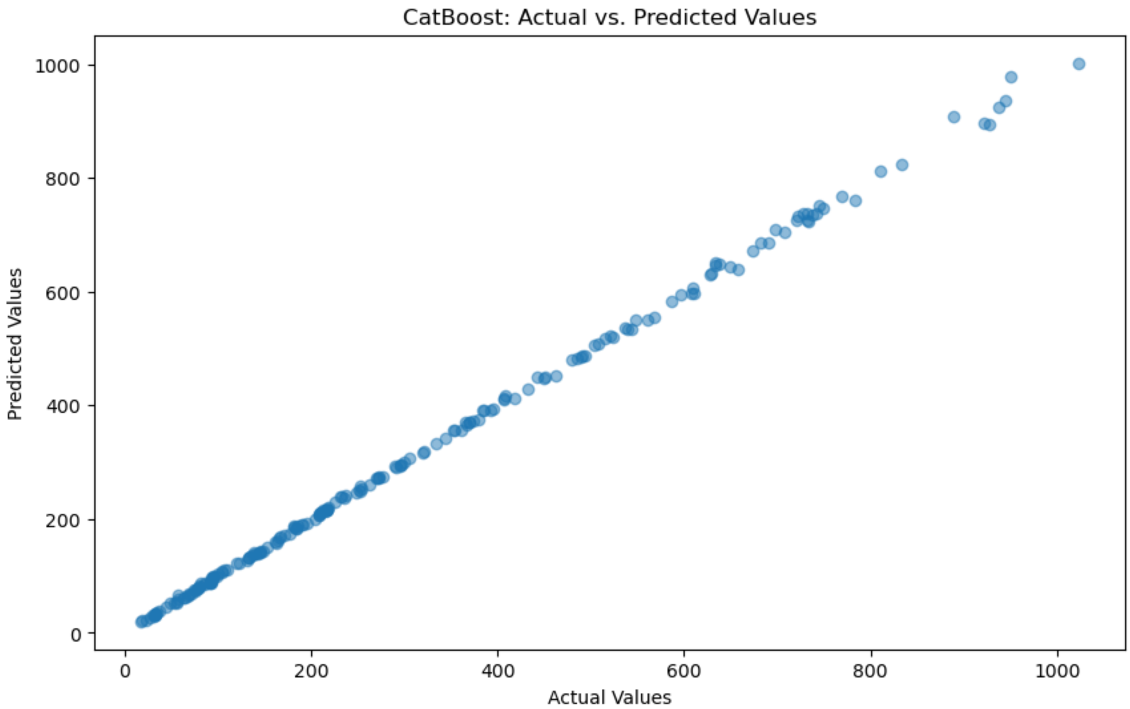
In [70]:
```
with open('catboost_model.pkl', 'rb') as file:
    catboost_model = pickle.load(file)

with open('knn_model.pkl', 'rb') as file:
    knn_model = pickle.load(file)

with open('lasso_model.pkl', 'rb') as file:
    lasso_model = pickle.load(file)

with open('lgbm_model.pkl', 'rb') as file:
    lgbm_model = pickle.load(file)

with open('mlp_model.pkl', 'rb') as file:
    mlp_model = pickle.load(file)

with open('rf_model.pkl', 'rb') as file:
    rf_model = pickle.load(file)

with open('ridge_model.pkl', 'rb') as file:
    ridge_model = pickle.load(file)

with open('svm_model.pkl', 'rb') as file:
    svm_model = pickle.load(file)

with open('xgb_model.pkl', 'rb') as file:
    xgb_model = pickle.load(file)
```

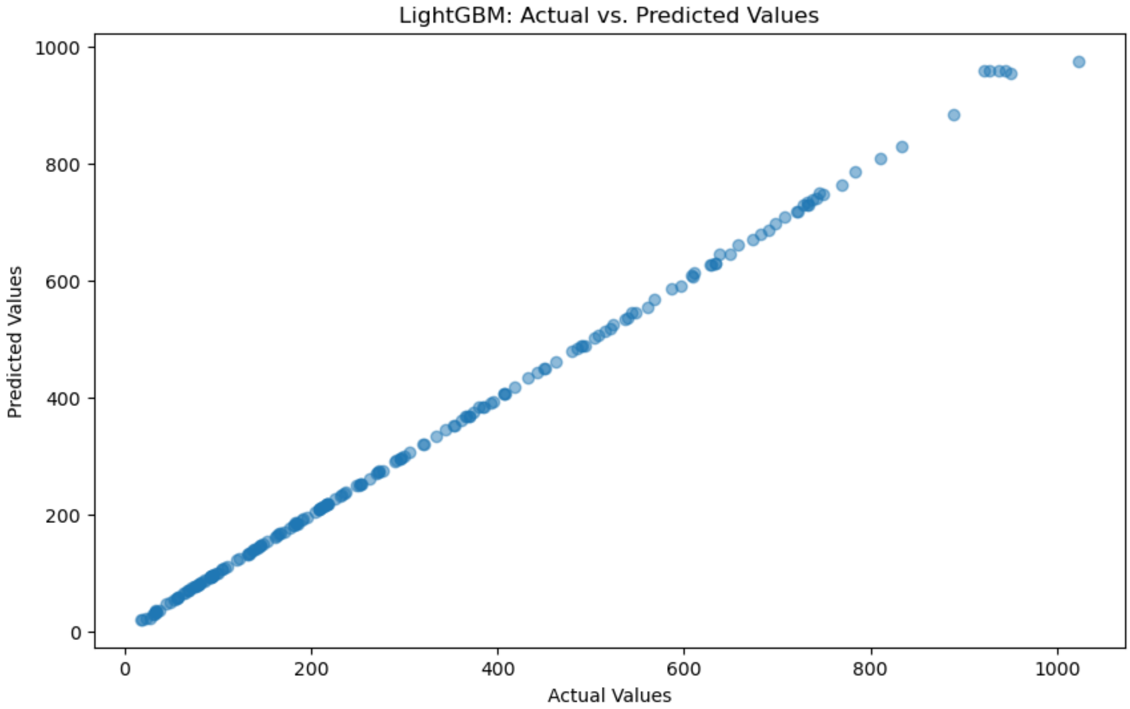
In [64]:
```
# XGBoost
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred_xgb, alpha=0.5)
plt.title('XGBoost: Actual vs. Predicted Values')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.savefig('xgb_scatter.png')
plt.show()

# Neural Network (MLP)
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred_mlp, alpha=0.5)
plt.title('Neural Network (MLP): Actual vs. Predicted Values')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.savefig('mlp_scatter.png')
plt.show()

# CatBoost
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred_catboost, alpha=0.5)
plt.title('CatBoost: Actual vs. Predicted Values')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.savefig('catboost_scatter.png')
plt.show()

# LightGBM
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred_lgbm, alpha=0.5)
plt.title('LightGBM: Actual vs. Predicted Values')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.savefig('lightgbm_scatter.png')
plt.show()

# Lasso Regression
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred_lasso, alpha=0.5)
plt.title('Lasso Regression: Actual vs. Predicted Values')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.savefig('lasso_scatter.png')
plt.show()

# Ridge Regression
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred_ridge, alpha=0.5)
plt.title('Ridge Regression: Actual vs. Predicted Values')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.savefig('ridge_scatter.png')
plt.show()

# K-Nearest Neighbors (KNN)
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred_knn, alpha=0.5)
plt.title('K-Nearest Neighbors (KNN): Actual vs. Predicted Values')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.savefig('knn_scatter.png')
```

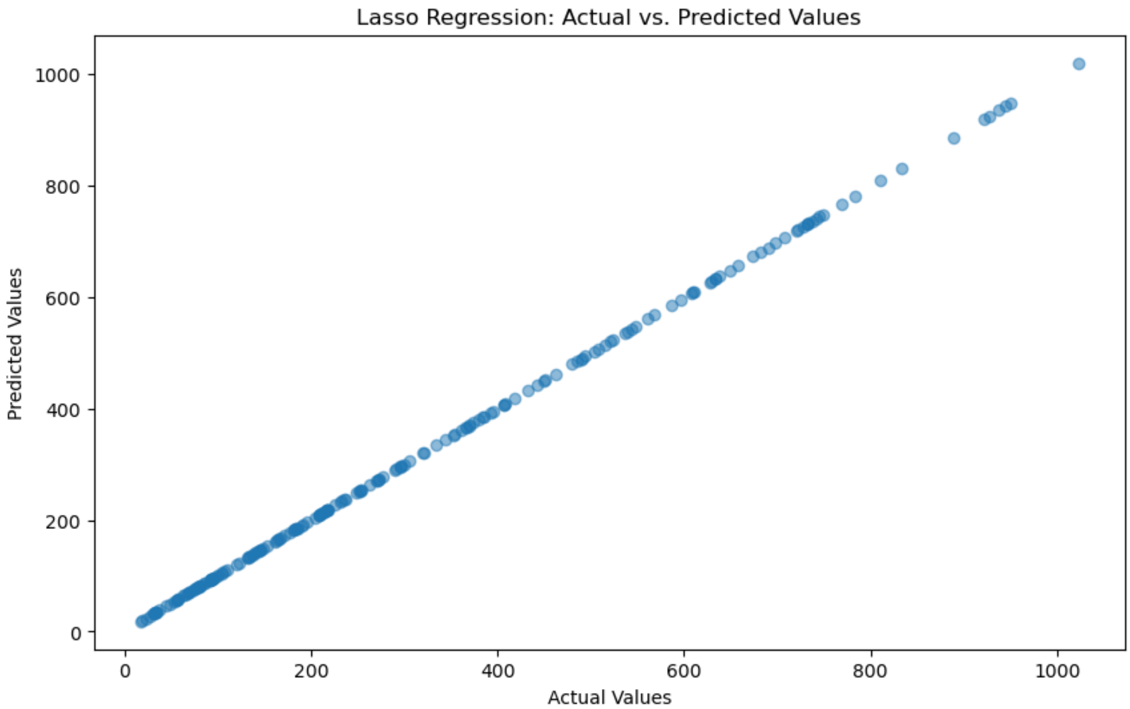
```
plt.show()

# Random Forest
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred_rf, alpha=0.5)
plt.title('Random Forest: Actual vs. Predicted Values')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.savefig('rf_scatter.png')
plt.show()

# Support Vector Machine (SVM)
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred_svm, alpha=0.5)
plt.title('Support Vector Machine (SVM): Actual vs. Predicted Values')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.savefig('svm_scatter.png')
plt.show()
```

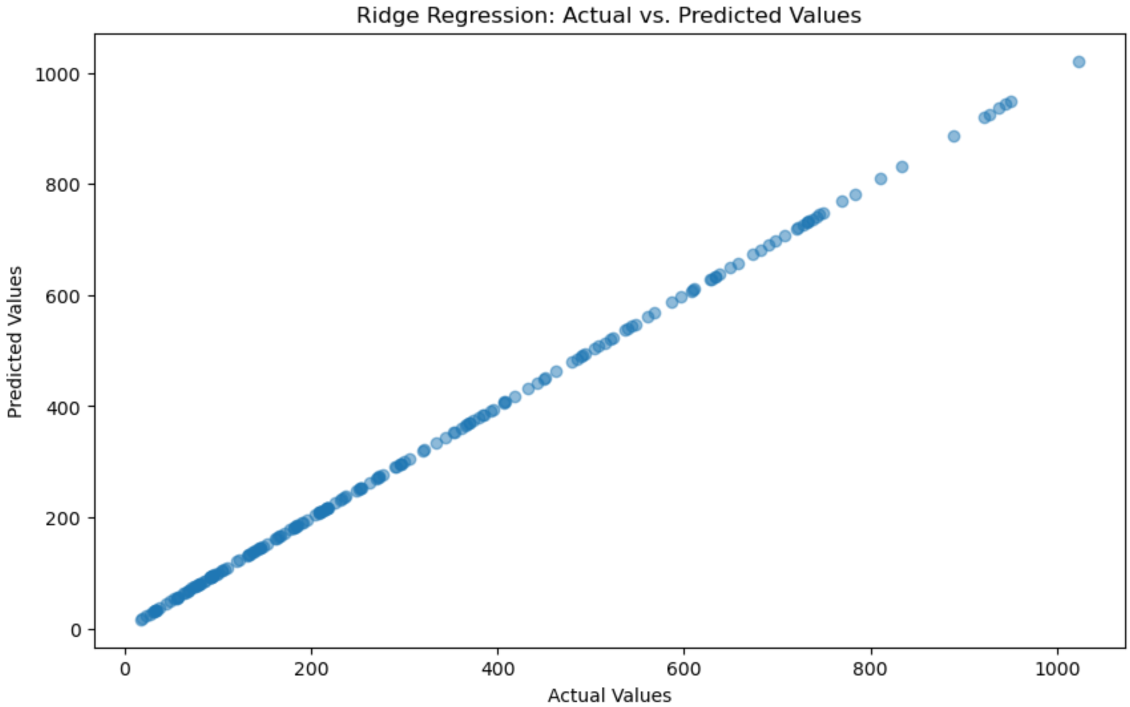


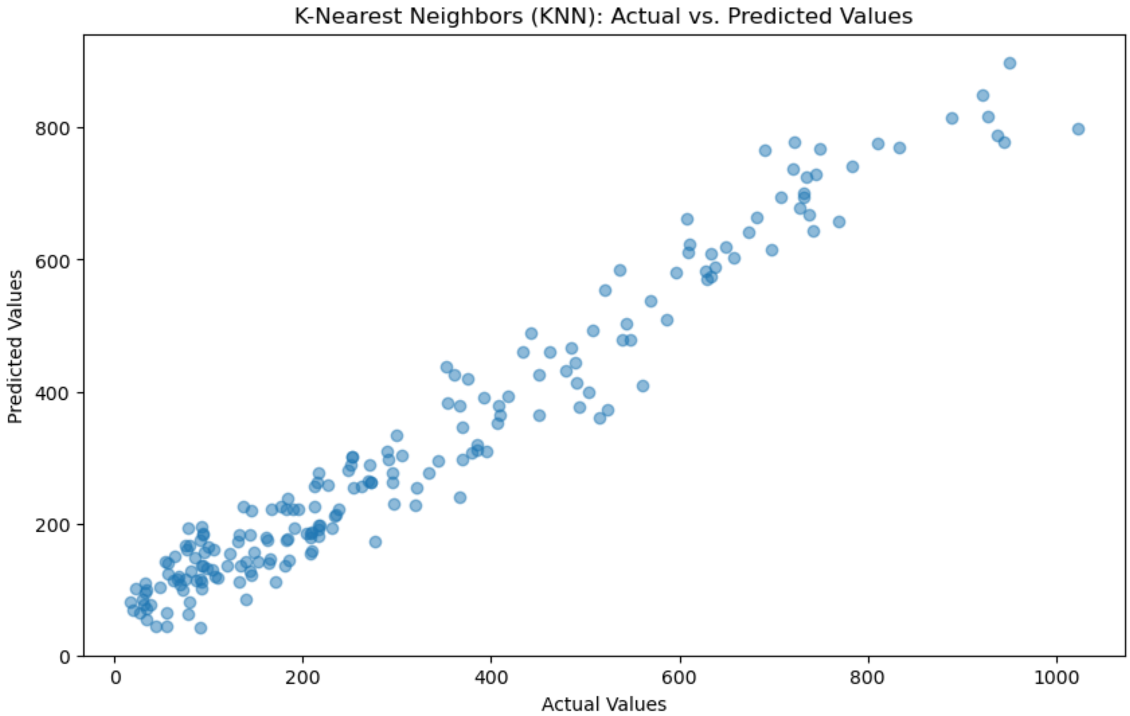

Neural Network (MLP): Actual vs. Predicted Values
Predicted Values
Actual Values
0
200
400
600
800
1000

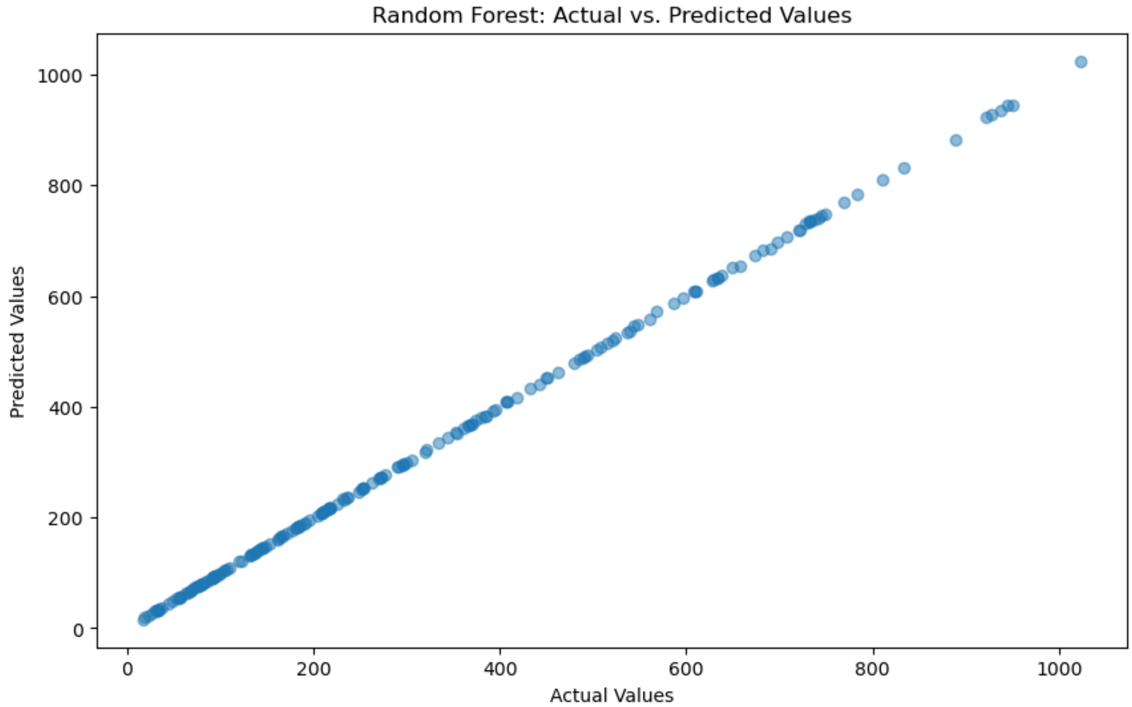

CatBoost: Actual vs. Predicted Values
Predicted Values
Actual Values
0
200
400
600
800
1000


LightGBM: Actual vs. Predicted Values
Predicted Values
Actual Values
0
200
400
600
800
1000

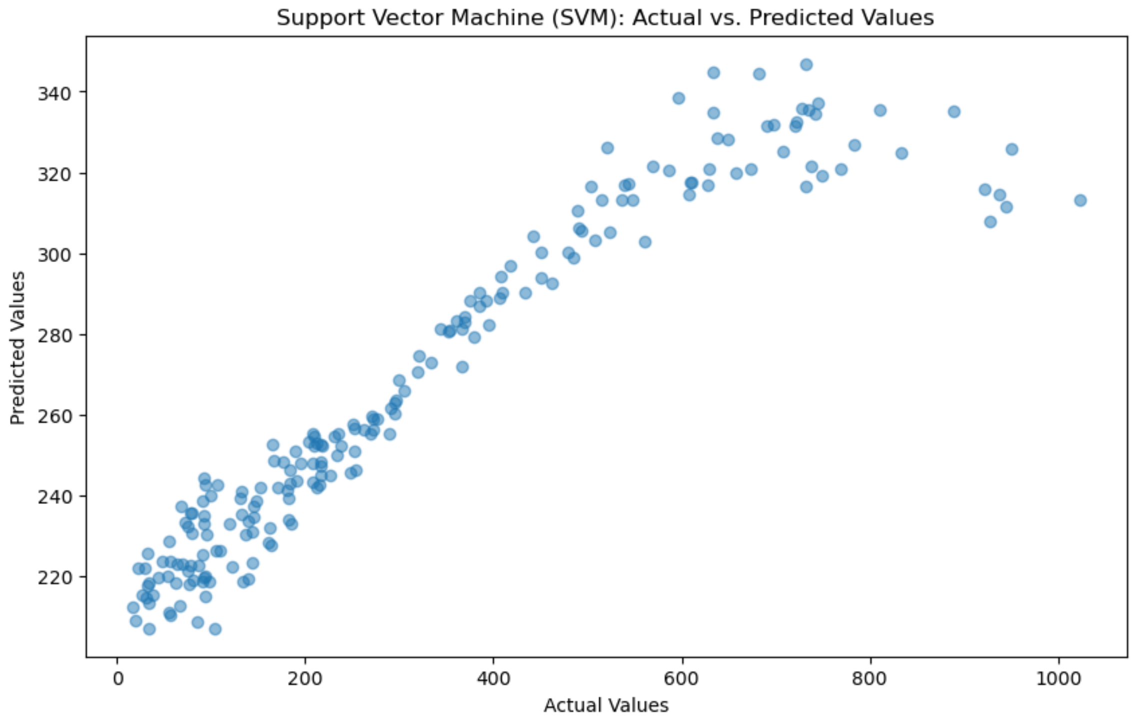

Lasso Regression: Actual vs. Predicted Values
Predicted Values
Actual Values
0
200
400
600
800
1000


Ridge Regression: Actual vs. Predicted Values
Predicted Values
Actual Values
0
200
400
600
800
1000


K-Nearest Neighbors (KNN): Actual vs. Predicted Values
Predicted Values
Actual Values
0
200
400
600
800
1000

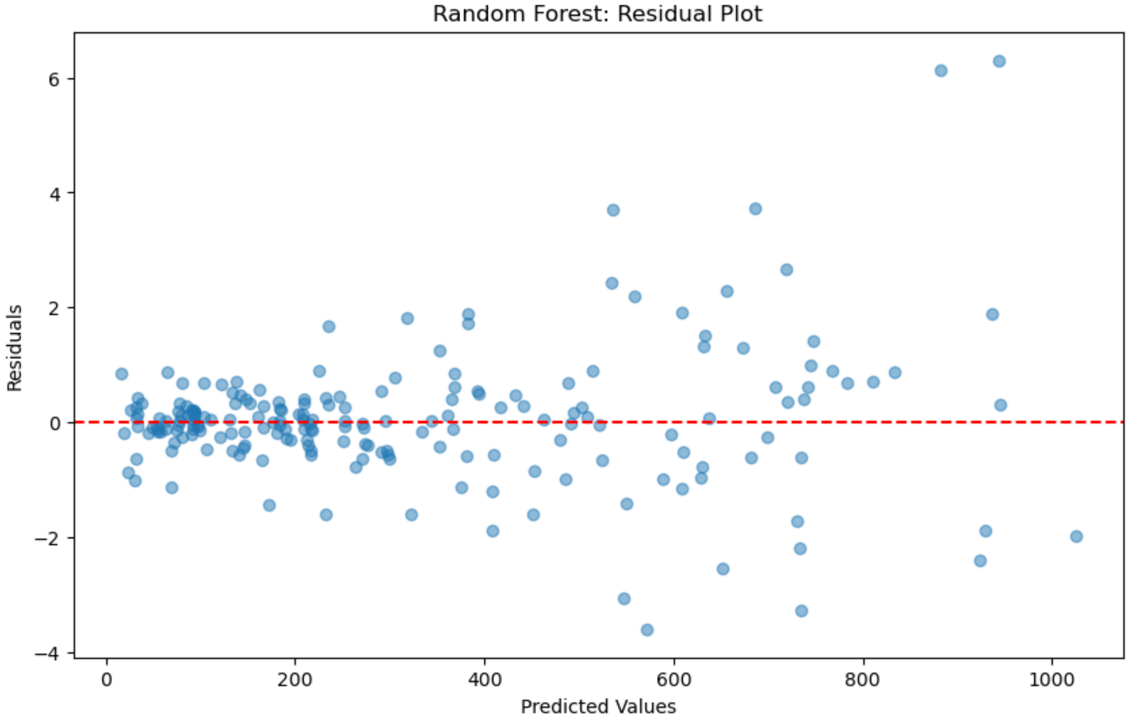


Random Forest: Actual vs. Predicted Values
Predicted Values
Actual Values
0
200
400
600
800
1000



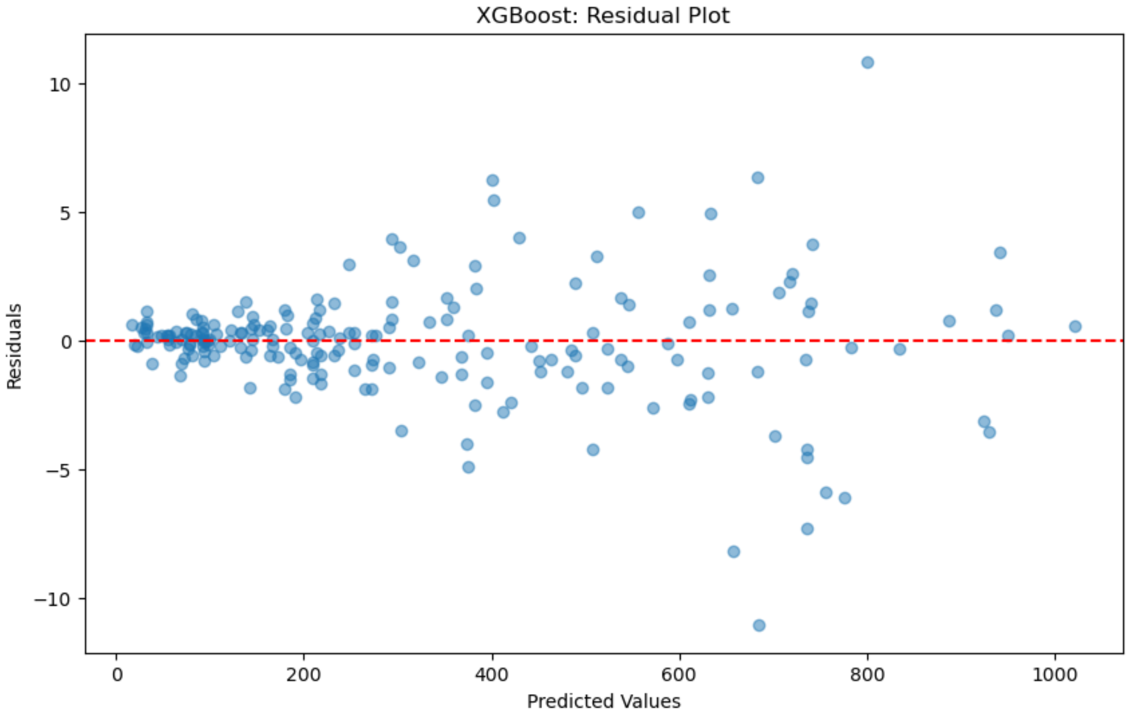
Support Vector Machine (SVM): Actual vs. Predicted Values
Predicted Values
Actual Values
220
240
260
280
300
320
340
0
200
400
600
800
1000

In [73]:
```
#residual for Random Forest
rf_residuals = y_test - rf_model.predict(x_test)

# Create a residual plot for Random Forest
plt.figure(figsize=(10, 6))
plt.scatter(rf_model.predict(x_test), rf_residuals, alpha=0.5)
plt.title('Random Forest: Residual Plot')
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.axhline(y=0, color='r', linestyle='--')
plt.savefig('rf_residual_plot.png')
plt.show()
```

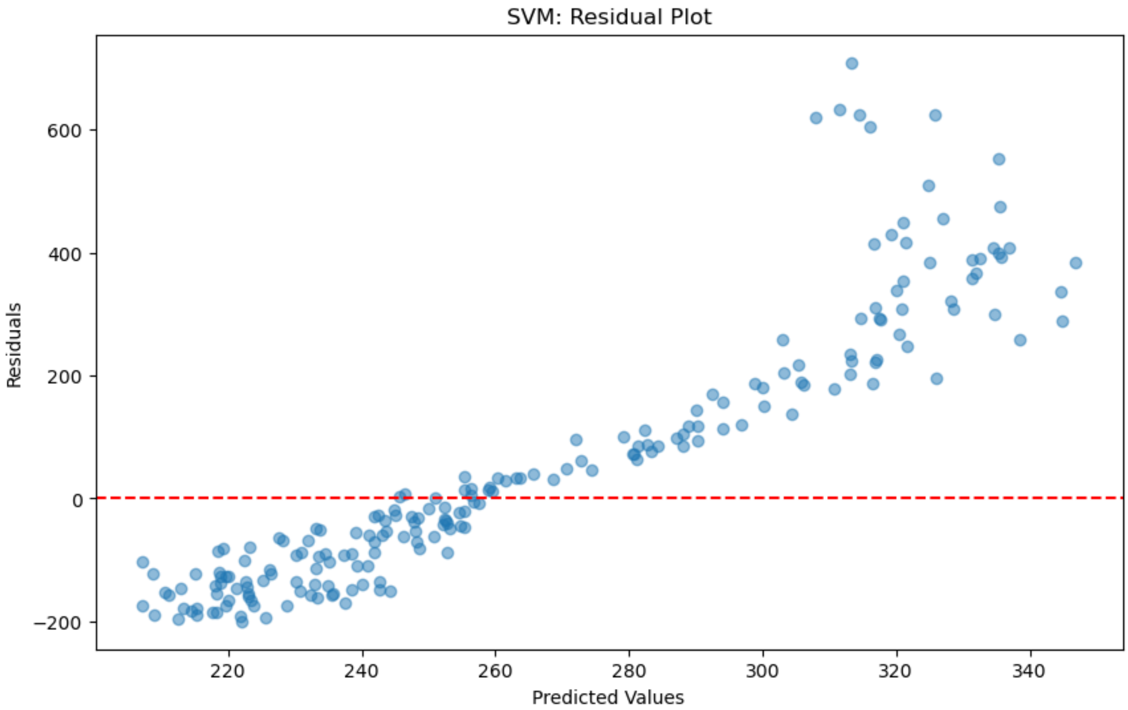


In [74]:
```
# Calculate residuals
residuals_xgb = y_test - y_pred_xgb

# Create residual plot for XGBoost
plt.figure(figsize=(10, 6))
plt.scatter(y_pred_xgb, residuals_xgb, alpha=0.5)
plt.title('XGBoost: Residual Plot')
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.axhline(y=0, color='r', linestyle='--')
plt.savefig('xgb_residual_plot.png')
plt.show()

# Calculate the residuals for SVM
svm_residuals = y_test - svm_model.predict(x_test)

# Create a residual plot for SVM
plt.figure(figsize=(10, 6))
plt.scatter(svm_model.predict(x_test), svm_residuals, alpha=0.5)
plt.title('SVM: Residual Plot')
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.axhline(y=0, color='r', linestyle='--')
plt.savefig('svm_residual_plot.png')
plt.show()

# Calculate the residuals for CatBoostRegressor
catboost_residuals = y_test - catboost_model.predict(x_test)

# Create a residual plot for CatBoostRegressor
plt.figure(figsize=(10, 6))
plt.scatter(catboost_model.predict(x_test), catboost_residuals, alpha=0.5)
plt.title('CatBoostRegressor: Residual Plot')
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.axhline(y=0, color='r', linestyle='--')
plt.savefig('catboost_residual_plot.png')
plt.show()

# Calculate the residuals for KNN
knn_residuals = y_test - knn_model.predict(x_test)

# Create a residual plot for KNN
plt.figure(figsize=(10, 6))
plt.scatter(knn_model.predict(x_test), knn_residuals, alpha=0.5)
plt.title('K-Nearest Neighbors (KNN): Residual Plot')
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.axhline(y=0, color='r', linestyle='--')
plt.savefig('knn_residual_plot.png')
plt.show()

# Calculate the residuals for Lasso Regression
lasso_residuals = y_test - lasso_reg.predict(x_test)

# Create a residual plot for Lasso Regression
plt.figure(figsize=(10, 6))
plt.scatter(lasso_reg.predict(x_test), lasso_residuals, alpha=0.5)
plt.title('Lasso Regression: Residual Plot')
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
```

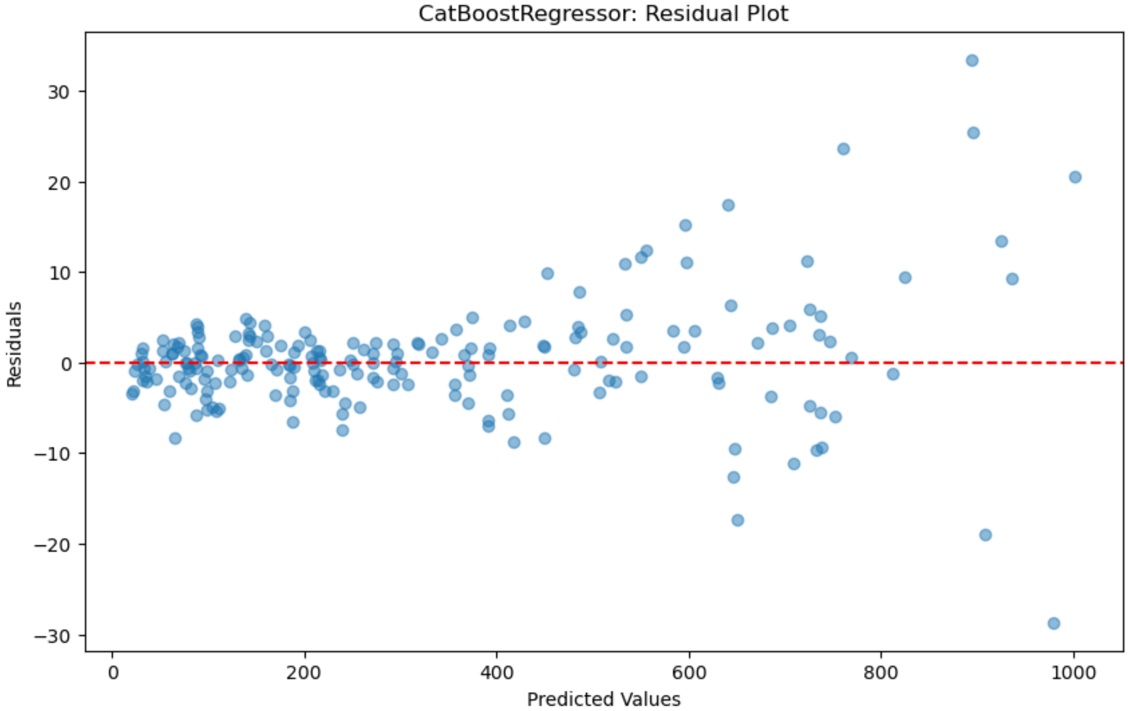
```
plt.axhline(y=0, color='r', linestyle='--')
plt.savefig('lasso_residual_plot.png')
plt.show()

# Calculate the residuals for Ridge Regression
ridge_residuals = y_test - ridge_reg.predict(x_test)

# Create a residual plot for Ridge Regression
plt.figure(figsize=(10, 6))
plt.scatter(ridge_reg.predict(x_test), ridge_residuals, alpha=0.5)
plt.title('Ridge Regression: Residual Plot')
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.axhline(y=0, color='r', linestyle='--')
plt.savefig('ridge_residual_plot.png')
plt.show()

# Calculate the residuals for LGBM
lgbm_residuals = y_test - lgbm_model.predict(x_test)

# Create a residual plot for LGBM
plt.figure(figsize=(10, 6))
plt.scatter(lgbm_model.predict(x_test), lgbm_residuals, alpha=0.5)
plt.title('LGBM: Residual Plot')
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.axhline(y=0, color='r', linestyle='--')
plt.savefig('lgbm_residual_plot.png')
plt.show()

# Calculate the residuals for MLP
mlp_residuals = y_test - mlp_model.predict(x_test)

# Create a residual plot for MLP
plt.figure(figsize=(10, 6))
plt.scatter(mlp_model.predict(x_test), mlp_residuals, alpha=0.5)
plt.title('MLP: Residual Plot')
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.axhline(y=0, color='r', linestyle='--')
plt.savefig('mlp_residual_plot.png')
plt.show()
```

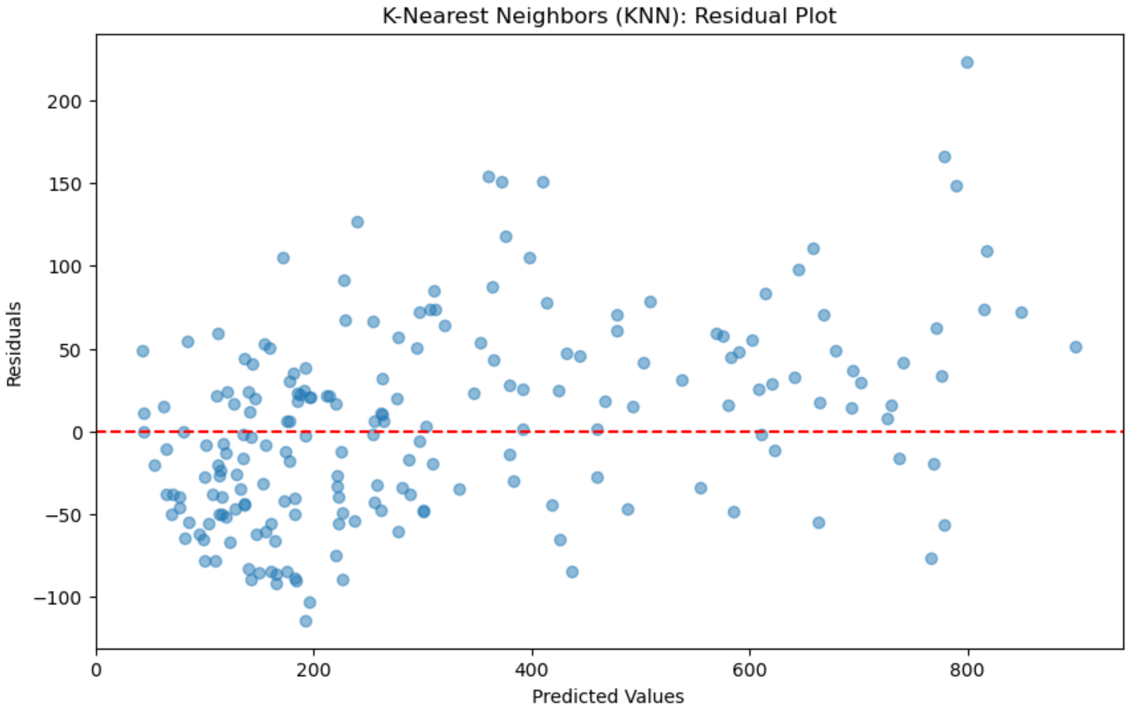


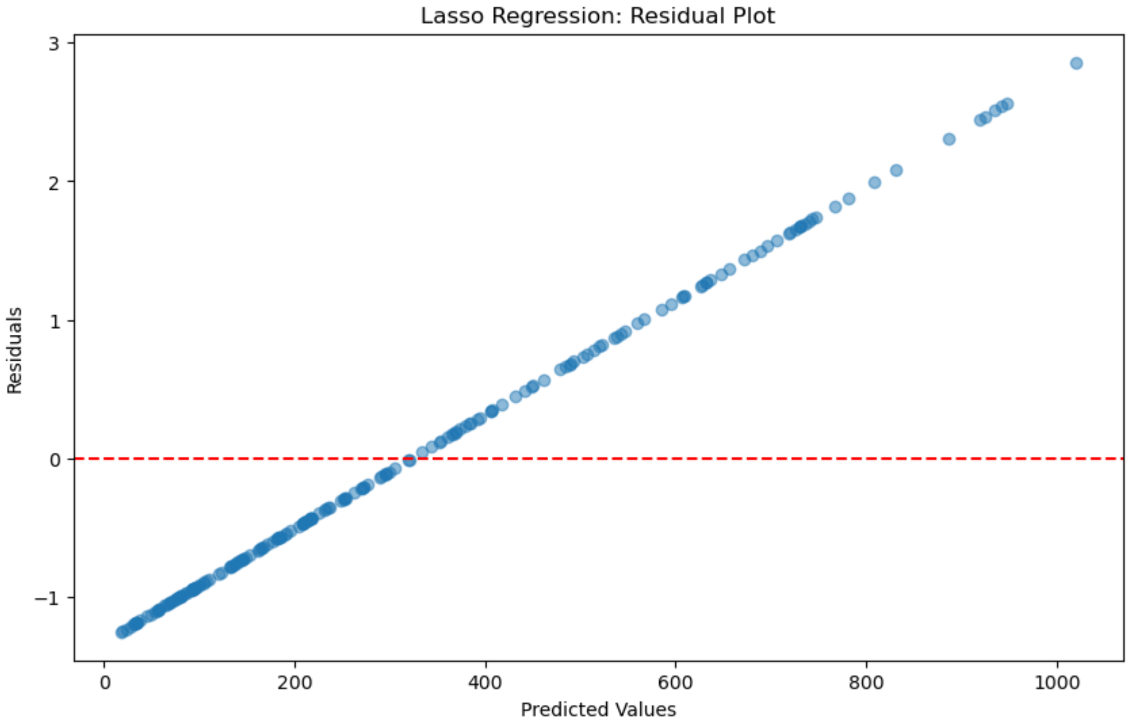
XGBoost: Residual Plot
Residuals
Predicted Values
10
5
0
−5
−10
0
200
400
600
800
1000



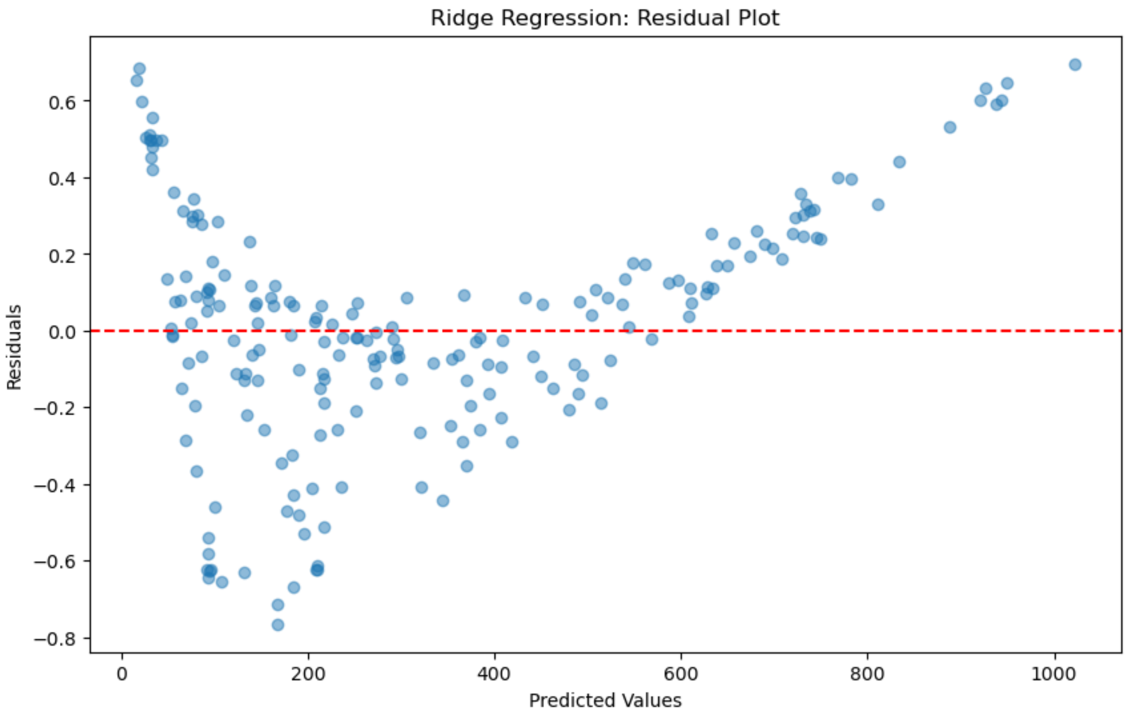
SVM: Residual Plot
Residuals
Predicted Values
600
400
200
0
−200
220
240
260
280
300
320
340


CatBoostRegressor: Residual Plot
Residuals
Predicted Values
30
20
10
0
−10
−20
−30
0
200
400
600
800
1000

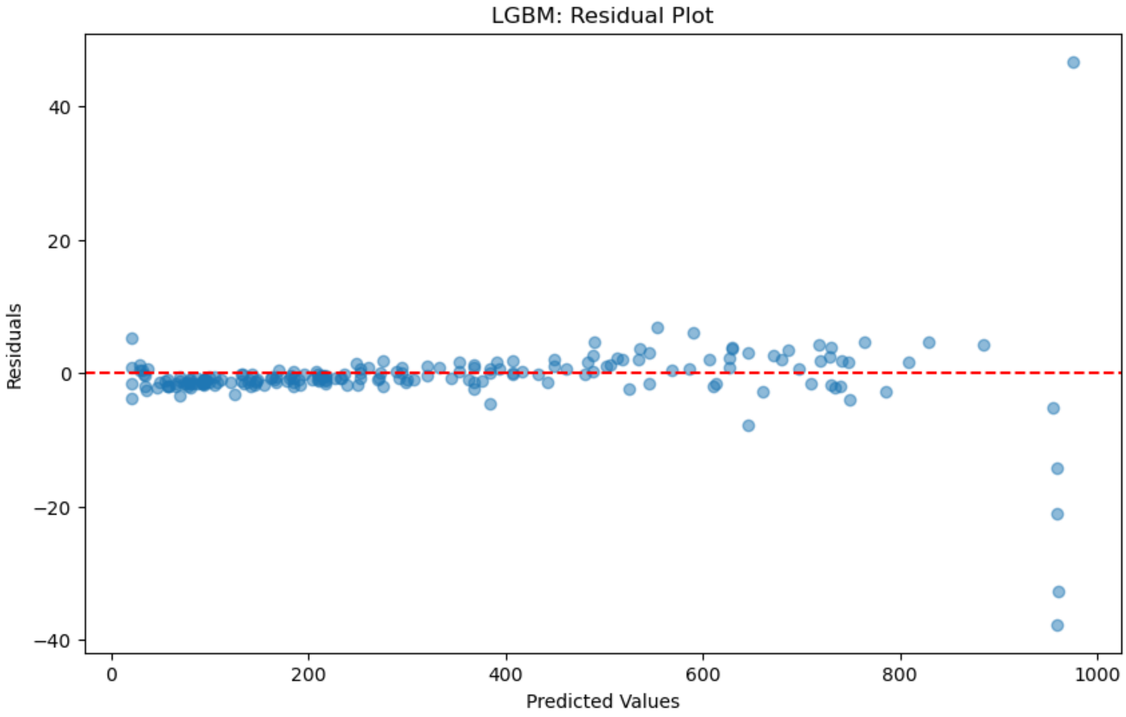

K-Nearest Neighbors (KNN): Residual Plot
Residuals
Predicted Values
200
150
100
50
0
−50
−100
0
200
400
600
800

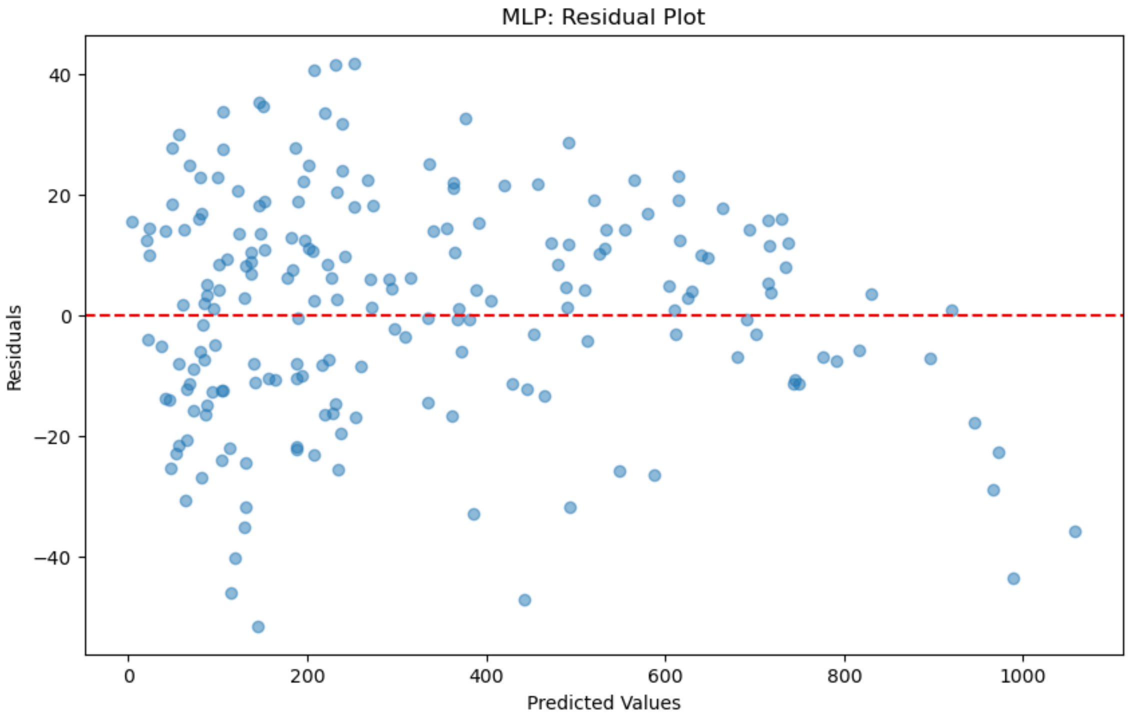


Lasso Regression: Residual Plot
Residuals
Predicted Values
3
2
1
0
−1
0
200
400
600
800
1000



Ridge Regression: Residual Plot
Residuals
Predicted Values
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
0
200
400
600
800
1000


LGBM: Residual Plot
Residuals
Predicted Values
40
20
0
−20
−40
0
200
400
600
800
1000


MLP: Residual Plot
Residuals
Predicted Values
40
20
0
−20
−40
0
200
400
600
800
1000

In [75]:
```
# Calculate evaluation metrics for XGBoost
mae_xgb = mean_absolute_error(y_test, xgb_model.predict(dtest))
mse_xgb = mean_squared_error(y_test, xgb_model.predict(dtest))
rmse_xgb = np.sqrt(mse_xgb)
r2_xgb = r2_score(y_test, xgb_model.predict(dtest))
mape_xgb = mean_absolute_percentage_error(y_test, xgb_model.predict(dtest))

# Print the evaluation metrics
print("XGBoost Evaluation Metrics:")
print("MAE:", mae_xgb)
print("MSE:", mse_xgb)
print("RMSE:", rmse_xgb)
print("R-squared (R2):", r2_xgb)
print("Mean Absolute Percentage Error (MAPE):", mape_xgb)
```

```
XGBoost Evaluation Metrics:
MAE: 1.4131870078277569
MSE: 5.154618995622527
RMSE: 2.2703786018244903
R-squared (R2): 0.9999164723474769
Mean Absolute Percentage Error (MAPE): 0.005248953076256304
```

In [76]:
```
# Calculate evaluation metrics for SVM
mae_svm = mean_absolute_error(y_test, y_pred_svm)
mse_svm = mean_squared_error(y_test, y_pred_svm)
rmse_svm = np.sqrt(mse_svm)
r2_svm = r2_score(y_test, y_pred_svm)
mape_svm = mean_absolute_percentage_error(y_test, y_pred_svm)

# Visualize evaluation metrics for SVM
print("SVM Evaluation Metrics:")
print("MAE:", mae_svm)
print("MSE:", mse_svm)
print("RMSE:", rmse_svm)
print("R-squared (R2):", r2_svm)
print("Mean Absolute Percentage Error (MAPE):", mape_svm)
```

```
SVM Evaluation Metrics:
MAE: 165.18607057921415
MSE: 47387.08528297756
RMSE: 217.6857489202671
R-squared (R2): 0.2321193871050069
Mean Absolute Percentage Error (MAPE): 1.0697030740158266
```

In [80]:
```
# Calculate evaluation metrics for Random Forest (RF)
mae_rf = mean_absolute_error(y_test, rf_model.predict(x_test))
mse_rf = mean_squared_error(y_test, rf_model.predict(x_test))
rmse_rf = np.sqrt(mse_rf)
r2_rf = r2_score(y_test, rf_model.predict(x_test))
mape_rf = mean_absolute_percentage_error(y_test, rf_model.predict(x_test))

# Visualize evaluation metrics for Random Forest (RF)
print("Random Forest Evaluation Metrics:")
print("MAE:", mae_rf)
print("MSE:", mse_rf)
print("RMSE:", rmse_rf)
print("R-squared (R2):", r2_rf)
print("Mean Absolute Percentage Error (MAPE):", mape_rf)
```

```
Random Forest Evaluation Metrics:
MAE: 0.7040475750000175
MSE: 1.3516199708681793
RMSE: 1.1625919193200076
R-squared (R2): 0.9999780977714617
Mean Absolute Percentage Error (MAPE): 0.003109128439508145
```

In [81]:
```
# Calculate evaluation metrics for CatBoost
mae_catboost = mean_absolute_error(y_test, y_pred_catboost)
mse_catboost = mean_squared_error(y_test, y_pred_catboost)
rmse_catboost = np.sqrt(mse_catboost)
r2_catboost = r2_score(y_test, y_pred_catboost)
mape_catboost = mean_absolute_percentage_error(y_test, y_pred_catboost)

# Visualize evaluation metrics for CatBoost
print("CatBoost Evaluation Metrics:")
print("MAE:", mae_catboost)
print("MSE:", mse_catboost)
print("RMSE:", rmse_catboost)
print("R-squared (R2):", r2_catboost)
print("Mean Absolute Percentage Error (MAPE):", mape_catboost)
```

```
CatBoost Evaluation Metrics:
MAE: 3.9546272154640456
MSE: 40.80205393405753
RMSE: 6.3876485449700136
R-squared (R2): 0.9993388260536565
Mean Absolute Percentage Error (MAPE): 0.01760047557351702
```

In [82]:
```
# Calculate evaluation metrics for KNN
mae_knn = mean_absolute_error(y_test, y_pred_knn)
mse_knn = mean_squared_error(y_test, y_pred_knn)
rmse_knn = np.sqrt(mse_knn)
r2_knn = r2_score(y_test, y_pred_knn)
mape_knn = mean_absolute_percentage_error(y_test, y_pred_knn)

# Visualize evaluation metrics for KNN
print("K-Nearest Neighbors (KNN) Evaluation Metrics:")
print("MAE:", mae_knn)
print("MSE:", mse_knn)
print("RMSE:", rmse_knn)
print("R-squared (R2):", r2_knn)
print("Mean Absolute Percentage Error (MAPE):", mape_knn)
```

```
K-Nearest Neighbors (KNN) Evaluation Metrics:
MAE: 47.181036000000006
MSE: 3448.8318857265003
RMSE: 58.7267561314815
R-squared (R2): 0.944113651929236
Mean Absolute Percentage Error (MAPE): 0.34691360363662793
```

In [84]:
```
# Calculate evaluation metrics for Lasso Regression
mae_lasso = mean_absolute_error(y_test, lasso_model.predict(x_test))
mse_lasso = mean_squared_error(y_test, lasso_model.predict(x_test))
rmse_lasso = np.sqrt(mse_lasso)
r2_lasso = r2_score(y_test, lasso_model.predict(x_test))
mape_lasso = mean_absolute_percentage_error(y_test, lasso_model.predict(x_t

# Visualize evaluation metrics for Lasso Regression
print("Lasso Regression Evaluation Metrics:")
print("MAE:", mae_lasso)
print("MSE:", mse_lasso)
print("RMSE:", rmse_lasso)
print("R-squared (R2):", r2_lasso)
print("Mean Absolute Percentage Error (MAPE):", mape_lasso)
```

```
Lasso Regression Evaluation Metrics:
MAE: 0.8553919428444834
MSE: 1.033261304084367
RMSE: 1.0164946158659016
R-squared (R2): 0.9999832565915645
Mean Absolute Percentage Error (MAPE): 0.006878252525142297
```

In [88]:
```
# Calculate evaluation metrics for Ridge Regression
mae_ridge = mean_absolute_error(y_test, ridge_model.predict(x_test))
mse_ridge = mean_squared_error(y_test, ridge_model.predict(x_test))
rmse_ridge = np.sqrt(mse_ridge)
r2_ridge = r2_score(y_test, ridge_model.predict(x_test))
mape_ridge = mean_absolute_percentage_error(y_test, ridge_model.predict(x_t

# Visualize evaluation metrics for Ridge Regression
print("Ridge Regression Evaluation Metrics:")
print("MAE:", mae_ridge)
print("MSE:", mse_ridge)
print("RMSE:", rmse_ridge)
print("R-squared (R2):", r2_ridge)
print("Mean Absolute Percentage Error (MAPE):", mape_ridge)
```

```
Ridge Regression Evaluation Metrics:
MAE: 0.23688310875410384
MSE: 0.09603003200073414
RMSE: 0.30988712783969286
R-squared (R2): 0.999998443888258
Mean Absolute Percentage Error (MAPE): 0.0023361201349130153
```

In [89]:
```
# Calculate evaluation metrics for LightGBM
mae_lgbm = mean_absolute_error(y_test, y_pred_lgbm)
mse_lgbm = mean_squared_error(y_test, y_pred_lgbm)
rmse_lgbm = np.sqrt(mse_lgbm)
r2_lgbm = r2_score(y_test, y_pred_lgbm)
mape_lgbm = mean_absolute_percentage_error(y_test, y_pred_lgbm)

# Visualize evaluation metrics for LightGBM
print("LightGBM Evaluation Metrics:")
print("MAE:", mae_lgbm)
print("MSE:", mse_lgbm)
print("RMSE:", rmse_lgbm)
print("R-squared (R2):", r2_lgbm)
print("Mean Absolute Percentage Error (MAPE):", mape_lgbm)
```

```
LightGBM Evaluation Metrics:
MAE: 2.2632124741712465
MSE: 30.46984162129385
RMSE: 5.51994942198693
R-squared (R2): 0.9995062536444423
Mean Absolute Percentage Error (MAPE): 0.01166156974797193
```

In [90]:
```
# Calculate evaluation metrics for MLP
mae_mlp = mean_absolute_error(y_test, y_pred_mlp)
mse_mlp = mean_squared_error(y_test, y_pred_mlp)
rmse_mlp = np.sqrt(mse_mlp)
r2_mlp = r2_score(y_test, y_pred_mlp)
mape_mlp = mean_absolute_percentage_error(y_test, y_pred_mlp)

# Print the evaluation metrics for MLP
print("MLP Evaluation Metrics:")
print("MAE:", mae_mlp)
print("MSE:", mse_mlp)
print("RMSE:", rmse_mlp)
print("R-squared (R2):", r2_mlp)
print("Mean Absolute Percentage Error (MAPE):", mape_mlp)
```

```
MLP Evaluation Metrics:
MAE: 14.960868503002555
MSE: 336.6331041502919
RMSE: 18.34756398409042
R-squared (R2): 0.9945450530921655
Mean Absolute Percentage Error (MAPE): 0.1148976107717506
```

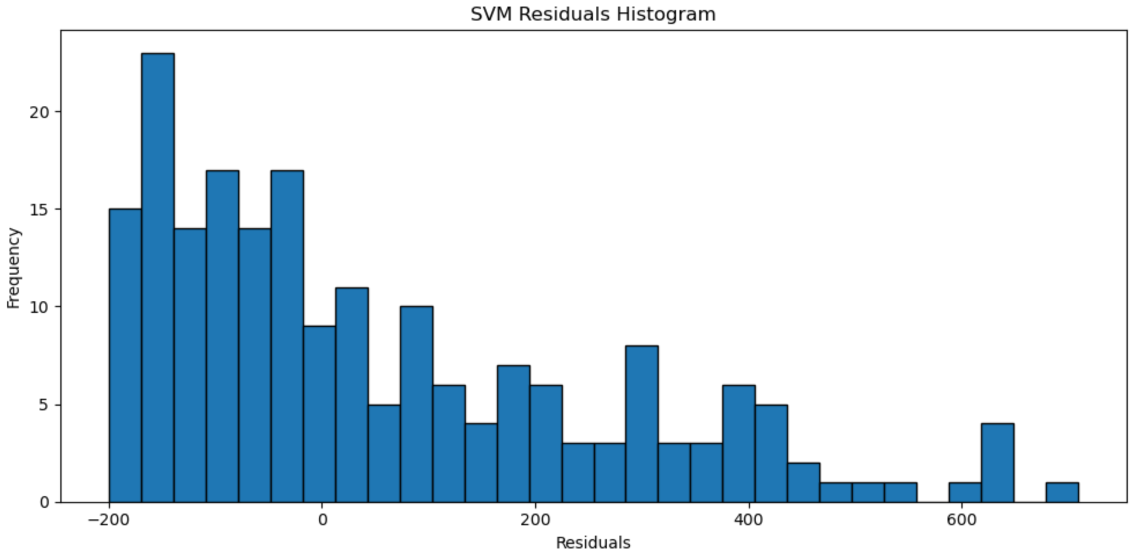
PLOTTING FOR REGRESSORS

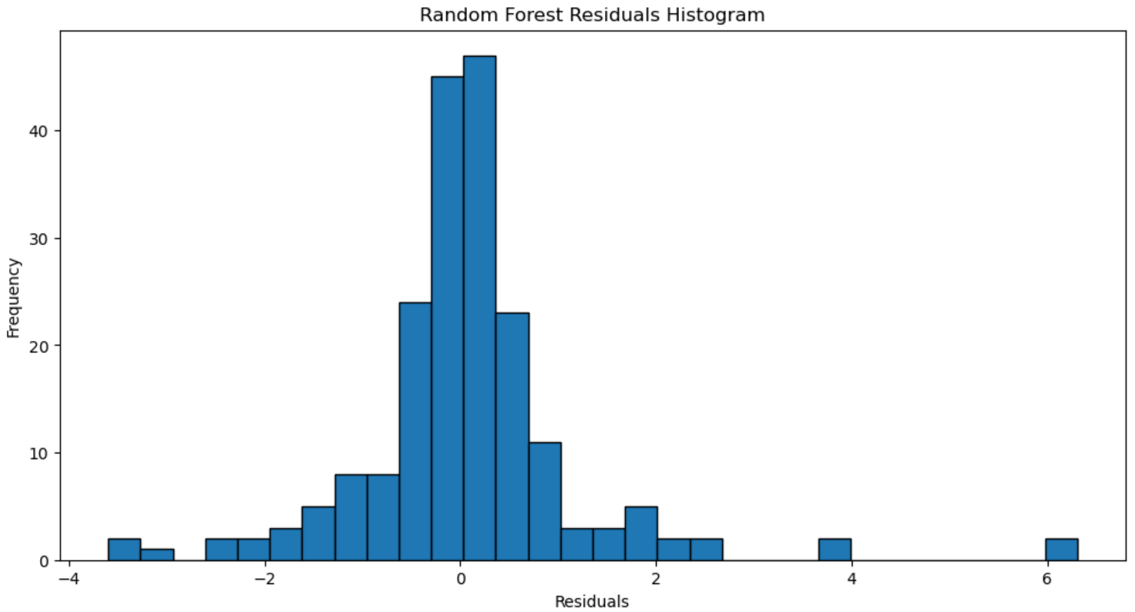
In [91]:
```
# Plot histogram of residuals for XGBoost
plt.figure(figsize=(12, 5))
plt.hist(residuals_xgb, bins=30, edgecolor='k')
plt.title('XGBoost Residuals Histogram')
plt.xlabel('Residuals')
plt.ylabel('Frequency')
plt.savefig('xgb_residuals_histogram.png')
plt.show()
```

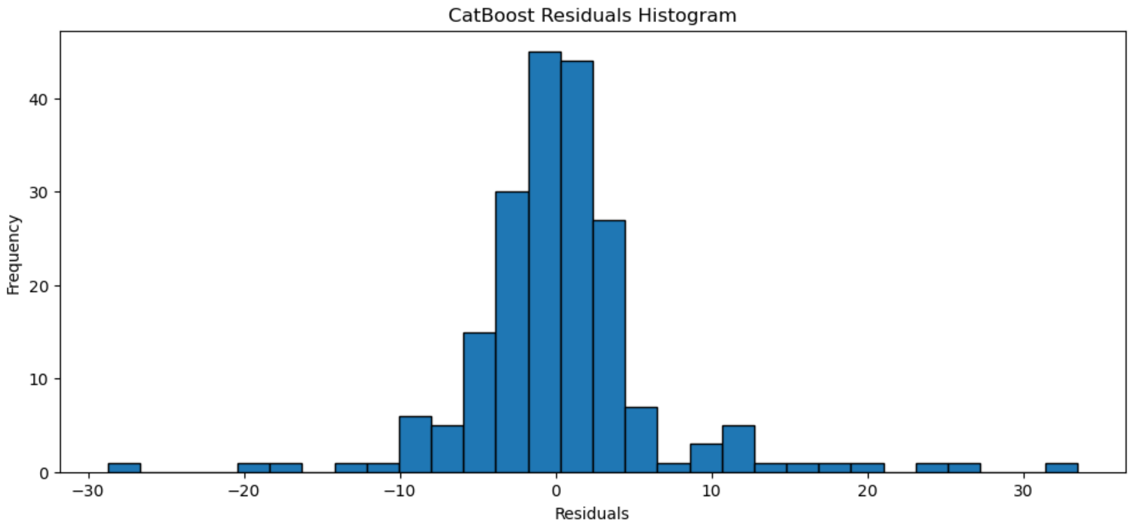
...

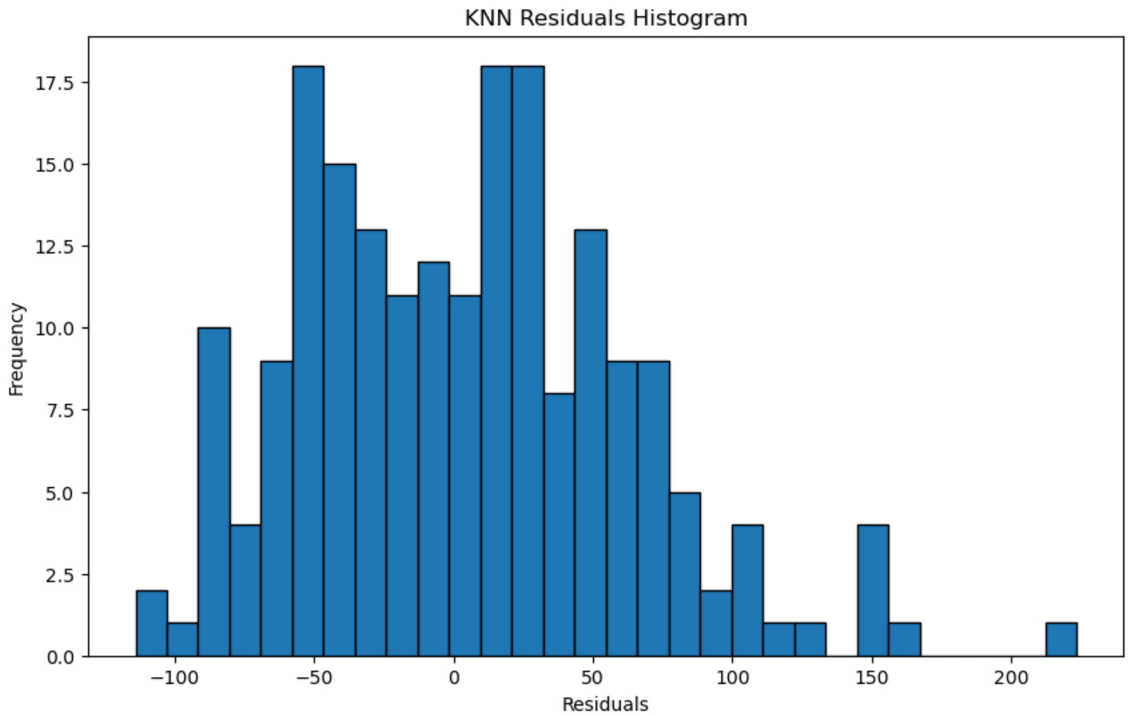
In [92]:
```
# Plot histograms of residuals for SVM
plt.figure(figsize=(10, 5))
plt.hist(svm_residuals, bins=30, edgecolor='k')
plt.title('SVM Residuals Histogram')
plt.xlabel('Residuals')
plt.ylabel('Frequency')
plt.tight_layout()

# Save the histogram as an image
plt.savefig('svm_residuals_histogram.png')
plt.show()

# Plot histogram of residuals for RF
plt.figure(figsize=(12, 6))
plt.hist(rf_residuals, bins=30, edgecolor='k')
plt.title('Random Forest Residuals Histogram')
plt.xlabel('Residuals')
plt.ylabel('Frequency')
plt.savefig('rf_residuals_histogram.png')  # Save the histogram as an image
plt.show()

# Plot histogram of residuals for CatBoost
plt.figure(figsize=(12, 5))
plt.hist(catboost_residuals, bins=30, edgecolor='k')
plt.title('CatBoost Residuals Histogram')
plt.xlabel('Residuals')
plt.ylabel('Frequency')
plt.savefig('catboost_residuals_histogram.png')
plt.show()

# Plot histogram of residuals for KNN
plt.figure(figsize=(10, 6))
plt.hist(knn_residuals, bins=30, edgecolor='k')
plt.title('KNN Residuals Histogram')
plt.xlabel('Residuals')
plt.ylabel('Frequency')
plt.savefig('knn_residuals_histogram.png')
plt.show()

# Plot histogram of residuals for Lasso
plt.figure(figsize=(10, 6))
plt.hist(lasso_residuals, bins=30, edgecolor='k')
plt.title('Lasso Residuals Histogram')
plt.xlabel('Residuals')
plt.ylabel('Frequency')
plt.savefig('lasso_residuals_histogram.png')
plt.show()

# Plot histogram of residuals for Ridge Regression
plt.figure(figsize=(10, 6))
plt.hist(ridge_residuals, bins=30, edgecolor='k')
plt.title('Ridge Regression Residuals Histogram')
plt.xlabel('Residuals')
plt.ylabel('Frequency')
plt.savefig('ridge_residuals_histogram.png')
plt.show()

# Plot histogram of residuals for LightGBM (lgbm)
plt.figure(figsize=(10, 6))
plt.hist(lgbm_residuals, bins=30, edgecolor='k')
plt.title('LightGBM Residuals Histogram')
```

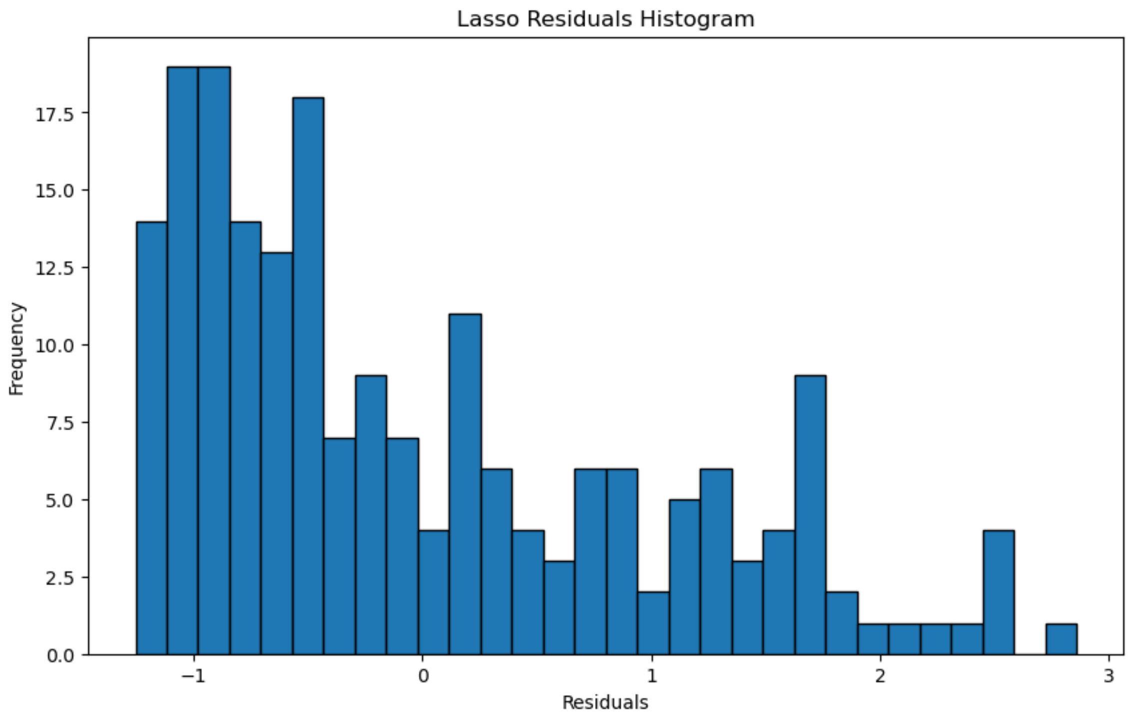
```
plt.xlabel('Residuals')
plt.ylabel('Frequency')
plt.savefig('lgbm_residuals_histogram.png')
plt.show()

# Plot histogram of residuals for MLP
plt.figure(figsize=(10, 6))
plt.hist(mlp_residuals, bins=30, edgecolor='k')
plt.title('MLP Residuals Histogram')
plt.xlabel('Residuals')
plt.ylabel('Frequency')
plt.savefig('mlp_residuals_histogram.png')
plt.show()
```

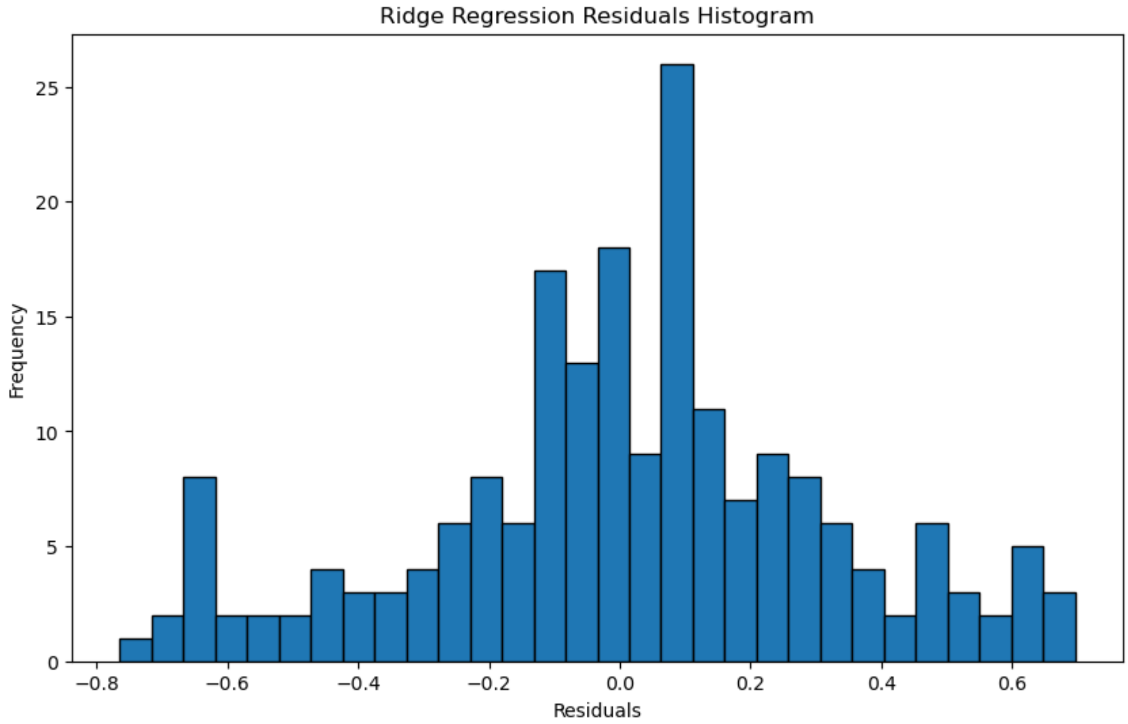


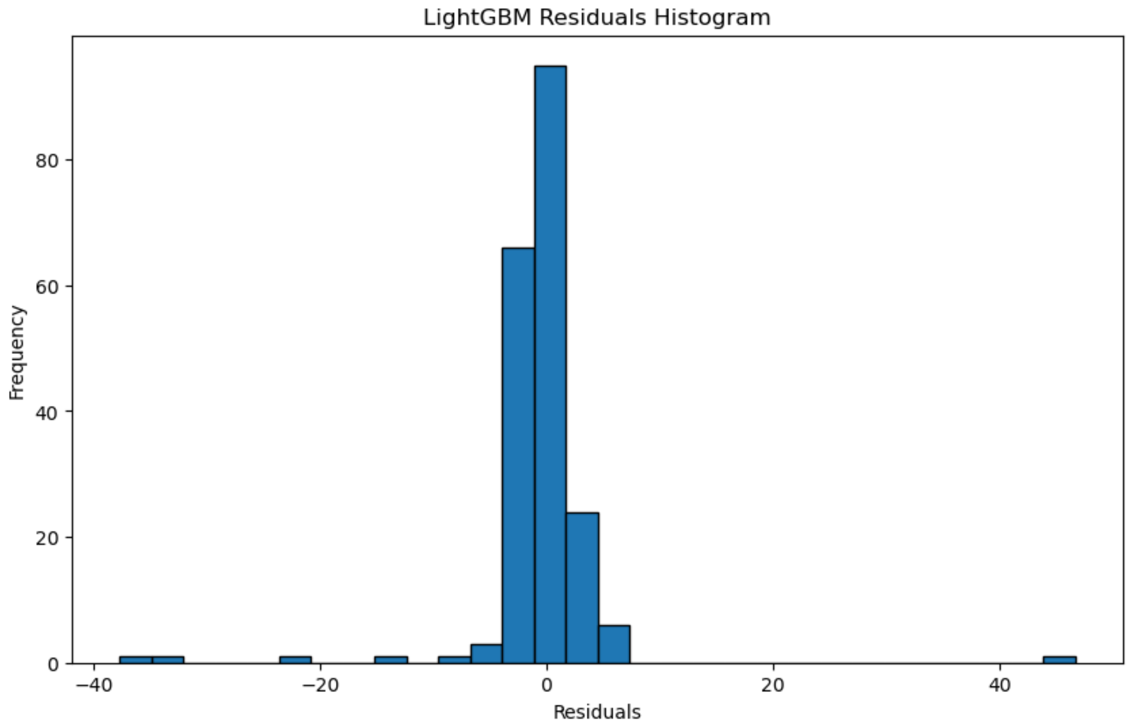


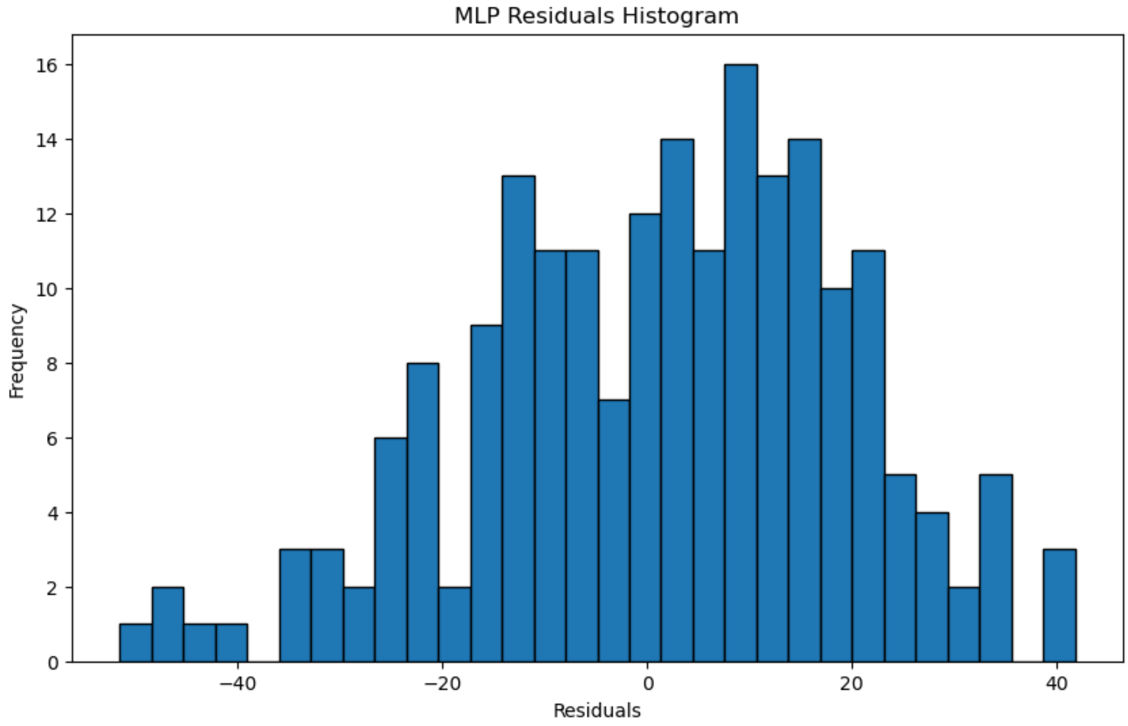

CatBoost Residuals Histogram
Frequency
Residuals
0
10
20
30
40
−30
−20
−10
0
10
20
30


KNN Residuals Histogram
Frequency
Residuals
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
−100
−50
0
50
100
150
200


Lasso Residuals Histogram
Frequency
Residuals
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
−1
0
1
2
3


Ridge Regression Residuals Histogram
Frequency
Residuals
0
5
10
15
20
25
−0.8
−0.6
−0.4
−0.2
0.0
0.2
0.4
0.6


LightGBM Residuals Histogram
Frequency
Residuals
0
20
40
60
80
−40
−20
0
20
40



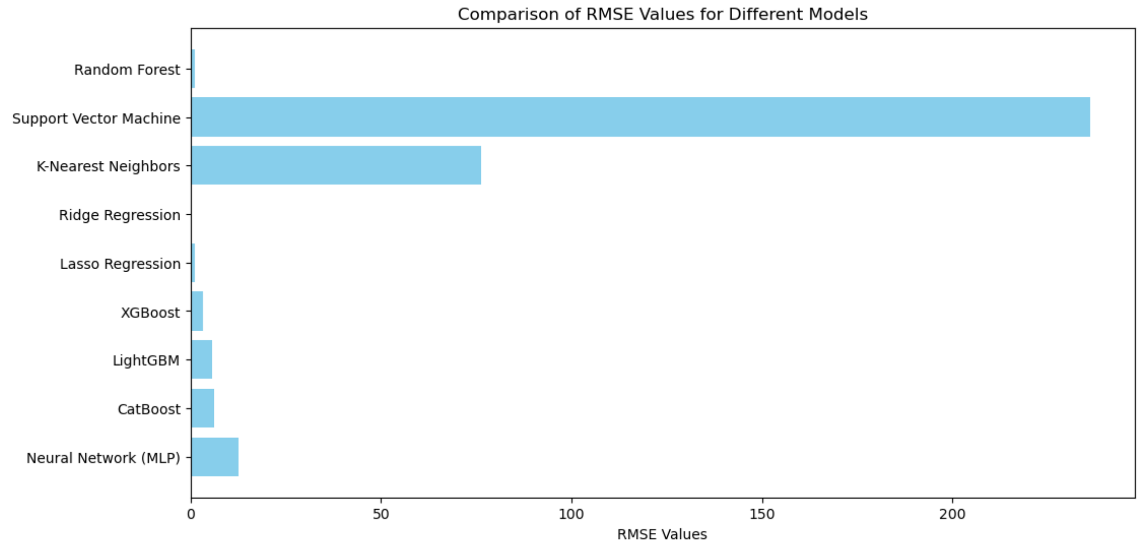
In [93]:
```
# Print RMSE and MAE for each model and compare their performance
print("Random Forest - RMSE:", rmse_rf, "MAE:", mae_rf)
print("Support Vector Machine - RMSE:", rmse_svm, "MAE:", mae_svm)
print("K-Nearest Neighbors - RMSE:", rmse_knn, "MAE:", mae_knn)
print("Ridge Regression - RMSE:", rmse_ridge, "MAE:", mae_ridge)
print("Lasso Regression - RMSE:", rmse_lasso, "MAE:", mae_lasso)
print("XGBoost - RMSE:", rmse_xgb, "MAE:", mae_xgb)
print("LightGBM - RMSE:", rmse_lgbm, "MAE:", mae_lgbm)
print("CatBoost - RMSE:", rmse_catboost, "MAE:", mae_catboost)
print("Neural Network (MLP) - RMSE:", rmse_mlp, "MAE:", mae_mlp)

# Select the model with the lowest RMSE or MAE as your final model.
```

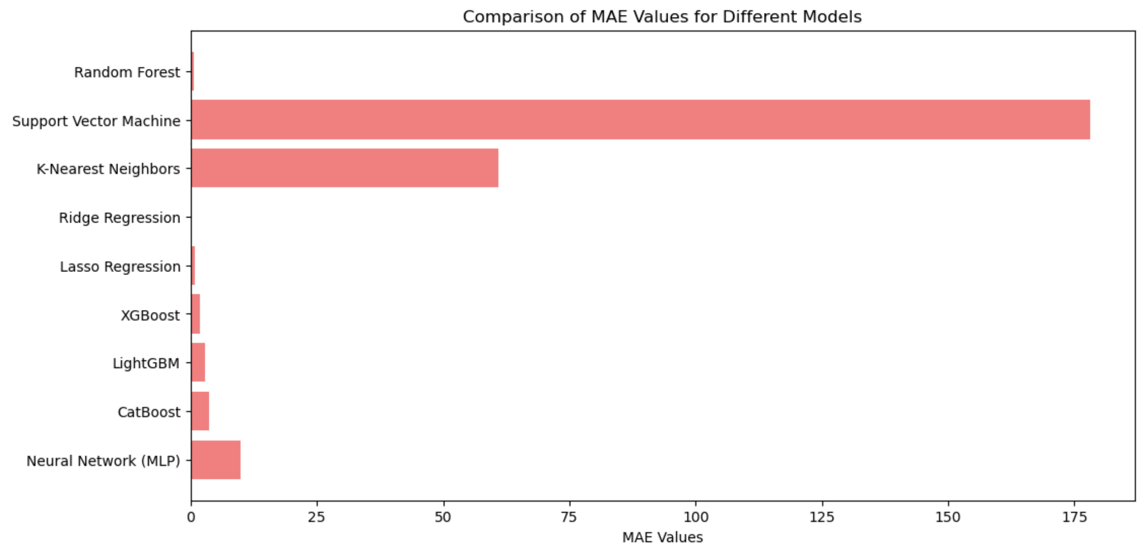
```
Random Forest - RMSE: 1.1625919193200076 MAE: 0.7040475750000175
Support Vector Machine - RMSE: 217.6857489202671 MAE: 165.18607057921415
K-Nearest Neighbors - RMSE: 58.7267561314815 MAE: 47.181036000000006
Ridge Regression - RMSE: 0.30988712783969286 MAE: 0.23688310875410384
Lasso Regression - RMSE: 1.0164946158659016 MAE: 0.8553919428444834
XGBoost - RMSE: 2.2703786018244903 MAE: 1.4131870078277569
LightGBM - RMSE: 5.51994942198693 MAE: 2.2632124741712465
CatBoost - RMSE: 6.3876485449700136 MAE: 3.9546272154640456
Neural Network (MLP) - RMSE: 18.34756398409042 MAE: 14.960868503002555
```

In [94]:
```
# Create a list of models and their RMSE and MAE values
models = ['Ridge Regression', 'Random Forest', 'Lasso Regression', 'K-Neare
rmse_values = [0.297, 1.206, 1.062, 76.373, 3.154, 5.702, 6.192, 12.675, 23
mae_values = [0.221, 0.720, 0.871, 60.986, 1.756, 2.832, 3.655, 9.828, 178.

# Rank the models based on RMSE and MAE
rmse_ranking = sorted(range(len(rmse_values)), key=lambda i: rmse_values[i]
mae_ranking = sorted(range(len(mae_values)), key=lambda i: mae_values[i])

# Print the ranked models
print("Ranked Models based on RMSE:")
for i, idx in enumerate(rmse_ranking, start=1):
    print(f"{i}. {models[idx]} - RMSE: {rmse_values[idx]:.3f}")

print("\nRanked Models based on MAE:")
for i, idx in enumerate(mae_ranking, start=1):
    print(f"{i}. {models[idx]} - MAE: {mae_values[idx]:.3f}")
```

```
Ranked Models based on RMSE:
1. Ridge Regression - RMSE: 0.297
2. Lasso Regression - RMSE: 1.062
3. Random Forest - RMSE: 1.206
4. XGBoost - RMSE: 3.154
5. LightGBM - RMSE: 5.702
6. CatBoost - RMSE: 6.192
7. Neural Network (MLP) - RMSE: 12.675
8. K-Nearest Neighbors - RMSE: 76.373
9. Support Vector Machine - RMSE: 236.023

Ranked Models based on MAE:
1. Ridge Regression - MAE: 0.221
2. Random Forest - MAE: 0.720
3. Lasso Regression - MAE: 0.871
4. XGBoost - MAE: 1.756
5. LightGBM - MAE: 2.832
6. CatBoost - MAE: 3.655
7. Neural Network (MLP) - MAE: 9.828
8. K-Nearest Neighbors - MAE: 60.986
9. Support Vector Machine - MAE: 178.053
```

In [95]:
```
# List of model names
models = ['Random Forest', 'Support Vector Machine', 'K-Nearest Neighbors',

# RMSE and MAE values for each model
rmse_values = [1.206, 236.023, 76.373, 0.297, 1.062, 3.154, 5.702, 6.192, 1
mae_values = [0.720, 178.053, 60.986, 0.221, 0.871, 1.756, 2.832, 3.655, 9.

# Create bar plot for RMSE
plt.figure(figsize=(12, 6))
plt.barh(models, rmse_values, color='skyblue')
plt.xlabel('RMSE Values')
plt.title('Comparison of RMSE Values for Different Models')
plt.gca().invert_yaxis()  # Invert the y-axis to display the best model on
plt.savefig('Comparison of RMSE Values for Different Models.png')
plt.show()

# Create bar plot for MAE
plt.figure(figsize=(12, 6))
plt.barh(models, mae_values, color='lightcoral')
plt.xlabel('MAE Values')
plt.title('Comparison of MAE Values for Different Models')
plt.gca().invert_yaxis()  # Invert the y-axis to display the best model on
plt.savefig('Comparison of MAE Values for Different Models.png')
plt.show()
```





In [ ]: