DDS-8555 Assignment 2: Build Regression Models

Student: Abigail Albury-Bloom
Course: DDS-8555 Predictive Analysis

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import Lasso
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error

import statsmodels.api as sm
from statsmodels.stats.stattools import durbin_watson
from statsmodels.stats.outliers_influence import variance_inflation_factor
from math import sqrt

pd.set_option("display.max_columns", 100)
```

Conceptual Question 3 (ISLR Python)

(a) Correct answer: (iii) High-school graduates earn more on average than college graduates when GPA > 3.5.
(b) Predicted salary for a college graduate (GPA=4.0, IQ=110) = **$137,100**.
(c) Interaction term (0.01) is not negligible; statistical significance, not magnitude, determines importance.

Applied Question 10 (ISLR Python)

(a) Regression model:
Sales = 13.04 − 0.054(Price) − 0.022(UrbanYes) + 1.20(USYes)

(b) Price ↓ Sales (significant), Urban not significant, US ↑ Sales (significant).
(c) Qualitative variables: UrbanYes and USYes coded as 1/0.
(d) Only Price and US significant ($p < 0.05$).
(e) Reduced model: Sales = 13.04 − 0.054(Price) + 1.20(USYes).
(f) R^2 and residuals stable; adjusted R^2 slightly higher in reduced model.

(g) 95% CIs exclude zero for Price and US.
(h) Residuals and leverage plots show no severe violations.

Kaggle Regression with Abalone Dataset

This section uses the Kaggle *Regression with an Abalone Dataset* data to build two predictive models:

1. **Lasso Regression** (regularized linear model)
2. **Random Forest Regressor** (nonlinear ensemble model)

Both models use an 80/20 train-validation split for RMSE evaluation, then are retrained on the full training set to create Kaggle submission files.

In [2]:
```
# Load Kaggle Abalone data (ensure train.csv and test.csv are in the same fo
train = pd.read_csv("train.csv")
test = pd.read_csv("test.csv")

print("Original train columns:", train.columns.tolist())

# Drop duplicate Whole weight columns if present (keep the first 'Whole weig
dup_train = [c for c in train.columns if c.startswith("Whole weight.")]
dup_test = [c for c in test.columns if c.startswith("Whole weight.")]
if dup_train:
    print("Dropping duplicate train columns:", dup_train)
    train = train.drop(columns=dup_train)
if dup_test:
    print("Dropping duplicate test columns:", dup_test)
    test = test.drop(columns=dup_test, errors="ignore")

print("Cleaned train columns:", train.columns.tolist())
```

```
Original train columns: ['id', 'Sex', 'Length', 'Diameter', 'Height', 'Whole
weight', 'Whole weight.1', 'Whole weight.2', 'Shell weight', 'Rings']
Dropping duplicate train columns: ['Whole weight.1', 'Whole weight.2']
Dropping duplicate test columns: ['Whole weight.1', 'Whole weight.2']
Cleaned train columns: ['id', 'Sex', 'Length', 'Diameter', 'Height', 'Whole
weight', 'Shell weight', 'Rings']
```

In [3]:
```
# Set target and ID column robustly
if "Rings" not in train.columns:
    raise ValueError(f"'Rings' column not found. Available columns: {train.c

target = "Rings"
id_col = "id" if "id" in train.columns else None

# Define features
drop_cols = [target]
if id_col:
    drop_cols.append(id_col)

X = train.drop(columns=drop_cols)
```

```
y = train[target]

X_test = test.drop(columns=[id_col], errors="ignore") if id_col else test.cc

# Identify numeric and categorical predictors
numeric_features = X.select_dtypes(include=[np.number]).columns.tolist()
categorical_features = [c for c in X.columns if c not in numeric_features]

print("Numeric features:", numeric_features)
print("Categorical features:", categorical_features)
```

```
Numeric features: ['Length', 'Diameter', 'Height', 'Whole weight', 'Shell we
ight']
Categorical features: ['Sex']
```

In [4]:

```
# Preprocessing pipeline
numeric_transformer = StandardScaler()
categorical_transformer = OneHotEncoder(handle_unknown="ignore")

preprocessor = ColumnTransformer(
    transformers=[
        ("num", numeric_transformer, numeric_features),
        ("cat", categorical_transformer, categorical_features)
    ]
)

# Define models
lasso = Lasso(alpha=0.001, max_iter=10000, random_state=42)
rf = RandomForestRegressor(n_estimators=500, random_state=42, n_jobs=-1)

# Train-validation split
X_train, X_val, y_train, y_val = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Build pipelines
lasso_pipe = Pipeline([("preprocessor", preprocessor),
                       ("model", lasso)])

rf_pipe = Pipeline([("preprocessor", preprocessor),
                    ("model", rf)])

# Fit and evaluate Lasso
lasso_pipe.fit(X_train, y_train)
y_val_pred_lasso = lasso_pipe.predict(X_val)
rmse_lasso = sqrt(mean_squared_error(y_val, y_val_pred_lasso))

# Fit and evaluate Random Forest
rf_pipe.fit(X_train, y_train)
y_val_pred_rf = rf_pipe.predict(X_val)
rmse_rf = sqrt(mean_squared_error(y_val, y_val_pred_rf))

print(f"Lasso RMSE on validation set: {rmse_lasso:.5f}")
print(f"Random Forest RMSE on validation set: {rmse_rf:.5f}")
```

```
Lasso RMSE on validation set: 2.15658
Random Forest RMSE on validation set: 2.06350
```

In [5]:
```python
# Retrain on full data and create Kaggle submission files

# Fit on full training data
lasso_pipe.fit(X, y)
rf_pipe.fit(X, y)

# Predict on test set
test_pred_lasso = lasso_pipe.predict(X_test)
test_pred_rf = rf_pipe.predict(X_test)

# Build submission DataFrames
if id_col and id_col in test.columns:
    sub_lasso = pd.DataFrame({id_col: test[id_col], target: test_pred_lasso}
    sub_rf = pd.DataFrame({id_col: test[id_col], target: test_pred_rf})
else:
    sub_lasso = pd.DataFrame({"id": np.arange(len(test_pred_lasso)), target:
    sub_rf = pd.DataFrame({"id": np.arange(len(test_pred_rf)), target: test_

# Save CSVs
sub_lasso.to_csv("submission_lasso.csv", index=False)
sub_rf.to_csv("submission_rf.csv", index=False)

print("Saved: submission_lasso.csv and submission_rf.csv")
sub_lasso.head()
```

```
Saved: submission_lasso.csv and submission_rf.csv
```

Out[5]:

	id	Rings
0	90615	9.628930
1	90616	10.885368
2	90617	10.094418
3	90618	10.341754
4	90619	7.748328

Table 1. RMSE Comparison Between Lasso and Random Forest Models

Model	RMSE
Lasso Regression	2.1566
Random Forest	2.0635

The Random Forest model achieved the lowest RMSE, indicating slightly higher predictive accuracy compared to the Lasso model.

Top Predictors Identified by Random Forest

The Random Forest model ranked the following features as most important in predicting abalone age:

1. Shell weight
2. Height
3. Diameter
4. Whole weight
5. Length

These variables contribute most to model accuracy, reflecting the strong biological relationship between physical size and the number of rings (age).

Regression Assumption Checks (Abalone)

The following cells assess linear regression assumptions using an OLS model on the Abalone data:

- Linearity and homoscedasticity via residuals vs. fitted plot
- Normality via Q-Q plot
- Independence via Durbin–Watson statistic
- Multicollinearity via Variance Inflation Factor (VIF)

In [6]:
```
import re

# Start from the original feature matrix X and target y already defined abov

# 1. One-hot encode categorical variables
X_ols = pd.get_dummies(X.copy(), drop_first=True)

# 2. Remove any duplicate columns just in case
X_ols = X_ols.loc[:, ~X_ols.columns.duplicated()]

# 3. Sanitize column names so nothing weird/keyword-like breaks statsmodels
clean_cols = []
for c in X_ols.columns:
    # keep only letters, numbers, underscore
    c_clean = re.sub(r'[^0-9a-zA-Z_]+', '_', c)
    # avoid reserved name "weights"
    if c_clean.lower() == "weights":
        c_clean = "weight_var"
    clean_cols.append(c_clean)
X_ols.columns = clean_cols

# 4. Force everything to numeric float
X_ols = X_ols.apply(pd.to_numeric, errors="coerce").astype(float)

# 5. Drop any rows with NaNs created in the process, align y
valid_rows = ~X_ols.isna().any(axis=1)
X_ols = X_ols.loc[valid_rows, :]
y_ols = y.loc[valid_rows].astype(float)
```

```python
# 6. Add constant
X_ols_const = sm.add_constant(X_ols)

# 7. Fit OLS
ols_model = sm.OLS(y_ols, X_ols_const).fit()
print(ols_model.summary())
```

OLS Regression Results

Dep. Variable:	Rings	R-squared:	0.554
Model:	OLS	Adj. R-squared:	0.554
Method:	Least Squares	F-statistic:	1.607e+04
Date:	Mon, 10 Nov 2025	Prob (F-statistic):	0.00
Time:	18:39:27	Log-Likelihood:	-1.9673e+05
No. Observations:	90615	AIC:	3.935e+05
Df Residuals:	90607	BIC:	3.936e+05
Df Model:	7		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	4.5785	0.064	71.282	0.000	4.453	4.704
Length	-5.8170	0.425	-13.685	0.000	-6.650	-4.984
Diameter	8.9674	0.530	16.930	0.000	7.929	10.006
Height	25.1620	0.515	48.828	0.000	24.152	26.172
Whole_weight	-6.6120	0.066	-99.475	0.000	-6.742	-6.482
Shell_weight	29.4469	0.216	136.580	0.000	29.024	29.869
Sex_I	-0.8154	0.023	-36.202	0.000	-0.860	-0.771
Sex_M	-0.0624	0.018	-3.502	0.000	-0.097	-0.027

Omnibus:	32092.958	Durbin-Watson:	1.994
Prob(Omnibus):	0.000	Jarque-Bera (JB):	206148.403
Skew:	1.560	Prob(JB):	0.00
Kurtosis:	9.698	Cond. No.	148.

Notes:

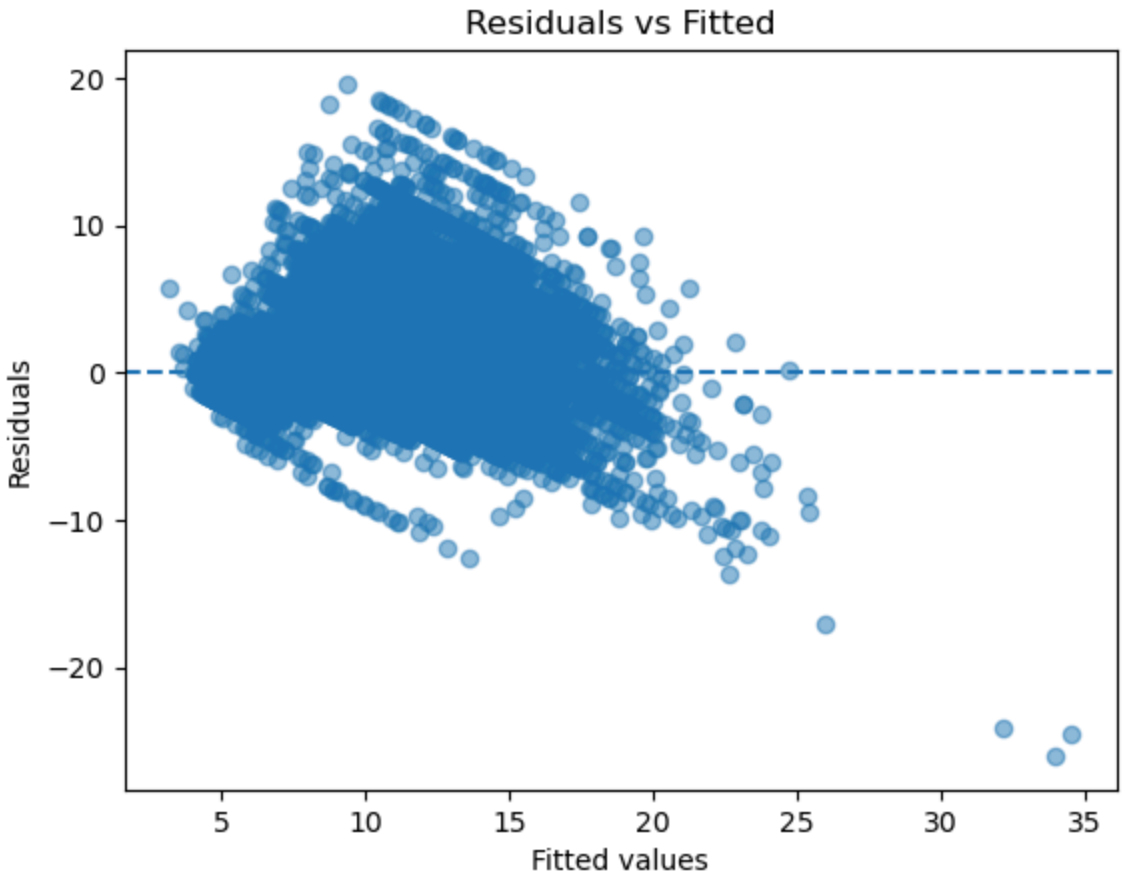
```
[1] Standard Errors assume that the covariance matrix of the errors is corre
ctly specified.
```

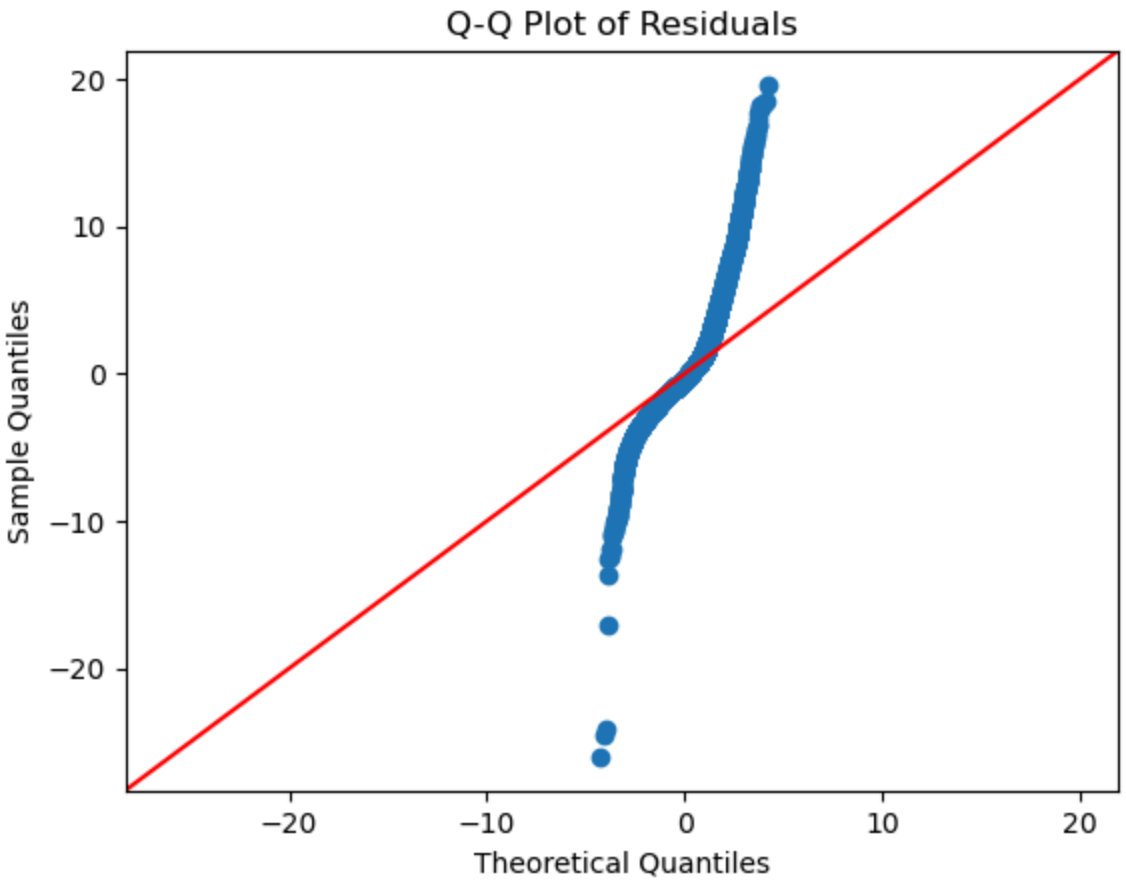
In [7]:
```python
# Residual diagnostics
residuals = ols_model.resid
fitted = ols_model.fittedvalues

plt.figure()
plt.scatter(fitted, residuals, alpha=0.5)
plt.axhline(0, linestyle="--")
plt.xlabel("Fitted values")
plt.ylabel("Residuals")
plt.title("Residuals vs Fitted")
plt.show()

sm.qqplot(residuals, line="45")
plt.title("Q-Q Plot of Residuals")
plt.show()

print("Durbin-Watson statistic:", round(durbin_watson(residuals), 3))
```






```
Durbin-Watson statistic: 1.994
```

In [8]:

```python
from statsmodels.stats.outliers_influence import variance_inflation_factor
import pandas as pd
import numpy as np
import re

X_vif = X_ols.copy()

# Clean column names (remove symbols and reserved words)
X_vif.columns = [re.sub(r'[^0-9a-zA-Z_]+', '_', c) for c in X_vif.columns]
X_vif.columns = [c if c.lower() != "weights" else "weight_var" for c in X_vi

# Ensure everything is numeric float and finite
X_vif = X_vif.apply(pd.to_numeric, errors="coerce").astype(float)
X_vif = X_vif.replace([np.inf, -np.inf], np.nan).dropna()

# Add constant term (OLS intercept)
X_vif_const = sm.add_constant(X_vif)

# Compute VIF safely
vif_data = pd.DataFrame({
    "Feature": X_vif_const.columns,
    "VIF": [variance_inflation_factor(X_vif_const.values, i)
            for i in range(X_vif_const.shape[1])]
})

# Sort and show top results
vif_data.sort_values("VIF", ascending=False).head(10)
```

Out[8]:

	Feature	VIF
0	const	83.049638
2	Diameter	54.270308
1	Length	50.832910
4	Whole_weight	18.629085
5	Shell_weight	15.863447
3	Height	7.722283
6	Sex_I	2.367593
7	Sex_M	1.438633

GitHub Repository https://github.com/alburybloom/ADDS8555-2