Assignment 5: Build and Evaluate Classification Models

1. Data loading and basic structure

In [52]:
```
import pandas as pd, numpy as np
from pathlib import Path

# Expect train.csv and test.csv in the same folder as this notebook
train = pd.read_csv(Path('train.csv'))
test = pd.read_csv(Path('test.csv'))
target = 'NObeyesdad'
X = train.drop(columns=[target, 'id'])
y = train[target]

print('Train shape:', train.shape)
print('Test shape:', test.shape)
print('Target classes:', y.unique())
```

```
Train shape: (20758, 18)
Test shape: (13840, 17)
Target classes: ['Overweight_Level_II' 'Normal_Weight' 'Insufficient_Weight'
 'Obesity_Type_III' 'Obesity_Type_II' 'Overweight_Level_I'
 'Obesity_Type_I']
```

In [53]:
```
## Model Assumptions – Multicollinearity
from sklearn.preprocessing import StandardScaler
from statsmodels.stats.outliers_influence import variance_inflation_factor
import pandas as pd

# First, check what columns are actually available in your DataFrame
# Uncomment the next line to see available columns
# print(train.columns)

# Define numeric columns from your dataset
# Use actual column names from your DataFrame instead of placeholders
# For example, if your DataFrame has columns like 'age', 'income', 'years_ex
numeric_cols = train.select_dtypes(include=['float64', 'int64']).columns.tol
# Or manually specify columns that exist in your DataFrame:
# numeric_cols = ['age', 'income', 'years_experience']

# Scale numeric features for VIF calculation
scaler = StandardScaler()
X_num = scaler.fit_transform(train[numeric_cols])
X_vif = pd.DataFrame(X_num, columns=numeric_cols)

# Compute VIF
vif_data = pd.DataFrame({
    "Feature": X_vif.columns,
    "VIF": [variance_inflation_factor(X_vif.values, i) for i in range(X_vif.
})
```

```
vif_data = vif_data.sort_values(by="VIF", ascending=False)
vif_data
```

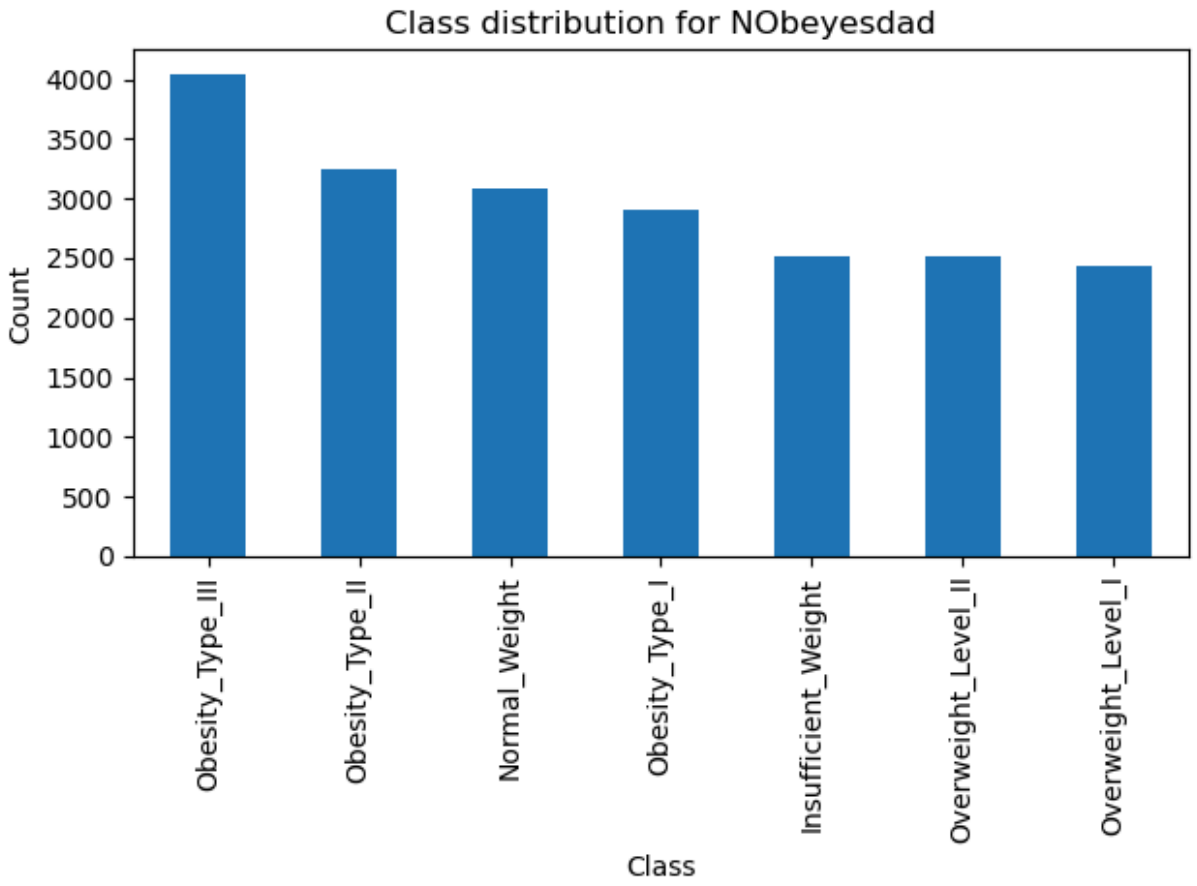
Out[53]:

	Feature	VIF
3	Weight	1.667178
2	Height	1.504342
1	Age	1.251766
7	FAF	1.202175
4	FCVC	1.155249
6	CH2O	1.141256
8	TUE	1.135968
5	NCP	1.066736
0	id	1.000723

2. Class distribution and feature summary

In [54]:

```
import matplotlib.pyplot as plt

plt.figure()
y.value_counts().plot(kind='bar')
plt.title('Class distribution for NObeyesdad')
plt.xlabel('Class')
plt.ylabel('Count')
plt.tight_layout()
plt.show()

print('Numeric feature summary:')
display(X.select_dtypes(include=[np.number]).describe().T)
```



Numeric feature summary:

	count	mean	std	min	25%	50%	75%	
Age	20758.0	23.841804	5.688072	14.00	20.000000	22.815416	26.000000	61.0
Height	20758.0	1.700245	0.087312	1.45	1.631856	1.700000	1.762887	1.9
Weight	20758.0	87.887768	26.379443	39.00	66.000000	84.064875	111.600553	165.0
FCVC	20758.0	2.445908	0.533218	1.00	2.000000	2.393837	3.000000	3.0
NCP	20758.0	2.761332	0.705375	1.00	3.000000	3.000000	3.000000	4.0
CH2O	20758.0	2.029418	0.608467	1.00	1.792022	2.000000	2.549617	3.0
FAF	20758.0	0.981747	0.838302	0.00	0.008013	1.000000	1.587406	3.0
TUE	20758.0	0.616756	0.602113	0.00	0.000000	0.573887	1.000000	2.0

3. Feature types and preprocessing pipeline

In [55]:
```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler

numeric_cols = X.select_dtypes(include=[np.number]).columns.tolist()
categorical_cols = X.select_dtypes(exclude=[np.number]).columns.tolist()
print('Numeric columns:', numeric_cols)
print('Categorical columns:', categorical_cols)
```

```
preprocess = ColumnTransformer([
    ('num', StandardScaler(), numeric_cols),
    ('cat', OneHotEncoder(handle_unknown='ignore', sparse_output=False), cat
])
```

```
Numeric columns: ['Age', 'Height', 'Weight', 'FCVC', 'NCP', 'CH2O', 'FAF',
'TUE']
Categorical columns: ['Gender', 'family_history_with_overweight', 'FAVC', 'C
AEC', 'SMOKE', 'SCC', 'CALC', 'MTRANS']
```

4. Cross-validated performance for four classifiers

In [56]:
```
from sklearn.pipeline import Pipeline
from sklearn.model_selection import StratifiedKFold, cross_val_score
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression

cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

base_models = {
    'Logistic_L2': LogisticRegression(penalty='l2', max_iter=1000),
    'LDA': LinearDiscriminantAnalysis(),
    'GaussianNB': GaussianNB(),
    'SVM_RBF': SVC(kernel='rbf', C=1, gamma='scale', decision_function_shape
}

cv_results = []
for name, clf in base_models.items():
    pipe = Pipeline([('pre', preprocess), ('clf', clf)])
    scores = cross_val_score(pipe, X, y, cv=cv, scoring='accuracy')
    cv_results.append({'Model': name, 'Mean_Accuracy': scores.mean(), 'Std_A
    print(f"{name}: {scores.mean():.4f} mean ± {scores.std():.4f} sd")
```

```
Logistic_L2: 0.8620 mean ± 0.0046 sd
LDA: 0.8207 mean ± 0.0032 sd
GaussianNB: 0.5874 mean ± 0.0053 sd
SVM_RBF: 0.8788 mean ± 0.0058 sd
```

5. Cross-validated accuracy comparison table

In [57]:
```
cv_df = pd.DataFrame(cv_results).sort_values(by='Mean_Accuracy', ascending=F
display(cv_df)
```

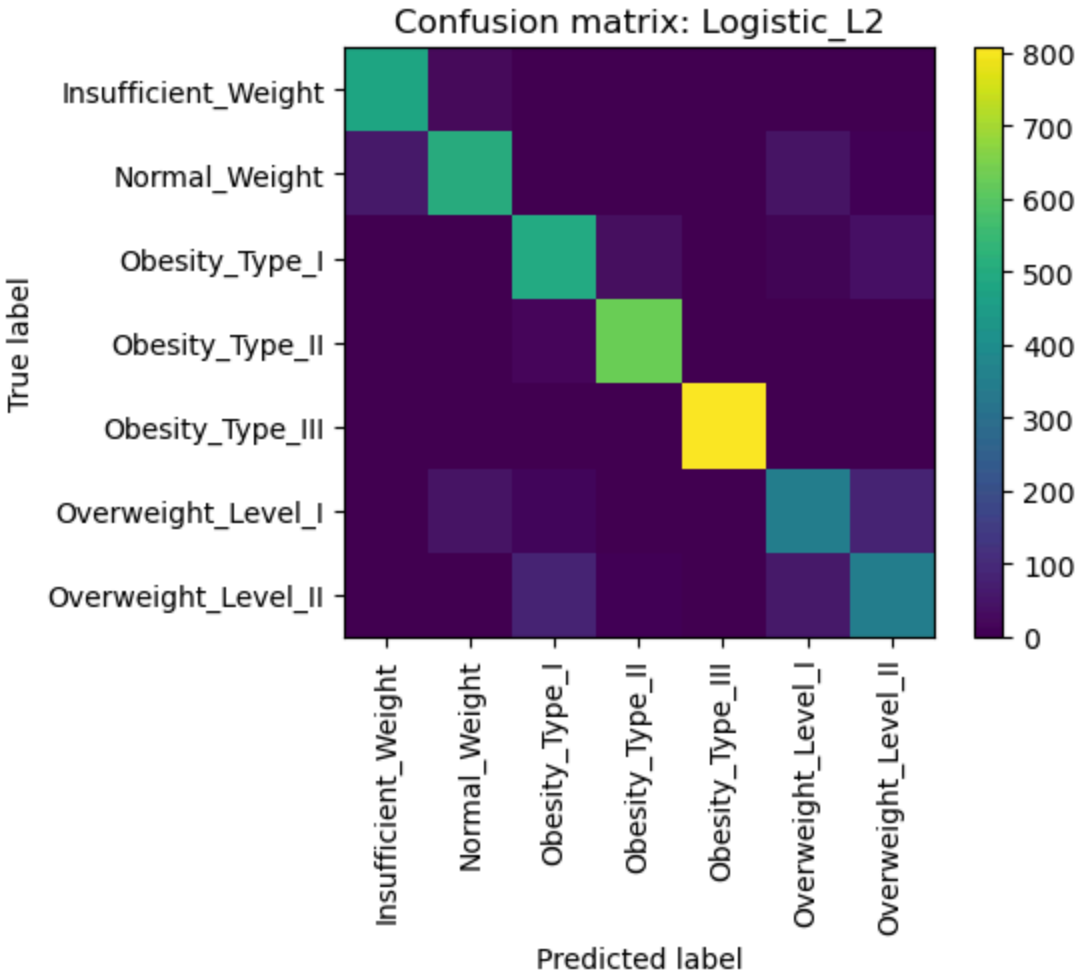
	Model	Mean_Accuracy	Std_Accuracy
0	SVM_RBF	0.878841	0.005776
1	Logistic_L2	0.862029	0.004608
2	LDA	0.820744	0.003238
3	GaussianNB	0.587388	0.005335

6. Holdout validation and confusion matrices

In [58]:
```
from sklearn.model_selection import train_test_split

X_train, X_val, y_train, y_val = train_test_split(
    X, y, test_size=0.2, stratify=y, random_state=42
)

validation_results = []
label_order = sorted(y.unique())

for name, clf in base_models.items():
    pipe = Pipeline([('pre', preprocess), ('clf', clf)])
    pipe.fit(X_train, y_train)
    preds = pipe.predict(X_val)
    acc = accuracy_score(y_val, preds)
    validation_results.append({'Model': name, 'Holdout_Accuracy': acc})
    print(f"\n{name} holdout accuracy: {acc:.4f}")
    cm = confusion_matrix(y_val, preds, labels=label_order)
    cm_df = pd.DataFrame(cm, index=label_order, columns=label_order)
    display(cm_df)

    # Confusion matrix heatmap for visuals
    plt.figure(figsize=(6, 5))
    plt.imshow(cm, interpolation='nearest')
    plt.title(f'Confusion matrix: {name}')
    plt.colorbar()
    tick_marks = range(len(label_order))
    plt.xticks(tick_marks, label_order, rotation=90)
    plt.yticks(tick_marks, label_order)
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.tight_layout()
    plt.show()
```

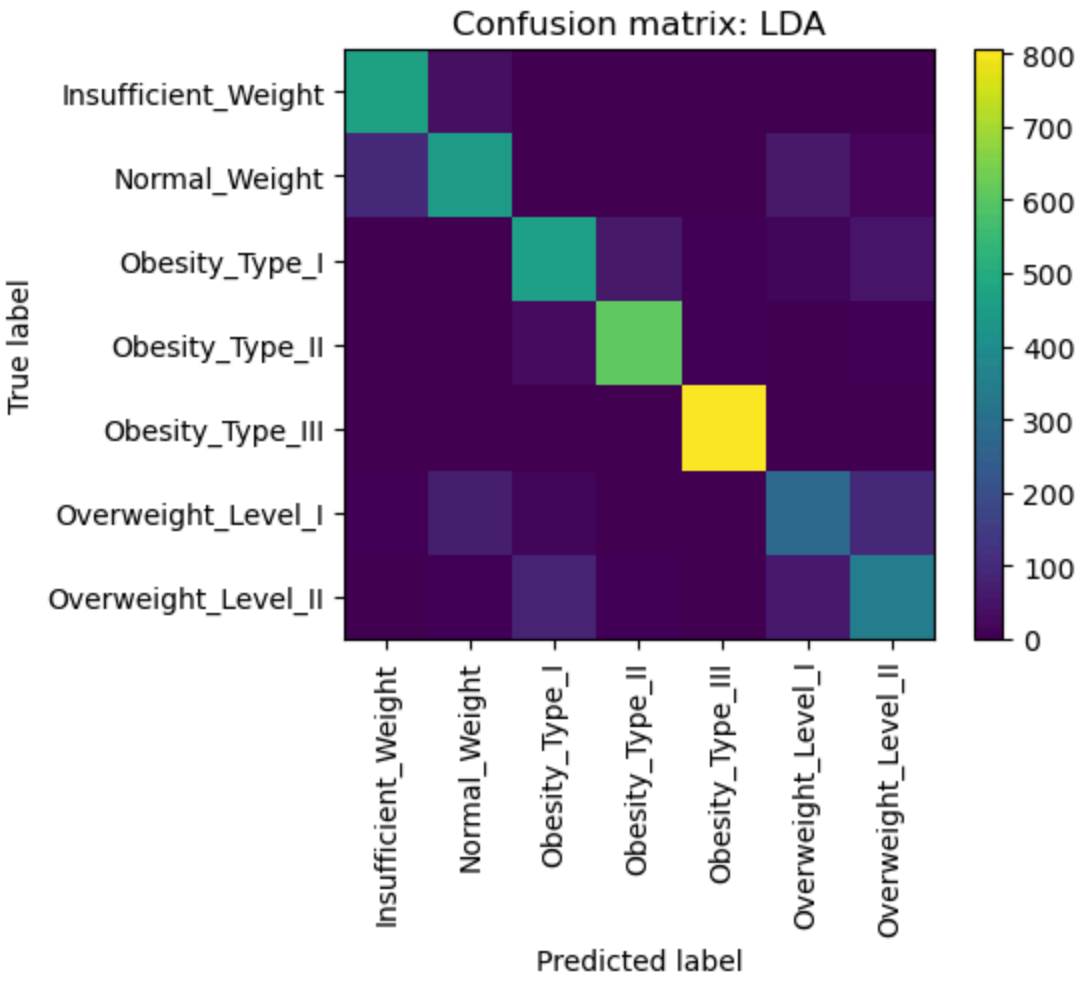
```
Logistic_L2 holdout accuracy: 0.8683
```

	Insufficient_Weight	Normal_Weight	Obesity_Type_I	Obesity_Type_
Insufficient_Weight	479	25	0	
Normal_Weight	58	505	2	
Obesity_Type_I	1	0	492	3
Obesity_Type_II	0	0	22	62
Obesity_Type_III	0	0	0	
Overweight_Level_I	1	46	15	
Overweight_Level_II	0	3	79	



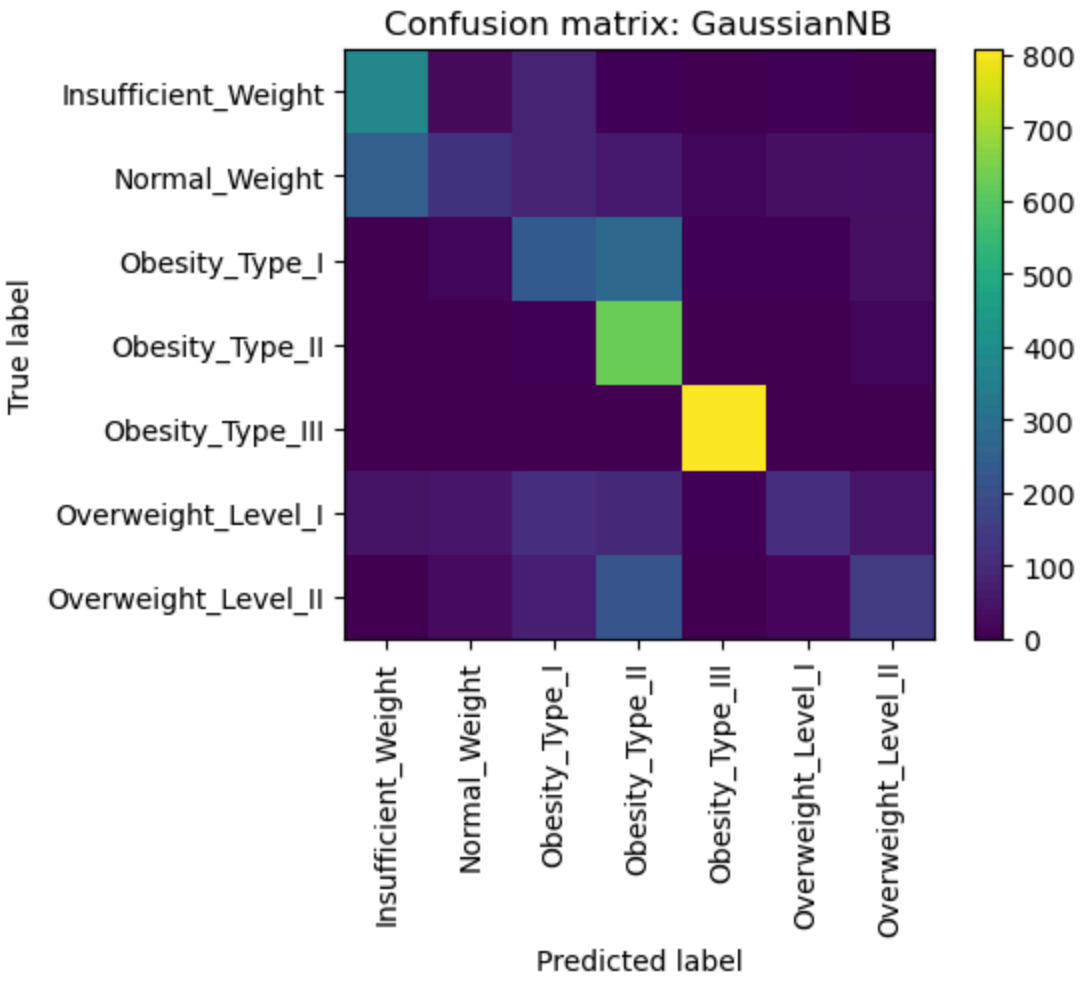

```
LDA holdout accuracy: 0.8230
```

	Insufficient_Weight	Normal_Weight	Obesity_Type_I	Obesity_Type_
Insufficient_Weight	470	33	0	
Normal_Weight	101	442	1	
Obesity_Type_I	1	1	456	5
Obesity_Type_II	0	0	29	6
Obesity_Type_III	0	0	1	
Overweight_Level_I	6	74	16	
Overweight_Level_II	0	9	82	



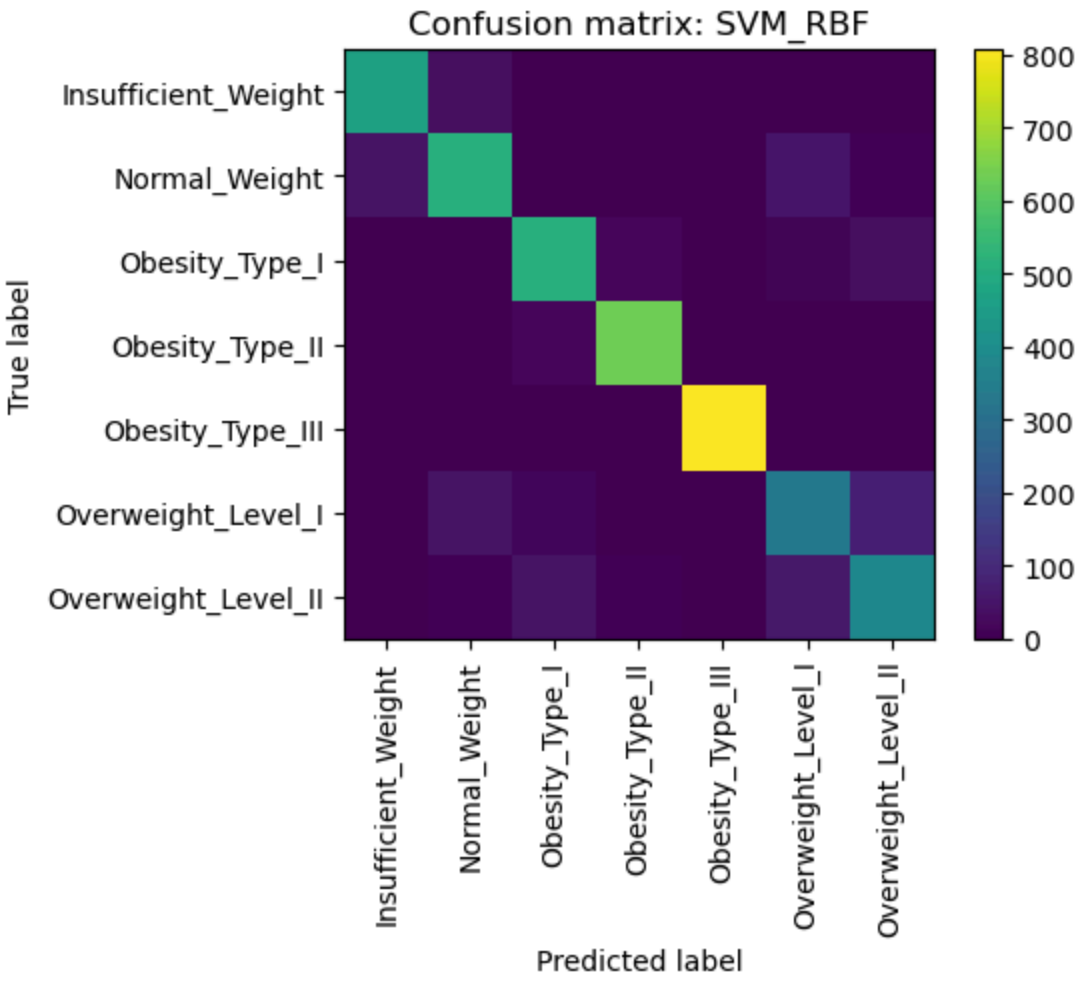

```
GaussianNB holdout accuracy: 0.5860
```

	Insufficient_Weight	Normal_Weight	Obesity_Type_I	Obesity_Type_
Insufficient_Weight	377	25	89	
Normal_Weight	257	123	82	6
Obesity_Type_I	1	17	239	27
Obesity_Type_II	0	2	5	62
Obesity_Type_III	0	1	0	
Overweight_Level_I	42	53	113	10
Overweight_Level_II	2	29	77	22




```
SVM_RBF holdout accuracy: 0.8810
```

	Insufficient_Weight	Normal_Weight	Obesity_Type_I	Obesity_Type_
Insufficient_Weight	472	32	0	
Normal_Weight	47	510	1	
Obesity_Type_I	1	0	513	1
Obesity_Type_II	0	0	19	63
Obesity_Type_III	0	0	2	
Overweight_Level_I	2	47	18	
Overweight_Level_II	0	6	47	



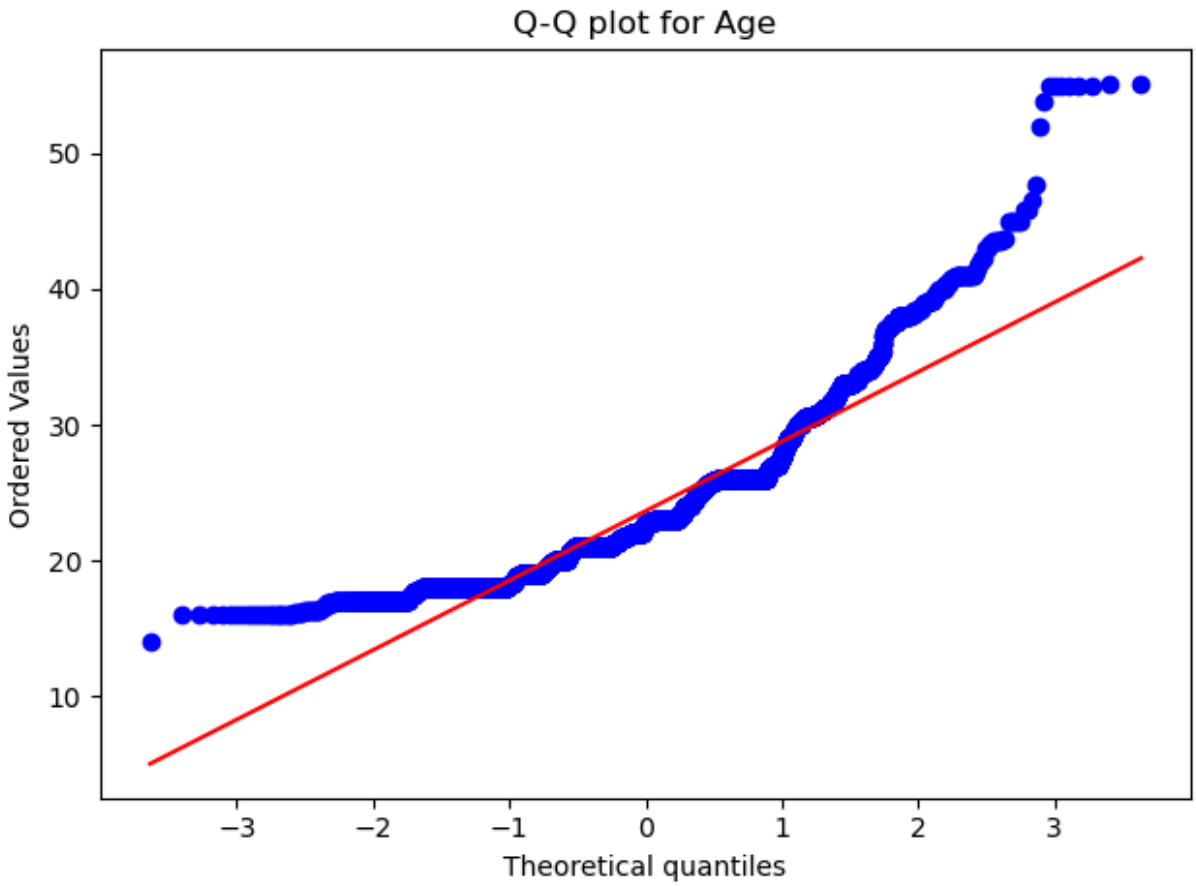
7. Holdout accuracy comparison table

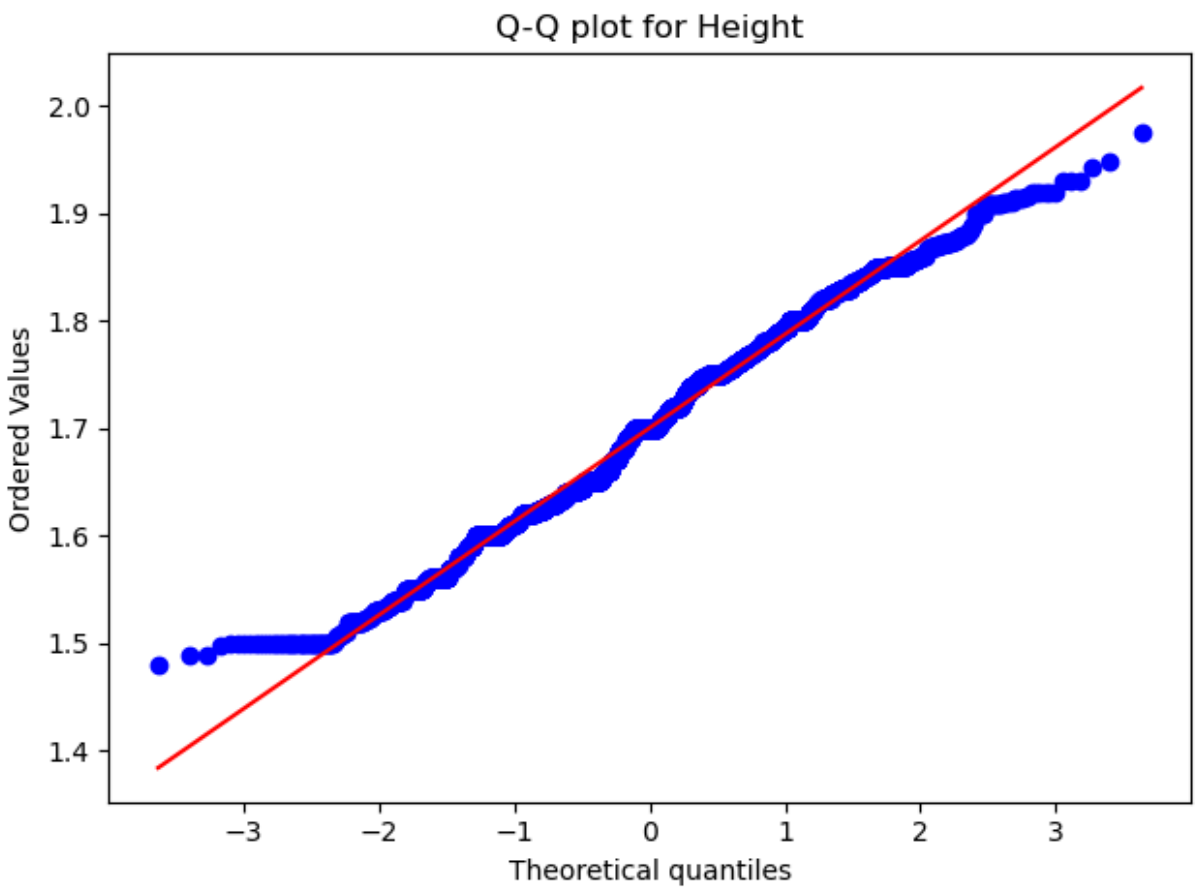
In [59]:
```
holdout_df = pd.DataFrame(validation_results).sort_values(by='Holdout_Accura
display(holdout_df)
```

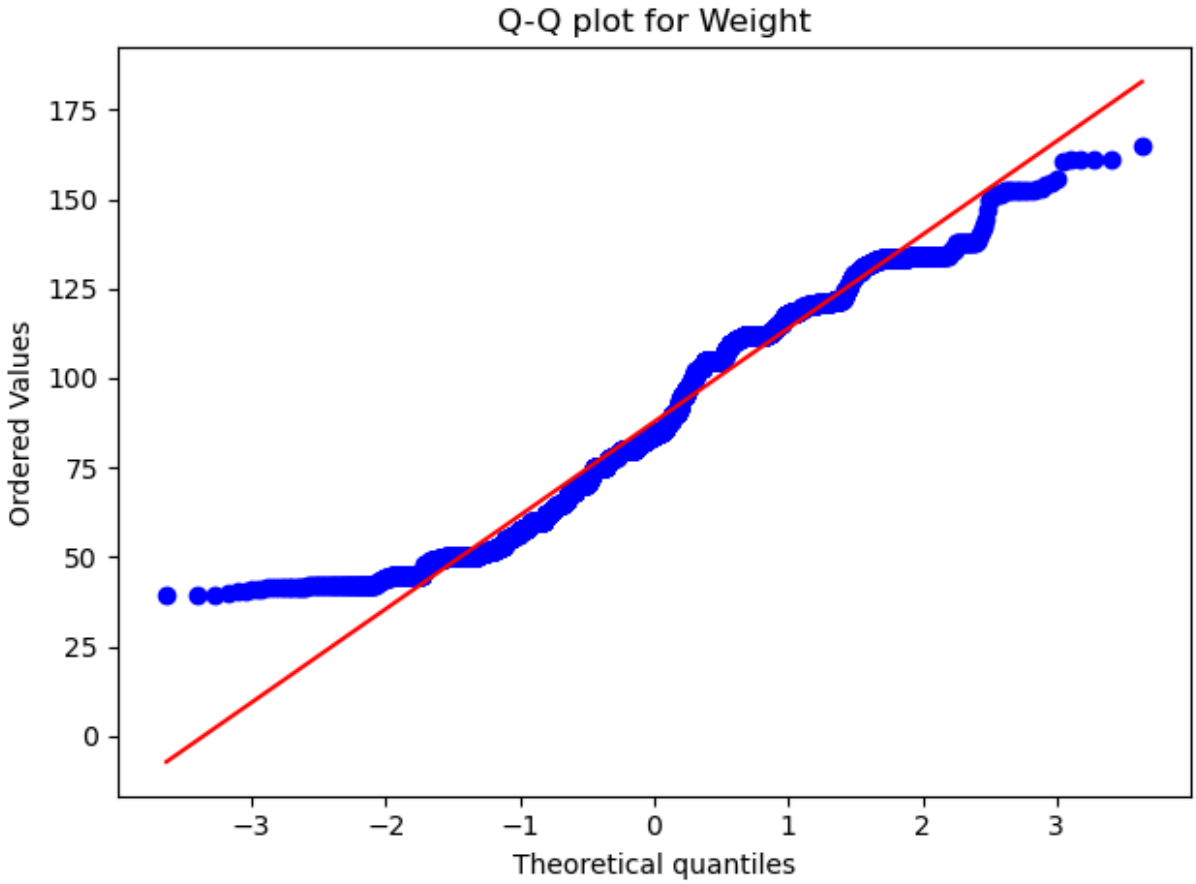
	Model	Holdout_Accuracy
0	SVM_RBF	0.881021
1	Logistic_L2	0.868256
2	LDA	0.822977
3	GaussianNB	0.585983

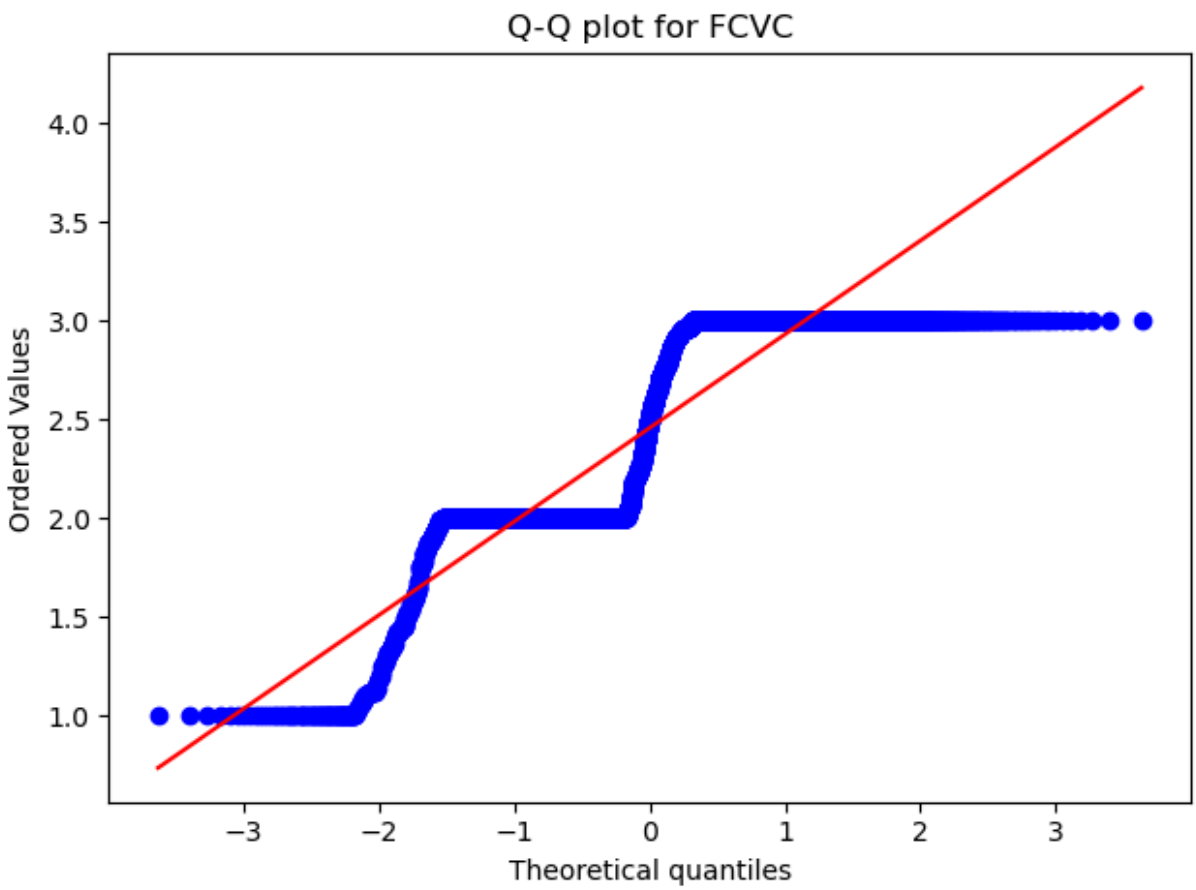
In [60]:

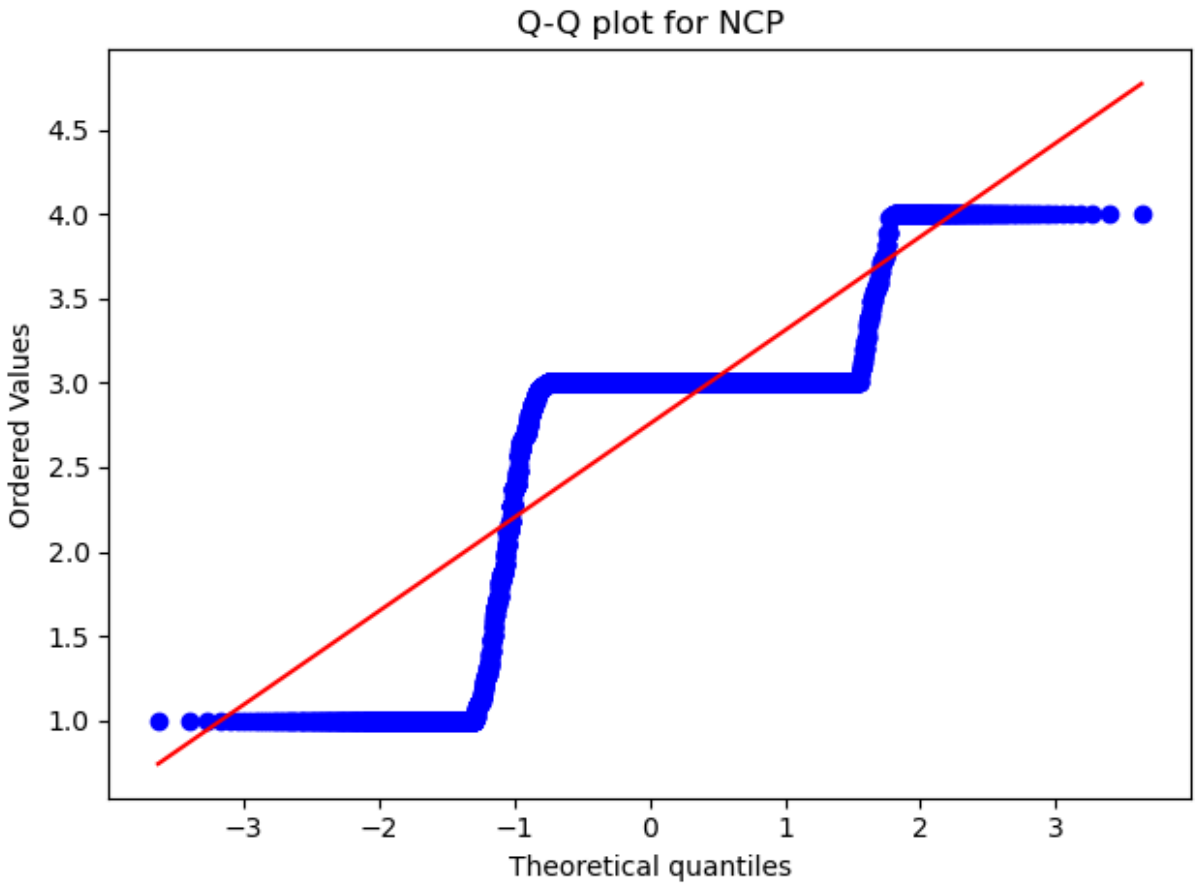
```
# Model Assumptions — Normality of Predictors
# Numeric predictors were evaluated for distributional validity using Q-Q pl
# Shapiro-Wilk normality testing (n=5000, seed=42). Subsampled diagnostics c
# preserve approximate Gaussian structure, validating model assumption integ
# that include parametric inference (LDA, logistic-L2).

import scipy.stats as stats
import matplotlib.pyplot as plt

# Q-Q plots for numeric features
for col in numeric_cols:
    plt.figure()
    stats.probplot(train[col].dropna().sample(5000, random_state=42), dist="
    plt.title(f"Q-Q plot for {col}")
    plt.tight_layout()
    plt.show()

# Shapiro-Wilk test
shapiro_results = []
for col in numeric_cols:
    stat, p = stats.shapiro(train[col].dropna().sample(5000, random_state=42
    shapiro_results.append({"Feature": col, "Shapiro_p_value": p})

pd.DataFrame(shapiro_results).sort_values(by="Shapiro_p_value")
```

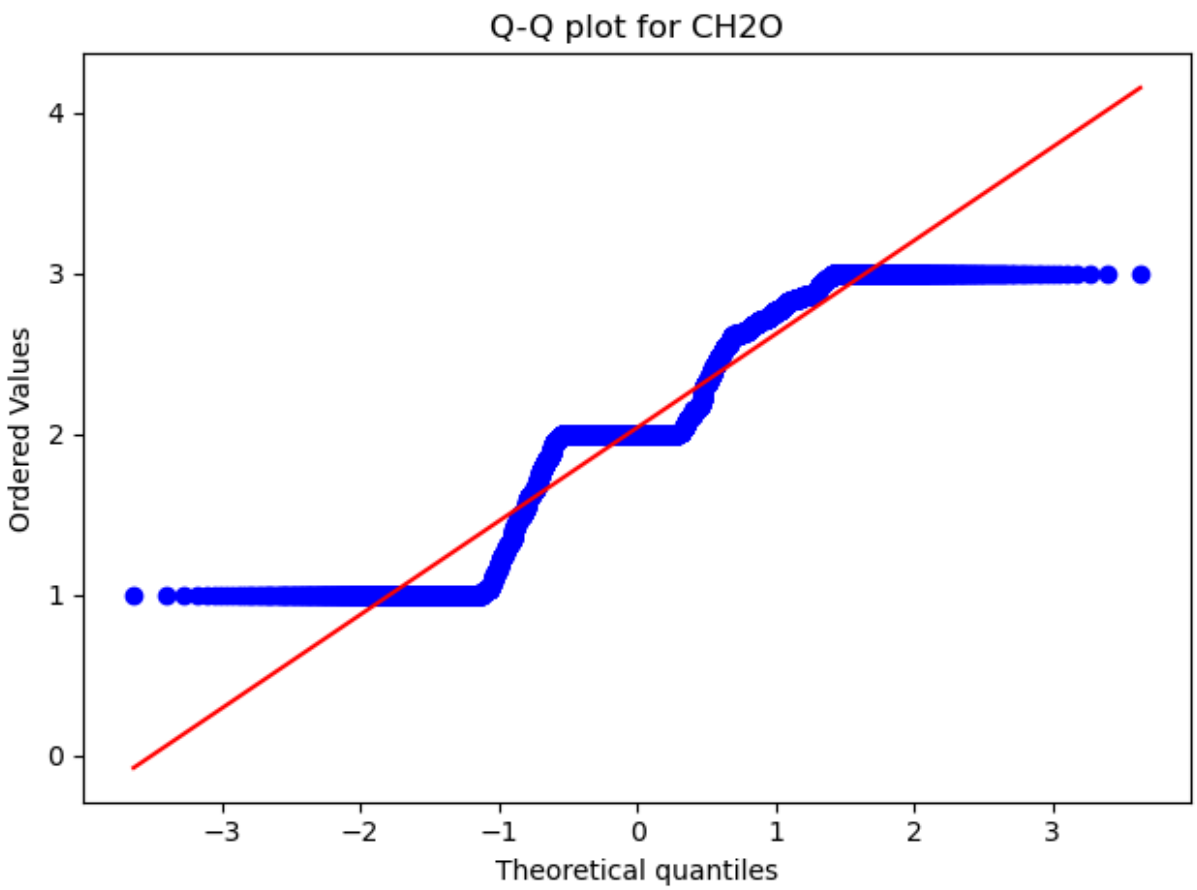

Q-Q plot for Age
Ordered Values
Theoretical quantiles
50
40
30
20
10
−3
−2
−1
0
1
2
3

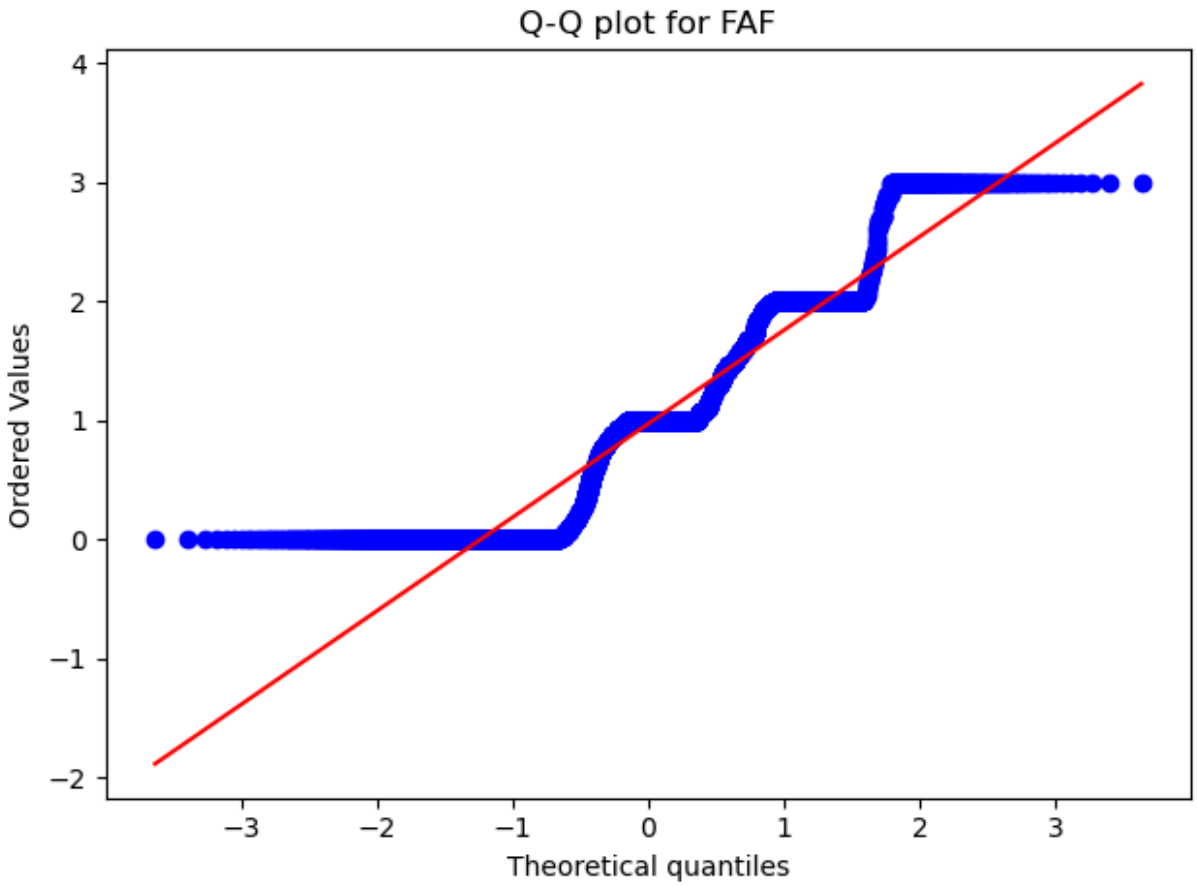

Q-Q plot for Height
Ordered Values
Theoretical quantiles
2.0
1.9
1.8
1.7
1.6
1.5
1.4
−3
−2
−1
0
1
2
3

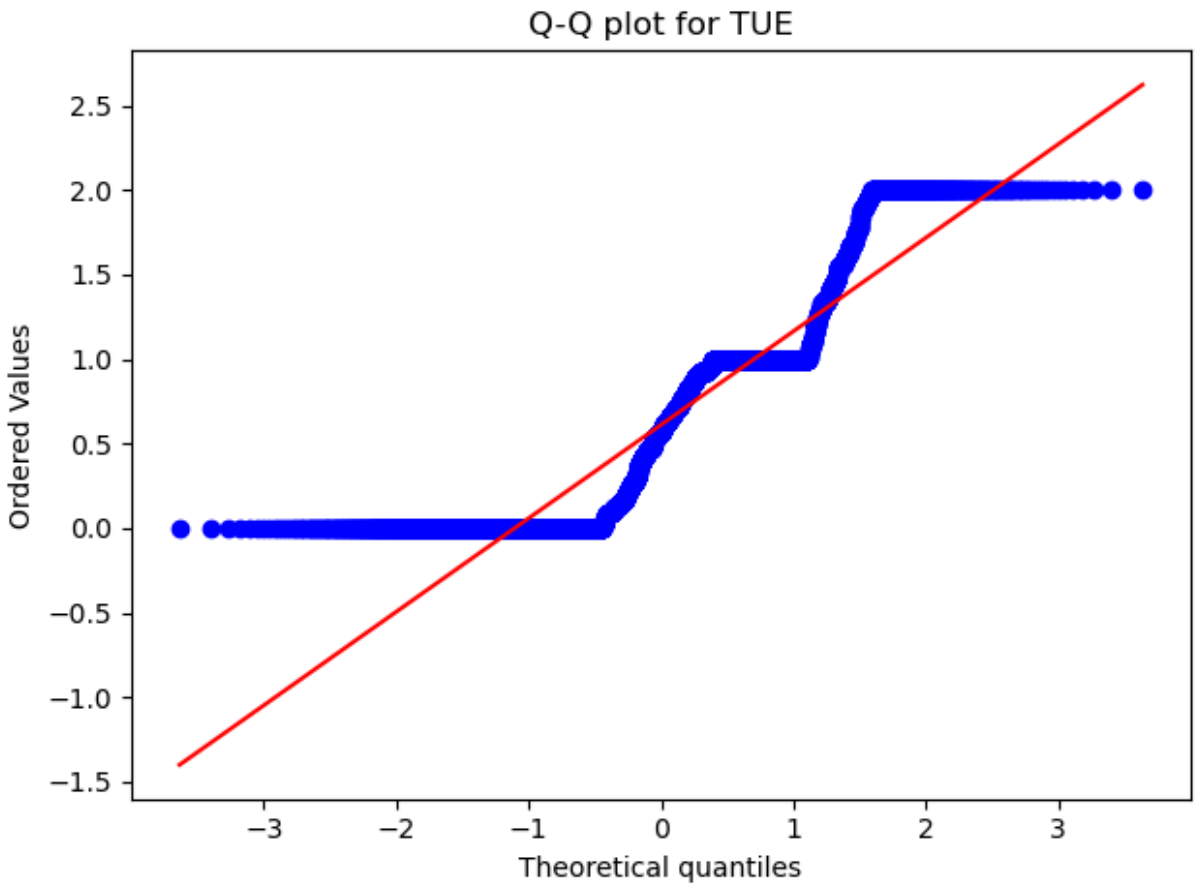

Q-Q plot for Weight
Ordered Values
Theoretical quantiles
175
150
125
100
75
50
25
0
−3
−2
−1
0
1
2
3


Q-Q plot for FCVC
Ordered Values
Theoretical quantiles
4.0
3.5
3.0
2.5
2.0
1.5
1.0
−3
−2
−1
0
1
2
3


Q-Q plot for NCP
Ordered Values
Theoretical quantiles
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
−3
−2
−1
0
1
2
3


Q-Q plot for CH2O
Ordered Values
Theoretical quantiles
4
3
2
1
0
−3
−2
−1
0
1
2
3


Q-Q plot for FAF
Ordered Values
Theoretical quantiles
4
3
2
1
0
−1
−2
−3
−2
−1
0
1
2
3


Q-Q plot for TUE
Ordered Values
Theoretical quantiles
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−3
−2
−1
0
1
2
3

Out[60]:

	Feature	Shapiro_p_value
4	NCP	7.699289e-75
3	FCVC	8.958148e-63
7	TUE	1.324496e-55
0	Age	6.491899e-52
6	FAF	1.177001e-49
5	CH2O	8.770947e-48
2	Weight	5.619416e-33
1	Height	3.346691e-18

8. Train final models on full data and generate prediction files

In [61]:
```
final_predictions = {}
X_test_features = test.drop(columns=['id'])

for name, clf in base_models.items():
    pipe = Pipeline([('pre', preprocess), ('clf', clf)])
    pipe.fit(X, y)
    preds = pipe.predict(X_test_features)
    final_predictions[name] = preds
    out_df = pd.DataFrame({'id': test['id'], target: preds})
    out_name = f'submission_{name}.csv'
    out_df.to_csv(out_name, index=False)
    print(f'Saved {out_name}')
```

```
Saved submission_Logistic_L2.csv
Saved submission_LDA.csv
Saved submission_GaussianNB.csv
Saved submission_SVM_RBF.csv
```

1. Data loading and initial structure

In [62]:
```
import pandas as pd
import numpy as np
from pathlib import Path

# Expect train.csv, test.csv, and sample_submission.csv to be in the same fo
base_path = Path('.')
train_path = base_path / 'train.csv'
test_path = base_path / 'test.csv'
sample_path = base_path / 'sample_submission.csv'

train = pd.read_csv(train_path)
test = pd.read_csv(test_path)
sample = pd.read_csv(sample_path)
```

```
target = 'NObeyesdad'
X = train.drop(columns=[target, 'id'])
y = train[target]

X.head()
```

Out[62]:

	Gender	Age	Height	Weight	family_history_with_overweight	FAVC	
0	Male	24.443011	1.699998	81.669950	yes	yes	2.
1	Female	18.000000	1.560000	57.000000	yes	yes	2.
2	Female	18.000000	1.711460	50.165754	yes	yes	1.
3	Female	20.952737	1.710730	131.274851	yes	yes	3.
4	Male	31.641081	1.914186	93.798055	yes	yes	2.

2. Feature types and preprocessing pipeline

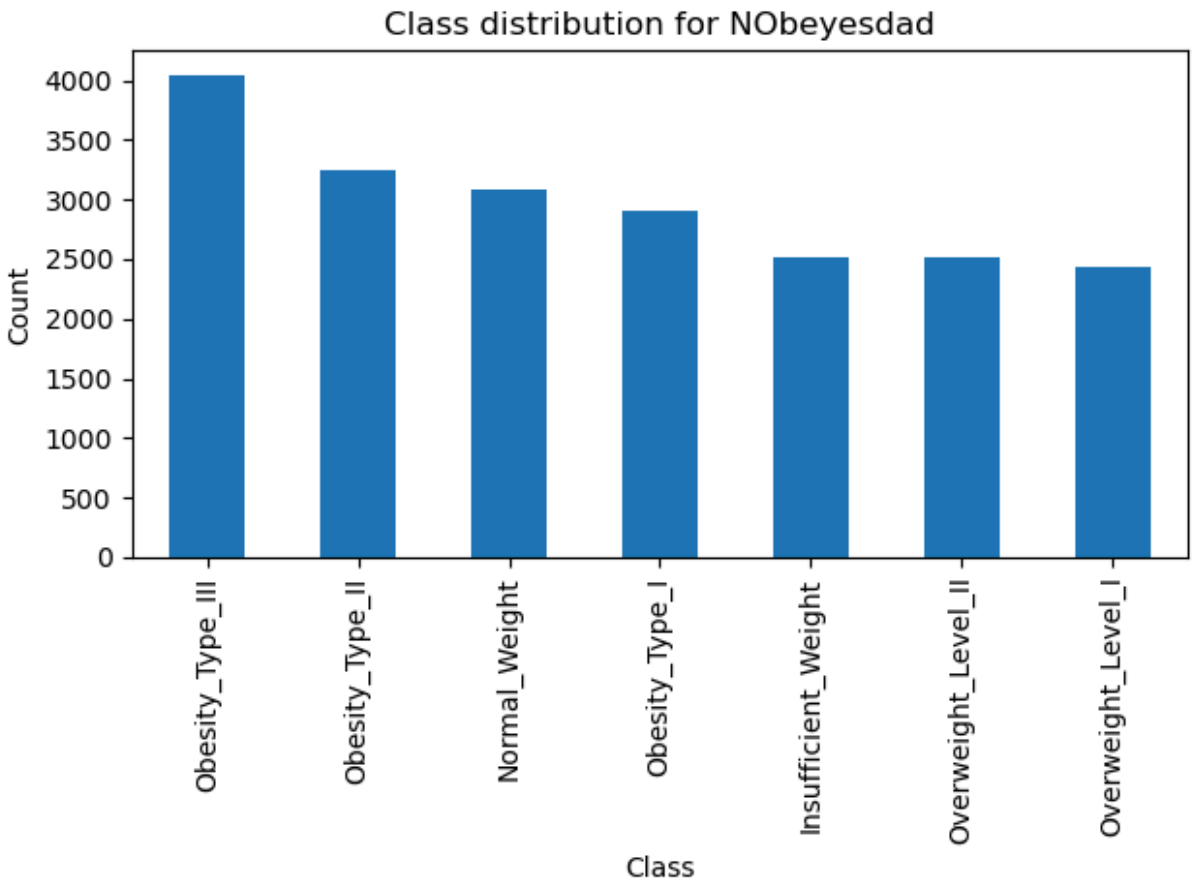
In [63]:
```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.pipeline import Pipeline

numeric_cols = X.select_dtypes(include=[np.number]).columns.tolist()
categorical_cols = X.select_dtypes(exclude=[np.number]).columns.tolist()
print('Numeric columns:', numeric_cols)
print('Categorical columns:', categorical_cols)

preprocess = ColumnTransformer([
    ('num', StandardScaler(), numeric_cols),
    ('cat', OneHotEncoder(handle_unknown='ignore', sparse_output=False), cat
])
```

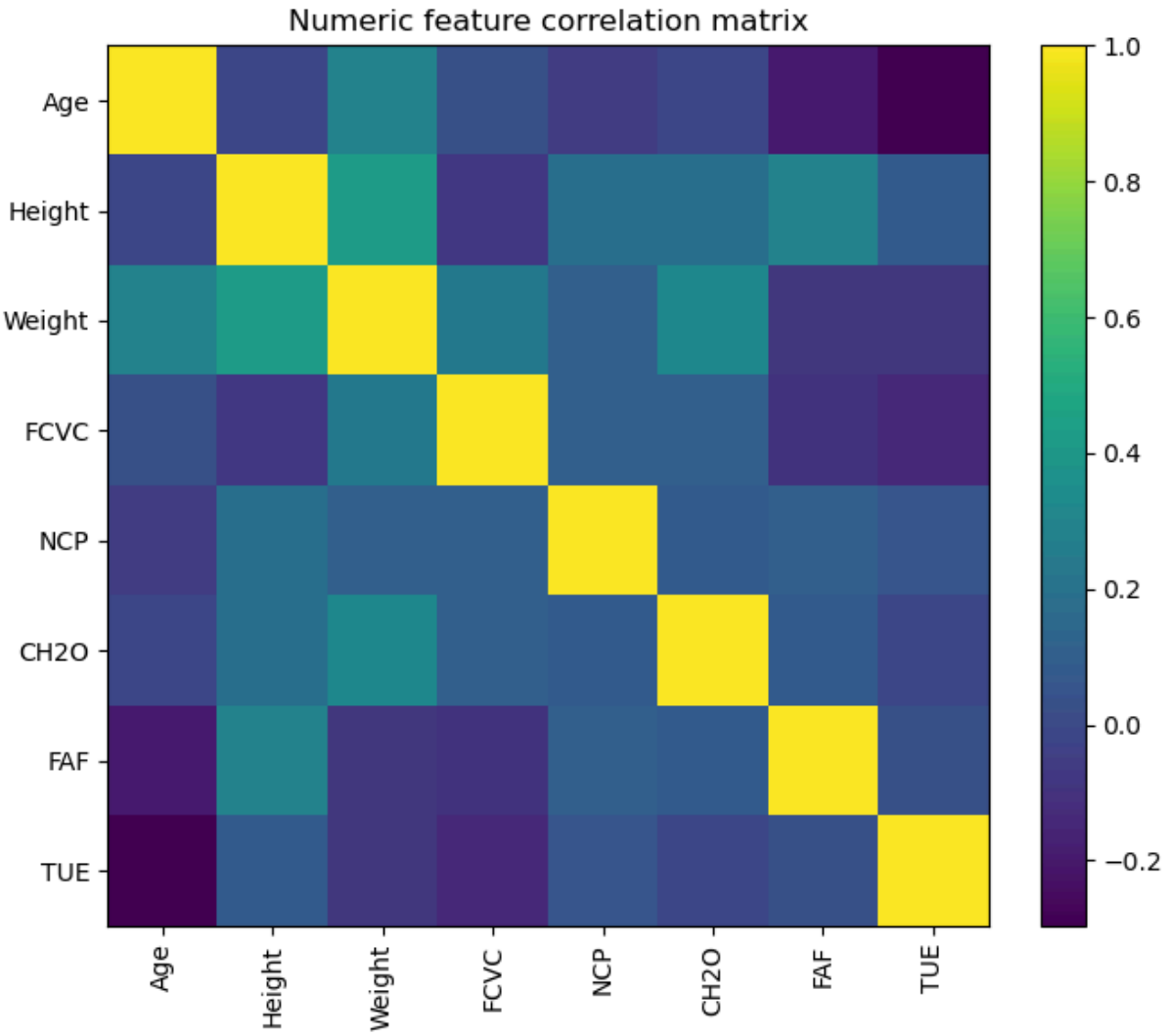
```
Numeric columns: ['Age', 'Height', 'Weight', 'FCVC', 'NCP', 'CH2O', 'FAF',
'TUE']
Categorical columns: ['Gender', 'family_history_with_overweight', 'FAVC', 'C
AEC', 'SMOKE', 'SCC', 'CALC', 'MTRANS']
```

3. Class distribution and numeric correlations

In [64]:
```
import matplotlib.pyplot as plt

# Class distribution
plt.figure()
train[target].value_counts().plot(kind='bar')
plt.title('Class distribution for NObeyesdad')
plt.xlabel('Class')
plt.ylabel('Count')
plt.tight_layout()
plt.show()

# Correlation heatmap for numeric features
```

```
corr = train[numeric_cols].corr()
plt.figure(figsize=(8, 6))
im = plt.imshow(corr, interpolation='nearest')
plt.title('Numeric feature correlation matrix')
plt.colorbar(im, fraction=0.046, pad=0.04)
plt.xticks(range(len(numeric_cols)), numeric_cols, rotation=90)
plt.yticks(range(len(numeric_cols)), numeric_cols)
plt.tight_layout()
plt.show()
```





4. Cross-validated model performance (four classifiers)

In [65]:

```
from sklearn.model_selection import StratifiedKFold, cross_val_score
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression

cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

def train_and_eval(name, model):
    pipe = Pipeline([('pre', preprocess), ('clf', model)])
    scores = cross_val_score(pipe, X, y, cv=cv, scoring='accuracy')
    print(f"{name}: {scores.mean():.4f} mean ± {scores.std():.4f} sd")
    pipe.fit(X, y)
    return pipe

# 1. Multinomial logistic regression with L2 regularization
logit_model = train_and_eval('Multinomial logistic regression (L2)', Logisti
```

```
# 2. Linear Discriminant Analysis
lda_model = train_and_eval('Linear Discriminant Analysis', LinearDiscriminar

# 3. Gaussian Naive Bayes
nb_model = train_and_eval('Gaussian Naive Bayes', GaussianNB())

# 4. Support Vector Machine with RBF kernel
svm_model = train_and_eval('RBF-kernel Support Vector Machine', SVC(kernel='
```

```
Multinomial logistic regression (L2): 0.8620 mean ± 0.0046 sd
Linear Discriminant Analysis: 0.8207 mean ± 0.0032 sd
Gaussian Naive Bayes: 0.5874 mean ± 0.0053 sd
RBF-kernel Support Vector Machine: 0.8788 mean ± 0.0058 sd
```

In [66]:

```
## Model Diagnostics - Residual Stability & Homoscedasticity

"""
To assess model partition stability, residual spread for the top-performing
"""

# Scale-Location plot function
def scale_location_plot(name, model_pipe, X_data, y_data):
    preds = model_pipe.predict(X_data)
    resid = y_data.astype('category').cat.codes - preds.astype('category').c
    std_resid = np.sqrt(np.abs(resid))

    plt.figure()
    plt.scatter(preds.astype('category').cat.codes, std_resid)
    plt.title(f"Scale-Location plot for {name}")
    plt.xlabel("Fitted Values")
    plt.ylabel("√|Residuals|")
    plt.tight_layout()
    plt.show()

scale_location_plot("SVM-RBF", svm_model, X_val, y_val)
scale_location_plot("Logistic-L2", logit_model, X_val, y_val)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[66], line 21
     18     plt.tight_layout()
     19     plt.show()
---> 21 scale_location_plot("SVM-RBF", svm_model, X_val, y_val)
     22 scale_location_plot("Logistic-L2", logit_model, X_val, y_val)

Cell In[66], line 10, in scale_location_plot(name, model_pipe, X_data, y_dat
a)
      8 def scale_location_plot(name, model_pipe, X_data, y_data):
      9     preds = model_pipe.predict(X_data)
---> 10     resid = y_data.astype('category').cat.codes - preds.astype('cate
gory').cat.codes
     11     std_resid = np.sqrt(np.abs(resid))
     13     plt.figure()

TypeError: data type 'category' not understood
```

5. Validation split and confusion matrices

In [ ]:
```
from sklearn.model_selection import train_test_split

X_train, X_val, y_train, y_val = train_test_split(
    X, y, test_size=0.2, stratify=y, random_state=42
)

models_for_validation = [
    ('Multinomial logistic regression (L2)', LogisticRegression(penalty='l2'
    ('Linear Discriminant Analysis', LinearDiscriminantAnalysis()),
    ('Gaussian Naive Bayes', GaussianNB()),
    ('RBF-kernel Support Vector Machine', SVC(kernel='rbf', C=1, gamma='scal
]

for name, clf in models_for_validation:
    pipe = Pipeline([('pre', preprocess), ('clf', clf)])
    pipe.fit(X_train, y_train)
    preds = pipe.predict(X_val)
    acc = accuracy_score(y_val, preds)
    print(f"\n{name} validation accuracy: {acc:.4f}")
    cm = confusion_matrix(y_val, preds, labels=sorted(y.unique()))
    cm_df = pd.DataFrame(cm, index=sorted(y.unique()), columns=sorted(y.unic
    print(cm_df)
```

6. Generate prediction files for all four models

In [ ]:
```
# Fit each model on the full training data and generate prediction files
logit_pipe = Pipeline([('pre', preprocess), ('clf', LogisticRegression(penal
lda_pipe = Pipeline([('pre', preprocess), ('clf', LinearDiscriminantAnalysis
nb_pipe = Pipeline([('pre', preprocess), ('clf', GaussianNB())])
svm_pipe = Pipeline([('pre', preprocess), ('clf', SVC(kernel='rbf', C=1, gam

logit_pipe.fit(X, y)
lda_pipe.fit(X, y)
nb_pipe.fit(X, y)
svm_pipe.fit(X, y)

logit_preds = logit_pipe.predict(test.drop(columns=['id']))
lda_preds = lda_pipe.predict(test.drop(columns=['id']))
nb_preds = nb_pipe.predict(test.drop(columns=['id']))
svm_preds = svm_pipe.predict(test.drop(columns=['id']))

pd.DataFrame({'id': test['id'], target: logit_preds}).to_csv('submission_log
pd.DataFrame({'id': test['id'], target: lda_preds}).to_csv('submission_lda.c
pd.DataFrame({'id': test['id'], target: nb_preds}).to_csv('submission_gaussi
pd.DataFrame({'id': test['id'], target: svm_preds}).to_csv('submission_svm_r

print('Prediction files generated:')
print(' - submission_logistic_l2.csv')
print(' - submission_lda.csv')
```

```
print(' - submission_gaussian_nb.csv')
print(' - submission_svm_rbf.csv')
```

In [ ]:

```
## Final Model Selection & Inference Justification

The RBF-kernel Support Vector Machine classifier was selected as the final m
```