Model Evaluation and Ridge Regression Summary with Code

1. Splitting Data

You can split your data using train_test_split from sklearn.model_selection to create training and testing datasets.

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

2. Generalization Error

Generalization error is the difference between the error on the training data and the error on the test data.

3. Cross-Validation

Use K-Fold cross-validation to iterate over train-test splits and average the results.

```
from sklearn.model_selection import cross_val_score

scores = cross_val_score(model, X, y, cv=5)
print("Average score:", scores.mean())
```

4. Polynomial Order Selection

Compare test errors of different polynomial orders to choose the best fit and avoid underfitting or overfitting.

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

mse_list = []
for degree in range(1, 6):
    poly = PolynomialFeatures(degree=degree)
    X_poly = poly.fit_transform(X_train)
    model = LinearRegression()
    model.fit(X_poly, y_train)
    X_test_poly = poly.transform(X_test)
    y_pred = model.predict(X_test_poly)
    mse = mean_squared_error(y_test, y_pred)
    mse_list.append(mse)

print("MSE for each degree:", mse_list)
```

5. Ridge Regression

Use Ridge regression when multicollinearity is present. It reduces overfitting by penalizing large coefficients.

```
from sklearn.linear_model import Ridge

ridge = Ridge(alpha=1.0)
ridge.fit(X_train, y_train)
y_pred = ridge.predict(X_test)
```

6. Alpha Optimization

Select alpha that maximizes R^2 score using a validation set.

```
from sklearn.metrics import r2_score

alphas = [0.1, 1, 10, 100]
best_score = -1
best_alpha = None

for alpha in alphas:
    ridge = Ridge(alpha=alpha)
    ridge.fit(X_train, y_train)
    y_pred = ridge.predict(X_test)
    score = r2_score(y_test, y_pred)
    if score > best_score:
        best_score = score
        best_alpha = alpha

print("Best alpha:", best_alpha, "with R^2:", best_score)
```

7. Grid Search

Automate hyperparameter tuning using GridSearchCV.

```
from sklearn.model_selection import GridSearchCV

param_grid = {'alpha': [0.01, 0.1, 1, 10]}
ridge = Ridge()
grid = GridSearchCV(ridge, param_grid, cv=5)
grid.fit(X, y)
print("Best Parameters:", grid.best_params_)
```