STAT 4710 Final Project -

Andrew Raine, Graham Branscom, Mahika Calyanakoti

Part 1: Imports and Loading the Dataset

[ ] ↳ *7 cells hidden*

Part 2: EDA

Data cleaning and feature engineering

```
#@title Data cleaning and feature engineering
# Checking for nans; none exist
na_counts = df.isna().sum()

# filtering out crab outliers
df = df[df.Height < 1]

# binarizing the age
median_age = df['Age'].median()
df['Young|Old'] = (df['Age'] > median_age).astype(int)
```

```
df.to_csv('cleaned_data.csv', index=False)
```

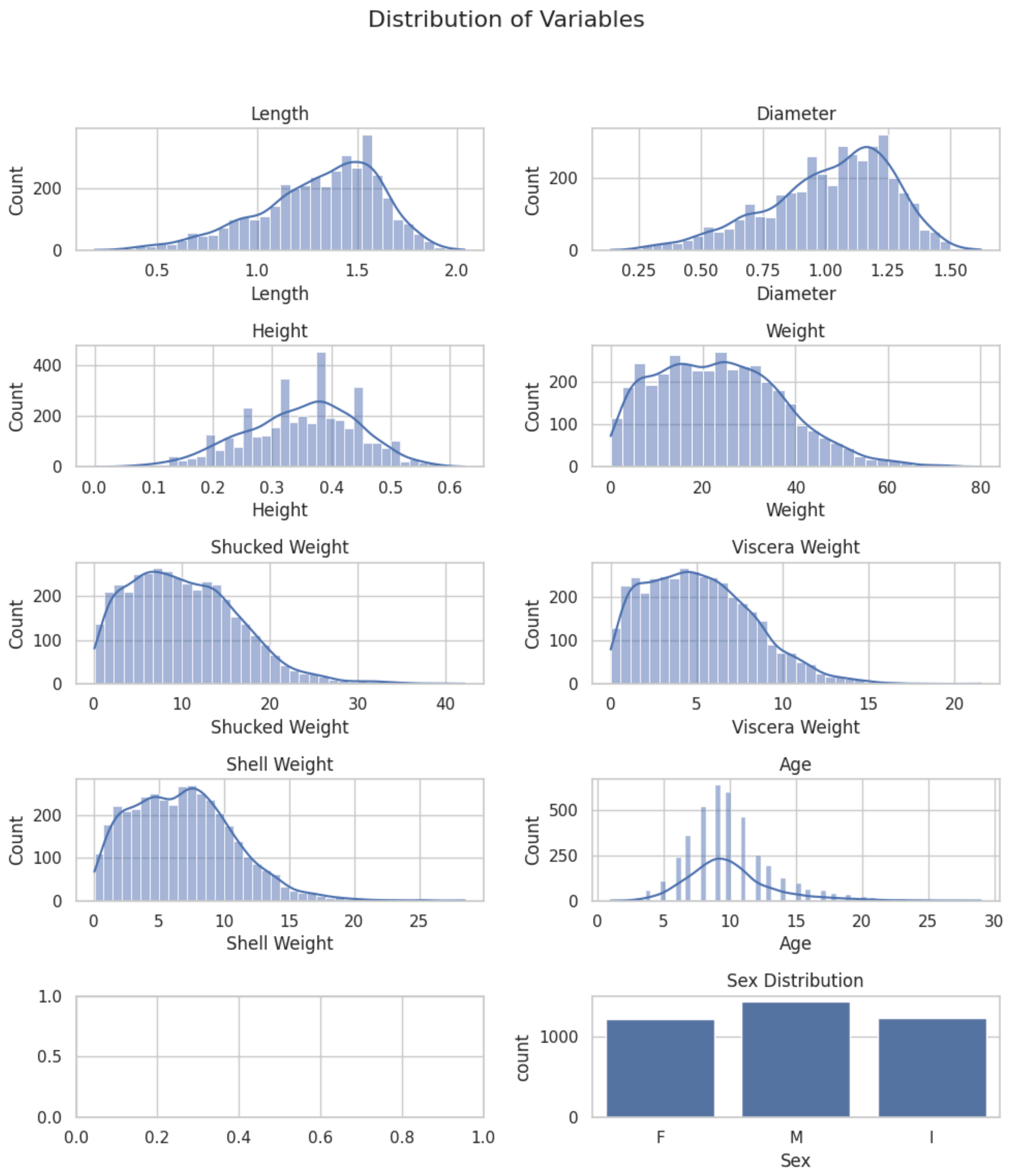
Crab summary

Show code

	Length	Diameter	Height	Weight	Shucked Weight	Viscera Weight	Shell Weight	Age	Young|Old
count	3891.000000	3891.000000	3891.000000	3891.000000	3891.000000	3891.000000	3891.000000	3891.000000	3891.000000
mean	1.311234	1.020827	0.348497	23.558959	10.202101	5.134797	6.794634	9.955281	0.350039
std	0.300408	0.248208	0.096037	13.880211	6.267671	3.101686	3.942167	3.221642	0.477043
min	0.187500	0.137500	0.000000	0.056699	0.028349	0.014175	0.042524	1.000000	0.000000
25%	1.125000	0.875000	0.287500	12.672227	5.336793	2.664853	3.713785	8.000000	0.000000
50%	1.362500	1.062500	0.362500	22.792998	9.539607	4.861939	6.662133	10.000000	0.000000
75%	1.537500	1.200000	0.412500	32.786197	14.266886	7.200773	9.355335	11.000000	1.000000
max	2.037500	1.625000	0.625000	80.101512	42.184056	21.545620	28.491248	29.000000	1.000000

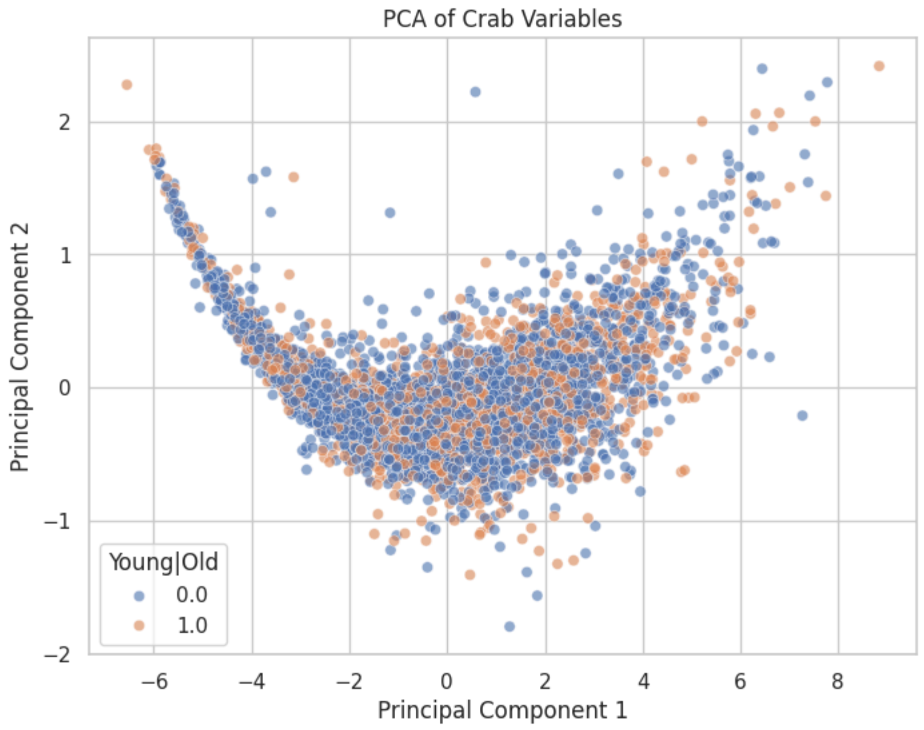
Variable distributions

Show code


Distribution of Variables
Length
Count
Length
Diameter
Count
Diameter
Height
Count
Height
Weight
Count
Weight
Shucked Weight
Count
Shucked Weight
Viscera Weight
Count
Viscera Weight
Shell Weight
Count
Shell Weight
Age
Count
Age
Sex Distribution
count
F
M
I
Sex

PCA

```
#@title PCA
from sklearn.decomposition import PCA

# Independent variables for PCA (excluding 'Age' and 'Young|Old')
pca_vars = ['Length', 'Diameter', 'Height', 'Weight', 'Shucked Weight', 'Viscera Weight', 'Shell Weight']
X_pca = df[pca_vars]

# Standardizing the data for PCA
scaler_pca = StandardScaler()
X_pca_scaled = scaler_pca.fit_transform(X_pca)

# Applying PCA to reduce dimensions to 2
pca = PCA(n_components=2)
principal_components = pca.fit_transform(X_pca_scaled)

# Creating a DataFrame for the principal components
pca_df = pd.DataFrame(data=principal_components, columns=['PC1', 'PC2'])
# Including the 'Young|Old' variable for color coding
pca_df['Young|Old'] = df['Young|Old']

# Plotting the 2D scatter plot of the two principal components with color coding
plt.figure(figsize=(8, 6))
sns.scatterplot(x='PC1', y='PC2', hue='Young|Old', data=pca_df, alpha=0.6)
plt.title('PCA of Crab Variables')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.grid(True)
plt.show()


print("\n\nPCA Loadings")
for pc_idx in range(2):
  for var, loading in list(zip(pca_vars, pca.components_[pc_idx])):
    print('\t' + var + ":", loading)
  print('\n')
```

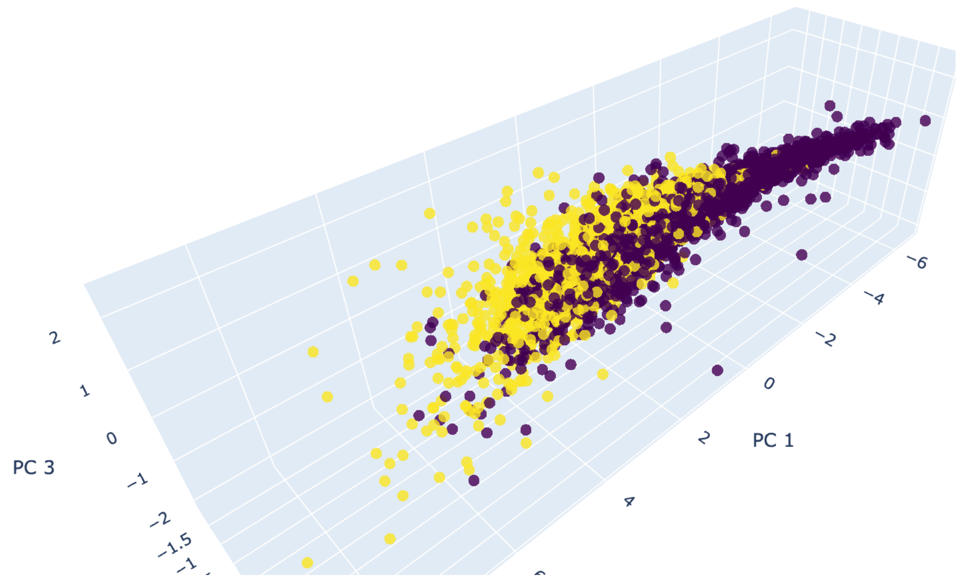


```
PCA Loadings
        Length: 0.3806276567338248
        Diameter: 0.3810108853408206
        Height: 0.36710538729474895
        Weight: 0.3874789211979131
        Shucked Weight: 0.37480648508033465
        Viscera Weight: 0.3782665434524006
        Shell Weight: 0.3761403106207747


        Length: -0.3043014272247193
        Diameter: -0.3390792470684685
        Height: -0.5624537187565186
        Weight: 0.3132958052943968
        Shucked Weight: 0.4984521051456185
        Viscera Weight: 0.3592745479043055
        Shell Weight: 0.019615206918008693
```

3D PCA of Crab Variables Colored By Young|Old

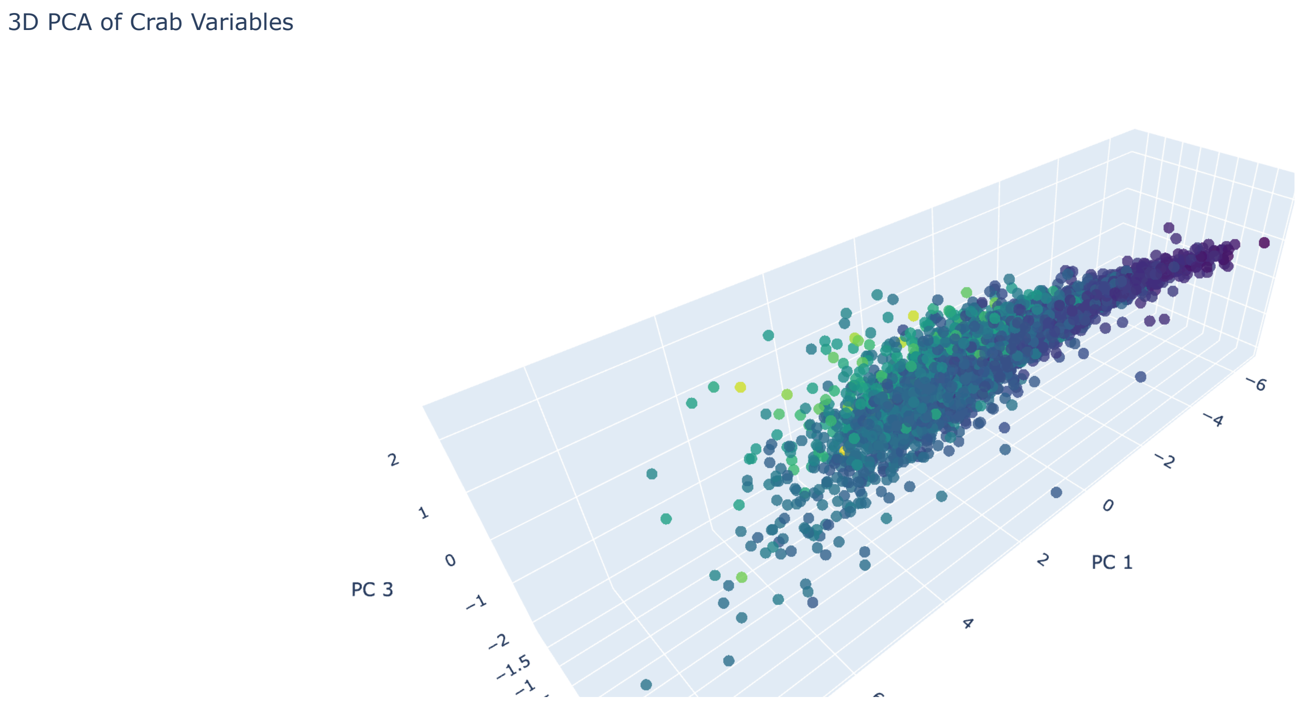
```
#@title 3D PCA of Crab Variables Colored By Young|Old
# Perform PCA to reduce to 3 dimensions for visualization
pca = PCA(n_components=3)
reduced_embeddings = pca.fit_transform(X_pca_scaled)

# Create an interactive 3D scatter plot
fig = go.Figure(data=[go.Scatter3d(
    x=reduced_embeddings[:, 0],
    y=reduced_embeddings[:, 1],
    z=reduced_embeddings[:, 2],
    mode='markers',
    marker=dict(
        size=5,
        color=df['Young|Old'],                # set color to an array/list of desired values
        colorscale='Viridis',        # choose a colorscale
        opacity=0.8
    )
)])

# Update the layout to add titles and axis labels
fig.update_layout(
    title='3D PCA of Crab Variables',
    scene=dict(
        xaxis_title='PC 1',
        yaxis_title='PC 2',
        zaxis_title='PC 3'
    ),
    margin=dict(r=10, l=10, b=10, t=30)
)

# Show the plot
fig.show()
```

3D PCA of Crab Variables



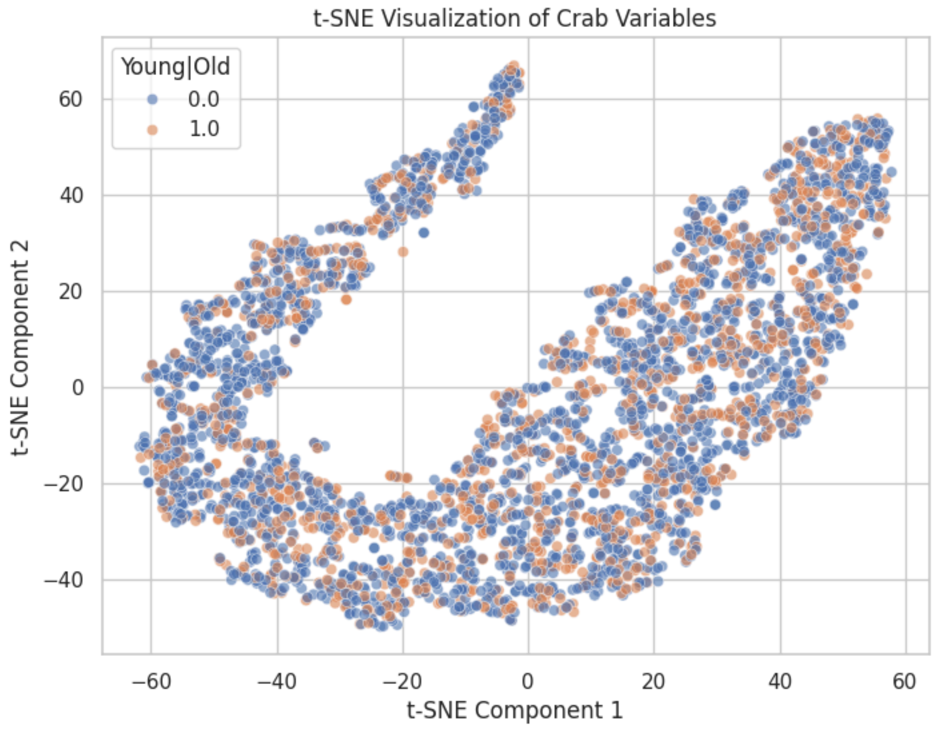
3D PCA of Crab Variables Colored By Age

Show code



t-SNE plot

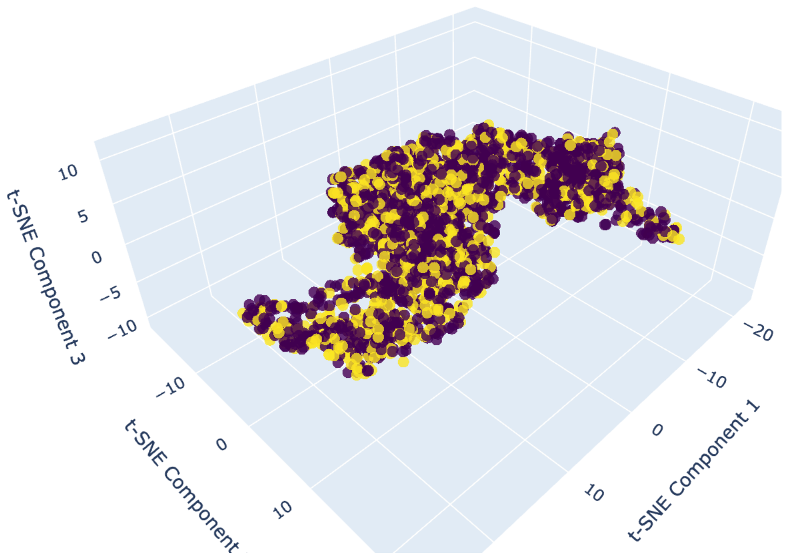
Show code



3D t-SNE plot

```python
#@title 3D t-SNE plot

from sklearn.manifold import TSNE
import plotly.graph_objs as go
import pandas as pd

# Assuming 'df' is your DataFrame and 'X_pca_scaled' is your scaled feature set.

# Setting up t-SNE with 3 components for a 3D visualization
tsne_3d = TSNE(n_components=3, random_state=42)

# Applying t-SNE to the scaled data
tsne_results_3d = tsne_3d.fit_transform(X_pca_scaled)

# Creating a DataFrame for the t-SNE components
tsne_df_3d = pd.DataFrame(data=tsne_results_3d, columns=['TSNE1', 'TSNE2', 'TSNE3'])
tsne_df_3d['Young|Old'] = df['Young|Old']

# Create an interactive 3D scatter plot using plotly
fig = go.Figure(data=[go.Scatter3d(
    x=tsne_df_3d['TSNE1'],
    y=tsne_df_3d['TSNE2'],
    z=tsne_df_3d['TSNE3'],
    mode='markers',
    marker=dict(
        size=5,
        color=tsne_df_3d['Young|Old'],  # Color by 'Young|Old' binary class
        colorscale='Viridis',           # Choose a colorscale
        opacity=0.8
    )
)])

# Update the layout to add titles and axis labels
fig.update_layout(
    title='3D t-SNE Visualization of Crab Variables',
    scene=dict(
        xaxis_title='t-SNE Component 1',
        yaxis_title='t-SNE Component 2',
        zaxis_title='t-SNE Component 3'
    ),
    margin=dict(r=10, l=10, b=10, t=30)
)

# Show the plot
fig.show()
```

3D t-SNE Visualization of Crab Variables



Part 3: Linear Models

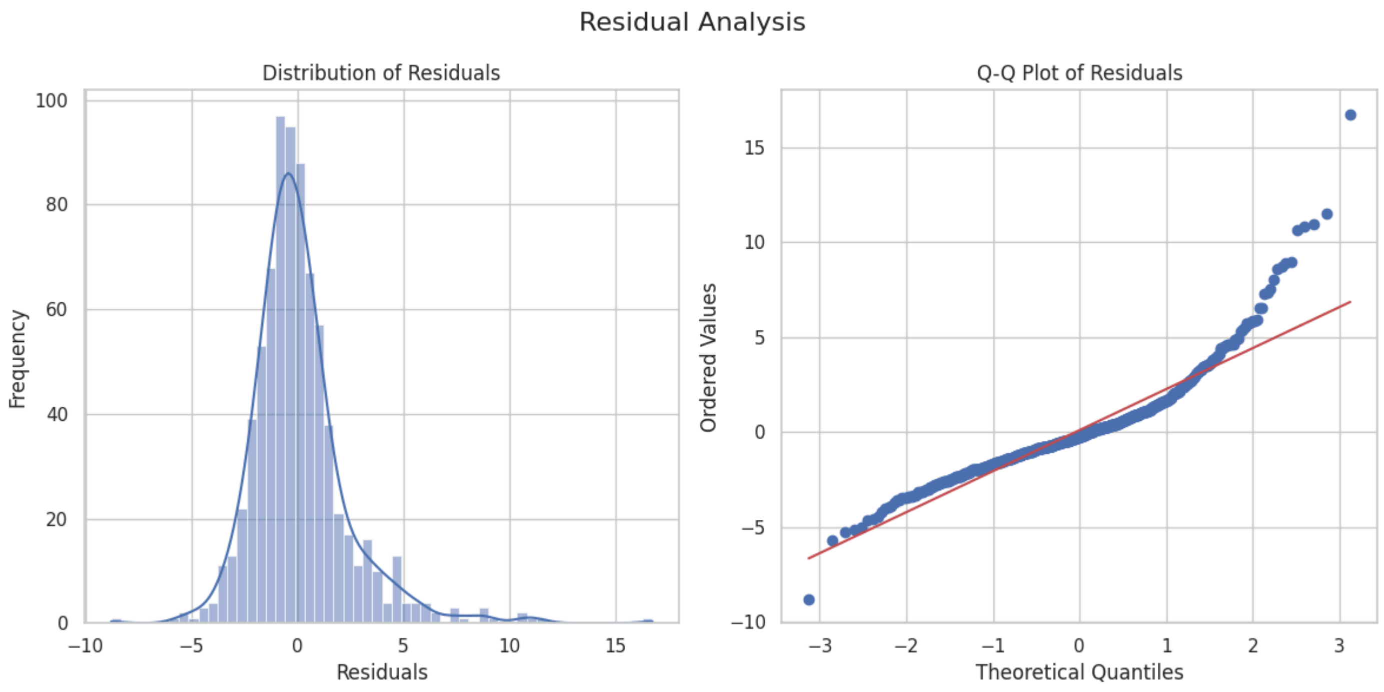
[ ] ↳ 5 cells hidden

Part 4: Tree Models

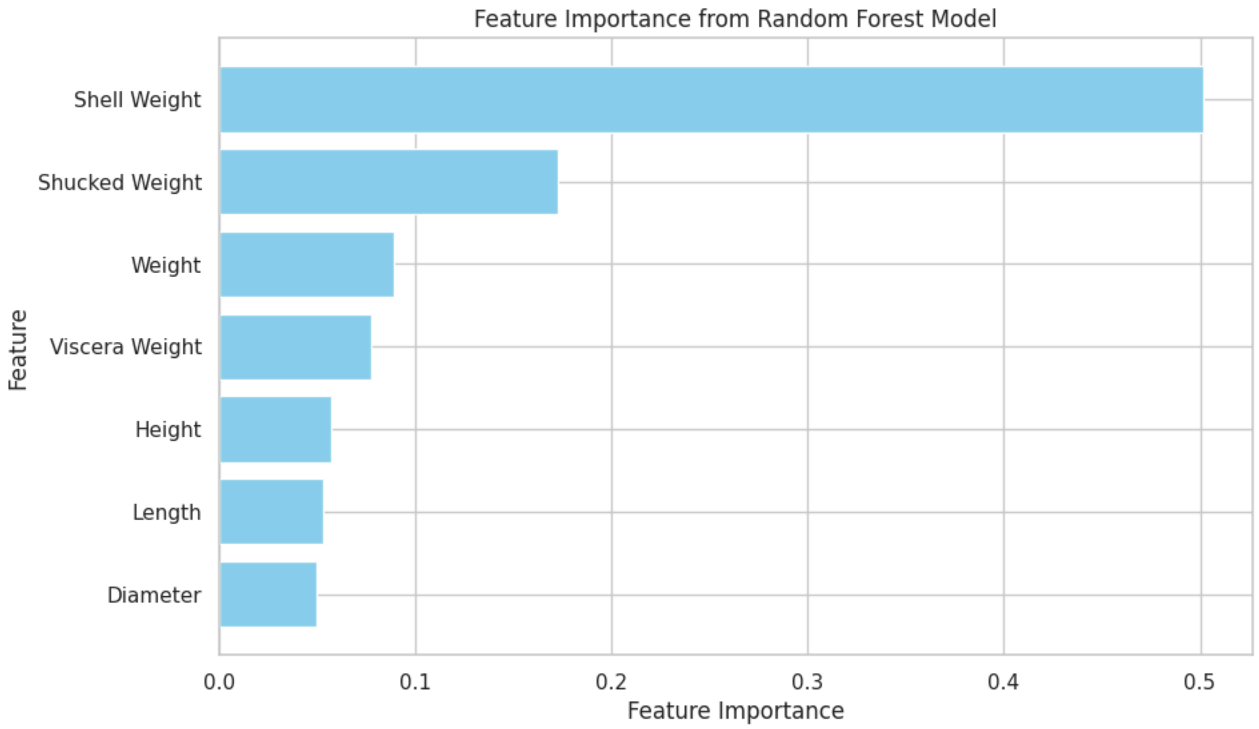
Random Forest Regressor

```python
#@title Random Forest Regressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
from scipy.stats import probplot
from sklearn.metrics import r2_score

# Selecting predictor variables (excluding 'Young|Old')
predictor_vars = ['Length', 'Diameter', 'Height', 'Weight', 'Shucked Weight', 'Viscera Weight', 'Shell Weight']
X_rf = df[predictor_vars]
y_rf = df['Age']

# Splitting the data into training and testing sets
X_train_rf, X_test_rf, y_train_rf, y_test_rf = train_test_split(X_rf, y_rf, test_size=0.2, random_state=42)

# Creating a random forest regressor model
random_forest_model = RandomForestRegressor(n_estimators=100, random_state=42)

# Fitting the model
random_forest_model.fit(X_train_rf, y_train_rf)

# Predicting on the test data
y_pred_rf = random_forest_model.predict(X_test_rf)

# Calculating residuals
residuals_rf = y_test_rf - y_pred_rf

# R^2 score
r2_rf = r2_score(y_test_rf, y_pred_rf)
print("R2:", r2_rf)

# Mean Squared Error
mse_rf = mean_squared_error(y_test_rf, y_pred_rf)
print("MSE:", mse_rf, "\n\n")


# Creating a figure to hold subplots
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(12, 6))
fig.suptitle('Residual Analysis', fontsize=16)

# Plotting the histogram of residuals
sns.histplot(residuals_rf, kde=True, ax=axes[0])
axes[0].set_title('Distribution of Residuals')
axes[0].set_xlabel('Residuals')
axes[0].set_ylabel('Frequency')

# Creating Q-Q plot
probplot(residuals_rf, dist="norm", plot=axes[1])
axes[1].set_title('Q-Q Plot of Residuals')
axes[1].set_xlabel('Theoretical Quantiles')
axes[1].set_ylabel('Ordered Values')

plt.tight_layout()
plt.show()
```

```
R2: 0.5220396296821315
MSE: 5.233945699614891
```



```
# Assuming best_rf_model is your trained RandomForestClassifier or RandomForestRegressor
feature_importances = random_forest_model.feature_importances_
feature_importances_sorted = sorted(zip(predictor_vars, feature_importances), key=lambda x: x[1], reverse=False)
sorted_feature_names, sorted_feature_importances = zip(*feature_importances_sorted)

plt.figure(figsize=(10, 6))
plt.barh(sorted_feature_names, sorted_feature_importances, color='skyblue')
# plt.barh(predictor_vars, feature_importances_sorted, color='skyblue')
plt.xlabel('Feature Importance')
plt.ylabel('Feature')
plt.title('Feature Importance from Random Forest Model')
plt.show()
```



XGBoost Regressor

Show code

```
R2: 0.4689048606432701
MSE: 5.815802508634486
```

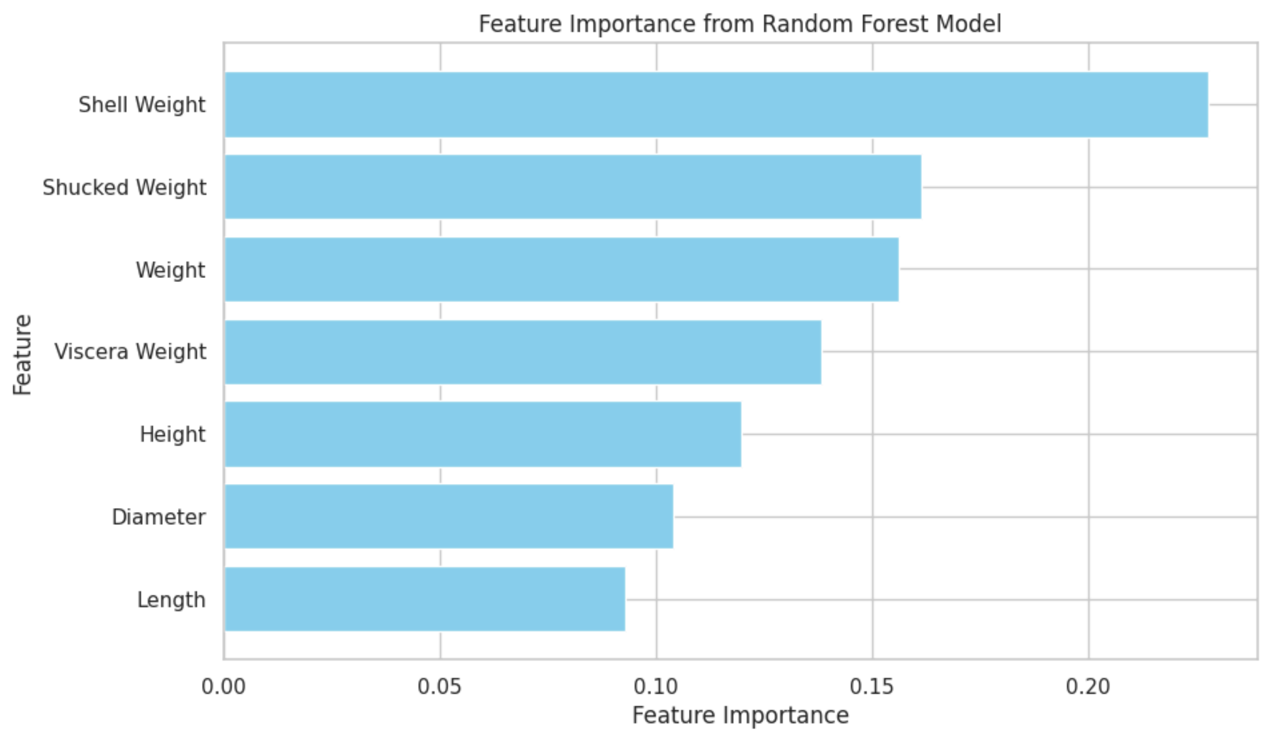
The random forest regressor is the best model yet, even beating the XGBoost regressor.

Random Forest Classification

```python
#@title Random Forest Classification
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, roc_auc_score

# Assuming 'Young|Old' is binary and properly encoded, i.e., 0 for 'Young' and 1 for 'Old'
# Prepare the data
y_classifier = df['Young|Old']

# Split the data into training and testing sets
X_train_class, X_test_class, y_train_class, y_test_class = train_test_split(X_rf, y_classifier, test_size=0.2, random_state=4

# Create a Random Forest Classifier model
random_forest_classifier = RandomForestClassifier(n_estimators=100, random_state=42)

# Fit the model
random_forest_classifier.fit(X_train_class, y_train_class)

# Predict on the test data
y_pred_class = random_forest_classifier.predict(X_test_class)
y_pred_proba_class = random_forest_classifier.predict_proba(X_test_class)[:, 1]  # probabilities for the positive class

# Evaluate the model
accuracy_class = accuracy_score(y_test_class, y_pred_class)
precision_class = precision_score(y_test_class, y_pred_class)
recall_class = recall_score(y_test_class, y_pred_class)
roc_auc_class = roc_auc_score(y_test_class, y_pred_proba_class)

# Residual-like analysis: Difference between actual and predicted probabilities
residuals_class = y_test_class - y_pred_proba_class

print("Accuracy:", accuracy_class)
print("Precision:", precision_class)
print("Recall:", recall_class)
print("ROC AUC:", roc_auc_class)
```

```
Accuracy: 0.7753530166880617
Precision: 0.73568281938326
Recall: 0.5921985815602837
ROC AUC: 0.8449277223625441
```

```python
# Assuming best_rf_model is your trained RandomForestClassifier or RandomForestRegressor
feature_importances = random_forest_classifier.feature_importances_
feature_importances_sorted = sorted(zip(predictor_vars, feature_importances), key=lambda x: x[1], reverse=False)
sorted_feature_names, sorted_feature_importances = zip(*feature_importances_sorted)

plt.figure(figsize=(10, 6))
plt.barh(sorted_feature_names, sorted_feature_importances, color='skyblue')
# plt.barh(predictor_vars, feature_importances_sorted, color='skyblue')
plt.xlabel('Feature Importance')
plt.ylabel('Feature')
plt.title('Feature Importance from Random Forest Model')
plt.show()
```



XGBoost Classification

```
#@title XGBoost Classification
# Prepare the data for classification
y_class = df['Young|Old']
X_train_class, X_test_class, y_train_class, y_test_class = train_test_split(X_rf, y_class, test_size=0.2, random_state=42)

# Create the XGBoost classifier model
xgb_classifier = xgb.XGBClassifier(objective='binary:logistic', n_estimators=100, seed=42, use_label_encoder=False, eval_metr

# Fit the model
xgb_classifier.fit(X_train_class, y_train_class)

# Predict on the test data
y_pred_class = xgb_classifier.predict(X_test_class)
y_pred_proba_class = xgb_classifier.predict_proba(X_test_class)[:, 1]  # probabilities for the positive class

# Evaluate the model
accuracy_class = accuracy_score(y_test_class, y_pred_class)
precision_class = precision_score(y_test_class, y_pred_class)
recall_class = recall_score(y_test_class, y_pred_class)
roc_auc_class = roc_auc_score(y_test_class, y_pred_proba_class)

print("Accuracy:", accuracy_class)
print("Precision:", precision_class)
print("Recall:", recall_class)
print("ROC AUC:", roc_auc_class)
```

```
Accuracy: 0.7573812580231065
Precision: 0.6882591093117408
Recall: 0.6028368794326241
ROC AUC: 0.834375044593804
```

Part 5: Artificial Neural Network

Data preparation

Show code

```
<ipython-input-126-57f296e4ce7b>:3: SettingWithCopyWarning:
```

```
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view
```

DataLoaders

Show code

Model

```
#@title Model
import torch.nn as nn

class FFN(nn.Module):
  def __init__(self, task):
    super(FFN, self).__init__()

    assert task in {"regression", "classification"}

    self.task = task

    self.backbone = nn.ModuleList([nn.BatchNorm1d(8, affine=False)])
    for i in range(5):
      self.backbone.append(nn.Linear(8, 8))
      self.backbone.append(nn.Tanh())
      # self.backbone.append(nn.Dropout(0.2))
    self.backbone.append(nn.Linear(8, 4))

    if task == "regression":
      self.head = nn.Linear(4, 1)
    else:
      self.head = nn.Linear(4, 2)


  def forward(self, x, raw=False):
    for layer in self.backbone:
      x = layer(x)
    x_logits = x
    x = self.head(x_logits)

    if self.task == "classification":
      x = nn.Softmax(dim=1)(x)

    if raw:
      return x, x_logits
    return x
```

Loss function and optimizer

Show code

Train and test functions

Show code

Classification

```
#@title Classification
class_model, class_test_data = trainAndTest("classification")
```

```
Epoch 1
Batch 10: Loss = 0.6867, Accuracy = 65.00%
Batch 20: Loss = 0.6746, Accuracy = 69.69%
Batch 30: Loss = 0.6700, Accuracy = 66.25%
Batch 40: Loss = 0.6583, Accuracy = 66.56%
Batch 50: Loss = 0.6425, Accuracy = 69.06%
Batch 60: Loss = 0.6570, Accuracy = 58.12%
Batch 70: Loss = 0.6247, Accuracy = 66.88%
Batch 80: Loss = 0.6158, Accuracy = 72.50%
Batch 90: Loss = 0.6129, Accuracy = 69.69%
Test Loss: 0.5949, Accuracy: 73.04%


Epoch 2
Batch 10: Loss = 0.5915, Accuracy = 69.69%
Batch 20: Loss = 0.6128, Accuracy = 67.19%
Batch 30: Loss = 0.5931, Accuracy = 70.31%
Batch 40: Loss = 0.5794, Accuracy = 72.81%
Batch 50: Loss = 0.5591, Accuracy = 74.69%
Batch 60: Loss = 0.5724, Accuracy = 70.94%
Batch 70: Loss = 0.5543, Accuracy = 73.44%
Batch 80: Loss = 0.5917, Accuracy = 70.31%
Batch 90: Loss = 0.5777, Accuracy = 70.31%
Test Loss: 0.5659, Accuracy: 74.58%


Epoch 3
Batch 10: Loss = 0.5434, Accuracy = 75.62%
Batch 20: Loss = 0.5478, Accuracy = 75.31%
Batch 30: Loss = 0.5965, Accuracy = 69.69%
Batch 40: Loss = 0.5666, Accuracy = 73.44%
Batch 50: Loss = 0.5483, Accuracy = 74.69%
Batch 60: Loss = 0.5623, Accuracy = 73.12%
Batch 70: Loss = 0.5544, Accuracy = 74.06%
Batch 80: Loss = 0.5720, Accuracy = 72.19%
Batch 90: Loss = 0.5334, Accuracy = 77.19%
Test Loss: 0.5465, Accuracy: 76.51%


Epoch 4
Batch 10: Loss = 0.5283, Accuracy = 78.12%
Batch 20: Loss = 0.5423, Accuracy = 75.94%
Batch 30: Loss = 0.5279, Accuracy = 77.50%
Batch 40: Loss = 0.5765, Accuracy = 71.25%
Batch 50: Loss = 0.5404, Accuracy = 75.94%
Batch 60: Loss = 0.5561, Accuracy = 73.44%
Batch 70: Loss = 0.5458, Accuracy = 75.31%
Batch 80: Loss = 0.5306, Accuracy = 76.88%
Batch 90: Loss = 0.5455, Accuracy = 76.25%
Test Loss: 0.5369, Accuracy: 76.89%


Epoch 5
Batch 10: Loss = 0.5460, Accuracy = 75.62%
Batch 20: Loss = 0.5245, Accuracy = 76.88%
Batch 30: Loss = 0.5317, Accuracy = 76.56%
Batch 40: Loss = 0.5309, Accuracy = 77.81%
Batch 50: Loss = 0.5264, Accuracy = 76.88%
```

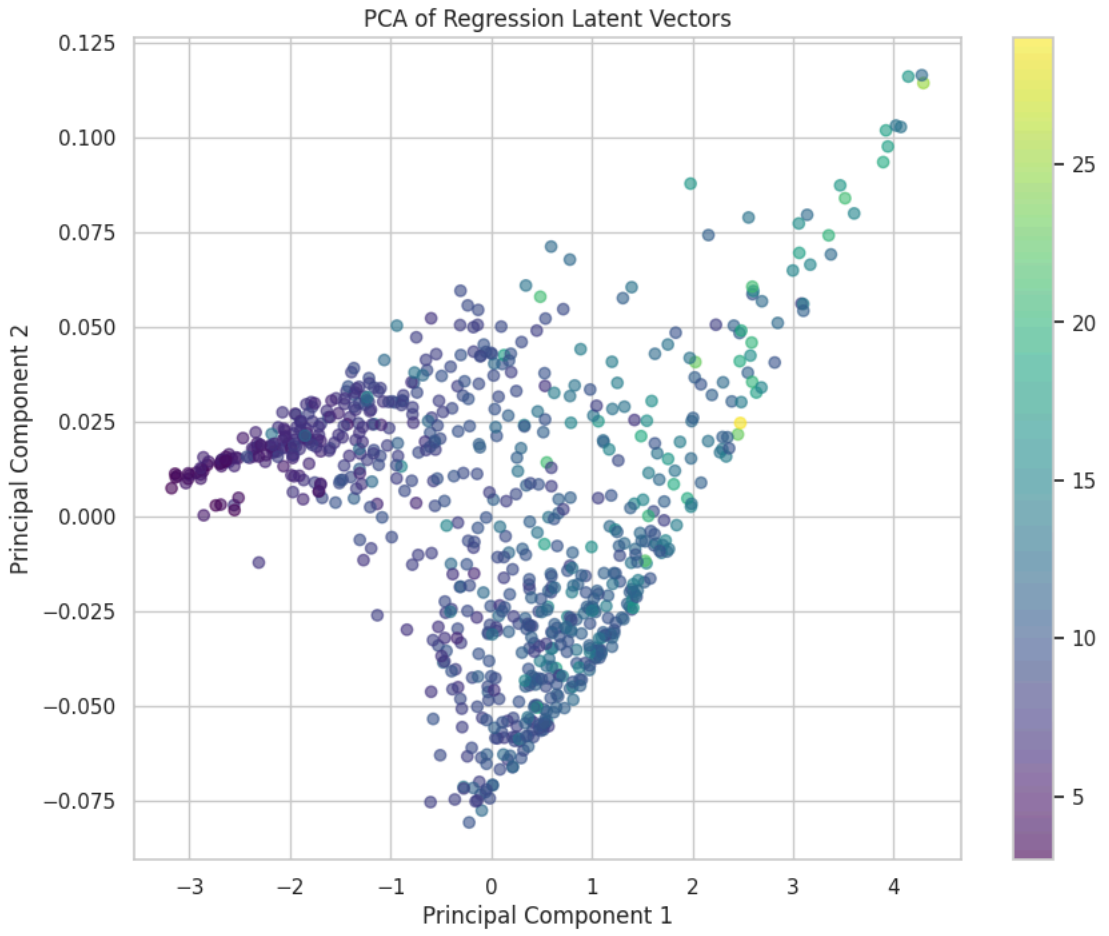
Regression

```
#@title Regression
reg_model, reg_test_data = trainAndTest("regression", 50)
```

```
Epoch 1
Batch 10: Loss = 91.1427
Batch 20: Loss = 91.2421
Batch 30: Loss = 94.0418
Batch 40: Loss = 90.6227
Batch 50: Loss = 87.9925
Batch 60: Loss = 79.3302
Batch 70: Loss = 82.3218
Batch 80: Loss = 71.0317
Batch 90: Loss = 72.8962
Test Loss: 58.3384


Epoch 2
Batch 10: Loss = 58.8409
Batch 20: Loss = 60.1046
Batch 30: Loss = 42.6817
Batch 40: Loss = 40.2391
Batch 50: Loss = 35.5226
Batch 60: Loss = 29.0844
Batch 70: Loss = 28.8568
Batch 80: Loss = 29.2687
Batch 90: Loss = 24.2316
Test Loss: 21.1164


Epoch 3
Batch 10: Loss = 20.4493
Batch 20: Loss = 15.2075
Batch 30: Loss = 14.2824
Batch 40: Loss = 15.0294
Batch 50: Loss = 14.6226
Batch 60: Loss = 12.1941
Batch 70: Loss = 15.1486
Batch 80: Loss = 12.5043
Batch 90: Loss = 11.0391
Test Loss: 10.0775


Epoch 4
Batch 10: Loss = 10.3545
Batch 20: Loss = 9.0284
Batch 30: Loss = 9.5050
Batch 40: Loss = 9.2756
Batch 50: Loss = 10.5331
Batch 60: Loss = 7.9792
Batch 70: Loss = 7.2549
Batch 80: Loss = 9.2489
Batch 90: Loss = 8.0763
Test Loss: 7.8557


Epoch 5
Batch 10: Loss = 7.9468
Batch 20: Loss = 5.3029
Batch 30: Loss = 6.9176
Batch 40: Loss = 7.0757
Batch 50: Loss = 8.2167
```

PCA on the neural network regression latent vectors

```python
#@title PCA on the neural network regression latent vectors
embeddings = []
labels = []
for data, label in reg_test_data:
  _, embedding = reg_model(data, raw=True)
  embeddings.append(embedding)
  labels.append(label)
embeddings = torch.cat(embeddings, axis=0)
labels = torch.cat(labels, axis=0)

# Detach the tensor and convert to NumPy for PCA processing
if embeddings.requires_grad:
    embeddings_np = embeddings.detach().cpu().numpy()
else:
    embeddings_np = embeddings.cpu().numpy()

# Perform PCA to reduce to 2 dimensions for visualization
pca = PCA(n_components=2)
reduced_embeddings = pca.fit_transform(embeddings_np)

# Plotting the embeddings with labels as color codes
plt.figure(figsize=(10, 8))
scatter = plt.scatter(reduced_embeddings[:, 0], reduced_embeddings[:, 1], c=labels, cmap='viridis', alpha=0.6)

plt.colorbar(scatter)  # Show color scale
plt.title('PCA of Regression Latent Vectors')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.grid(True)
plt.show()
```



Interactive 3D projection of regression latent vectors