In [ ]:

```
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 StandardScaler

np.random.seed(10)
```

In [ ]:

```
data=pd.read_csv('HWs/HW2/Files/data/abalone.csv')
```

Question 1

Your goal is to predict abalone age, which is calculated as the number ofrings plus 1.5. Notice there currently is no age variable in the data set. Add age to the data set

In [ ]:

```
data.head()
```

Out[ ]:

	type	longest_shell	diameter	height	whole_weight	shucked_weight	viscera_weight	shell_weight	ring
0	M	0.455	0.365	0.095	0.5140	0.2245	0.1010	0.150	1
1	M	0.350	0.265	0.090	0.2255	0.0995	0.0485	0.070	
2	F	0.530	0.420	0.135	0.6770	0.2565	0.1415	0.210	
3	M	0.440	0.365	0.125	0.5160	0.2155	0.1140	0.155	1
4	I	0.330	0.255	0.080	0.2050	0.0895	0.0395	0.055	

In [ ]:

```
data['age'] = data['rings'] + 1.5
data.head()
```

Out[ ]:

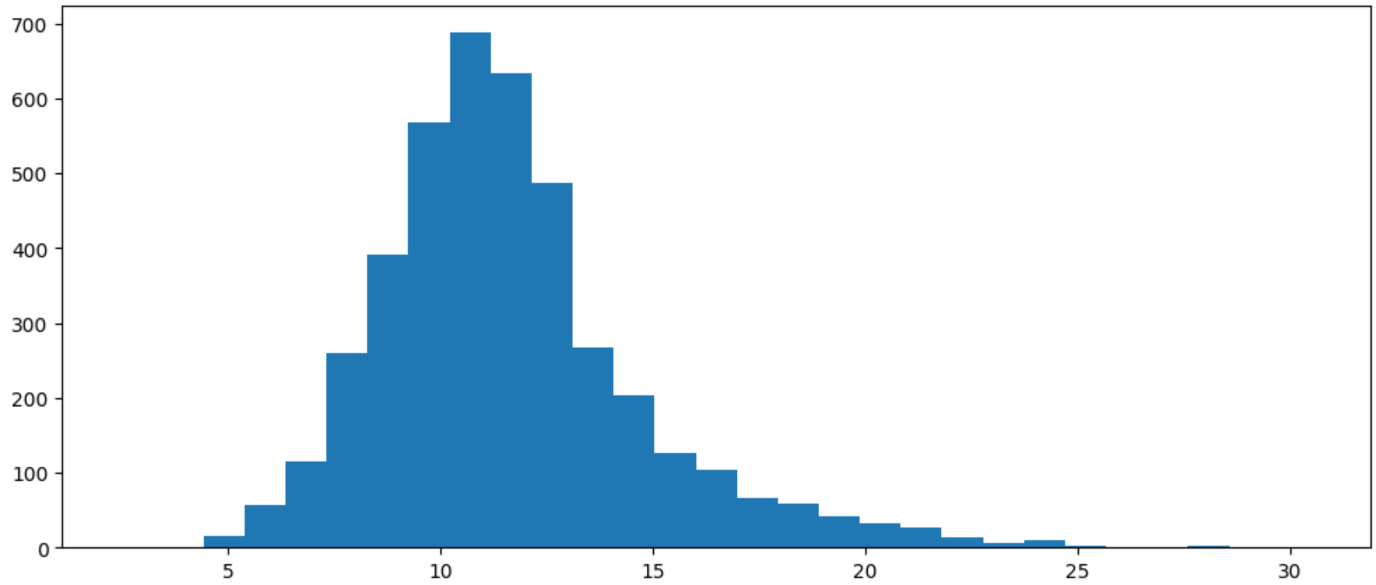
	type	longest_shell	diameter	height	whole_weight	shucked_weight	viscera_weight	shell_weight	ring
0	M	0.455	0.365	0.095	0.5140	0.2245	0.1010	0.150	1
1	M	0.350	0.265	0.090	0.2255	0.0995	0.0485	0.070	
2	F	0.530	0.420	0.135	0.6770	0.2565	0.1415	0.210	
3	M	0.440	0.365	0.125	0.5160	0.2155	0.1140	0.155	1
4	I	0.330	0.255	0.080	0.2050	0.0895	0.0395	0.055	

Assess and describe the distribution of age

The distribution of age seems normal but slightly skewed right.

In [ ]:

```
plt.figure(figsize=(12,5))
plt.hist(data['age'], bins=29)
plt.show()
```



Question 2

I will complete this exercise after question 3 when features are added and data is centered, scaled.

Question 3

Using the training data, create a recipe predicting the outcome variable, age , with all other predictor variables. Note that you should not include rings to predict age . Explain why you shouldn't use rings to predict age.

You shouldn't use rings to predict age, since rings is a linear function of age. This would allow a model to fit perfectly.

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4177 entries, 0 to 4176
Data columns (total 10 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   type            4177 non-null   object 
 1   longest_shell   4177 non-null   float64
 2   diameter        4177 non-null   float64
 3   height          4177 non-null   float64
 4   whole_weight    4177 non-null   float64
 5   shucked_weight  4177 non-null   float64
 6   viscera_weight  4177 non-null   float64
 7   shell_weight    4177 non-null   float64
 8   rings           4177 non-null   int64  
 9   age             4177 non-null   float64
dtypes: float64(8), int64(1), object(1)
memory usage: 326.5+ KB
```

We need to encode the "type" feature since it is categorical:

In [ ]:

```python
data = pd.get_dummies(data, columns=['type'],dtype=int, drop_first=True)
data.head()
```

Out[ ]:

	longest_shell	diameter	height	whole_weight	shucked_weight	viscera_weight	shell_weight	rings	ag
0	0.455	0.365	0.095	0.5140	0.2245	0.1010	0.150	15	16.
1	0.350	0.265	0.090	0.2255	0.0995	0.0485	0.070	7	8.
2	0.530	0.420	0.135	0.6770	0.2565	0.1415	0.210	9	10.
3	0.440	0.365	0.125	0.5160	0.2155	0.1140	0.155	10	11.
4	0.330	0.255	0.080	0.2050	0.0895	0.0395	0.055	7	8.

We are to make interaction terms between type and shucked_weight, longest_shell and diameter, and shucked_weight and shell_weight.

In [ ]:

```python
data['shucked_weight*longest_shell'] = data['shucked_weight'] * data['longest_shell']
data['longest_shell*diamter'] = data['longest_shell'] * data['diameter']
data['shucked_weight*shell_weight'] = data['shucked_weight'] * data['shell_weight']
```

Next we will center and scale all predictors.

In [ ]:

```python
scaler = StandardScaler()
scaled_array = scaler.fit_transform(data)
scaled_data = pd.DataFrame(scaled_array, columns = data.columns)
scaled_data.head()
```

Out[ ]:

	longest_shell	diameter	height	whole_weight	shucked_weight	viscera_weight	shell_weight	r
0	-0.574558	-0.432149	-1.064424	-0.641898	-0.607685	-0.726212	-0.638217	1.57
1	-1.448986	-1.439929	-1.183978	-1.230277	-1.170910	-1.205221	-1.212987	-0.91
2	0.050033	0.122130	-0.107991	-0.309469	-0.463500	-0.356690	-0.207139	-0.28
3	-0.699476	-0.432149	-0.347099	-0.637819	-0.648238	-0.607600	-0.602294	0.02
4	-1.615544	-1.540707	-1.423087	-1.272086	-1.215968	-1.287337	-1.320757	-0.91

In [ ]:

```python
X = data.drop(['rings','age'], axis=1)
Y = data['age']
X_train, X_test, Y_train, Y_test = train_test_split(
    X, Y, test_size=.25
)
```

Question 4

In [ ]:

```python
from sklearn.linear_model import LinearRegression
```

In [ ]:

```python
lm = LinearRegression()
```

Question 5

In [ ]:
```
from sklearn.neighbors import KNeighborsRegressor
```

In [ ]:
```
KNN = KNeighborsRegressor(n_neighbors=7)
```

Question 6

In [ ]:
```
lm.fit(X_train,Y_train);
KNN.fit(X_train,Y_train);
```

Question 7

In [ ]:
```
dict = {
        'longest_shell': .5,
        'diameter': .1,
        'height': .3,
        'whole_weight': 4,
        'shucked_weight': 1,
        'viscera_weight': 2,
        'shell_weight': 1,
        'type_I': 0,
        'type_M': 0
    }

pred_lm = pd.DataFrame(dict, index=[0])
pred_lm['shucked_weight*longest_shell'] = pred_lm['shucked_weight'] * pred_lm['longest_s
pred_lm['longest_shell*diamter'] = pred_lm['longest_shell'] * pred_lm['diameter']
pred_lm['shucked_weight*shell_weight'] = pred_lm['shucked_weight'] * pred_lm['shucked_we

pred_lm
```

Out[ ]:

	longest_shell	diameter	height	whole_weight	shucked_weight	viscera_weight	shell_weight	type_I	ty
0	0.5	0.1	0.3	4	1	2	1	0	

In [ ]:
```
lm.predict(pred_lm)[0]
```

Out[ ]:
```
10.544456063970292
```

Question 8

In [ ]:
```
from sklearn.metrics import mean_absolute_error, r2_score, mean_squared_error
```

In [ ]:
```
pred_lm = lm.predict(X_test)
rmse_lm  = mean_squared_error(pred_lm,Y_test)**.5
r2_lm = r2_score(pred_lm, Y_test)
mae_lm = mean_absolute_error(pred_lm, Y_test)
print("Linear Regression Metrics:")
print(f"RMSE: {round(rmse_lm,2)}, R^2: {round(r2_lm,4)}, MAE: {round(mae_lm,2)}")


pred_KNN = KNN.predict(X_test)
rmse_knn = mean_squared_error(pred_KNN,Y_test)**.5
```

```
r2_knn = r2_score(pred_KNN, Y_test)
mae_knn = mean_absolute_error(pred_KNN, Y_test)
print("KNN Metrics:")
print(f"RMSE: {round(rmse_knn,2)}, R^2: {round(r2_knn,4)}, MAE: {round(mae_knn,2)}")
```

```
Linear Regression Metrics:
RMSE: 2.15, R^2: 0.2842, MAE: 1.51
KNN Metrics:
RMSE: 2.16, R^2: 0.2092, MAE: 1.51
```

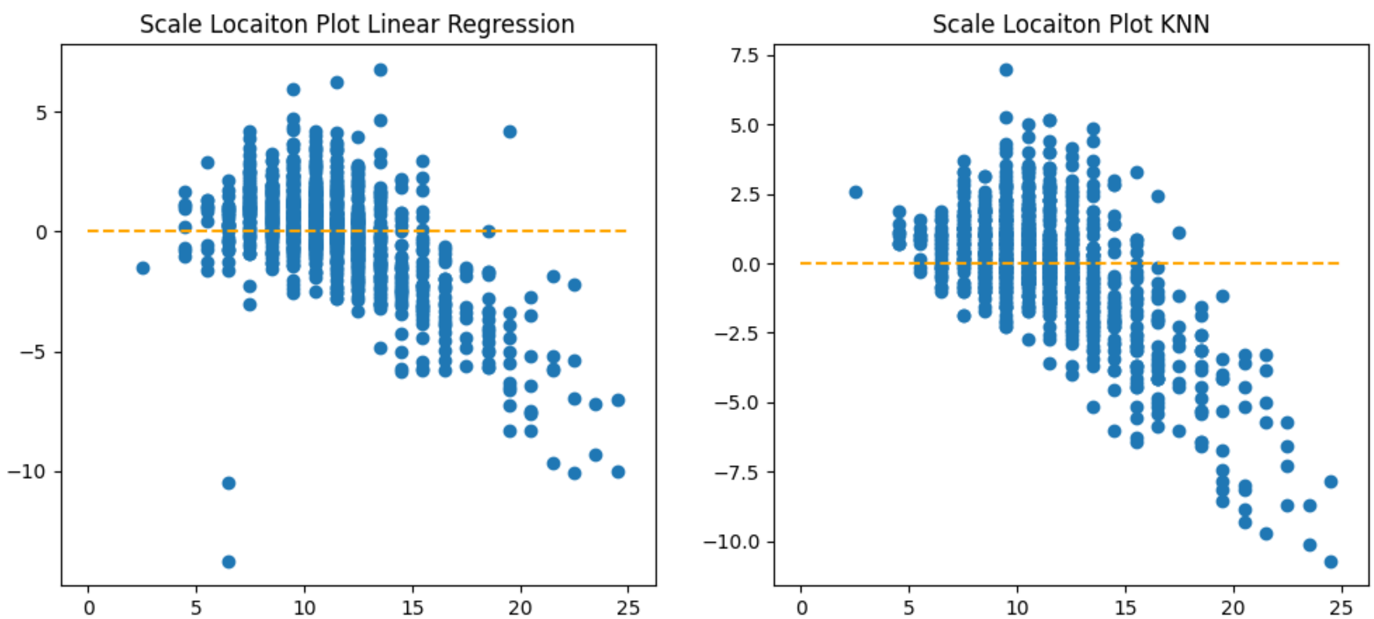
The R^2 values indicate that 28% and 21% of the total variance in the data was captured by the linear and KNN models, respectively.

In [ ]:
```
residuals_lm = pred_lm - Y_test
residuals_knn = pred_KNN - Y_test

fig, ax = plt.subplots(1,2, figsize=(12,5))

ax[0].scatter(Y_test,residuals_lm)
ax[0].plot(range(26), [0]*len(range(26)), ls='--', color='orange')
ax[0].set_title('Scale Locaiton Plot Linear Regression')

ax[1].scatter(Y_test,residuals_knn)
ax[1].plot(range(26), [0]*len(range(26)), ls='--', color='orange')
ax[1].set_title('Scale Locaiton Plot KNN')
plt.show()
```



In [ ]:

Question 9

The models showed almost identical performance based on RMSE and MAE metrics, but the linear model performed noticeably better as shown by the R^2 metric. It is surprising to see that the models could have such similar performance based on 2 metrics, yet so different for a third. I would have expected all 3 to be consistently close.

In lecture, we presented the general bias-variance tradeoff, which takes the form:

$$E[(y_0 - \hat{f}(x_0))^2] = Var(\hat{f}(x_0)) + [Bias(\hat{f}(x_0))]^2 + Var(\epsilon)$$

where the underlying model $Y = f(X) + \epsilon$ satisfies the following:

- ϵ is a zero-mean random noise term and X is non-random (all randomness in Y comes from ϵ);
- (x_0, y_0) represents a test observation, independent of the training set, drawn from the same model;
- $\hat{f}(.)$ is the estimate of f obtained from the training set.

Question 10

Which term(s) in the bias-variance tradeoff above represent the reducible error? Which term(s) represent the irreducible error?

Reducible error comes from variance and bias, while irreducible error is the variance of the random noise term.

Question 11

Using the bias-variance tradeoff above, demonstrate that the expected test error is always at least as large as the irreducible error.

Clearly, the RHS (the expected test error) cannot be less than the irreducible error. In some cases, it is possible for the first two terms to be zero, but as its name suggests, irreducible error is non-zero in almost all situations and presents a lower bound for the RHS.

Question 12

Prove the bias-variance tradeoff

Proof of Bias/Variance tradeoff:

$$E[(Y-\hat{f})^2] = \overset{a}{E[Y^2]} - 2\overset{b}{E[Y\hat{f}]} + \overset{c}{E[\hat{f}^2]}$$

$$a = E[(f+\epsilon)^2] = E[\underbrace{f^2}_{\text{not RV}}] + 2E[\underbrace{f}\epsilon] + E[\epsilon^2] \qquad E[\epsilon]=0 \Rightarrow E[\epsilon^2] = Var[\epsilon]$$

$$= f^2 + 2fE[\epsilon] + Var[\epsilon] = f^2 + Var[\epsilon]$$

$$b \quad -2E[Y\hat{f}] = -2E[\hat{f}(f+\epsilon)] = -2E[\hat{f}f] - 2E[\underbrace{\hat{f}\epsilon}_{\text{idpt}}] = -2fE[\hat{f}] - 2E[\hat{f}]\overset{0}{E[\epsilon]}$$

$$= -2fE[\hat{f}]$$

$$c \quad E[\hat{f}^2] = Var[\hat{f}] + E[\hat{f}]^2$$

$$a + b + c = \underbrace{f^2 - 2fE[\hat{f}] + E[\hat{f}]^2}_{(f-E[\hat{f}])^2} + Var[\hat{f}] + Var[\epsilon]$$

$$= Bias(\hat{f})^2 + Var[\hat{f}] + Var[\epsilon]$$