Abalone Age Prediction

```
from google.colab import drive

drive.mount("/content/drive")
```

```
     Mounted at /content/drive
```

```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns
```

```
df = pd.read_csv("/content/drive/My Drive/Colab Notebooks/Assignment 4/abalone.csv")
```

```
df.head()
```

	Sex	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	Shell weight	Rings
0	M	0.455	0.365	0.095	0.5140	0.2245	0.1010	0.150	15
1	M	0.350	0.265	0.090	0.2255	0.0995	0.0485	0.070	7
2	F	0.530	0.420	0.135	0.6770	0.2565	0.1415	0.210	9
3	M	0.440	0.365	0.125	0.5160	0.2155	0.1140	0.155	10
4	I	0.330	0.255	0.080	0.2050	0.0895	0.0395	0.055	7

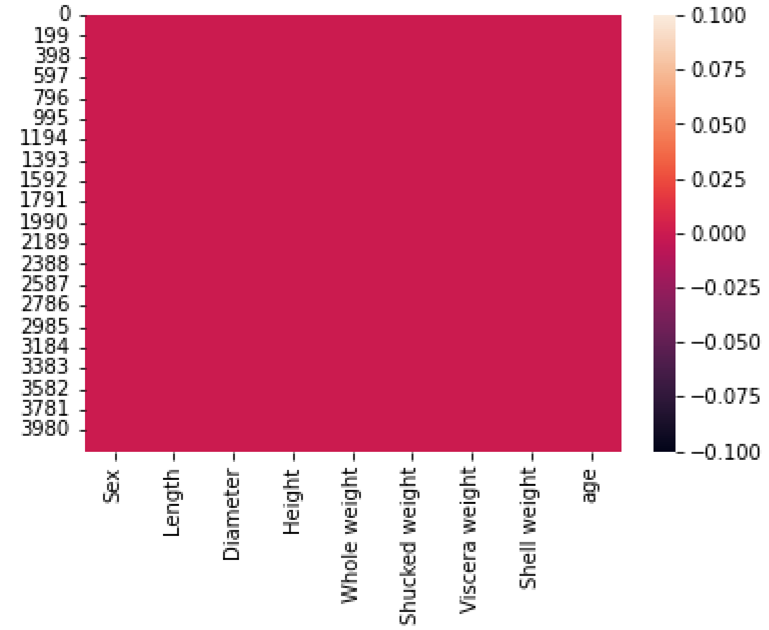
```
df.describe()
```

	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight

```
df['age'] = df['Rings']+1.5
df = df.drop('Rings', axis = 1)
```

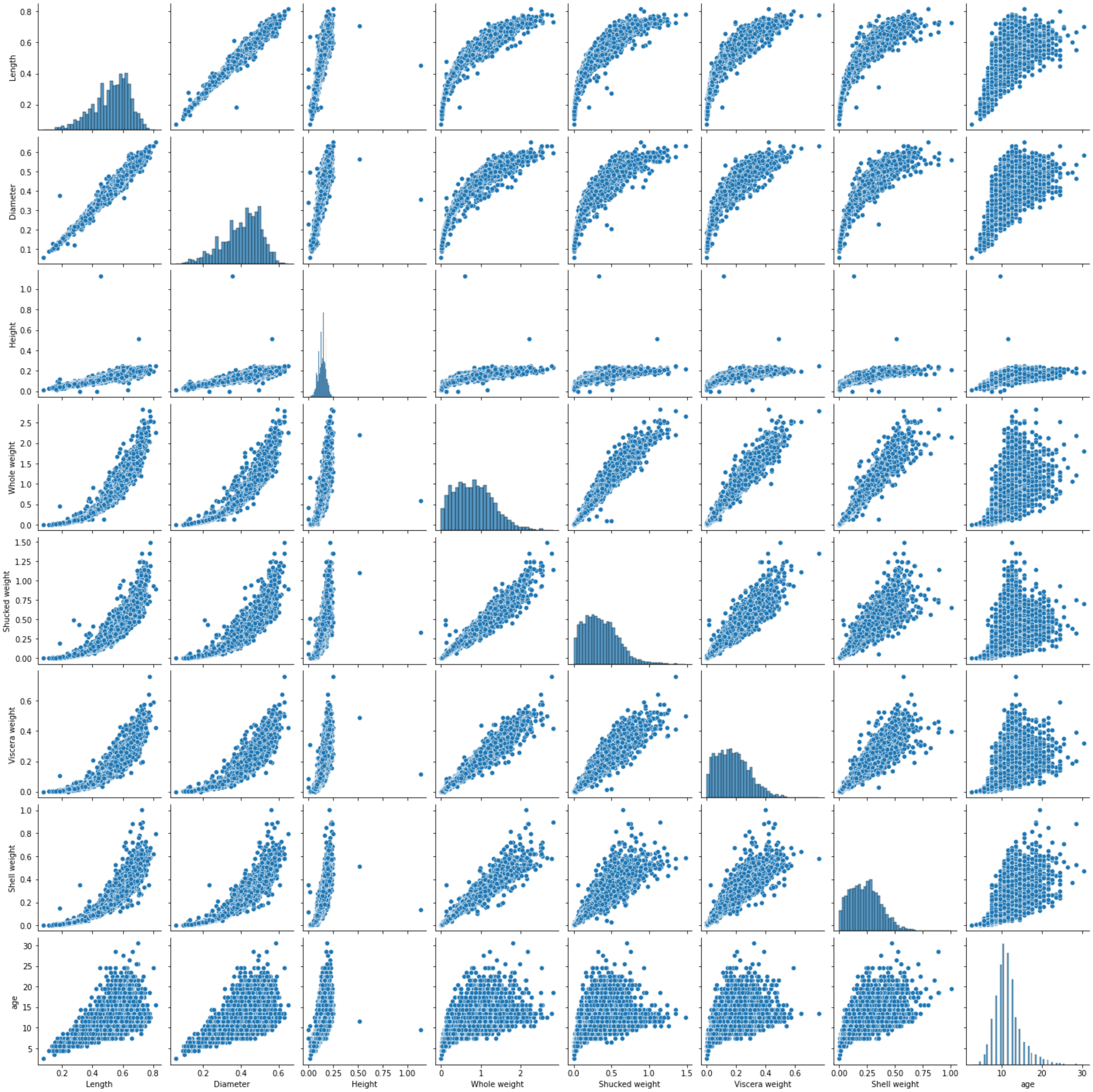
```
sns.heatmap(df.isnull())
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1d06a4be90>
```



```
sns.pairplot(df)
```

```
<seaborn.axisgrid.PairGrid at 0x7f1d040c1590>
```



```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4177 entries, 0 to 4176
Data columns (total 9 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Sex             4177 non-null   object 
 1   Length          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   age             4177 non-null   float64
dtypes: float64(8), object(1)
memory usage: 293.8+ KB
```

```
numerical_features = df.select_dtypes(include = [np.number]).columns
categorical_features = df.select_dtypes(include = [np.object]).columns
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: DeprecationWarning: `np
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/rele
```

```
numerical_features
```

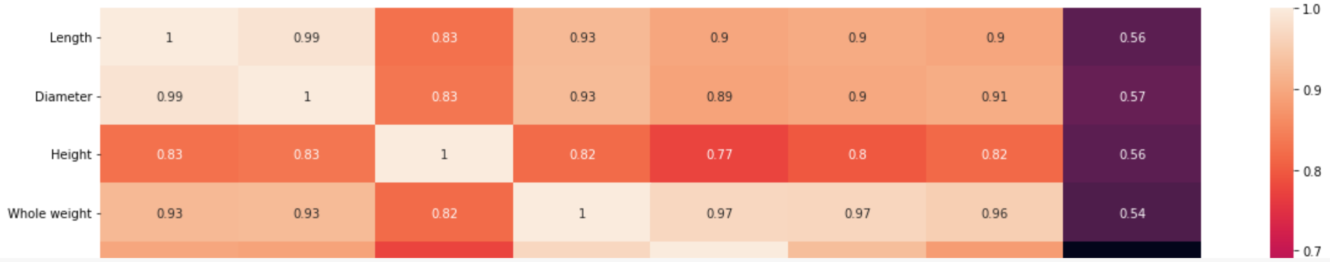
```
Index(['Length', 'Diameter', 'Height', 'Whole weight', 'Shucked weight',
       'Viscera weight', 'Shell weight', 'age'],
      dtype='object')
```

```
categorical_features
```

```
Index(['Sex'], dtype='object')
```

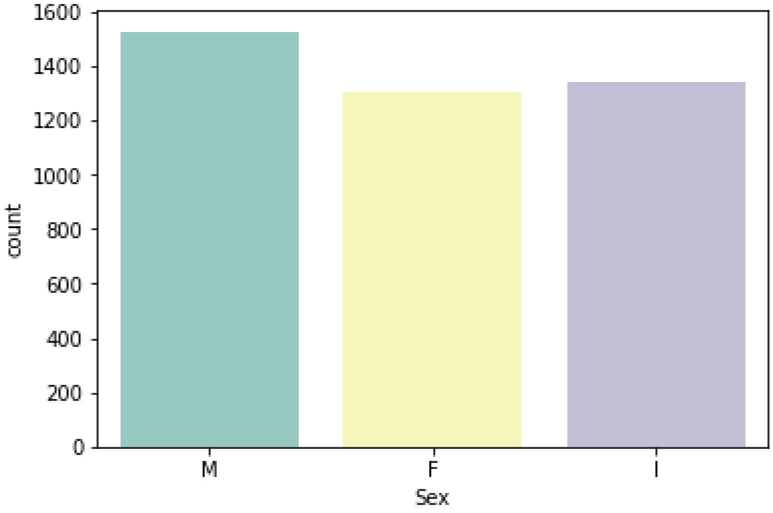
```
plt.figure(figsize = (20,7))
sns.heatmap(df[numerical_features].corr(),annot = True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1d02b4fa90>
```



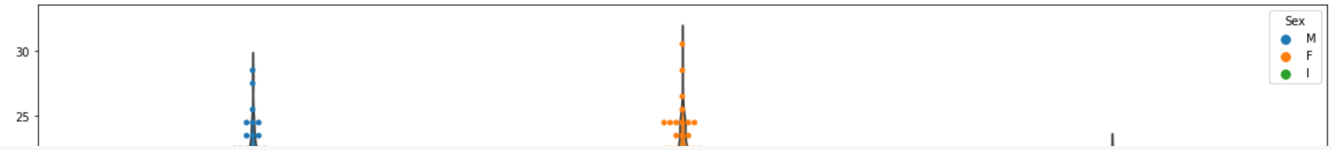
```
sns.countplot(x = 'Sex', data = df, palette = 'Set3')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1d00478c50>
```



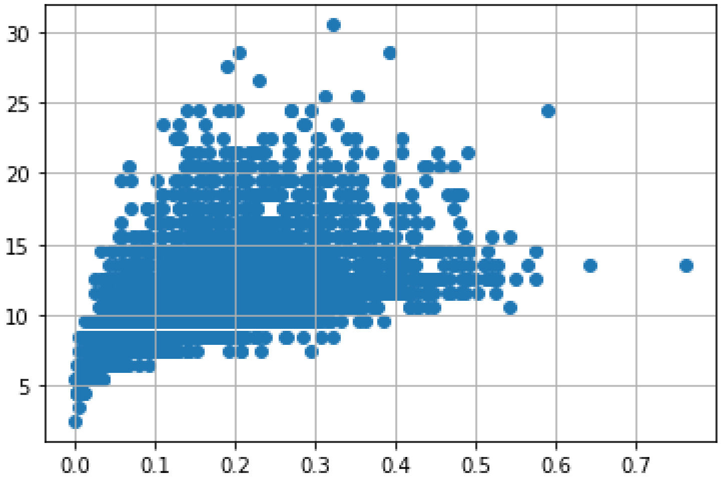
```
plt.figure(figsize = (20,7))
sns.swarmplot(x = 'Sex', y = 'age', data = df, hue = 'Sex')
sns.violinplot(x = 'Sex', y = 'age',data = df)
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/categorical.py:1296: UserWarning: 56.2% o
  warnings.warn(msg, UserWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/categorical.py:1296: UserWarning: 52.2% o
  warnings.warn(msg, UserWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/categorical.py:1296: UserWarning: 58.5% o
  warnings.warn(msg, UserWarning)
<matplotlib.axes._subplots.AxesSubplot at 0x7f1d003e7cd0>
```



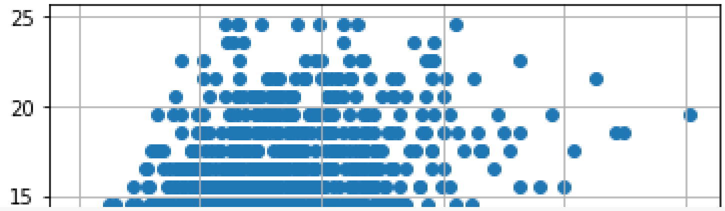
```
df = pd.get_dummies(df)
dummy_df = df
```

```
var = 'Viscera weight'
plt.scatter(x = df[var], y = df['age'])
plt.grid(True)
```



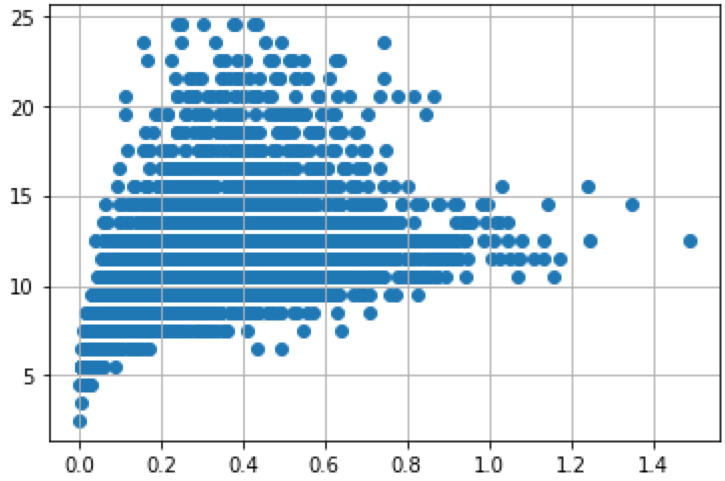
```
df.drop(df[(df['Viscera weight'] > 0.5) &
          (df['age'] < 20)].index, inplace = True)
df.drop(df[(df['Viscera weight']<0.5) & (
df['age'] > 25)].index, inplace = True)
```

```
var = 'Shell weight'
plt.scatter(x = df[var], y = df['age'])
plt.grid(True)
```



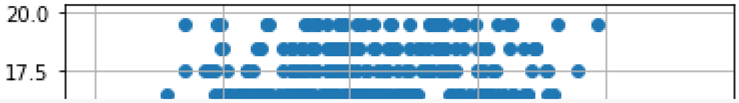
```
df.drop(df[(df['Shell weight'] > 0.6) &
          (df['age'] < 25)].index, inplace = True)
df.drop(df[(df['Shell weight']<0.8) & (
df['age'] > 25)].index, inplace = True)
```

```
var = 'Shucked weight'
plt.scatter(x = df[var], y = df['age'])
plt.grid(True)
```



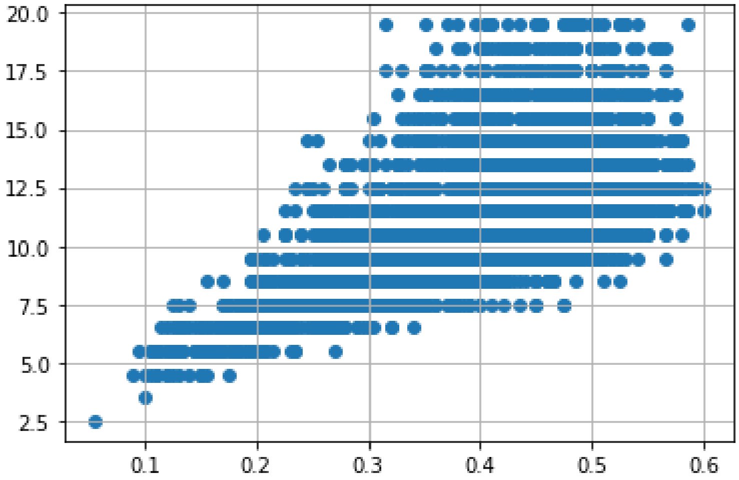
```
df.drop(df[(df['Shucked weight'] >= 1) &
          (df['age'] < 20)].index, inplace = True)
df.drop(df[(df['Viscera weight']<1) & (
df['age'] > 20)].index, inplace = True)
```

```
var = 'Whole weight'
plt.scatter(x = df[var], y = df['age'])
plt.grid(True)
```



```
df.drop(df[(df['Whole weight'] >= 2.5) &
          (df['age'] < 25)].index, inplace = True)
df.drop(df[(df['Whole weight']<2.5) & (
df['age'] > 25)].index, inplace = True)
```

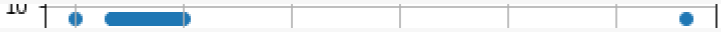
```
var = 'Diameter'
plt.scatter(x = df[var], y = df['age'])
plt.grid(True)
```



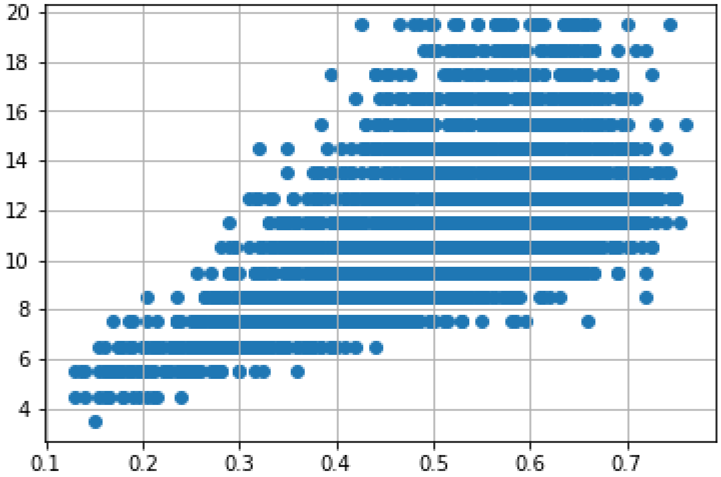
```
df.drop(df[(df['Diameter'] <0.1) &
          (df['age'] < 5)].index, inplace = True)
df.drop(df[(df['Diameter']<0.6) & (
df['age'] > 25)].index, inplace = True)
df.drop(df[(df['Diameter']>=0.6) & (
df['age'] < 25)].index, inplace = True)
```

```
var = 'Height'
plt.scatter(x = df[var], y = df['age'])
plt.grid(True)
```

```
df.drop(df[(df['Height'] > 0.4) &
          (df['age'] < 15)].index, inplace = True)
df.drop(df[(df['Height']<0.4) & (
df['age'] > 25)].index, inplace = True)
```



```
var = 'Length'
plt.scatter(x = df[var], y = df['age'])
plt.grid(True)
```



```
df.drop(df[(df['Length'] <0.1) &
          (df['age'] < 5)].index, inplace = True)
df.drop(df[(df['Length']<0.8) & (
df['age'] > 25)].index, inplace = True)
df.drop(df[(df['Length']>=0.8) & (
df['age'] < 25)].index, inplace = True)
```

```
X = df.drop('age', axis = 1)
y = df['age']
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.feature_selection import SelectKBest
```

```
standardScale = StandardScaler()
standardScale.fit_transform(X)

selectkBest = SelectKBest()
X_new = selectkBest.fit_transform(X, y)

X_train, X_test, y_train, y_test = train_test_split(X_new, y, test_size = 0.25)
```

Linear Regression

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

```
lm = LinearRegression()
lm.fit(X_train, y_train)
```

```
     LinearRegression()
```

```
y_train_pred = lm.predict(X_train)
y_test_pred = lm.predict(X_test)
```

```
from sklearn.metrics import mean_absolute_error, mean_squared_error
s = mean_squared_error(y_train, y_train_pred)
print('Mean Squared error of training set :%2f'%s)

p = mean_squared_error(y_test, y_test_pred)
print('Mean Squared error of testing set :%2f'%p)
```

```
     Mean Squared error of training set :3.576827
     Mean Squared error of testing set :3.504032
```

```
from sklearn.metrics import r2_score
s = r2_score(y_train, y_train_pred)
print('R2 Score of training set:%.2f'%s)

p = r2_score(y_test, y_test_pred)
print('R2 Score of testing set:%.2f'%p)
```

```
     R2 Score of training set:0.54
     R2 Score of testing set:0.53
```

Ridge

```
from sklearn.linear_model import Ridge
```

```
ridge_mod = Ridge(alpha=0.01, normalize=True)
ridge_mod.fit(X_train, y_train)
ridge_mod.fit(X_test, y_test)
ridge_model_pred = ridge_mod.predict(X_test)
ridge_mod.score(X_train, y_train)
```

```
     /usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_base.py:145: FutureWarning
```

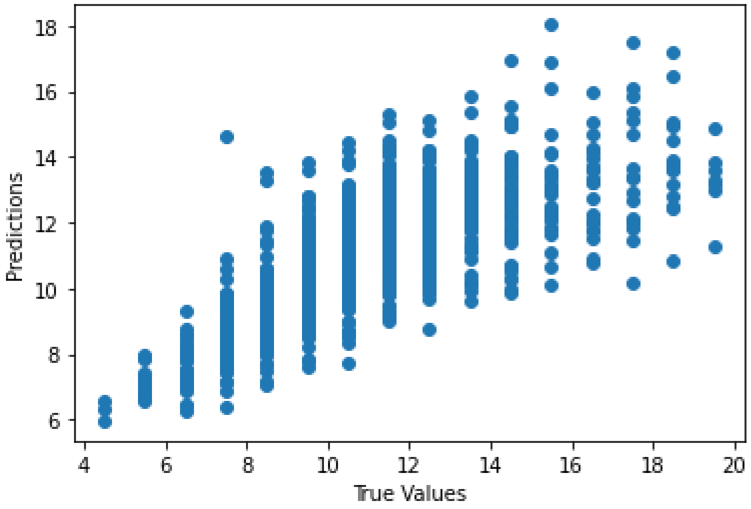
```
If you wish to scale the data, use Pipeline with a StandardScaler in a preprocessing sta

from sklearn.pipeline import make_pipeline

model = make_pipeline(StandardScaler(with_mean=False), Ridge())

If you wish to pass a sample_weight parameter, you need to pass it as a fit parameter tc

kwargs = {s[0] + '__sample_weight': sample_weight for s in model.steps}
model.fit(X, y, **kwargs)

Set parameter alpha to: original_alpha * n_samples.
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_base.py:145: FutureWarning
If you wish to scale the data, use Pipeline with a StandardScaler in a preprocessing sta

from sklearn.pipeline import make_pipeline

model = make_pipeline(StandardScaler(with_mean=False), Ridge())

If you wish to pass a sample_weight parameter, you need to pass it as a fit parameter tc

kwargs = {s[0] + '__sample_weight': sample_weight for s in model.steps}
model.fit(X, y, **kwargs)

Set parameter alpha to: original_alpha * n_samples.
  FutureWarning,
0.527965287146412
```

```
ridge_mod.score(X_test, y_test)
```

```
0.5274395494435754
```

```
plt.scatter(y_test, ridge_model_pred)
plt.xlabel('True Values')
plt.ylabel('Predictions')
```

```
Text(0, 0.5, 'Predictions')
```



Support Vector Regression

```
from sklearn.svm import SVR
```

```
svr = SVR(kernel = 'linear')
svr.fit(X_train, y_train)
svr.fit(X_test, y_test)
```

```
SVR(kernel='linear')
```

```
y_train_pred = svr.predict(X_train)
y_test_pred = svr.predict(X_test)

svr.score(X_train, y_train)
```

```
0.4500466118036842
```

```
svr.score(X_test, y_test)
```

```
0.4412774013022851
```

Random Forest Regression

```
from sklearn.ensemble import RandomForestRegressor
```

```
regr = RandomForestRegressor(max_depth=2, random_state=0,
                              n_estimators=100)
```

```
regr.fit(X_train, y_train)
regr.fit(X_test, y_test)
```

```
RandomForestRegressor(max_depth=2, random_state=0)
```

```
y_train_pred = regr.predict(X_train)
y_test_pred = regr.predict(X_test)

regr.score(X_train, y_train)
```

```
0.41518258557017296
```

```
regr.score(X_test, y_test)
```

```
0.4526102129905978
```

Gradient Boosting Regressor

```
from sklearn.ensemble import GradientBoostingRegressor
```

```
gbr = GradientBoostingRegressor()
gbr.fit(X_train, y_train)
gbr.fit(X_test, y_test)
```

```
     GradientBoostingRegressor()
```

```
y_train_pred = regr.predict(X_train)
y_test_pred = regr.predict(X_test)
```

```
regr.score(X_train, y_train)
```

```
     0.41518258557017296
```

KNeighbors Regressor

```
from sklearn.neighbors import KNeighborsRegressor
```

```
knn = KNeighborsRegressor(n_neighbors =4 )
knn.fit(X_train, y_train)
knn.fit(X_test, y_test)
```

```
     KNeighborsRegressor(n_neighbors=4)
```

```
y_train_pred = knn.predict(X_train)
y_test_pred = knn.predict(X_test)


knn.score(X_train, y_train)
```

```
     0.467886775273044
```

```
knn.score(X_test, y_test)
```

```
     0.6892132267547362
```

✓ 0s completed at 9:50 PM ● ✕