In [19]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import KFold
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor
from sklearn.neural_network import MLPRegressor
```

In [20]:
```
df = pd.read_csv('vgsales.csv')
# Rank - Ranking of overall sales
# Name - The games name
# Platform - Platform of the games release (i.e. PC, PS4, etc.)
# Year - Year of the game's release
# Genre - Genre of the game
# Publisher - Publisher of the game
# NA_Sales - Sales in North America (in millions)
# EU_Sales - Sales in Europe (in millions)
# JP_Sales - Sales in Japan (in millions)
# Other_Sales - Sales in the rest of the world (in millions)
# Global_Sales - Total worldwide sales.
```

In [21]:
```
df_copy = df.copy()
df_copy = df_copy.drop(['Global_Sales', 'Rank'], 1)
df_copy = df_copy.dropna()
df_copy.head(10)
```

Out[21]:

	Name	Platform	Year	Genre	Publisher	NA_Sales	EU_Sales	JP_Sales	Other_Sales
0	Wii Sports	Wii	2006.0	Sports	Nintendo	41.49	29.02	3.77	8.46
1	Super Mario Bros.	NES	1985.0	Platform	Nintendo	29.08	3.58	6.81	0.77
2	Mario Kart Wii	Wii	2008.0	Racing	Nintendo	15.85	12.88	3.79	3.31
3	Wii Sports Resort	Wii	2009.0	Sports	Nintendo	15.75	11.01	3.28	2.96
4	Pokemon Red/Pokemon Blue	GB	1996.0	Role-Playing	Nintendo	11.27	8.89	10.22	1.00
5	Tetris	GB	1989.0	Puzzle	Nintendo	23.20	2.26	4.22	0.58
6	New Super Mario Bros.	DS	2006.0	Platform	Nintendo	11.38	9.23	6.50	2.90
7	Wii Play	Wii	2006.0	Misc	Nintendo	14.03	9.20	2.93	2.85
8	New Super Mario Bros. Wii	Wii	2009.0	Platform	Nintendo	14.59	7.06	4.70	2.26
9	Duck Hunt	NES	1984.0	Shooter	Nintendo	26.93	0.63	0.28	0.47

DATA PREPROCESSING

'Platforms' Column

In [22]:
```
# Creates Nintendo Column
nintendo_mapping = {'DS': 1.0, 'Wii': 1.0, 'GBA': 1.0,
                    'GC': 1.0, '3DS': 1.0, 'N64': 1.0,
                    'SNES': 1.0, 'WiiU': 1.0, 'NES': 1.0,
                    'GB': 1.0}
df_copy['Nintendo'] = df_copy['Platform'].map(nintendo_mapping).fillna(0.0)

# Creates PlayStation Column
playstation_mapping = {'PS2': 1.0, 'PS3': 1.0, 'PSP': 1.0,
                       'PS': 1.0, 'PSV': 1.0, 'PS4': 1.0}
df_copy['PlayStation'] = df_copy['Platform'].map(playstation_mapping).fillna(0.0)

# Creates XBox Column
xbox_mapping = {'X360': 1.0, 'XB': 1.0, 'XOne': 1.0}
df_copy['XBox'] = df_copy['Platform'].map(xbox_mapping).fillna(0.0)

# Creates PC
pc_mapping = {'PC': 1.0}
df_copy['XBox'] = df_copy['Platform'].map(pc_mapping).fillna(0.0)

# Creates SEGA Column
sega_mapping = {'SAT': 1.0, 'GEN': 1.0, 'SCD': 1.0,
                'GG': 1.0, 'DC': 1.0}
df_copy['SEGA'] = df_copy['Platform'].map(sega_mapping).fillna(0.0)

# Creates Other Platforms Column
other_mapping = {'2600': 1.0, 'NG': 1.0,
                 'WS': 1.0, '3DO': 1.0, 'TG16': 1.0,
                 'PCFX': 1.0}
df_copy['Other_Plats'] = df_copy['Platform'].map(other_mapping).fillna(0.0)

df_copy = df_copy.drop(['Platform'], 1)
```

'Publisher' Column

In [23]:
```
df_copy = df_copy.drop(['Publisher'], 1)
```

'Genre' Column

In [24]:
```
df_copy['Action'] = df_copy['Genre'].map({'Action': 1.0}).fillna(0.0)
df_copy['Sports'] = df_copy['Genre'].map({'Sports': 1.0}).fillna(0.0)
df_copy['Misc'] = df_copy['Genre'].map({'Misc': 1.0}).fillna(0.0)
df_copy['Role-Playing'] = df_copy['Genre'].map({'Role-Playing': 1.0}).fillna(0.0)
df_copy['Shooter'] = df_copy['Genre'].map({'Shooter': 1.0}).fillna(0.0)
df_copy['Adventure'] = df_copy['Genre'].map({'Adventure': 1.0}).fillna(0.0)
df_copy['Racing'] = df_copy['Genre'].map({'Racing': 1.0}).fillna(0.0)
df_copy['Platform'] = df_copy['Genre'].map({'Platform': 1.0}).fillna(0.0)
df_copy['Simulation'] = df_copy['Genre'].map({'Simulation': 1.0}).fillna(0.0)
df_copy['Fighting'] = df_copy['Genre'].map({'Fighting': 1.0}).fillna(0.0)
df_copy['Strategy'] = df_copy['Genre'].map({'Strategy': 1.0}).fillna(0.0)
df_copy['Puzzle'] = df_copy['Genre'].map({'Puzzle': 1.0}).fillna(0.0)

df_copy = df_copy.drop(['Genre'], 1)
```

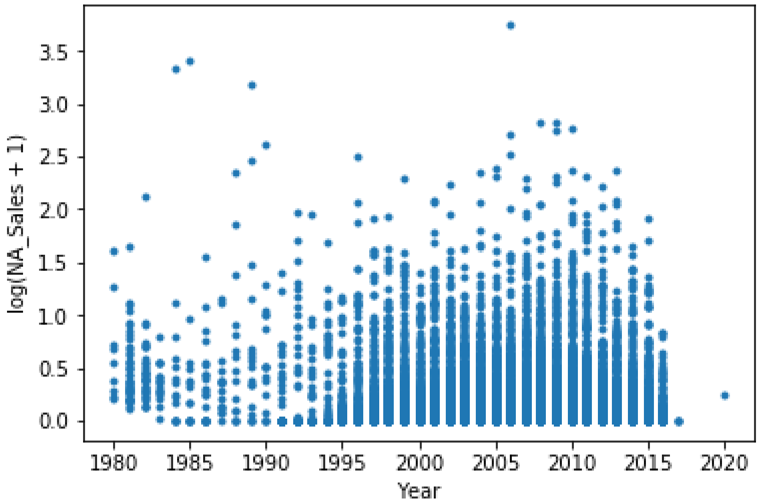
'Name' Column

In [25]:
```
df_copy = df_copy.drop('Name', axis=1)
```

'Year' Column

In [26]:
```
plt.plot(df_copy['Year'], np.log(1+df_copy['NA_Sales']), '.')
plt.xlabel('Year')
plt.ylabel('log(NA_Sales + 1)')
```

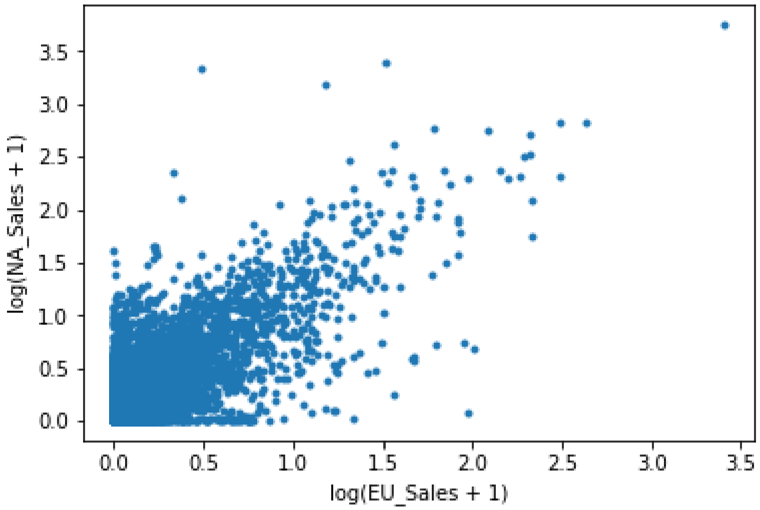
Out[26]:
```
Text(0,0.5,'log(NA_Sales + 1)')
```



'EU_Sales' Column

In [27]:
```
plt.plot(np.log(1+df_copy['EU_Sales']), np.log(1+df_copy['NA_Sales']), '.')
plt.xlabel('log(EU_Sales + 1)')
plt.ylabel('log(NA_Sales + 1)')
```

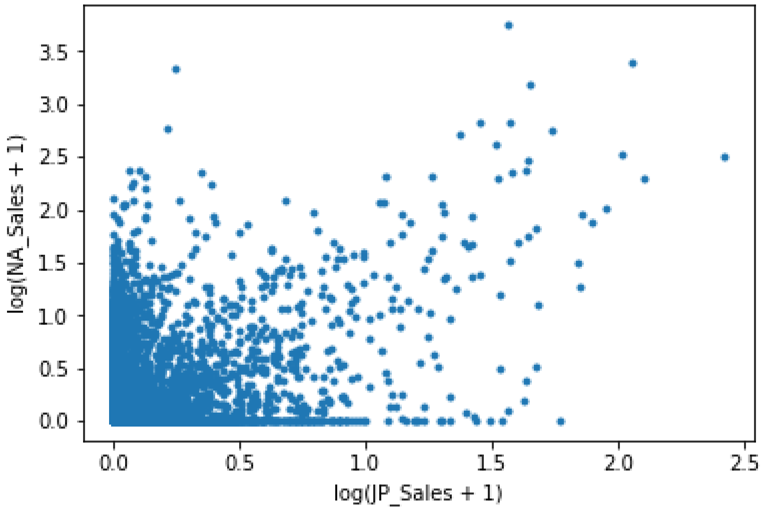
Out[27]:
```
Text(0,0.5,'log(NA_Sales + 1)')
```



'JP_Sales' Column

In [28]:
```
plt.plot(np.log(1+df_copy['JP_Sales']), np.log(1+df_copy['NA_Sales']), '.')
plt.xlabel('log(JP_Sales + 1)')
plt.ylabel('log(NA_Sales + 1)')
```

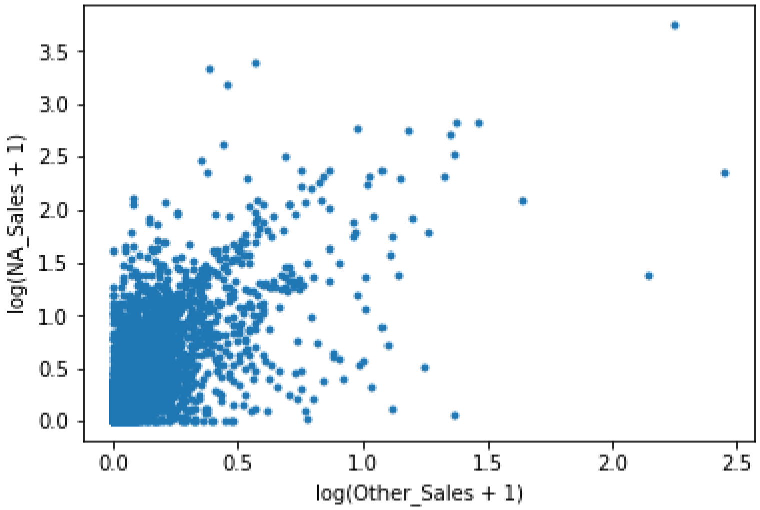
Out[28]:
```
Text(0,0.5,'log(NA_Sales + 1)')
```



'Other_Sales' Column

In [29]:
```
plt.plot(np.log(1+df_copy['Other_Sales']), np.log(1+df_copy['NA_Sales']), '.')
plt.xlabel('log(Other_Sales + 1)')
plt.ylabel('log(NA_Sales + 1)')
```

Out[29]:
```
Text(0,0.5,'log(NA_Sales + 1)')
```



ANALYSIS

Multiple Linear Regression

In [30]:
```
df_copy.head()
```

Out[30]:

	Year	NA_Sales	EU_Sales	JP_Sales	Other_Sales	Nintendo	PlayStation	XBox	SEGA	Other_Plats	...	Misc	Role-Playing	Shooter
0	2006.0	41.49	29.02	3.77	8.46	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0
1	1985.0	29.08	3.58	6.81	0.77	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0
2	2008.0	15.85	12.88	3.79	3.31	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0
3	2009.0	15.75	11.01	3.28	2.96	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0
4	1996.0	11.27	8.89	10.22	1.00	1.0	0.0	0.0	0.0	0.0	...	0.0	1.0	0.0

5 rows × 22 columns

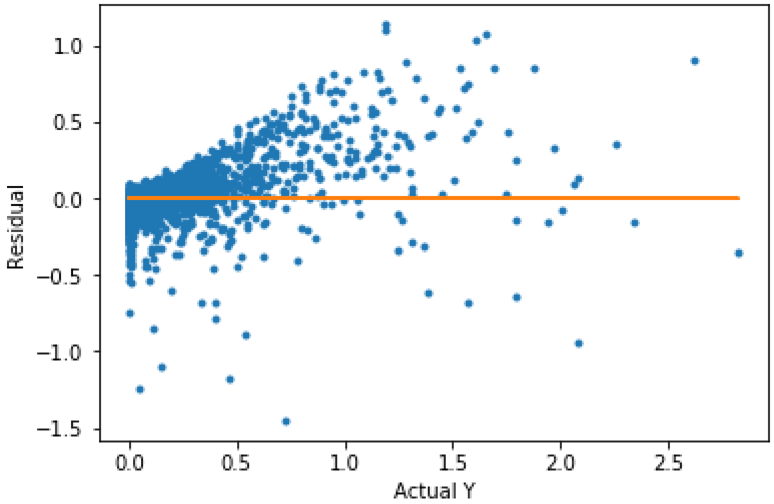
In [31]:
```
# Make dataset with transformed variables
mlr_dataset = df_copy.copy()
mlr_dataset['NA_Sales'] = np.log(1 + mlr_dataset['NA_Sales'])
mlr_dataset['EU_Sales'] = np.log(1 + mlr_dataset['EU_Sales'])
mlr_dataset['JP_Sales'] = np.log(1 + mlr_dataset['JP_Sales'])
mlr_dataset['Other_Sales'] = np.log(1 + mlr_dataset['Other_Sales'])


# Create design matrix and response variables
y = mlr_dataset['NA_Sales']
X = mlr_dataset.drop('NA_Sales', 1)
columns = list(X.columns.values)
columns.insert(0,'Intercept')
X_design = sm.add_constant(X)
X_design = np.asarray(X_design)
y = np.asarray(y)

# split data into training and testing data
X_train, X_test, y_train, y_test = train_test_split(X_design, y, test_size=0.20)

# Create Multiple Linear Regression model and fit to training data
LinRegr_Regressor = sm.OLS(y_train, X_train)
result = LinRegr_Regressor.fit()

# Make predictions on the testing data and calcuate MSE
y_pred = result.predict(X_test)
MSE = mean_squared_error(y_test, y_pred)

print("The test MSE is determined to be " + repr(MSE))

residuals = y_test - y_pred
plt.plot(y_test, residuals, '.')
plt.plot(y_test, 0*y_test)
plt.xlabel('Actual Y')
plt.ylabel('Residual')

# Create table of model prediction results
data = {'Attributes': columns,
        'Coefficient Beta_i': result.params,
        't-Values': result.tvalues,
        'p-Values': result.pvalues
       }
```

```
ModelResults_df = pd.DataFrame(data)
ModelResults_df.round(4) # Round values in table to 4-decimal places
```

```
The test MSE is determined to be 0.026776624314745485
```

Out[31]:

	Attributes	Coefficient Beta_i	t-Values	p-Values
0	Intercept	12.0585	22.0936	0.0000
1	Year	-0.0064	-21.8812	0.0000
2	EU_Sales	0.6511	52.7401	0.0000
3	JP_Sales	0.0480	4.6260	0.0000
4	Other_Sales	0.9046	33.2441	0.0000
5	Nintendo	-0.0403	-8.4446	0.0000
6	PlayStation	-0.1056	-22.7656	0.0000
7	XBox	-0.1440	-19.5516	0.0000
8	SEGA	-0.1656	-13.1917	0.0000
9	Other_Plats	0.0498	2.8177	0.0048
10	Action	1.0106	21.9321	0.0000
11	Sports	1.0109	22.2683	0.0000
12	Misc	0.9985	21.7082	0.0000
13	Role-Playing	1.0016	21.6486	0.0000
14	Shooter	1.0274	22.5460	0.0000
15	Adventure	0.9844	21.2152	0.0000
16	Racing	0.9846	21.7408	0.0000
17	Platform	1.0411	22.9587	0.0000
18	Simulation	1.0080	21.9590	0.0000
19	Fighting	1.0244	22.4247	0.0000
20	Strategy	0.9791	21.3867	0.0000
21	Puzzle	0.9881	21.4809	0.0000


1.0
0.5
0.0
−0.5
−1.0
−1.5
Residual
0.0
0.5
1.0
1.5
2.0
2.5
Actual Y

In [32]:
```
# Create design matrix and response variables
y = np.asarray(mlr_dataset['NA_Sales'])
X = np.asarray(mlr_dataset.drop('NA_Sales', 1))

# Perform cross-validation to determine model performance
K = 10
kfold = KFold(n_splits=K, shuffle=True)

sum_test_MSE = 0
for train_index, test_index in kfold.split(X):
    X_train_curr, X_test_curr = X[train_index], X[test_index]
    y_train_curr, y_test_curr = y[train_index], y[test_index]

    LinRegr_Regressor = LinearRegression()
    LinRegr_Regressor.fit(X_train_curr, y_train_curr)

    y_pred = LinRegr_Regressor.predict(X_test_curr)
    MSE_error = mean_squared_error(y_test_curr, y_pred)

    sum_test_MSE = sum_test_MSE + MSE_error

cv_MSE = sum_test_MSE/K

print("The cross-validated test MSE is determined to be " + repr(cv_MSE))
```

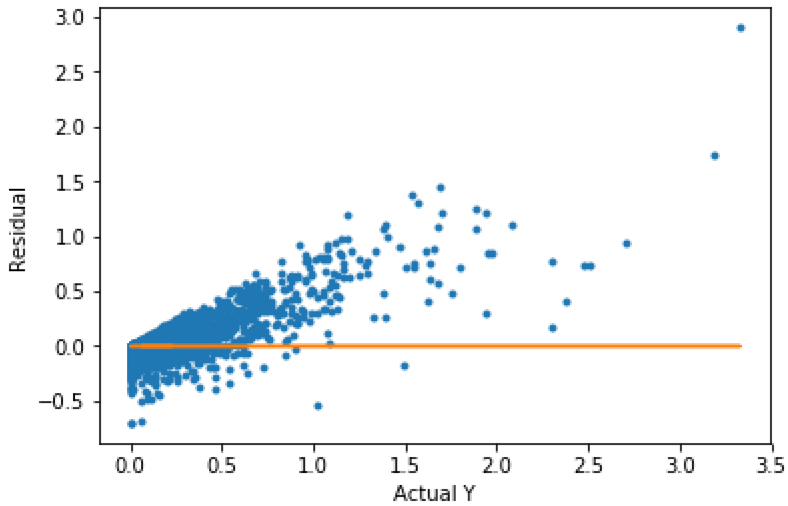
```
The cross-validated test MSE is determined to be 0.027203615637890026
```

K-Nearest Neighbors

In [33]:
```
knn_dataset = df_copy.copy()

# Make dataset with transformed variables
knn_dataset['NA_Sales'] = np.log(1 + knn_dataset['NA_Sales'])
knn_dataset['EU_Sales'] = np.log(1 + knn_dataset['EU_Sales'])
knn_dataset['JP_Sales'] = np.log(1 + knn_dataset['JP_Sales'])
knn_dataset['Other_Sales'] = np.log(1 + knn_dataset['Other_Sales'])


# Create design matrix and response variables
y = np.asarray(knn_dataset['NA_Sales'])
X = np.asarray(knn_dataset.drop('NA_Sales', 1))

# split data into training and testing data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)

# Create KNN model and fit to training data
KNN_Regressor = KNeighborsRegressor(n_neighbors=5)
KNN_Regressor.fit(X_train, y_train)

# Make predictions on the testing data
y_pred = KNN_Regressor.predict(X_test)
MSE_error = mean_squared_error(y_test, y_pred)

print("The test MSE is determined to be " + repr(MSE))


residuals = y_test - y_pred
plt.plot(y_test, residuals, '.')
plt.plot(y_test, 0*y_test)
plt.xlabel('Actual Y')
plt.ylabel('Residual')
```

```
The test MSE is determined to be 0.026776624314745485
```

Out[33]:
```
Text(0,0.5,'Residual')
```

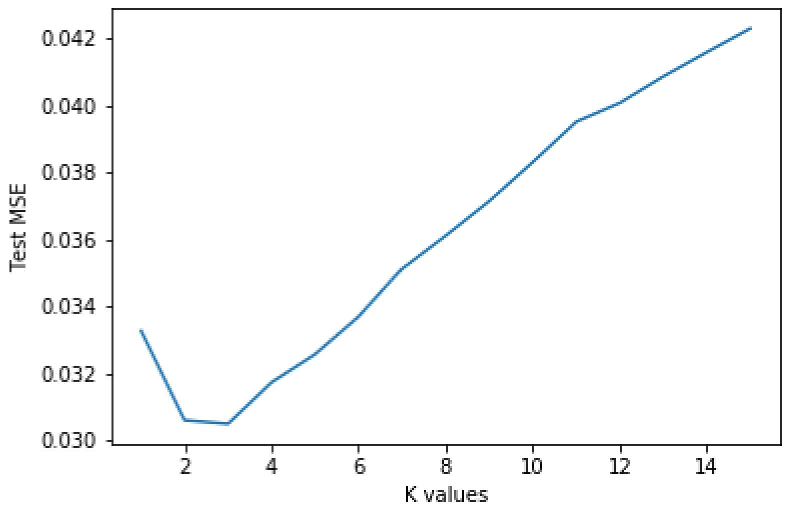


In [38]:
```
knn_dataset = df_copy.copy()

# Make dataset with transformed variables
knn_dataset['NA_Sales'] = np.log(1 + knn_dataset['NA_Sales'])
knn_dataset['EU_Sales'] = np.log(1 + knn_dataset['EU_Sales'])
knn_dataset['JP_Sales'] = np.log(1 + knn_dataset['JP_Sales'])
knn_dataset['Other_Sales'] = np.log(1 + knn_dataset['Other_Sales'])


# Create design matrix and response variables
y = np.asarray(knn_dataset['NA_Sales'])
X = np.asarray(knn_dataset.drop('NA_Sales', 1))

best_score = np.inf
best_k = None
poss_k_list = np.linspace(1,15,15).astype(int)
score_list = []
for k_value in (poss_k_list):

    # Perform cross-validation to determine model performance
    kfold = KFold(n_splits=10, shuffle=True)

    sum_test_MSE = 0
    for train_index, test_index in kfold.split(X):
        X_train_curr, X_test_curr = X[train_index], X[test_index]
        y_train_curr, y_test_curr = y[train_index], y[test_index]

        KNN_Regressor = KNeighborsRegressor(n_neighbors=k_value)
        KNN_Regressor.fit(X_train_curr, y_train_curr)

        y_pred = KNN_Regressor.predict(X_test_curr)
        MSE_error = mean_squared_error(y_test_curr, y_pred)

        sum_test_MSE = sum_test_MSE + MSE_error

    cv_MSE = sum_test_MSE/10
    score_list.append(cv_MSE)

    if cv_MSE < best_score:
        best_score = cv_MSE
        best_k = k_value
```

```
print("The best k-value is determined to be " + repr(best_k))
print("The corresponding cross-validated test MSE is determined to be " + repr(best_score))

plt.plot(poss_k_list, score_list)
plt.xlabel('K values')
plt.ylabel('Test MSE')
```

```
The best k-value is determined to be 3
The corresponding cross-validated test MSE is determined to be 0.030490218329300385
```

Out[38]: Text(0,0.5,'Test MSE')

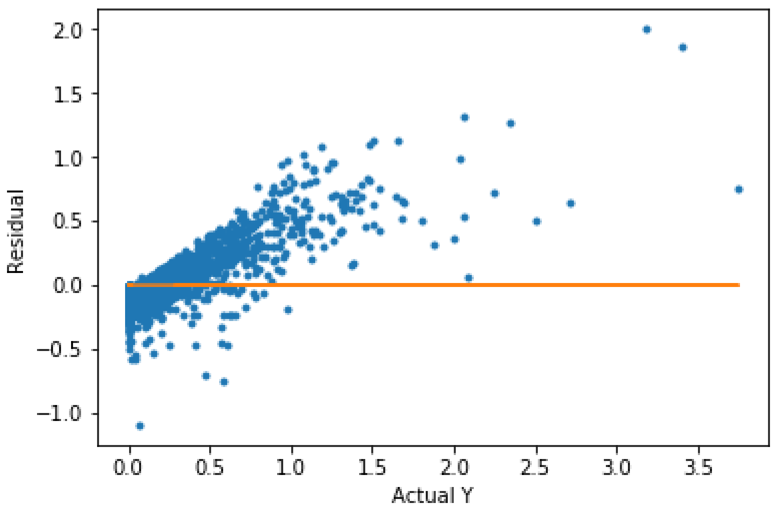


Artificial Neural Network

In [35]:
```
ann_dataset = df_copy.copy()

# Make dataset with transformed variables
ann_dataset['NA_Sales'] = np.log(1 + ann_dataset['NA_Sales'])
ann_dataset['EU_Sales'] = np.log(1 + ann_dataset['EU_Sales'])
ann_dataset['JP_Sales'] = np.log(1 + ann_dataset['JP_Sales'])
ann_dataset['Other_Sales'] = np.log(1 + ann_dataset['Other_Sales'])

# Create design matrix and response variables
y = np.asarray(ann_dataset['NA_Sales'])
X = np.asarray(ann_dataset.drop('NA_Sales', 1))


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)


# creates a KNN Classifier object, which is used to create a classification model of the training set
MLPerceptron_Model = MLPRegressor(hidden_layer_sizes=(21, 21, 21), max_iter=1000)
MLPerceptron_Model.fit(X_train, y_train)

# Make predictions using the Artifical Neual Network classification model
y_pred = MLPerceptron_Model.predict(X_test)
MSE_error = mean_squared_error(y_test, y_pred)

print("The test MSE is determined to be " + repr(MSE_error))

residuals = y_test - y_pred
plt.plot(y_test, residuals, '.')
plt.plot(y_test, 0*y_test)
plt.xlabel('Actual Y')
plt.ylabel('Residual')
```

```
The test MSE is determined to be 0.03632611938602802
```

Out[35]:
```
Text(0,0.5,'Residual')
```

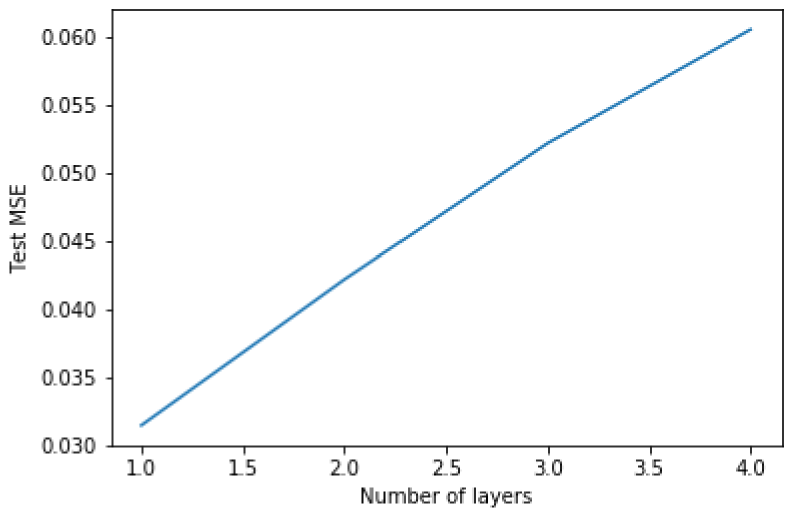


In [37]:
```
ann_dataset = df_copy.copy()

# Make dataset with transformed variables
ann_dataset['NA_Sales'] = np.log(1 + ann_dataset['NA_Sales'])
ann_dataset['EU_Sales'] = np.log(1 + ann_dataset['EU_Sales'])
ann_dataset['JP_Sales'] = np.log(1 + ann_dataset['JP_Sales'])
ann_dataset['Other_Sales'] = np.log(1 + ann_dataset['Other_Sales'])


# Create design matrix and response variables
y = np.asarray(ann_dataset['NA_Sales'])
X = np.asarray(ann_dataset.drop('NA_Sales', 1))


best_tuple = None
best_score = np.inf
poss_layers_list = [(21), (21, 21), (21, 21, 21), (21, 21, 21, 21)]
score_list = []
for curr_layers in (poss_layers_list):

    # Perform cross-validation to determine model performance
    kfold = KFold(n_splits=10, shuffle=True)
    sum_test_MSE = 0
    for train_index, test_index in kfold.split(X):
        X_train_curr, X_test_curr = X[train_index], X[test_index]
        y_train_curr, y_test_curr = y[train_index], y[test_index]

        MLPerceptron_Model = MLPRegressor(hidden_layer_sizes=curr_layers, max_iter=1000)
        MLPerceptron_Model.fit(X_train_curr, y_train_curr)

        # Make predictions using the Artifical Neual Network classification model
        y_pred = MLPerceptron_Model.predict(X_test_curr)
        MSE_error = mean_squared_error(y_test_curr, y_pred)

        sum_test_MSE = sum_test_MSE + MSE_error

    cv_MSE = sum_test_MSE/10
    score_list.append(cv_MSE)

    if cv_MSE < best_score:
        best_score = cv_MSE
        best_tuple = curr_layers
```

```
print("The best layer distribution is determined to be " + repr(best_tuple))
print("The corresponding cross-validated test MSE is determined to be " + repr(best_score))


plt.plot(np.linspace(1,4,4), score_list)
plt.xlabel('Number of layers')
plt.ylabel('Test MSE')
```

```
The best layer distribution is determined to be 21
The corresponding cross-validated test MSE is determined to be 0.031444811839850616
```

Out[37]: `Text(0,0.5,'Test MSE')`



In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: