In [10]:
```
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.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.neighbors import KNeighborsClassifier, KNeighborsRegressor
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import KFold
```

In [11]:
```
df = pd.read_csv('Boston.csv')
df_copy = df.copy()
df_copy.rename(columns={'Unnamed: 0': 'i'}, inplace=True)
df_copy.head()
```

Out[11]:

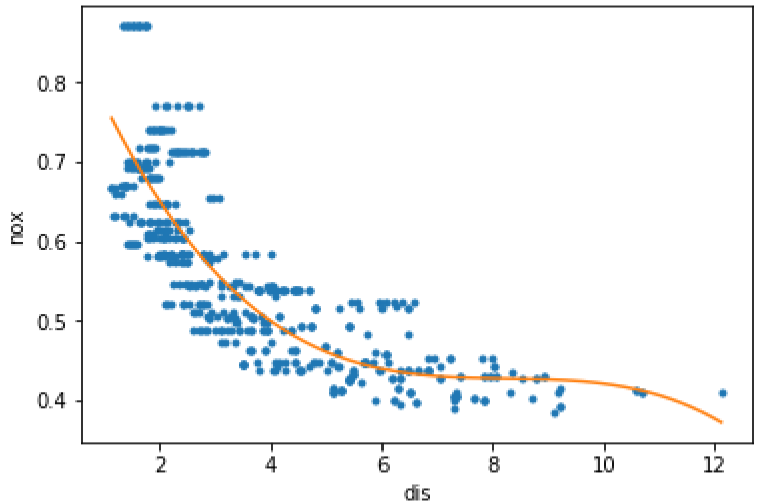
	i	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	black	lstat	medv
0	1	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0
1	2	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6
2	3	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
3	4	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
4	5	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	36.2

Problem a

In [12]:
```
y = df_copy['nox']
X = df_copy['dis']


# Make dataset of desired predictors
predictors = {
            'dis': X,
            'dis^2': X**2,
            'dis^3': X**3
               }
data = pd.DataFrame(predictors)
data = np.asarray(data)
data = sm.add_constant(data)


# Create the Multiple Linear Regression Model and fit it
MLRmodel = sm.OLS(y, data)
result = MLRmodel.fit()

# Create table of model prediction results
data = {'Coefficient Beta_i': result.params,
        't-Values': result.tvalues,
        'p-Values': result.pvalues
       }
data_analysis = pd.DataFrame(data)
data_analysis.round(4) # Round values in table to 4-decimal places
print(data_analysis)
# Plot the cubic fit
# Create Linear Regression Function
x_values = np.linspace(df_copy['dis'].min(),df_copy['dis'].max(),3000)
function = 0
for i in range(4):
    function = function + result.params[i]*(x_values**(i))

plt.plot(X,y, '.')
plt.plot(x_values, function)
plt.xlabel('dis')
plt.ylabel('nox')
```

	Coefficient Beta_i	t-Values	p-Values
const	0.934128	45.110365	9.853624e-179
x1	-0.182082	-12.388763	6.078843e-31
x2	0.021928	7.476412	3.428917e-13
x3	-0.000885	-5.123959	4.274950e-07

Out[12]: Text(0,0.5,'nox')

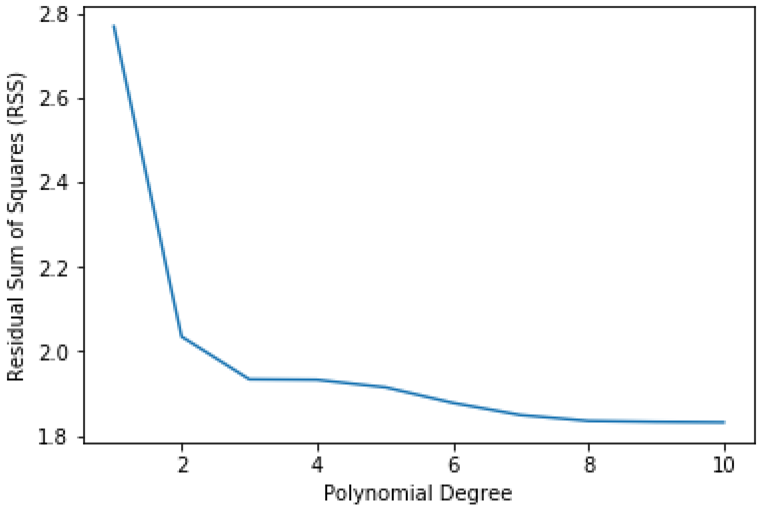


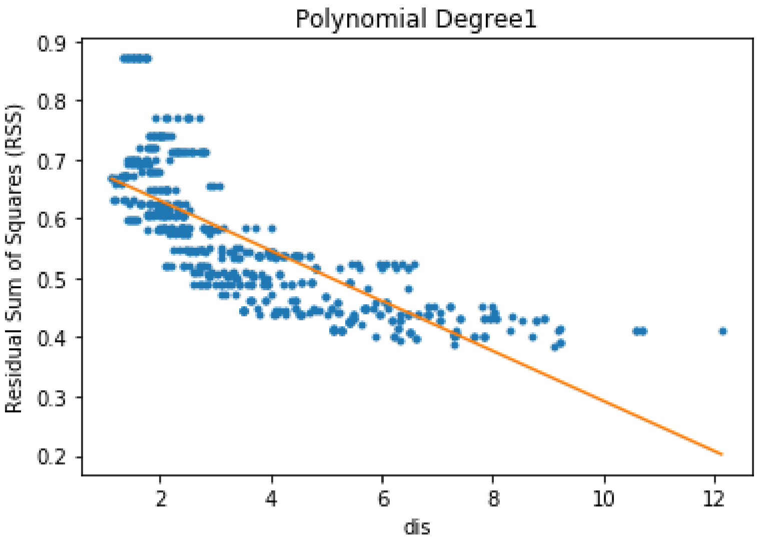
Problem b

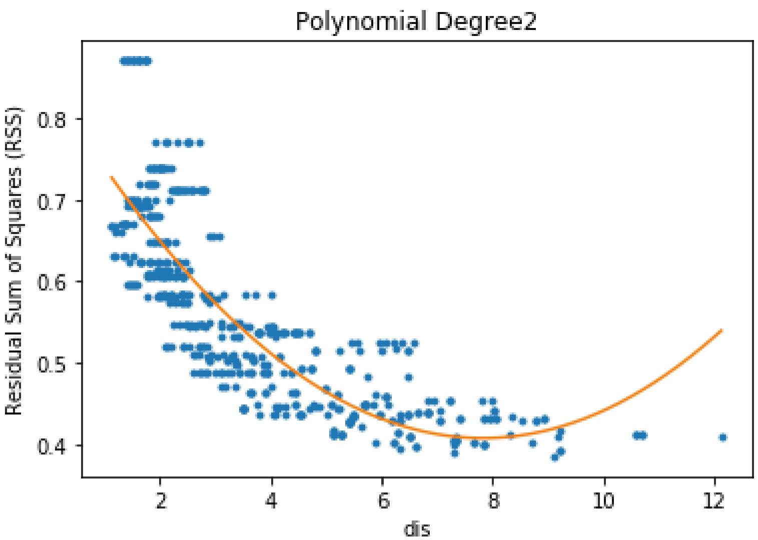
In [13]:
```
# Extract predictor x and response y as arrays
y = np.asarray(df_copy['nox'])
X = np.reshape(np.asarray(df_copy['dis']), (-1,1))


# Generate polynomial features up to degree 10
max_degree = 10
poly = PolynomialFeatures(degree=max_degree, include_bias=False)
x_new = poly.fit_transform(X)


function_list = []
RSS_list = []
poss_degrees = np.linspace(1,max_degree,max_degree).astype(int)
x_values = np.linspace(df_copy['dis'].min(),df_copy['dis'].max(),3000)
for degree in (poss_degrees):
    # Obtain the terms of 1 to current degree
    x_curr = x_new[:, :degree]
    x_curr = sm.add_constant(x_curr)

    # Create the Multiple Linear Regression Model and fit it
    MLRmodel = sm.OLS(y, x_curr)
    result = MLRmodel.fit()

    # make predictions and calculate RSS
    y_pred = result.predict(x_curr)
    RSS = np.sum((y-y_pred)**2)
    RSS_list.append(RSS)

    function = 0

    for i in range(degree+1):
        function = function + result.params[i]*(x_values**(i))

    function_list.append(function)

plt1 = plt.figure(1)
plt.plot(poss_degrees, RSS_list)
plt.xlabel('Polynomial Degree')
plt.ylabel('Residual Sum of Squares (RSS)')

predictors = {'Degree': poss_degrees,
```

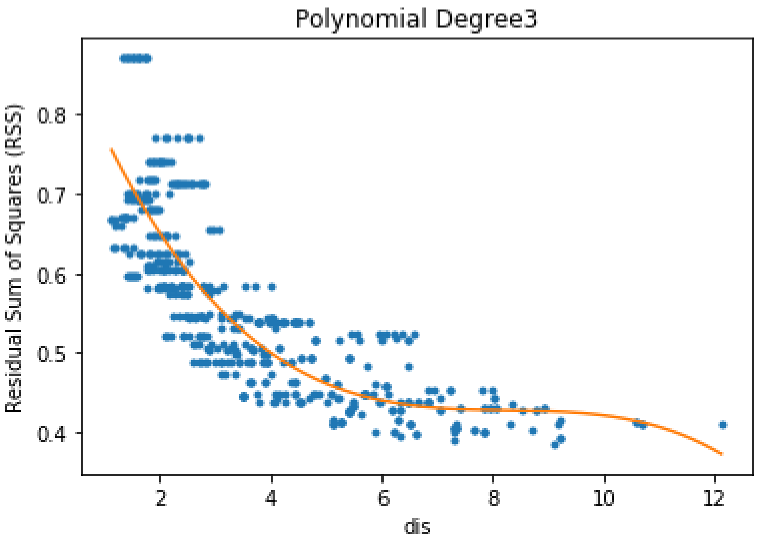
```
                'RSS': RSS_list}
data = pd.DataFrame(predictors)
print(data)

for i in range(10):
    plt2 = plt.figure(i+2)
    plt.plot(X,y,'.')
    plt.plot(x_values, function_list[i])
    plt.title('Polynomial Degree' + repr(i+1))
    plt.xlabel('dis')
    plt.ylabel('Residual Sum of Squares (RSS)')
```

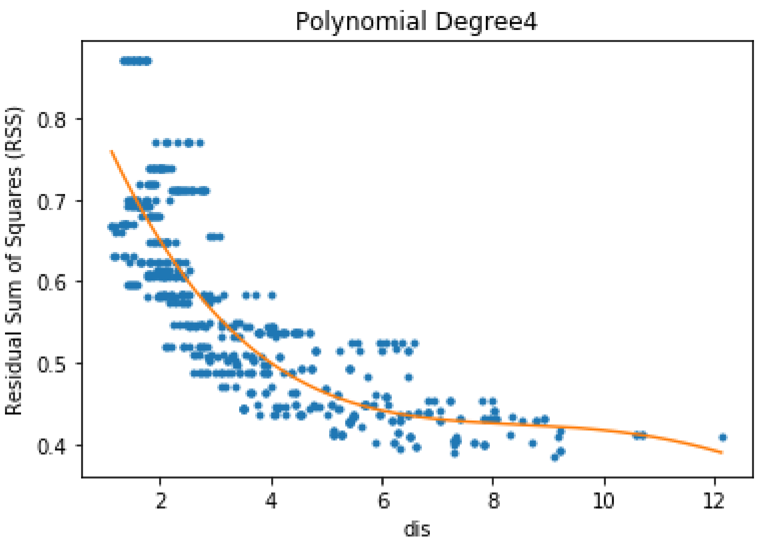
	Degree	RSS
0	1	2.768563
1	2	2.035262
2	3	1.934107
3	4	1.932981
4	5	1.915290
5	6	1.878257
6	7	1.849484
7	8	1.835630
8	9	1.833331
9	10	1.832171

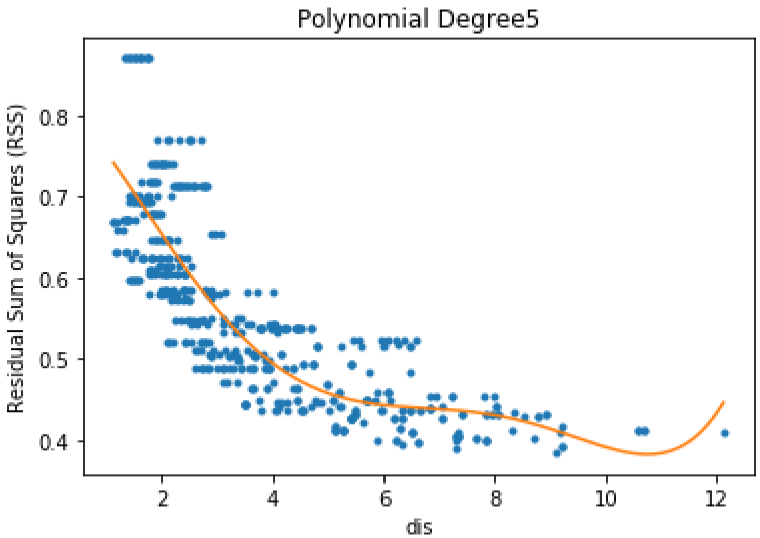


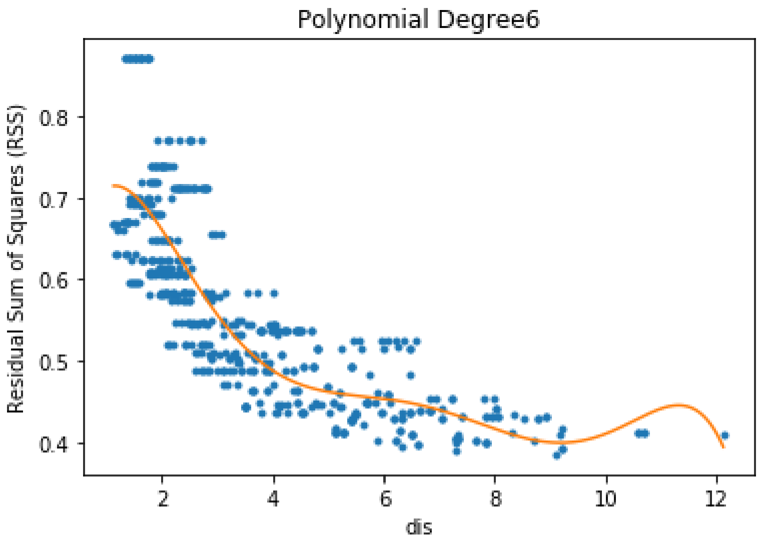

Polynomial Degree1
Residual Sum of Squares (RSS)
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
2
4
6
8
10
12
dis

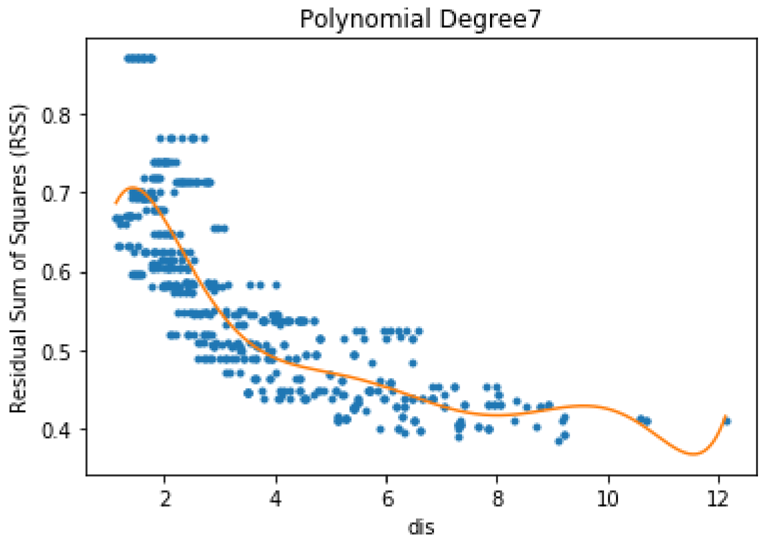

Polynomial Degree2
Residual Sum of Squares (RSS)
0.8
0.7
0.6
0.5
0.4
2
4
6
8
10
12
dis

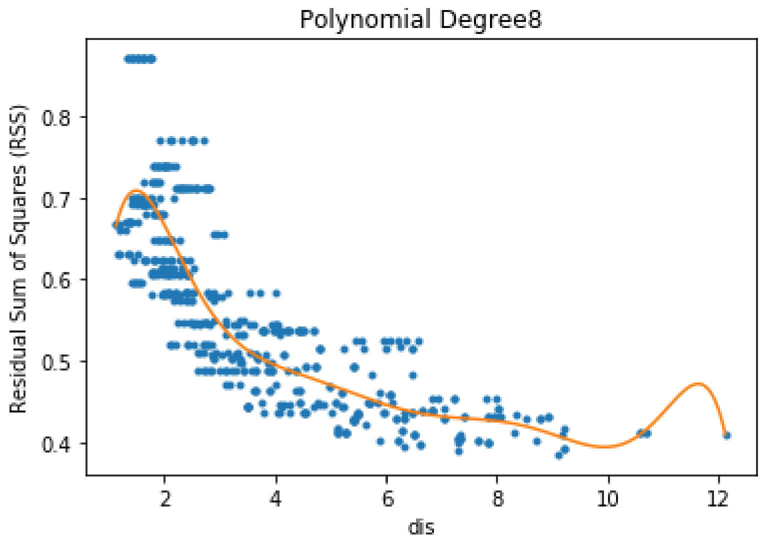

Polynomial Degree3
Residual Sum of Squares (RSS)
0.4
0.5
0.6
0.7
0.8
2
4
6
8
10
12
dis

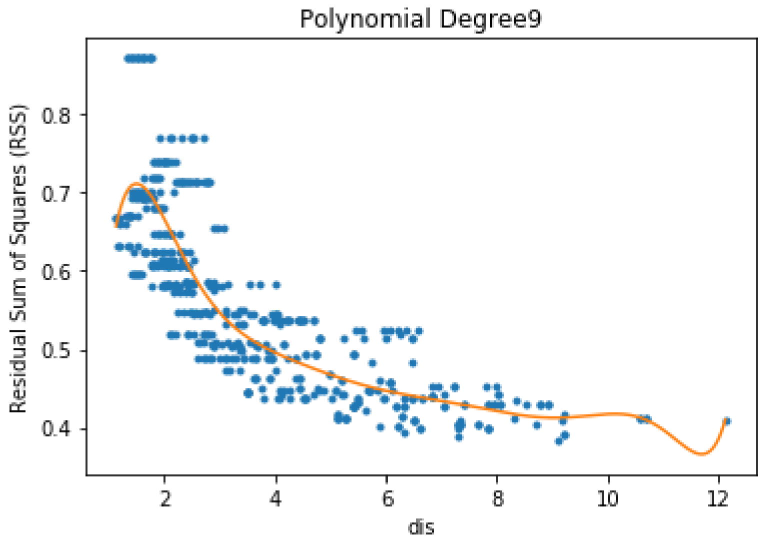

Polynomial Degree4
Residual Sum of Squares (RSS)
0.4
0.5
0.6
0.7
0.8
2
4
6
8
10
12
dis

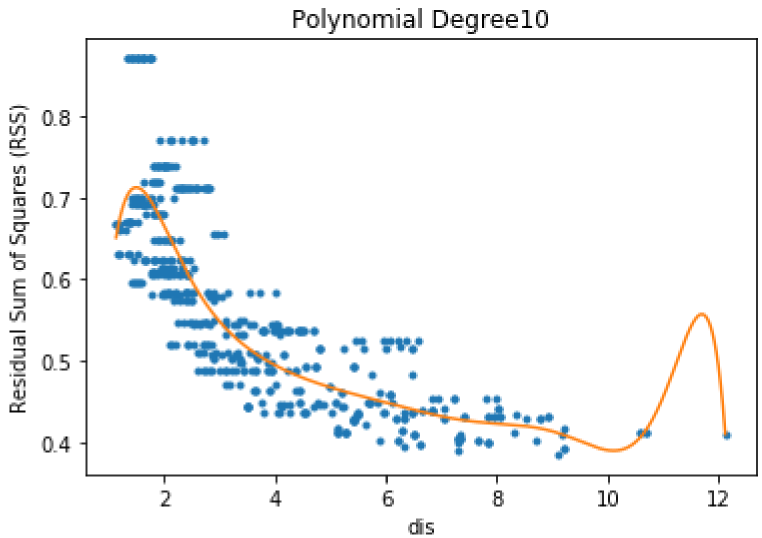

Polynomial Degree5
Residual Sum of Squares (RSS)
0.4
0.5
0.6
0.7
0.8
2
4
6
8
10
12
dis


Polynomial Degree6
Residual Sum of Squares (RSS)
0.4
0.5
0.6
0.7
0.8
2
4
6
8
10
12
dis


Polynomial Degree7
Residual Sum of Squares (RSS)
0.8
0.7
0.6
0.5
0.4
2
4
6
8
10
12
dis


Polynomial Degree8
Residual Sum of Squares (RSS)
0.8
0.7
0.6
0.5
0.4
2
4
6
8
10
12
dis


Polynomial Degree9
Residual Sum of Squares (RSS)
0.8
0.7
0.6
0.5
0.4
2
4
6
8
10
12
dis


Polynomial Degree10
Residual Sum of Squares (RSS)
0.8
0.7
0.6
0.5
0.4
2
4
6
8
10
12
dis

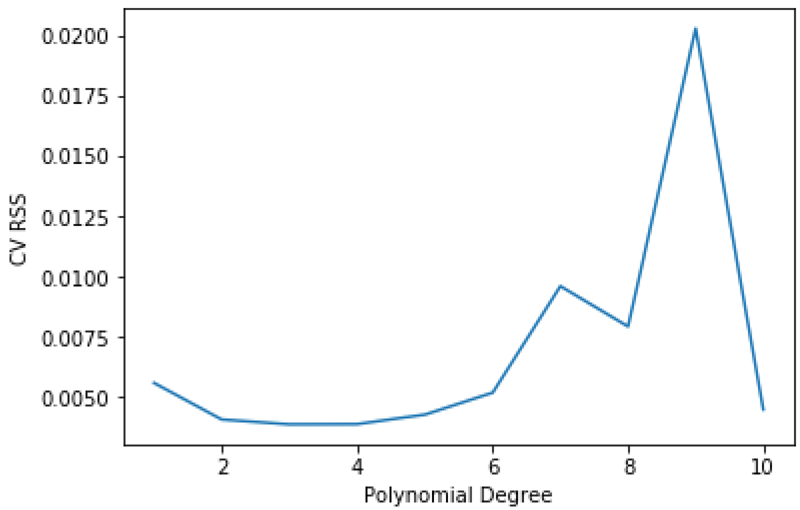
Problem c

In [16]:
```
# Extract predictor x and response y as arrays
y = np.asarray(df_copy['nox'])
x = np.reshape(np.asarray(df_copy['dis']), (-1,1))


# Generate polynomial features up to degree 10
max_degree = 10
poly = PolynomialFeatures(degree=max_degree, include_bias=False)
x_new = poly.fit_transform(x)


K = 10
lowest_error = np.inf
best_degree = None
poss_degrees = np.linspace(1,max_degree,max_degree).astype(int)
cv_error_list = []
for degree in (poss_degrees):
    # Obtain the terms of 1 to current degree
    x_curr = x_new[:, :degree]

    # K-Fold splitter
    kfold = KFold(n_splits=K, shuffle=True)
    sum_test_errors = 0 # Initialize the total test error
    for train_index, test_index in kfold.split(x_curr): # For each group
        # Obtain training and testing data
        X_train, X_test = x_curr[train_index], x_curr[test_index]
        y_train, y_test = y[train_index], y[test_index]

        # Create Linear Regression model and fit to the training data
        LinRegr_classifier = LinearRegression()
        LinRegr_classifier.fit(X_train, y_train)

        # Make predictions and calculate Residual Square Error
        y_pred = LinRegr_classifier.predict(X_test)
        test_error = np.sum((y_test - y_pred)**2)/len(y_test)

        sum_test_errors = sum_test_errors + test_error

    # cross-validated test error for polynomial model of degree "degree"
    current_test_error = sum_test_errors/K
    cv_error_list.append(current_test_error)
```

```
    if current_test_error < lowest_error:
        lowest_error = current_test_error
        best_degree = degree

print("The best degree for polynomial model is " + repr(best_degree))
plt.plot(poss_degrees, cv_error_list)
plt.xlabel('Polynomial Degree')
plt.ylabel('CV RSS')
```

```
The best degree for polynomial model is 3
```

Out[16]: `Text(0,0.5,'CV RSS')`

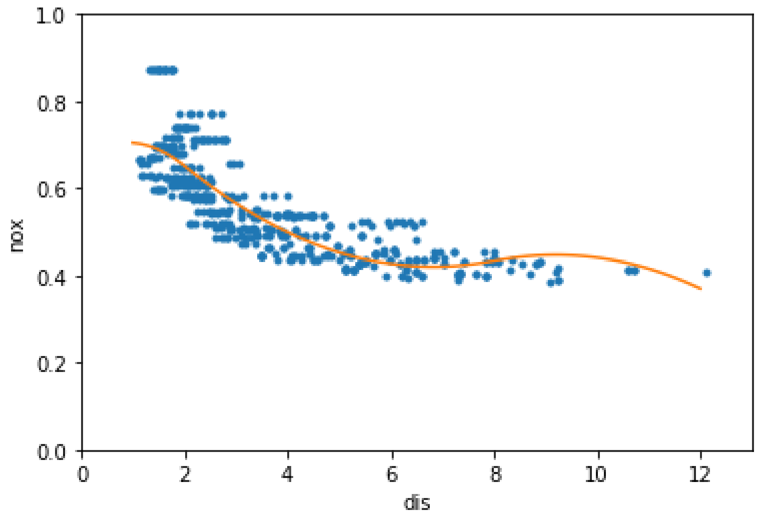


Problem d

In [17]:
```
from patsy import dmatrix
from sklearn.metrics import mean_squared_error

# Dividing data into train and validation datasets
X = df_copy['dis']
y = df_copy['nox']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)


# degrees of freedom = degree + knots

# ---------------- Generate Cubic Spline with 3 knots ------------------
# Generating cubic spline with 3 knots
knots1 = "(2,8)"
degree1 = "2"
transformed_x_train = dmatrix("bs(train, knots=" + knots1 + ", degree=" + degree1 + ", include_intercept=Fals
e)",
                        {"train": X_train},
                        return_type='dataframe')
# Fitting Generalised linear model on transformed dataset
spline = sm.GLM(y_train, transformed_x_train).fit()


print(spline.summary())


# --------------- Make Predictions with Splines -------------------------
# Predictions on both splines
transformed_x_test = dmatrix("bs(valid, knots=" + knots1 +  ", degree=" + degree1 + ", include_intercept=Fals
e)",
                             {"valid": X_test},
                             return_type='dataframe')
# transformed_x_test
y_pred1 = spline.predict(transformed_x_test)


# Calculating MSE values
mse1 = mean_squared_error(y_test, y_pred1)
print("The MSE was determined to be " + repr(mse1))
```

```
# --------------- PLotting the Splines ------------------

# We will plot the graph for 70 observations only
x_values = np.linspace(1, 12,70)

# Make predictions on x-values using fitted splines
transformed_x_values = dmatrix("bs(xp, knots=" + knots1 + ", degree=" + degree1 + ", include_intercept=Fals
e)",
                             {"xp": x_values},
                             return_type='dataframe')
y_pred1 = spline.predict(transformed_x_values)


# Plot the splines and error bands
plt.plot(X, y, '.')
plt.plot(x_values, y_pred1)
plt.xlim(0,13)
plt.ylim(0,1)
plt.xlabel('dis')
plt.ylabel('nox')
plt.show()
```

```
                 Generalized Linear Model Regression Results
==============================================================================
Dep. Variable:                    nox   No. Observations:                  354
Model:                            GLM   Df Residuals:                      349
Model Family:                Gaussian   Df Model:                            4
Link Function:               identity   Scale:                       0.0038580
Method:                          IRLS   Log-Likelihood:                 483.91
Date:                Sun, 05 Dec 2021   Deviance:                       1.3464
Time:                        22:30:03   Pearson chi2:                     1.35
No. Iterations:                     3   Covariance Type:             nonrobust
==============================================================================================================
====================
                                                                   coef    std err          z      P>|z|
[0.025      0.975]
--------------------------------------------------------------------------------------------------------------
--------------------
Intercept                                                        0.7043      0.022     32.528      0.000
0.662       0.747
bs(train, knots=(2, 8), degree=2, include_intercept=False)[0]   -0.0047      0.025     -0.185      0.853
-0.054       0.045
bs(train, knots=(2, 8), degree=2, include_intercept=False)[1]   -0.3411      0.023    -14.643      0.000
-0.387      -0.295
bs(train, knots=(2, 8), degree=2, include_intercept=False)[2]   -0.2241      0.033     -6.892      0.000
-0.288      -0.160
bs(train, knots=(2, 8), degree=2, include_intercept=False)[3]   -0.3340      0.055     -6.035      0.000
-0.443      -0.226
==============================================================================================================
====================
The MSE was determined to be 0.004040387937487178
```

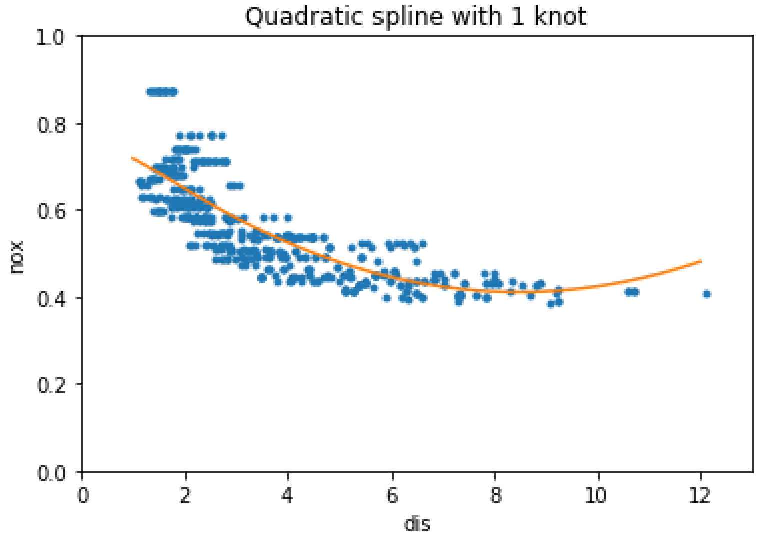


Problem e

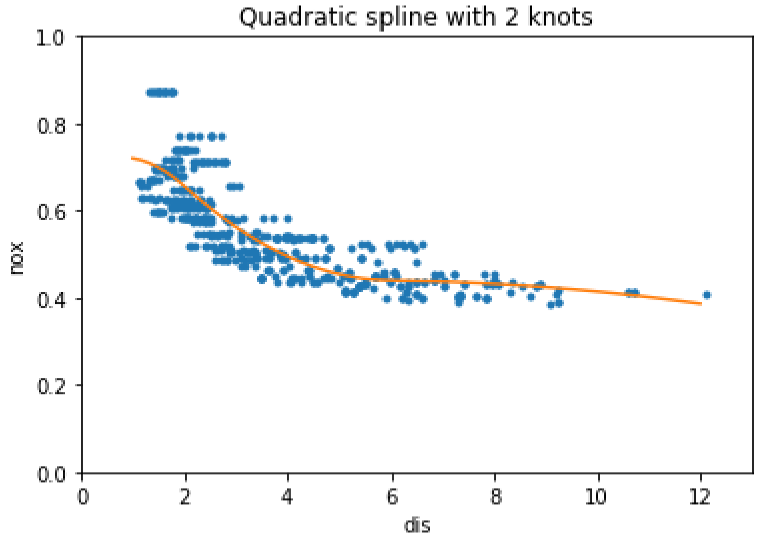
In [23]:
```
def spline(knots, degree):

    # Dividing data into train and validation datasets
    X = df_copy['dis']
    y = df_copy['nox']
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)


    # degrees of freedom = degree + knots

    # ---------------- Generate Cubic Spline with 1 knots ------------------
    # Generating cubic spline with 3 knots

    transformed_x_train = dmatrix("bs(train, knots=" + knots + ", degree=" + degree + ", include_intercept=Fa
lse)",
                                  {"train": X_train},
                                  return_type='dataframe')
    # Fitting Generalised linear model on transformed dataset
    spline = sm.GLM(y_train, transformed_x_train).fit()


    # --------------- Make Predictions with Splines -------------------------
    # Predictions on both splines
    transformed_x_test = dmatrix("bs(valid, knots=" + knots +  ", degree=" + degree + ", include_intercept=Fa
lse)",
                                       {"valid": X_test},
                                       return_type='dataframe')
    y_pred = spline.predict(transformed_x_test)


    # Calculating RSS values
    RSS = np.sum((y_test - y_pred)**2)
    print("The RSS of the below fitted spline is " + repr(RSS))



    # --------------- PLotting the Splines ------------------

    # We will plot the graph for 70 observations only
    x_values = np.linspace(1, 12, 70)

    # Make predictions on x-values using fitted splines
```

```
    transformed_x_values = dmatrix("bs(xp, knots=" + knots + ", degree=" + degree + ", include_intercept=Fals
e)",
                                   {"xp": x_values},
                                   return_type='dataframe')
    y_pred = spline.predict(transformed_x_values)

    # Plot the splines and error bands
    plt.plot(X, y, '.')
    plt.plot(x_values, y_pred)
    plt.xlim(0,13)
    plt.ylim(0,1)
    plt.xlabel('dis')
    plt.ylabel('nox')
    plt.show()

plt1 = plt.figure(1)
plt.title("Quadratic spline with 1 knot")
spline("(2,)", "2")

plt1 = plt.figure(2)
plt.title("Quadratic spline with 2 knots")
spline("(2,6)", "2")

plt1 = plt.figure(3)
plt.title("Quadratic spline with 3 knots")
spline("(2,3,6)", "2")

plt1 = plt.figure(4)
plt.title("Quadratic spline with 4 knots")
spline("(2,3,4,6)", "2")

plt1 = plt.figure(5)
plt.title("Quadratic spline with 5 knots")
spline("(2,3,4,6,9)", "2")
```

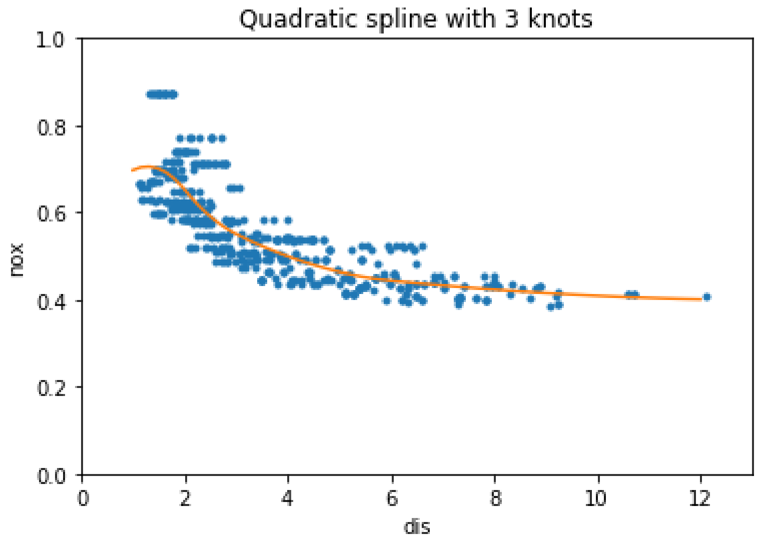
The RSS of the below fitted spline is 0.7136620830924438



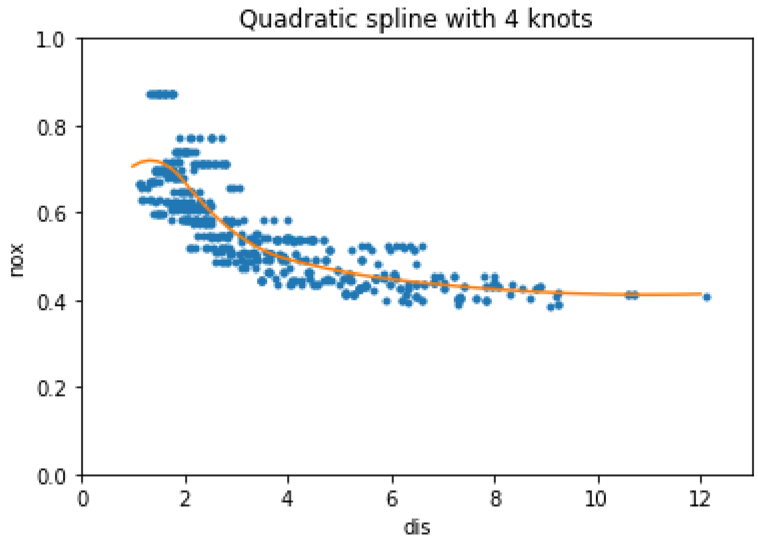
The RSS of the below fitted spline is 0.48386490597023546



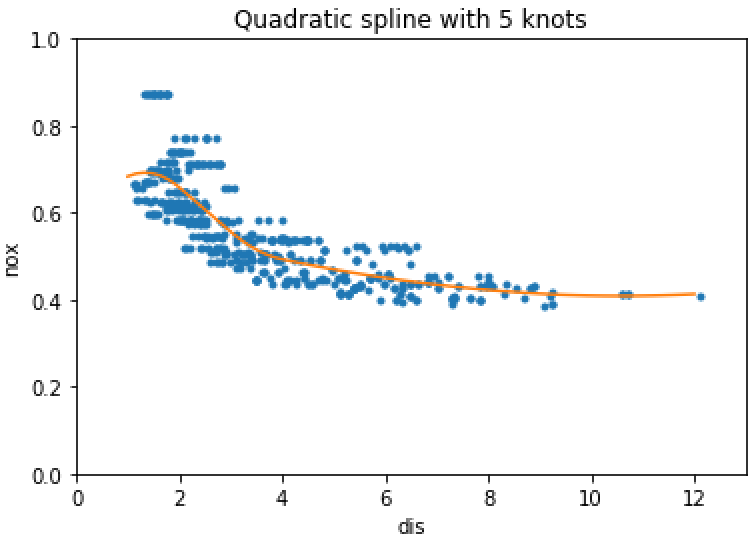
The RSS of the below fitted spline is 0.5608284286576994



The RSS of the below fitted spline is 0.623102769824426



The RSS of the below fitted spline is 0.6352010793514369


Quadratic spline with 5 knots
nox
dis
1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8
10
12

Problem f

In [21]:
```
def spline_cv(knots, degree):
    X = df_copy['dis']
    X_array = np.reshape(np.asarray(X), (-1,1))
    y = df_copy['nox']
    y_array = np.reshape(np.asarray(y), (-1,1))

    kfold = KFold(n_splits=10, shuffle=True)
    sum_test_errors = 0
    for train_index, test_index in kfold.split(X_array):
        # print("TRAIN:", train_index, "TEST:", test_index)
        X_train, X_test = X_array[train_index], X_array[test_index]
        y_train, y_test = y_array[train_index], y_array[test_index]

        # Generating spline and fit to training data
        transformed_x_train = dmatrix("bs(train, knots=" + knots + ", degree=" + degree + ", include_intercep
t=False)",
                                {"train": X_train},
                                return_type='dataframe')
        # Fitting Generalised linear model on transformed dataset
        spline = sm.GLM(y_train, transformed_x_train).fit()

        # Make predictions on testing data
        transformed_x_test = dmatrix("bs(valid, knots=" + knots +  ", degree=" + degree + ", include_intercep
t=False)",
                                     {"valid": X_test},
                                     return_type='dataframe')
        y_pred = spline.predict(transformed_x_test)

        # Calculating RSS values
        y_test = y_test.ravel()
        y_pred = y_pred.ravel()
        RSS = np.sum((y_test - y_pred)**2)

        sum_test_errors = sum_test_errors+RSS

    current_test_error = sum_test_errors/K
    return current_test_error
```

In [22]:

```
error1 = spline_cv("(2,)", "2")
error2 = spline_cv("(2,6)", "2")
error3 = spline_cv("(2,2.5,6)", "2")
error4 = spline_cv("(2,2.5,3.5,5)", "2")
error5 = spline_cv("(2,2.5,3.5,5,6)", "2")
error6 = spline_cv("(2,2.5,3.5,5,6,6.5)", "2")
error7 = spline_cv("(2,2.5,3.5,5,6,6.5,7)", "2")

print("The test error for a spline with 3 d.o.f. was " + repr(error1))
print("The test error for a spline with 4 d.o.f. was " + repr(error2))
print("The test error for a spline with 5 d.o.f. was " + repr(error3))
print("The test error for a spline with 6 d.o.f. was " + repr(error4))
print("The test error for a spline with 7 d.o.f. was " + repr(error5))
print("The test error for a spline with 8 d.o.f. was " + repr(error6))
print("The test error for a spline with 9 d.o.f. was " + repr(error7))
```

```
The test error for a spline with 3 d.o.f. was 0.23502044938832448
The test error for a spline with 4 d.o.f. was 0.19763944200417297
The test error for a spline with 5 d.o.f. was 0.19076251851533974
The test error for a spline with 6 d.o.f. was 0.1935515463269743
The test error for a spline with 7 d.o.f. was 0.19275111344987855
The test error for a spline with 8 d.o.f. was 0.19417028939887468
The test error for a spline with 9 d.o.f. was 0.19185318240580793
```

In [ ]:

In [ ]: