ESE 542: Statistics for Data Science **Spring 2019**

HW 5

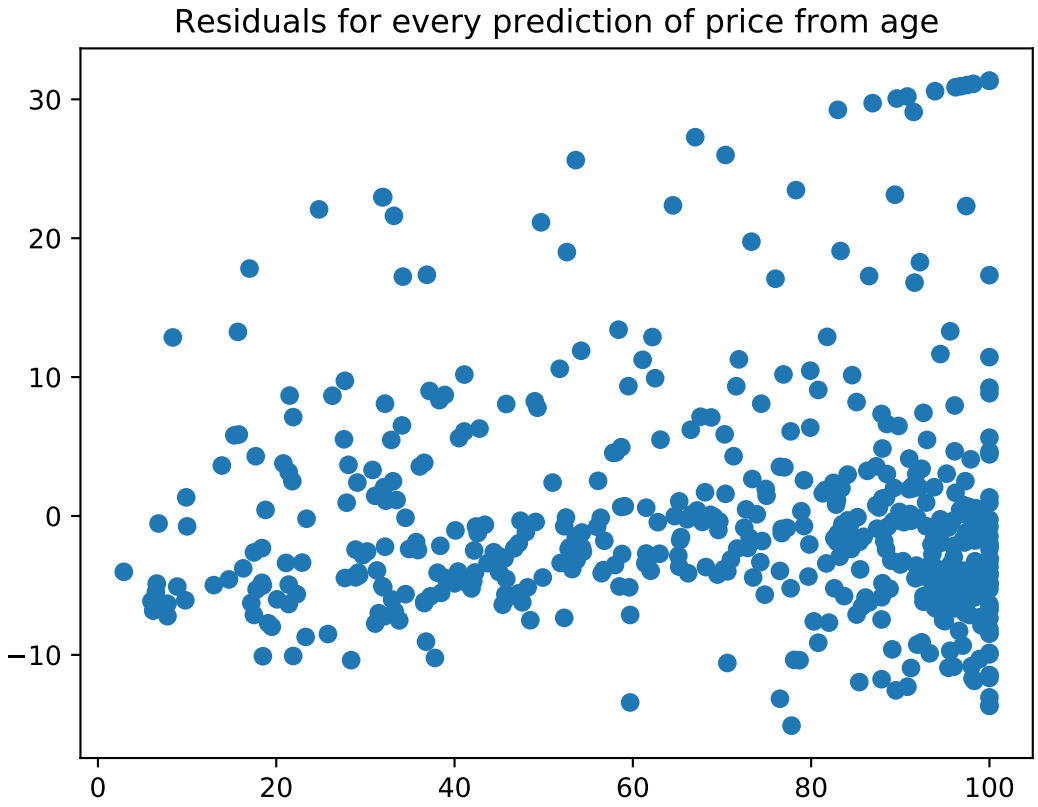
Name: *Tyler Olivieri*

1. (a) The model summary shows $\beta_0 = 30.9787$ and $\beta_1 = -.12$, β_1 is just the slope of the line in the linear model. It explains the rate of change of the price with age. or the average effect on price of one unit increase in age. The hypothesis test corresponding to β_1 has the null hypothesis that $\beta_1 = 0$ (no relationship between age and price). With a p-value of 0, we reject this hypothesis with any significance, meaning there is certainly a relationship ($\beta_1 \neq 0$).

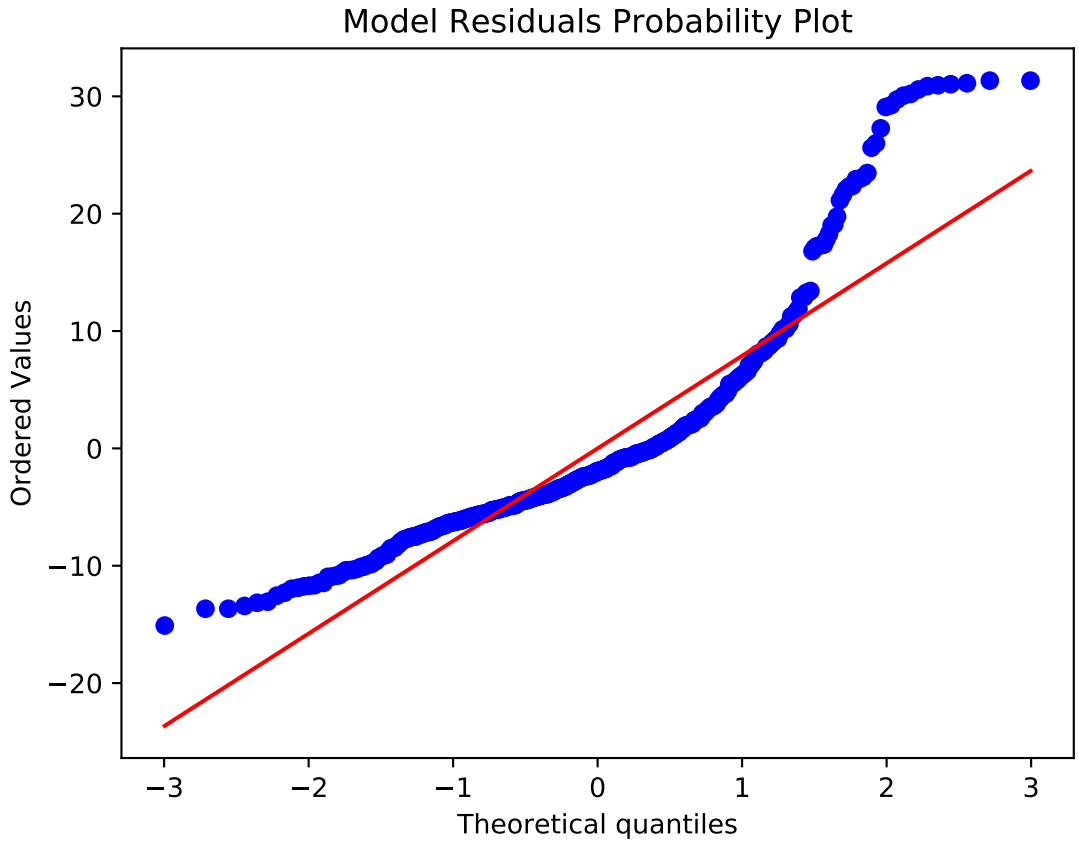
   (b) The R^2 value is .363, meaning that the model did not approximate the real data very well. There is a lot of variance in the data not explained by the model.

   (c) There are two parameters β_0 and β_1. The residual degrees of freedom is n-2 (the 2 comes from 2 parameters). It is the dimension of the space of the residuals. The residuals are constrained by two equations $\widehat{e}_1 + ... + \widehat{e}_n = 0$ and $x_1\widehat{e}_1 + ... + x_n\widehat{e}_n = 0$, then we have n-2 degrees of freedom. The residuals form a space of dimension n-2.

   (d) I expect the residuals to be without a strong pattern for a linear (true) model. A pattern in the residuals would indicate some non-linearity in the data. I do not see much of a pattern, perhaps it is not centered around 0, but the residuals have high variance and a K-S test shows that that the residuals are not distributed normally.



(e) If the model is true, I expect the probability plot to be directly on the linear line. However, we see that is not quite the case here.



(f) Yes and no. We have a significantly significant predictor for age and price, however, we have a low r-squared which suggests that we are not explaining the variance in the model very well. Also, the probability plot and kstest suggests that the model does not have normally distributed residuals. The non-normally distributed residuals suggests some potential pattern in the data the linear model cannot capture.

I think that the low r-squared could be OK in this scenario, meaning that the housing price has a lot of variance or complexity, and a linear model can't capture this. I would use this predictor over nothing, but would seek out a better predictor, most likely try something non-linear.

```
from sklearn.datasets import load_boston
import statsmodels.formula.api as smf
import statsmodels.api as sm
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats

# load the boston house price dataset fro sklearn
boston = load_boston()
boston_age_org = boston.data[:,6]
boston_price = boston.target

# (a)
```

```
# find how the variable age relates to housing prices in Boston
# assume model medv_i = B_0 + B_1*age_i + e_i where e_i ~ N(0,sigma^2)

# add intercept for B_0
boston_age = sm.add_constant(boston_age_org)
model = sm.OLS(boston_price,boston_age).fit()

print(model.summary())

# print data vs predictions
#fig, ax = plt.subplots(figsize=(8,6))
#ax.plot(boston_age_org, boston_price, 'o', label="data")
#ax.plot(boston_age_org, model.fittedvalues, 'r--.', label="OLS")
#ax.legend(loc='best');
#plt.show()

# (d) Recall that the residuals are the difference between the true response and the p

plt.scatter(boston_age_org, model.resid)
plt.title("Residuals for every prediction of price from age")
plt.show()

# (e) generate probability plot
stats.probplot(model.resid, plot= plt)
plt.title("Model Residuals Probability Plot")
plt.show()

print(stats.kstest(model.resid, 'norm'))
```

Output of Regression:

==========

Dep. Variable: y R-squared: 0.142

Model: OLS Adj. R-squared: 0.140

Method: Least Squares F-statistic: 83.48

Date: Thu, 04 Apr 2019 Prob (F-statistic): 1.57e-18

Time: 17:50:32 Log-Likelihood: -1801.5

No. Observations: 506 AIC: 3607.

Df Residuals: 504 BIC: 3615.

Df Model: 1

Covariance Type: nonrobust

==========

coef std err t P¿—t— [0.025 0.975]

---

const 30.9787 0.999 31.006 0.000 29.016 32.942

x1 -0.1232 0.013 -9.137 0.000 -0.150 -0.097

==========

Omnibus: 170.034 Durbin-Watson: 0.613
Prob(Omnibus): 0.000 Jarque-Bera (JB): 456.983
Skew: 1.671 Prob(JB): 5.85e-100
Kurtosis: 6.240 Cond. No. 195.
==========

2. (a) The covariance matrix for our predictors is

$$\begin{bmatrix} - & NOX & RM & DIS & LSTAT \\ NOX & 1. & -0.30218819 & -0.76923011 & 0.59087892 \\ RM & -0.30218819 & 1. & 0.20524621 & -0.61380827 \\ DIS & -0.76923011 & 0.20524621 & 1. & -0.49699583 \\ LSTAT & 0.59087892 & -0.61380827 & -0.49699583 & 1. \end{bmatrix}$$

(b) The VIF calculation for each predictor is

$$\begin{bmatrix} CONSTANT & NOX & RM & DIS & LSTAT \\ 276.741 & 2.853 & 1.639 & 2.496 & 2.299 \end{bmatrix}$$

Multi-collinearity is a problem because coefficient estimates of multiple regression can change erraticaly in response to small changes in the model or data. In perfect multicolinearity, the OLS estimator optimal solutions do not exist. $X^T X$ does not have an inverse. In multicolinearity, X is not full rank.

(c) Since the VIF values calculated were all under 10, we do not need to drop any predictors for this model. The solution to the linear regression is below.
OLS Regression Results
==========
Dep. Variable: y R-squared: 0.659
Model: OLS Adj. R-squared: 0.656
Method: Least Squares **F-statistic: 241.6**
Date: Thu, 04 Apr 2019 **Prob (F-statistic): 2.09e-115**
Time: 18:14:47 Log-Likelihood: -1568.4
No. Observations: 506 AIC: 3147.
Df Residuals: 501 BIC: 3168.
Df Model: 4
Covariance Type: nonrobust
==========
coef std err t P¿—t— [0.025 0.975]

---

const 12.0360 3.990 3.016 0.003 4.196 19.876
x1 -14.5379 3.500 -4.154 0.000 -21.414 -7.662
x2 4.8634 0.438 11.115 0.000 4.004 5.723
x3 -0.9670 0.180 -5.367 0.000 -1.321 -0.613
x4 -0.6587 0.051 -12.917 0.000 -0.759 -0.558
==========
Omnibus: 128.609 Durbin-Watson: 0.866
Prob(Omnibus): 0.000 Jarque-Bera (JB): 362.399
Skew: 1.222 Prob(JB): 2.02e-79
Kurtosis: 6.349 Cond. No. 311.
==========

(d) The F-statistic was computed and shown in the model summary above. Due to the F-statistic being high, with a very small p-value, we can reject the null hypothesis. The null hypothesis is that the intercept only model (no predictor variables) is equal to the model with the predictor variables. In other words, our model with the predictor variables gives a better fit than a model with just a y-intercept or at least one predictor coeficient, $\beta_j \neq 0$.

```
from sklearn.datasets import load_boston
import statsmodels.formula.api as smf
import statsmodels.api as sm
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
import statsmodels.stats.outliers_influence as st

# load the boston house price dataset from sklearn
boston = load_boston()
boston_price = boston.target
boston_nox = boston.data[:,4]
boston_rm = boston.data[:,5]
boston_dis = boston.data[:,7]
boston_lstat = boston.data[:,12]
print(boston.feature_names[4])
print(boston.feature_names[5])
print(boston.feature_names[7])
print(boston.feature_names[12])
# (a)
# Using perason correlation, compute and report the correlation matrix for our four pr
#
corr_matrix = np.corrcoef([boston_nox, boston_rm, boston_dis, boston_lstat])
print(corr_matrix)


# (b)
# Compute variance inflation factor (VIF) for each of the predictor variables
features = np.column_stack([boston_nox, boston_rm, boston_dis, boston_lstat])
features = sm.add_constant(features)
vif = [st.variance_inflation_factor(features, i) for i in range(features.shape[1])]
print(vif)


# (c)
# Drop predictors with VIF values over 10
# run linear regression on remaining variables to predict the target

model = sm.OLS(boston_price,features).fit()
print(model.summary())
```

3. (a) The OLS results for the model is shown below.

OLS Regression Results

==========

Dep. Variable: y R-squared: 0.229

Model: OLS Adj. R-squared: 0.226
Method: Least Squares F-statistic: 74.65
Date: Thu, 04 Apr 2019 Prob (F-statistic): 4.08e-29
Time: 18:42:34 Log-Likelihood: -1774.5
No. Observations: 506 AIC: 3555.
Df Residuals: 503 BIC: 3568.
Df Model: 2
Covariance Type: nonrobust

==========

coef std err t P¿—t— [0.025 0.975]

---

const 41.6722 1.762 23.652 0.000 38.211 45.134
x1 7.8224 1.424 5.494 0.000 5.025 10.620
x2 -35.4798 3.121 -11.369 0.000 -41.611 -29.349

==========

Omnibus: 156.666 Durbin-Watson: 0.744
Prob(Omnibus): 0.000 Jarque-Bera (JB): 383.211
Skew: 1.581 Prob(JB): 6.12e-84
Kurtosis: 5.859 Cond. No. 11.4

==========

(b) β_0 explains average price of houses not bordering the Charles River.

(c) β_1 explains the average difference between houses that border the Charles river and those that don't.

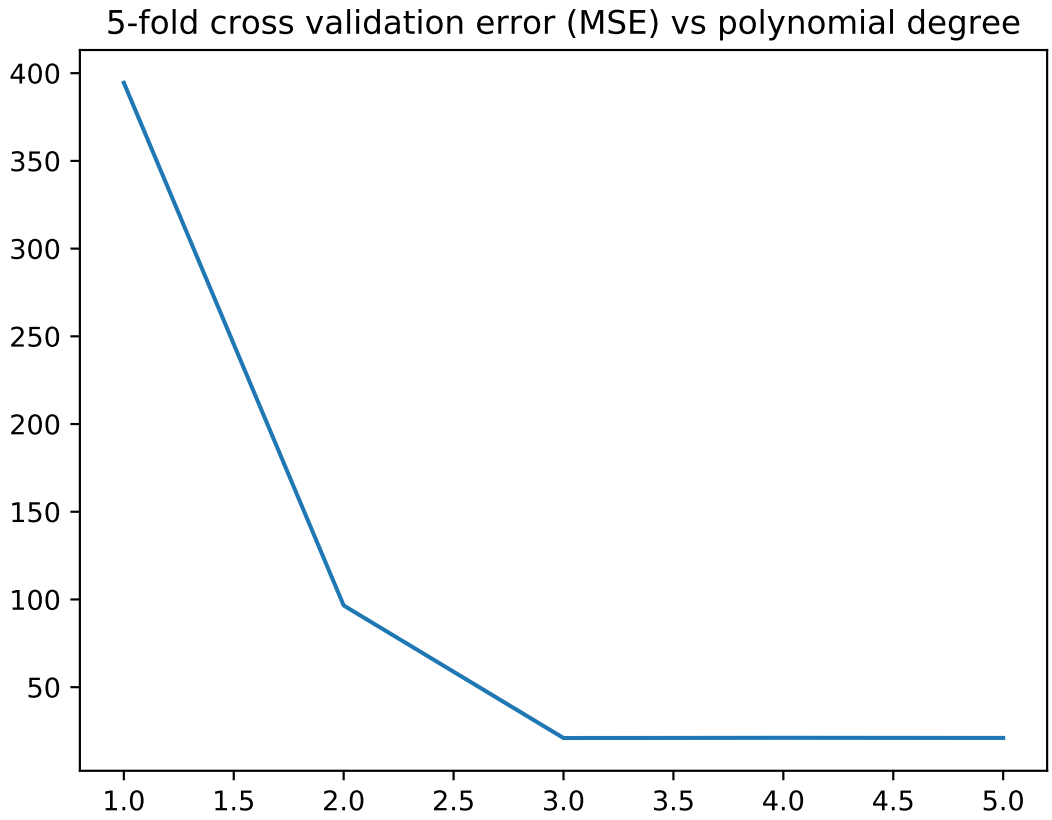
(d) We don't include two dummy variables because then there is a problem with multi-colinearity. If one variable is 0, the other is 1. There is a perfect relationship between them. Consider that the constant term is a vector of ones. Then, by construction, the two dummy variables will add to be a vector of 1s. This occurs because When a variable is 0, the other is 1 and vice versa. $0+1 = 1+0 = 1$ Therefore, the intercept is a linear combination of the dummy variables, creating the co-linearity.

```
from sklearn.datasets import load_boston
import statsmodels.formula.api as smf
import statsmodels.api as sm
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
import statsmodels.stats.outliers_influence as st

# load the boston house price dataset from sklearn
boston = load_boston()
boston_price = boston.target
boston_nox = boston.data[:,4]
boston_chas = boston.data[:,3]

# (a) fit OLS model to medv_i = B_0 + B_1*chas_i + B2*nox + e_i
features = np.column_stack([boston_chas, boston_nox])
features = sm.add_constant(features)
model = sm.OLS(boston_price,features).fit()
print(model.summary())
```

4. (a) The value of k used was 5. The plot shows the k-fold cross validation error.



(b) I chose the smallest polynomial model (smallest in degree) that gave the lowest MSE. It actually flattened up to around 10, with minor variations or actually increasing. The polynomial with the smallest MSE was a polynomial with degree 3, $\beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3$. The model results and the fitted polynomial against the labels (with confidence intervals) are plotted below.

OLS Regression Results

==========

Dep. Variable: y R-squared: 0.947

Model: OLS Adj. R-squared: 0.946

Method: Least Squares F-statistic: 1161.

Date: Thu, 04 Apr 2019 Prob (F-statistic): 1.79e-124

Time: 20:20:55 Log-Likelihood: -584.23

No. Observations: 200 AIC: 1176.

Df Residuals: 196 BIC: 1190.

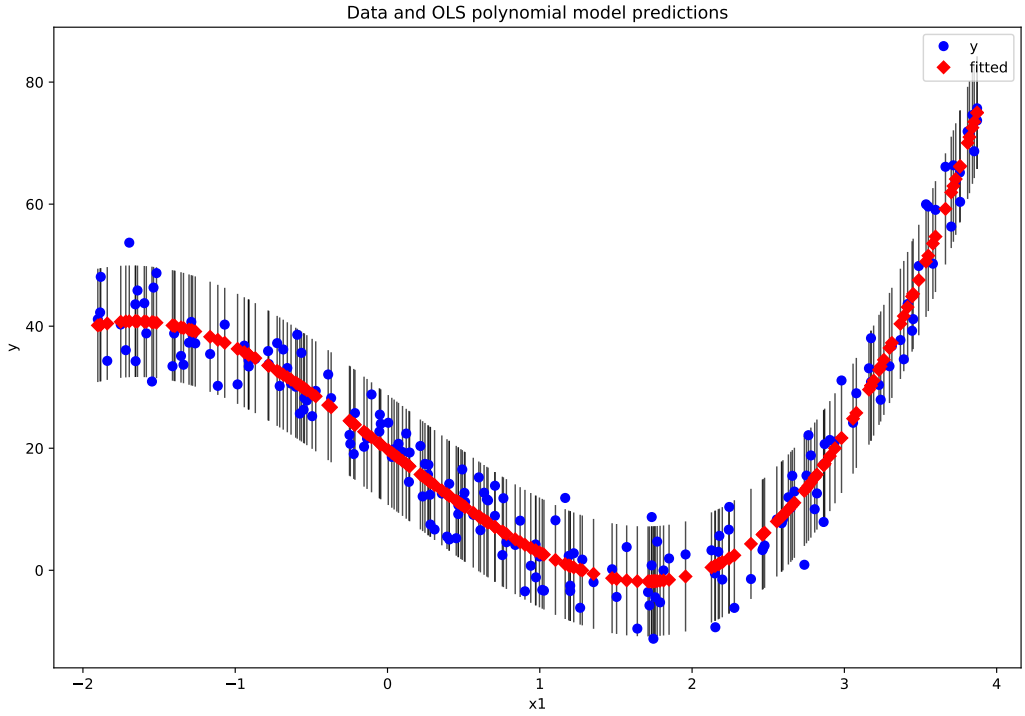
Df Model: 3

Covariance Type: nonrobust

==========

coef std err t P¿—t— [0.025 0.975]

```
const 19.7951 0.536 36.941 0.000 18.738 20.852
x1 -19.0345 0.379 -50.192 0.000 -19.782 -18.287
x2 -0.1079 0.279 -0.386 0.700 -0.659 0.443
x3 2.2490 0.084 26.672 0.000 2.083 2.415
==============================================================================
Omnibus: 0.116 Durbin-Watson: 1.954
Prob(Omnibus): 0.944 Jarque-Bera (JB): 0.206
Skew: 0.053 Prob(JB): 0.902
Kurtosis: 2.883 Cond. No. 37.5
==============================================================================
```




```
import numpy as np
import csv
from sklearn.model_selection import KFold
import statsmodels.api as sm
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt

file_name = "/home/snazyman/stat_ds/data_science/hw5/poly_data_space.csv"

with open(file_name) as csvfile:
    i = 0
    data_reader = csv.reader(csvfile)
    predictor = np.empty([200,1], dtype=float)
    target = np.empty([200,1], dtype=float)
```

```
for row in data_reader:
    predictor[i] = float(row[0])
    target[i] = float(row[1])
    i = i + 1

# pick k here ~ n_splits
k = 5
kf = KFold(n_splits = k)

# pick a set of polynomial models
degree_list = [1, 2, 3, 4, 5]

CV_error = []

# loop over them
for poly_model in degree_list:
    i = 0

    # construct model
    # constant (for intercept)
    features = np.ones([200,1], dtype=float)

    MSE = np.zeros([k,1], dtype=float)

    # append polynomial powers of the predictor
    for power in range(1,poly_model+1):
        feature_power = np.power(predictor, power)
        features = np.column_stack([features, feature_power])

    # train and evaluate model
    for train_index, test_index in kf.split(predictor):

        # get current folds index(s)
        features_train, features_test = features[train_index], features[test_index
        target_train, target_test = target[train_index], target[test_index]


        # train polynomial model
        model = sm.OLS(target_train,features_train).fit()

        # evaluate polynomial model on test fold
        target_predicted = model.predict(features_test)

        # compute MSE
        MSE[i] = mean_squared_error(target_test, target_predicted)
        i = i + 1

    # average MSE over k folds to get k fold cross validation error
    CV_error.append(MSE.mean())
```

```
# plot results
print(CV_error)
plt.plot(degree_list, CV_error)
plt.title(f"{k}-fold cross validation error (MSE) vs polynomial degree")
plt.show()

# (b) choose best model (model with lowest {k}-fold cross validation error (MSE))
# run model on entire dataset. Report the coefficients. Plot the data and fitted p
idx = np.argmin(CV_error)
degree = degree_list[idx]

# recreate best model with best degree
features = np.ones([200,1], dtype=float)
for power in range(1,degree+1):
    feature_power = np.power(predictor, power)
    features = np.column_stack([features, feature_power])

# train model
model_final = sm.OLS(target,features).fit()
print(model_final.summary())

# print data vs predictions
fig, ax = plt.subplots(figsize=(12, 8))
fig = sm.graphics.plot_fit(model_final,1, ax=ax)

ax.legend(loc='best');
plt.title("Data and OLS polynomial model predictions")
plt.show()
```