Chapter 2: Predictions and model objects

Created	@September 18, 2024 11:44 PM
Class	Introduction to Regression with statsmodels in Python

Making Predictions with Linear Regression Models

Key Details

- Focuses on using linear regression models to make predictions
- Uses fish dataset (bream) with length as explanatory variable and mass as response variable
- Demonstrates how to create and visualize predictions

1. Fitting the Model

```
from statsmodels.formula.api import ols

model = ols(formula="mass_g ~ length_cm", data=bream_data).fi
```

```
t()
print(model.params)
```

2. Creating Prediction Data

2.1 Generating Explanatory Data

```
import numpy as np
import pandas as pd

explanatory_data = pd.DataFrame({
    'length_cm': np.arange(20, 40)
})
```

3. Making Predictions

3.1 Using the predict() Method

```
predictions = model.predict(explanatory_data)
```

3.2 Combining Predictions with Explanatory Data

```
prediction_data = explanatory_data.assign(
    mass_g=model.predict(explanatory_data)
)
```

4. Visualizing Predictions

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

fig = plt.figure()
sns.regplot(x='length_cm', y='mass_g', data=bream_data)
```

```
sns.scatterplot(x='length_cm', y='mass_g', data=prediction_da
ta, color='red', marker='s')
plt.show()
```

5. Extrapolation

5.1 Making Predictions Outside the Data Range

```
small_bream = pd.DataFrame({'length_cm': [10]})
print(model.predict(small_bream))
```

5.2 Cautions about Extrapolation

- Can lead to unrealistic or impossible predictions
- Understanding data context is crucial for determining appropriateness of extrapolation

Key Takeaways

- Linear regression allows predictions based on explanatory variables
- Use pandas DataFrame to structure prediction input data
- Combine predictions with input data for easier analysis
- Visualize predictions alongside original data to understand model performance
- Be cautious when extrapolating beyond the range of original data

Extracting Information from Linear Regression Models

Key Details

- Explores various attributes and methods of fitted OLS model objects
- Demonstrates how to access and interpret model information

- Focuses on coefficients, fitted values, residuals, and summary statistics

1. Accessing Model Coefficients

```
model.params
```

- Returns a pandas Series with intercept and slope

2. Fitted Values

```
model.fittedvalues
```

- Predictions on the original dataset
- Equivalent to `model.predict(original_data)`
- Returns a pandas Series of length equal to the number of rows in the original dataset

3. Residuals

```
model.resid
```

- Measure of model inaccuracy
- Calculated as actual response value minus predicted response value
- One residual per row in the original dataset

4. Model Summary

```
print(model.summary())
```

4.1 Summary Components

1. Dependent variable(s) and regression type
2. Model performance metrics

3. Coefficient details
    - Coefficient values (same as `params`)
    - Standard errors
    - t-statistics
    - p-values (for statistical significance)
4. Diagnostic statistics

5. Visualizing Residuals

- Can be plotted as vertical lines on a regression plot
- Each line represents the difference between actual and predicted values

Key Takeaways

- OLS model objects contain rich information about the fitted model
- `params` provides quick access to model coefficients
- `fittedvalues` gives predictions for the original dataset
- `resid` helps assess model accuracy
- `summary()` method provides a comprehensive overview of the model, including performance metrics and statistical tests
- Understanding these components is crucial for model interpretation and diagnostics

Regression to the Mean

Key Details

- Concept related to, but distinct from, regression modeling
- A property of data, not a type of model
- Linear regression can quantify its effect

- Explains why extreme cases tend to become more average over time

Components of Response Values

- Fitted value (prediction by the model)
- Residual (difference between actual and predicted)

Reasons for Residuals

1. Model imperfections
2. Inherent randomness in data

Concept Explanation

- Extreme responses often due to randomness or luck
- Extremes don't persist over time as luck runs out
- Eventually, extreme cases will look more like average cases

Classic Dataset: Father-Son Heights

- Collected by Karl Pearson
- Over 1000 pairs of father-son height measurements
- 19th-century study on biological inheritance

Visualization

```
import matplotlib.pyplot as plt

df = pd.read_csv("Pearson.txt", sep="\t", header=0)

# Scatter plot
plt.scatter(fathers_heights, sons_heights)

# Line of equal height
plt.axline((0, 0), slope=1, color='green', linewidth=2)
```

```
# Set equal axis scaling
plt.axis('equal')

# Add regression line
sns.regplot(x=fathers_heights, y=sons_heights, color='black')

plt.xlabel("Fathers' Heights")
plt.ylabel("Sons' Heights")
plt.show()
```

Observations

- Regression line less steep than line of equal height
- Short fathers tend to have taller sons (on average)
- Tall fathers tend to have shorter sons (on average)

Quantifying Predictions

- Sons' heights as response variable
- Fathers' heights as explanatory variable

Example Predictions

- Very tall father (190 cm): Predicted son's height = 183 cm
- Very short father (150 cm): Predicted son's height = 163 cm

Key Takeaways

- Extreme values tend to become less extreme in subsequent measurements or generations
- Linear regression can be used to quantify the regression to the mean effect
- Important concept in various fields, including sports and finance

Transforming Variables in Linear Regression

Key Details

- Sometimes the relationship between explanatory and response variables is not linear
- Transforming variables can help fit a linear regression model
- Common transformations: cubing, square root
- Transformations can be applied to explanatory variable, response variable, or both

Example 1: Perch Mass vs Length

Data Characteristics

- Upward curve in mass vs length data for perch
- Contrast with bream, which had a strong linear relationship

Transformation Applied

- Length cubed (x^3) used as explanatory variable

```
# Create new column with length cubed
perch_data['length_cubed'] = perch_data['length'] ** 3

# Plot with transformed x-axis
sns.regplot(x='length_cubed', y='mass', data=perch_data)
plt.xlabel('Length³')
plt.ylabel('Mass')
plt.show()
```

Modeling with Transformed Variable

```
model = smf.ols(formula='mass ~ length_cubed', data=perch_dat
a).fit()

# Create explanatory DataFrame
explanatory_data = pd.DataFrame({'length_cubed': [length**3 f
or length in lengths],
                                 'length': lengths})

# Add predictions
explanatory_data = explanatory_data.assign(mass_pred=model.pr
edict(explanatory_data))
```

Visualization

- Plot mass vs length cubed
- Add predictions as red points
- Plot mass vs original length to show non-linear predictions

Example 2: Facebook Advertising Dataset

Data Characteristics

- Impressions vs spend data
- Majority of points cramped in bottom-left of original plot

Transformation Applied

- Square root transformation on both variables

```
sns.regplot(x=np.sqrt(fb_data['spent_usd']),
            y=np.sqrt(fb_data['impressions']))
plt.xlabel('√Spend (USD)')
plt.ylabel('√Impressions')
plt.show()
```

Modeling with Transformed Variables

```
model = smf.ols(formula='np.sqrt(impressions) ~ np.sqrt(spent
_usd)', data=fb_data).fit()

explanatory_data = pd.DataFrame({'spent_usd_sqrt': np.sqrt(sp
ends),
                                 'spent_usd': spends})

# Predict and back-transform
explanatory_data = explanatory_data.assign(
    impressions_pred=model.predict(explanatory_data)**2
)
```

Key Takeaways

- Transformations can linearize relationships for better model fit
- Common transformations: cubing for 3D growth, square root for right-skewed distributions
- Back-transformation may be necessary when predicting with transformed response variables
- Visualization before and after transformation helps in assessing model fit