Learning to Predict with Regression Models in Statsmodels (Python)

Authored by

Mohammed looti

October 27, 2025

The Power of Prediction in Statistical Modeling

One of the most valuable capabilities afforded by a properly constructed regression model is its ability to generate reliable forecasts on novel, previously unseen data points. This forecasting capability is central to modern data science and decision-making across virtually all industries. Within the ecosystem of Python, the powerful Statsmodels library offers a sophisticated and statistically rigorous framework not only for fitting complex statistical models but also for efficiently translating those models into actionable predictions.

This comprehensive guide is designed to walk you through the precise methodology required to harness a fitted regression model from the **Statsmodels** module and apply it to forecast outcomes for new observations. The core mechanism for achieving this is highly streamlined and relies on a remarkably intuitive syntax:

model.predict(df_new)

This concise command is engineered to compute the predicted response values for every observation contained within a new data structure, typically formatted as a Pandas DataFrame, which we refer to generically as **df_new**. It seamlessly leverages the coefficients and statistical parameters learned during the training phase of your previously fitted **Statsmodels** regression object, represented here as **model**. Mastering this application is absolutely critical for practical data analysis, effective business forecasting, and turning statistical theory into tangible results.

Preparing the Foundation: Structuring Data for Regression Analysis

To provide a clear, practical demonstration of the prediction process, we must first establish and prepare a suitable sample dataset. For this example, we will utilize a Pandas DataFrame that simulates hypothetical student performance data. This dataset incorporates key features: the number of **hours studied**, the total count of **prep exams taken**, and the resulting **final score** achieved by each student in the cohort.

This initial DataFrame will serve as our authoritative training data, the empirical basis from which we will derive and estimate our regression model. Before proceeding to the computationally intensive model fitting phase, it is paramount that the data is well-structured, clean, and appropriately formatted, with clear definitions for both the input features (predictors) and the target variable (response).

import pandas as pd

#create DataFrame
df = pd.DataFrame({'hours': ,

'exams': ,
'score': })

#view head of DataFrame
df.head()

hours exams score
0 1 1 76
1 2 3 78
2 2 3 85
3 4 5 88
4 2 2 72

The resulting structure clearly shows that each row corresponds to a single student observation, and the columns detail their respective academic metrics. Our primary analytical objective is to successfully predict the student's **score** by modeling its relationship with the independent variables: **hours studied** and **exams taken**. This setup ensures we have the necessary input to move forward with statistical estimation.

Fitting the Model: Implementing OLS using Statsmodels

Once the training data is prepared, the subsequent critical step involves fitting a linear regression model. We will utilize the highly conventional and robust Ordinary Least Squares (OLS) function, which is readily available within the Statsmodels module. Within the context of our model specification, the variables **"hours"** and **"exams"** will function as our primary predictor variables, collectively influencing the behavior of the response variable, which is the **"score"**.

The model fitting procedure requires several specific actions. First, we clearly define our independent variables (denoted as **x**) and the dependent variable (denoted as **y**). A mandatory prerequisite for fitting an OLS model using **Statsmodels** is the explicit inclusion of a constant term to the predictor variables using the function `sm.add_constant(x)`. This constant represents the model's intercept, which is the estimated value of the response when all predictor variables are set to zero. Omitting this step would incorrectly force the regression plane to pass through the origin, severely limiting the model's accuracy and applicability to real-world data. Finally, the model is fitted and instantiated as a regression object using the command `sm.OLS(y, x).fit()`.

Following the successful fitting process, invoking `model.summary()` produces a detailed statistical report on the regression outcomes. This summary is indispensable for comprehensive model evaluation, providing metrics like the **R-squared** (which quantifies the proportion of variance explained by the predictors) and **p-values** (which assess the statistical significance of each

predictor). For the specific purpose of prediction, the **"coef"** column is of chief importance, as these values represent the estimated coefficients that precisely define the linear relationship between our input features and the expected response.

import statsmodels.api as sm

#define predictor and response variables
y = df
x = df]

#add constant to predictor variables
x = sm.add_constant(x)

#fit linear regression model
model = sm.OLS(y, x).fit()

#view model summary
print(model.summary())

OLS Regression Results
==========
===
Dep. Variable: score R-squared: 0.718
Model: OLS Adj. R-squared: 0.661
Method: Least Squares F-statistic: 12.70
Date: Fri, 05 Aug 2022 Prob (F-statistic): 0.00180
Time: 09:24:38 Log-Likelihood: -38.618
No. Observations: 13 AIC: 83.24
Df Residuals: 10 BIC: 84.93
Df Model: 2
Covariance Type: nonrobust
==========
===
coef std err t P>|t|
---
const 71.4048 4.001 17.847 0.000 62.490 80.319
hours 5.1275 1.018 5.038 0.001 2.860 7.395
exams -1.2121 1.147 -1.057 0.315 -3.768 1.344
==========
===
Omnibus: 1.103 Durbin-Watson: 1.248

Prob(Omnibus): 0.576 Jarque-Bera (JB): 0.803
Skew: -0.289 Prob(JB): 0.669
Kurtosis: 1.928 Cond. No. 11.7
==========
===

By extracting the estimated coefficients from the summary table, we can explicitly define our fitted linear regression equation, which serves as the algebraic formula underpinning all subsequent predictions:

Score = 71.4048 + 5.1275(hours) - 1.2121(exams)

This equation is the core mathematical engine that will generate forecasted outcomes for any new data points we introduce.

Forecasting Future Outcomes: Structuring and Applying New Data

With our regression model successfully trained and the coefficients established, the next critical step is applying it to make predictions for a set of new, hypothetical students. This mandates the creation of a new dataset, which must be structured identically to the independent variables used during the initial training phase.

It is absolutely essential that this new dataset, designated **df_new**, maintains structural parity with the model's expectations. This means it must include the exact same predictor variables (**hours** and **exams**) in the correct order. Crucially, because our fitted **Statsmodels** object was trained with an intercept term, the **df_new** must also explicitly incorporate this constant term. We achieve this by applying `sm.add_constant(df_new)`.

Maintaining this structural consistency--especially ensuring the presence of the constant--is not merely a suggestion but a necessity for the `predict()` method to function accurately. If the constant term were omitted from **df_new**, the model would incorrectly assume a zero intercept, resulting in predictions that deviate significantly from the relationship learned during model training. Therefore, this preparatory step of ensuring exact structural alignment between training inputs and prediction inputs is non-negotiable for generating accurate forecasts.

#create new DataFrame
df_new = pd.DataFrame({'hours': ,
'exams': })

#add column for constant
df_new = sm.add_constant(df_new)

#view new DataFrame
print(df_new)

const hours exams
0 1.0 1 1
1 1.0 2 1
2 1.0 2 4
3 1.0 4 3
4 1.0 5 3

With the **df_new** Pandas DataFrame now correctly formatted and containing the necessary constant term, it is fully prepared to be input into our fitted **Statsmodels** object for immediate prediction generation.

Interpreting Predictions and Confirming the Mathematical Basis

The moment of execution arrives when we apply the `predict()` method to the prepared **df_new** DataFrame. This method is the gateway to forecasting, taking the new data as input and returning an array of predicted values corresponding to the **"score"** for each new student observation.

The command `model.predict(df_new)` is both powerful and computationally efficient. It directly translates the coefficients and statistical relationships derived during the training phase to extrapolate outcomes for the unseen data. Specifically, for every row in **df_new**, the model applies the derived regression equation, substituting the new values for **"hours"** and **"exams"** to calculate a corresponding predicted **"score"**. This direct application showcases the practical utility of the regression model, transforming abstract statistical findings into concrete, actionable forecasts.

The output of the `predict()` function is a sequence of values, typically a Pandas Series, where each numerical entry represents the point prediction for the response variable for the respective row in the input DataFrame. It is vital to remember that these are best-estimate forecasts and should be treated as such, acknowledging the inherent uncertainty associated with any statistical prediction.

#predict scores for the five new students
model.predict(df_new)

0 75.320242
1 80.447734
2 76.811480
3 88.278550
4 93.406042

dtype: float64

These results provide the quantitative forecast for each new student's performance. Based on the output, we can interpret the results as follows:

The first student, with 1 hour studied and 1 exam taken, is predicted to achieve a score of approximately **75.32**.
The second student is forecasted to achieve a score of approximately **80.45**.
The third student, despite having 2 hours studied, is penalized by taking 4 exams, resulting in a slightly lower predicted score of **76.81** (reflecting the negative coefficient on the exams variable).

To fully validate the integrity of these predictions, we can revisit the linear regression equation derived from our OLS coefficients:

Score = 71.4048 + 5.1275(hours) - 1.2121(exams)

Let's manually calculate the predicted score for the first student in **df_new** (1 hour, 1 exam) by substituting their feature values into the equation:

Score = 71.4048 + 5.1275(1) - 1.2121(1) = 75.3202

This manual calculation precisely matches the first predicted value generated by the `predict()` method, thereby confirming the mathematical foundation and accuracy of the forecasting process within **Statsmodels**.

Conclusion and Further Steps

The capacity to make accurate and statistically sound forecasts is arguably the most valuable tool in statistical modeling and data science. The Statsmodels library in Python effectively demystifies this complex process, providing a robust, clear, and efficient workflow for generating predictions from any fitted regression model.

By diligently adhering to the structured steps detailed in this guide--encompassing rigorous data preparation, precise model fitting using OLS, ensuring structural parity when preparing new data (including the mandatory intercept), and utilizing the efficient `predict()` method--you gain the confidence to apply your statistical models to anticipate future outcomes. This capability is paramount across diverse professional fields, enabling highly informed, data-driven decision-making based on transparent and robust statistical analysis.

Additional Resources

For those interested in deepening their expertise in Python programming and statistical modeling techniques, we recommend exploring supplementary tutorials. These resources often cover advanced topics, alternative model specifications, and common data manipulation tasks, helping you to expand upon the foundational concepts of regression analysis discussed here and broaden your analytical toolkit.