

Linear Regression Demonstration with Streamlit



Here's a summarized log of the key prompts, rephrased where necessary, along with the corresponding code and key changes. This log demonstrates how the project evolved following the CRISP-DM framework, with each step improving and enhancing the functionality based on natural language prompts.

Prompt 1:

User:

Write Python code to get data points in the form of $y = ax + b + \text{noise}$, where noise follows a normal distribution $N(\text{mean}, \text{variance})$. Notice that a , b , mean, and variance will be variables in the next stage of this project.

Response:

```
python

import numpy as np
import matplotlib.pyplot as plt

def generate_data(a, b, mean, variance, num_points=100, x_range=(0, 10)):
    x = np.linspace(x_range[0], x_range[1], num_points)
    noise = np.random.normal(mean, np.sqrt(variance), num_points)
    y = a * x + b + noise
    return x, y

a = 2.5
b = 5
mean = 0
variance = 4
num_points = 100
x_range = (0, 10)

x, y = generate_data(a, b, mean, variance, num_points, x_range)

plt.scatter(x, y, label="Data points")
plt.plot(x, a * x + b, color='red', label=f"y = {a}x + {b} (without noise)")
plt.title("Generated Data with Noise")
plt.xlabel("x")
plt.ylabel("y")
plt.legend()
plt.show()
```

Prompt 2:

User:

Write Python code to demonstrate how to solve a linear regression problem, following the CRISP-DM 6-step process, using the above function to get data points.

Response:

```
python

import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Step 1: Business Understanding
# Step 2: Data Understanding
def generate_data(a, b, mean, variance, num_points=100, x_range=(0, 10)):
    x = np.linspace(x_range[0], x_range[1], num_points)
    noise = np.random.normal(mean, np.sqrt(variance), num_points)
    y = a * x + b + noise
    return x, y

a = 2.5
b = 5
mean = 0
variance = 4
num_points = 100
x_range = (0, 10)
x, y = generate_data(a, b, mean, variance, num_points, x_range)

# Step 3: Data Preparation
X = x.reshape(-1, 1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 4: Modeling
linear_model = LinearRegression()
linear_model.fit(X_train, y_train)

# Step 5: Evaluation
y_pred = linear_model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Step 6: Deployment
plt.scatter(X_test, y_test, color="blue", label="Test Data")
plt.plot(X_test, y_pred, color="red", label="Regression Line")
plt.title("Linear Regression on Test Data")
plt.xlabel("x")
plt.ylabel("y")
plt.legend()
plt.show()

print(f"Mean Squared Error: {mse:.4f}")
print(f"R-squared: {r2:.4f}")
```

Prompt 3:

User:

Deploy/demonstrate the project using an interactive web interface using Streamlit to introduce linear regression to the user. It is important to let users understand how variables impact the result of linear regression by following these instructions:

1. Allow the user to set variables to get data in the form: $y = aX + b + c * \text{noise}$.
2. Variables (range): n (10 ~ 500), a (-10 ~ 10), b (1 ~ 50), c (0 ~ 100), mean of noise (0 ~ 10), variance of noise (5 ~ 50).
3. Allow user input using sliders.
4. Finally, show the regression line in red color.

Response:

```
python

import numpy as np
import matplotlib.pyplot as plt
import streamlit as st
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score

def generate_data(a, b, c, mean, variance, num_points=100, x_range=(0, 10)):
    x = np.linspace(x_range[0], x_range[1], num_points)
    noise = np.random.normal(mean, np.sqrt(variance), num_points)
    y = a * x + b + c * noise
    return x, y

st.title("Linear Regression Demo")

n = st.slider("Number of Points (n)", 10, 500, 100)
a = st.slider("Slope (a)", -10.0, 10.0, 2.5)
b = st.slider("Intercept (b)", 1.0, 50.0, 5.0)
c = st.slider("Noise Multiplier (c)", 0.0, 100.0, 20.0)
mean = st.slider("Mean of Noise", 0.0, 10.0, 0.0)
variance = st.slider("Variance of Noise", 5.0, 50.0, 10.0)

x, y = generate_data(a, b, c, mean, variance, n)

X = x.reshape(-1, 1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = LinearRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

plt.scatter(X_test, y_test, color="blue", label="Test Data")
plt.plot(X_test, y_pred, color="red", label="Regression Line")
plt.title("Linear Regression")
plt.xlabel("x")
plt.ylabel("y")
plt.legend()
st.pyplot(plt)
```

Prompt 4:

User:

I'd like to add fundamental information to measure the regression result like R^2 , MSE, MAE, and the comparison of slope and intercept (predicted vs actual) into the plot. Keep the label in the lower right corner and place the metrics in the upper left corner.

Key Change:

```
python

# Added metrics to the plot
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)

metrics_text = f"R²: {r2:.4f}\nMSE: {mse:.4f}\nMAE: {mae:.4f}\nPredicted a = {model.coef_[0]:.4f}, Predicted b = {model.intercept_:.4f}"

plt.scatter(X_test, y_test, color="blue", label="Test Data")
plt.plot(X_test, y_pred, color="red", label="Regression Line")
plt.text(0.05, 0.95, metrics_text, transform=plt.gca().transAxes, fontsize=10,
         verticalalignment='top', bbox=dict(boxstyle="round", alpha=0.1))
plt.legend(loc="lower right")
st.pyplot(plt)
```

Prompt 5:

User:

I would like to:

1. Add some information for step 2 (data understanding), e.g., a histogram of x , y , or a randomly sampled scatter plot.
2. Adjust labels and metrics dynamically based on the slope value.
3. For $a > 0$: place the label in the lower right and metrics in the upper left. For $a < 0$: place the label in the upper right and metrics in the lower left.

Key Change:

```
python

# Added histogram and random sample
st.write("### Histogram of X and Y")
fig, (ax1, ax2) = plt.subplots(1, 2)
ax1.hist(x, bins=20, color="blue")
ax2.hist(y, bins=20, color="green")
st.pyplot(fig)

# Added random sample scatter plot
sampled_data = data.sample(10)
fig, ax = plt.subplots()
ax.scatter(sampled_data["x"], sampled_data["y"])
st.pyplot(fig)

# Dynamic label and metric positioning based on slope a
if a > 0:
    ax.text(0.05, 0.95, metrics_text, transform=ax.transAxes, fontsize=10,
           verticalalignment='top')
    ax.legend(loc="lower right")
else:
    ax.text(0.05, 0.05, metrics_text, transform=ax.transAxes, fontsize=10,
           verticalalignment='bottom')
    ax.legend(loc="upper right")
```

Prompt 6:

User:

I'd like to enhance the score to 9.5/10 for both criteria. Based on your feedback, I think the following suggestions would be beneficial:

1. Step-by-step tutorial.
2. Tooltip explanations.
3. Residual plot.

Key Change:

```
python

# Added step-by-step tutorial
st.sidebar.write("""
## Step-by-Step Tutorial:
1. Set Parameters.
2. Visualize Data.
3. Train the Model.
4. Evaluate the Model.
5. Residual Analysis.
""")

# Added tooltips for metrics explanation
st.sidebar.info("R²: Proportion of variance in y explained by x. MSE: Mean Squared Error.
MAE: Mean Absolute Error.")

# Added residual plot
residuals = y_test - y_test_pred
fig, ax = plt.subplots()
ax.scatter(X_test, residuals)
ax.axhline(0, color='red')
st.pyplot(fig)
```

Prompt 7:

User:

1. Shrink the height of the residual plot by 50% and add an explanation below it. 2. In Step 6, adjust labels and metrics dynamically to avoid overlap with data points based on slope. 3. Add a purple dashed line representing the actual slope and intercept.

Key Change:

```
python

# Shrinking residual plot and adding explanation
fig, ax = plt.subplots(figsize=(8, 2)) # Shrink height by 50%
ax.scatter(X_test, residuals)
ax.axhline(0, color='red')
st.pyplot(fig)
st.write("Residuals show the difference between actual and predicted values.")

# Dynamic label/metric positioning and actual line
if a > 0:
    ax.text(0.05, 0.95, metrics_text, transform=ax.transAxes, fontsize=10)
    ax.legend(loc="lower right")
else:
    ax.text(0.05, 0.05, metrics_text, transform=ax.transAxes, fontsize=10)
    ax.legend(loc="upper right")

# Purple dashed line for actual slope and intercept
actual_line = a * X_test + b
ax.plot(X_test, actual_line, color="purple", linestyle="--", label=f"Actual: a={a}, b={b}")
```

This interaction history shows how the project evolved, with each step aligned with the **CRISP-DM framework** and the user's evolving requirements. Each prompt naturally led to enhanced functionality using Python and Streamlit to build an interactive linear regression demonstration.