Skills Network

Lab: Generative AI for Models Development

Estimated time: 30 minutes

In this lab, we will use generative AI to create Python scripts to develop and evaluate different predictive models for a given data set.

Learning objectives

In this lab, you will learn how to use generative AI to create Python codes that can:

- Use linear regression in one variable to fit the parameters to a model
- Use linear regression in multiple variables to fit the parameters to a model
- Use polynomial regression in a single variable to fit the parameters to a model
- Create a pipeline for performing linear regression using multiple features in polynomial scaling
- Use the grid search with cross-validation and ridge regression to create a model with optimum hyperparameters

Code execution environment

To test the prompt-generated code, keep the Jupyter Notebook (in the link below) open in a separate tab in your web browser. The notebook has some setup instructions that you should complete now. Jupyter-Lite Test Environment

The data set for this lab is available in the following URL.

URL = https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-DA0101EN-Coursera/laptop_pricing_dataset_mod2.csv

```
URL = "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-DA0101EN-Coursera/laptop_pricing_dataset_mod2.csv"
```

The dataset is a filtered and modified version of the Laptop Price Prediction using specifications dataset, available under the Database Contents License (DbCL) v1.0 on the Kaggle website.

Complete the setup in the Jupyter Notebook and then proceed further.

Building the prompt: Importing data set

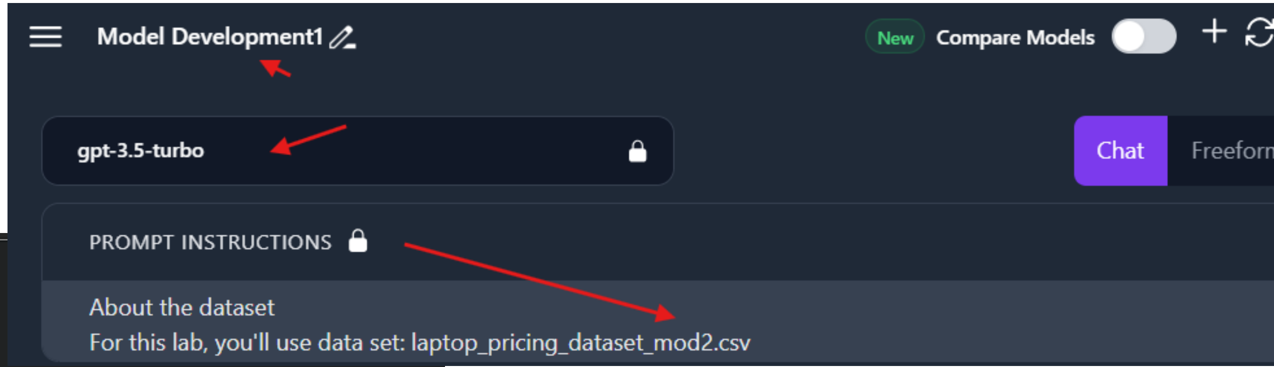
Now you will start building your prompt. The first step is to ask the Gen AI model to generate a code to import the provided dataset to a Pandas' data frame. You must specify if you imported the data. Then, you should see the dataset headers in the first row of the CSV file.

You can structure the prompt to create the code as follows:

```
Write a Python code that can perform the following tasks.
Read the CSV file, located on a given file path, into a pandas data frame, assuming that the first row of the file can be used as the headers for the data.
```


```
Write a Python code that can perform the following tasks.
Read the CSV file, located on a given file path, into a pandas data frame, assuming that the first row of the file can be used as the headers for the data.
```


Running this prompt using the GPT 3.5 engine in the GenAI classroom should result in a response that contains the code, as shown below.



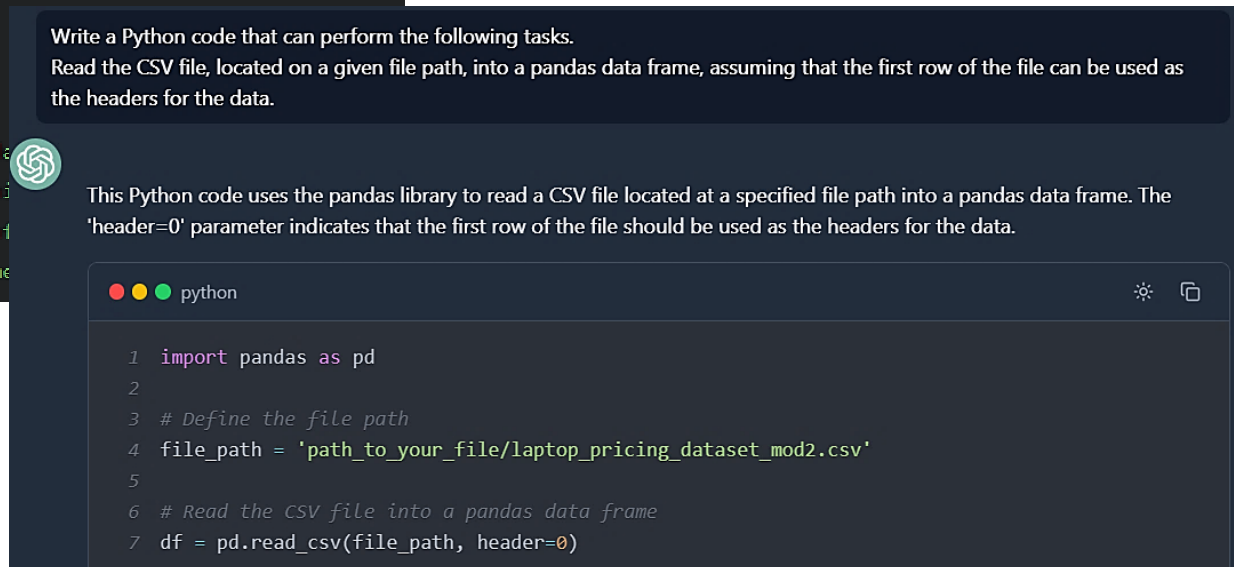

```python
import pandas as pd

# Define the file path
file_path = "path/to/your/csv/file.csv"

# Read the CSV file into a pandas data frame
data_frame = pd.read_csv(file_path)

# Assume the first row of the file can be used as the headers for the data
# If the file doesn't have headers, you can remove the 'header' parameter
data_frame = pd.read_csv(file_path, header=0)

# Additional details:
# - The 'pd.read_csv()' function is used to read a CSV file into a panda
# - The 'header' parameter in the 'pd.read_csv()' function specifies whi
#   By default, it is set to 'infer', which means pandas will try to inf
#   If the headers are in the first row, you can set the 'header' parame
```



You may copy the code from the response and try it on the Jupyter Notebook with appropriate modifications in the file path.

Linear regression in one variable

You can now ask the generative AI model to generate a script to create a linear regression model using a single attribute as the training feature to predict a target attribute.

You can write the prompt as follows:

```
Write a Python code that performs the following tasks.
1. Develops and trains a linear regression model that uses one attribute of a data frame as the source variable and another as a target variable.
2. Calculate and display the MSE and R^2 values for the trained model
```

It will generate a response as shown below:

```python
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Assume you have a pandas data frame called 'data_frame' with two columns: 'source_variable' and 'target_variable'

# Extract the source variable and target variable from the data frame
X = data_frame[['source_variable']]
y = data_frame['target_variable']

# Initialize a linear regression model
model = LinearRegression()

# Train the model using the source and target variables
model.fit(X, y)

# Make predictions using the trained model
y_pred = model.predict(X)

# Calculate the mean squared error (MSE)
mse = mean_squared_error(y, y_pred)

# Calculate the coefficient of determination (R^2)
r2 = r2_score(y, y_pred)

# Display the MSE and R^2 values
print("Mean Squared Error (MSE):", mse)
print("Coefficient of Determination (R^2):", r2)

# Additional details:
# - The 'LinearRegression' class from the 'sklearn.linear_model' module is used to create a linear regression model.
# - The 'fit()' method is used to train the model using the source and target variables.
# - The 'predict()' method is used to make predictions using the trained model.
# - The 'mean_squared_error()' function from the 'sklearn.metrics' module is used to calculate the MSE.
# - The 'r2_score()' function from the 'sklearn.metrics' module is used to calculate the R^2 value.
```

python

```python
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Assuming 'source_attribute' and 'target_attribute' are
X = df[['source_attribute']]
y = df['target_attribute']

# Initialize the linear regression model
model = LinearRegression()

# Train the model
model.fit(X, y)

# Make predictions
y_pred = model.predict(X)

# Calculate Mean Squared Error
mse = mean_squared_error(y, y_pred)

# Calculate R^2
r2 = r2_score(y, y_pred)

print('Mean Squared Error:', mse)
print('R^2:', r2)
```

You can use this code to develop a linear regression model with the **target variable** as `Price` and the **source variable** as **CPU_frequency**. Try this out in the **Test environment**.

Linear regression in multiple variables

You can modify the prompt to ask for a code that uses multiple attributes as input for training a model to predict the value of a target variable.

For that, you can use the **prompt** as shown below:

```
Write a Python code that performs the following tasks.
1. Develops and trains a linear regression model that uses some attributes of a data frame as the source variables and one of the attributes as a target variable.
2. Calculate and display the MSE and R^2 values for the trained model.
```

You should see a **similar response** to the following:

```python
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Assume you have a pandas data frame called 'data_frame' with multiple columns: 'source_variable_1', 'source_variable_2', ..., 'target_variable'

# Extract the source variables and target variable from the data frame
X = data_frame[['source_variable_1', 'source_variable_2', ...]]
y = data_frame['target_variable']

# Initialize a linear regression model
model = LinearRegression()

# Train the model using the source and target variables
model.fit(X, y)

# Make predictions using the trained model
y_pred = model.predict(X)

# Calculate the mean squared error (MSE)
mse = mean_squared_error(y, y_pred)

# Calculate the coefficient of determination (R^2)
r2 = r2_score(y, y_pred)

# Display the MSE and R^2 values
print("Mean Squared Error (MSE):", mse)
print("Coefficient of Determination (R^2):", r2)

# Additional details:
# - The 'LinearRegression' class from the 'sklearn.linear_model' module is used to create a
# - The 'fit()' method is used to train the model using the source and target variables.
# - The 'predict()' method is used to make predictions using the trained model.
# - The 'mean_squared_error()' function from the 'sklearn.metrics' module is used to calculate the MSE.
# - The 'r2_score()' function from the 'sklearn.metrics' module is used to calculate the R^2 value.
```

```python
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Assuming 'source_columns' is a list of columns used as source variables and 'target_column' is the target variable
X = df[['source_column1', 'source_column2']]
y = df['target_column']

# Create and train the linear regression model
model = LinearRegression()
model.fit(X, y)

# Make predictions
y_pred = model.predict(X)

# Calculate Mean Squared Error and R^2
mse = mean_squared_error(y, y_pred)
r2 = r2_score(y, y_pred)

print('Mean Squared Error:', mse)
print('R^2:', r2)
```

You can use the generated code and build a **linear regression model** for the data set that uses `CPU_frequency`, `RAM_GB`, `Storage_GB_SSD`, `CPU_core`, `OS`, `GPU` and `Category` and `source variables` and `Price` as the `target variable`. You can check the code generated in the webpage browser.

You may compare the performance of the **two models** by comparing their **MSE** and **R^2 values**.

Polynomial regression

You can explore creating a model that uses higher-order features from original attributes. Higher orders of the same feature allow better generalization on the target variable. It is called **polynomial** regression, and you can use Generative AI to create a code for this.

Assume you are given a single attribute as the source variable and one as a target variable. You must create a model using polynomial regression for a given order. You can also make the model for different order values and compare their performance based on **MSE** and **R^2 scores**.

You can write a similar prompt to the following code:

```
Write a Python code that performs the following tasks.
1. Develops and trains multiple polynomial regression models, with orders 2, 3, and 5, that use one attribute of a data frame as the source variable and another as a target variable.
2. Calculate and display the MSE and R^2 values for the trained models.
3. Compare the performance of the models.
```

The **expected output** should look as shown below:

Write a Python code that performs the following tasks.
1. Develops and trains multiple polynomial regression models, with orders 2, 3, and 5, that use one attribute of a data frame as the source variable and another as a target variable.
2. Calculate and display the MSE and R^2 values for the trained models.
3. Compare the performance of the models.

```python
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error, r2_score

# Assuming 'source_column' and 'target_column' are the columns used as source and target variables
def polynomial_regression(df, source_column, target_column, degrees=[2, 3, 5]):
    X = df[[source_column]]
    y = df[target_column]
    results = {}
    for degree in degrees:
        poly_features = PolynomialFeatures(degree=degree)
        X_poly = poly_features.fit_transform(X)
        model = LinearRegression()
        model.fit(X_poly, y)
        y_pred = model.predict(X_poly)
        mse = mean_squared_error(y, y_pred)
        r2 = r2_score(y, y_pred)
        results[f'Degree {degree}'] = {'Mean Squared Error': mse, 'R^2': r2}
    return results

results = polynomial_regression(df, 'source_column', 'target_column')
for degree, values in results.items():
    print(f'Degree {degree}:')
    print('Mean Squared Error:', values['Mean Squared Error'])
    print('R^2:', values['R^2'])
```

```python
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error, r2_score

# Assume you have a pandas data frame called 'data_frame' with two columns: 'source_variable' and 'target_variable'

# Extract the source variable and target variable from the data frame
X = data_frame[['source_variable']]
y = data_frame['target_variable']

# Initialize lists to store the MSE and R^2 values for each model
mse_values = []
r2_values = []

# Loop through the polynomial orders
for order in [2, 3, 5]:
    # Create polynomial features
    polynomial_features = PolynomialFeatures(degree=order)
    X_poly = polynomial_features.fit_transform(X)

    # Initialize a linear regression model
    model = LinearRegression()

    # Train the model using the polynomial features and target variable
    model.fit(X_poly, y)

    # Make predictions using the trained model
    y_pred = model.predict(X_poly)

    # Calculate the mean squared error (MSE)
    mse = mean_squared_error(y, y_pred)

    # Calculate the coefficient of determination (R^2)
    r2 = r2_score(y, y_pred)

    # Append the MSE and R^2 values to the lists
    mse_values.append(mse)
    r2_values.append(r2)

    # Display the MSE and R^2 values for the current model
    print(f"Polynomial Order {order}:")
    print("Mean Squared Error (MSE):", mse)
    print("Coefficient of Determination (R^2):", r2)
    print()

# Compare the performance of the models
best_order = np.argmin(mse_values)
worst_order = np.argmax(r2_values)

print("Model Comparison:")
print(f"Best Polynomial Order: {best_order + 2}")
print(f"Worst Polynomial Order: {worst_order + 2}")

# Additional details:
# - The 'PolynomialFeatures' class from the 'sklearn.preprocessing' module is used to create polynomial features.
# - The 'fit_transform()' method is used to transform the source variable into polynomial features.
# - The 'degree' parameter in the 'PolynomialFeatures' class specifies the maximum degree of the polynomial features.
# - The 'argmin()' and 'argmax()' functions from the 'numpy' module are used to find the index of the minimum and maximum values in a list, respectively.
```

You can use the **relevant** part of the code in your script.

You can see that the model can generate sophisticated code using functions to create and train models with different orders and evaluate their performance for each of them.

Try to run the generated code on the **testing interface** with the **source variable** as **CPU frequency** and the **target variable** as **Price**.

Creating a Pipeline

Pipelines are processes containing a sequence of steps that lead to creating a trained model.

You will now use the Generative AI model to create a pipeline for performing feature scaling, creating polynomial features for multiple attributes, and performing linear regression using these variables.

You can build a **similar prompt** to the following code:

```
Write a Python code that performs the following tasks.
1. Create a pipeline that performs parameter scaling, Polynomial Feature generation, and Linear regression. Use the set of multiple features as before to create this pipeline.
2. Calculate and display the MSE and R^2 values for the trained model.
```

The **expected response** is as shown below:

```python
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error, r2_score

# Assuming 'source_columns' is a list of columns used as source variables and 'target_column' is the target variable
X = df[['source_column1', 'source_column2']]
y = df['target_column']

# Create a pipeline
pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('poly_features', PolynomialFeatures(degree=3)),
    ('linear_reg', LinearRegression())
])

# Fit the pipeline
pipeline.fit(X, y)

# Make predictions
y_pred = pipeline.predict(X)

# Calculate Mean Squared Error and R^2
mse = mean_squared_error(y, y_pred)
r2 = r2_score(y, y_pred)

print('Mean Squared Error:', mse)
print('R^2:', r2)
```

You can use the **relevant** part in your code.

Make appropriate changes to the attributes you will use in the code. Consider the same set of attributes as used for **multiple feature linear regression**, which is **CPU_frequency**, **RAM_GB**, **Storage_GB_SSD**, **CPU_core**, **OS**, **GPU** and **Category** as **source variables**, and **Price** as the **target variable**.

```python
import pandas as pd
import numpy as np
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Assume you have a pandas data frame called 'data_frame' with multiple columns: 'source_variable_1', 'source_variable_2', ..., 'target_variable'

# Extract the source variables and target variable from the data frame
X = data_frame[['source_variable_1', 'source_variable_2', ...]]
y = data_frame['target_variable']

# Create a pipeline that performs parameter scaling, polynomial feature generation, and linear regression
pipeline = make_pipeline(
    StandardScaler(),
    PolynomialFeatures(degree=2),
    LinearRegression()
)

# Train the model using the source and target variables
pipeline.fit(X, y)

# Make predictions using the trained model
y_pred = pipeline.predict(X)

# Calculate the mean squared error (MSE)
mse = mean_squared_error(y, y_pred)

# Calculate the coefficient of determination (R^2)
r2 = r2_score(y, y_pred)

# Display the MSE and R^2 values
print("Mean Squared Error (MSE):", mse)
print("Coefficient of Determination (R^2):", r2)

# Additional details:
# - The 'make_pipeline()' function from the 'sklearn.pipeline' module is used to create a pipeline.
# - The 'StandardScaler' class from the 'sklearn.preprocessing' module is used to perform parameter scaling.
# - The 'PolynomialFeatures' class from the 'sklearn.preprocessing' module is used to create polynomial features.
# - The 'LinearRegression' class from the 'sklearn.linear_model' module is used for linear regression.
# - The pipeline automatically applies the transformations in the specified order.
```

Grid search and Ridge regression

An improved way to train your model is to use ridge regression instead of linear regression. You can use the polynomial features of multiple attributes. One of the key factors of ridge regression is using the parameter `alpha` as a hyperparameter for training. Using grid search, one can determine the optimum value of the hyperparameter for the given set of features. Grid search also uses cross-validation training to train and prepare the optimum model.

You can use generative AI to create the Python code to perform a grid search for the optimum ridge regression model, which uses polynomial features generated from multiple parameters.

You may use the following prompt to create this:

```
Write a Python code that performs the following tasks.
1. Use polynomial features for some of the attributes of a data frame.
2. Perform Grid search on a ridge regression model for a set of values of hyperparameter alpha and polynomial features as input.
3. Use cross-validation in the Grid search.
4. Evaluate the resulting model's MSE and R^2 values.
```

The **resulting code** will look as shown below:

```python
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error, r2_score

# Assuming 'source_columns' is a list of columns used as source variables and 'target_column' is the target variable
X = df[['source_column1', 'source_column2']]
y = df['target_column']

# Create polynomial features
poly_features = PolynomialFeatures(degree=3)
X_poly = poly_features.fit_transform(X)

# Define the Ridge regression model
ridge = Ridge()

# Set up the hyperparameters to search
param_grid = {
    'alpha': [0.1, 1, 10]
}

# Perform Grid Search with cross-validation
grid_search = GridSearchCV(ridge, param_grid, cv=5)
grid_search.fit(X_poly, y)

# Get the best model
best_model = grid_search.best_estimator_

# Make predictions
y_pred = best_model.predict(X_poly)

# Calculate Mean Squared Error and R^2
mse = mean_squared_error(y, y_pred)
r2 = r2_score(y, y_pred)

print('Mean Squared Error:', mse)
print('R^2:', r2)
```

You can test this code for the data set on the testing environment.

You make use of the following parametric values for this purpose.

Source Variables: `CPU_frequency`, `RAM_GB`, `Storage_GB_SSD`, `CPU_core`, `OS`, `GPU` and `Category`
Target Variable: `Price`
Set of values for **alpha**: **0.0001,0.001,0.01, 0.1, 1, 10**
Cross Validation: **4-fold**
Polynomial Feature order: 2

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import GridSearchCV, cross_val_score
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error, r2_score

# Assume you have a pandas data frame called 'data_frame' with multiple columns: 'source_variable_1', 'source_variable_2', ..., 'target_variable'

# Extract the source variables and target variable from the data frame
X = data_frame[['source_variable_1', 'source_variable_2', ...]]
y = data_frame['target_variable']

# Create polynomial features
polynomial_features = PolynomialFeatures()

# Transform the source variables into polynomial features
X_poly = polynomial_features.fit_transform(X)

# Define the hyperparameter values for the grid search
param_grid = {'alpha': [0.1, 1.0, 10.0], 'degree': [2, 3, 4]}

# Initialize a ridge regression model
model = Ridge()

# Perform grid search with cross-validation
grid_search = GridSearchCV(model, param_grid, cv=5)

# Train the model using the polynomial features and target variable
grid_search.fit(X_poly, y)

# Make predictions using the trained model
y_pred = grid_search.predict(X_poly)

# Calculate the mean squared error (MSE)
mse = mean_squared_error(y, y_pred)

# Calculate the coefficient of determination (R^2)
r2 = r2_score(y, y_pred)

# Display the MSE and R^2 values
print("Mean Squared Error (MSE):", mse)
print("Coefficient of Determination (R^2):", r2)

# Additional details:
# - The 'PolynomialFeatures' class from the 'sklearn.preprocessing' module is used to create polynomial features.
# - The 'GridSearchCV' class from the 'sklearn.model_selection' module is used to perform grid search with cross-validation.
# - The 'Ridge' class from the 'sklearn.linear_model' module is used for ridge regression.
# - The 'fit_transform()' method is used to transform the source variables into polynomial features.
# - The 'param_grid' parameter in the 'GridSearchCV' class specifies the hyperparameter values to search over.
# - The 'cv' parameter in the 'GridSearchCV' class specifies the number of folds for cross-validation.
# - The best model found by grid search can be accessed using the 'best_estimator_' attribute of the grid search object.
```

Conclusion

Congratulations! You have completed the lab on Data preparation.

With this, you have learned how to use generative AI to create Python codes that can:

- Implement linear regression in single variable
- Implement linear regression in multiple variables
- Implement polynomial regression for different orders of a single variable
- Create a pipeline that implements polynomial scaling for multiple variables and performs linear regression on them
- Apply a grid search to create an optimum ridge regression model for multiple features

Author(s)

Abhishek Gagneja