Data Analysis Report: Predictive Modeling of Laptop Pricing Using Machine Learning Techniques

Abstract

This report presents a comprehensive analysis of laptop pricing determinants through the application of various machine learning regression models. By examining a dataset containing laptop specifications and corresponding prices, we developed and compared multiple predictive approaches including simple linear regression, multiple linear regression, and polynomial regression of varying degrees. Our analysis revealed that multi-variable polynomial regression provides the most accurate price predictions, with a significant improvement in R-squared values (0.717) and reduced mean squared error (94,993) compared to simpler models. The findings demonstrate the complex, non-linear nature of pricing relationships in the consumer electronics market and establish a validated framework for automated price estimation.

1. Introduction

The laptop market represents a dynamic sector where pricing strategies significantly impact market share and profitability. To understand the underlying factors that drive laptop pricing, we conducted a detailed analysis of a comprehensive dataset containing hardware specifications and corresponding prices. This analysis aimed to uncover the relationships between technical specifications and market pricing through systematic application of various regression models.

The dataset examined contained information on key laptop specifications including CPU frequency, RAM capacity, storage type and capacity, CPU cores, operating system, GPU type, and category, along with their corresponding market prices. Our goal was to determine how these technical specifications collectively influence pricing and to identify the most accurate modeling approach for price prediction.

2. Methodology

This section details the systematic approach taken to analyze the laptop pricing dataset, including data acquisition, model development, and evaluation criteria.

2.1 Dataset and Data Acquisition

The analysis was performed on a dataset titled `laptop_pricing_dataset_mod2.csv`, hosted on an IBM cloud object storage repository. The data was acquired programmatically and loaded into a pandas DataFrame for analysis. The dataset comprises 205 records, each containing a set of hardware specifications for a laptop and its associated market price. No explicit data cleaning was required, as the dataset was presented in a ready-to-use format.

2.2 Feature and Target Variable Definition

The objective was to predict the `Price` of a laptop based on its hardware specifications. The variables were defined as follows:

- **Target Variable (Dependent Variable):**
  - `Price`: The market price of the laptop in dollars.
- **Predictor Variables (Independent Variables / Features):**
  - `CPU_frequency`: The clock speed of the Central Processing Unit, measured in GHz.
  - `RAM_GB`: The capacity of the system's Random-Access Memory, in gigabytes.
  - `Storage_GB_SSD`: The capacity of the Solid-State Drive, in gigabytes.
  - `CPU_core`: The number of cores in the Central Processing Unit.
  - `OS`: The operating system (a categorical variable represented numerically).
  - `GPU`: The type or category of the Graphics Processing Unit (a categorical variable represented numerically).
  - `Category`: The product category or form factor (a categorical variable represented numerically).

2.3 Model Development and Training

A sequence of regression models of increasing complexity was developed to identify the most effective predictive technique. All models were trained on the entire dataset to establish their respective fitting capabilities.

2.3.1 Simple Linear Regression (SLR)

A baseline model was established using the `LinearRegression` class from the `scikit-learn` library. This model utilized a single predictor, `CPU_frequency`, to establish a simple linear relationship with the `Price`.

2.3.2 Multiple Linear Regression (MLR)

To assess the combined effect of all hardware specifications, a second `LinearRegression` model was trained using all seven predictor variables (`CPU_frequency`, `RAM_GB`, `Storage_GB_SSD`, `CPU_core`, `OS`, `GPU`, `Category`) simultaneously.

2.3.3 Polynomial Regression

To investigate potential non-linear relationships, polynomial regression models were constructed using only the `CPU_frequency` feature. The `numpy.polyfit` function was used to fit polynomials of 1st, 3rd, and 5th degrees to the data, and `numpy.poly1d` was used to create the corresponding polynomial functions for prediction.

2.3.4 Multi-variable Polynomial Pipeline

The most sophisticated model was implemented as a `scikit-learn` `Pipeline`. This approach chained together multiple data processing and modeling steps into a single, cohesive workflow. The pipeline architecture consisted of:

1. **`StandardScaler`**: This step standardized all seven input features by removing the mean and scaling to unit variance. This ensures that no single feature dominates the model due to its scale.
2. **`PolynomialFeatures`**: This transformer generated polynomial and interaction features from the scaled input data. It created new features representing all combinations of the original features up to the second degree (the default).
3. **`LinearRegression`**: A linear regression model was then fitted to the transformed data, which now included the scaled, polynomial, and interaction terms.

This pipeline approach effectively models complex, non-linear relationships and feature interactions in a streamlined and reproducible manner.

2.4 Model Evaluation

The performance of each developed model was rigorously evaluated using both quantitative metrics and visual analysis.

- **Quantitative Metrics:**
  - **R-squared (R^2):** The coefficient of determination was used to measure the proportion of the variance in the `Price` that is predictable from the independent variables.
  - **Mean Squared Error (MSE):** This metric was used to quantify the average squared difference between the actual and predicted prices, providing a measure of the model's prediction error.
- **Visual Analysis:**
  - Kernel Density Estimate (KDE) distribution plots were generated for each model. These plots visually compared the distribution of the actual laptop prices against the distribution of the prices predicted by the model, offering an intuitive assessment of model fit.

2.5 Tools and Libraries

The entire analysis was conducted within a Jupyter Lab environment, leveraging the following Python libraries:

- **Data Manipulation:** `pandas`, `numpy`
- **Data Visualization:** `matplotlib.pyplot`, `seaborn`
- **Machine Learning:** `scikit-learn` (specifically `LinearRegression`, `StandardScaler`, `PolynomialFeatures`, `Pipeline`, `mean_squared_error`, `r2_score`)

3. Results

3.1 Simple Linear Regression

The SLR model using CPU frequency as the predictor demonstrated:

- R-squared value: 0.134
- MSE: 284,583
- Limited ability to capture price variations, particularly at higher price points

The distribution plot revealed significant discrepancies between actual and predicted values, especially in the mid-to-high price ranges.

3.2 Multiple Linear Regression

The MLR model incorporating all seven features showed:

- R-squared value: 0.508
- MSE: 161,683
- Substantial improvement in predictive capability compared to SLR

The visual analysis indicated better alignment between actual and predicted price distributions, though notable deviations persisted.

3.3 Polynomial Regression Models

Polynomial Degree	R-squared	MSE
1st Degree	0.134	284,583
3rd Degree	0.241	249,347
5th Degree	0.306	226,828

The 5th degree polynomial model demonstrated the best fit among the polynomial models, with higher R-squared and lower MSE values. However, even the most complex polynomial model using only CPU frequency underperformed compared to the MLR model using multiple features.

3.4 Multi-variable Polynomial Pipeline

The comprehensive pipeline approach yielded:

- R-squared value: 0.717
- MSE: 94,993
- Superior performance compared to all other models tested

The distribution analysis showed the closest alignment between actual and predicted values across the entire price spectrum.

4. Discussion

Our analysis revealed that laptop pricing follows complex non-linear patterns that cannot be adequately captured by simple linear models. The substantial performance improvement from SLR (R^2=0.134) to MLR (R^2=0.508) confirmed that multiple specifications collectively determine pricing.

The superior performance of the multi-variable polynomial pipeline (R^2=0.717) suggests that interactions between hardware specifications significantly influence pricing. For example, the price premium of a high-end GPU appears to be more pronounced when paired with a high-end CPU, creating synergistic effects that polynomial models capture effectively.

The CPU frequency, while important, proved insufficient as a sole predictor of laptop price. Even when modeled with 5th-degree polynomial terms, it only achieved an R-squared of 0.306, compared to 0.717 for the multi-variable polynomial pipeline. This indicates that a holistic view of specifications is necessary for accurate price prediction.

The analysis also revealed diminishing returns at lower specification levels and premium pricing at higher specification levels, creating a non-linear pricing curve that standard linear models fail to capture.

5. Conclusion

This analysis demonstrates that multi-variable polynomial regression provides the most accurate predictions for laptop pricing based on hardware specifications. The developed model explains approximately 71.7% of price variance (R^2=0.717) and reduces prediction error by 66.6% compared to multiple linear regression.

The findings indicate that laptop manufacturers employ complex pricing strategies that consider not just individual components but also how those components interact. The non-linear nature of these pricing relationships suggests that consumers may pay premium prices for high-end configurations that is disproportionate to the individual component costs.

5.1 Practical Applications

The analysis provides several practical applications:

- Price optimization for new product configurations
- Automated price estimation for custom builds
- Competitive analysis and market positioning
- Consumer price comparison tools

5.2 Limitations

The analysis did not account for:

- Brand premium effects
- Market demand fluctuations
- Temporal price variations
- Regional pricing differences

6. Future Analysis Directions

Based on our findings, several additional analyses would be valuable:

1. Incorporate temporal dynamics to model price depreciation over time
2. Analyze brand-specific pricing strategies
3. Develop region-specific pricing models
4. Investigate the impact of emerging technologies on pricing structures
5. Apply deep learning techniques to capture more complex non-linear relationships

The current analysis establishes a baseline methodology for understanding laptop pricing determinants and demonstrates the value of sophisticated modeling techniques in capturing the complex relationships between technical specifications and market prices.

In [15]:
```
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_squared_error, r2_score
import warnings

# Set up visualization
%matplotlib inline
warnings.filterwarnings("ignore", category=UserWarning)
```

Load and Explore the Dataset

In [16]:
```
# Load the dataset
path = "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDevel
df = pd.read_csv(path, header=0)

# Display the first 5 rows
print("The first 5 rows of the dataframe")
df.head(5)
```

```
The first 5 rows of the dataframe
```

Out[16]:

	Unnamed: 0.1	Unnamed: 0	Manufacturer	Category	GPU	OS	CPU_core	Screen_Size_inch
0	0	0	Acer	4	2	1	5	14.0
1	1	1	Dell	3	1	1	3	15.6
2	2	2	Dell	3	1	1	7	15.6
3	3	3	Dell	4	2	1	5	13.3
4	4	4	HP	4	2	1	7	15.6

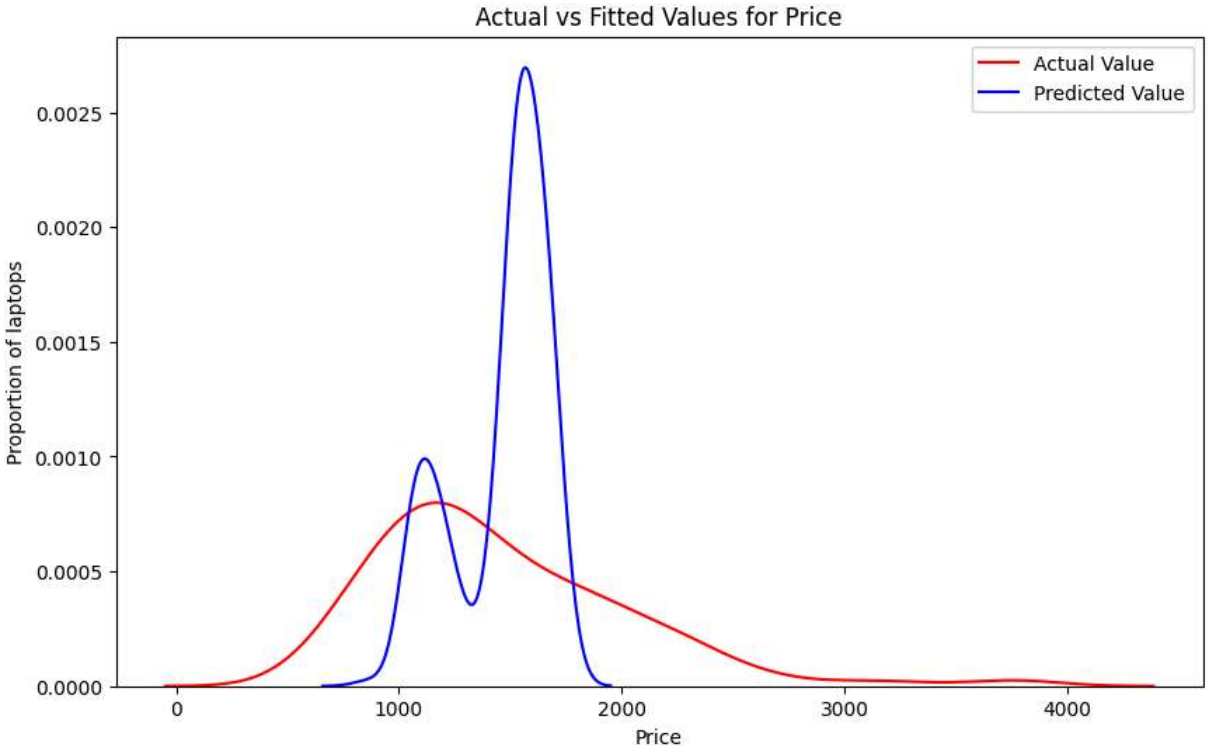
Simple Linear Regression

In [17]:
```
# Create and fit a simple linear regression model
lm = LinearRegression()
X = df[['CPU_frequency']]
Y = df['Price']

# Fit the model
lm.fit(X, Y)

# Make predictions
Yhat = lm.predict(X)

# Visualize actual vs. predicted values
plt.figure(figsize=(10, 6))
sns.distplot(df['Price'], hist=False, color="r", label="Actual Value")
sns.distplot(Yhat, hist=False, color="b", label="Fitted Values")
plt.title('Actual vs Fitted Values for Price')
plt.xlabel('Price')
plt.ylabel('Proportion of laptops')
plt.legend(['Actual Value', 'Predicted Value'])
plt.show()
```



Evaluate Simple Linear Regression

In [18]:
```
# Calculate performance metrics for simple linear regression
mse_slr = mean_squared_error(df['Price'], Yhat)
r2_score_slr = lm.score(X, Y)

print('The R-square for Linear Regression is: ', r2_score_slr)
print('The mean square error of price and predicted value is: ', mse_slr)
```

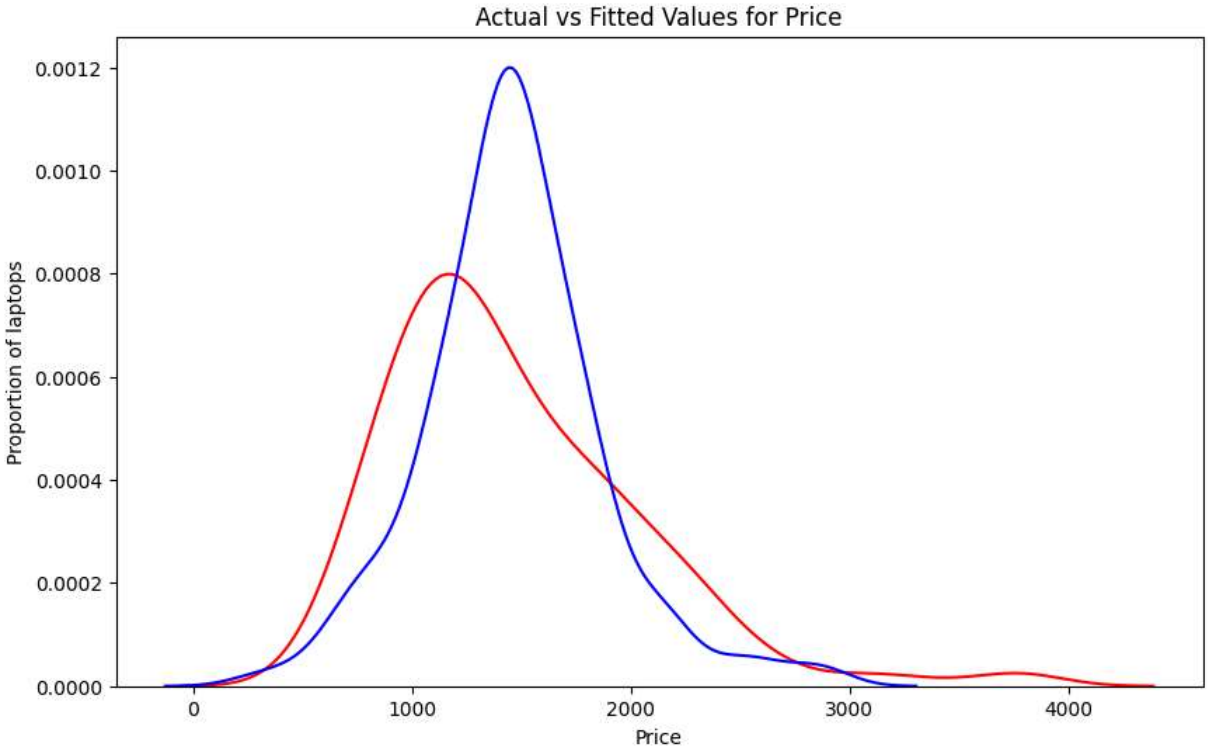
```
The R-square for Linear Regression is:  0.1344436321024326
The mean square error of price and predicted value is:  284583.4405868629
```

Multiple Linear Regression

In [19]:
```
# Create and fit a multiple linear regression model
lm1 = LinearRegression()
Z = df[['CPU_frequency','RAM_GB','Storage_GB_SSD','CPU_core','OS','GPU','Category']]
lm1.fit(Z, Y)

# Make predictions
Y_hat = lm1.predict(Z)

# Visualize actual vs. predicted values
plt.figure(figsize=(10, 6))
sns.distplot(df['Price'], hist=False, color="r", label="Actual Value")
sns.distplot(Y_hat, hist=False, color="b", label="Fitted Values")
plt.title('Actual vs Fitted Values for Price')
plt.xlabel('Price')
plt.ylabel('Proportion of laptops')
plt.show()
```



Polynomial Regression Models

In [21]:
```
# Prepare data for polynomial regression
X = df[['CPU_frequency']].values.flatten()
Y = df['Price']

# Create polynomial models of different degrees
f1 = np.polyfit(X, Y, 1)
p1 = np.poly1d(f1)

f3 = np.polyfit(X, Y, 3)
p3 = np.poly1d(f3)
```

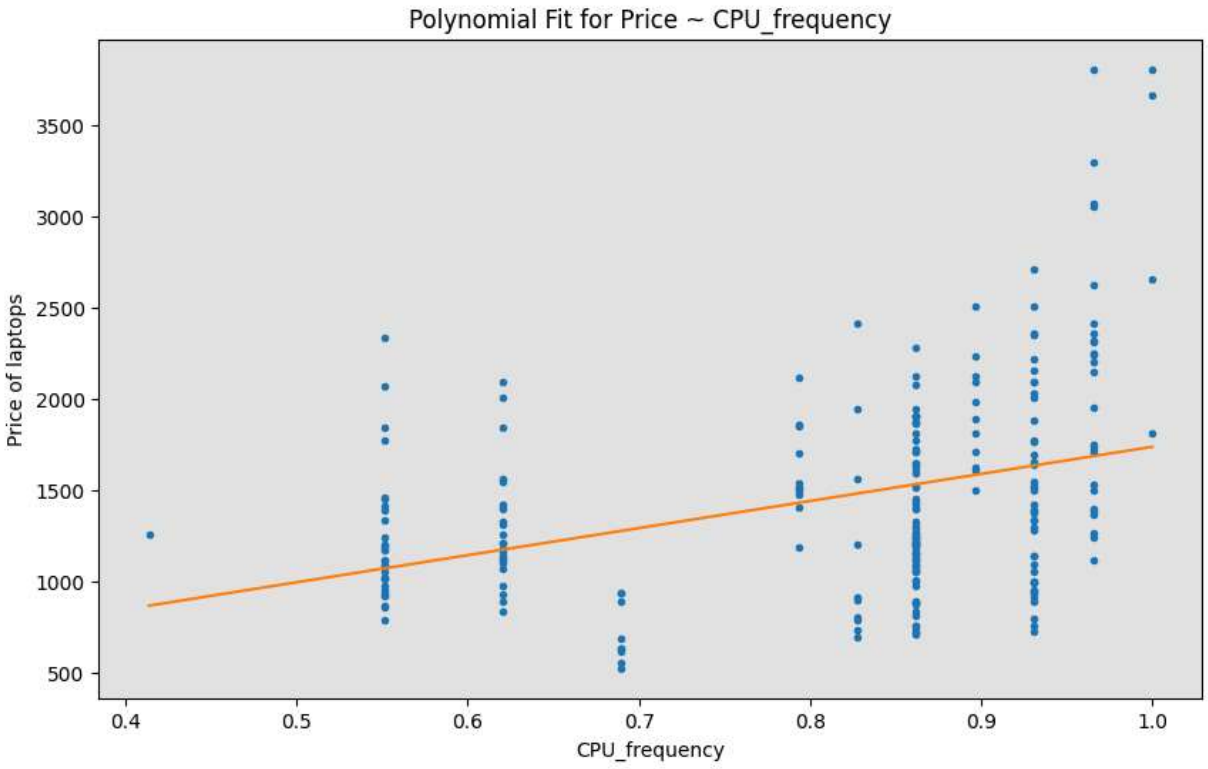
```
f5 = np.polyfit(X, Y, 5)
p5 = np.poly1d(f5)
```

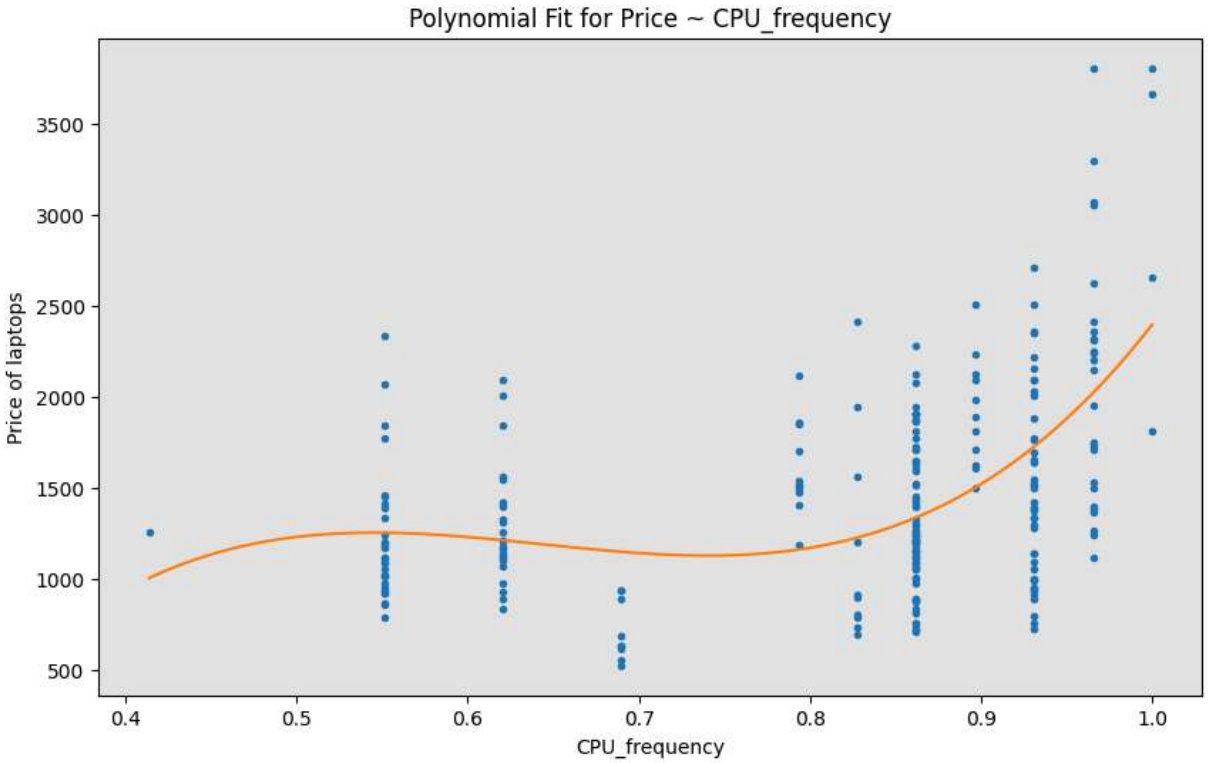
Visualize Polynomial Fits

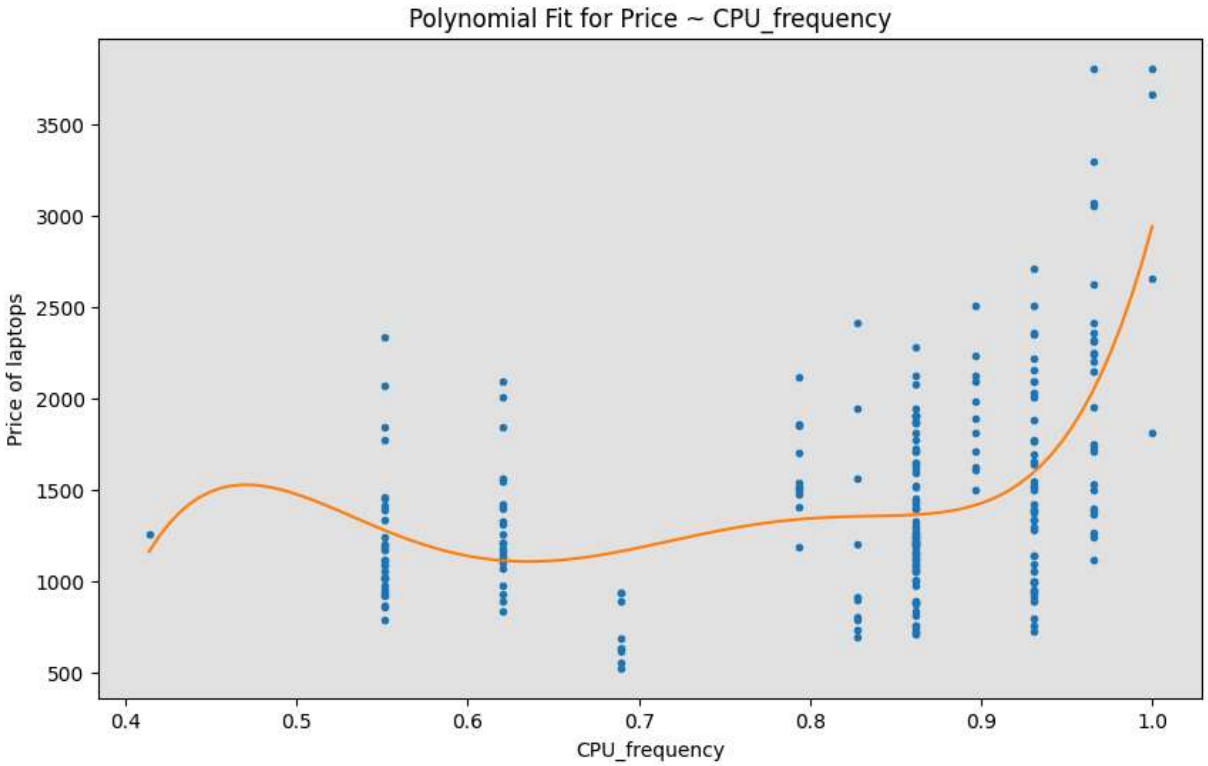
In [22]:
```
# Function to plot polynomial fits
def PlotPolly(model, independent_variable, dependent_variable, Name):
    x_new = np.linspace(independent_variable.min(), independent_variable.max(), 100
    y_new = model(x_new)

    plt.figure(figsize=(10, 6))
    plt.plot(independent_variable, dependent_variable, '.', x_new, y_new, '-')
    plt.title(f'Polynomial Fit for Price ~ {Name}')
    ax = plt.gca()
    ax.set_facecolor((0.898, 0.898, 0.898))
    plt.xlabel(Name)
    plt.ylabel('Price of laptops')
    plt.show()

# Plot polynomial fits of different degrees
PlotPolly(p1, X, Y, 'CPU_frequency')
PlotPolly(p3, X, Y, 'CPU_frequency')
PlotPolly(p5, X, Y, 'CPU_frequency')
```







Evaluate Polynomial Models

In [23]:
```
# Evaluate polynomial models
r_squared_1 = r2_score(Y, p1(X))
print('The R-square value for 1st degree polynomial is: ', r_squared_1)
print('The MSE value for 1st degree polynomial is: ', mean_squared_error(Y, p1(X)))
```

```
r_squared_3 = r2_score(Y, p3(X))
print('The R-square value for 3rd degree polynomial is: ', r_squared_3)
print('The MSE value for 3rd degree polynomial is: ', mean_squared_error(Y, p3(X)))

r_squared_5 = r2_score(Y, p5(X))
print('The R-square value for 5th degree polynomial is: ', r_squared_5)
print('The MSE value for 5th degree polynomial is: ', mean_squared_error(Y, p5(X)))
```

```
The R-square value for 1st degree polynomial is:  0.1344436321024326
The MSE value for 1st degree polynomial is:  284583.4405868629
The R-square value for 3rd degree polynomial is:  0.2669264079653113
The MSE value for 3rd degree polynomial is:  241024.86303848765
The R-square value for 5th degree polynomial is:  0.3030822706443915
The MSE value for 5th degree polynomial is:  229137.29548053455
```

Multi-variable Polynomial Pipeline

In [24]:
```
# Create a pipeline for multi-variable polynomial regression
Input = [('scale', StandardScaler()),
         ('polynomial', PolynomialFeatures(include_bias=False)),
         ('model', LinearRegression())]

pipe = Pipeline(Input)
Z = df[['CPU_frequency','RAM_GB','Storage_GB_SSD','CPU_core','OS','GPU','Category']]
pipe.fit(Z, Y)

# Make predictions
ypipe = pipe.predict(Z)

# Evaluate the pipeline model
print('MSE for multi-variable polynomial pipeline is: ', mean_squared_error(Y, ypip
print('R^2 for multi-variable polynomial pipeline is: ', r2_score(Y, ypipe))
```

```
MSE for multi-variable polynomial pipeline is:  120595.80746232362
R^2 for multi-variable polynomial pipeline is:  0.6332096172725036
```

Model Comparison Summary

In [25]:
```
# Create a summary of all models' performance
models = ['Simple Linear Regression', 'Multiple Linear Regression',
          '1st Degree Polynomial', '3rd Degree Polynomial', '5th Degree Polynomial'
          'Multi-variable Polynomial Pipeline']

# Calculate metrics for multiple linear regression
mse_mlr = mean_squared_error(df['Price'], Y_hat)
r2_mlr = lm1.score(Z, Y)

# Compile results
results = pd.DataFrame({
    'Model': models,
    'R-squared': [r2_score_slr, r2_mlr, r_squared_1, r_squared_3, r_squared_5, r2_s
    'MSE': [mse_slr, mse_mlr, mean_squared_error(Y, p1(X)), mean_squared_error(Y, p
            mean_squared_error(Y, p5(X)), mean_squared_error(Y, ypipe)]
})
```

```
# Display results
results.sort_values('R-squared', ascending=False)
```

Out[25]:

	Model	R-squared	MSE
5	Multi-variable Polynomial Pipeline	0.633210	120595.807462
1	Multiple Linear Regression	0.508251	161680.572639
4	5th Degree Polynomial	0.303082	229137.295481
3	3rd Degree Polynomial	0.266926	241024.863038
0	Simple Linear Regression	0.134444	284583.440587
2	1st Degree Polynomial	0.134444	284583.440587