# Black-Scholes Option Pricer

## Created By: Ted Kratt

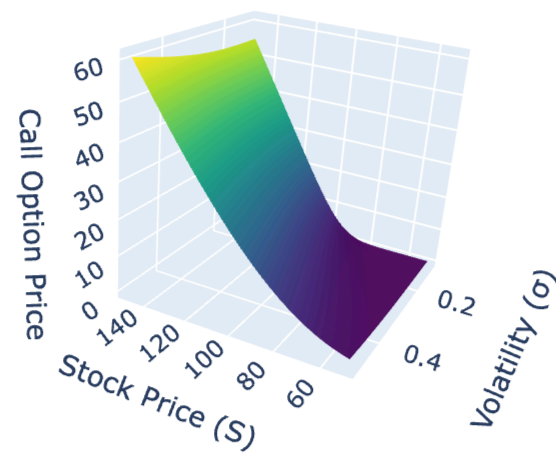
In [22]:

```
import math
from scipy.stats import norm

def black_scholes(S, K, T, r, sigma, option_type="call"):

    d1 = (math.log(S / K) + (r + 0.5 * sigma ** 2) * T) / (sigma * math.
    d2 = d1 - sigma * math.sqrt(T)


    if option_type == "call":
        price = S * norm.cdf(d1) - K * math.exp(-r * T) * norm.cdf(d2)
    elif option_type == "put":
        price = K * math.exp(-r * T) * norm.cdf(-d2) - S * norm.cdf(-d1)
    else:
        raise ValueError("Invalid option type. Use 'call' or 'put'.")

    return price


S = 100
K = 105
T = 1
r = 0.05
sigma = 0.2

call_price = black_scholes(S, K, T, r, sigma, option_type="call")
put_price = black_scholes(S, K, T, r, sigma, option_type="put")

print(f"Call Option Price: {call_price:.2f}")
print(f"Put Option Price: {put_price:.2f}")
```

```
Call Option Price: 8.02
Put Option Price: 7.90
```

In [23]:
```
import numpy as np
import plotly.graph_objects as go

# Define the Black-Scholes function
def black_scholes(S, K, T, r, sigma, option_type="call"):
    d1 = (math.log(S / K) + (r + 0.5 * sigma ** 2) * T) / (sigma * math.
    d2 = d1 - sigma * math.sqrt(T)

    if option_type == "call":
        return S * norm.cdf(d1) - K * math.exp(-r * T) * norm.cdf(d2)
    elif option_type == "put":
        return K * math.exp(-r * T) * norm.cdf(-d2) - S * norm.cdf(-d1)



K = 100
T = 1
r = 0.05


S = np.linspace(50, 150, 50)  #price range
sigma = np.linspace(0.1, 0.5, 50)  # volatility range
S_grid, sigma_grid = np.meshgrid(S, sigma)


call_prices = np.array([
    black_scholes(S, K, T, r, sigma, option_type="call")
    for S, sigma in zip(np.ravel(S_grid), np.ravel(sigma_grid))
]).reshape(S_grid.shape)


fig = go.Figure(data=[go.Surface(z=call_prices, x=S, y=sigma, colorscale
fig.update_layout(
    title="Black-Scholes Call Option Price Surface",
    scene=dict(
        xaxis_title="Stock Price (S)",
        yaxis_title="Volatility (σ)",
        zaxis_title="Call Option Price"
    )
)
fig.show()
```

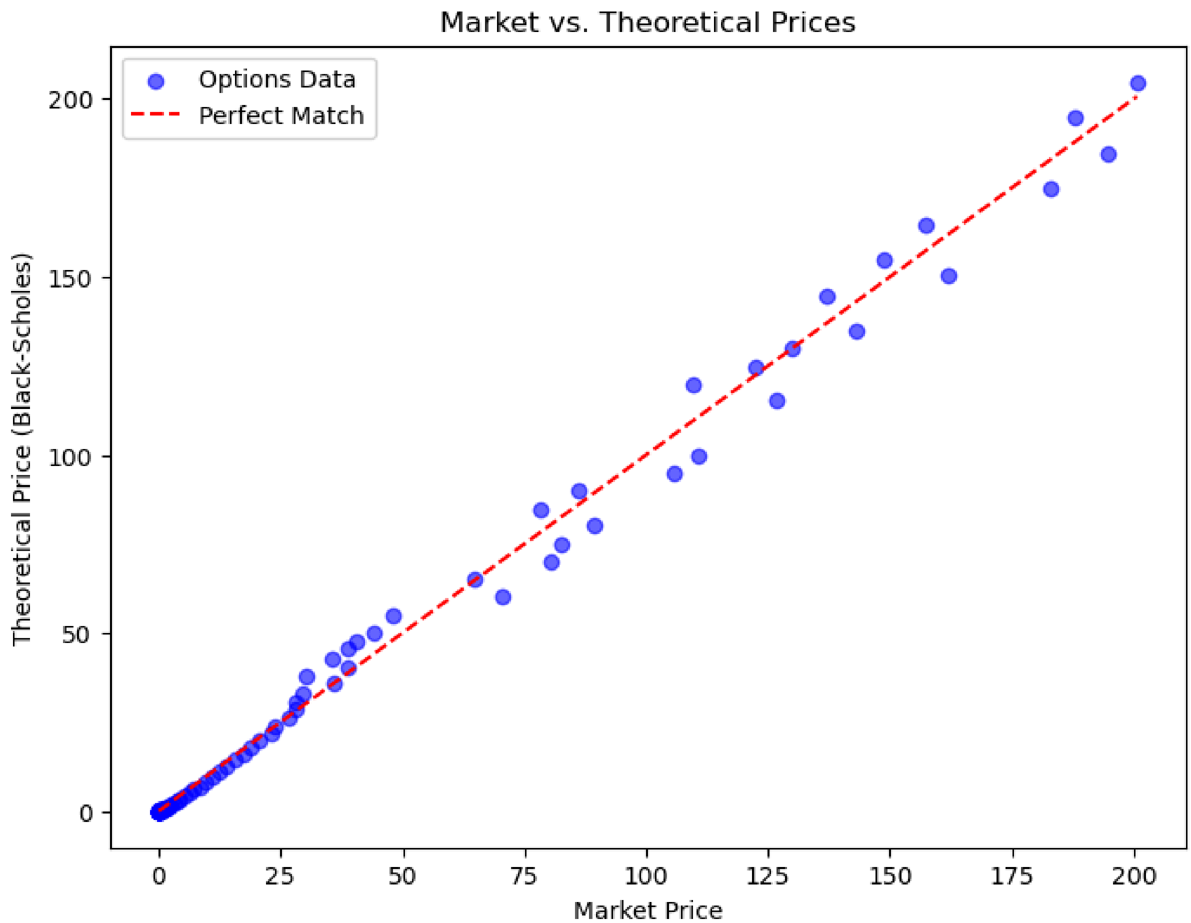
Black-Scholes Call Option Price Surface



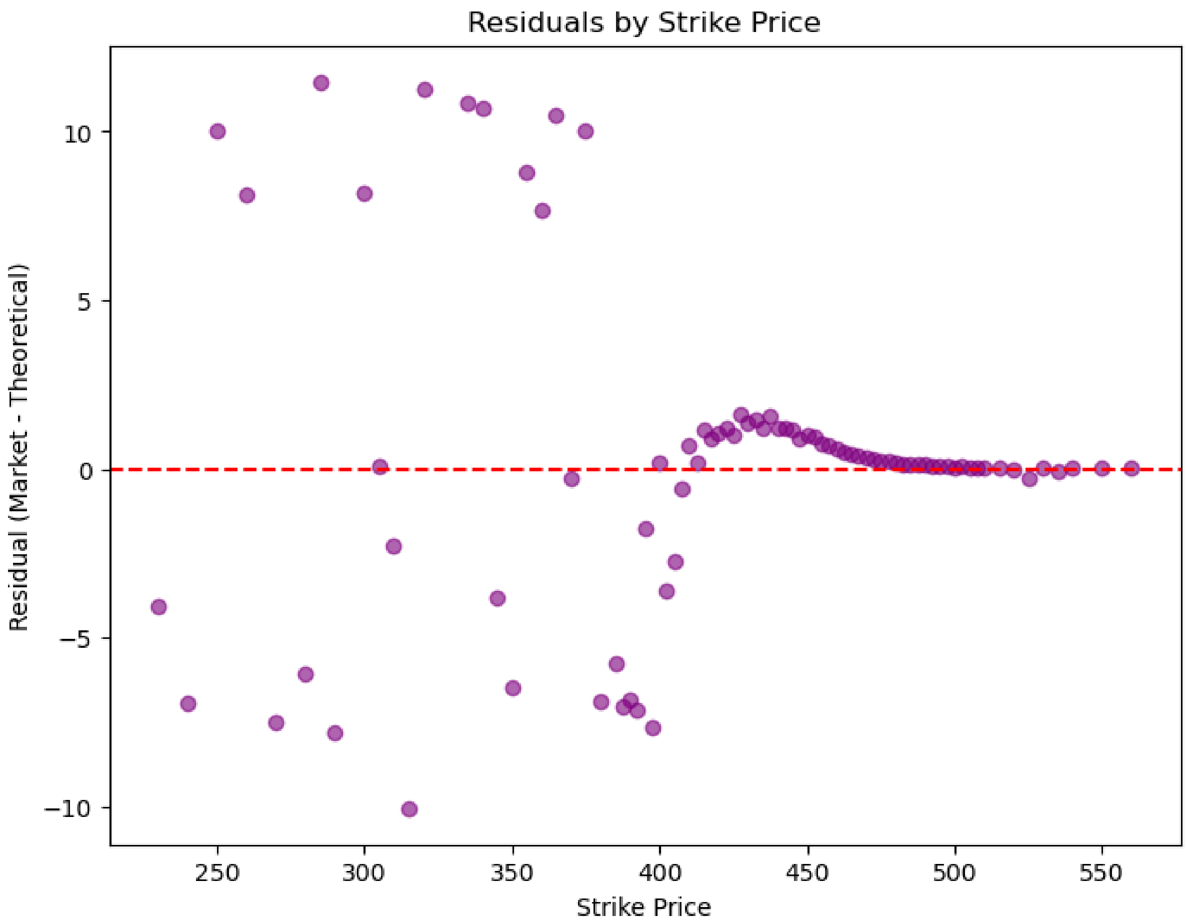
In [29]:

```
import yfinance as yf
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error
import pandas as pd
from mpl_toolkits.mplot3d import Axes3D


def black_scholes(S, K, T, r, sigma, option_type="call"):
    d1 = (math.log(S / K) + (r + 0.5 * sigma ** 2) * T) / (sigma * math.
    d2 = d1 - sigma * math.sqrt(T)
    if option_type == "call":
        return S * norm.cdf(d1) - K * math.exp(-r * T) * norm.cdf(d2)
    elif option_type == "put":
        return K * math.exp(-r * T) * norm.cdf(-d2) - S * norm.cdf(-d1)
    else:
        raise ValueError("Invalid option type. Use 'call' or 'put'.")


stock = yf.Ticker("MSFT")
options_dates = stock.options
expiry_date = options_dates[0]
options_chain = stock.option_chain(expiry_date)
calls = options_chain.calls


latest_stock_price = stock.history().iloc[-1]["Close"]


real_prices = []
calculated_prices = []
strike_prices = []
implied_vols = []
T = (pd.Timestamp(expiry_date) - pd.Timestamp.now()).days / 365
r = 0.05

#option chain loop
for _, row in calls.iterrows():
    K = row['strike']
    market_price = row['lastPrice']
    sigma = row['impliedVolatility']

    if not np.isnan(sigma) and not np.isnan(market_price):

        theoretical_price = black_scholes(latest_stock_price, K, T, r, s


        real_prices.append(market_price)
        calculated_prices.append(theoretical_price)
        strike_prices.append(K)
        implied_vols.append(sigma)
```

In [30]:
```
#market vs theoretical prices
plt.figure(figsize=(8, 6))
plt.scatter(real_prices, calculated_prices, c='blue', alpha=0.6, label="
plt.plot([min(real_prices), max(real_prices)], [min(real_prices), max(re
plt.xlabel("Market Price")
plt.ylabel("Theoretical Price (Black-Scholes)")
plt.title("Market vs. Theoretical Prices")
plt.legend()
plt.show()
```

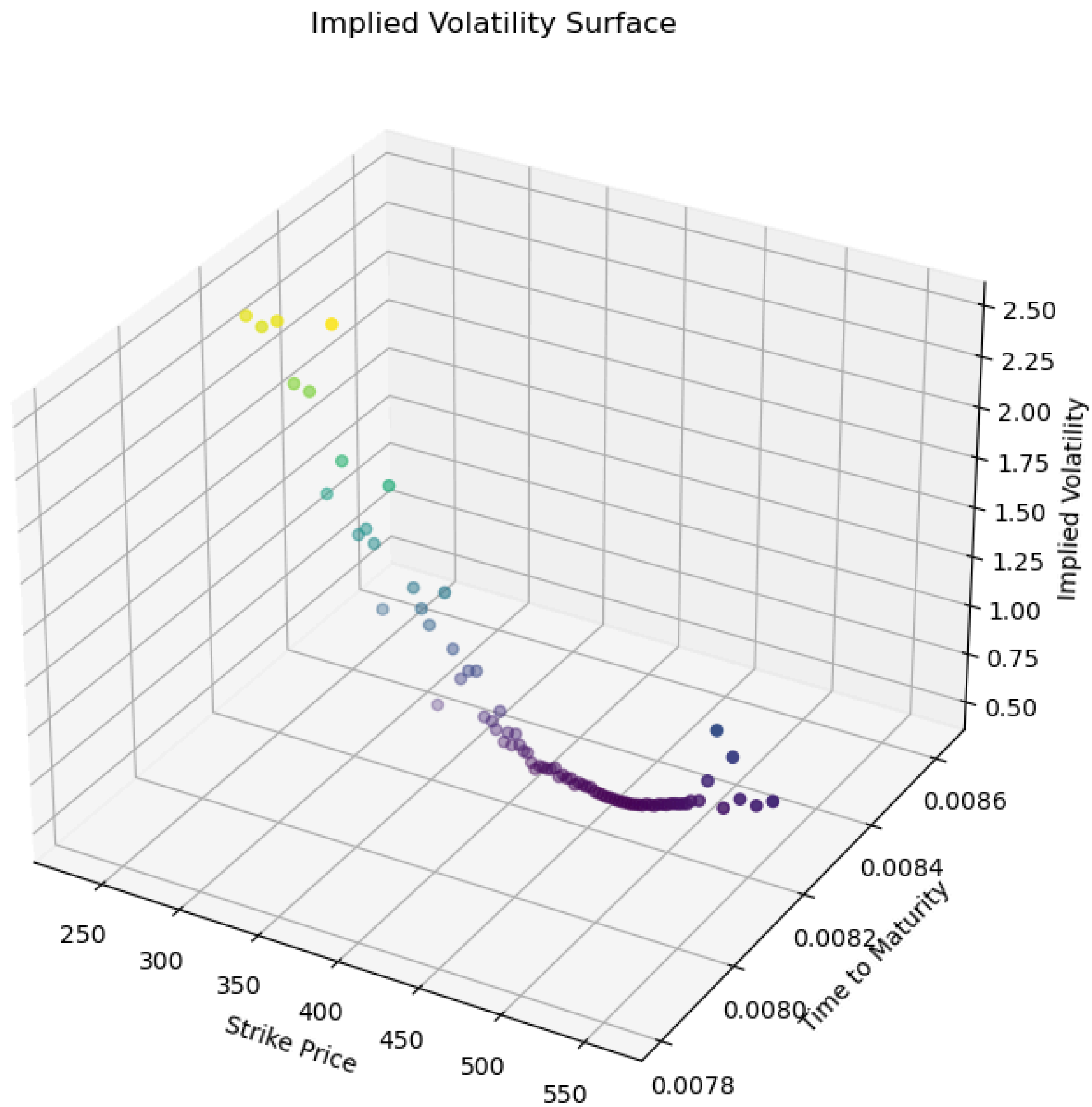


In [27]:
```
#residual plot
residuals = np.array(real_prices) - np.array(calculated_prices)
plt.figure(figsize=(8, 6))
plt.scatter(strike_prices, residuals, c='purple', alpha=0.6)
plt.axhline(0, color='red', linestyle='--')
plt.xlabel("Strike Price")
plt.ylabel("Residual (Market - Theoretical)")
plt.title("Residuals by Strike Price")
plt.show()
```




```
Mean Squared Error: 21.80444467469177
```

In [25]:
```
#implied volatility surface
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(strike_prices, [T] * len(strike_prices), implied_vols, c=impl
ax.set_xlabel("Strike Price")
ax.set_ylabel("Time to Maturity")
ax.set_zlabel("Implied Volatility")
ax.set_title("Implied Volatility Surface")
plt.show()
```




In [28]:
```
#mean squared error
mse = mean_squared_error(real_prices, calculated_prices)
print(f"Mean Squared Error: {mse}")
```

```
Mean Squared Error: 21.80444467469177
```

In [ ]: