```
 #Loading the Data
import pandas as pd

# Sample data creation for demonstration
data = {
    'Age': [25, 30, 45, None, 22],
    'Salary': [50000, 60000, 80000, 90000, None],
    'Country': ['USA', 'France', 'Germany', 'USA', 'France'],
    'Purchased': ['No', 'Yes', 'No', 'Yes', 'No']
}

# Creating a DataFrame
df = pd.DataFrame(data)

# Display the data
print(df)
```

```
    Age   Salary  Country Purchased
0  25.0  50000.0      USA        No
1  30.0  60000.0   France       Yes
2  45.0  80000.0  Germany        No
3   NaN  90000.0      USA       Yes
4  22.0      NaN   France        No
```

Handling Missing Values

```
from sklearn.impute import SimpleImputer

# Handling missing values for numerical columns
imputer = SimpleImputer(strategy='mean')
df['Age'] = imputer.fit_transform(df[['Age']])
df['Salary'] = imputer.fit_transform(df[['Salary']])

print("After handling missing values:")
print(df)
```

```
After handling missing values:
    Age   Salary  Country Purchased
0  25.0  50000.0      USA        No
1  30.0  60000.0   France       Yes
2  45.0  80000.0  Germany        No
3  30.5  90000.0      USA       Yes
4  22.0  70000.0   France        No
```

```
?SimpleImputer
```

3. Encoding Categorical Data

```
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.compose import ColumnTransformer

# Encoding categorical data for 'Country' using OneHotEncoder
# The 'Country' column will be replaced with three new columns (one for each country)
ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), ['Country'])], remainder='passthrough')
df_encoded = ct.fit_transform(df)

# Convert the result back to a DataFrame for easier viewing
df_encoded = pd.DataFrame(df_encoded, columns=['France', 'Germany', 'USA', 'Age', 'Salary', 'Purchased'])

print("After encoding categorical data:")
print(df_encoded)

# Encoding 'Purchased' column using LabelEncoder
label_encoder = LabelEncoder()
df_encoded['Purchased'] = label_encoder.fit_transform(df_encoded['Purchased'])

print("After encoding 'Purchased' column:")
print(df_encoded)
```

```
After encoding categorical data:
  France Germany  USA   Age   Salary Purchased
0    0.0     0.0  1.0  25.0  50000.0        No
1    1.0     0.0  0.0  30.0  60000.0       Yes
2    0.0     1.0  0.0  45.0  80000.0        No
3    0.0     0.0  1.0  30.5  90000.0       Yes
4    1.0     0.0  0.0  22.0  70000.0        No
After encoding 'Purchased' column:
  France Germany  USA   Age   Salary  Purchased
```

```
   0    0.0     0.0  1.0  25.0  50000.0          0
   1    1.0     0.0  0.0  30.0  60000.0          1
   2    0.0     1.0  0.0  45.0  80000.0          0
   3    0.0     0.0  1.0  30.5  90000.0          1
   4    1.0     0.0  0.0  22.0  70000.0          0
```

4. Feature Scaling

```
from sklearn.preprocessing import StandardScaler

# Feature scaling for 'Age' and 'Salary'
scaler = StandardScaler()
df_encoded[['Age', 'Salary']] = scaler.fit_transform(df_encoded[['Age', 'Salary']])

print("After feature scaling:")
print(df_encoded)
# z = (x - u) / s
```

```
After feature scaling:
   France Germany  USA       Age    Salary  Purchased
0    0.0     0.0  1.0 -0.695145 -1.414214          0
1    1.0     0.0  0.0 -0.063195 -0.707107          1
2    0.0     1.0  0.0  1.832656  0.707107          0
3    0.0     0.0  1.0  0.000000  1.414214          1
4    1.0     0.0  0.0 -1.074315  0.000000          0
```

5. Splitting the Data into Training and Testing Sets

```
from sklearn.model_selection import train_test_split

# Splitting the data into features (X) and target (y)
X = df_encoded.drop('Purchased', axis=1)
y = df_encoded['Purchased']

# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

print("Training set:")
print(X_train)
print(y_train)

print("Testing set:")
print(X_test)
print(y_test)
```

```
Training set:
   France Germany  USA       Age    Salary
4    1.0     0.0  0.0 -1.074315  0.000000
2    0.0     1.0  0.0  1.832656  0.707107
0    0.0     0.0  1.0 -0.695145 -1.414214
3    0.0     0.0  1.0  0.000000  1.414214
4    0
2    0
0    0
3    1
Name: Purchased, dtype: int64
Testing set:
   France Germany  USA       Age    Salary
1    1.0     0.0  0.0 -0.063195 -0.707107
1    1
Name: Purchased, dtype: int64
```

6. Feature engineering involves creating new features or transforming existing ones to improve model performance. Here are some common techniques with Python code examples:
7. Creating New Features Date-Time Features: Extracting components like year, month, day, or hour from a datetime column. Interaction Features: Combining two or more features to create interaction terms.
8. Polynomial Features Polynomial Transformations: Generating polynomial and interaction features.
9. Binning Binning Continuous Variables: Converting a continuous variable into categorical by binning.
10. Log Transformation Log Transformation: Applying a logarithmic transformation to reduce skewness.
11. Feature Selection Removing Low Variance Features: Removing features with low variance. Let's demonstrate these techniques with code.

```
import pandas as pd
import numpy as np

# Sample data for feature engineering
data = {
    'Age': [25, 30, 45, 35, 22],
    'Salary': [50000, 60000, 80000, 90000, 75000],
    'Country': ['USA', 'France', 'Germany', 'USA', 'France'],
    'Purchased': ['No', 'Yes', 'No', 'Yes', 'No'],
    'JoinDate': pd.to_datetime(['2015-03-01', '2017-07-12', '2018-01-01', '2020-02-20', '2019-05-15'])
}

df = pd.DataFrame(data)
print("Original DataFrame:")
print(df)
```

```
Original DataFrame:
   Age  Salary  Country Purchased   JoinDate
0   25   50000      USA        No 2015-03-01
1   30   60000   France       Yes 2017-07-12
2   45   80000  Germany        No 2018-01-01
3   35   90000      USA       Yes 2020-02-20
4   22   75000   France        No 2019-05-15
```

```
#a) Date-Time Features
# Extracting year, month, and day from 'JoinDate'
df['Year'] = df['JoinDate'].dt.year
df['Month'] = df['JoinDate'].dt.month
df['Day'] = df['JoinDate'].dt.day

print("After extracting date-time features:")
print(df)
```

```
After extracting date-time features:
   Age  Salary  Country Purchased   JoinDate  Year  Month  Day
0   25   50000      USA        No 2015-03-01  2015      3    1
1   30   60000   France       Yes 2017-07-12  2017      7   12
2   45   80000  Germany        No 2018-01-01  2018      1    1
3   35   90000      USA       Yes 2020-02-20  2020      2   20
4   22   75000   France        No 2019-05-15  2019      5   15
```

```
#b) Interaction Features
# Creating interaction between 'Age' and 'Salary'
df['Age_Salary_Interaction'] = df['Age'] * df['Salary']

print("After creating interaction feature:")
print(df)
```

```
After creating interaction feature:
   Age  Salary  Country Purchased   JoinDate  Year  Month  Day  \
0   25   50000      USA        No 2015-03-01  2015      3    1
1   30   60000   France       Yes 2017-07-12  2017      7   12
2   45   80000  Germany        No 2018-01-01  2018      1    1
3   35   90000      USA       Yes 2020-02-20  2020      2   20
4   22   75000   France        No 2019-05-15  2019      5   15

   Age_Salary_Interaction
0                 1250000
1                 1800000
2                 3600000
3                 3150000
4                 1650000
```

```
#2. Polynomial Features

from sklearn.preprocessing import PolynomialFeatures

# Creating polynomial features for 'Age' and 'Salary'
poly = PolynomialFeatures(degree=2, include_bias=False)
poly_features = poly.fit_transform(df[['Age', 'Salary']])

# Convert the result back to a DataFrame
df_poly = pd.DataFrame(poly_features, columns=poly.get_feature_names_out(['Age', 'Salary']))

# Concatenate with original DataFrame
df = pd.concat([df, df_poly], axis=1)

print("After polynomial transformation:")
print(df)
```

```
After polynomial transformation:
   Age  Salary  Country Purchased   JoinDate  Year  Month  Day  \
0   25   50000      USA        No 2015-03-01  2015      3    1
```

```
   1   30   60000    France        Yes 2017-07-12  2017      7   12
   2   45   80000  Germany         No 2018-01-01  2018      1    1
   3   35   90000      USA        Yes 2020-02-20  2020      2   20
   4   22   75000    France        No 2019-05-15  2019      5   15

     Age_Salary_Interaction   Age   Salary   Age^2  Age Salary      Salary^2
   0                1250000  25.0  50000.0   625.0   1250000.0  2.500000e+09
   1                1800000  30.0  60000.0   900.0   1800000.0  3.600000e+09
   2                3600000  45.0  80000.0  2025.0   3600000.0  6.400000e+09
   3                3150000  35.0  90000.0  1225.0   3150000.0  8.100000e+09
   4                1650000  22.0  75000.0   484.0   1650000.0  5.625000e+09
```

```
#3. Binning Continuous Variables

# Binning 'Age' into categories: 'Young', 'Middle-aged', 'Senior'
bins = [0, 25, 40, 100]
labels = ['Young', 'Middle-aged', 'Senior']
df['Age_Group'] = pd.cut(df['Age'], bins=bins, labels=labels)

print("After binning 'Age':")
print(df)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-11-434f33adc81e> in <cell line: 6>()
      4 bins = [0, 25, 40, 100]
      5 labels = ['Young', 'Middle-aged', 'Senior']
----> 6 df['Age_Group'] = pd.cut(df['Age'], bins=bins, labels=labels)
      7
      8 print("After binning 'Age':")

                              1 frames
/usr/local/lib/python3.10/dist-packages/pandas/core/reshape/tile.py in _preprocess_for_cut(x)
    610         x = np.asarray(x)
    611     if x.ndim != 1:
--> 612         raise ValueError("Input array must be 1 dimensional")
    613
    614     return x

ValueError: Input array must be 1 dimensional
```

```
#Log Transformation
# Log transformation of 'Salary' to reduce skewness
df['Log_Salary'] = np.log(df['Salary'])

print("After log transformation of 'Salary':")
print(df)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-12-570904002dd3> in <cell line: 3>()
      1 #Log Transformation
      2 # Log transformation of 'Salary' to reduce skewness
----> 3 df['Log_Salary'] = np.log(df['Salary'])
      4
      5 print("After log transformation of 'Salary':")

                              1 frames
/usr/local/lib/python3.10/dist-packages/pandas/core/frame.py in _set_item_frame_value(self, key, value)
   4237
   4238         if len(value.columns) != 1:
-> 4239             raise ValueError(
   4240                 "Cannot set a DataFrame with multiple columns to the single "
   4241                 f"column {key}"

ValueError: Cannot set a DataFrame with multiple columns to the single column Log_Salary
```

```
#5. Feature Selection
#a) Removing Low Variance Features

from sklearn.feature_selection import VarianceThreshold

# Removing features with low variance (variance threshold of 0.01)
selector = VarianceThreshold(threshold=0.01)
df_high_var = selector.fit_transform(df[['Age', 'Salary', 'Age_Salary_Interaction']])

# Convert back to DataFrame
df_high_var = pd.DataFrame(df_high_var, columns=['Age', 'Salary', 'Age_Salary_Interaction'])

print("After removing low variance features:")
print(df_high_var)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-13-7dcdb6f5e111> in <cell line: 11>()
      9 
     10 # Convert back to DataFrame
---> 11 df_high_var = pd.DataFrame(df_high_var, columns=['Age', 'Salary', 'Age_Salary_Interaction'])
     12 
     13 print("After removing low variance features:")

2 frames
/usr/local/lib/python3.10/dist-packages/pandas/core/internals/construction.py in _check_values_indices_shape_match(values, index, columns)
    418         passed = values.shape
    419         implied = (len(index), len(columns))
--> 420         raise ValueError(f"Shape of passed values is {passed}, indices imply {implied}")
    421 
    422 

ValueError: Shape of passed values is (5, 5), indices imply (5, 3)
```

```python
from sklearn.feature_selection import VarianceThreshold

# Removing features with low variance (variance threshold of 0.01)
selector = VarianceThreshold(threshold=0.01)
df_high_var = selector.fit_transform(df[['Age', 'Salary', 'Age_Salary_Interaction']])

# Convert back to DataFrame
df_high_var = pd.DataFrame(df_high_var, columns=['Age', 'Salary', 'Age_Salary_Interaction'])

print("After removing low variance features:")
print(df_high_var)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-14-10614f4c18b7> in <cell line: 8>()
      6 
      7 # Convert back to DataFrame
----> 8 df_high_var = pd.DataFrame(df_high_var, columns=['Age', 'Salary', 'Age_Salary_Interaction'])
      9 
     10 print("After removing low variance features:")

2 frames
/usr/local/lib/python3.10/dist-packages/pandas/core/internals/construction.py in _check_values_indices_shape_match(values, index, columns)
    418         passed = values.shape
    419         implied = (len(index), len(columns))
--> 420         raise ValueError(f"Shape of passed values is {passed}, indices imply {implied}")
    421 
    422 

ValueError: Shape of passed values is (5, 5), indices imply (5, 3)
```