laa9ynoje

November 10, 2025

```
[83]: import pandas as pd
      import numpy as np
      import plotly.express as px
      from sklearn.model_selection import train_test_split
      from sklearn.preprocessing import StandardScaler
      from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
      from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
      from sklearn.metrics import confusion_matrix, classification_report, roc_curve,␣
       ↪auc
      import plotly.graph_objects as go
      from plotly.subplots import make_subplots
```

```
[84]: data = pd.read_csv('credit_risk_data-1.csv')

      data.head()
```

```
[84]:   application_id application_date  loan_amount  annual_income  \
      0       APP_2328       2022-01-01    132221.82       60451.82
      1        APP_558       2022-01-01    134906.42      114634.08
      2       APP_2477       2022-01-01     30285.19       82772.53
      3        APP_741       2022-01-01     32516.09       94023.36
      4        APP_145       2022-01-02     77900.99       53515.02

         employment_years  job_stability_score  credit_score  credit_utilization  \
      0               6.6                0.898           679               0.106
      1              10.3                0.808           718               0.030
      2              12.1                0.964           768               0.174
      3               9.1                0.690           670               0.141
      4               7.2                0.679           651               0.097

         payment_history_score  open_credit_lines  debt_to_income_ratio  \
      0                  0.876                  1                 0.451
      1                  0.719                  4                 0.090
      2                  0.775                  6                 0.201
      3                  0.993                  3                 0.322
      4                  0.946                  2                 0.222
```

```
   savings_ratio  asset_value  age education_level marital_status  \
0          0.500    352569.55   41     High School        Married
1          0.235    224364.21   46         Masters       Divorced
2          0.172    514765.55   44     High School        Widowed
3          0.368    182541.72   26       Bachelors         Single
4          0.324    223691.29   50      Associates         Single

   residential_stability  loan_status
0                    3.5            0
1                   11.4            0
2                    8.6            0
3                    3.9            0
4                    9.6            0
```

[85]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2500 entries, 0 to 2499
Data columns (total 18 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   application_id         2500 non-null   object
 1   application_date       2500 non-null   object
 2   loan_amount            2500 non-null   float64
 3   annual_income          2500 non-null   float64
 4   employment_years       2500 non-null   float64
 5   job_stability_score    2500 non-null   float64
 6   credit_score           2500 non-null   int64
 7   credit_utilization     2500 non-null   float64
 8   payment_history_score  2500 non-null   float64
 9   open_credit_lines      2500 non-null   int64
 10  debt_to_income_ratio   2500 non-null   float64
 11  savings_ratio          2500 non-null   float64
 12  asset_value            2500 non-null   float64
 13  age                    2500 non-null   int64
 14  education_level        2500 non-null   object
 15  marital_status         2500 non-null   object
 16  residential_stability  2500 non-null   float64
 17  loan_status            2500 non-null   int64
dtypes: float64(10), int64(4), object(4)
memory usage: 351.7+ KB
```

[86]:
```
data.describe()
```

[86]:
```
         loan_amount  annual_income  employment_years  job_stability_score  \
count    2500.000000    2500.000000       2500.000000          2500.000000
mean   155716.305344   67707.807596          6.675640             0.634643
```

```
std     149605.357952   27302.931731          3.488021             0.293276
min       5000.000000   15000.000000          0.000000             0.011000
25%      42984.517500   47475.317500          4.000000             0.375500
50%      97054.315000   66963.475000          6.700000             0.752000
75%     213214.992500   87347.642500          9.300000             0.866000
max     500000.000000  149929.960000         19.300000             0.999000

       credit_score  credit_utilization  payment_history_score  \
count   2500.000000         2500.000000            2500.000000
mean     681.728400            0.358176               0.740733
std       88.683309            0.289995               0.285966
min      334.000000            0.004000               0.029000
25%      642.750000            0.131000               0.517500
50%      700.000000            0.246000               0.880500
75%      743.000000            0.592250               0.956000
max      850.000000            0.998000               1.000000

       open_credit_lines  debt_to_income_ratio  savings_ratio     asset_value  \
count        2500.000000           2500.000000    2500.000000     2500.000000
mean            3.451600              0.408094       0.320784   175666.741236
std             2.083793              0.224736       0.192079   182652.568930
min             0.000000              0.009000       0.000000      550.630000
25%             2.000000              0.228000       0.161000    49513.082500
50%             3.000000              0.359000       0.327000   121018.750000
75%             5.000000              0.565000       0.464000   235513.902500
max            11.000000              0.979000       0.893000  1000000.000000

               age  residential_stability  loan_status
count  2500.000000            2500.000000  2500.000000
mean     42.045600               6.023200     0.265600
std      12.092395               3.205397     0.441741
min      18.000000               0.000000     0.000000
25%      34.000000               3.600000     0.000000
50%      42.000000               5.900000     0.000000
75%      50.000000               8.400000     1.000000
max      75.000000              16.400000     1.000000
```

[87]:
```
data.isna().sum()
```

[87]:
```
application_id           0
application_date         0
loan_amount              0
annual_income            0
employment_years         0
job_stability_score      0
credit_score             0
credit_utilization       0
```

```
payment_history_score    0
open_credit_lines        0
debt_to_income_ratio     0
savings_ratio            0
asset_value              0
age                      0
education_level          0
marital_status           0
residential_stability    0
loan_status              0
dtype: int64
```

This database contains 17 columns, there is no null values in the database. In general this is a good database to work, as it contains several variables to work and clean data to determine loan_status.

[88]:
```
fig = px.histogram(
            data,
            x='loan_status',
            title='<b>Distribution of Loan Status</b>',
            color='loan_status'  # Optional: This gives each bar a different␣
 ↪color
        )
fig.update_layout(
    title_x = 0.5
)

fig.show()
```

[89]:
```
default_rate = data['loan_status'].mean()
print(f"The exact default rate is: {default_rate}")
```

```
The exact default rate is: 0.2656
```

[90]:
```
key_predictors = ['credit_score', 'annual_income', 'debt_to_income_ratio',␣
 ↪'payment_history_score', 'savings_ratio']

#Visualization improved with the use of gemini
for col in key_predictors:
    # No inner loop needed.
    # Pass the full 'data' and use 'color' to differentiate.
    fig = px.histogram(
        data,
        x=col,
        color='loan_status',  # Automatically creates colors for 0 and 1
        barmode='overlay',    # Layers the histograms (use 'group' for␣
 ↪side-by-side)
        title=f'<b>Distribution of {col} by Loan Status</b>'
    )

    # Add opacity so the overlaid histograms are visible
    fig.update_traces(opacity=0.75)

    fig.update_layout(
        title_x=0.5,
        xaxis_title=col,
        yaxis_title='Count'
    )
    fig.show()
```

It is a clear difference in the distributions for the two loan_status types, this means both groups are really appart from each other.

[91]:
```
eduaction_levels = data['education_level'].unique()
martial_stats = data['marital_status'].unique()
```

[92]:
```
education_defaults = data.groupby('education_level')['loan_status'].mean().
  ↪reset_index()

education_defaults = education_defaults.rename(columns={'loan_status': 'Mean␣
  ↪Default Rate'})


fig_edu = px.bar(
    education_defaults,
    x='education_level',
    y='Mean Default Rate',
    title='<b>Mean Default Rate by Education Level</b>',
    labels={'education_level': 'Education Level', 'Mean Default Rate': 'Average␣
  ↪Default Rate'}
)

# Sort the bars by the default rate (highest to lowest) for easier reading
fig_edu.update_layout(xaxis={'categoryorder':'total descending'}, title_x = 0.5)


fig_edu.show()
```

[93]:
```
marital_defaults = data.groupby('marital_status')['loan_status'].mean().
  ↪reset_index()
marital_defaults = marital_defaults.rename(columns={'loan_status': 'Mean␣
  ↪Default Rate'})

# Create the bar plot
fig_marital = px.bar(
    marital_defaults,
    x='marital_status',
    y='Mean Default Rate',
    title='<b>Mean Default Rate by Marital Status</b>',
    labels={'marital_status': 'Marital Status', 'Mean Default Rate': 'Average␣
↪Default Rate'}
)

# Sort the bars by the default rate
fig_marital.update_layout(xaxis={'categoryorder':'total descending'}, title_x =␣
↪0.5)

fig_marital.show()
```

[94]:
```
all_num_predictors = ['loan_amount', 'annual_income', 'employment_years',␣
↪'job_stability_score', 'credit_score', 'credit_utilization',␣
↪'payment_history_score', 'open_credit_lines', 'debt_to_income_ratio',␣
↪'savings_ratio', 'asset_value', 'age', 'residential_stability']



corr_mat = data[all_num_predictors].corr()

# heatmap con Plotly
fig = px.imshow(
    corr_mat,
    x=corr_mat.columns,
    y=corr_mat.index,
    color_continuous_scale='RdBu_r',
    zmin=-1, zmax=1,
    aspect="auto",
    text_auto=".2f"
)

fig.update_layout(
    title='<b>Correlation Matrix for all numeric predictors</b>',
    width=800,
    height=800,
    xaxis_title="Variables",
    yaxis_title="Variables",
    title_x =0.5
)

fig.show()
```

There are few variables with high collinearity, the one that have the most is payment_history_score with job_stabilty_score, this is the only correlation greater than 0.8.

[95]:
```python
#Data Preprocessing
print("Original data shape:", data.shape)
print("Original categorical variables:")
print("Education levels:", data['education_level'].unique())
print("Marital status:", data['marital_status'].unique())
```

```
Original data shape: (2500, 18)
Original categorical variables:
Education levels: ['High School' 'Masters' 'Bachelors' 'Associates' 'Doctorate']
Marital status: ['Married' 'Divorced' 'Widowed' 'Single']
```

[96]:
```python
# Create dummy variables for categorical features
data_processed = pd.get_dummies(data, columns=['education_level',␣
↪'marital_status'], drop_first=True)

print("New data shape:", data_processed.shape)
print("Columns created:")
new_categorical_columns = [col for col in data_processed.columns if␣
↪'education_level_' in col or 'marital_status_' in col]
for col in new_categorical_columns:
    print(f"  - {col}")
```

```
New data shape: (2500, 23)
Columns created:
  - education_level_Bachelors
  - education_level_Doctorate
  - education_level_High School
  - education_level_Masters
  - marital_status_Married
  - marital_status_Single
  - marital_status_Widowed
```

[97]:
```python
# Drop non-predictive columns
x = data_processed.drop(['application_id', 'loan_status'], axis=1)
y = data_processed['loan_status']

print(f"Predictors x: {x.shape}")
print(f"Target y: {y.shape}")
print(f"N of features: {x.shape[1]}")
print(f"Feature names: {list(x.columns)}")
```

```
Predictors x: (2500, 21)
Target y: (2500,)
N of features: 21
Feature names: ['application_date', 'loan_amount', 'annual_income',
'employment_years', 'job_stability_score', 'credit_score', 'credit_utilization',
'payment_history_score', 'open_credit_lines', 'debt_to_income_ratio',
'savings_ratio', 'asset_value', 'age', 'residential_stability',
'education_level_Bachelors', 'education_level_Doctorate', 'education_level_High
School', 'education_level_Masters', 'marital_status_Married',
'marital_status_Single', 'marital_status_Widowed']
```

[98]:
```
#Train test split
X_train, X_test, y_train, y_test = train_test_split(
    x, y,
    test_size=0.2,
    random_state=42,
    stratify=y
)

print(f"Training set X_train: {X_train.shape}, y_train: {y_train.shape}")
print(f"Test set X_test: {X_test.shape}, y_test: {y_test.shape}")

print(f"Target distribution in dataset:")
print(y.value_counts(normalize=True))
print(f"Target distribution in training set:")
print(y_train.value_counts(normalize=True))
print(f"Target distribution in test set:")
print(y_test.value_counts(normalize=True))
```

```
Training set X_train: (2000, 21), y_train: (2000,)
Test set X_test: (500, 21), y_test: (500,)
Target distribution in dataset:
loan_status
0    0.7344
1    0.2656
Name: proportion, dtype: float64
Target distribution in training set:
loan_status
0    0.7345
1    0.2655
Name: proportion, dtype: float64
Target distribution in test set:
loan_status
0    0.734
1    0.266
Name: proportion, dtype: float64
```

[99]:
```
# Clean date column and standardize
if 'application_date' in X_train.columns:
    X_train['application_date'] = pd.to_datetime(X_train['application_date'])
    X_test['application_date'] = pd.to_datetime(X_test['application_date'])
    ref_date = X_train['application_date'].max()
    X_train['app_date_days'] = (ref_date - X_train['application_date']).dt.days
    X_test['app_date_days'] = (ref_date - X_test['application_date']).dt.days
    X_train = X_train.drop('application_date', axis=1)
    X_test = X_test.drop('application_date', axis=1)

# Standardization
scaler = StandardScaler()
X_train_scaled = pd.DataFrame(scaler.fit_transform(X_train), columns=X_train.
 ↪columns, index=X_train.index)
X_test_scaled = pd.DataFrame(scaler.transform(X_test), columns=X_train.columns,␣
 ↪index=X_test.index)

print(f"Data standardized Train: {X_train_scaled.shape}, Test: {X_test_scaled.
 ↪shape}")
```

```
Data standardized Train: (2000, 21), Test: (500, 21)
```

0.1 Statistical Assumption Testing

Statistical Assumptions for LDA vs. QDA

Linear Discriminant Analysis and Quadratic Discriminant Analysis are based on the assumption of multivariate normality that predictors follow approximately a multivariate normal distribution within each class (defaulters vs. non-defaulters).

Evaluation of Assumptions Based on EDA:

Multivariate Normality: From our EDA histograms, we observe that: - credit_score: Shows approximately normal distribution for both classes - annual_income: Reasonably normal distribution with some right skew - debt_to_income_ratio: Approximately normal for non-defaulters, slightly skewed for defaulters

While not all variables are perfectly normal, both LDA and QDA are reasonably robust to moderate deviations from normality, especially with our sample size of 2,500 observations.

Homogeneity of Covariance Matrices (Key Differentiating Assumption): - LDA assumes that all classes share the same covariance matrix - QDA relaxes this assumption and allows each class to have its own covariance matrix

Evidence from our EDA: - The distribution plots show clear differences in variance between defaulters and non-defaulters for several variables - For example, in `credit_score`, defaulters show wider spread and lower mean compared to non-defaulters - Similar pattern observed in `annual_income` and other financial metrics

Hypothesis: Given the visible differences in distributions and variances between the two classes in our EDA plots, we expect that the covariance matrices are unequal. Therefore, we hypothesize that QDA will outperform LDA as it can better capture these class-specific covariance structures.

[100]:
```
lda = LinearDiscriminantAnalysis()
lda.fit(X_train_scaled, y_train)

# Check model performance on training data
train_accuracy = lda.score(X_train_scaled, y_train)
```

```
test_accuracy = lda.score(X_test_scaled, y_test)

print(f"LDA Training Accuracy: {train_accuracy:.4f}")
print(f"LDA Test Accuracy: {test_accuracy:.4f}")

#Interpret Coefficients
lda_coef = pd.DataFrame({
    'Feature': X_train_scaled.columns,
    'Coefficient': lda.coef_[0]
})

#Sort by absolute value to identify most important features
lda_coef['Abs_Coefficient'] = np.abs(lda_coef['Coefficient'])
lda_coef_sorted = lda_coef.sort_values('Abs_Coefficient', ascending=False)

print("Top 10 Most Important Features from LDA:")
print(lda_coef_sorted.head(10).to_string(index=False))
```

```
LDA Training Accuracy: 1.0000
LDA Test Accuracy: 1.0000
Top 10 Most Important Features from LDA:
              Feature  Coefficient  Abs_Coefficient
payment_history_score   -15.471241        15.471241
  job_stability_score   -13.051790        13.051790
   credit_utilization    11.766475        11.766475
 debt_to_income_ratio     4.471469         4.471469
         credit_score    -3.980499         3.980499
        savings_ratio    -2.995611         2.995611
     employment_years    -2.367811         2.367811
residential_stability    -1.702223         1.702223
        annual_income    -1.587700         1.587700
    open_credit_lines    -1.275071         1.275071
```

[101]:
```
# Create a bar plot of top coefficients
fig_lda_coef = px.bar(
    lda_coef_sorted.head(10),
    x='Coefficient',
    y='Feature',
    orientation='h',
    title='<b>Top 10 Most Important LDA Coefficients</b>',
    color='Coefficient',
    color_continuous_scale='RdBu_r',
    labels={'Coefficient': 'Standardized Coefficient Value', 'Feature': ''}
)
fig_lda_coef.update_layout(
    title_x=0.5,
    yaxis={'categoryorder': 'total ascending'},
    xaxis_title="Coefficient Value",
    yaxis_title="Feature",
    showlegend=False
)

fig_lda_coef.show()
```

0.2 LDA Coefficients Interpretation

The coefficients tell us the importance and direction of each variable's relationship with default risk, a big negative coefficient reduces default risk while a big positive one increases it.

Key Drivers of Default Risk:

1. **payment_history_score (Coefficient: -15.47):** This is, by a wide margin, the most important predictor the negative sign indicates that a higher payment history score is strongly associated with a lower probability of default (being a good payer).
2. **job_stability_score (Coefficient: -13.05):** The second most important driver, similarly, a higher job stability score dramatically decreases the likelihood of default.
3. **credit_utilization (Coefficient: 11.77):** This is the strongest positive driver, the positive sign means that higher credit utilization (using a higher percentage of your available credit) is strongly associated with higher default risk.
4. **debt_to_income_ratio (Coefficient: 4.47):** A positive coefficient indicating that as an applicant's debt to income ratio increases so does their probability of default.
5. **credit_score (Coefficient: -3.98):** As expected, a higher credit score has a negative sign, reducing the likelihood of default.

Key Insights: The model identifies a clear risk profile. Default risk increases massively for applicants who use a large amount of their available credit and carry a high debt burden relative to their income. Conversely, risk drops significantly for those with a strong payment history and stable employment.

[102]:
```
qda = QuadraticDiscriminantAnalysis()
qda.fit(X_train_scaled, y_train)

# Check model performance
qda_train_accuracy = qda.score(X_train_scaled, y_train)
print(f"QDA Training Accuracy: {qda_train_accuracy:.4f}")
qda_test_accuracy = qda.score(X_test_scaled, y_test)
print(f"QDA Test Accuracy: {qda_test_accuracy:.4f}")
```

```
QDA Training Accuracy: 0.9995
QDA Test Accuracy: 1.0000
```

[103]:
```
#predictions and probability scores
y_pred_lda = lda.predict(X_test_scaled)
y_pred_qda = qda.predict(X_test_scaled)
y_score_lda = lda.predict_proba(X_test_scaled)[:, 1]
y_score_qda = qda.predict_proba(X_test_scaled)[:, 1]
```

[104]:
```python
#subplots for confusion matrices
fig = make_subplots(
    rows=1, cols=2,
    subplot_titles=['LDA Confusion Matrix', 'QDA Confusion Matrix'],
    horizontal_spacing=0.15
)

# LDA
cm_lda = confusion_matrix(y_test, y_pred_lda)
heatmap_lda = go.Heatmap(
    z=cm_lda,
    x=['Predicted 0', 'Predicted 1'],
    y=['Actual 0', 'Actual 1'],
    text=cm_lda,
    texttemplate="%{text}",
    textfont={"size": 16},
    colorscale='Blues',
    showscale=False
)
fig.add_trace(heatmap_lda, 1, 1)

# QDA
cm_qda = confusion_matrix(y_test, y_pred_qda)
heatmap_qda = go.Heatmap(
    z=cm_qda,
    x=['Predicted 0', 'Predicted 1'],
    y=['Actual 0', 'Actual 1'],
    text=cm_qda,
    texttemplate="%{text}",
    textfont={"size": 16},
    colorscale='Blues',
    showscale=False
)
fig.add_trace(heatmap_qda, 1, 2)

fig.update_layout(
    title_text="<b>Confusion Matrices: LDA vs QDA</b>",
    title_x=0.5,
    width=800,
    height=400
)

fig.show()
```

[105]:
```python
print("LDA Classification Report")
print(classification_report(y_test, y_pred_lda))
print("QDA Classification Report")
print(classification_report(y_test, y_pred_qda))
```

```
LDA Classification Report
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       367
           1       1.00      1.00      1.00       133

    accuracy                           1.00       500
   macro avg       1.00      1.00      1.00       500
weighted avg       1.00      1.00      1.00       500

QDA Classification Report
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       367
           1       1.00      1.00      1.00       133

    accuracy                           1.00       500
   macro avg       1.00      1.00      1.00       500
weighted avg       1.00      1.00      1.00       500
```

[106]:
```
#Calculate ROC curves and AUC
fpr_lda, tpr_lda, _ = roc_curve(y_test, y_score_lda)
fpr_qda, tpr_qda, _ = roc_curve(y_test, y_score_qda)
auc_lda = auc(fpr_lda, tpr_lda)
auc_qda = auc(fpr_qda, tpr_qda)

print(f"LDA AUC: {auc_lda:.4f}")
print(f"QDA AUC: {auc_qda:.4f}")

fig_roc = go.Figure()
# LDA ROC curve
fig_roc.add_trace(go.Scatter(
    x=fpr_lda, y=tpr_lda,
    mode='lines',
    name=f'LDA (AUC = {auc_lda:.4f})',
    line=dict(width=3, color='blue')
))

# QDA ROC curve
fig_roc.add_trace(go.Scatter(
    x=fpr_qda, y=tpr_qda,
    mode='lines',
    name=f'QDA (AUC = {auc_qda:.4f})',
    line=dict(width=3, color='red')
))

#random classifier line
fig_roc.add_trace(go.Scatter(
    x=[0, 1], y=[0, 1],
    mode='lines',
    name='Random Classifier',
    line=dict(dash='dash', color='black')
))

fig_roc.update_layout(
    title='<b>ROC Curves: LDA vs QDA</b>',
    title_x=0.5,
    xaxis_title='False Positive Rate',
    yaxis_title='True Positive Rate',
    width=700,
    height=500,
    showlegend=True
)

fig_roc.update_xaxes(range=[0, 1])
fig_roc.update_yaxes(range=[0, 1])

fig_roc.show()
```

```
LDA AUC: 1.0000
QDA AUC: 1.0000
```

In this analysis, we compared two classification models, Linear Discriminant Analysis and Quadratic Discriminant Analysis, to predict loan default risk, the evaluation was performed on a held out test set (20% of the data).

Test Set Performance Summary:

Metric	LDA Model	QDA Model
Accuracy	1.0000	1.0000
AUC	1.0000	1.0000
Recall (Class 1 - Default)	1.00	1.00
Precision (Class 1 - Default)	1.00	1.00

Results Analysis: Both models achieved perfect performance on the test set, with 100% accuracy and an AUC of 1.0000, the confusion matrices and classification reports confirm that both models correctly classified every single defaulter and non defaulter in the test set.

This result indicates the dataset is perfectly separable. As noted in the EDA and confirmed by the LDA coefficients, there are features (most notably payment_history_score and job_stability_score) that draw a perfect line between the two classes.

Model Selection: Given that both models have identical, perfect performance, either could be chosen.

The LDA Model is preferable for its interpretability, as it provides direct coefficients that explain why an applicant is flagged as high risk, which is invaluable for the business. QDA Model confirmed our hypothesis from Section 4 (that the covariance matrices were unequal), but its complexity (a quadratic decision boundary) is unnecessary when a simple linear boundary (LDA) already solves the problem perfectly.

Technical Recommendation: I select the LDA model, it achieves perfect accuracy while offering the benefit of model interpretability, allowing LendSmart to understand the key factors driving risk.