Awarding Body: Arden University
Programme Name: MSc Data Science
Module Name (and Part if applicable): Machine Learning
Assessment Title: The analysis of multiple regression machine learning models to predict credit scores
Student Number: STU237176
Tutor Name: Nour Albaarini
Word Count: 4403 Please refer to the Word Count Policy on your Module Page for guidance

The analysis of multiple regression machine learning models to predict credit scores

Task 1- The machine learning project

Introduction

This study compares and analyses different machine learning regression models to predict future credit scores. Machine learning is an evolving field used extensively in the financial sector and by SMART Banking plc to complete this evaluation.

Loading and preprocessing data

The first step of creating any machine learning algorithm is to import the essential packages (Figure 1) including NumPy which is the foundation of numerical computing for example, Pandas to help the work with data structures, Statsmodels for the statistical analysis. For data visualization Matplotlib and Seaborn are added. Scikit- learn offers all the regression models and related tools.

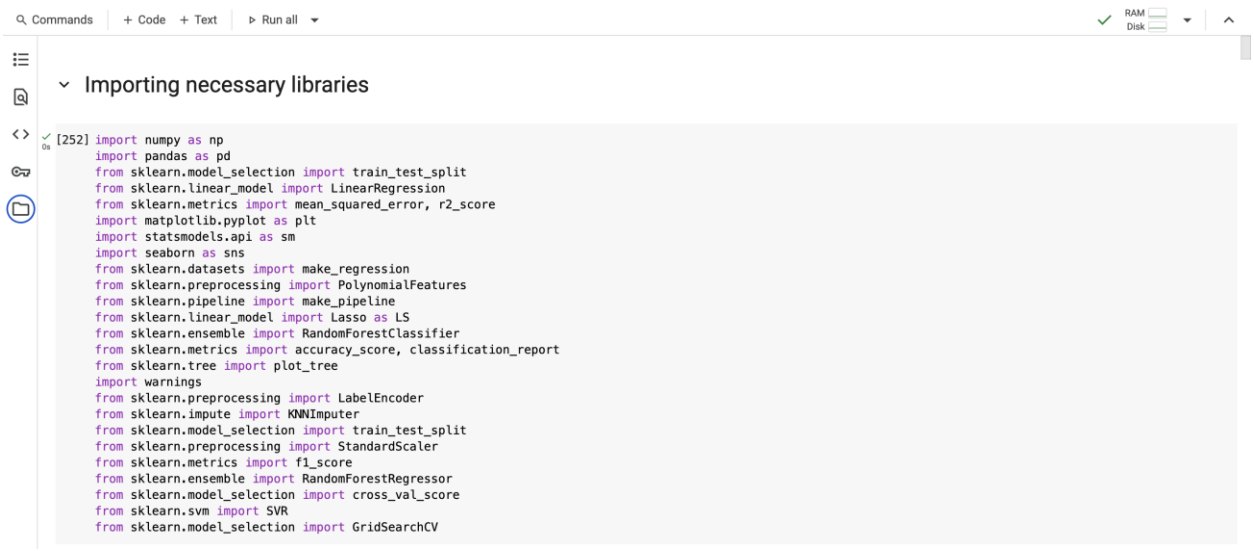


Figure 1: Importing necessary libraries

Exploratory data analysis and performing descriptive statistics are crucial steps in preparing the data for regression models after the data has been loaded (Figure 2). Analysing the basic statistics will help to gain a better understanding of the dataset, find patterns, relationships, and outliers. Part of the mentioned data analysis is to apply the

df.describe() and *df.info()* functions (Figure 3). Figure 4 shows the number of missing values in each column.

Data preprocessing

```
[254] df = pd.read_csv('SMART Banking data_4035.csv')
      df.head(10)
```

	customer_id	credit_score	country	gender	age	tenure	balance	estimated_salary	Unnamed: 8
0	1563462	619.0	France	Female	42.0	2.0	NaN	11348.88	NaN
1	15647311	68.0	Spain	Female	41.0	1.0	8387.86	112542.58	NaN
2	1561934	52.0	France	Female	42.0	8.0	15966.80	113931.57	NaN
3	1571354	699.0	France	Female	39.0	1.0	NaN	93826.63	NaN
4	15737888	85.0	Spain	Female	43.0	2.0	12551.82	7984.10	NaN
5	1557412	645.0	Spain	Male	44.0	8.0	113755.78	149756.71	NaN
6	15592531	822.0	France	Male	5.0	7.0	NaN	162.80	NaN
7	15656148	376.0	Germany	Female	29.0	4.0	11546.74	119346.88	NaN
8	15792365	51.0	France	Male	44.0	4.0	14251.70	7494.50	NaN
9	15592389	684.0	France	Male	27.0	2.0	13463.88	71725.73	NaN

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
[3] df.tail(10)
```

	customer_id	credit_score	country	gender	age	tenure	balance	estimated_salary	Unnamed: 8
2010	1566147	87.0	Spain	Male	45.0	6.0	2226291.0	148684.81	NaN
2011	15168982	26.0	France	Male	24.0	6.0	1393791.0	54764.30	NaN
2012	1557765	735.0	France	Male	41.0	8.0	1739291.0	17286.17	NaN
2013	155979	63.0	Spain	Female	32.0	8.0	1116291.0	38555.46	NaN
2014	1569839	50.0	Spain	Female	38.0	4.0	1326871.0	118913.53	NaN
2015	15725797	662.0	France	Male	46.0	3.0	NaN	8487.75	NaN
2016	1562546	84.0	France	Female	38.0	5.0	NaN	187611.16	NaN
2017	15738291	557.0	France	Male	25.0	3.0	NaN	121458.29	NaN
2018	1536816	656.0	Germany	Female	36.0	2.0	1368152.0	1741.95	NaN
2019	15777211	671.0	France	Male	44.0	9.0	NaN	38433.35	NaN

Figure 2: Loading the banking data

```
df.describe()
```

	customer_id	credit_score	age	tenure	balance	estimated_salary	Unnamed: 8
count	2.020000e+03	2009.000000	2014.000000	1938.000000	1.309000e+03	2020.000000	0.0
mean	9.593544e+06	525.802887	35.508937	4.858101	9.466032e+04	67223.410317	NaN
std	7.142025e+06	248.621152	14.029019	2.655002	4.147254e+05	64795.973787	NaN
min	1.567000e+03	5.000000	2.000000	1.000000	1.950000e+01	6.360000	NaN
25%	1.569085e+06	486.000000	29.000000	2.000000	1.282972e+04	9985.462500	NaN
50%	1.559484e+07	618.000000	36.000000	5.000000	8.276742e+04	44250.070000	NaN
75%	1.571238e+07	694.000000	43.000000	7.000000	1.295557e+05	124809.885000	NaN
max	1.581536e+07	849.000000	82.000000	9.000000	1.431293e+07	222626.980000	NaN

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2020 entries, 0 to 2019
Data columns (total 9 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   customer_id       2020 non-null   int64  
 1   credit_score      2009 non-null   float64
 2   country           2020 non-null   object 
 3   gender            2020 non-null   object 
 4   age               2014 non-null   float64
 5   tenure            1938 non-null   float64
 6   balance           1309 non-null   float64
 7   estimated_salary  2020 non-null   float64
 8   Unnamed: 8        0 non-null      float64
dtypes: float64(6), int64(1), object(2)
memory usage: 142.2+ KB
```

Figure 3: Statistics and data types

```
df.isnull().sum()
```

	0
customer_id	0
credit_score	11
country	0
gender	0
age	6
tenure	82
balance	711
estimated_salary	0
Unnamed: 8	2020

dtype: int64

Figure 4: Looking for missing values

Missing values are likely to lead to biased results and poor performance in machine learning models if they are not handled correctly. Columns like *Unnamed: 8* and *customer _id* are unnecessary to the investigation therefore they can be eliminated (Figure 5). The missing values in the *balance* column will be replaced with the median value of the balances instead of deleting these instances.

Dropping unnecessary columns; rows with missing data

```
df_dropped_column = df.drop(columns=['Unnamed: 8','customer_id'])
df_new = df_dropped_column.dropna(subset=['credit_score', 'age', 'tenure'])
```

Figure 5: Dealing with the missing values in the *credit score, age,* and *tenure* columns

As it was mentioned previously, exploratory data analysis is an important step to recognize patterns in the data, therefore it is useful to have a look at the distribution of the different variables: the *credit score* (Figure 6, 7)*, balance* (Figure 8, 9), *estimated salary* (Figure 10, 11), *age* (Figure 12, 13)*, tenure* (Figure 14, 15) and *gender* (Figure 16,17).

Visualisation

```
plt.figure(figsize=(10, 6))
sns.histplot(data=df_new, x='credit_score', hue='country', bins=30, kde=True)
plt.title('Distribution of credit score')
plt.xlabel('Credit Score')
plt.ylabel('Frequency')
plt.grid(axis='y', alpha=0.75)
plt.show()
```

Figure 6: Distribution of credit score code

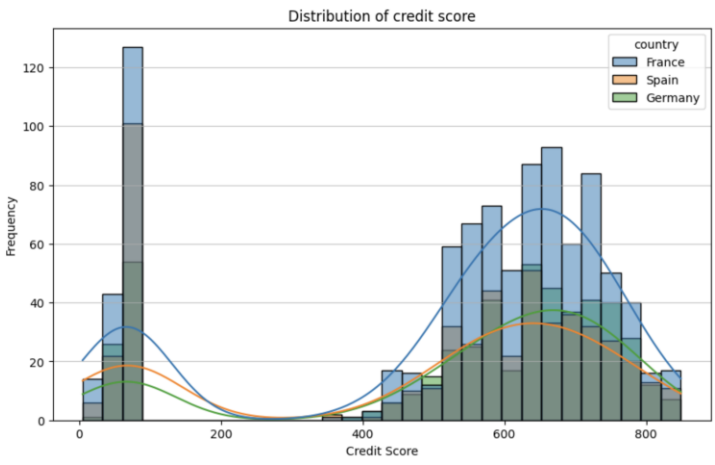


Figure 7: Distribution of credit score

```
plt.figure(figsize=(10, 6))
sns.histplot(data=df_new, x='balance', hue='country', bins=30, kde=True)
plt.title('Distribution of balance')
plt.xlabel('Balance')
plt.ylabel('Frequency')
plt.grid(axis='y', alpha=0.75)
plt.show()
```

Figure 8: Distribution of balance code

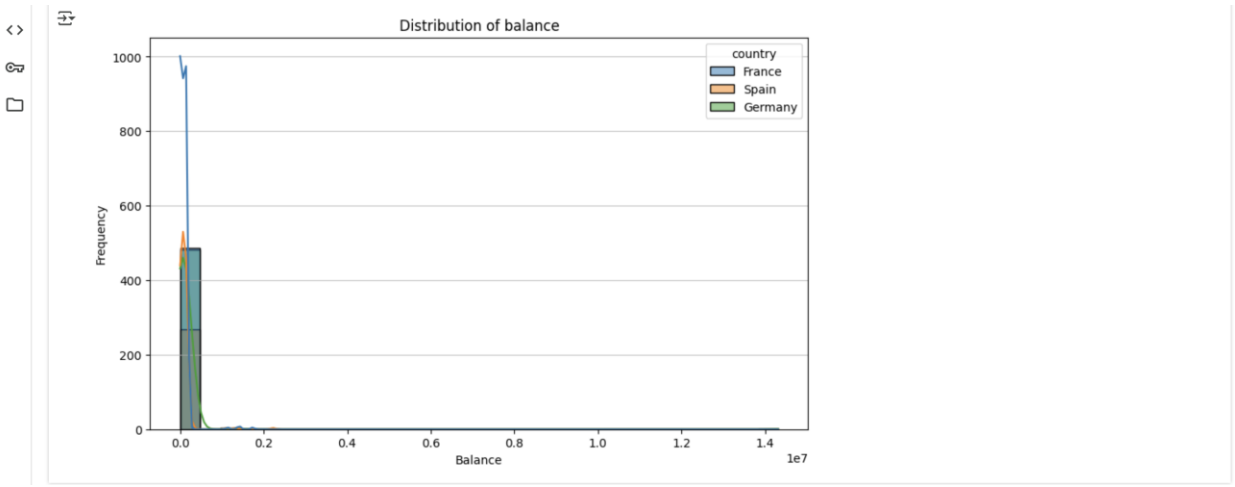


Figure 9: Distribution of balance

```
plt.figure(figsize=(10, 6))
sns.histplot(data=df_new, x='estimated_salary', hue='country', bins=30, kde=True)
plt.title('Distribution of estimated salary')
plt.xlabel('Estimated salary')
plt.ylabel('Frequency')
plt.grid(axis='y', alpha=0.75)
plt.show()
```

Figure 10: Distribution of salary code

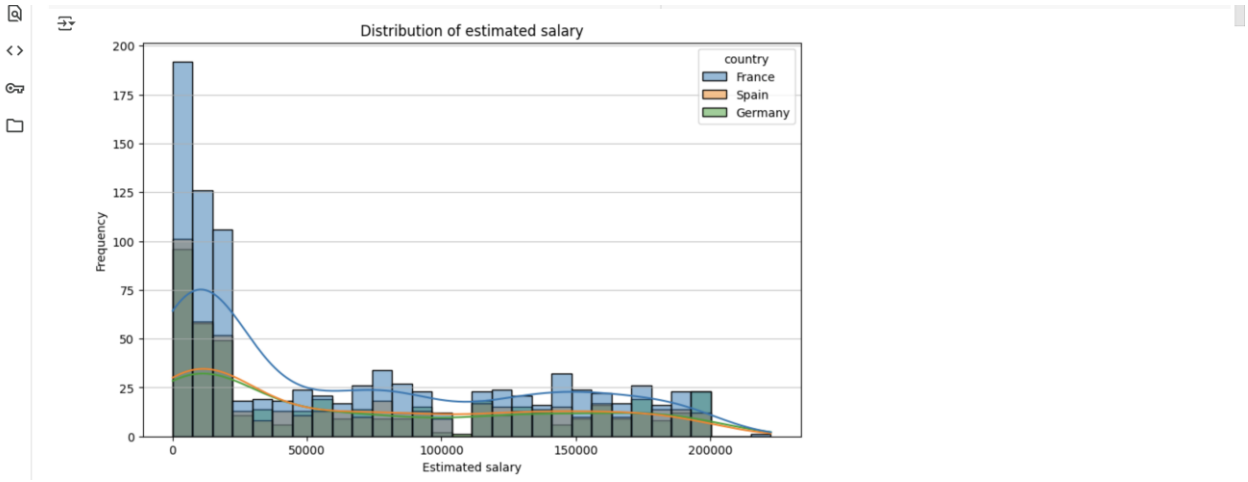


Figure 11: Distribution of salary

```
plt.figure(figsize=(10, 6))
sns.histplot(data=df_new, x='age', hue='country', bins=30, kde=True)
plt.title('Distribution of age')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.grid(axis='y', alpha=0.75)
plt.show()
```

Figure 12: Distribution of age code

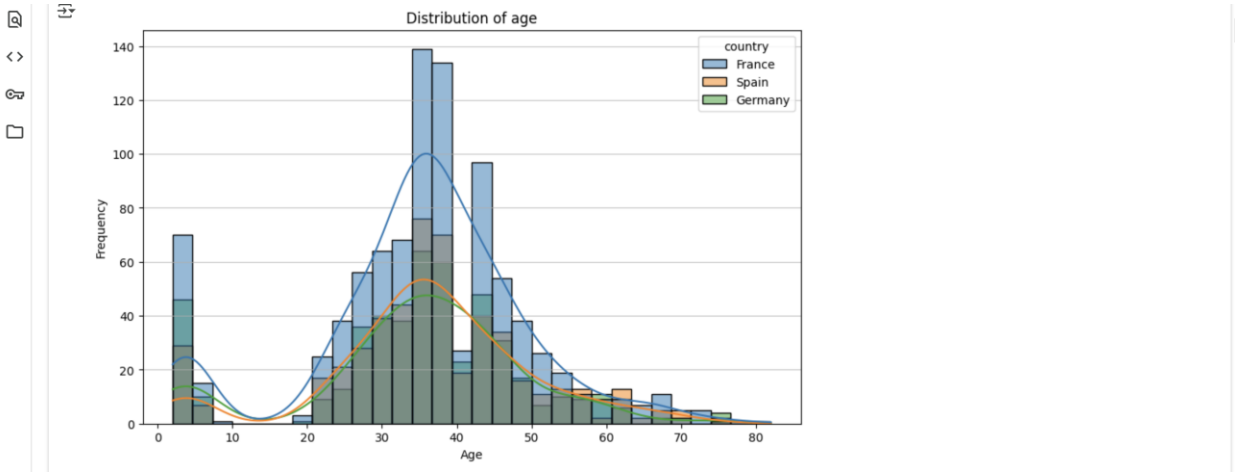


Figure 13: Distribution of age

```
plt.figure(figsize=(10, 6))
sns.histplot(data=df_new, x='tenure', hue='country', bins=30, kde=True)
plt.title('Distribution of tenure')
plt.xlabel('Tenure')
plt.ylabel('Frequency')
plt.grid(axis='y', alpha=0.75)
plt.show()
```

Figure 14: Distribution of tenure code

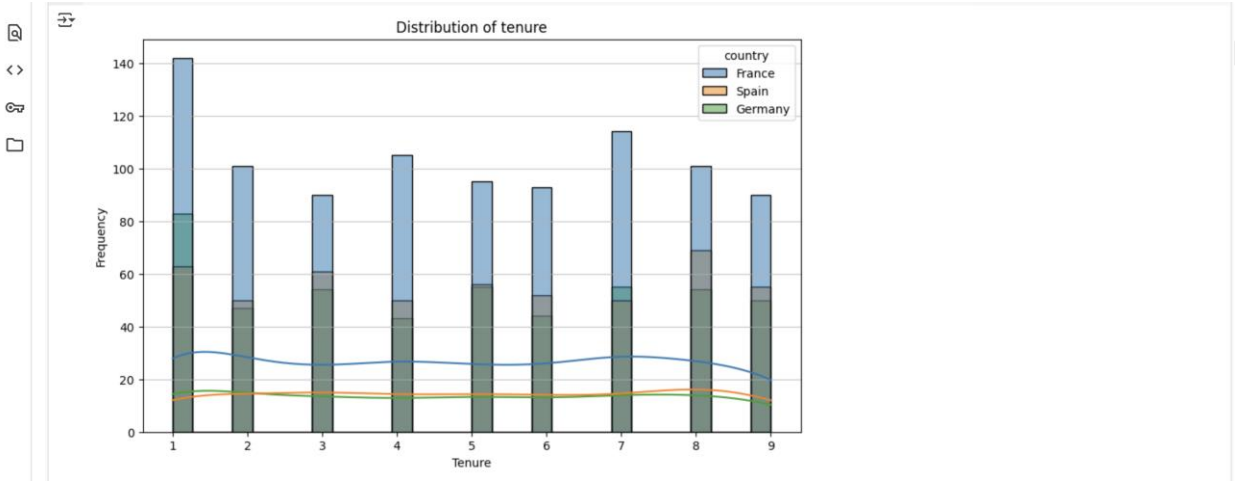


Figure 15: Distribution of tenure

```
plt.figure(figsize=(10, 6))
sns.histplot(data=df_new, x='gender', hue='country', multiple='dodge', shrink=.8)
plt.title('Distribution of Gender by Country')
plt.xlabel('Gender')
plt.ylabel('Frequency')
plt.grid(axis='y', alpha=0.75)
plt.show()
```

Figure 16: Distribution of gender code

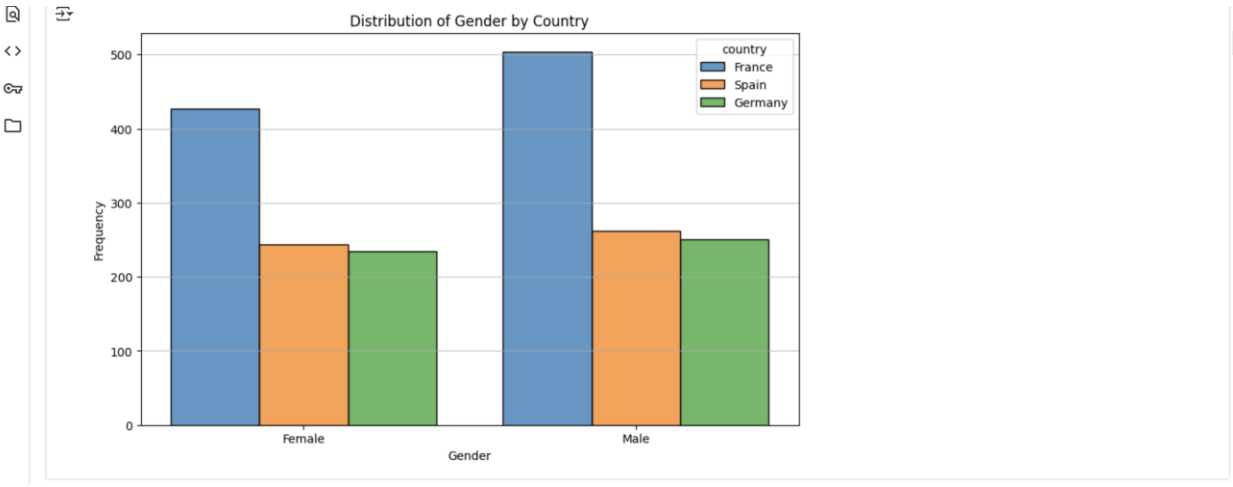


Figure 17: Distribution of gender by countries

Correlations

```
[266] df_numeric = df_new.select_dtypes(include=np.number)
```

```
df_numeric.corr()
```

	credit_score	age	tenure	balance	estimated_salary
credit_score	1.000000	-0.024310	-0.007174	-0.004613	-0.011239
age	-0.024310	1.000000	-0.048441	0.028572	0.017054
tenure	-0.007174	-0.048441	1.000000	-0.018230	0.006070
balance	-0.004613	0.028572	-0.018230	1.000000	-0.001102
estimated_salary	-0.011239	0.017054	0.006070	-0.001102	1.000000

Figure 18: Correlation

To reveal the relationship between features, it is vital to use correlation. Executing the *.corr()* function shows that the correlation between the different variables is extremely low (Figure 18), which means that there is no linear relationship between them. The correlation function can only be executed with numerical values, so the *gender* and *country* variables are not included in the function. This correlation is visualised on the heatmap below (Figure 19).

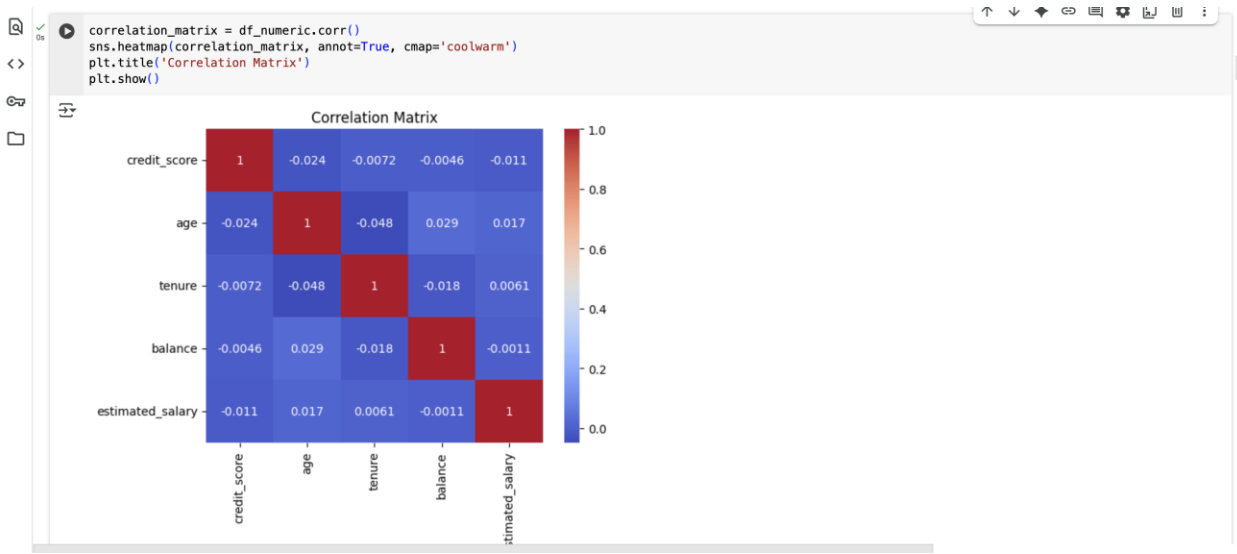


Figure 19: Correlation matrix

It is also important that outliers are removed from the dataset together with missing values. Some of these missing values have already been dropped; however, removing all the missing values from the *balance* column would notably affect the outcome due to their high number (711 exactly). Therefore, these values were replaced with the median *balance* value (Figure 20). Replacing missing values with the mean *balance* is another possibility, but because mean values are hugely affected by skewed data distribution and outliers, the median *balance* is a more suitable option. Another reason for filling in the missing *balance* values instead of removing them is the importance of the *balance* feature for this investigation. There are 394 outliers in the *credit score* column below the lower bound or above the upper bound (Figure 21), 221 in the *age column* (Figure 22), and 11 in the *balance* column (Figure 23). The outliers in the *credit score* column are not necessarily errors; they could be the effect of the wide range of credit scores. The main goal of this study is the prediction of future credit score values, also referred to as the dependent variable; therefore, outliers will remain in the credit score column. Figure 24 shows the shape of the new data frame after outliers have been removed.

Handling missing data and outliers

```
[269] median_balance = df_new['balance'].median()
      df_new['balance']=df_new['balance'].fillna(median_balance)
```

```
/tmp/ipython-input-269-228177869.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_new['balance']=df_new['balance'].fillna(median_balance)
```

```
for column in df_new.select_dtypes(include=np.number).columns:
  Q1 = df_new[column].quantile(0.25)
  Q3 = df_new[column].quantile(0.75)
  IQR = Q3 - Q1
  lower_bound = Q1 - 1.5 * IQR
  upper_bound = Q3 + 1.5 * IQR
  outliers = df_new[(df_new[column] < lower_bound) | (df_new[column] > upper_bound)]
  print(f'Outliers in {column}:')
  display(outliers)
```

Figure 20: Missing data in *balance* column

Outliers in credit_score:

	credit_score	country	gender	age	tenure	balance	estimated_salary
1	68.0	Spain	Female	41.0	1.0	8387.86	112542.58
2	52.0	France	Female	42.0	8.0	15966.80	113931.57
4	85.0	Spain	Female	43.0	2.0	12551.82	7984.10
8	51.0	France	Male	44.0	4.0	14251.70	7494.50
22	51.0	Spain	Female	38.0	4.0	83132.90	118913.53
...	...	...	...	...	...	...	...
2010	87.0	Spain	Male	45.0	6.0	2226291.00	148684.81
2011	26.0	France	Male	24.0	6.0	1393791.00	54764.30
2013	63.0	Spain	Female	32.0	8.0	1116291.00	38555.46
2014	50.0	Spain	Female	38.0	4.0	1326871.00	118913.53
2016	84.0	France	Female	38.0	5.0	83132.90	187611.16

394 rows × 7 columns

Figure 21: Outliers in *credit score*

Outliers in age:

	credit_score	country	gender	age	tenure	balance	estimated_salary
6	822.0	France	Male	5.0	7.0	83132.90	162.80
40	472.0	Spain	Male	4.0	4.0	83132.90	7154.22
58	511.0	Spain	Female	66.0	4.0	83132.90	1643.11
85	652.0	Spain	Female	75.0	1.0	83132.90	114675.75
87	729.0	France	Male	3.0	9.0	83132.90	151869.35
...	...	...	...	...	...	...	...
1964	8.0	France	Female	4.0	5.0	97764.41	9864.15
1980	554.0	France	Female	3.0	9.0	83132.90	432.30
1981	476.0	Spain	Female	69.0	1.0	1533.73	13426.34
1983	748.0	Spain	Female	4.0	4.0	83132.90	132368.47
1993	824.0	Germany	Male	6.0	8.0	13425.17	15346.16

221 rows × 7 columns

Figure 22: Outliers in *age*

Outliers in tenure:

	credit_score	country	gender	age	tenure	balance	estimated_salary

Outliers in balance:

	credit_score	country	gender	age	tenure	balance	estimated_salary
1996	534.0	France	Male	62.0	2.0	1135596.0	42763.12
1998	630.0	France	Male	43.0	5.0	1427823.0	29483.35
2007	612.0	Germany	Male	45.0	3.0	14312933.0	61327.26
2008	753.0	Germany	Male	58.0	1.0	1326291.0	1597.67
2009	540.0	Spain	Female	24.0	9.0	326291.0	2446.41
2010	87.0	Spain	Male	45.0	6.0	2226291.0	148684.81
2011	26.0	France	Male	24.0	6.0	1393791.0	54764.30
2012	735.0	France	Male	41.0	8.0	1739291.0	17286.17
2013	63.0	Spain	Female	32.0	8.0	1116291.0	38555.46
2014	50.0	Spain	Female	38.0	4.0	1326871.0	118913.53
2018	656.0	Germany	Female	36.0	2.0	1368152.0	1741.95

Outliers in estimated_salary:

	credit_score	country	gender	age	tenure	balance	estimated_salary

Figure 23: Outliers in *balance*

```
df_cleaned = df_new.copy()
for column in df_cleaned.select_dtypes(include=np.number).columns:
  Q1 = df_cleaned[column].quantile(0.25)
  Q3 = df_cleaned[column].quantile(0.75)
  IQR = Q3 - Q1
  lower_bound = Q1 - 3.0 * IQR
  upper_bound = Q3 + 3.0 * IQR
  df_cleaned = df_cleaned[(df_cleaned[column] >= lower_bound) & (df_cleaned[column] <= upper_bound)]

print("Shape of DataFrame after removing outliers:")
display(df_cleaned.shape)
```

```
Shape of DataFrame after removing outliers:
(1912, 7)
```

Figure 24: Shape of the cleaned data frame

New histograms of the cleaned data frame demonstrate the changes in the *credit score* (Figure 25, 26), *the balance* (Figure 27, 28), and the *age* columns (Figure 29, 30).

Visualisation after data cleaning ('credit score', 'age', 'balance' had outliers removed)

```
plt.figure(figsize=(10, 6))
sns.histplot(data=df_cleaned, x='credit_score', hue='country', bins=30, kde=True)
plt.title('Distribution of credit score')
plt.xlabel('Credit Score')
plt.ylabel('Frequency')
plt.grid(axis='y', alpha=0.75)
plt.show()
```

Figure 25: *Credit score* after cleaning code

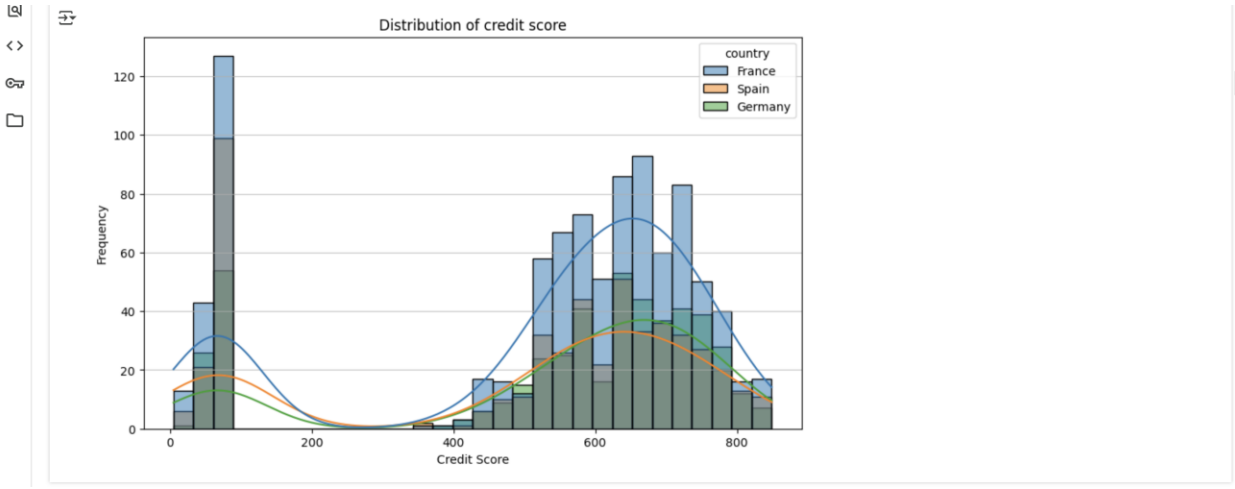


Figure 26: *Credit score* after cleaning

```
plt.figure(figsize=(10, 6))
sns.histplot(data=df_cleaned, x='balance', hue='country', bins=30, kde=True)
plt.title('Distribution of balance')
plt.xlabel('Balance')
plt.ylabel('Frequency')
plt.grid(axis='y', alpha=0.75)
plt.show()
```

Figure 27: *Balance* after cleaning code

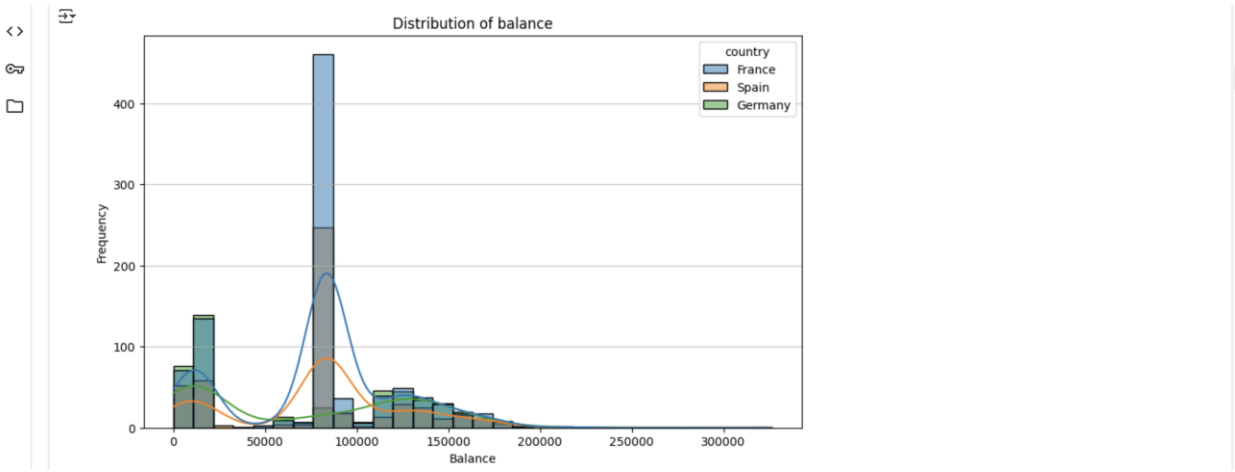


Figure 28: *Balance* after cleaning

```
plt.figure(figsize=(10, 6))
sns.histplot(data=df_cleaned, x='age', hue='country', bins=30, kde=True)
plt.title('Distribution of age')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.grid(axis='y', alpha=0.75)
plt.show()
```

Figure 29: *Age* after cleaning code

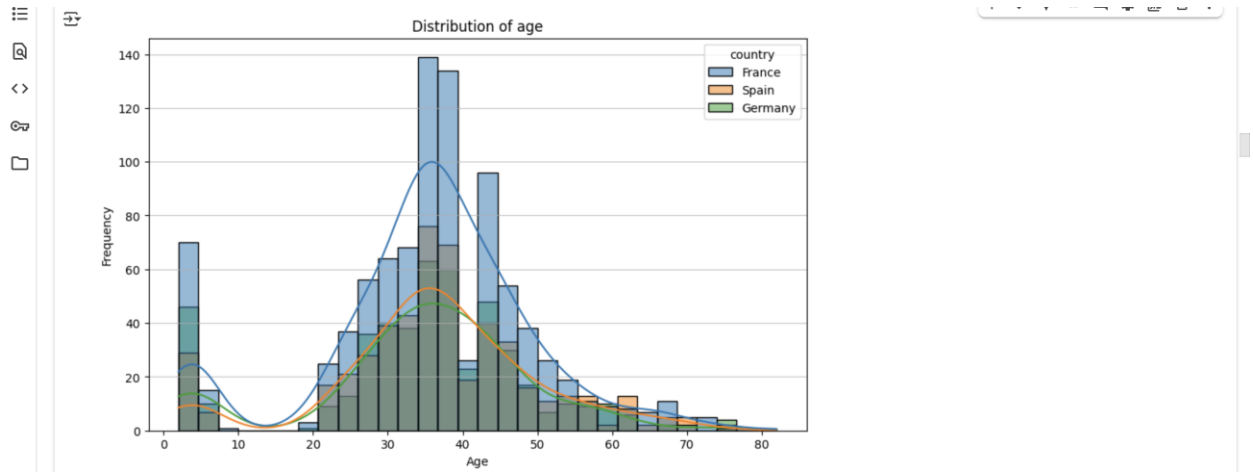
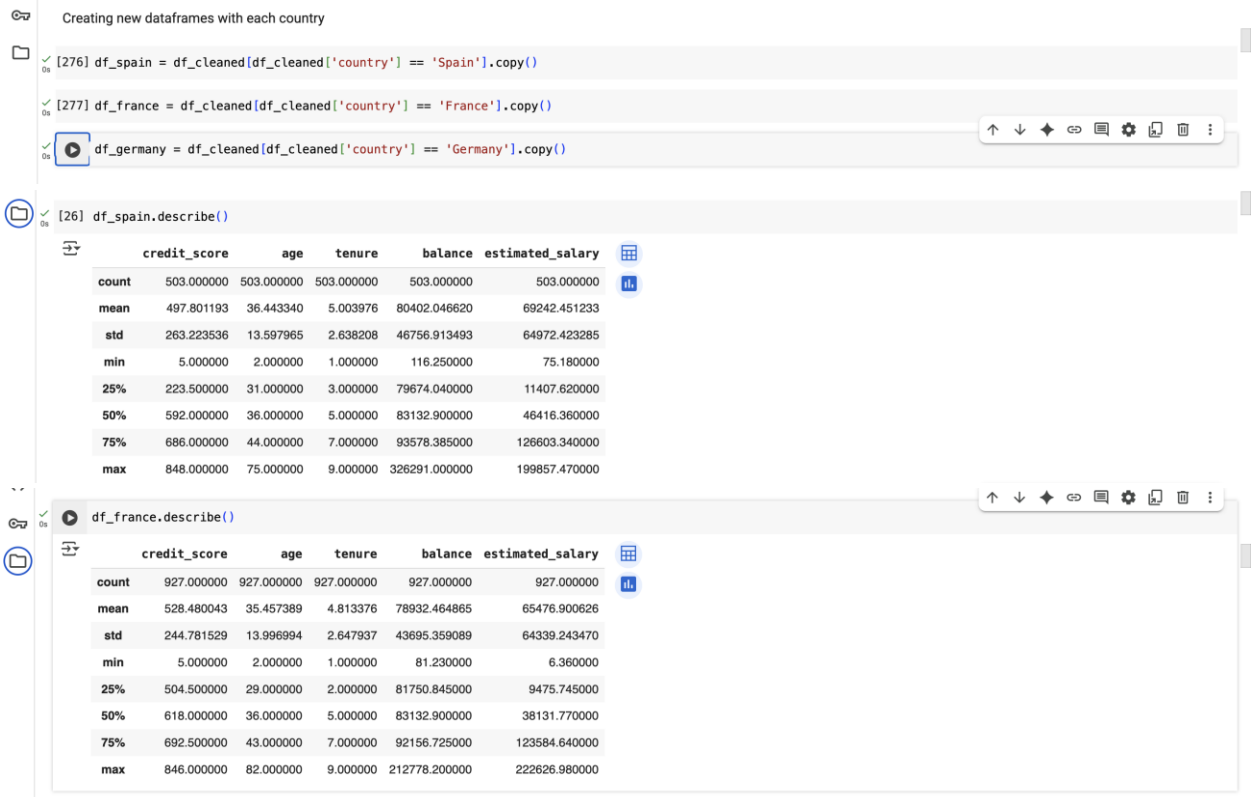


Figure 30: *Age* after cleaning

The manager instructed that the data be analysed separately by country in addition to the collective analysis. New data frames that have been created for each country are shown in Figure 31.



Creating new dataframes with each country

```
[276] df_spain = df_cleaned[df_cleaned['country'] == 'Spain'].copy()
[277] df_france = df_cleaned[df_cleaned['country'] == 'France'].copy()
      df_germany = df_cleaned[df_cleaned['country'] == 'Germany'].copy()
```

```
[26] df_spain.describe()
```

	credit_score	age	tenure	balance	estimated_salary
count	503.000000	503.000000	503.000000	503.000000	503.000000
mean	497.801193	36.443340	5.003976	80402.046620	69242.451233
std	263.223536	13.597965	2.638208	46756.913493	64972.423285
min	5.000000	2.000000	1.000000	116.250000	75.180000
25%	223.500000	31.000000	3.000000	79674.040000	11407.620000
50%	592.000000	36.000000	5.000000	83132.900000	46416.360000
75%	686.000000	44.000000	7.000000	93578.385000	126603.340000
max	848.000000	75.000000	9.000000	326291.000000	199857.470000

```
df_france.describe()
```

	credit_score	age	tenure	balance	estimated_salary
count	927.000000	927.000000	927.000000	927.000000	927.000000
mean	528.480043	35.457389	4.813376	78932.464865	65476.900626
std	244.781529	13.996994	2.647937	43695.359089	64339.243470
min	5.000000	2.000000	1.000000	81.230000	6.360000
25%	504.500000	29.000000	2.000000	81750.845000	9475.745000
50%	618.000000	36.000000	5.000000	83132.900000	38131.770000
75%	692.500000	43.000000	7.000000	92156.725000	123584.640000
max	846.000000	82.000000	9.000000	212778.200000	222626.980000

```
[27] df_germany.describe()
```

	credit_score	age	tenure	balance	estimated_salary
count	482.000000	482.000000	482.000000	482.000000	482.000000
mean	552.217842	34.520747	4.794606	71917.159378	70899.157199
std	235.594785	14.406649	2.701607	59480.122877	67205.624012
min	7.000000	2.000000	1.000000	19.500000	32.900000
25%	521.500000	29.000000	2.000000	12613.750000	9843.240000
50%	634.000000	36.000000	5.000000	77303.935000	47261.700000
75%	713.750000	43.000000	7.000000	126796.450000	131804.692500
max	849.000000	75.000000	9.000000	214346.960000	199761.290000

Figure 31: Creating new data frames

Regression machine learning models

The multiple linear regression model

The first type of regression model tested was the multiple linear regression model, collectively across all three countries and individually within Spain, France and Germany. The method followed throughout the study is the following: every regression model is tested in aggregate first, then independently by *country*. A scatter plot belongs to each regression model to represent how the model's prediction compares to the actual credit scores. A multiple linear regression model is used if the dataset has multiple independent variables to influence the one dependent variable, which is the *credit score* in this scenario. Independent variables are *balance*, and *estimated salary*. The dataset is divided into training and testing sets with a 70-30% ratio which is then fitted to the model. The model works by removing features where the p- values are below the 0.05 significance level. Mean squared error and R-squared are chosen performance measures; the results will be compared at the end of the study to determine the most accurate model.
The multiple linear regression model is based on a linear relationship between *credit score* and the independent variables included in the data set.
The following pictures show how the regression model performs with the collective data (Figure 32, 33), within Spain (Figure 34, 35), France (Figure 36, 37), and within Germany (Figure 38, 39).

Figure 32: Multiple linear regression model

```
[284] mse = mean_squared_error(y_test, y_pred)
      r2 = r2_score(y_test, y_pred)

      print(f"Mean Squared Error on Test Set: {mse:.2f}")
      print(f"R-squared on Test Set: {r2:.2f}")

      Mean Squared Error on Test Set: 63366.84
      R-squared on Test Set: -0.01

      plt.figure(figsize=(8, 6))
      plt.scatter(y_test, y_pred, color='skyblue', label='Actual vs Predicted')
      plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', lw=2, label='Ideal Prediction')
      plt.xlabel('Actual Credit Score')
      plt.ylabel('Predicted Credit Score')
      plt.title('Actual vs Predicted Credit Scores (Multiple Linear Regression')
      plt.legend()
      plt.grid(True)
      plt.show()
```

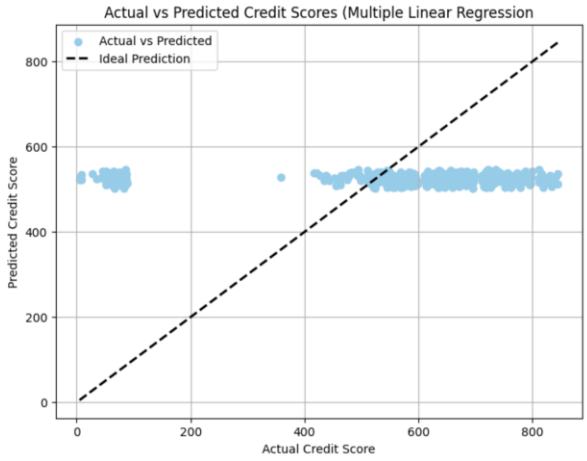


Figure 33: Multiple linear regression

Spain

```
Spain

[286] x=df_spain[['balance', 'estimated_salary']]
      y=df_spain['credit_score']

[287] x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=42)

[288] model = LinearRegression()
      model.fit(x_train, y_train)

      LinearRegression
      LinearRegression()

[289] y_pred=model.predict(x_test)
      x_train = sm.add_constant(x_train)
      x_optimal = x_train

      while True:
          regressor_OLS = sm.OLS(y_train, x_optimal).fit()
          p_values = regressor_OLS.pvalues
          if p_values.max() > 0.05:
              removed_feature = p_values.idxmax()
              x_optimal = x_optimal.drop(removed_feature, axis=1)
          else:
              break
```

Figure 34: Multiple linear regression Spain code

```
[291] mse = mean_squared_error(y_test, y_pred)
      r2 = r2_score(y_test, y_pred)

      print(f"Mean Squared Error on Test Set: {mse:.2f}")
      print(f"R-squared on Test Set: {r2:.2f}")

Mean Squared Error on Test Set: 65492.97
R-squared on Test Set: -0.02
```

```
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred, color='skyblue', label='Actual vs Predicted')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', lw=2, label='Ideal Prediction')
plt.xlabel('Actual Credit Score')
plt.ylabel('Predicted Credit Score')
plt.title('Actual vs Predicted Credit Scores (Multiple Linear Regression Spain')
plt.legend()
plt.grid(True)
plt.show()
```

Figure 35: Multiple linear regression Spain

France

```
France

[293] x=df_france[['balance', 'estimated_salary']]
      y=df_france['credit_score']

[294] x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=42)

[295] model = LinearRegression()
      model.fit(x_train, y_train)

▾ LinearRegression
LinearRegression()

[296] y_pred=model.predict(x_test)
      x_train = sm.add_constant(x_train)
      x_optimal = x_train
```

```
while True:
    regressor_OLS = sm.OLS(y_train, x_optimal).fit()
    p_values = regressor_OLS.pvalues
    if p_values.max() > 0.05:
        removed_feature = p_values.idxmax()
        x_optimal = x_optimal.drop(removed_feature, axis=1)
    else:
        break
```

Figure 36: Multiple linear regression France code

```
[298] mse = mean_squared_error(y_test, y_pred)
      r2 = r2_score(y_test, y_pred)

      print(f"Mean Squared Error on Test Set: {mse:.2f}")
      print(f"R-squared on Test Set: {r2:.2f}")

      Mean Squared Error on Test Set: 62698.17
      R-squared on Test Set: -0.01
```

```
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred, color='skyblue', label='Actual vs Predicted')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', lw=2, label='Ideal Prediction')
plt.xlabel('Actual Credit Score')
plt.ylabel('Predicted Credit Score')
plt.title('Actual vs Predicted Credit Scores (Multiple Linear Regression France')
plt.legend()
plt.grid(True)
plt.show()
```

Figure 37: Multiple linear regression France

Germany

Germany

```
[300] x=df_germany[['balance', 'estimated_salary']]
      y=df_germany['credit_score']
```

```
[301] x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=42)
```

```
[302] model = LinearRegression()
      model.fit(x_train, y_train)

      LinearRegression
      LinearRegression()
```

```
[303] y_pred=model.predict(x_test)
      x_train = sm.add_constant(x_train)
      x_optimal = x_train
```

```
while True:
    regressor_OLS = sm.OLS(y_train, x_optimal).fit()
    p_values = regressor_OLS.pvalues
    if p_values.max() > 0.05:
        removed_feature = p_values.idxmax()
        x_optimal = x_optimal.drop(removed_feature, axis=1)
    else:
        break
```

Figure 38: Multiple linear regression Germany code

```
[305] mse = mean_squared_error(y_test, y_pred)
      r2 = r2_score(y_test, y_pred)

      print(f"Mean Squared Error on Test Set: {mse:.2f}")
      print(f"R-squared on Test Set: {r2:.2f}")

Mean Squared Error on Test Set: 53704.09
R-squared on Test Set: -0.00
```

```
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred, color='skyblue', label='Actual vs Predicted')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', lw=2, label='Ideal Prediction')
plt.xlabel('Actual Credit Score')
plt.ylabel('Predicted Credit Score')
plt.title('Actual vs Predicted Credit Scores (Multiple Linear Regression Germany')
plt.legend()
plt.grid(True)
plt.show()
```

Figure 39: Multiple linear regression Germany

The polynomial regression model

As an extension of linear regression, the polynomial regression model could be used if the data does not follow a straight line and adding higher degrees will help the model to find non-linear patterns in the data set. The execution of this regression model shown collectively in Figures 40 and 41, for Spain in Figures 42 and 43, for France in Figures 44, 45, and for Germany in Figures 46 and 47.

Polynomial regression

```
[307] degree = 2
      model2 = make_pipeline(PolynomialFeatures(degree), LinearRegression())
      x =df_cleaned[['balance', 'estimated_salary','tenure','age']]
      y =df_cleaned['credit_score']
      x_train_poly, x_test_poly, y_train_poly, y_test_poly = train_test_split(x, y, test_size=0.3, random_state=42)
      model2.fit(x_train_poly, y_train_poly)
```

Pipeline
- PolynomialFeatures
- LinearRegression

```
y_pred_poly=model2.predict(x_test_poly)
mse_poly = mean_squared_error(y_test_poly, y_pred_poly)
r2_poly = r2_score(y_test_poly, y_pred_poly)

print(f"Mean Squared Error (Polynomial Regression) on Test Set: {mse_poly:.2f}")
print(f"R-squared (Polynomial Regression) on Test Set: {r2_poly:.2f}")
```

```
Mean Squared Error (Polynomial Regression) on Test Set: 63306.28
R-squared (Polynomial Regression) on Test Set: -0.01
```

Figure 40: Polynomial regression

```
plt.figure(figsize=(8, 6))
plt.scatter(y_test_poly, y_pred_poly, color='lightcoral', label='Actual vs Predicted')
plt.plot([y_test_poly.min(), y_test_poly.max()], [y_test_poly.min(), y_test_poly.max()], 'k--', lw=2, label='Ideal Prediction')
plt.xlabel('Actual Credit Score')
plt.ylabel('Predicted Credit Score')
plt.title('Actual vs Predicted Credit Scores (Polynomial Regression)')
plt.legend()
plt.grid(True)
plt.show()
```

Figure 41: Polynomial regression

Spain

Figure 42: Polynomial regression Spain

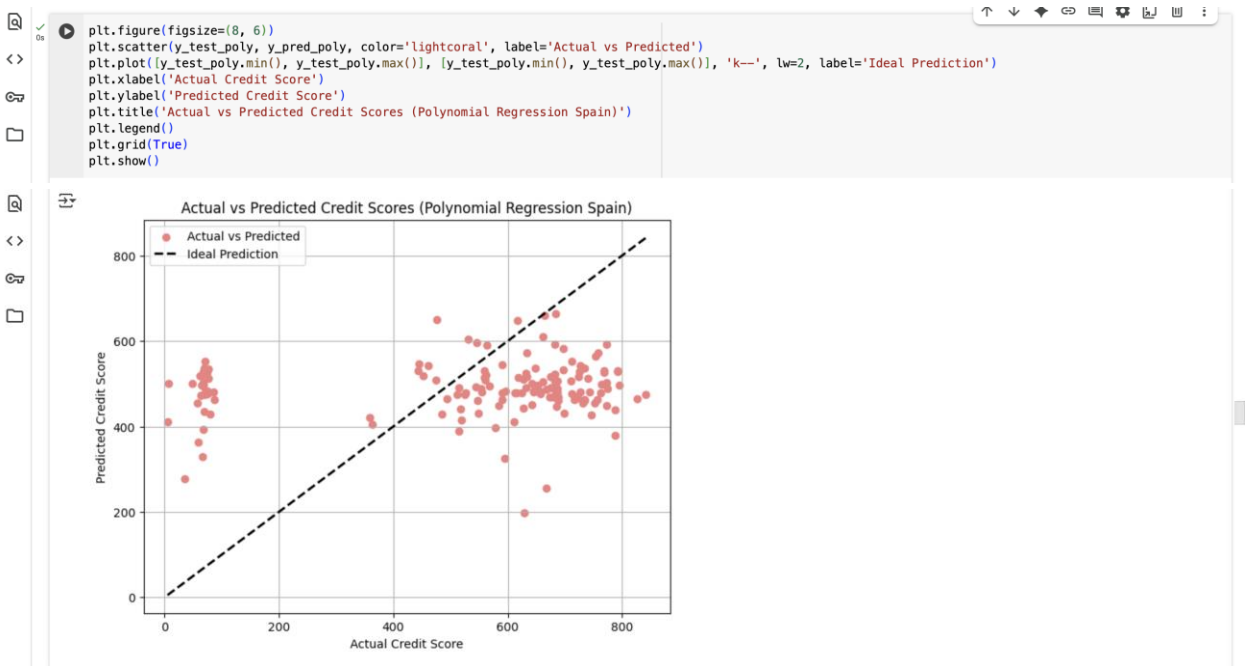


Figure 43: Polynomial regression Spain

France

```
France

[313] degree = 2
      model2 = make_pipeline(PolynomialFeatures(degree), LinearRegression())
      x =df_france[['balance', 'estimated_salary','tenure','age']]
      y =df_france['credit_score']
      x_train_poly, x_test_poly, y_train_poly, y_test_poly = train_test_split(x, y, test_size=0.3, random_state=42)
      model2.fit(x_train_poly, y_train_poly)

      Pipeline
        PolynomialFeatures
        LinearRegression

      y_pred_poly=model2.predict(x_test_poly)
      mse_poly = mean_squared_error(y_test_poly, y_pred_poly)
      r2_poly = r2_score(y_test_poly, y_pred_poly)

      print(f"Mean Squared Error (Polynomial Regression) on Test Set: {mse_poly:.2f}")
      print(f"R-squared (Polynomial Regression) on Test Set: {r2_poly:.2f}")

      Mean Squared Error (Polynomial Regression) on Test Set: 63329.02
      R-squared (Polynomial Regression) on Test Set: -0.02
```

Figure 44: Polynomial regression France

```
plt.figure(figsize=(8, 6))
plt.scatter(y_test_poly, y_pred_poly, color='lightcoral', label='Actual vs Predicted')
plt.plot([y_test_poly.min(), y_test_poly.max()], [y_test_poly.min(), y_test_poly.max()], 'k--', lw=2, label='Ideal Prediction')
plt.xlabel('Actual Credit Score')
plt.ylabel('Predicted Credit Score')
plt.title('Actual vs Predicted Credit Scores (Polynomial Regression France)')
plt.legend()
plt.grid(True)
plt.show()
```

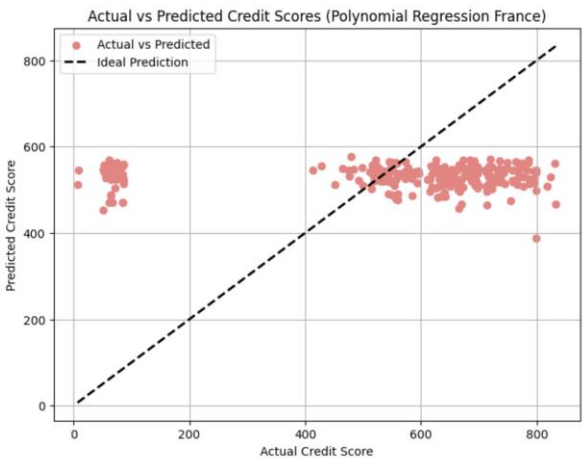


Figure 45: Polynomial regression France

Germany

```
Germany

[316] degree = 2
      model2 = make_pipeline(PolynomialFeatures(degree), LinearRegression())
      x =df_germany[['balance', 'estimated_salary','tenure','age']]
      y =df_germany['credit_score']
      x_train_poly, x_test_poly, y_train_poly, y_test_poly = train_test_split(x, y, test_size=0.3, random_state=42)
      model2.fit(x_train_poly, y_train_poly)

      Pipeline
        PolynomialFeatures
        LinearRegression

      y_pred_poly=model2.predict(x_test_poly)
      mse_poly = mean_squared_error(y_test_poly, y_pred_poly)
      r2_poly = r2_score(y_test_poly, y_pred_poly)

      print(f"Mean Squared Error (Polynomial Regression) on Test Set: {mse_poly:.2f}")
      print(f"R-squared (Polynomial Regression) on Test Set: {r2_poly:.2f}")

      Mean Squared Error (Polynomial Regression) on Test Set: 55729.63
      R-squared (Polynomial Regression) on Test Set: -0.04
```

Figure 46: Polynomial regression Germany

```
plt.figure(figsize=(8, 6))
plt.scatter(y_test_poly, y_pred_poly, color='lightcoral', label='Actual vs Predicted')
plt.plot([y_test_poly.min(), y_test_poly.max()], [y_test_poly.min(), y_test_poly.max()], 'k--', lw=2, label='Ideal Prediction')
plt.xlabel('Actual Credit Score')
plt.ylabel('Predicted Credit Score')
plt.title('Actual vs Predicted Credit Scores (Polynomial Regression Germany)')
plt.legend()
plt.grid(True)
plt.show()
```

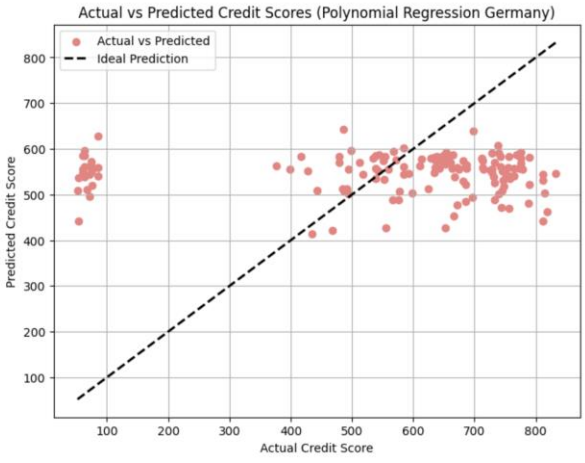


Figure 47: Polynomial regression Germany

The random forest regression model

The random forest regression model creates multiple decision trees, then averages the results from these decision trees to predict numerical values. This regression model is based on continuous data; therefore, categorical data need to be encoded into numerical data. Figures 48–67 demonstrate the random forest regression model. The *n_estimator* and *max_depth* are important hyperparameters to tune the model, where the *n_estimator* will determine the number of decision trees created, with the assigned maximum depth of each of these trees.

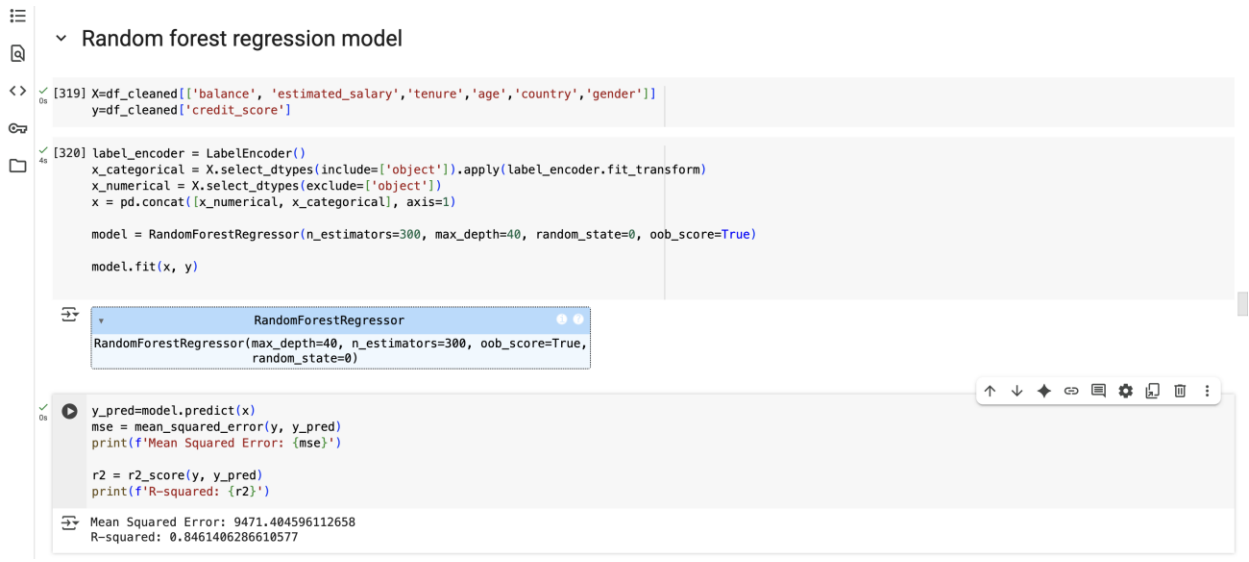


Figure 48: Random forest regression

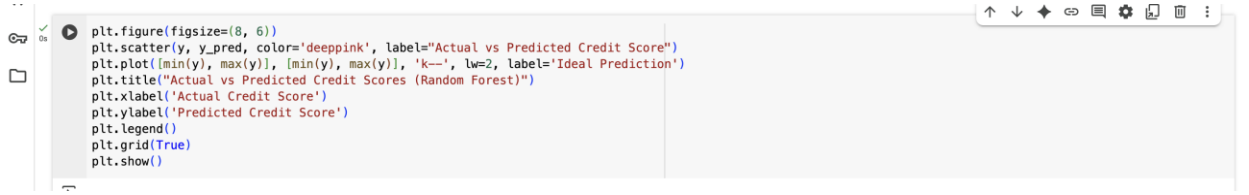


Figure 49: Random forest regression

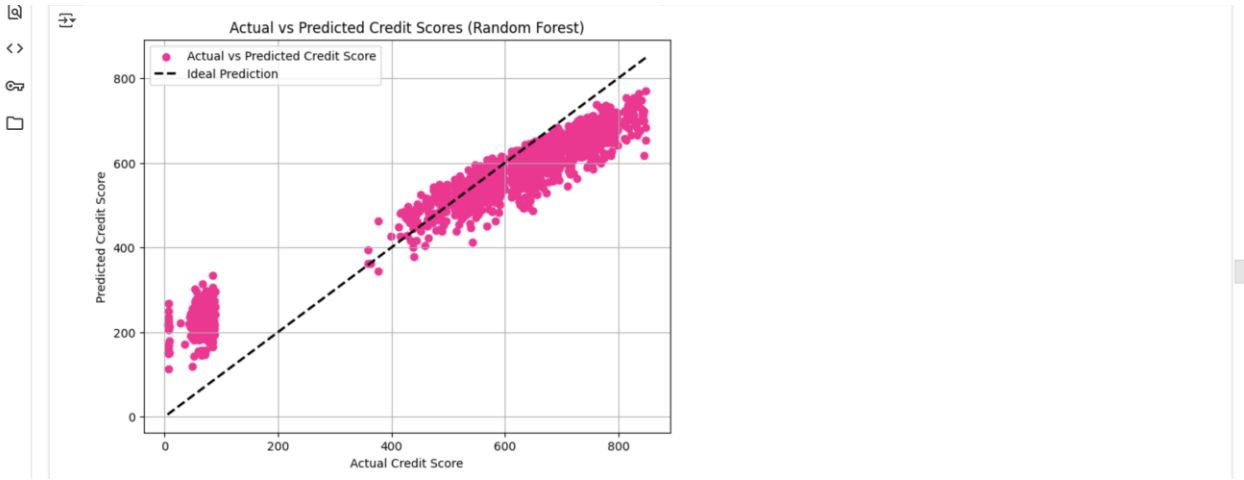


Figure 50: Random forest regression

```
plt.figure(figsize=(20, 10))
individual_tree = model.estimators_[0]
plot_tree(individual_tree, feature_names=x.columns, filled=True, rounded=True, fontsize=10, max_depth=4)
plt.title('One Decision Tree from Random Forest')
plt.show()
```

Figure 51: Random forest one decision tree

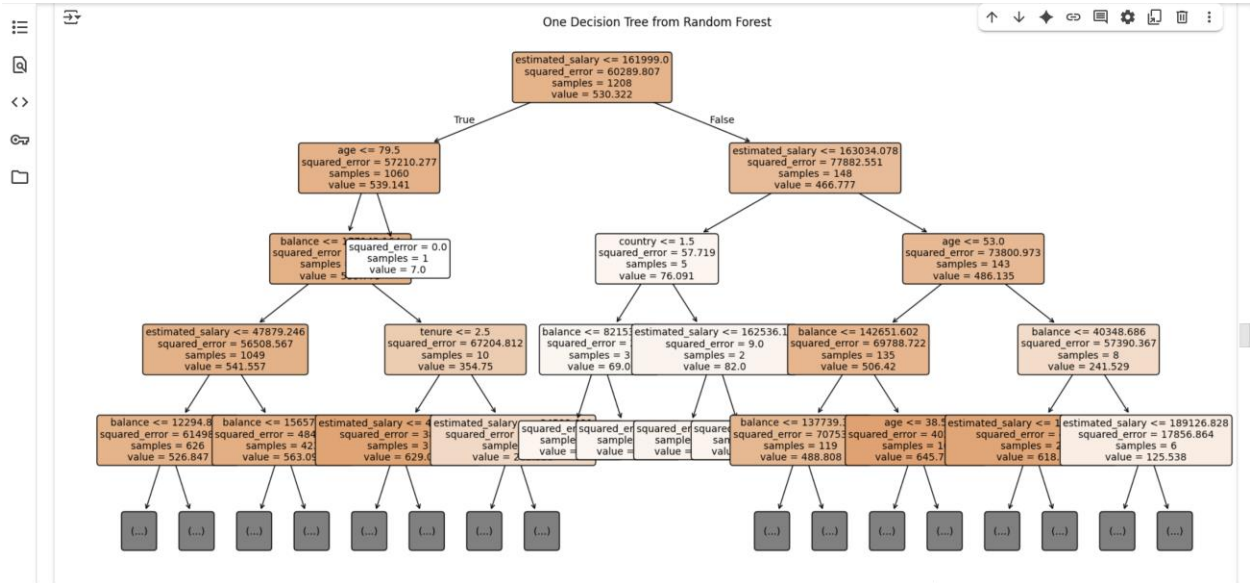


Figure 52: Random forest one decision tree

Spain

```
Spain

[324] X=df_spain[['balance', 'estimated_salary','tenure','age','country','gender']]
      y=df_spain['credit_score']

[325] label_encoder = LabelEncoder()
      x_categorical = X.select_dtypes(include=['object']).apply(label_encoder.fit_transform)
      x_numerical = X.select_dtypes(exclude=['object'])
      x = pd.concat([x_numerical, x_categorical], axis=1)

      model = RandomForestRegressor(n_estimators=400, max_depth=40,random_state=0, oob_score=True)

      model.fit(x, y)

      RandomForestRegressor
      RandomForestRegressor(max_depth=40, n_estimators=400, oob_score=True,
                            random_state=0)

      y_pred=model.predict(x)
      mse = mean_squared_error(y, y_pred)
      print(f'Mean Squared Error: {mse}')

      r2 = r2_score(y, y_pred)
      print(f'R-squared: {r2}')

      Mean Squared Error: 10721.87456291004
      R-squared: 0.8449450792709087
```

Figure 53: Random forest regression Spain

```
plt.figure(figsize=(8, 6))
plt.scatter(y, y_pred, color='deeppink', label="Actual vs Predicted Credit Score")
plt.plot([min(y), max(y)], [min(y), max(y)], 'k--', lw=2, label='Ideal Prediction')
plt.title("Actual vs Predicted Credit Scores Spain (Random Forest)")
plt.xlabel('Actual Credit Score')
plt.ylabel('Predicted Credit Score')
plt.legend()
plt.grid(True)
plt.show()
```

Figure 54: Random forest regression Spain

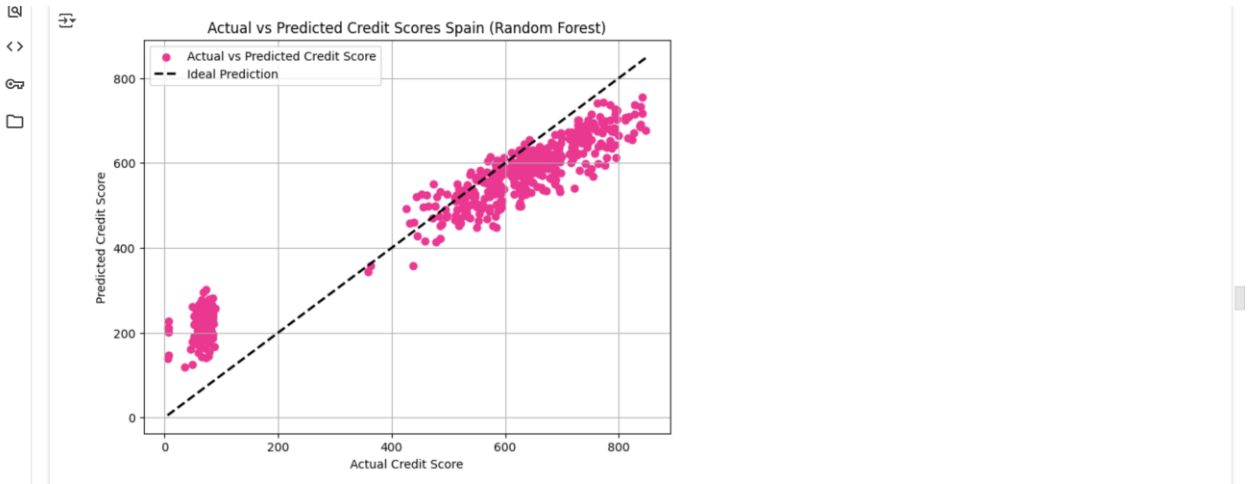


Figure 55: Random forest regression Spain

```
plt.figure(figsize=(20, 10))
individual_tree = model.estimators_[1]
plot_tree(individual_tree, feature_names=x.columns, filled=True, rounded=True, fontsize=10, max_depth=4)
plt.title('One Decision Tree from Random Forest Spain')
plt.show()
```

Figure 56: Random forest one decision tree Spain

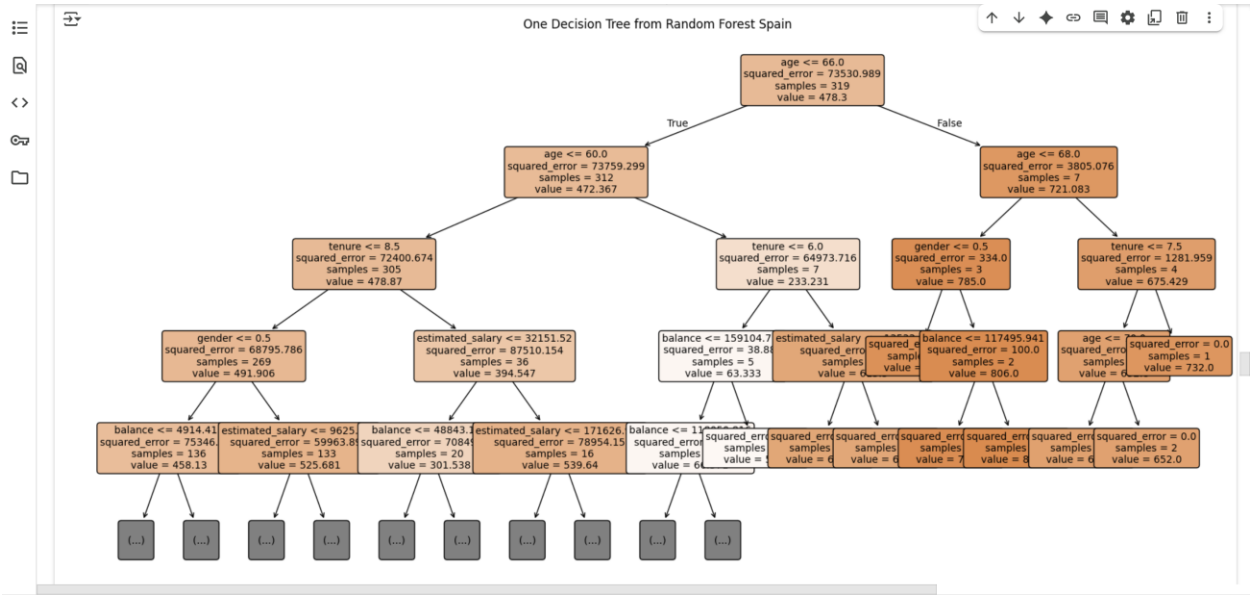


Figure 57: Random forest one decision tree Spain

France

```
France

[329] X=df_france[['balance', 'estimated_salary','tenure','age','country','gender']]
      y=df_france['credit_score']

[330] label_encoder = LabelEncoder()
      x_categorical = X.select_dtypes(include=['object']).apply(label_encoder.fit_transform)
      x_numerical = X.select_dtypes(exclude=['object'])
      x = pd.concat([x_numerical, x_categorical], axis=1)

      model = RandomForestRegressor(n_estimators=300, max_depth=50, random_state=0, oob_score=True)

      model.fit(x, y)

RandomForestRegressor(max_depth=50, n_estimators=300, oob_score=True,
                      random_state=0)

      y_pred=model.predict(x)
      mse = mean_squared_error(y, y_pred)
      print(f'Mean Squared Error: {mse}')

      r2 = r2_score(y, y_pred)
      print(f'R-squared: {r2}')

Mean Squared Error: 9377.95321398326
R-squared: 0.8433178504549335
```

Figure 58: Random forest regression France

```
plt.figure(figsize=(8, 6))
plt.scatter(y, y_pred, color='deeppink', label="Actual vs Predicted Credit Score")
plt.plot([min(y), max(y)], [min(y), max(y)], 'k--', lw=2, label='Ideal Prediction')
plt.title("Actual vs Predicted Credit Scores France (Random Forest)")
plt.xlabel('Actual Credit Score')
plt.ylabel('Predicted Credit Score')
plt.legend()
plt.grid(True)
plt.show()
```

Figure 59: Random forest regression France

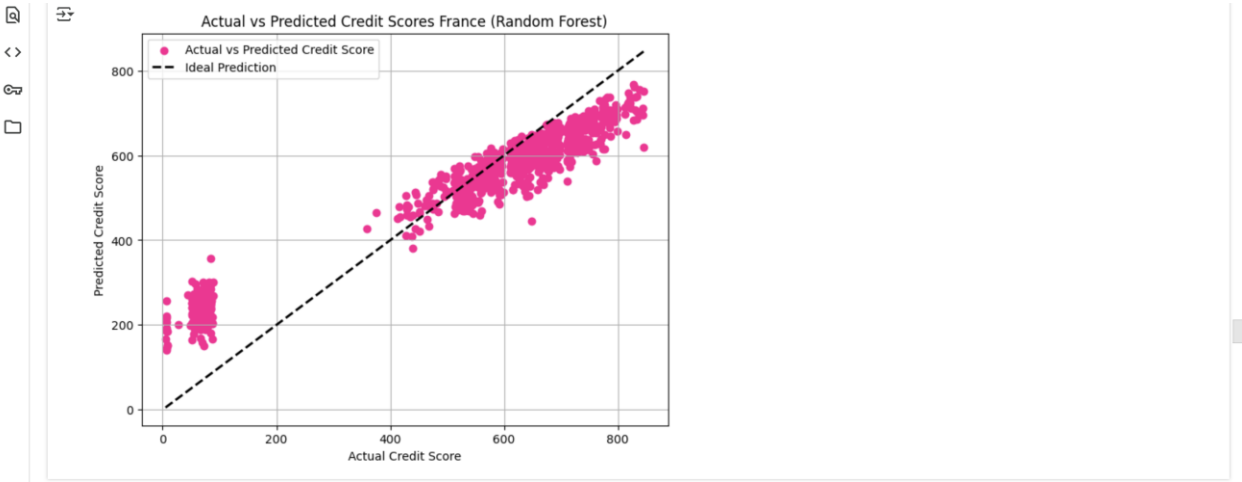


Figure 60: Random forest regression France

```
plt.figure(figsize=(20, 10))
individual_tree = model.estimators_[4]
plot_tree(individual_tree, feature_names=x.columns, filled=True, rounded=True, fontsize=10, max_depth=4)
plt.title('One Decision Tree from Random Forest France')
plt.show()
```

Figure 61: Random forest one decision tree France

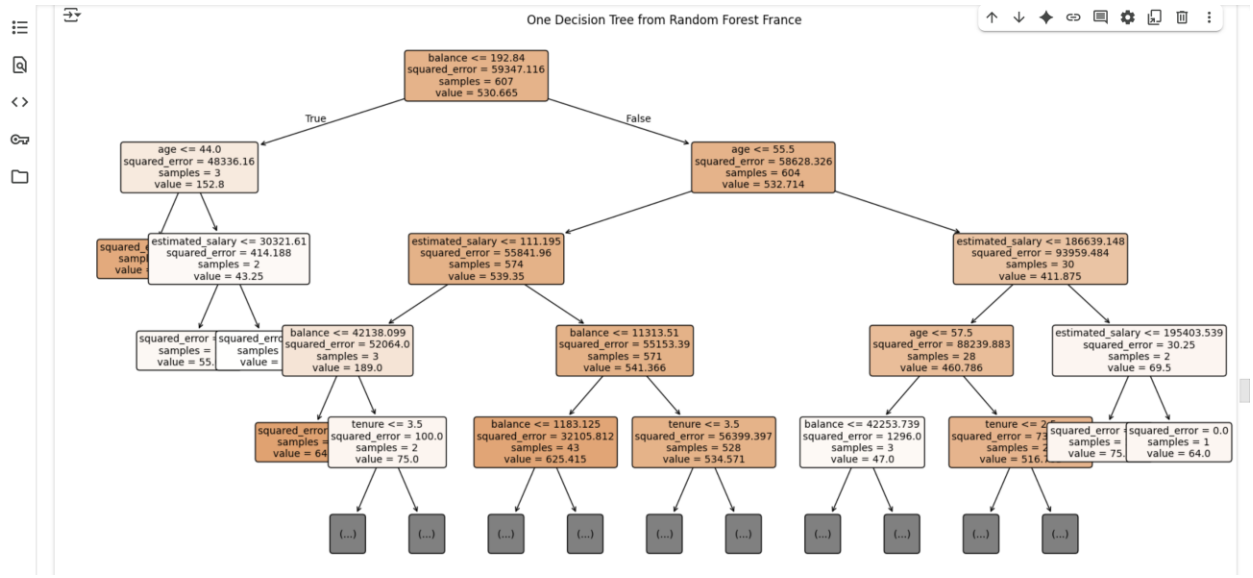


Figure 62: Random forest one decision tree France

Germany

Figure 63: Random forest regression Germany

```
plt.figure(figsize=(8, 6))
plt.scatter(y, y_pred, color='deeppink', label="Actual vs Predicted Credit Score")
plt.plot([min(y), max(y)], [min(y), max(y)], 'k--', lw=2, label='Ideal Prediction')
plt.title("Actual vs Predicted Credit Scores Germany (Random Forest)")
plt.xlabel('Actual Credit Score')
plt.ylabel('Predicted Credit Score')
plt.legend()
plt.grid(True)
plt.show()
```

Figure 64: Random forest regression Germany

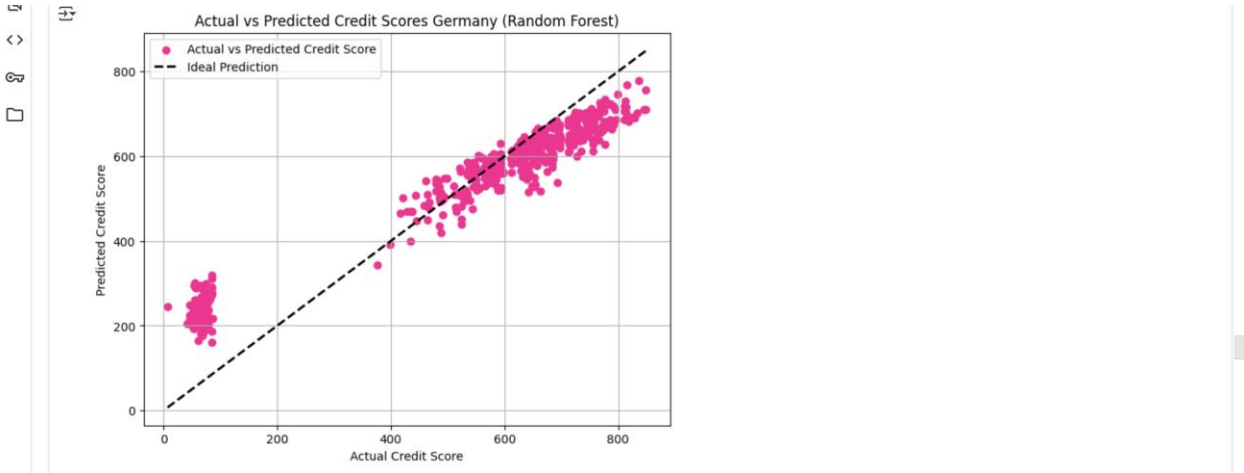


Figure 65: Random forest regression Germany

```
plt.figure(figsize=(20, 10))
individual_tree = model.estimators_[0]
plot_tree(individual_tree, feature_names=x.columns, filled=True, rounded=True, fontsize=10, max_depth=4)
plt.title('One Decision Tree from Random Forest Germany')
plt.show()
```

Figure 66: Random forest one decision tree Germany

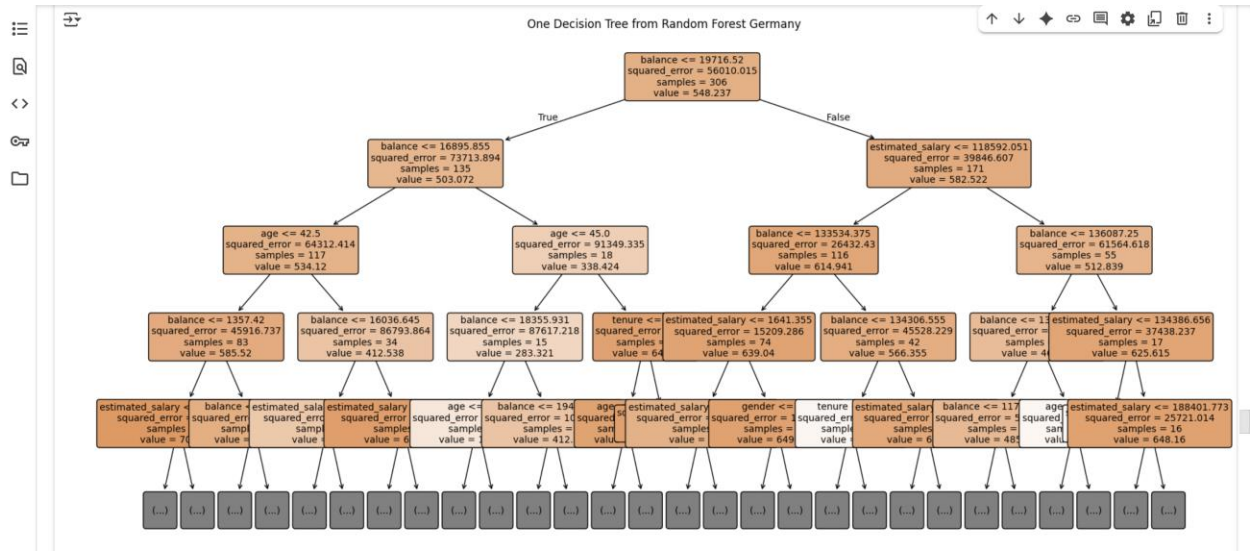


Figure 67: Random forest one decision tree Germany

The support vector regression model

A support vector machine (SVM) is a versatile model that can be used for a variety of tasks including classification, regression, and outlier detection and the model implementations include a tolerance parameter: epsilon. Support vector models use a mathematical function called a kernel that is used to transform each input point into a higher-dimensional space (Mishra, 2020).

The type of support vector machine used for regression analysis is called support vector regression or SVR. This algorithm fits data points to a hyperplane where epsilon represents the tolerance for error, gamma is the inverse of the influence of training samples, and C is the regularization parameter. The execution of this model shown in Figures 68-70, Figures 71-73 for Spain, Figures 74-76 for France, and Figures 77-79 for Germany.

SVR

```
[339] X=df_cleaned[['balance', 'estimated_salary','tenure','age','country','gender']]
      y=df_cleaned['credit_score']

      label_encoder = LabelEncoder()
      x_categorical = X.select_dtypes(include=['object']).apply(label_encoder.fit_transform)
      x_numerical = X.select_dtypes(exclude=['object'])
      x = pd.concat([x_numerical, x_categorical], axis=1)

      scaler_X = StandardScaler()
      scaler_y = StandardScaler()
      X_scaled = scaler_X.fit_transform(x)
      y_scaled = scaler_y.fit_transform(y.values.reshape(-1, 1)).ravel()
      svr_rbf= SVR(kernel='rbf', C=300, gamma=1, epsilon=0.1)
      svr_rbf.fit(X_scaled,y_scaled)
```

```
SVR(C=300, gamma=1)
```

```
Y_pred_scaled = svr_rbf.predict(X_scaled)
y_pred = scaler_y.inverse_transform(Y_pred_scaled.reshape(-1, 1))
mse = mean_squared_error(y, y_pred)
print(f'Mean Squared Error: {mse}')

r2 = r2_score(y, y_pred)
print(f'R-squared: {r2}')
```

```
Mean Squared Error: 9048.787308160856
R-squared: 0.8530058860345966
```

Figure 68: SVR

```
plt.figure(figsize=(8, 6))
plt.scatter(y, y_pred, color='plum', label="Actual vs Predicted Credit Score")
plt.plot([min(y), max(y)], [min(y), max(y)], color='red', linestyle='--', label='Ideal Prediction Line')
plt.title(" Actual vs Predicted Credit Scores (SVR)")
plt.xlabel('Actual Credit Score')
plt.ylabel('Predicted Credit Score')
plt.legend()
plt.grid(True)
plt.show()
```

Figure 69: SVR

Figure 70: SVR

Spain

Spain

```
[343] X=df_spain[['balance', 'estimated_salary','tenure','age','country','gender']]
      y=df_spain['credit_score']

      label_encoder = LabelEncoder()
      x_categorical = X.select_dtypes(include=['object']).apply(label_encoder.fit_transform)
      x_numerical = X.select_dtypes(exclude=['object'])
      x = pd.concat([x_numerical, x_categorical], axis=1)

      scaler_X = StandardScaler()
      scaler_y = StandardScaler()
      X_scaled = scaler_X.fit_transform(x)
      y_scaled = scaler_y.fit_transform(y.values.reshape(-1, 1)).ravel()
      svr_rbf= SVR(kernel='rbf', C=300, gamma=1, epsilon=0.1)
      svr_rbf.fit(X_scaled,y_scaled)
```

```
SVR(C=300, gamma=1)
```

```
Y_pred_scaled = svr_rbf.predict(X_scaled)
y_pred = scaler_y.inverse_transform(Y_pred_scaled.reshape(-1, 1))
mse = mean_squared_error(y, y_pred)
print(f'Mean Squared Error: {mse}')

r2 = r2_score(y, y_pred)
print(f'R-squared: {r2}')
```

```
Mean Squared Error: 6576.218884260731
R-squared: 0.9048976844661527
```

Figure 71: SVR Spain

```
plt.figure(figsize=(8, 6))
plt.scatter(y, y_pred, color='plum', label="Actual vs Predicted Credit Score")
plt.plot([min(y), max(y)], [min(y), max(y)], color='red', linestyle='--', label='Ideal Prediction Line')
plt.title(" Actual vs Predicted Credit Scores Spain (SVR)")
plt.xlabel('Actual Credit Score')
plt.ylabel('Predicted Credit Score')
plt.legend()
plt.grid(True)
plt.show()
```

Figure 72: SVR Spain

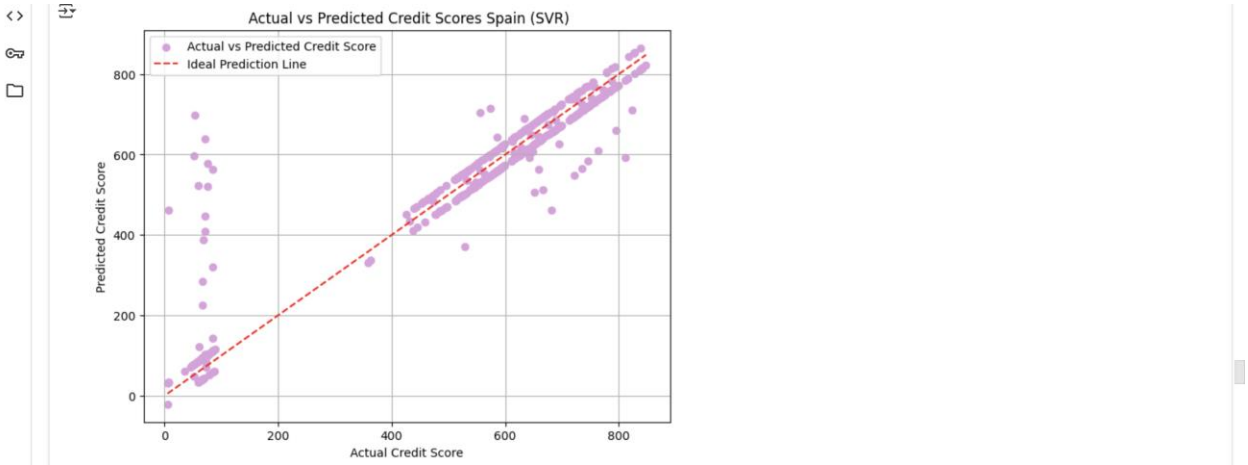


Figure 73: SVR Spain

France

France

```
[346] X=df_france[['balance', 'estimated_salary','tenure','age','country','gender']]
      y=df_france['credit_score']

      label_encoder = LabelEncoder()
      x_categorical = X.select_dtypes(include=['object']).apply(label_encoder.fit_transform)
      x_numerical = X.select_dtypes(exclude=['object'])
      x = pd.concat([x_numerical, x_categorical], axis=1)

      scaler_X = StandardScaler()
      scaler_y = StandardScaler()
      X_scaled = scaler_X.fit_transform(x)
      y_scaled = scaler_y.fit_transform(y.values.reshape(-1, 1)).ravel()
      svr_rbf= SVR(kernel='rbf', C=300, gamma=1, epsilon=0.1)
      svr_rbf.fit(X_scaled,y_scaled)
```

```
SVR(C=300, gamma=1)
```

```
Y_pred_scaled = svr_rbf.predict(X_scaled)
y_pred = scaler_y.inverse_transform(Y_pred_scaled.reshape(-1, 1))
mse = mean_squared_error(y, y_pred)
print(f'Mean Squared Error: {mse}')

r2 = r2_score(y, y_pred)
print(f'R-squared: {r2}')
```

```
Mean Squared Error: 12708.833261745873
R-squared: 0.7876671733986604
```

Figure 74: SVR France

```
plt.figure(figsize=(8, 6))
plt.scatter(y, y_pred, color='plum', label="Actual vs Predicted Credit Score")
plt.plot([min(y), max(y)], [min(y), max(y)], color='red', linestyle='--', label='Ideal Prediction Line')
plt.title(" Actual vs Predicted Credit Scores France (SVR)")
plt.xlabel('Actual Credit Score')
plt.ylabel('Predicted Credit Score')
plt.legend()
plt.grid(True)
plt.show()
```

Figure 75: SVR France

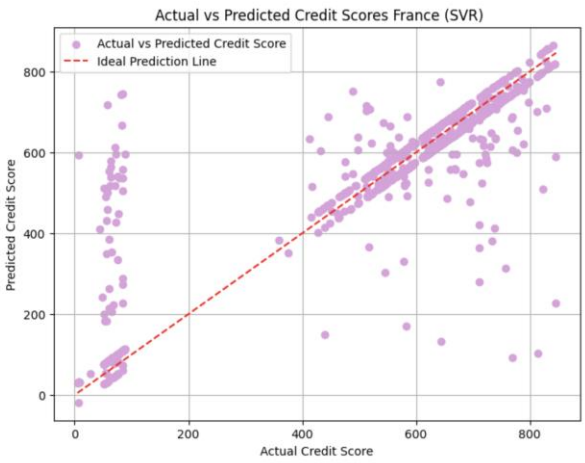


Figure 76: SVR France

Germany

```
Germany

[349] X=df_germany[['balance', 'estimated_salary','tenure','age','country','gender']]
      y=df_germany['credit_score']

      label_encoder = LabelEncoder()
      x_categorical = X.select_dtypes(include=['object']).apply(label_encoder.fit_transform)
      x_numerical = X.select_dtypes(exclude=['object'])
      x = pd.concat([x_numerical, x_categorical], axis=1)

      scaler_X = StandardScaler()
      scaler_y = StandardScaler()
      X_scaled = scaler_X.fit_transform(x)
      y_scaled = scaler_y.fit_transform(y.values.reshape(-1, 1)).ravel()
      svr_rbf= SVR(kernel='rbf', C=300, gamma=1, epsilon=0.1)
      svr_rbf.fit(X_scaled,y_scaled)

      SVR
      SVR(C=300, gamma=1)

      Y_pred_scaled = svr_rbf.predict(X_scaled)
      y_pred = scaler_y.inverse_transform(Y_pred_scaled.reshape(-1, 1))
      mse = mean_squared_error(y, y_pred)
      print(f'Mean Squared Error: {mse}')

      r2 = r2_score(y, y_pred)
      print(f'R-squared: {r2}')

      Mean Squared Error: 3474.4717620038714
      R-squared: 0.9372722942287244
```

Figure 77: SVR Germany

```
      plt.figure(figsize=(8, 6))
      plt.scatter(y, y_pred, color='plum', label="Actual vs Predicted Credit Score")
      plt.plot([min(y), max(y)], [min(y), max(y)], color='red', linestyle='--', label='Ideal Prediction Line')
      plt.title(" Actual vs Predicted Credit Scores Germany (SVR)")
      plt.xlabel('Actual Credit Score')
      plt.ylabel('Predicted Credit Score')
      plt.legend()
      plt.grid(True)
      plt.show()
```

Figure 78: SVR Germany

Figure 79: SVR Germany

The LASSO regression

The final regression model of the study is based on the least absolute shrinkage and selection operator (LASSO) technique. This model could be efficacious if linear regression did not perform well, and the model automatically selects the more important features from the independent variables. The hyperparameter *alpha* controls this feature selection, where a higher *alpha* number means fewer attributes are included in the model. The performance of the LASSO regression is displayed in Figures 80-82, in Figures 83-85 for Spain, Figures 86-88 for France, and Figures 89-91 for Germany.

Figure 80: LASSO

Figure 81: LASSO

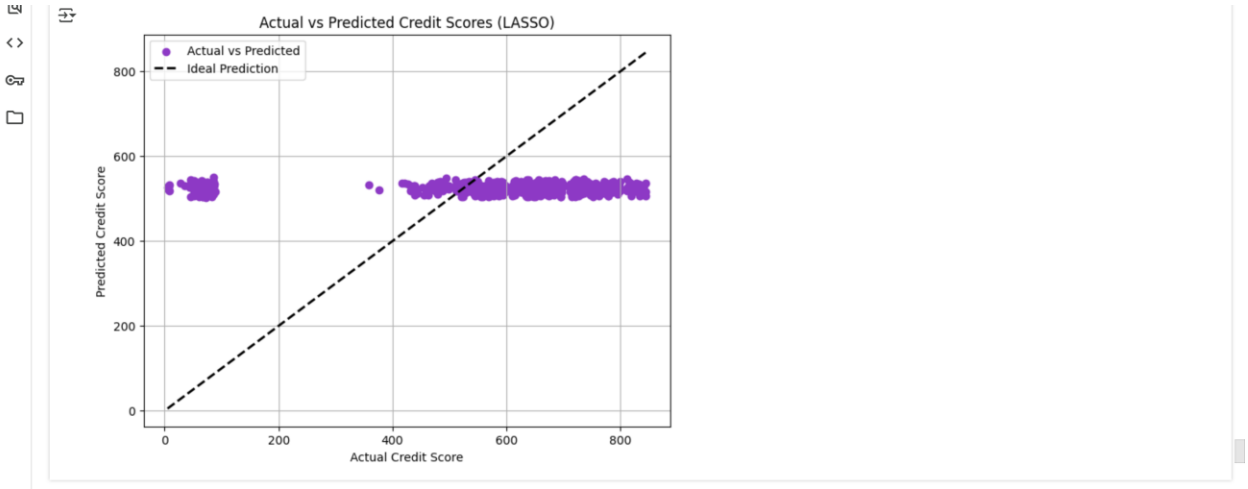


Figure 82: LASSO

Spain

```
Spain

[355] X = df_spain[['balance', 'estimated_salary','tenure','age','country','gender']]
      y = df_spain['credit_score']

      label_encoder = LabelEncoder()
      x_categorical = X.select_dtypes(include=['object']).apply(label_encoder.fit_transform)
      x_numerical = X.select_dtypes(exclude=['object'])
      x = pd.concat([x_numerical, x_categorical], axis=1)



      X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.4, random_state=42)


      lasso = LS(alpha=3000)

      lasso.fit(X_train, y_train)

      Lasso
      Lasso(alpha=3000)

      y_pred = lasso.predict(X_test)
      mse = mean_squared_error(y_test, y_pred)
      r2 = r2_score(y_test, y_pred)

      print(f"Mean Squared Error: {mse:.2f}")
      print(f"R-squared: {r2:.2f}")

      Mean Squared Error: 63468.03
      R-squared: -0.02
```

Figure 83: LASSO Spain

```
      plt.figure(figsize=(8, 6))
      plt.scatter(y_test, y_pred, color='darkorchid', label='Actual vs Predicted')
      plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', lw=2, label='Ideal Prediction')
      plt.xlabel('Actual Credit Score')
      plt.ylabel('Predicted Credit Score')
      plt.title('Actual vs Predicted Credit Scores Spain (LASSO)')
      plt.legend()
      plt.grid(True)
      plt.show()
```

Figure 84: LASSO Spain

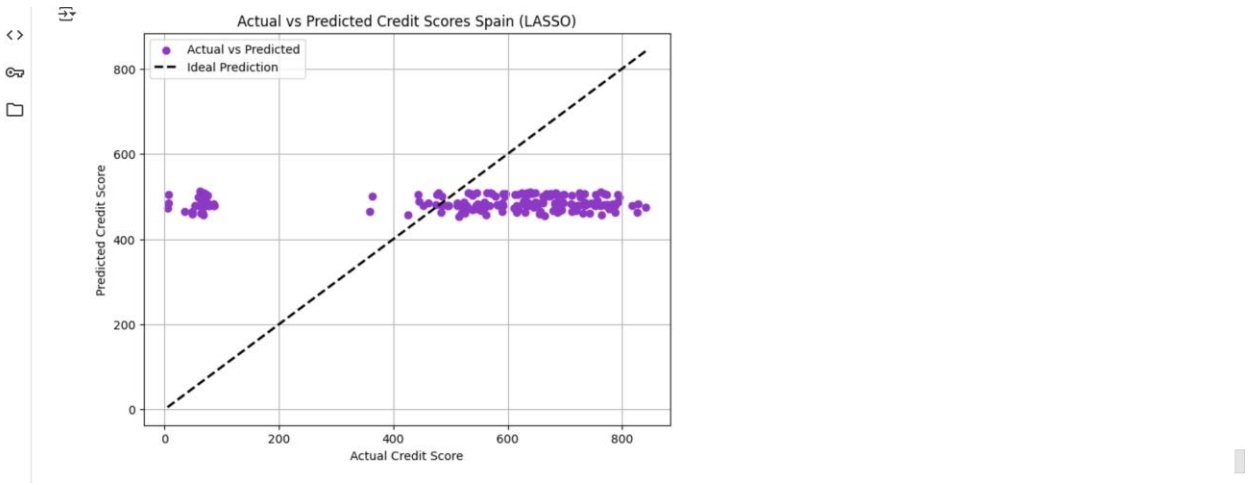


Figure 85: LASSO Spain

France

Figure 86: LASSO France

Figure 87: LASSO France

Figure 88: LASSO France

Germany

Figure 89: LASSO Germany

Figure 90: LASSO Germany

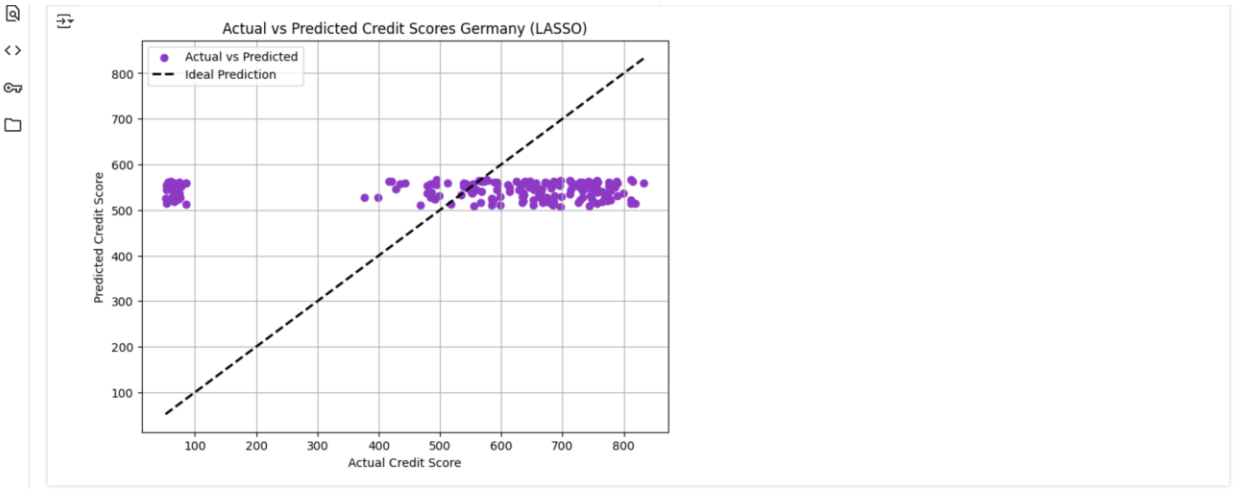


Figure 91: LASSO Germany

Task 2- The machine learning report

Machine learning models are essential in the financial sector for enhancing decision-making procedures, especially in loan approval and credit risk assessments (Rani and Gupta, 2024). The data analyst team at SMART banking plc has created this study to find the most accurate machine learning algorithm to predict future credit scores.

The provided dataset includes features such as *age*, *country, customer id*, *gender*, *balance*, *estimated salary, tenure,* and *credit score,* which is also the target variable (Figure 92). The *customer id* is irrelevant for this study. It does not provide any useful information, along with the column: *Unnamed: 8,* which includes empty rows only, therefore these columns will be removed. The *age* ranges from 2 to 82 with a 35.5 average. The three countries included in the analysis are Spain, France and Germany. For these countries, the credit score column has an average of 526, a minimum of 5, and a maximum of 849. Figure 93 also shows that the average balance is € 94, 660.32. Collectively the salary for the three countries stands at € 67, 223.41. The tenure refers to the number of years required to repay a loan, ranging from 1 to 9 years in this case, with the mean being around 5 years.

Data preprocessing

```
[254] df = pd.read_csv('SMART Banking data_4035.csv')
      df.head(10)
```

	customer_id	credit_score	country	gender	age	tenure	balance	estimated_salary	Unnamed: 8
0	1563462	619.0	France	Female	42.0	2.0	NaN	11348.88	NaN
1	15647311	68.0	Spain	Female	41.0	1.0	8387.86	112542.58	NaN
2	1561934	52.0	France	Female	42.0	8.0	15966.80	113931.57	NaN
3	1571354	699.0	France	Female	39.0	1.0	NaN	93826.63	NaN
4	15737888	85.0	Spain	Female	43.0	2.0	12551.82	7984.10	NaN
5	1557412	645.0	Spain	Male	44.0	8.0	113755.78	149756.71	NaN
6	15592531	822.0	France	Male	5.0	7.0	NaN	162.80	NaN
7	15656148	376.0	Germany	Female	29.0	4.0	11546.74	119346.88	NaN
8	15792365	51.0	France	Male	44.0	4.0	14251.70	7494.50	NaN
9	15592389	684.0	France	Male	27.0	2.0	13463.88	71725.73	NaN

Next steps: Generate code with df | View recommended plots | New interactive sheet

Figure 92: First ten rows of the dataset

```
df.describe()
```

	customer_id	credit_score	age	tenure	balance	estimated_salary	Unnamed: 8
count	2.020000e+03	2009.000000	2014.000000	1938.000000	1.309000e+03	2020.000000	0.0
mean	9.593544e+06	525.802887	35.508937	4.858101	9.466032e+04	67223.410317	NaN
std	7.142025e+06	248.621152	14.029019	2.655002	4.147254e+05	64795.973787	NaN
min	1.567000e+03	5.000000	2.000000	1.000000	1.950000e+01	6.360000	NaN
25%	1.569085e+06	486.000000	29.000000	2.000000	1.282972e+04	9985.462500	NaN
50%	1.559484e+07	618.000000	36.000000	5.000000	8.276742e+04	44250.070000	NaN
75%	1.571238e+07	694.000000	43.000000	7.000000	1.295557e+05	124809.885000	NaN
max	1.581536e+07	849.000000	82.000000	9.000000	1.431293e+07	222626.980000	NaN

Figure 93: Exploratory data analysis

Figure 93 illustrates that the banking data set contains 2020 rows among other pertinent information about the data.

The *balance* and the *estimated salary* are the two most important features because *credit score* prediction is requested to be based on these, although some regression models include more of the features to improve the model's accuracy.

Data preprocessing

Once the data set is loaded, the next step is to inspect and clean the data as the most important part of preprocessing. The data cleaning includes removing or replacing all the missing values and removing outliers. Without data cleaning, the regression models give biased results and weak performances. The missing data continues to plague data analysis methods even as these methods gain more sophistication (Larose, 2015).

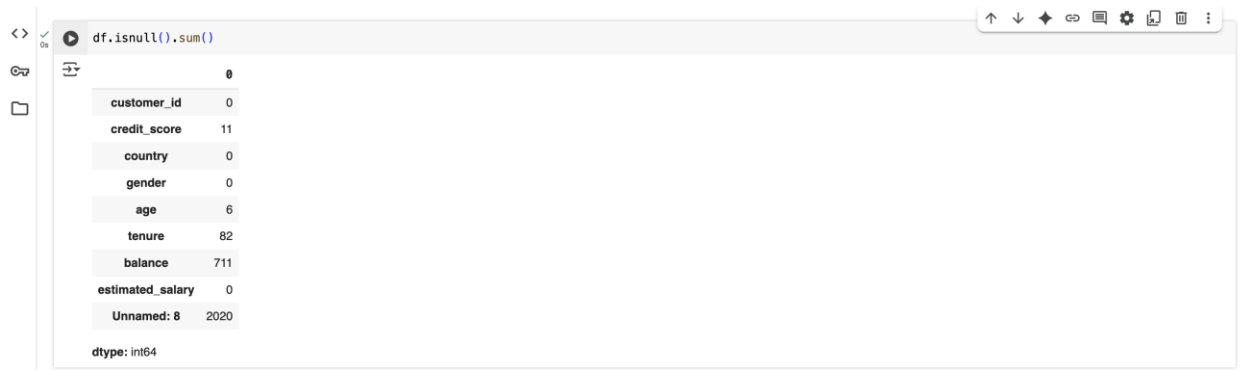


Figure 94: Missing values

Figure 94 shows the number of missing values in each column. There are multiple different ways to handle missing data; what is: 11 rows in the *credit score*, 6 rows in *age,* and 82 in the *tenure* column. All these rows were removed along with the entire *customer id* and *Unnamed: 8* columns. Since these two columns are irrelevant for the analysis, their removal will not negatively impact the regression models (Figure 95). However, the *balance* feature has 711 missing values, which could be caused by the information being too sensitive for some individuals. Instead of removing them (what would not be wise as it is 35% of the 2020 rows), the missing values are replaced with the median value of the balances. This phenomenon occurs because the median is unaffected by outliers, unlike the mean for example. The detection of the outliers is the next essential part of

preprocessing, followed by the removal of all these extremely out of range values. Figure 96 illustrates the number of outliers in each column.

Figure 95: Dropping unnecessary columns and rows with missing values

Outliers in credit_score:

	credit_score	country	gender	age	tenure	balance	estimated_salary
1	68.0	Spain	Female	41.0	1.0	8387.86	112542.58
2	52.0	France	Female	42.0	8.0	15966.80	113931.57
4	85.0	Spain	Female	43.0	2.0	12551.82	7984.10
8	51.0	France	Male	44.0	4.0	14251.70	7494.50
22	51.0	Spain	Female	38.0	4.0	83132.90	118913.53
...	...	...	...	...	...	...	...
2010	87.0	Spain	Male	45.0	6.0	2226291.00	148684.81
2011	26.0	France	Male	24.0	6.0	1393791.00	54764.30
2013	63.0	Spain	Female	32.0	8.0	1116291.00	38555.46
2014	50.0	Spain	Female	38.0	4.0	1326871.00	118913.53
2016	84.0	France	Female	38.0	5.0	83132.90	187611.16

394 rows × 7 columns

Outliers in age:

	credit_score	country	gender	age	tenure	balance	estimated_salary
6	822.0	France	Male	5.0	7.0	83132.90	162.80
40	472.0	Spain	Male	4.0	4.0	83132.90	7154.22
58	511.0	Spain	Female	66.0	4.0	83132.90	1643.11
85	652.0	Spain	Female	75.0	1.0	83132.90	114675.75
87	729.0	France	Male	3.0	9.0	83132.90	151869.35
...	...	...	...	...	...	...	...
1964	8.0	France	Female	4.0	5.0	97764.41	9864.15
1980	554.0	France	Female	3.0	9.0	83132.90	432.30
1981	476.0	Spain	Female	69.0	1.0	1533.73	13426.34
1983	748.0	Spain	Female	4.0	4.0	83132.90	132368.47
1993	824.0	Germany	Male	6.0	8.0	13425.17	15346.16

221 rows × 7 columns

Outliers in tenure:

credit_score	country	gender	age	tenure	balance	estimated_salary

Outliers in balance:

	credit_score	country	gender	age	tenure	balance	estimated_salary
1996	534.0	France	Male	62.0	2.0	1135596.0	42763.12
1998	630.0	France	Male	43.0	5.0	1427823.0	29483.35
2007	612.0	Germany	Male	45.0	3.0	14312933.0	61327.26
2008	753.0	Germany	Male	58.0	1.0	1326291.0	1597.67
2009	540.0	Spain	Female	24.0	9.0	326291.0	2446.41
2010	87.0	Spain	Male	45.0	6.0	2226291.0	148684.81
2011	26.0	France	Male	24.0	6.0	1393791.0	54764.30
2012	735.0	France	Male	41.0	8.0	1739291.0	17286.17
2013	63.0	Spain	Female	32.0	8.0	1116291.0	38555.46
2014	50.0	Spain	Female	38.0	4.0	1326871.0	118913.53
2018	656.0	Germany	Female	36.0	2.0	1368152.0	1741.95

Outliers in estimated_salary:

credit_score	country	gender	age	tenure	balance	estimated_salary

Figure 96: Outliers

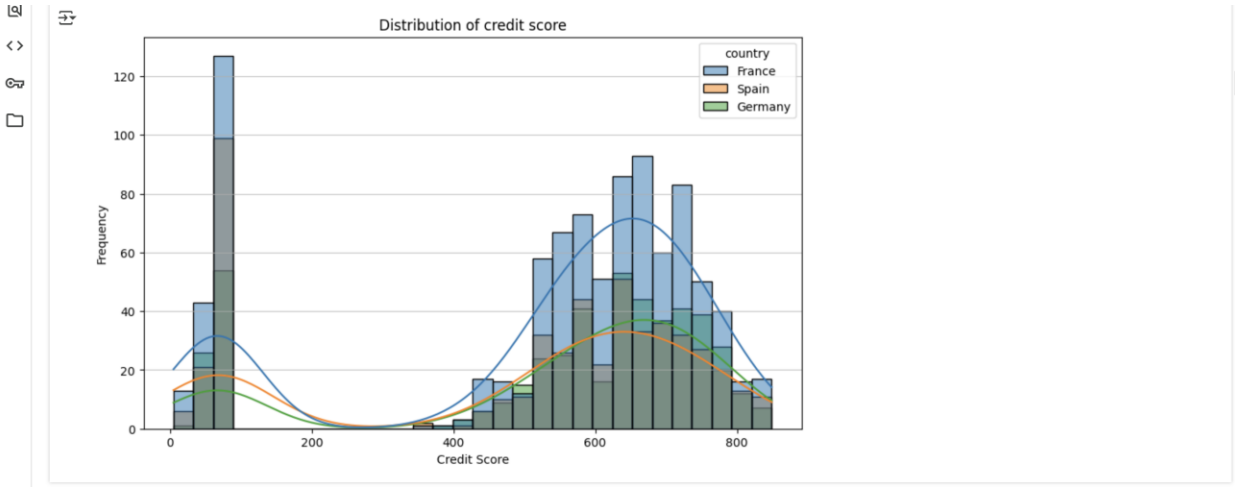


Figure 97: Distribution of credit score after cleaning

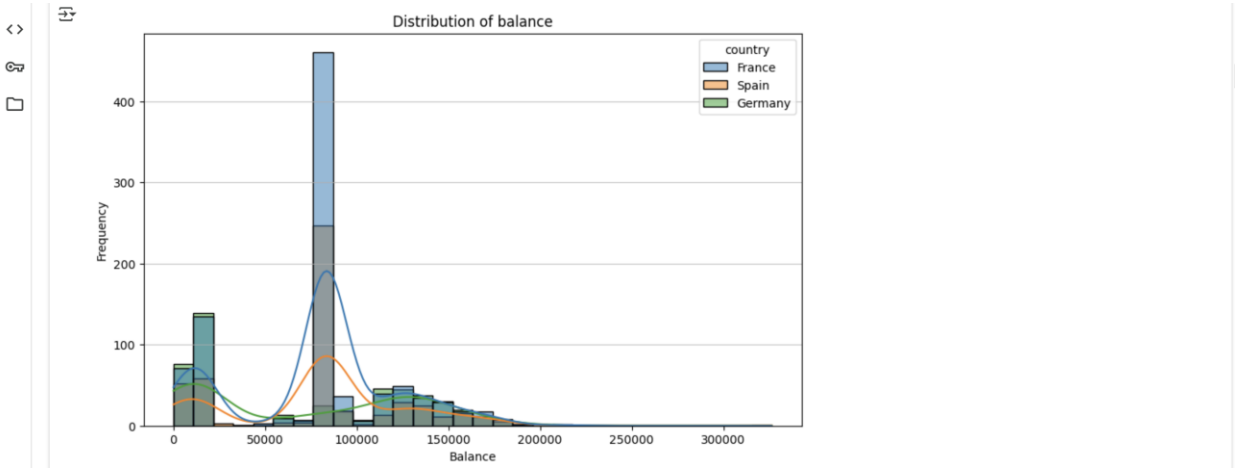


Figure 98: Distribution of balance after cleaning

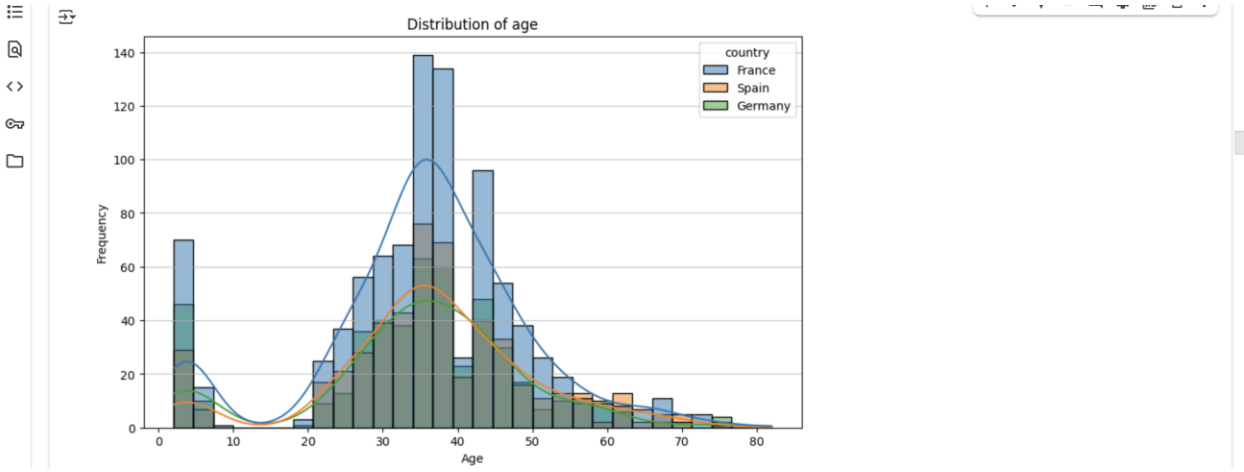


Figure 99: Distribution of age after cleaning

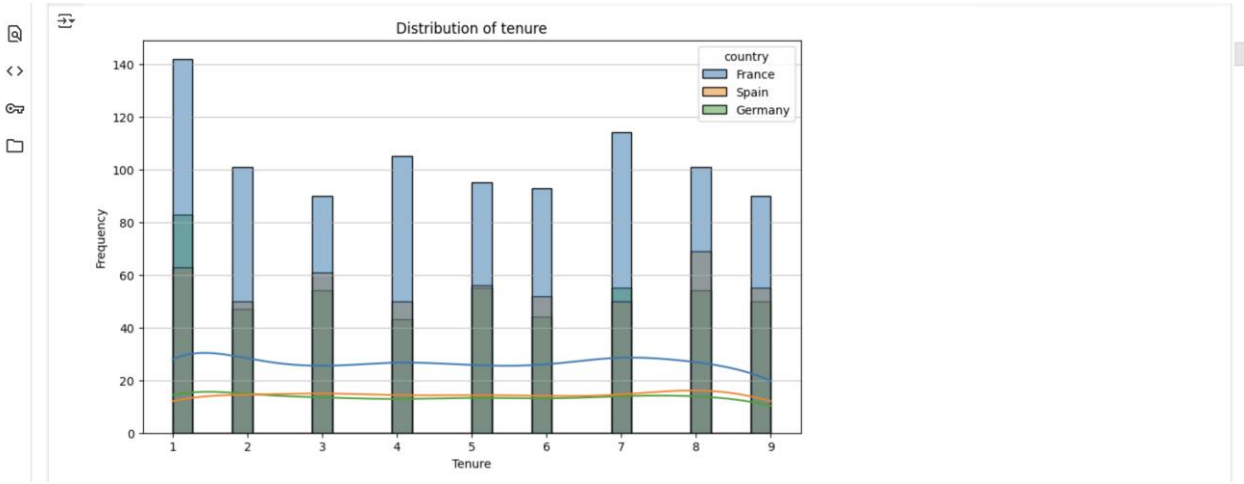


Figure 100: Distribution of tenure after cleaning

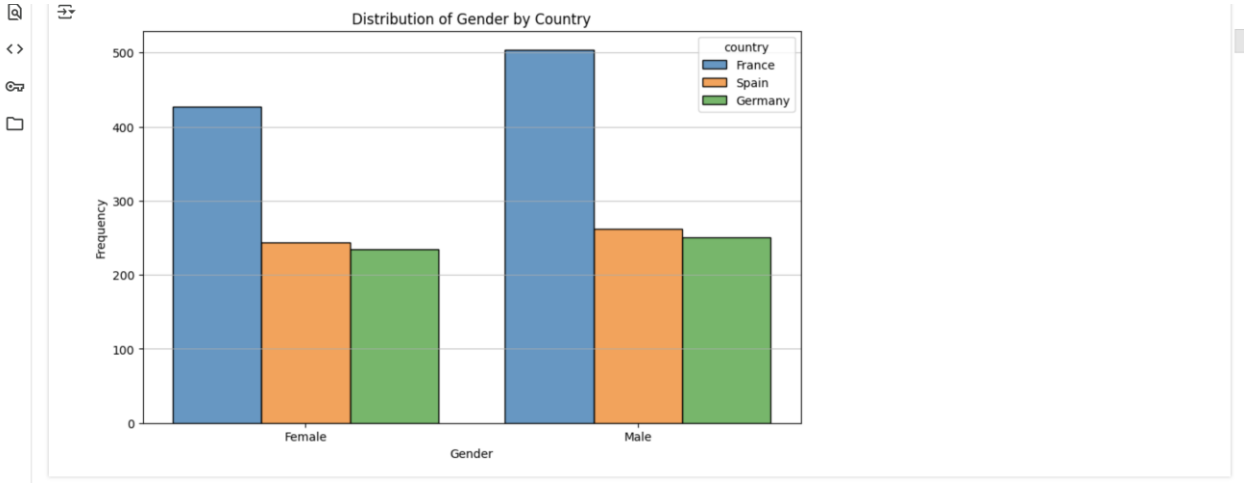
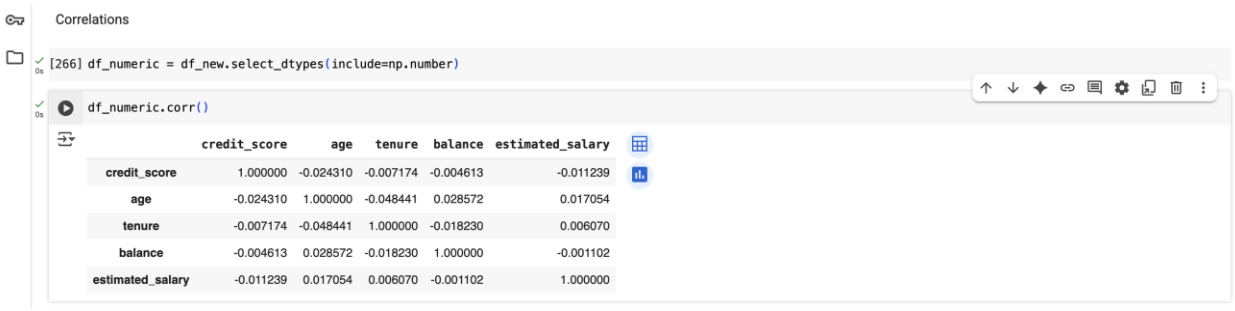


Figure 101: Distribution of gender by country

France has significantly higher frequency in each category compared to Spain and Germany. The distribution of each feature is thoroughly similar in the latter two countries.

Analyzing the" Distribution of credit score" chart (Figure 97) showed an immense gap in the data between the *credit score* values of 100 and 350. The first part of data points exhibit a direct proportion, while the second part follows the normal distribution curve. The same discontinuity is visible in the" Distribution of age" histogram (Figure 99) between the ages of 10 and 18, and a fall noticeable around the *age* 40. An investigation is recommended to find the reason behind this decrease in frequency. All three countries in question have the same distribution patterns. The frequency of the *balance* of 800 000 is extremely high (Figure 98) due to the high number of missing values which have been earlier replaced with the median value of the *balance*. The distribution of female and male genders is nearly equal for Spain and Germany; however, in France the number of males is marginally higher than females (Figure 101).

It is necessary to inspect the correlation between all numerical features to understand relationships in the data as well as help with decision making. The correlations between the variables are significantly low, close to 0, as seen in Figure 102. The heatmap helps to visualise this correlation (Figure 103).


Correlations

```
[266] df_numeric = df_new.select_dtypes(include=np.number)
```

```
df_numeric.corr()
```

	credit_score	age	tenure	balance	estimated_salary
credit_score	1.000000	-0.024310	-0.007174	-0.004613	-0.011239
age	-0.024310	1.000000	-0.048441	0.028572	0.017054
tenure	-0.007174	-0.048441	1.000000	-0.018230	0.006070
balance	-0.004613	0.028572	-0.018230	1.000000	-0.001102
estimated_salary	-0.011239	0.017054	0.006070	-0.001102	1.000000

Figure 102: Correlation

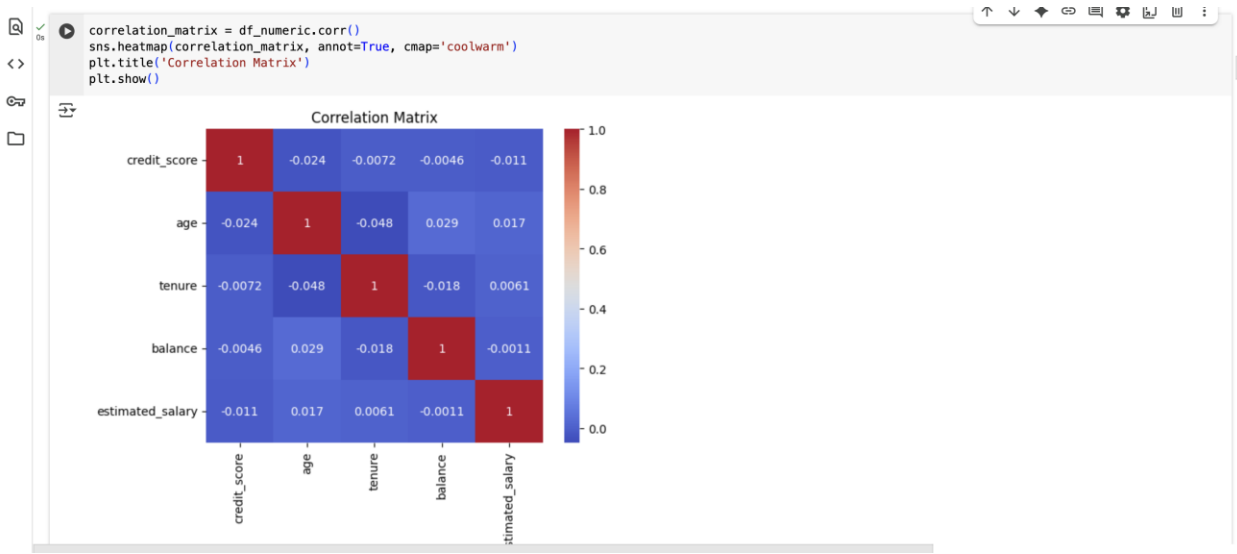


Figure 103: Correlation heatmap

Machine learning models

Machine learning is changing every aspect of our lives. Nowadays, algorithms accomplish tasks that only expert humans could perform, therefore this is the most exciting time to adopt this technology in finance (de Prado, 2018). This analysis tests five different machine learning regression models, first collectively in three countries, followed by testing each country individually to find the most appropriate regression model for future credit score predictions.

Multiple linear regression model

Multiple regression is a method used in statistical analysis to determine the value of a dependent variable (the *credit score* in this case) based upon the value of more independent variables such as *savings* and *salary*. This concept was developed in the early twentieth century and evolved from linear regression, the most basic form of predictive analysis (Sheposh, 2015). It is applicable to numerous predictive modeling situations, such as: predicting customer activity on credit cards from their demographics and historical activity patterns, predicting expenditures on vacation travel based on historical frequent flyer data, predicting staffing requirements at help desks based on historical data and product and sales information, predicting sales from cross-selling of products from

historical information, and predicting the impact of discounts on sales in retail outlets (Shmueli et al., 2018). The multiple linear regression model performs the best if it is a linear relationship between *credit score* and *balance*, or between *credit score* and *estimated salary* in the given dataset. The heatmap clearly demonstrates the lack of correlation between these features; therefore, linear regression models cannot perform well in this case (Figure 103).

Polynomial regression model

The polynomial regression model is a more flexible extension of the linear regression, used if the data follows a curve instead of a straight line. This regression model can find nonlinear patterns in the data. *Tenure* and *age* are also included in the attributes of this model, in addition to *balance* and *estimated salary*.

Random forest regression model

The random forest is one of the most popular and efficient techniques used in machine learning for regression (Mojsilović et al. 2023). This method relies on randomly constructed decision trees based on a subset of the training data (Zhang et al. 2024).

Random forest regression can find and model more complex relationships between variables and performs well with large amounts of data. *Balance, estimated salary, tenure, age, country,* and *gender* are all featured. The random forest regression model gives accurate predictions even if data is missing from the set. The tree-like structure makes data interpretation easier (Figure 104).

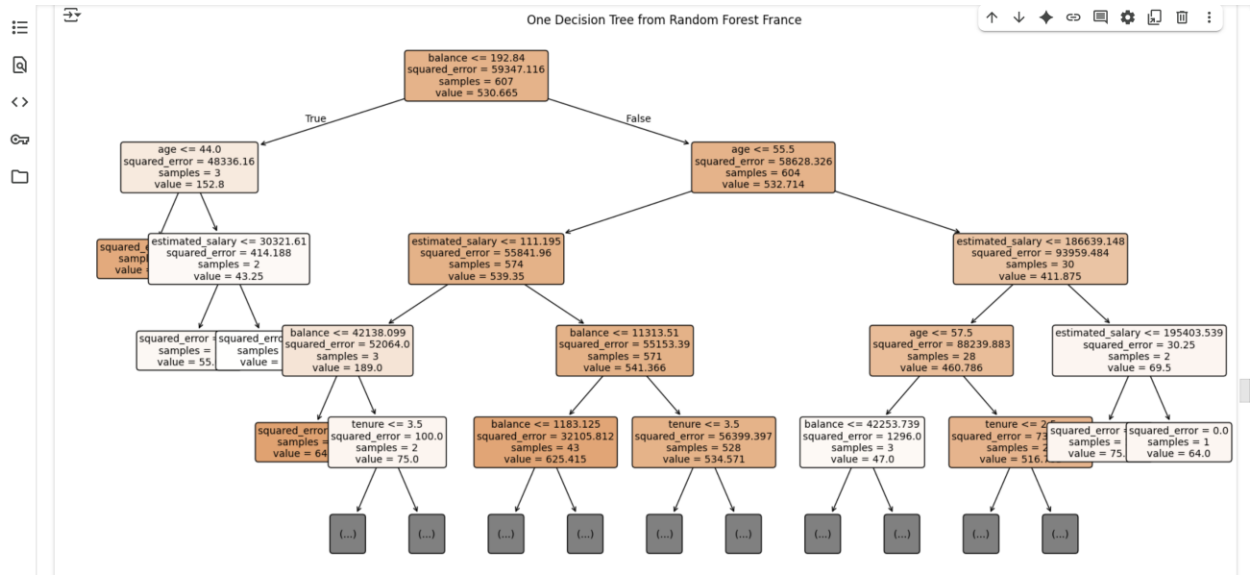


Figure 104: Example of one decision tree France

SVR

Support vector regression performs optimally with small data sets but is also suitable for more complex data. Developed from the linear regression model, it allows the model to have a specific tolerance of error, which makes it more flexible and accurate.
Support vector regression is a type of support vector machine. During training the SVM learns how important each of the training data points is to represent the decision boundary. Typically, only a subset of those training points matters for defining the decision boundary, and these are called support vectors, given the name of the model (Müller and Guido, 2016). SVR provides a mathematical framework for transforming data into higher dimensional spaces where nonlinear relationships become linear (Jadhav and Pisal, 2025).

LASSO regression model

The last model introduced in this analysis is the LASSO regression model. LASSO performs feature selection, which is useful when the data set is high dimensional. It identifies the more important attributes and simplifies the model.

Evaluation of the different models

The greatest way to determine the most appropriate regression model is to visualise their performance. The same colors represent the same type of machine learning model, and the diagonal line on the visualisations represents the ideal prediction line, where the future predicted score equals the actual credit score. The closer the points are to the ideal line, the better the credit score prediction is.

Collective data

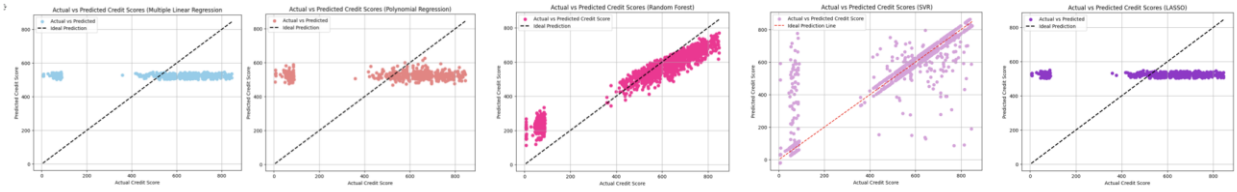


Figure 105: Model comparison

Looking at the data collectively in all three countries: Spain, France and Germany, the support vector regression model performs the best (Figure 105). Random forest regression also gives satisfactory results. The models performed significantly better if higher *credit score* values were used (over 350). The vast divide between data points is detectable on every scatter diagram, showing that the data points are obviously segregated into two groups and that all the tested models behave differently in these two groups. Completely removing the smaller group with low credit scores could be another interesting study in the future. None of the linear regression models perform well in this case because there is no linear correlation between any of the features.

Spain

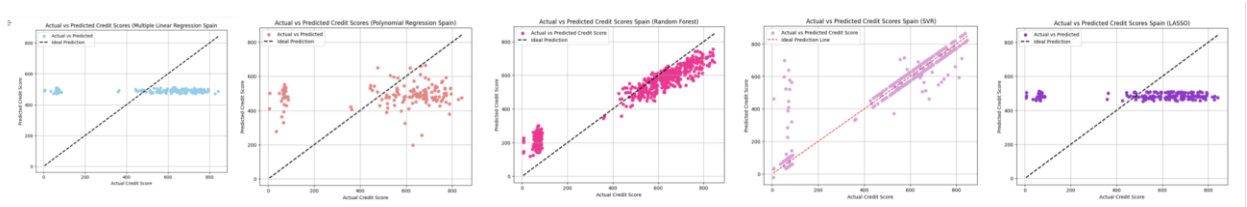


Figure 106: Model comparison for Spain

Similarly to the collective data, the support vector regression model seems to be closer to the ideal prediction line compared to the random forest regression model, which had a strong performance. Random forest regression is next to the best model to predict credit scores in Spain.

France

Figure 107: Model comparison for France

France is the only country of the analysis where the random forest regression model seems to perform better than the SVR. SVR once more provided acceptable results, but random forest regression executed better predictions.

Germany

Figure 108: Model comparison for Germany

Like for Spain, SVR achieved the most accurate credit score predictions for Germany, almost identical to the diagonal ideal prediction line, as shown in Figure 108.

Comparing different performance metrics is useful for deciding on the most accurate regression model. The Mean squared error (MSE) and R- squared are standard evaluation measures to analyse accuracy. The Mean squared error measures how close the prediction is to the actual values in the test set. Lower numbers mean the performance is better. In this case Mean squared error is the average squared difference between the predicted and actual credit scores. For example, MSE= 3474 using SVR in Germany means that the credit score prediction could be 59 points away from the actual credit score. R-squared is a number between 0 and 1, where 1 means the model is the perfect fit.

MSE

	Collective	Spain	France	Germany
Multiple linear regression	63367	65493	62698	53704
Polynomial regression	63306	65567	63329	55730
Random forest regression	9471	10721	9377	8357
Support vector regression	9048	6576	12708	3474
LASSO regression	61636	63468	59652	53988

Figure 109: MSE

The SVR model achieved the lowest MSE values in Spain, Germany and for the combined dataset of all three countries. The random forest regression model has the best performance in France according to the mean squared error (Figure109). This outcome supports the previous theory about the most appropriate model to use. Analysing the R-squared numbers provides equal results regarding the search for the most effective machine learning model.

R-squared

	Collective	Spain	France	Germany
Multiple linear regression	-0.01	-0.02	-0.01	0.00
Polynomial regression	-0.01	-0.02	-0.02	-0.04
Random forest regression	0.846	0.84	0.84	0.85
Support vector regression	0.853	0.90	0.79	0.94
LASSO regression	-0.01	-0.02	-0.01	-0.02

Figure 110: R-squared

Conclusion

The analysis clearly shows that the SVR model executes the most accurate predictions for future credit scores, based not only on salary and saving, but also on other factors such as age, gender, and the length of loan repayment period. Considering only the *salary* and *saving* attributes would be insufficient to predict credit scores as there is no linear relationship between these variables. A higher balance does not necessarily mean higher credit scores.

Support vector machines, including support vector regression, are powerful models and perform well on a variety of datasets. They work effectively on data with few or with many features, but the high number of samples could become more challenging (Müller and Guido, 2016).

The banking data provided for this study is generally considered as a small dataset, which contributes to the SVR's favorable performance. This excludes France, as the dataset for France is larger than those for Spain or Germany. The random forest regression model can perform well without careful preprocessing and tuning of parameters, unlike the SVR model. This is why, these days, most people use tree-based models.

However, adding new alternative features could improve future predictions.
Factors used to measure credit scores are constantly changing and evolving. Back in the 20th century, local newspapers were the source of information until 1956, when the idea of using data to measure the level of risk of lending money came. Fifteen years later the biggest revolution was the first consumer-credit score: a number (higher number= better rating). Nowadays data can be collected from a customer's smartphone, from their travel or gambling habits. Even psychological quizzes are commonly used, or it is possible that social media accounts become a new source of data for credit analysis in the future (The Economist, 2019).
In addition, criminal history (Finlay, 2012, cited by Wandmacher, 2025) or employment history (Hayashi, 2019, cited by Wandmacher, 2025) could be considered new factors for more accurate predictions in credit scoring (Wandmacher, 2025).

These potential new factors can be beneficial and have a good impact on future machine learning models used in the finance sector, but they come with risks as well, such as violating customer privacy rights, for instance. This is another fascinating research field to see how far machine learning regression models can evolve in the future.

Reference list

Jadhav, A.A. and Pisal, P.S. (2025). '*Harnessing Kernel Tricks for Nonlinear Problem Solving: SVM Applications*', 2025 5th International Conference on Innovative Research in Applied Science, Engineering and Technology (IRASET). Fez, Morocco: IEEE. pp. 1–7.

Larose, D. T. (2015). *Data Mining and Predictive Analytics*. 2nd edition. Hoboken NJ: John Wiley & Sons, Inc.

López de Prado, M.M. (2018). *Advances in financial machine learning*. Hoboken NJ: John Wiley & Sons, Inc. (Accessed: 15 June 2025).

Mishra, A. (2020). *Machine Learning for iOS Developers*. Hoboken NJ: John Wiley & Sons.

Mojsilović, M., Cvejić, R., Pepié, S., Karabašević, D., Saračević, M., and Stanujkić, D. (2023). *Statistical evaluation of the achievements of professional students by combination of the random forest algorithm and the ANFIS method.* Heliyon*, 9(3)*, e21768: Elsevier Ltd.

Müller, A. C. and Guido, S. (2016). *Introduction to Machine Learning with Python: A Guide for Data Scientists*. Sebastopol CA: O'Reilly Media.

Rani, R. and Gupta, S. (2024). *Predicting Home Loan Approvals Using Random Forest Classifiers: A Comprehensive Machine Learning Approach*, 2024 3rd International Conference for Advancement in Technology (ICONAT). Goa, India: IEEE. pp. 1–4.

Sheposh, R. (2025). *Multiple regression*. Ipswich MA: Salem Press.

Shmueli, G., Bruce, P. C., and Gedeck, P. (2018). *Data Mining for Business Analytics: Concepts, Techniques and Applications in R, Python, and Jmp*. New York: John Wiley & Sons.

The Economist (2019). *A brief history – and future – of credit scores.* The Economist, 6 July. London: The Economist Group Limited.

Wandmacher, R. et al. (2025). A *Global Comparison of Credit Bureaus Based on Data Utilization in Credit Scoring*, International Journal of Banking & Finance (IJBF), 20(1). Sintok, Kedah Darul Aman, Malaysia: Universiti Utara Malaysia (UUM) Press. pp. 23–38.

Zhang, L., Zhang, Y., Ren, K., Li, D., & Yang, Y. (2024). *MLCopilot: Unleashing the Power of Large Language Models in Solving Machine Learning Tasks.* In Proceedings of the 18th

Conference of the European Chapter of the Association for Computational Linguistics (EACL 2024). St. Julian's, Malta: Association for Computational Linguistics. pp. 2931-2959.

Appendix

https://colab.research.google.com/drive/14OoeCke4F-obi5-WPM7ynFEJ5uP5Q8Rh?usp=sharing