Credit risk score prediction and scorecard build

This notebook will show how to simulate credit risk score, evaluate model prediction and build a Scorecard example for each loan

```
It also complement the information provided in the notebook
Credit_risk_modeling__Expected_Loss__EL__PD_LGD_EAD
```

Notes

- The original dataset have around 2 millions of loan transactions - from 2014 to 2018
- The machine learning model - GLM was built using data between 2015 and 2017 only
- All loan status are avaliable at loan_status and was filtered by Charged Off and Fully paid only
- The coeficients of the GLM model is used to build the credit risk score as an example for each loan
- The original dataset loan sample did not have contract id for each loan. To simulate contract_id you can use the PySpark function monotonically_increasing_id with other datasets if you want to simulate the execution with bigger files

Environment

This notebook will use 2 frameworks to run the entire solution

- Spark 3.3.1 and
- H2O Cluster 3.38.0.4

In [1]:
```
import os
import h2o
from pyspark.sql import SparkSession
import pandas as pd
# from deltalake import DeltaTable

## Spark function to generate unique contract_id
from pyspark.sql.functions import monotonically_increasing_id

import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
def fshape(dataframe1):
    print('Shape : ', dataframe1.count(), len(dataframe1.columns))

def fhead(dataframe1, num_records=3):
    pd.options.display.max_columns = None
    return dataframe1.limit(num_records).toPandas()

def fsummary(dataframe1):
    return dataframe1.summary().toPandas()

## default Spark appName - se preferir
spark = SparkSession.builder.appName('Spark3-ML-quick-app').master('local[*]').getOrCreate()
sc = spark.sparkContext
spark
```

Out[2]: **SparkSession - in-memory**

SparkContext

Spark UI

Version	v3.3.1
Master	local[*]
AppName	Spark3-ML-quick-app

In [3]:
```
h2o.connect(ip = '172.25.238.198')
h2o.remove_all()
```

```
Connecting to H2O server at http://172.25.238.198:54321 ... successful.
Warning: Your H2O cluster version is too old (3 months and 25 days)!Please download and install the latest version from http://h
2o.ai/download/
```

H2O_cluster_uptime:	4 hours 6 mins
H2O_cluster_timezone:	America/Sao_Paulo
H2O_data_parsing_timezone:	UTC
H2O_cluster_version:	3.38.0.4
H2O_cluster_version_age:	3 months and 25 days !!!
H2O_cluster_name:	userds1
H2O_cluster_total_nodes:	1
H2O_cluster_free_memory:	5.172 Gb
H2O_cluster_total_cores:	12
H2O_cluster_allowed_cores:	12
H2O_cluster_status:	locked, healthy
H2O_connection_url:	http://172.25.238.198:54321
H2O_connection_proxy:	null
H2O_internal_security:	False
Python_version:	3.9.13 final

In [4]:
```
data_dir = '/tmp/Credit_Risk_Modeling/test_47k_loan_2018.parquet/'
parquet_files = []
# Get a list of all the Parquet files in the directory
parquet_files = [os.path.join(data_dir, f) for f in os.listdir(data_dir) if f.endswith(".parquet")]
# parquet_files[0]
filename_scorecard = parquet_files[0]
```

The GLM was build using a train dataset with 2 millions of loan transactions

Goals of this GLM model

- Predict the probability of default (PD)
- Provide a scorecard table ... simulate FICO score sample - credit risk score between 300 and 850

Load data and GLM model into H2O Cluster

In [5]:
```
## Load data into H2O Cluster
hdf_test_scorecard = h2o.upload_file(filename_scorecard, destination_frame='hdf_test_scorecard_dat2_silver.hex')
```

```
Parse progress: |██████████████████████████████████████████████████████████████| (done) 100%
```

In [6]:
```
hdf_test_scorecard.head(3)
```

Out[6]:

contract_id	loan_status_good_vs_bad	acc_now_delinq	addr_state	annual_inc	chargeoff_within_12_mths	credit_conversion_factor_CCF	delinq_2yrs	dti
4.29497e+10	1	0	CA	100000	0	0	0	30.46
4.29497e+10	1	0	OH	45000	0	0	0	50.53
4.29497e+10	1	0	WA	100000	0	0	0	18.92

```
[3 rows x 50 columns]
```

In [7]:
```
hdf_test_scorecard.shape
```

Out[7]:
```
(47182, 50)
```

In [8]:
```
def fnc_percent_print(metric):
    return round(metric * 100 , 4)
```

In [9]:
```
dir_h2o_model = 'tmp/Credit_Risk_Modeling/h2o_glm_gbm_model/'
glm_model = h2o.load_model('/mnt/d/'+dir_h2o_model+'fit_glm_2015_2017.model')

# glm_model.model_performance()
print(' GLM model accuracy - ', round(glm_model.accuracy()[0][1] * 100 , 4), ' % ')

print(' --- Max precision of ' , fnc_percent_print(glm_model.find_threshold_by_max_metric(metric='precision')),
      ' % with threshold adjustment ')
```

```
 GLM model accuracy -  78.3499  %
 --- Max precision of  96.2257  % with threshold adjustment
```

Info about classification metrics evaluation - machine learning model

Accuracy is a commonly used evaluation metric for classification models. It measures the proportion of correct predictions made by the model out of the total number of predictions made. Accuracy is a useful metric for balanced datasets where the number of positive and negative instances are roughly equal. However, it can be misleading in imbalanced datasets where the number of positive and negative instances are significantly different. In such cases, other evaluation metrics such as precision, recall, and F1-score may be more appropriate.

`Accuracy = (#True_Positive + #True_Negatives) / (#True_Positives + #True_Negatives + #False_Positives + #False_Negatives)`

Precision : Measures how precise/accurate your model is. It's the ratio between the correctly identified positives (true positives) and all identified positives. The precision metric reveals how many of the predicted classes are correctly labeled.

`Precision = #True_Positive / (#True_Positive + #False_Positive)`

Recall : Measures the model's ability to predict actual positive classes. It's the ratio between the predicted true positives and what was actually tagged. The recall metric reveals how many of the predicted classes are correct.

`Recall = #True_Positive / (#True_Positive + #False_Negatives)`

F1 score : The F1 score is a function of Precision and Recall. It's needed when you seek a balance between Precision and Recall.

`F1 Score = 2 * Precision * Recall / (Precision + Recall)`

Note

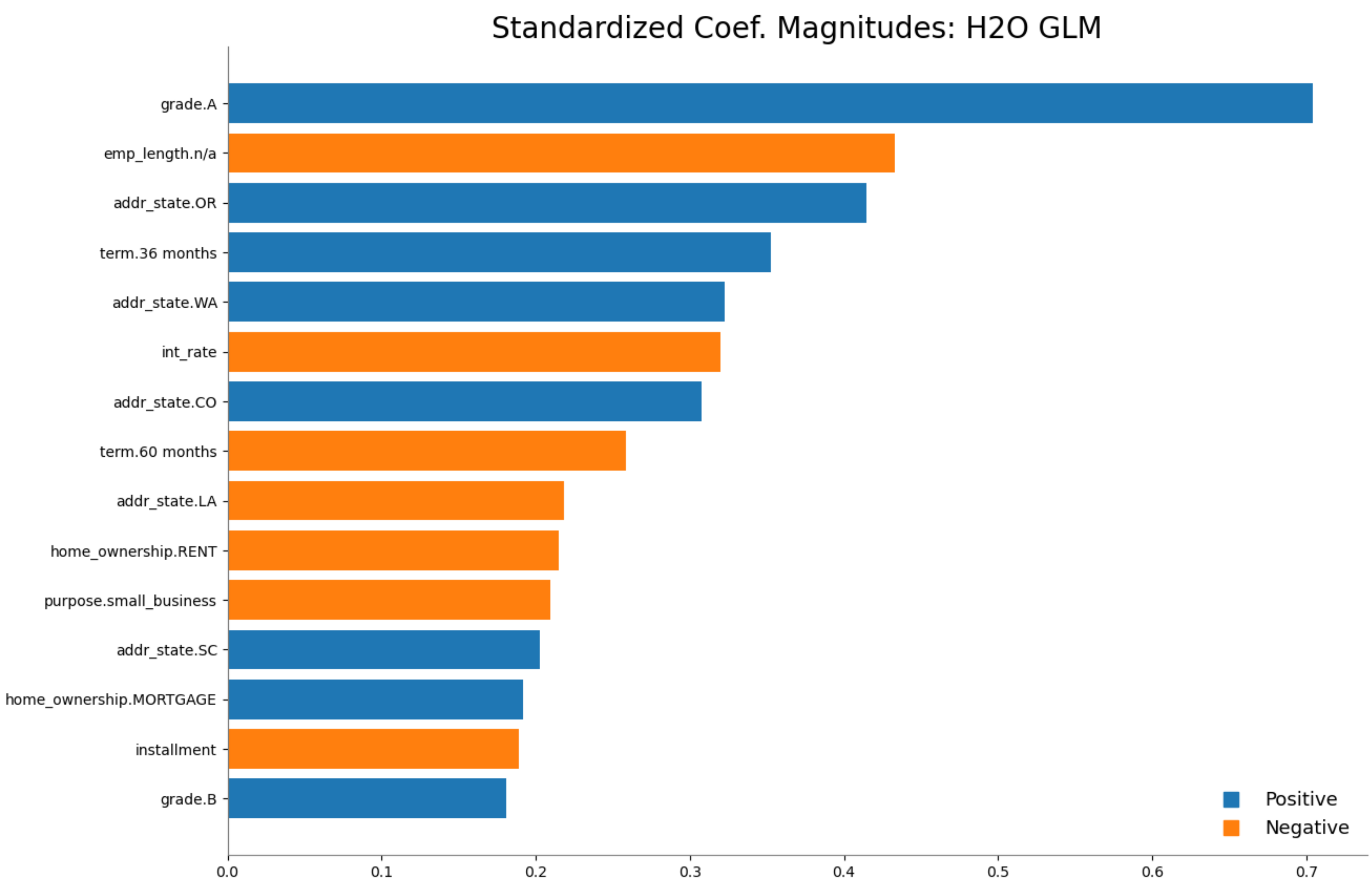
Precision, recall and F1 score are calculated for each class separately (class-level evaluation) and for the model collectively (model-level evaluation).

The top 15 feature importances, considering positive and negative impacts of payment (fully paid - not default), are as follows:

- Loans with Grade A and a contract term of 36 months are likely to have a positive impact on fully paid status.
- Loans with bigger interest rates and longer contract terms, specifically 60 months, are likely to have a negative impact on payment, resulting in an increased probability of loan default and a lower credit risk score.

In [10]:
```
glm_model.std_coef_plot(num_of_features=15)
```

Out[10]:
```
<h2o.plot._plot_result._MObject at 0x1ed16d780a0>
```



GLM Model Prediction: Probability of Default (PD)

- PD represents the probability of default, where p1 is the probability of payment, and p0 is the probability of default.

For the purpose of building a credit risk score and scorecard simulation, we will utilize the coefficients of the machine learning model as presented below.

In [11]:
```
hdf_prediction = glm_model.predict(hdf_test_scorecard)
hdf_prediction.head(3)
```

```
glm prediction progress: |████████████████████████████████████████████████| (done) 100%
```

Out[11]:

predict	p0	p1
1	0.40357	0.59643
1	0.431006	0.568994
1	0.0375703	0.96243

```
[3 rows x 3 columns]
```

In [12]:

```
hdf_scorecard_v3 = hdf_prediction.concat(hdf_test_scorecard[['contract_id', 'loan_status_good_vs_bad']], axis=1)
hdf_scorecard_v3.head(3)
```

Out[12]:

predict	p0	p1	contract_id	loan_status_good_vs_bad
1	0.40357	0.59643	4.29497e+10	1
1	0.431006	0.568994	4.29497e+10	1
1	0.0375703	0.96243	4.29497e+10	1

```
[3 rows x 5 columns]
```

In [13]:

```
## Export h2o dataframe for future use in Spark
credit_risk_score_filename= '/tmp/Credit_Risk_Modeling/loan_h2oFrame_credit_risk_score_2018_scorecard_contract_id.csv.gz'
h2o.export_file(hdf_scorecard_v3,
                '/mnt/d' + credit_risk_score_filename,
                force=True, compression='gzip')
```

```
Export File progress: |████████████████████████████████████████████████| (done) 100%
```

Scorecard - Generation

- Credit risk score - simulation only / FICO score between 300 and 850
- Credit risk score used be applied at customer level, and the example below shows for each loan (unique contract id)

Python pandas example

In [14]:

```
# Get coefficients from the model
coefficients = glm_model.coef()

df_scorecard = pd.DataFrame({'Feature_name' : coefficients.keys(),
                             'Coefficients_raw' : coefficients.values()})
df_scorecard['Coefficients'] = df_scorecard['Coefficients_raw'].round(3)

df_scorecard['Original_feature_name'] = df_scorecard['Feature_name'].str.split('.').str[0]

## FICO score sample
min_score = 300
max_score = 850

min_sum_coef = df_scorecard.groupby('Original_feature_name')['Coefficients'].min().sum()
print('Min sum coef ', min_sum_coef)

max_sum_coef = df_scorecard.groupby('Original_feature_name')['Coefficients'].max().sum()
print('Max sum coef ', max_sum_coef)

df_scorecard['Score_Calculation'] = df_scorecard['Coefficients'] * (max_score - min_score) / (max_sum_coef - min_sum_coef)

df_scorecard['Score_Calculation'][0] = ((
    (df_scorecard['Coefficients'][0] - min_sum_coef) / (max_sum_coef - min_sum_coef)) * (max_score - min_score) + min_score)

print(' -------------------- FICO SCORE simulation - credit risk score')
min_sum_score_prel = df_scorecard.groupby('Original_feature_name')['Score_Calculation'].min().sum().round()
print(' Credit score - min :', min_sum_score_prel)

max_sum_score_prel = df_scorecard.groupby('Original_feature_name')['Score_Calculation'].max().sum().round()
print(' Credit score - max :', max_sum_score_prel)

print('-- Scorecard generation .csv')
df_scorecard.to_csv('/data_dir_tmp/GITHUB_Risk_Management/Credit_Risk_Modeling/data_s3/dat2_silver/loan_scorecard_coef.csv',
                   index=None)

df_scorecard.head(5)
```

```
Min sum coef  0.6180000000000001
Max sum coef  4.0520000000000005
 -------------------- FICO SCORE simulation - credit risk score
 Credit score - min : 300.0
 Credit score - max : 850.0
-- Scorecard generation .csv
```

Out[14]:

	Feature_name	Coefficients_raw	Coefficients	Original_feature_name	Score_Calculation
0	Intercept	2.323289	2.323	Intercept	573.078043
1	addr_state.AK	0.000000	0.000	addr_state	0.000000
2	addr_state.AL	-0.131340	-0.131	addr_state	-20.981363
3	addr_state.AR	-0.166008	-0.166	addr_state	-26.587070
4	addr_state.AZ	0.000000	0.000	addr_state	0.000000

In [15]:
```
# ### Scorecard sample using python Pandas and H2O dataframe instead of H2O MODEL (same results)
# pd2_hdf_scorecard = hdf_scorecard_v2.as_data_frame()
# # pd2_hdf_scorecard.head(3)

# ## Sample variables
# # min_score_fico = 300
# # max_score_fico = 850

# ## Calculate Credit risk score using pandas
# fnc_fico_score = lambda x_scaled, min_score, max_score: (
#     (x_scaled * (max_score - min_score)) + min_score
# )
# pd2_hdf_scorecard['credit_score_FICO'] = pd2_hdf_scorecard['p1'].apply(fnc_fico_score, args=(300, 850))

# print(' Credit risk score - min: ', pd2_hdf_scorecard['credit_score_FICO'].round().min())

# print(' Credit risk score - max: ', pd2_hdf_scorecard['credit_score_FICO'].round().max())

# pd2_hdf_scorecard.head(3)
```

Read prediction generated using H2O GLM with Spark

In [16]:
```
## Read prediction generated using H2O GLM with Spark
sdf_hdf_contract_id_only = spark.read.csv(credit_risk_score_filename, inferSchema=True, header=True)
# fhead(sdf_hdf_contract_id_only)
```

Credit risk score function using Spark

In [17]:
```
def fnc_credit_risk_score_udf(credit_score, min_score=300, max_score=850):
    """
    This function implements Credit Risk score with range defined
    Returns
    -------
    Credit Risk score - range
    """
    score = round (
        (credit_score * (max_score - min_score)) + min_score
        , 3)

    return score

## Register the formula to be used by Spark-SQL
from pyspark.sql.types import FloatType
spark.udf.register('fnc_credit_risk_score_udf', fnc_credit_risk_score_udf, FloatType())
```

Out[17]:
```
<function __main__.fnc_credit_risk_score_udf(credit_score, min_score=300, max_score=850)>
```

In [18]:
```
sdf_hdf_contract_id_only.createOrReplaceTempView('TB_CREDIT_SCORE')

sql_qry = """
    SELECT fnc_credit_risk_score_udf(TB_CREDIT_SCORE.p1) as credit_risk_score
        , TB_CREDIT_SCORE.*
    FROM TB_CREDIT_SCORE
    WHERE 1 = 1
"""
sdf_credit_score = spark.sql(sql_qry)
# sdf_credit_score.printSchema()
# sdf_credit_score.show(5)
fshape(sdf_credit_score)
fhead(sdf_credit_score.select('contract_id', 'credit_risk_score'))
```

```
Shape :   47182 6
```

Out[18]:

	contract_id	credit_risk_score
0	42949673060	628.036987
1	42949673112	612.947021
2	42949673130	829.335999

Model Evaluation using Spark

Note : The divergence between the H2O model metrics and the Spark model evaluation is because there are different datasets

- `(H2O model training and model performance vs Spark evaluation with model prediction with h2o and threshold)`

- Wrapper function to print metrics

In [19]:
```python
def fnc_classification_metrics(dataframe1, label='label', prediction='prediction'):
    from pyspark.ml.evaluation import BinaryClassificationEvaluator, MulticlassClassificationEvaluator
    from pyspark.sql.functions import expr, col

    ## obs. as colunas precisam ter o nome label e prediction
    df = dataframe1.select(label, prediction)
    cols = ['label', 'prediction']
    df = df.toDF(*cols)


    # cast label column to Double
    df = df.withColumn("label", df["label"].cast("Double"))
    df = df.withColumn("prediction", df["prediction"].cast("Double"))

    # assuming your DataFrame has the following column names: "label" and "prediction"
    predictionsAndLabels = df.select("label", "prediction")

    # create BinaryClassificationEvaluator object
    binary_evaluator = BinaryClassificationEvaluator(labelCol="label", rawPredictionCol="prediction", metricName="areaUnderROC")

    # create MulticlassClassificationEvaluator object
    multiclass_evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction")

    # compute classification metrics for binary classification
    areaUnderROC = binary_evaluator.evaluate(predictionsAndLabels)
    areaUnderPR = binary_evaluator.setMetricName("areaUnderPR").evaluate(predictionsAndLabels)
    # f1Score = binary_evaluator.setMetricName("f1").evaluate(predictionsAndLabels)

    # compute classification metrics for multiclass classification
    accuracy = multiclass_evaluator.evaluate(predictionsAndLabels, {multiclass_evaluator.metricName: "accuracy"})
    precision = multiclass_evaluator.evaluate(predictionsAndLabels, {multiclass_evaluator.metricName: "weightedPrecision"})
    recall = multiclass_evaluator.evaluate(predictionsAndLabels, {multiclass_evaluator.metricName: "weightedRecall"})
    f1Score = multiclass_evaluator.evaluate(predictionsAndLabels, {multiclass_evaluator.metricName: "f1"})
    confusionMatrix = predictionsAndLabels.groupBy("label", "prediction").count().orderBy("label", "prediction").toPandas()

    # print classification metrics
    print("")
    print("Multiclass Classification Metrics:")
    print("")
    print("Accuracy = %s" % accuracy)
    print("Precision = %s" % precision)
    print("Recall = %s" % recall)
    print("F1 Score = %s" % f1Score)
    print("")
    print("")
    print("Confusion Matrix:")
    print(confusionMatrix)

    print("")
    print("\nBinary Classification Metrics:")
    print("")
#     print("Area Under ROC = %s" % areaUnderROC)
    print("Area Under PR = %s" % areaUnderPR)
#     print("F1 Score = %s" % f1Score)
#     print("Confusion Matrix:")
#     print(binary_evaluator.evaluate(predictionsAndLabels, {binary_evaluator.metricName: "confusionMatrix"}))

fnc_classification_metrics(sdf_credit_score,label='loan_status_good_vs_bad', prediction='predict')

# fnc_classification_metrics(sdf_hdf_contract_id_only,label='loan_status_good_vs_bad', prediction='predict')
```

```
Multiclass Classification Metrics:

Accuracy = 0.8475689881734559
Precision = 0.789360220635455
Recall = 0.8475689881734559
F1 Score = 0.7975474784612404


Confusion Matrix:
   label  prediction  count
0    0.0         0.0    431
1    0.0         1.0   6511
2    1.0         0.0    681
3    1.0         1.0  39559


Binary Classification Metrics:

Area Under PR = 0.8586224751103805
```

In [20]:
```python
fhead(sdf_credit_score)
```

Out[20]:

	credit_risk_score	predict	p0	p1	contract_id	loan_status_good_vs_bad
0	628.036987	1	0.403570	0.596430	42949673060	1
1	612.947021	1	0.431006	0.568994	42949673112	1
2	829.335999	1	0.037570	0.962430	42949673130	1

In [21]:
```
## Export Spark dataframe - Contract_id and Credit_Risk_Score only for Github
(sdf_credit_score.coalesce(1)
 .write.format('parquet').mode('overwrite').save(
     '/tmp/Credit_Risk_Modeling/loan_credit_risk_score_2018_contract_id_test47k.parquet'))
```

Plot chart

In [4]:
```
## Chart review sample
# sdf_credit_score = spark.read.parquet(
#     '/tmp/Credit_Risk_Modeling/loan_credit_risk_score_2018_contract_id_test47k.parquet'
# )
# fshape(sdf_credit_score)
```

In [5]:
```
fhead(sdf_credit_score.select('contract_id', 'credit_risk_score'))
```

Out[5]:

	contract_id	credit_risk_score
0	42949673060	628.036987
1	42949673112	612.947021
2	42949673130	829.335999

In [6]:
```
sdf_credit_score.createOrReplaceTempView('TB_CREDIT_RISK_SCORE_RPT')
rpt_TB_CREDIT_SCORE = """
    SELECT case when loan_status_good_vs_bad = 1 then 'Fully paid'
        WHEN loan_status_good_vs_bad = 0 then 'Charged Off'
        ELSE 'Not mapped'
    END Actual
    , CASE
        WHEN predict = 1 THEN 'Full payment'
        WHEN predict = 0 THEN 'Default'
    END Prediction
    , AVG(credit_risk_score) as mean_credit_risk_score
    -- , tb.*
    FROM TB_CREDIT_RISK_SCORE_RPT
    WHERE 1 = 1
    GROUP BY 1,2

"""
sdf_rpt = spark.sql(rpt_TB_CREDIT_SCORE)
sdf_rpt.printSchema()
```

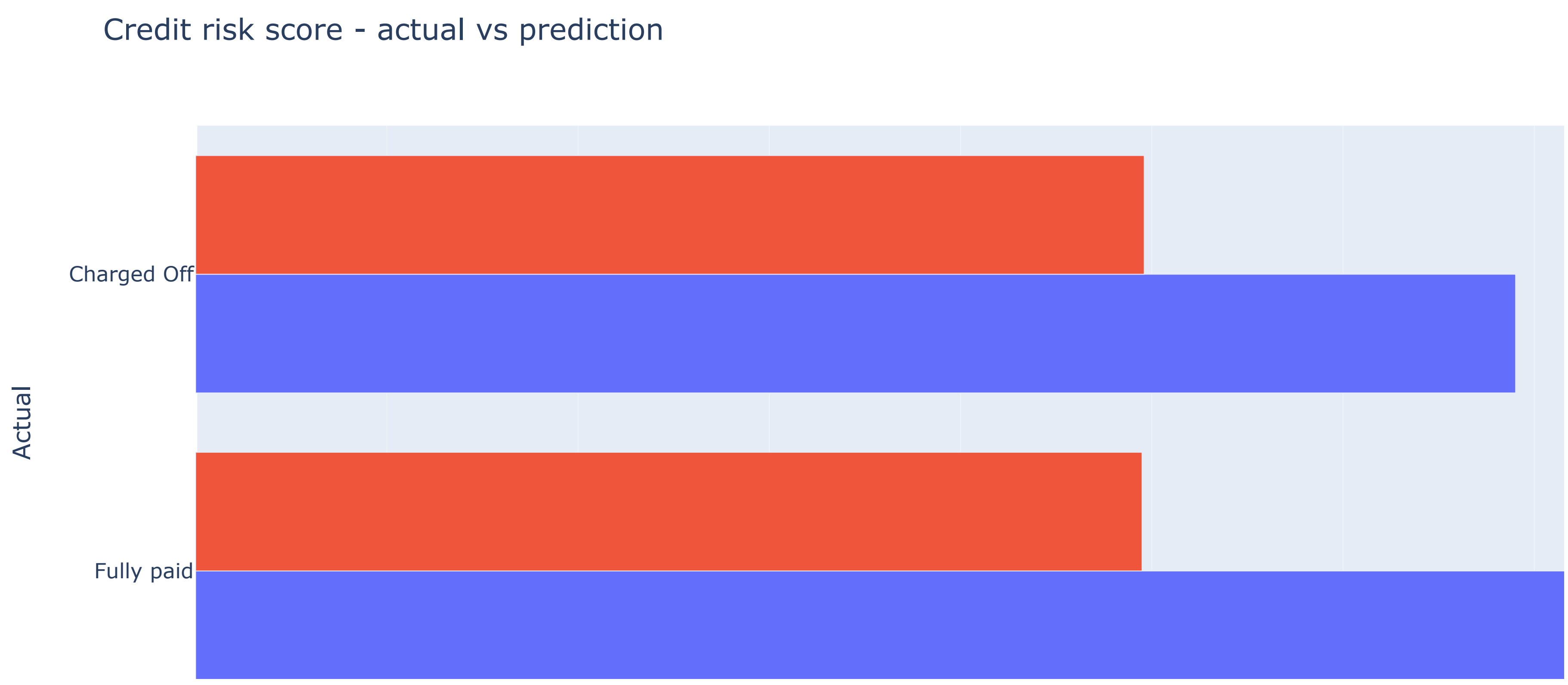
```
root
 |-- Actual: string (nullable = false)
 |-- Prediction: string (nullable = true)
 |-- mean_credit_risk_score: double (nullable = true)
```

In [7]:
```
fhead(sdf_rpt,5)
```

Out[7]:

	Actual	Prediction	mean_credit_risk_score
0	Fully paid	Full payment	738.906921
1	Charged Off	Default	496.044353
2	Fully paid	Default	494.916955
3	Charged Off	Full payment	690.263199

In [8]:
```
# BAR Chart
sdf_rpt.pandas_api().plot.barh(y='Actual', x='mean_credit_risk_score',
                               color='Prediction', barmode='group', title=' Credit risk score - actual vs prediction ')
```




```
In [28]: ## Export HTML VIEWER
         # !jupyter nbconvert --to html Credit_risk_score_with_scorecard__test_prediction.ipynb
```

```
In [ ]:
```