BANK CUSTOMER CHURN

Nursena Baykır

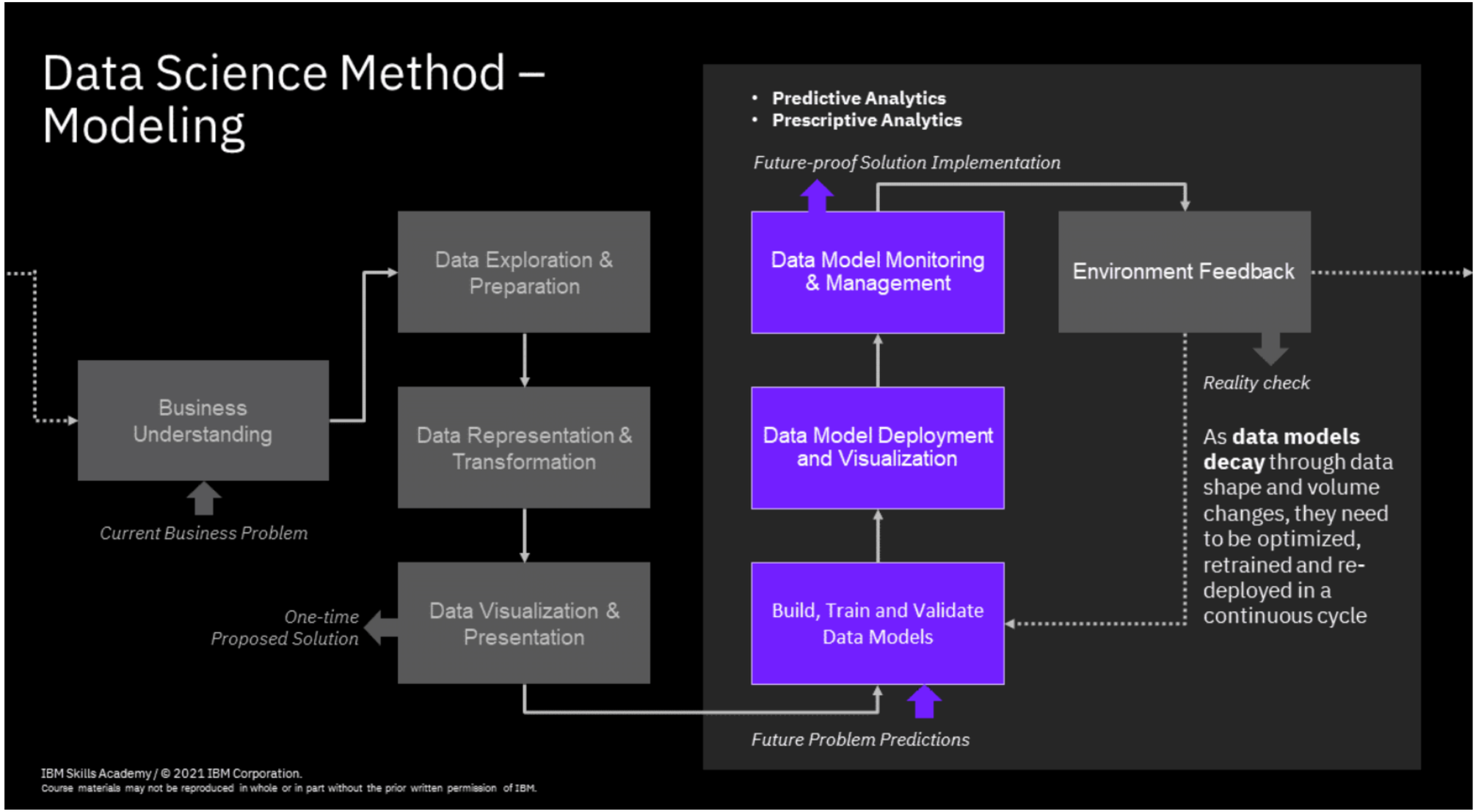
MİNİMALİST

COMPANY PRESENTATİON

ProjeninAmaçları

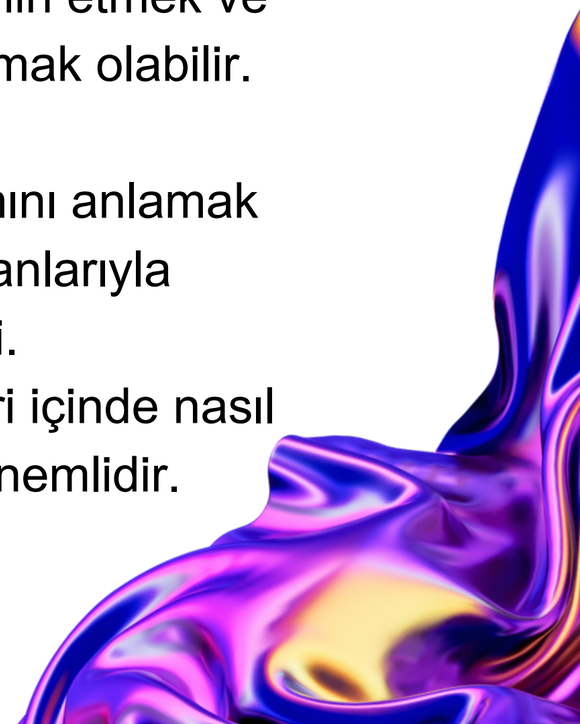
- Müşteri Kaybının Önlenmesi
- Operasyonel Verimliliğin Artırılması
- Müşteri Sadakatinin Artırılması
- Pazarlama Etkinliğinin Artırılması
- Müşteri Davranışının Anlaşılması

Data Science Lifecycle



Business Understanding

- Veri Bilimi ile İlgili Hedeflerin Belirlenmesi
  - Proje kapsamında belirlenen iş hedefleri doğrultusunda veri bilimi ile elde edilmesi beklenen sonuçları belirlenmesi.
  - Örneğin, bir müşteri churn projesinde, hedefler müşteri ayrılma olasılıklarını tahmin etmek ve bu ayrılmaların nedenlerini anlamak olabilir.
- Bağlamın ve İş Süreçlerinin Anlaşılması
  - İş süreçlerini ve projenin bağlamını anlamak için iş birimi liderleri ve uzmanlarıyla görüşmeler yapılmasi.
  - Veri bilimi çözümünün iş süreçleri içinde nasıl entegre olacağını anlamak önemlidir.



EDA

```
df
```

	customer_id	credit_score	country	gender	age	tenure	balance	products_number	credit_card	active_member	estimated_salary	churn
	15634602	619	France	Female	42	2	0.00	1	1	1	101348.88	1
	15647311	608	Spain	Female	41	1	83807.86	1	0	1	112542.58	0
	15619304	502	France	Female	42	8	159660.80	3	1	0	113931.57	1
	15701354	699	France	Female	39	1	0.00	2	0	0	93826.63	0
	15737888	850	Spain	Female	43	2	125510.82	1	1	1	79084.10	0
	...	...	...	...	...	...	...	...	...	...	...	...
5	15606229	771	France	Male	39	5	0.00	2	1	0	96270.64	0
6	15569892	516	France	Male	35	10	57369.61	1	1	1	101699.77	0
7	15584532	709	France	Female	36	7	0.00	1	0	1	42085.58	1
8	15682355	772	Germany	Male	42	3	75075.31	2	1	0	92888.52	1
9	15628319	792	France	Female	28	4	130142.79	1	1	0	38190.78	0

0 rows × 12 columns

Statistics

```
df.describe([.25,.50,.75,.99,])
```

	credit_score	age	tenure	balance	products_number	credit_card	active_member	estimated_salary	churn
count	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.00000	10000.000000	10000.000000	10000.000000
mean	650.528800	38.660800	5.012800	76485.889288	1.530200	0.70550	0.515100	100090.239881	0.203700
std	96.653299	9.746704	2.892174	62397.405202	0.581654	0.45584	0.499797	57510.492818	0.402769
min	350.000000	18.000000	0.000000	0.000000	1.000000	0.00000	0.000000	11.580000	0.000000
25%	584.000000	32.000000	3.000000	0.000000	1.000000	0.00000	0.000000	51002.110000	0.000000
50%	652.000000	37.000000	5.000000	97198.540000	1.000000	1.00000	1.000000	100193.915000	0.000000
75%	718.000000	44.000000	7.000000	127644.240000	2.000000	1.00000	1.000000	149388.247500	0.000000
99%	850.000000	62.000000	10.000000	185967.985400	3.000000	1.00000	1.000000	198069.734500	1.000000
max	850.000000	62.000000	10.000000	250898.090000	4.000000	1.00000	1.000000	199992.480000	1.000000

Müşterilerin bankada kalma eğilimi 5 yıldır

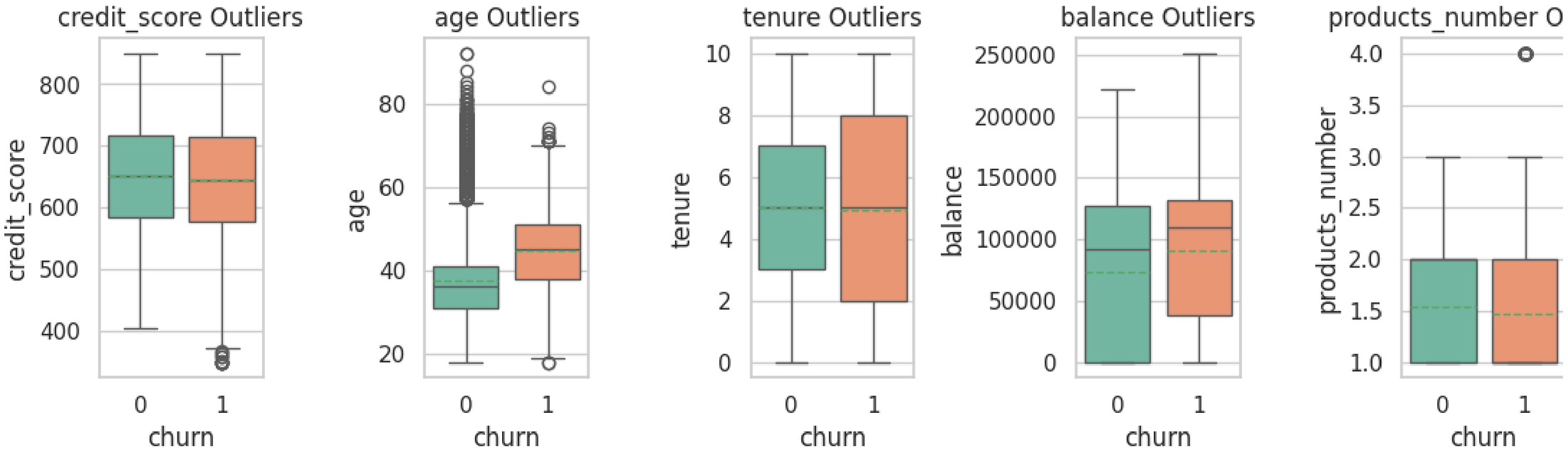
Müiterilerin yarısı aktif üyedir.

Müşterilerin yarısından fazlası kredi kartı kullanıyor.

Outlier Values

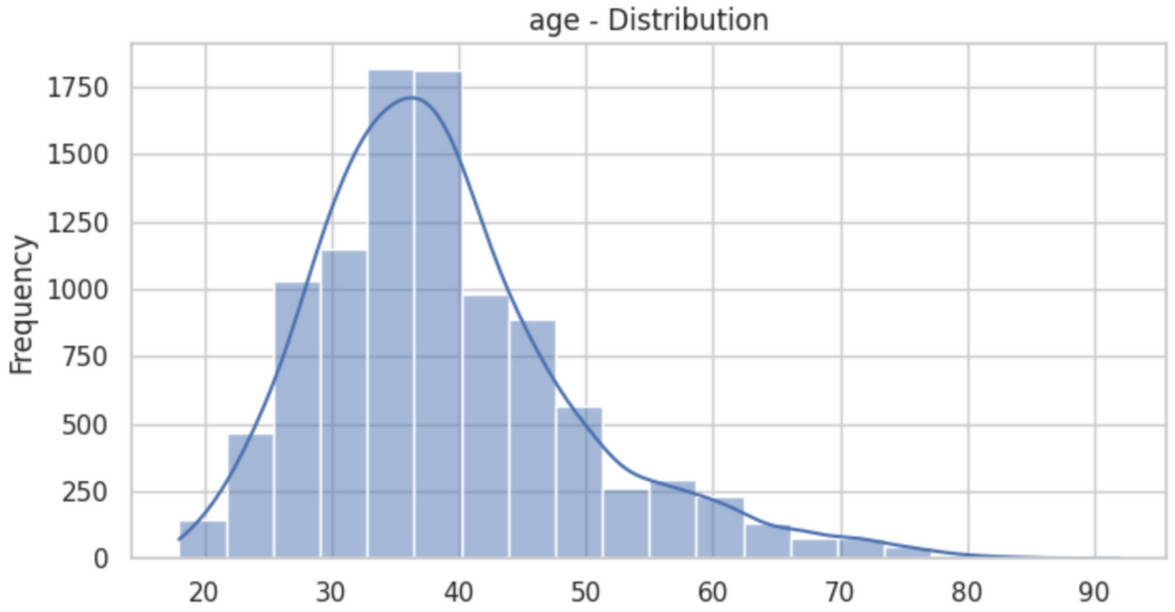
```
[35] def boxplot_outliers(dataframe):
         """
         Creates horizontal boxplots for numerical columns to visualize outliers in relation to the target variable.

         Parameters:
         - dataframe (pd.DataFrame): The DataFrame containing the data.

         Returns:
         - None
         """
         plt.figure(figsize=(18, 6))

         numerical_columns = dataframe.select_dtypes(include=['int', 'float']).columns

         for i, column in enumerate(numerical_columns, 1):
             if column != 'churn':
                 plt.subplot(2, len(numerical_columns)-1, i)
                 sns.boxplot(x='churn', y=column, data=dataframe, showmeans=True, meanline=True, palette="Set2", hue='churn', legend=False)
                 plt.title(f'{column} Outliers')

         plt.tight_layout()
         plt.show()
```

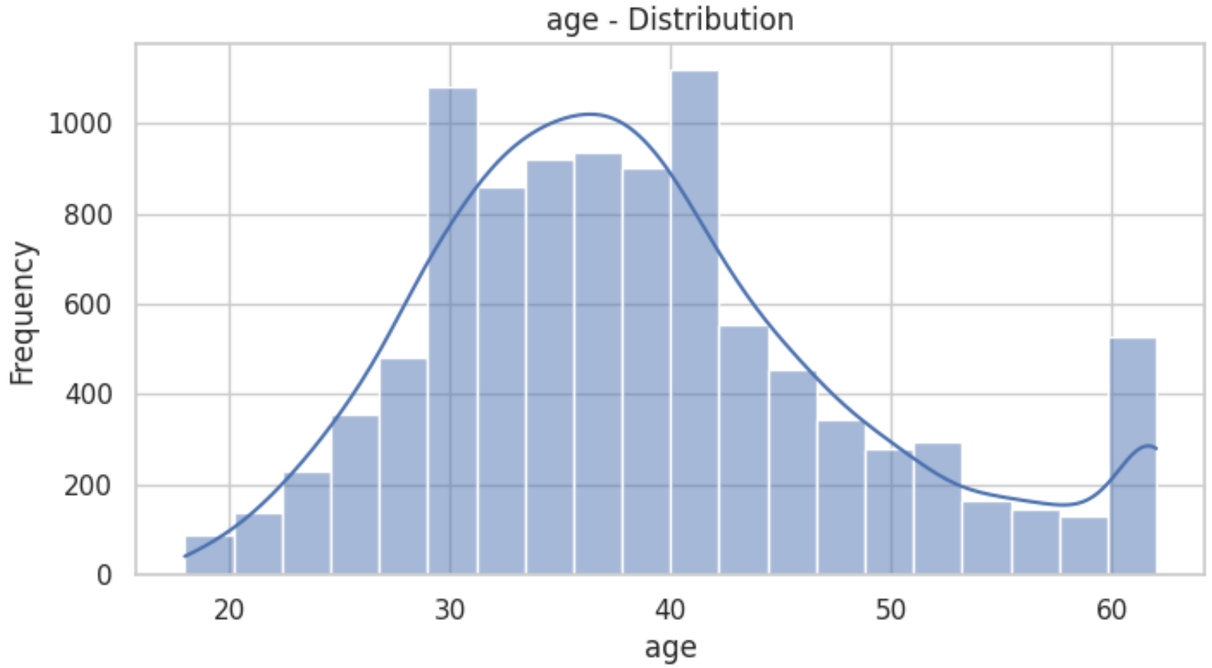
```
[36] boxplot_outliers(df)
```



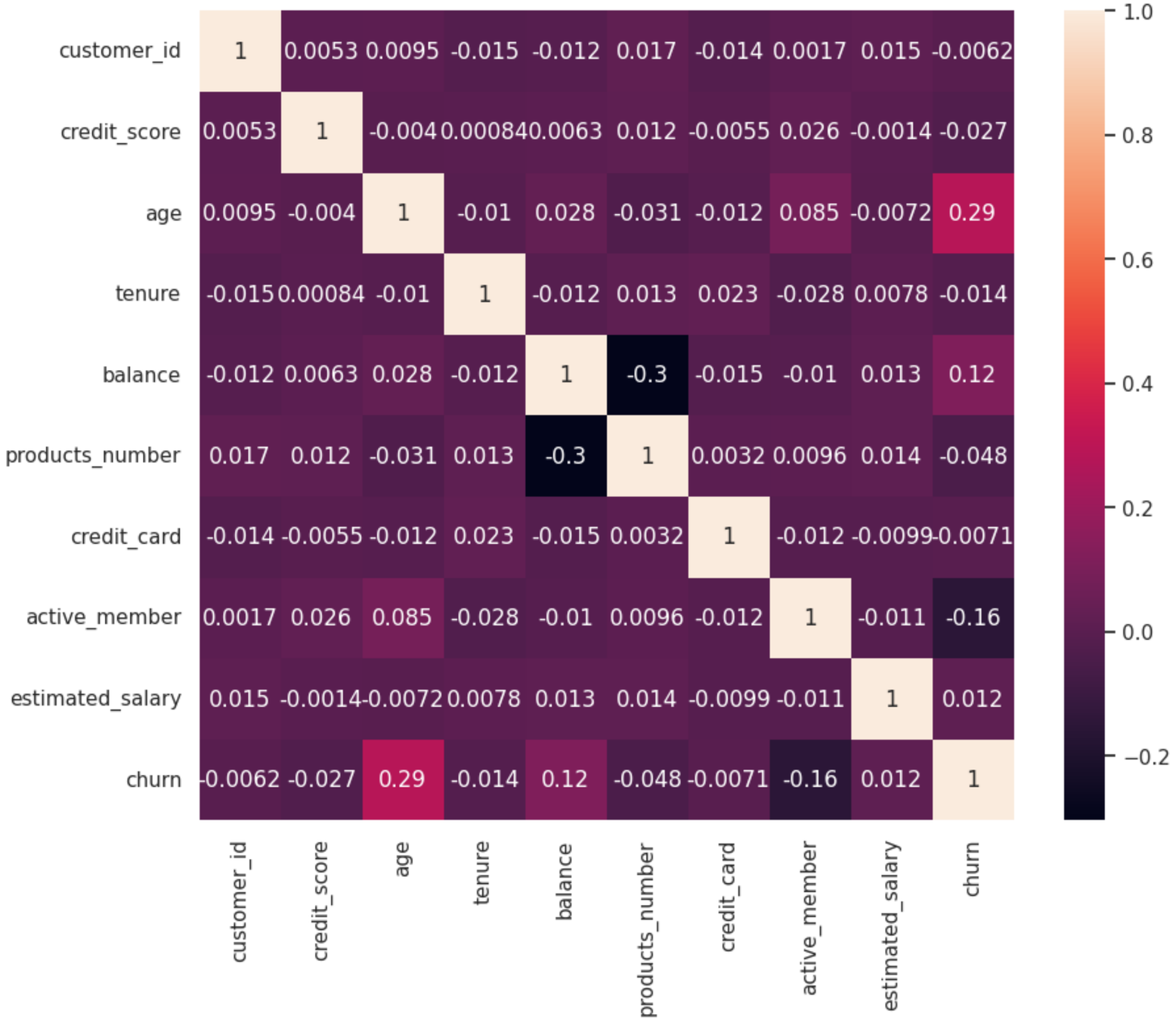
Outlier Data Cleaning

```
#visualizing age feature
visualize_numeric_column(df,"age")
```



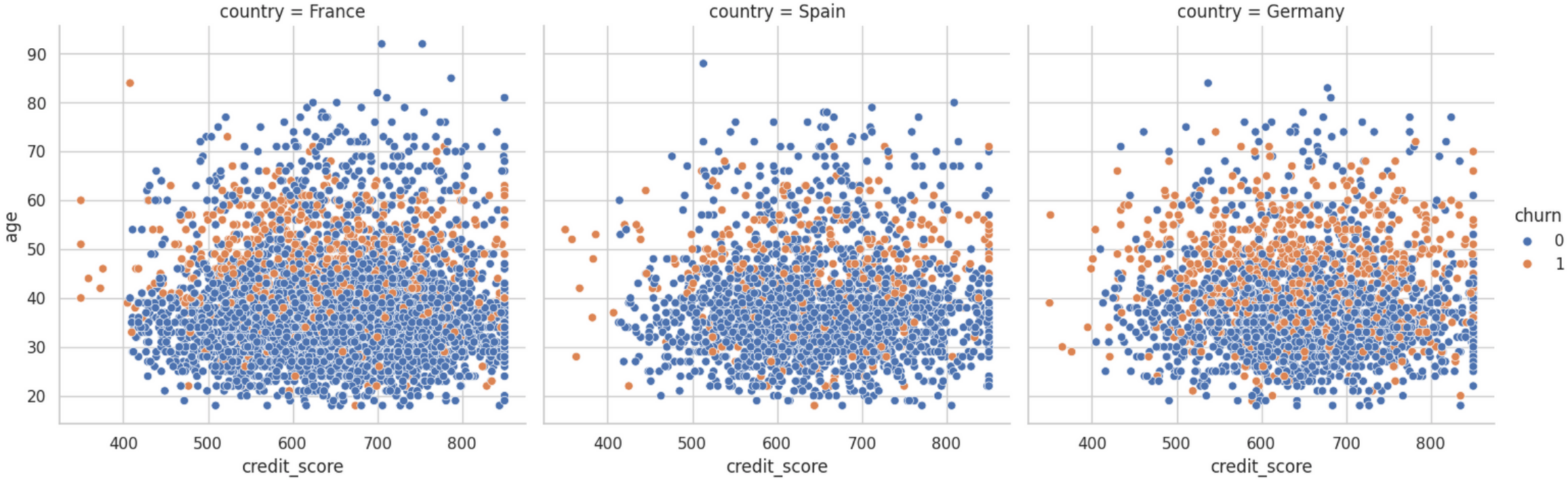


Heatmap

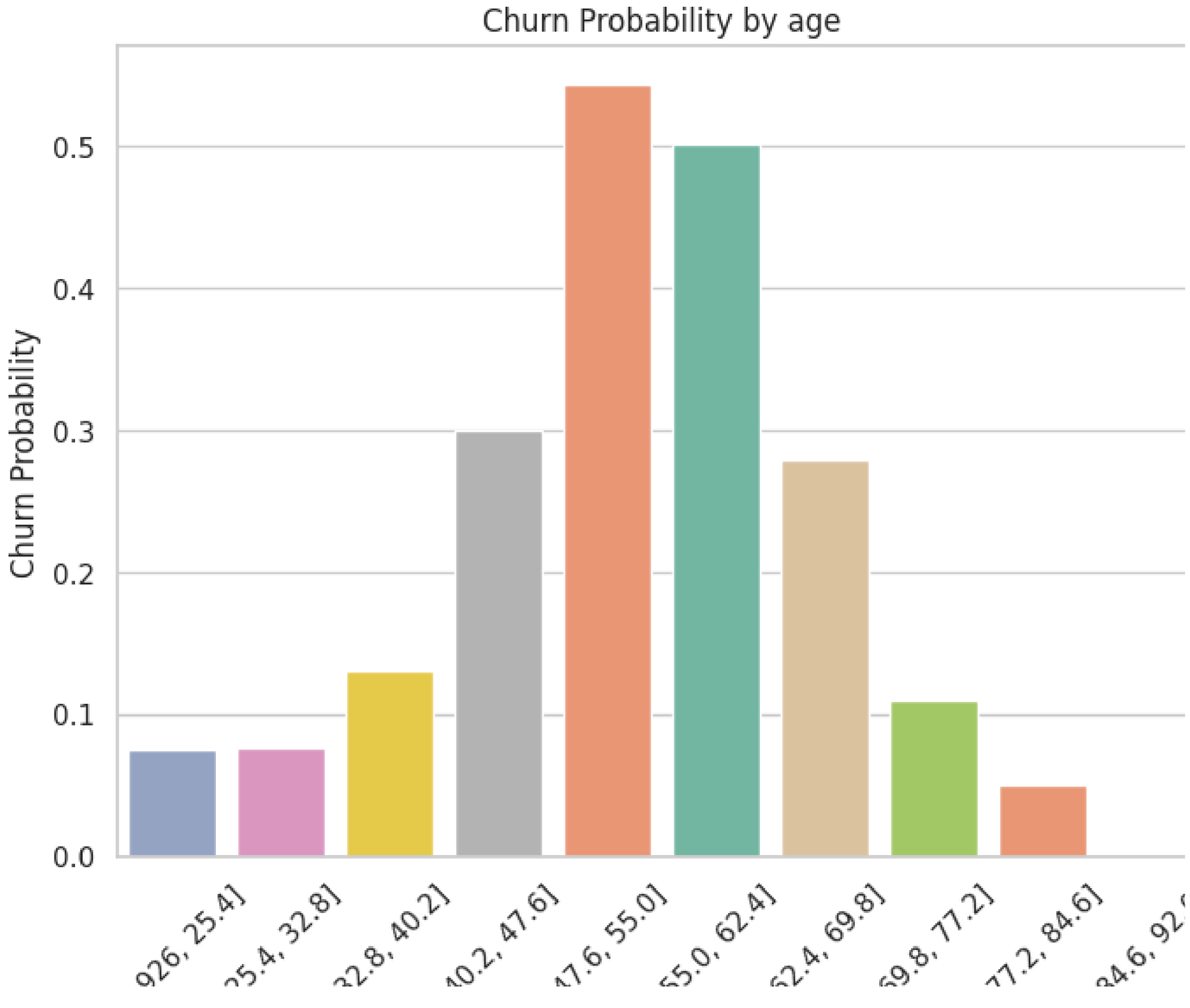

customer_id 1 0.0053 0.0095 -0.015 -0.012 0.017 -0.014 0.0017 0.015 -0.0062
credit_score 0.0053 1 -0.004 0.00084 0.0063 0.012 -0.0055 0.026 -0.0014 -0.027
age 0.0095 -0.004 1 -0.01 0.028 -0.031 -0.012 0.085 -0.0072 0.29
tenure -0.015 0.00084 -0.01 1 -0.012 0.013 0.023 -0.028 0.0078 -0.014
balance -0.012 0.0063 0.028 -0.012 1 -0.3 -0.015 -0.01 0.013 0.12
products_number 0.017 0.012 -0.031 0.013 -0.3 1 0.0032 0.0096 0.014 -0.048
credit_card -0.014 -0.0055 -0.012 0.023 -0.015 0.0032 1 -0.012 -0.0099 -0.0071
active_member 0.0017 0.026 0.085 -0.028 -0.01 0.0096 -0.012 1 -0.011 -0.16
estimated_salary 0.015 -0.0014 -0.0072 0.0078 0.013 0.014 -0.0099 -0.011 1 0.012
churn -0.0062 -0.027 0.29 -0.014 0.12 -0.048 -0.0071 -0.16 0.012 1
customer_id
credit_score
age
tenure
balance
products_number
credit_card
active_member
estimated_salary
churn
1.0
0.8
0.6
0.4
0.2
0.0
−0.2

Correlations by Country Feature

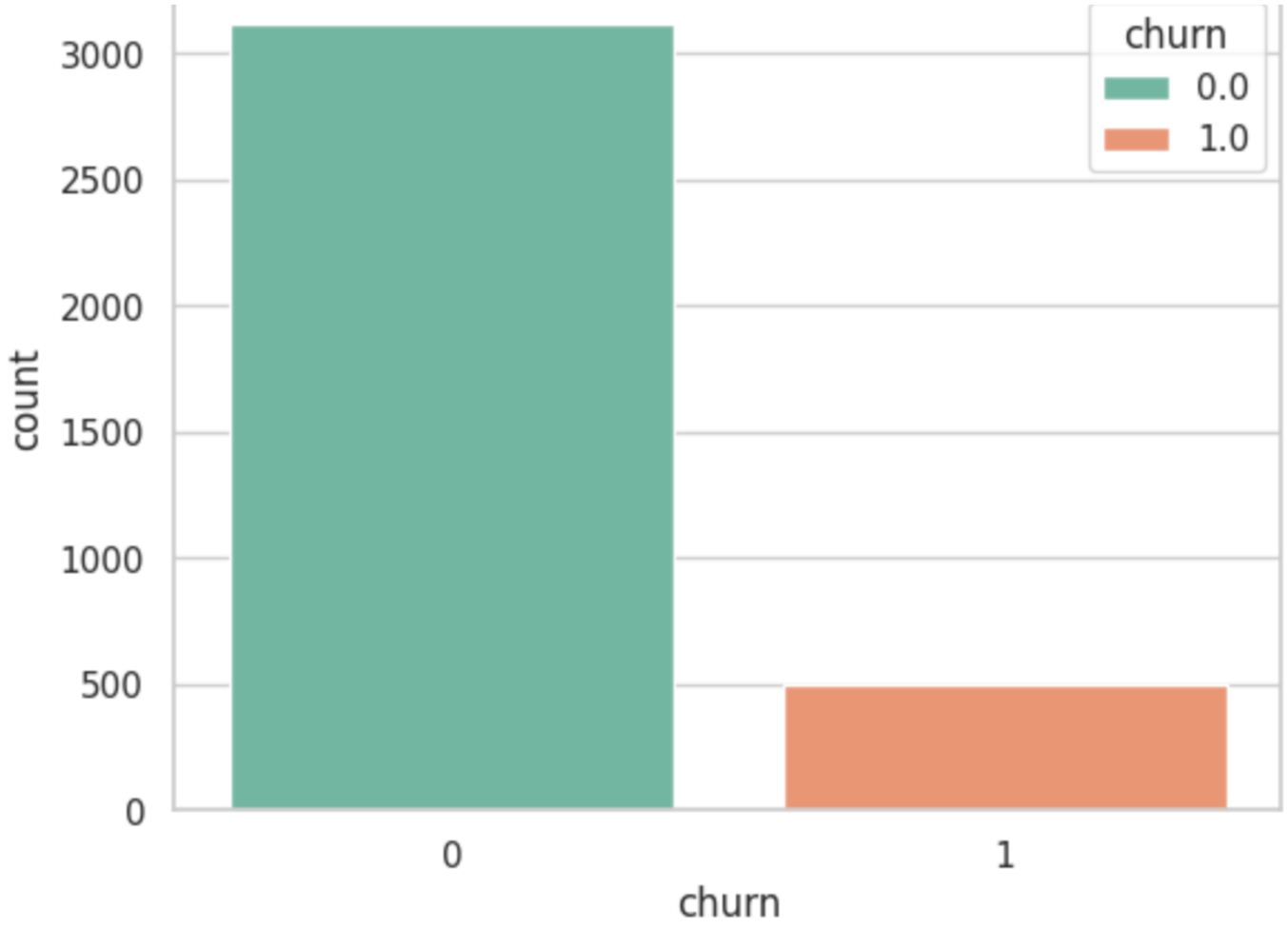
```
[24] visualize_scatter_by_column(df,"credit_score","age","country","churn")
```



Churn Probability



Churn counts



Preprocessing

```
def preprocess_workflow(dataframe,encoded_column="country",one_hot_encoder_column=False,label_encoder_column=False,column="gender"):
  """
  Perform preprocessing steps on the input dataframe.

    Parameters:
    - dataframe (pd.DataFrame): The input dataframe to be processed.
    - encoded_column (str): The column to be one-hot encoded if one_hot_encoder_column is True.
    - one_hot_encoder_column (bool): Whether to perform one-hot encoding.
    - label_encoder_column (bool): Whether to perform label encoding.
    - column (str): The column to be label encoded if label_encoder_column is True.

    Returns:
    - X (pd.DataFrame): The processed features.
    - X_train (pd.DataFrame): The training set features.
    - X_test (pd.DataFrame): The testing set features.
    - y_train (pd.Series): The training set labels.
    - y_test (pd.Series): The testing set labels.
    - y (pd.Series): The original labels.

  """
  X,y = data_split(dataframe)

  if one_hot_encoder_column:
    cat=[encoded_column]
    X = one_hot_encoder(X,cat)
```

Encoder methods

```
[45] def label_encoder_method(dataframe,column):
       """
       Encode the specified column using LabelEncoder.

         Parameters:
         - dataframe (pd.DataFrame): The input DataFrame.
         - column (str): The column to be label-encoded.

       """
       X,y =data_split(dataframe)
       lbl_enc = LabelEncoder()
       X[column] = lbl_enc.fit_transform(X[column])
     label_encoder_method(df,"gender")
```

```
def one_hot_encoder(dataframe,categoric_column,drop_first=False):

  """
  Apply one-hot encoding to the specified categorical column(s) in the DataFrame.
  - dataframe (pd.DataFrame): The input DataFrame.
  - categoric_column (str or list of str): The column(s) to be one-hot encoded.
  """
  dataframe=pd.get_dummies(data=dataframe,columns=categoric_column,drop_first=drop_first)
  return dataframe
```

Creating a Base Model

```
def random_forest_model():
    """
    Creates and trains a Random Forest model on the data.
    Before use this function be consider that you run preprocess_workflow function.

    """
    preprocess_workflow(df)
    model = RandomForestClassifier()
    model.fit(X_train, y_train)
    return model
random_model = random_forest_model()
```

Model Evaluation

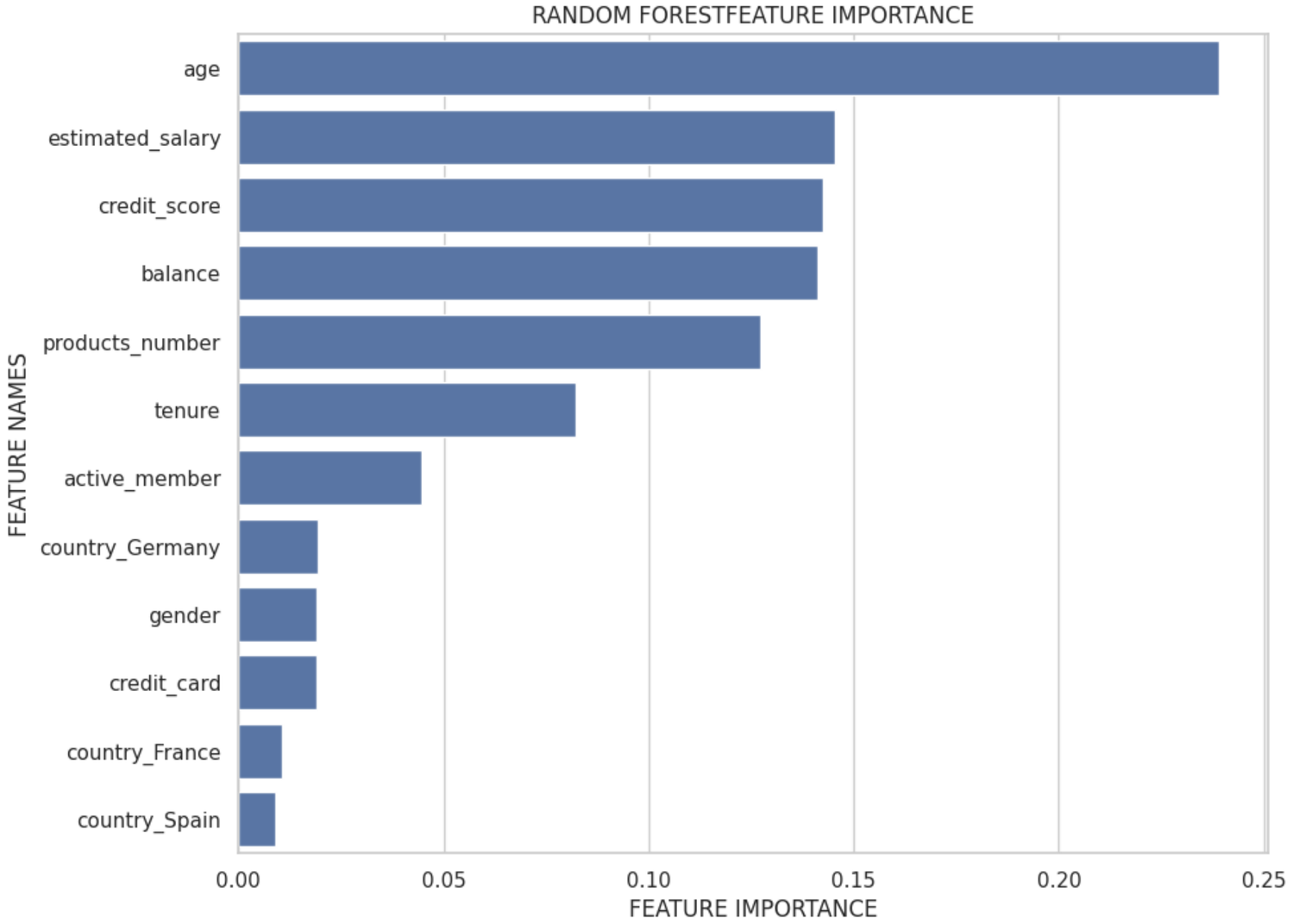
```python
def model_evaluation_metrics(model):
    print("*"*10)
    print("training scores")
    print(classification_report(y_train, model.predict(X_train)))
    print("tesitng scores")
    print(classification_report(y_test, model.predict(X_test)))
    y_pred=model.predict(X_test)
    y_pred_prob = model.predict_proba(X_test)[:,1]
    print("**********")
    print("roc_auc_score:",roc_auc_score(y_test,y_pred_prob))
    print("**********")
    print("Confuison Matrix")
    cm = confusion_matrix(y_test,y_pred)
    print(cm)
    print("**********")
    disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=[0, 1])
    disp.plot()
    plt.show()
```

```
[51] model_evaluation_metrics(random_model)
```

Model Reports

```
**********
training scores
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      6370
           1       1.00      1.00      1.00      1630

    accuracy                           1.00      8000
   macro avg       1.00      1.00      1.00      8000
weighted avg       1.00      1.00      1.00      8000

tesitng scores
              precision    recall  f1-score   support

           0       0.87      0.97      0.92      1593
           1       0.77      0.44      0.56       407

    accuracy                           0.86      2000
   macro avg       0.82      0.70      0.74      2000
weighted avg       0.85      0.86      0.84      2000

************
roc_auc_score: 0.8505678251440962
************
Confuison Matrix
[[1540   53]
 [ 228  179]]
```

Selecting Feature



Crreating Base Models

```
[63] def base_models(X, y, scoring="accuracy"):
        """
        Base models gives us accuracy with all models
        """
        print("Base Models....")
        models = [('LR', LogisticRegression()),
                        ('KNN', KNeighborsClassifier()),
                        ("SVC", SVC()),
                        ("DT", DecisionTreeClassifier()),
                        ("RF", RandomForestClassifier()),
                        ('GBM', GradientBoostingClassifier())

                        ]

        for name, classifier in models:
            cv_results = model_selection.cross_validate(classifier, X, y, cv=3, scoring=scoring)
            print(f"{scoring}: {round(cv_results['test_score'].mean(), 4)} ({name}) ")

    base_models(X, y, scoring="accuracy")
```

```
Base Models....
accuracy: 0.7946 (LR)
accuracy: 0.7603 (KNN)
accuracy: 0.7963 (SVC)
accuracy: 0.7562 (DT)
accuracy: 0.8344 (RF)
accuracy: 0.8419 (GBM)
```

Creating GBM Model

```
def gbm_production_model(dataframe,learning_rate=0.1,max_depth=3,min_samples_split=2,n_estimators=100):
  dataframe = dataframe.select_dtypes(["int","float"])
  """
  Create and evaluate a Gradient Boosting Machine (GBM) model for production using specified parameters.

    Parameters:
    - dataframe (DataFrame): The dataset for model training and evaluation.
   Returns:
    - GradientBoostingClassifier: The GBM production model.
    - DataFrame: Features of the training set.
    - DataFrame: Features of the testing set.
    - Series: Labels of the training set.
    - Series: Labels of the testing set.
    - Series: All label
    - Series: Predicted
  """
  X,y = data_split(data
  X,X_train,X_test,y_tr                                                    w(dataframe)
  production_model = Gr                                                    _rate=learning_rate,max_depth=max_depth,min_samples_split=min_samples_split,n_es
  production_model.fit(X_train,y_train)
  y_pred = production_model.predict(X_test)
  return production_model,X_train,X_test,y_train,y_test,X,y,y_pred
```

DataFrame: X_train

Generate

View

DataFrame with shape (8000, 6)

Metrics

```
training scores
              precision    recall  f1-score   support

           0       0.87      0.97      0.92      6370
           1       0.77      0.44      0.56      1630

    accuracy                           0.86      8000
   macro avg       0.82      0.70      0.74      8000
weighted avg       0.85      0.86      0.84      8000

tesitng scores
              precision    recall  f1-score   support

           0       0.86      0.96      0.91      1593
           1       0.71      0.37      0.48       407

    accuracy                           0.84      2000
   macro avg       0.78      0.66      0.69      2000
weighted avg       0.83      0.84      0.82      2000

************
roc_auc_score: 0.8262067923084873
************
Confuison Matrix
[[1532   61]
 [ 258  149]]
```

Hyperparameter-optimization

```
[69] knn_params = {"n_neighbors": range(2, 50)}

     cart_params = {'max_depth': range(1, 20),
                    "min_samples_split": range(2, 30)}

     rf_params = {"max_depth": [8, 15, None],
                  "max_features": [5, 7, "auto"],
                  "min_samples_split": [15, 20],
                  "n_estimators": [200, 300]}


     gbm_params = {"learning_rate": [0.01, 0.1, 0.2],
                   "n_estimators": [100, 200, 300],
                   "max_depth": [3, 5, 7],
                   "min_samples_split": [2, 5, 10]}

     models = [("KNN", KNeighborsClassifier(), knn_params),
               ("CART",DecisionTreeClassifier(), cart_params),
               ("RF",  RandomForestClassifier(), rf_params),
               ("GBM", GradientBoostingClassifier(), gbm_params)]
```

Getting Best Parameters

```
best_models = hyperparameter_optimization(X, y)
```

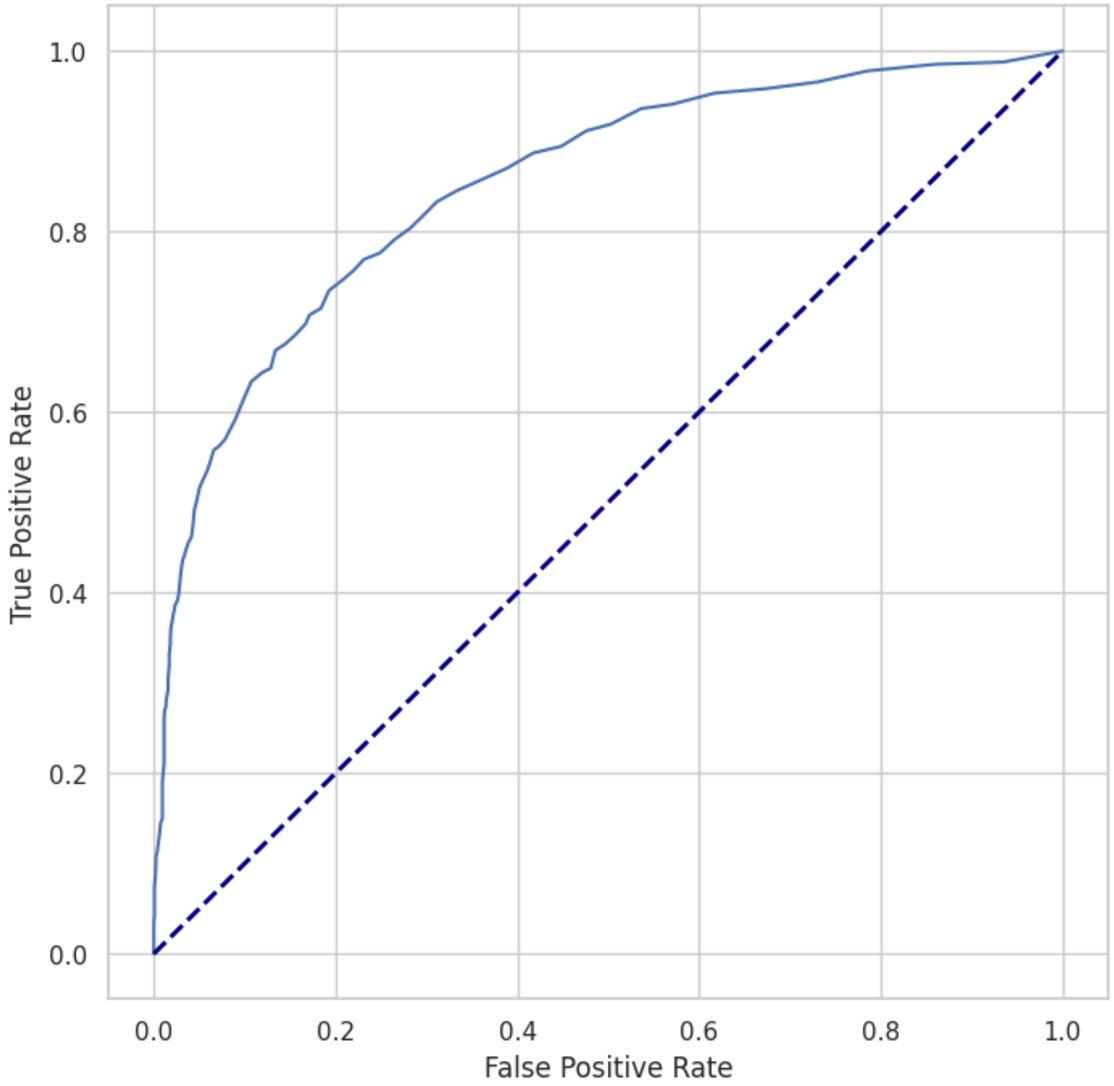
```
Hyperparameter Optimization....
########## KNN ##########
roc_auc (Before): 0.5246
roc_auc (After): 0.5598
KNN best params: {'n_neighbors': 43}

########## CART ##########
roc_auc (Before): 0.6359
roc_auc (After): 0.8196
CART best params: {'max_depth': 5, 'min_samples_split': 2}

########## RF ##########
roc_auc (Before): 0.807
roc_auc (After): 0.8309
RF best params: {'max_depth': 8, 'max_features': 'auto', 'min_samples_split': 20, 'n_estimators': 300}

########## GBM ##########
roc_auc (Before): 0.8285
roc_auc (After): 0.8305
GBM best params: {'learning_rate': 0.01, 'max_depth': 5, 'min_samples_split': 10, 'n_estimators': 300}
```

Model Evaluation After Best Parameters

```
model_evaluation_metrics(production_model)
```

```
**********
training scores
              precision    recall  f1-score   support

           0       0.87      0.96      0.91      6370
           1       0.73      0.46      0.56      1630

    accuracy                           0.85      8000
   macro avg       0.80      0.71      0.74      8000
weighted avg       0.84      0.85      0.84      8000

tesitng scores
              precision    recall  f1-score   support

           0       0.86      0.95      0.91      1593
           1       0.69      0.40      0.51       407

    accuracy                           0.84      2000
   macro avg       0.77      0.68      0.71      2000
weighted avg       0.83      0.84      0.82      2000

************
roc_auc_score: 0.8289514475955153
************
```

Roc Curve


1.0
0.8
0.6
0.4
0.2
0.0
True Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate

Save and Load models

```
[76] joblib.dump(production_model, 'gbm_model_production.joblib')
```

```
['gbm_model_production.joblib']
```

```
loaded_model = joblib.load('gbm_model_production.joblib')
```

Predicitons

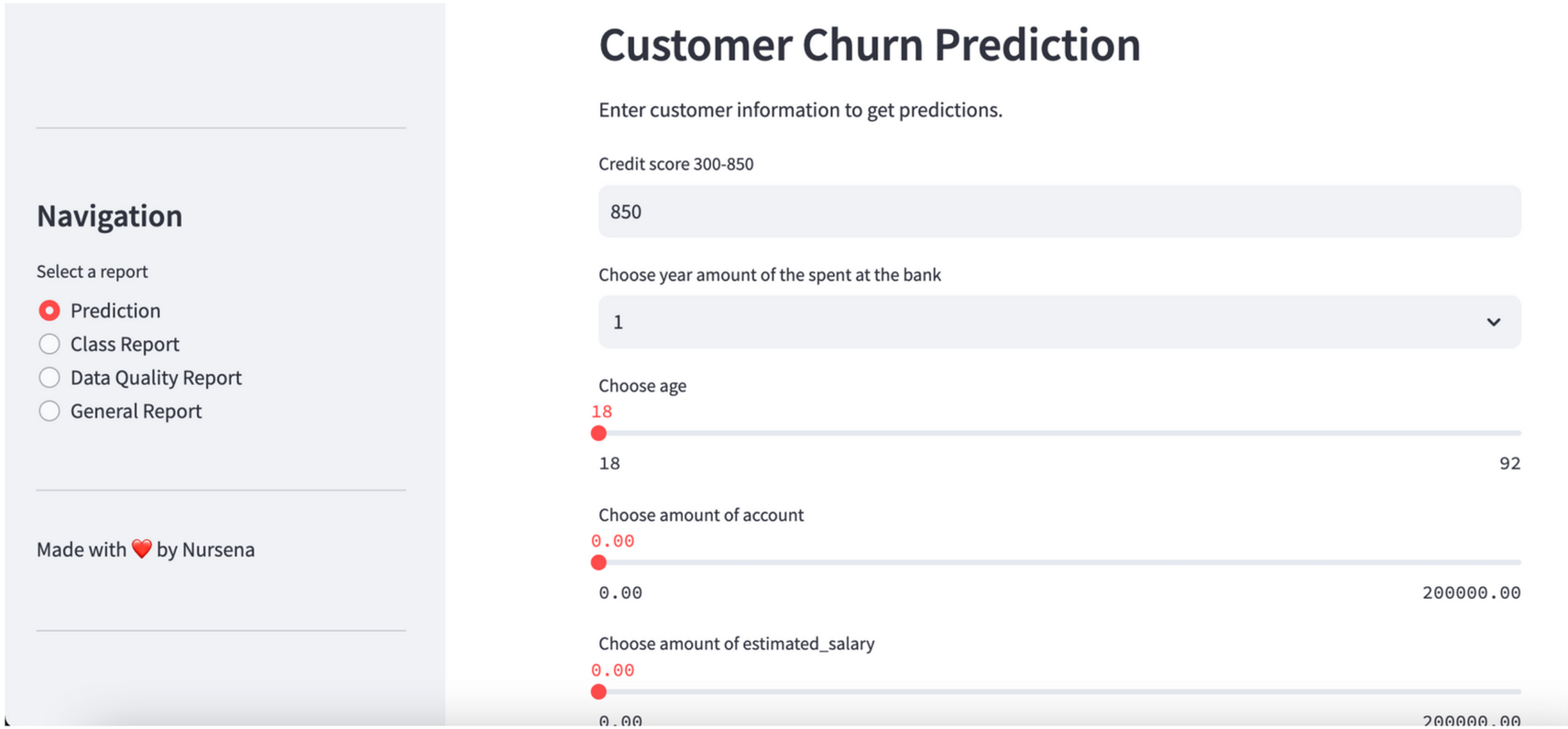
```
[78] new_data=[600, 19, 1000.0, 1,10000,2]
     new_=np.array(new_data).reshape(1,-1)
```

```
[79] prediction_gbm_model=loaded_model.predict(new_)
     prediction_gbm_model[0]
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: Use
  warnings.warn(
0
```

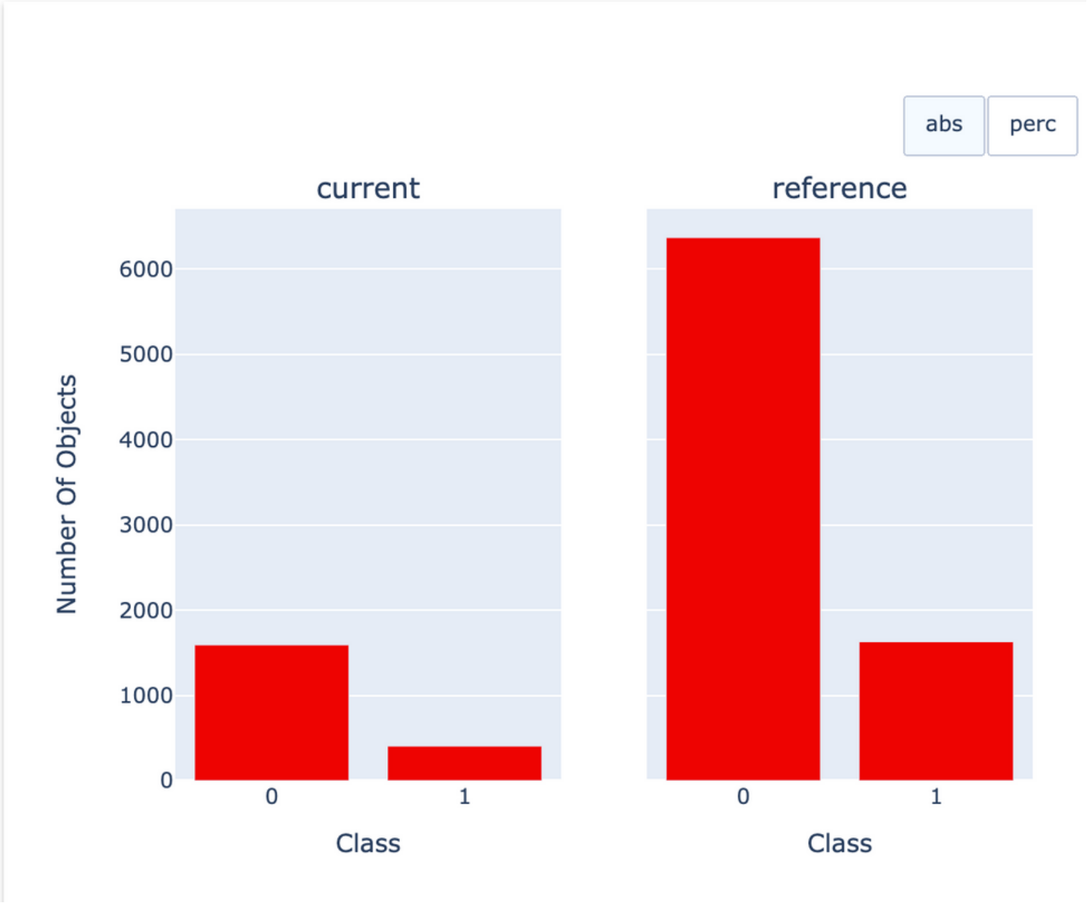
Deployment Data

Streamlit Website

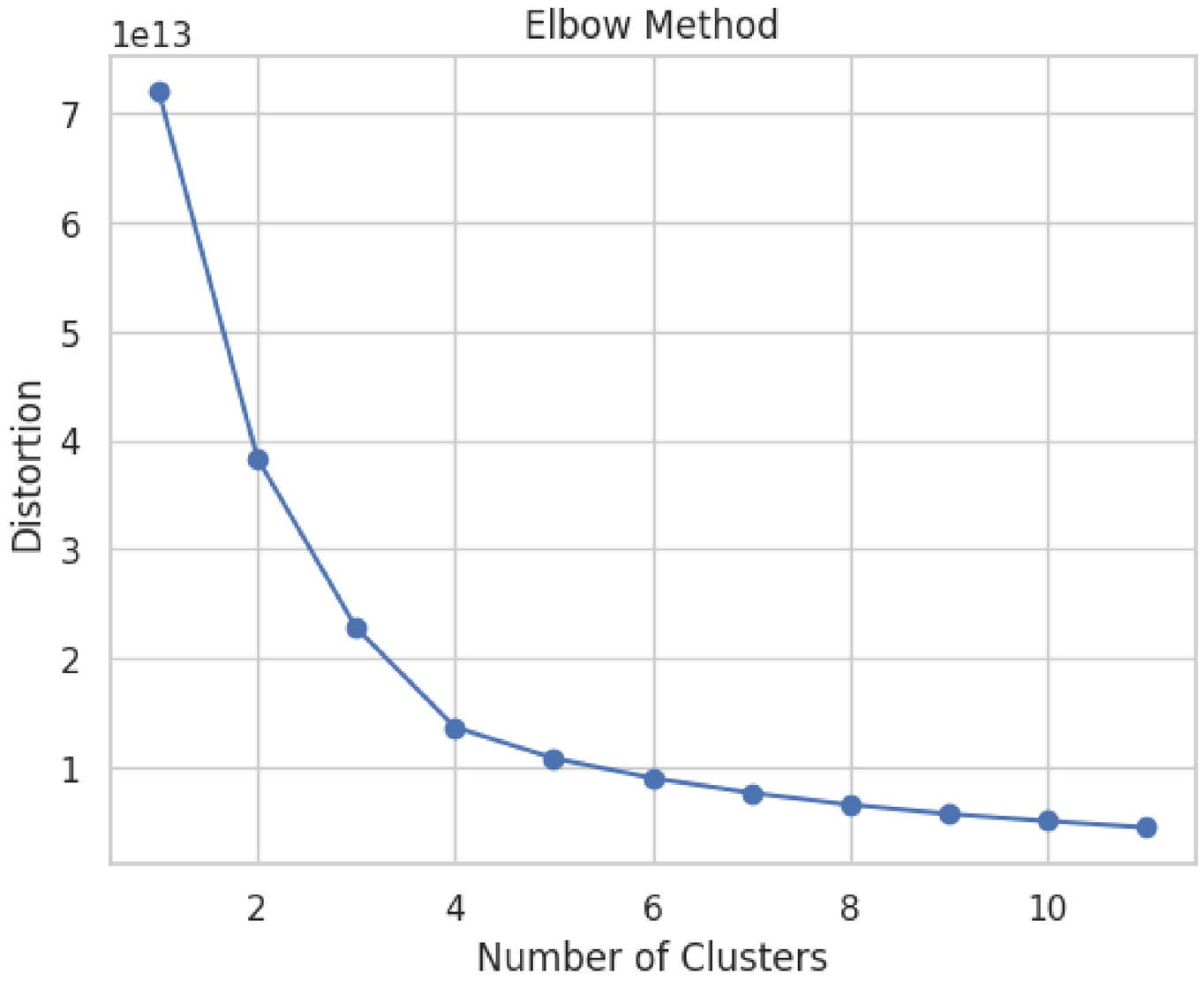


Monitoring

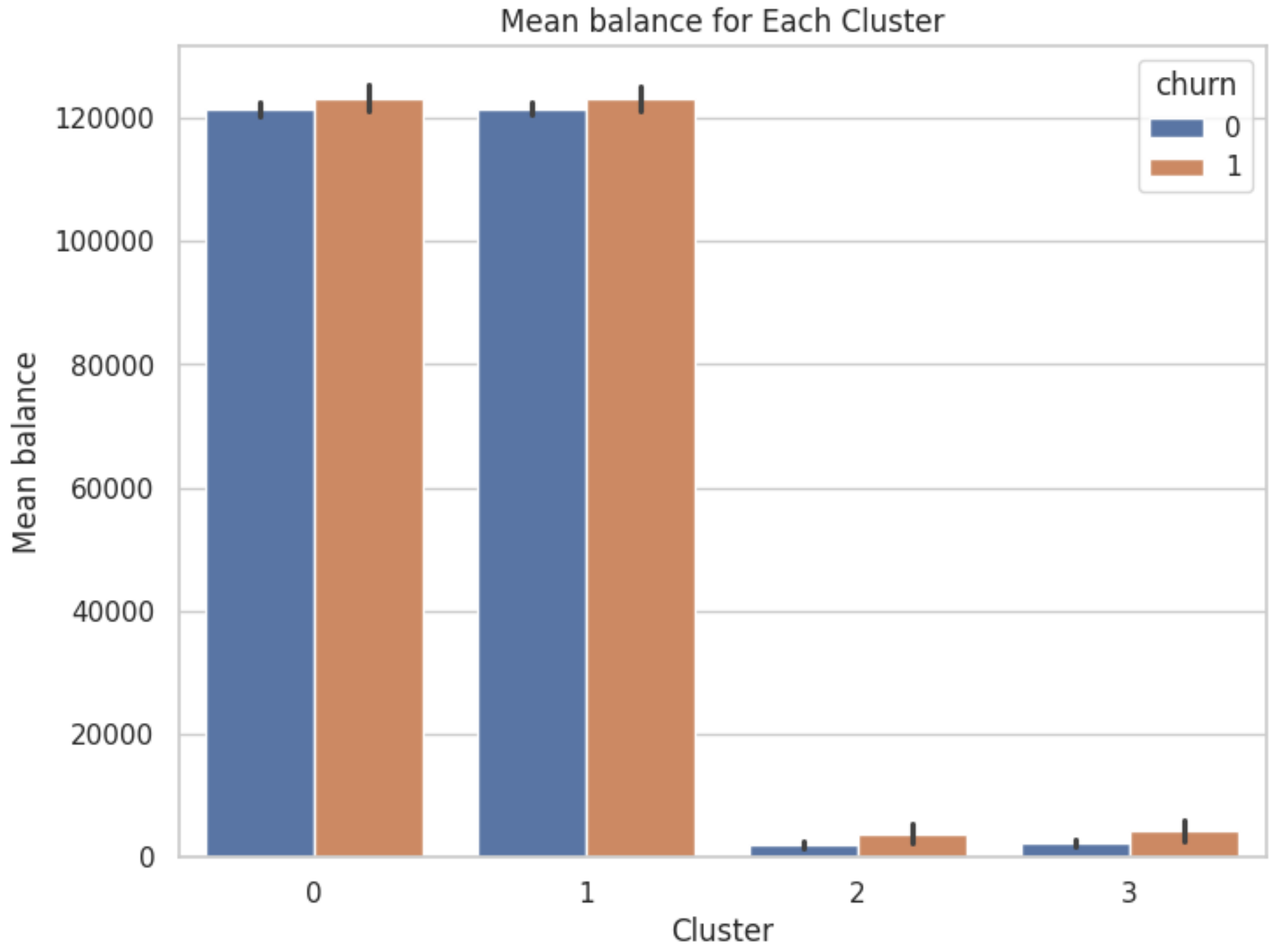
EVIDENTLY AI



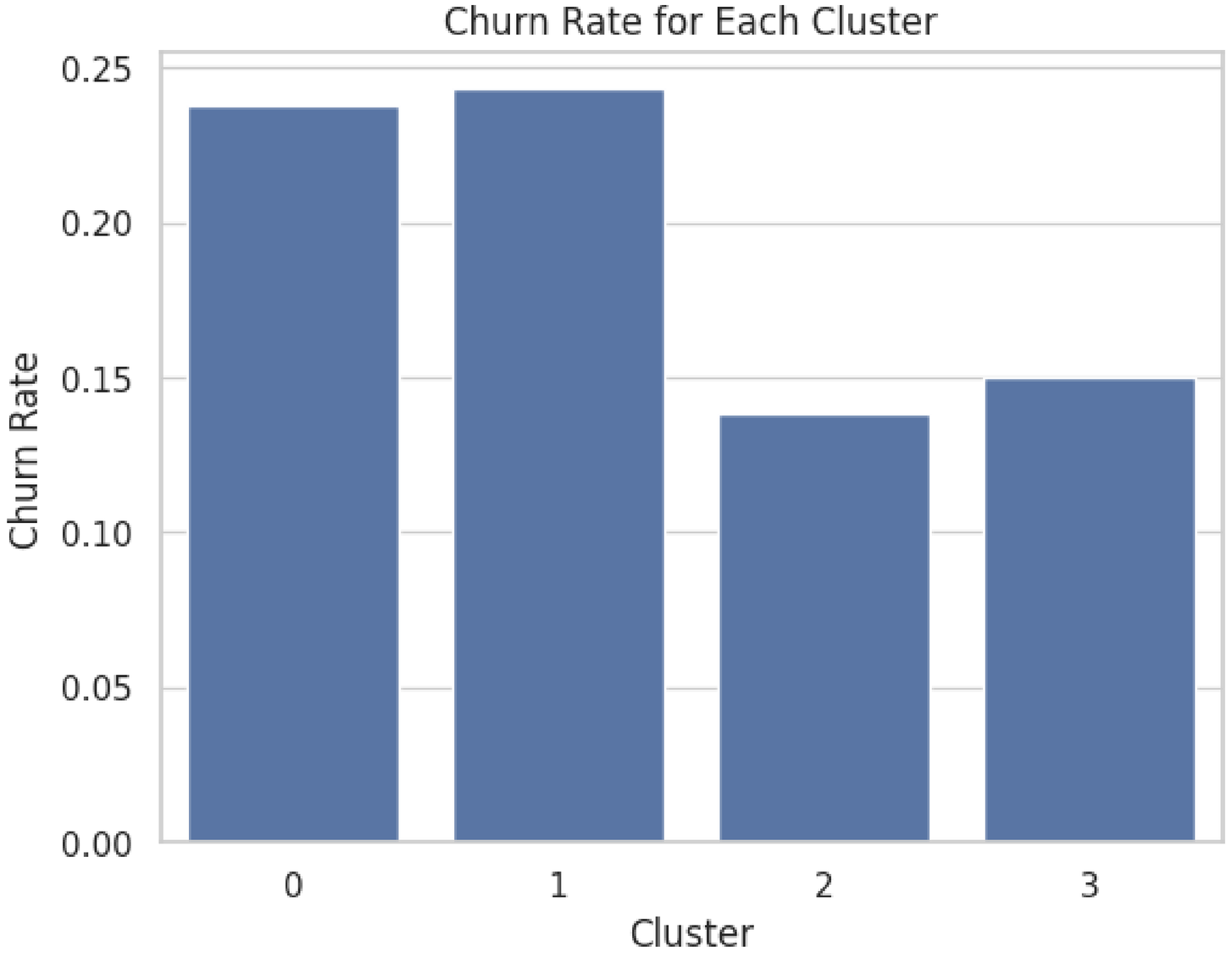
Elbow method Cluster



Churn by Balance Mean



Churn Mean


Churn Rate for Each Cluster
Churn Rate
0.25
0.20
0.15
0.10
0.05
0.00
0
1
2
3
Cluster

Teşekkür Ederim

- Projenin ana hedefleri doğrultusunda başarılı sonuçlar elde ettik.
- Tahmin modelimiz, müşteri churn'ünü öngörmede yüksek doğruluk ve başarı elde etti.
- Modelin performansını artırmak adına daha fazla öznitelik eklemeyi düşünebiliriz.
- Veri temizleme aşamasında karşılaşılan bazı zorluklar, gelecekteki projelerde daha etkili bir veri temizleme süreci geliştirmemize yol açabilir.
- Modelin iş süreçlerine entegrasyonu sırasında yaşanan bazı zorluklar vardı, bu konuda daha fazla iyileştirme yapabiliriz.
- Modelin doğruluğunu ve performansını artırmak adına yeni teknikler ve yöntemler denemeye açık olmalıyız.