Churn Analysis

Daniel, López Coto

Main Goals

Customer retention

Getting insights from data analysis

Univariate Variable exploration

Number of features: 99

Numeric: 78

Categorical: 21

Target variable: Churn

1: Leave

0: Stays

Company stop billing about
$ 2,9 million/month due to churn.

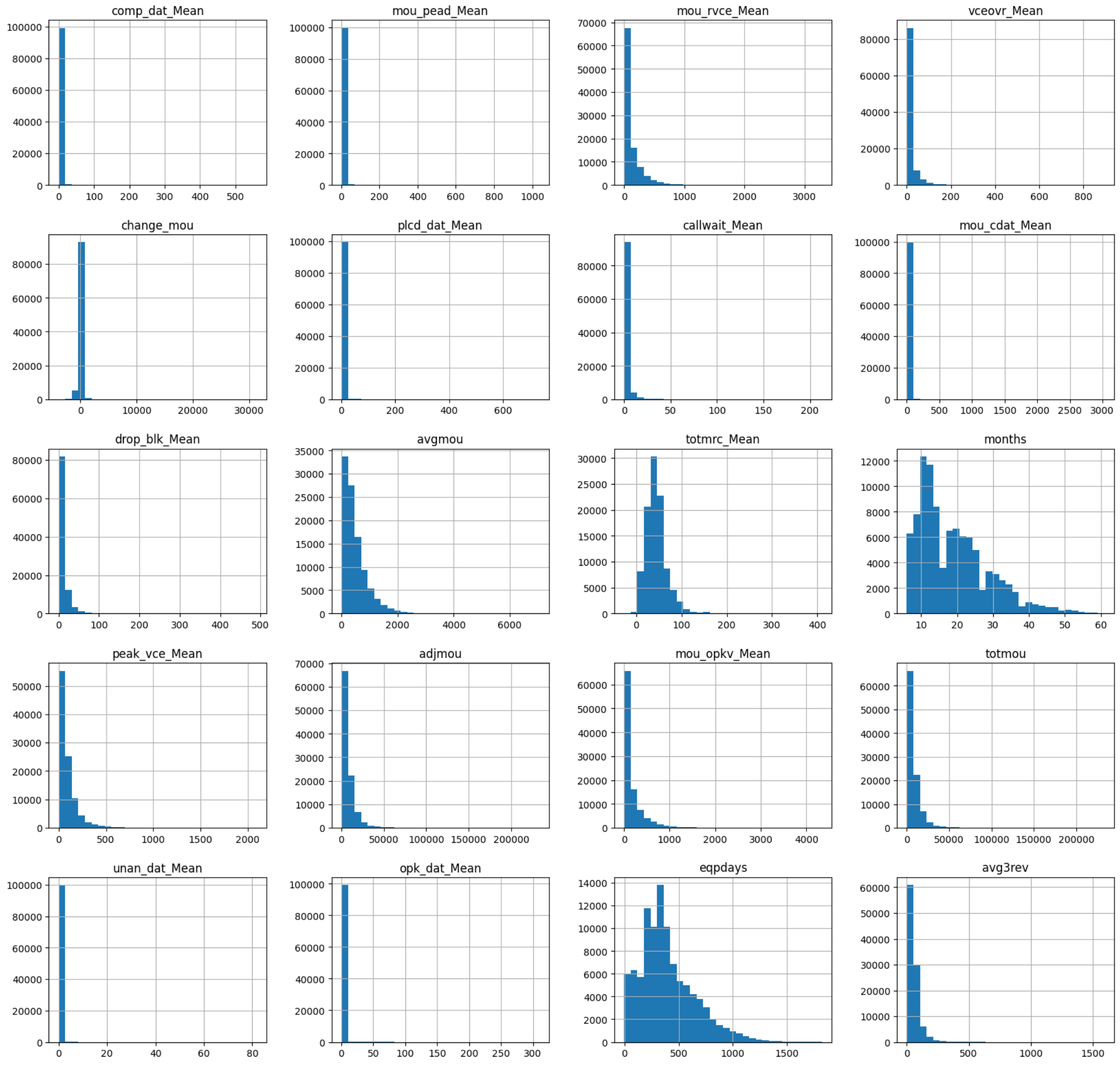
Missing values: 43 features.

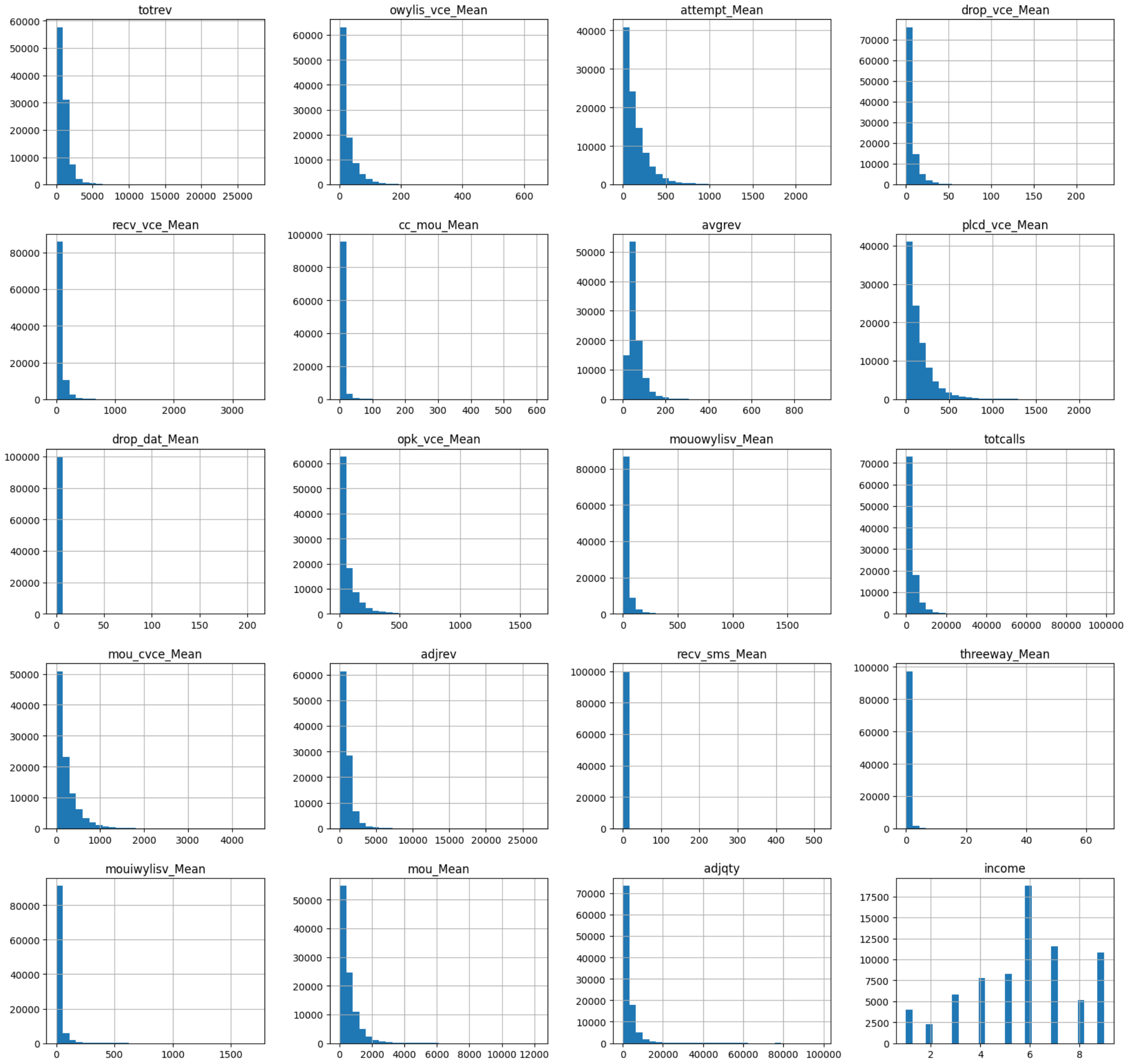
10 features with > 10% of missing values

numbcars	0.49366
dwllsize	0.38308
HHstatin	0.37923
ownrent	0.33706
dwlltype	0.31909
lor	0.30190
income	0.25436
adults	0.23019
infobase	0.22079
hnd_webcap	0.10189

Univariate Variable exploration

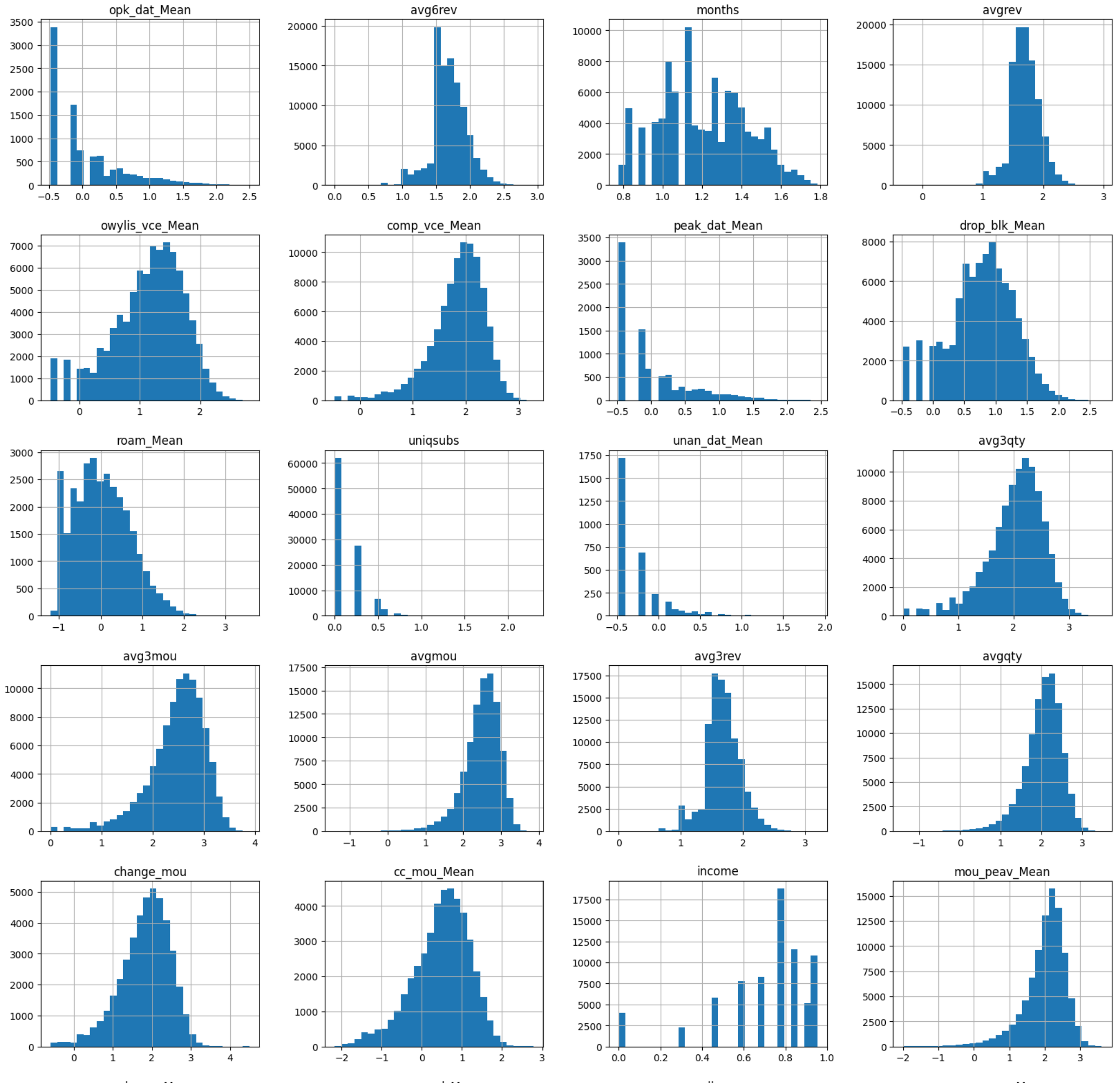
Distribution shapes

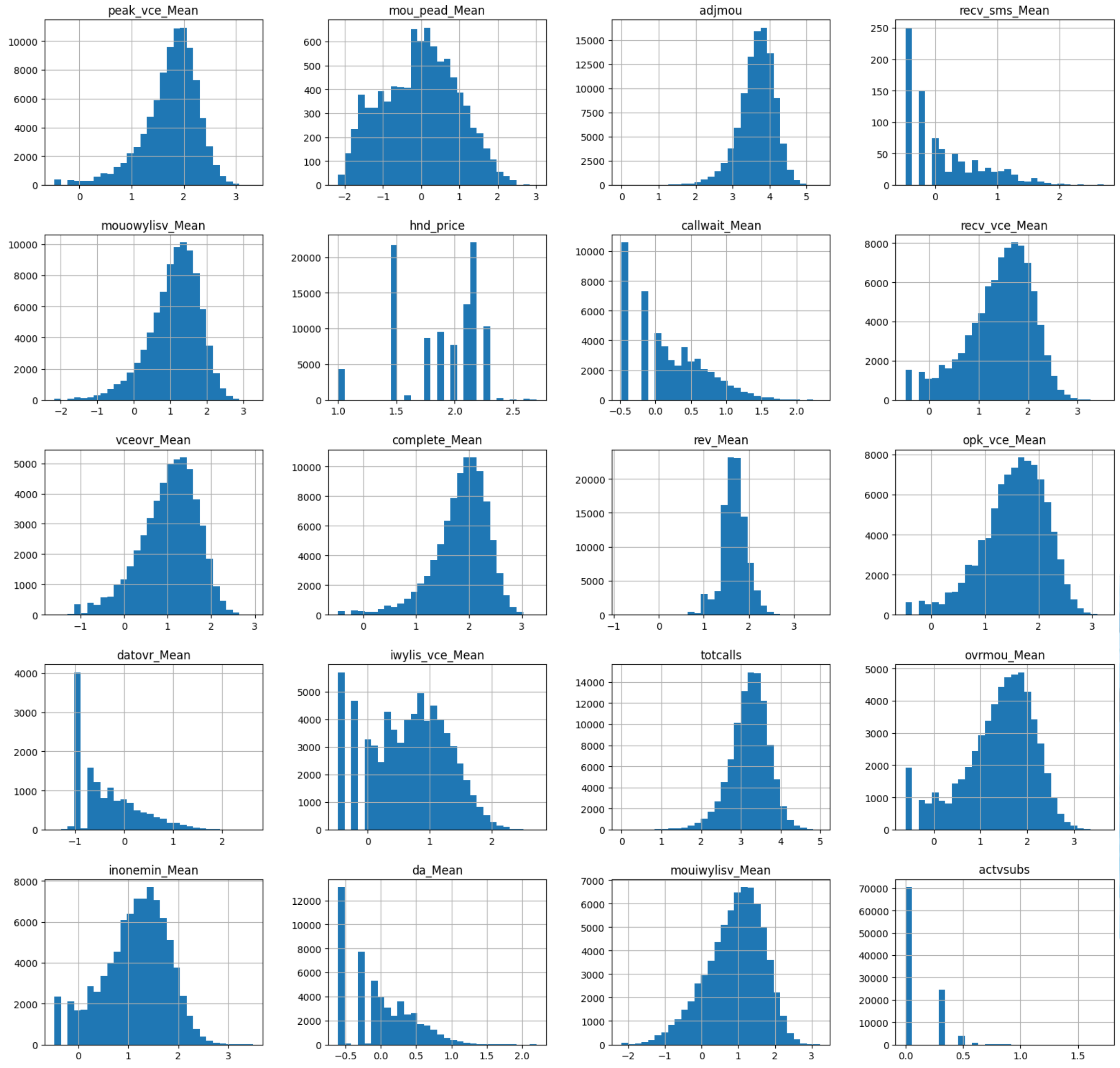




Univariate Variable exploration

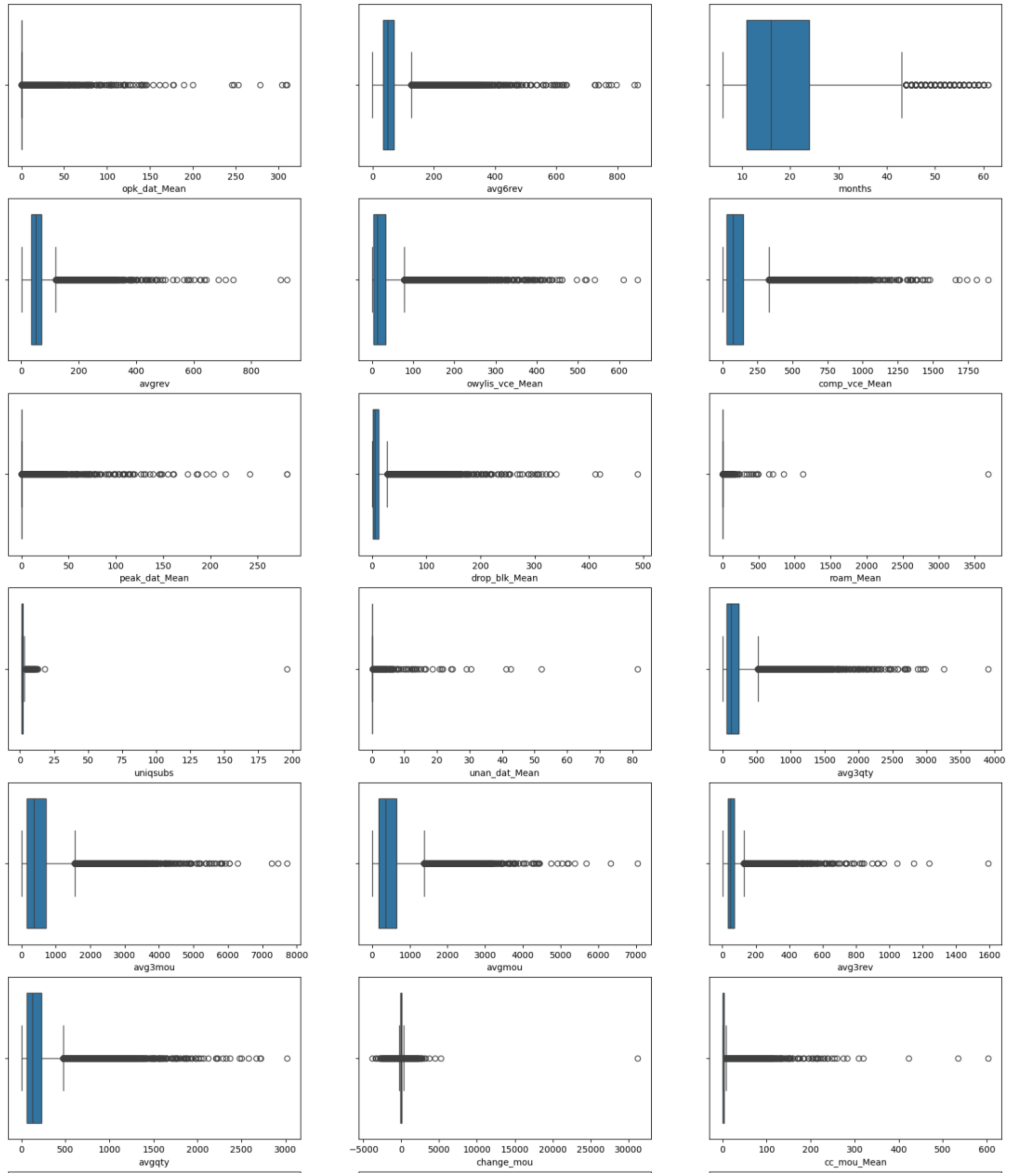
Distribution shapes (log transformed)

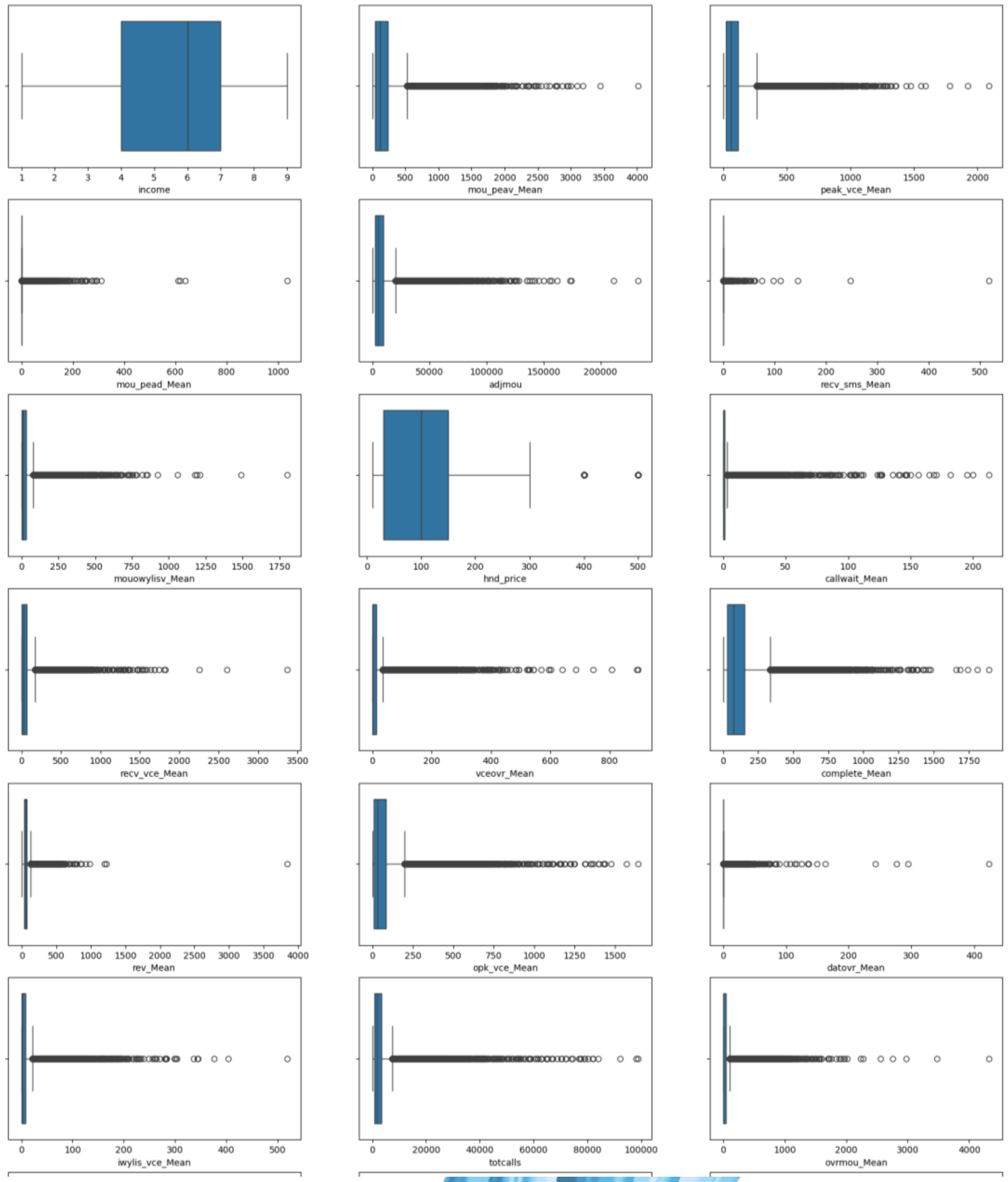




Univariate Variable exploration

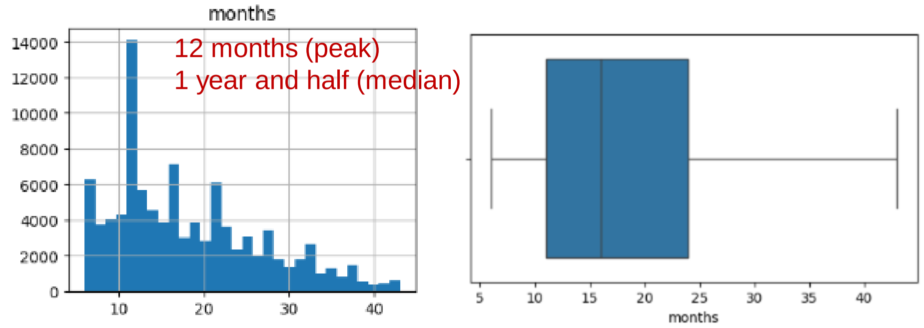
Outlier inspection

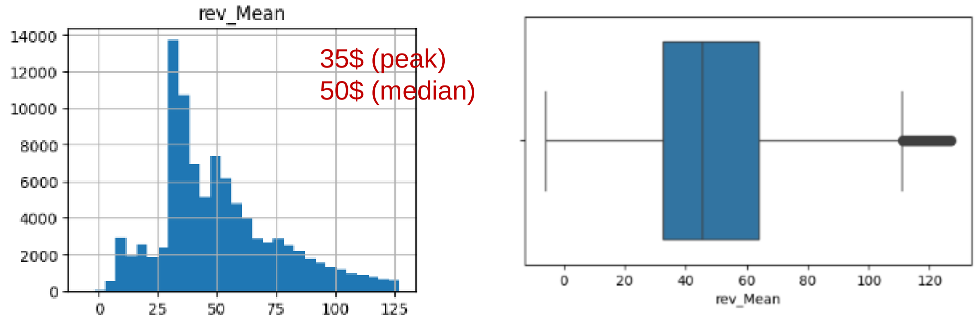


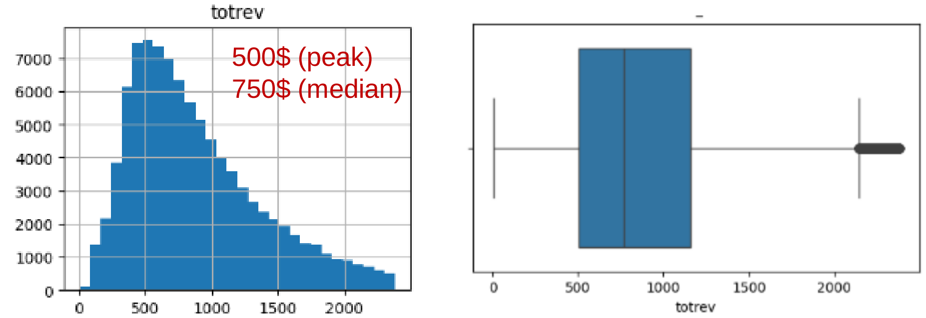


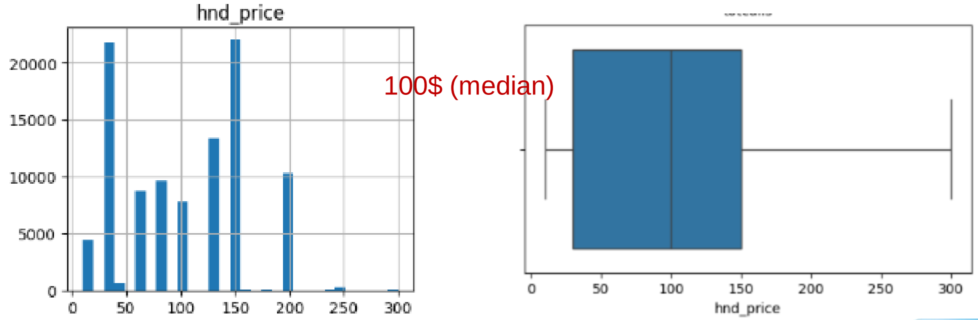
Univariate Variable exploration

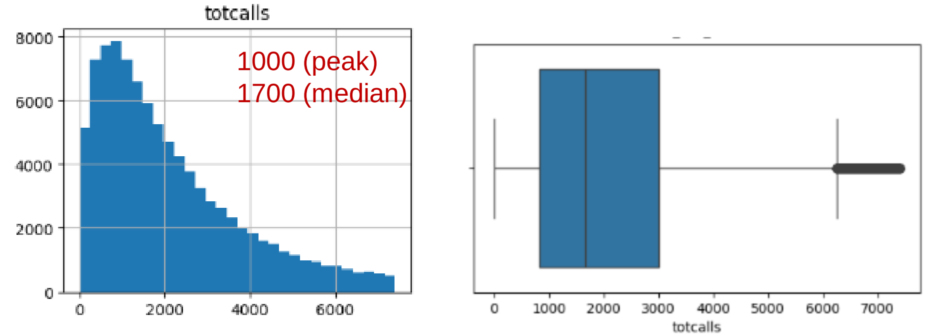
After removing outliers

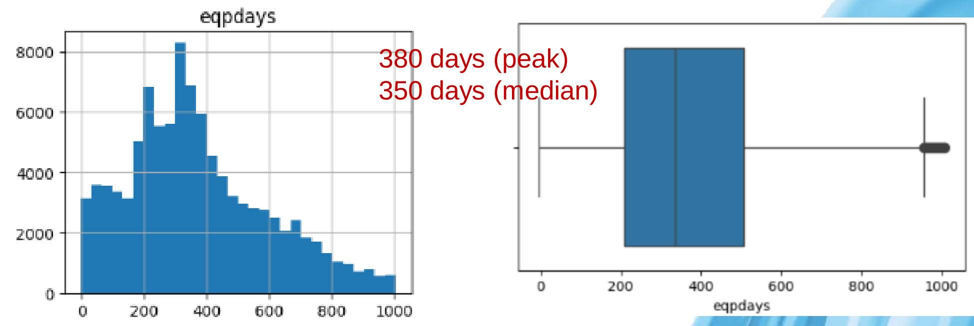












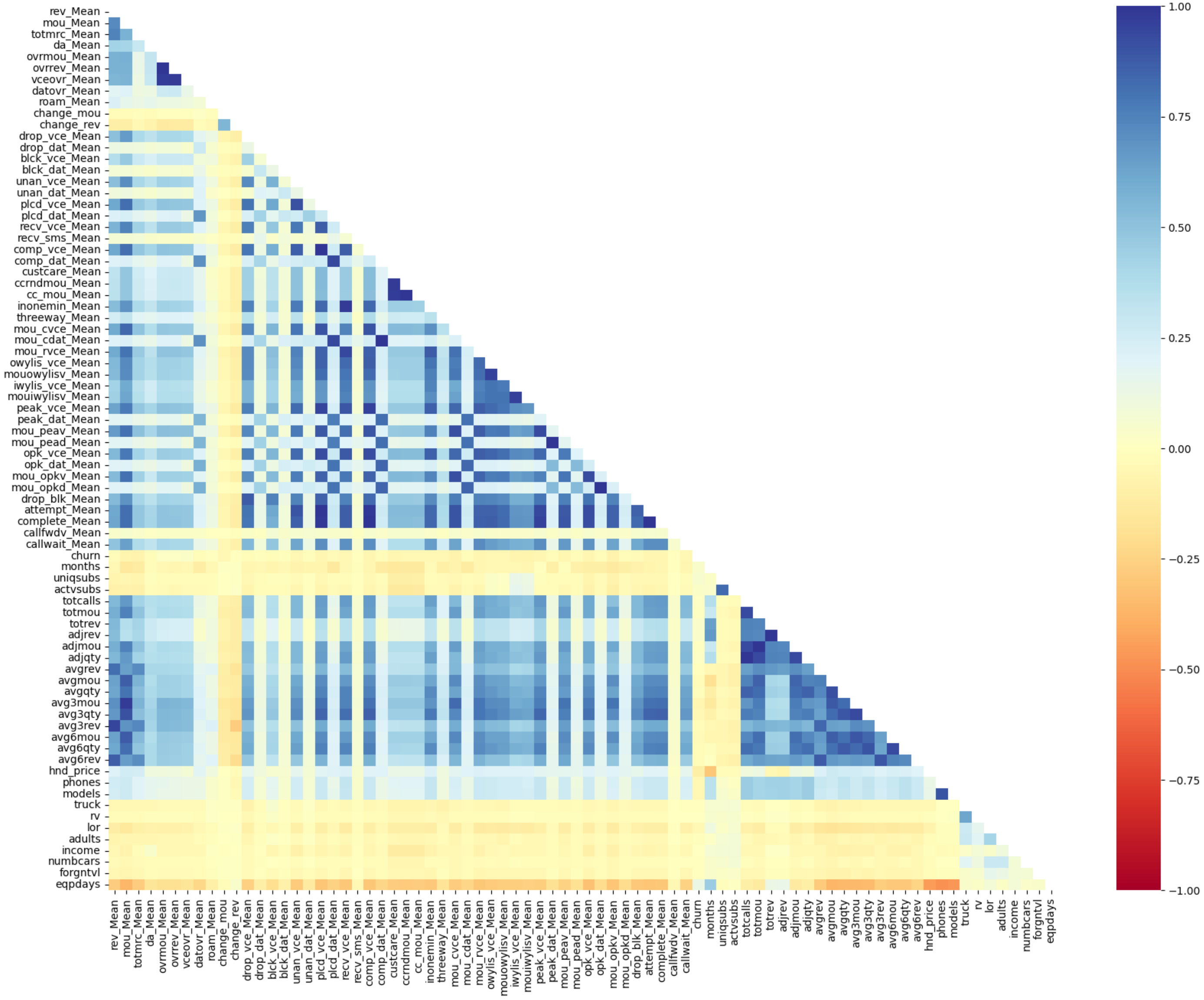
Univariate Variable exploration

Correlation coefficient

Lot of features highly correlated.

Features related with calls
(minutes of use, revenue, voice-data calls, etc)
represent same type of information.

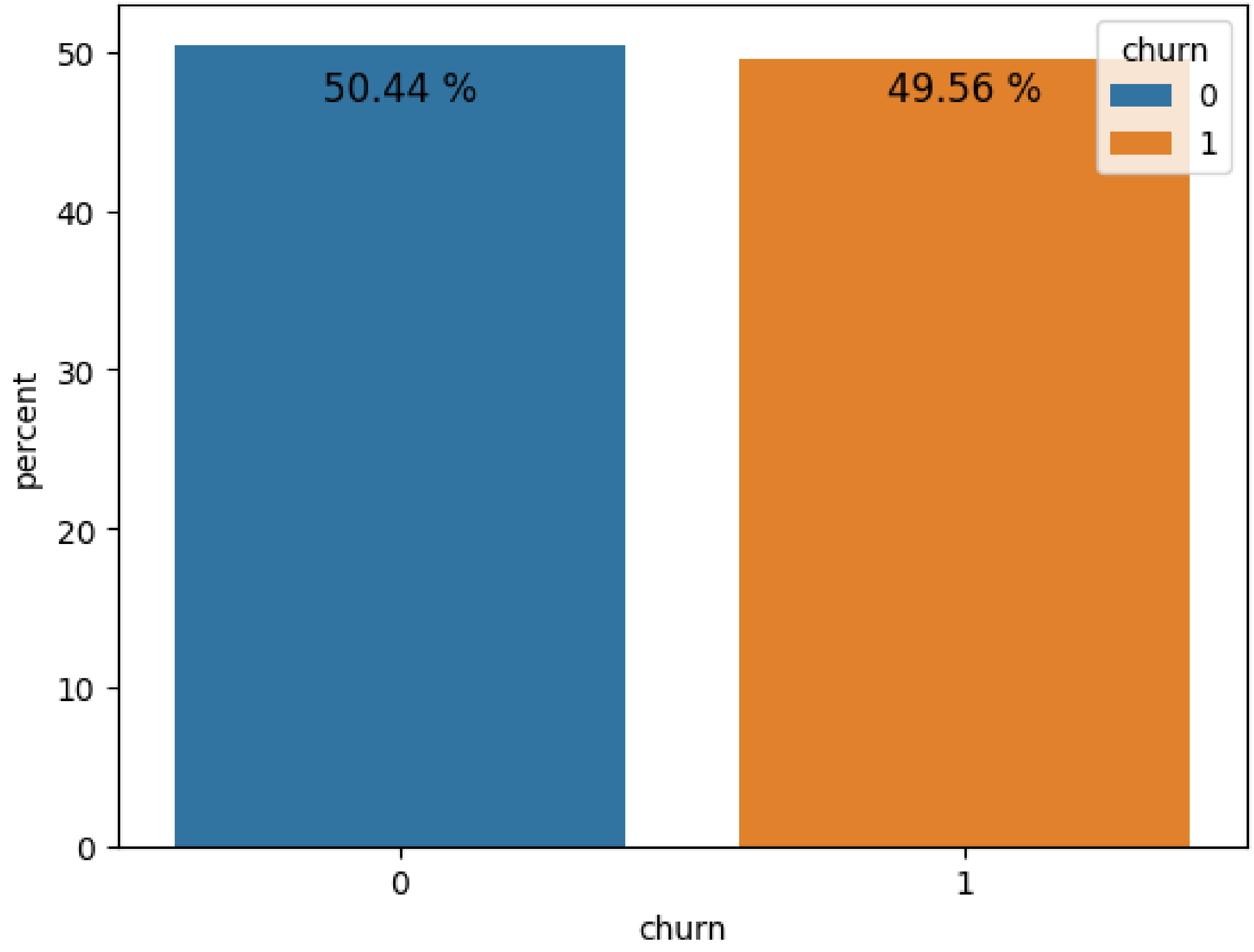
This indicate that these variables have to be
removed or transformed.



Univariate Variable exploration

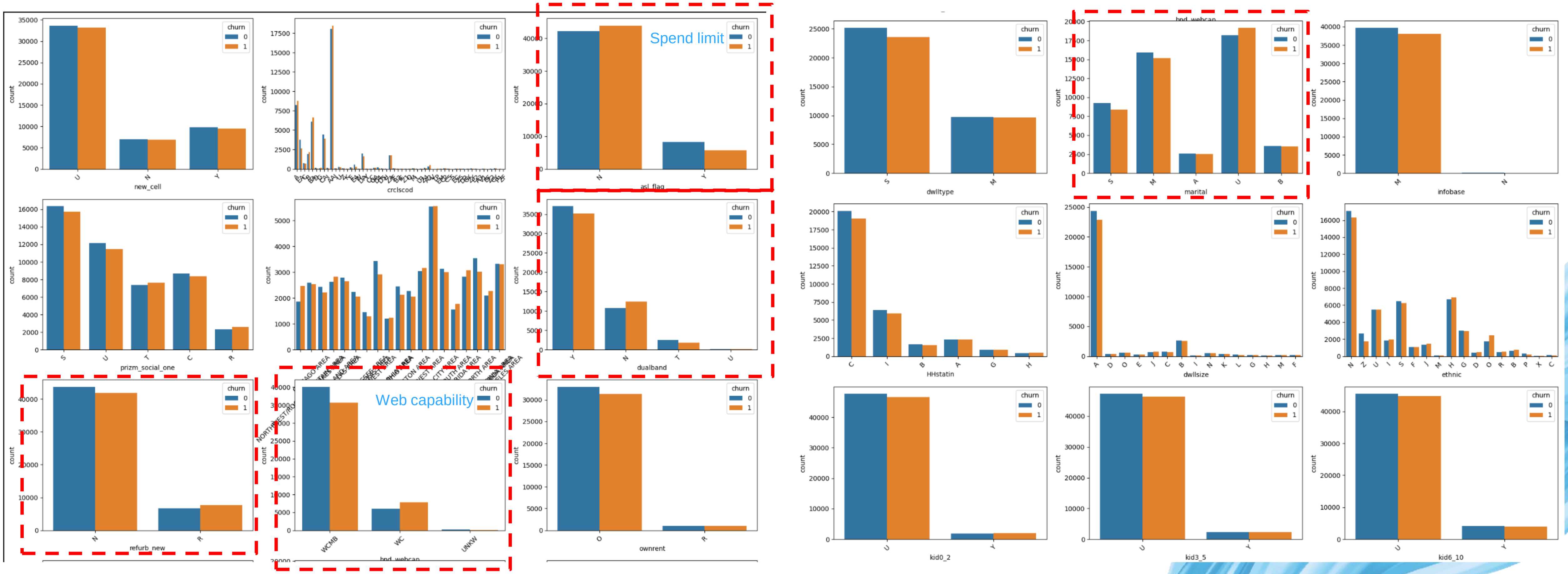
Target variable: Churn

Churn ratio: Not imbalanced target



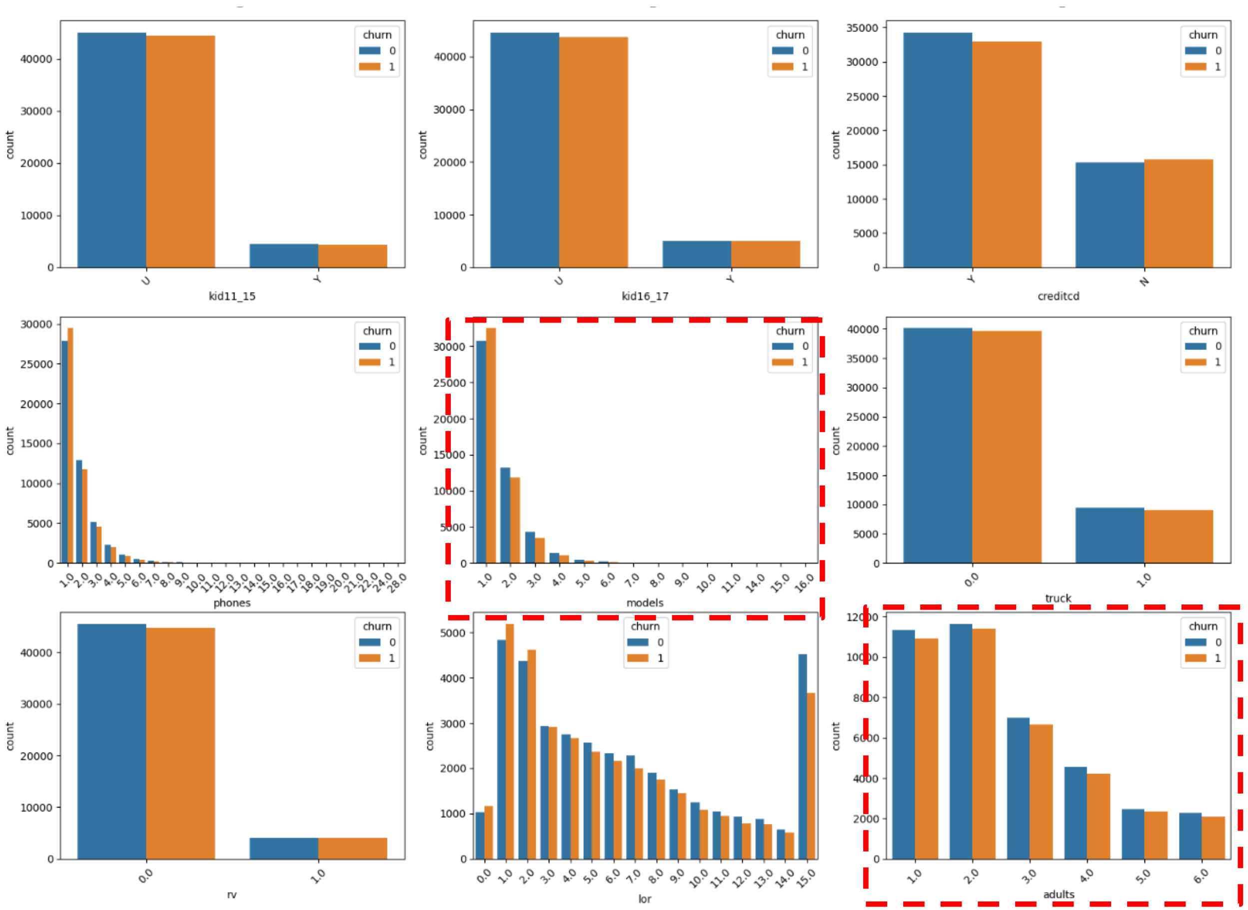
Churn-features relationship

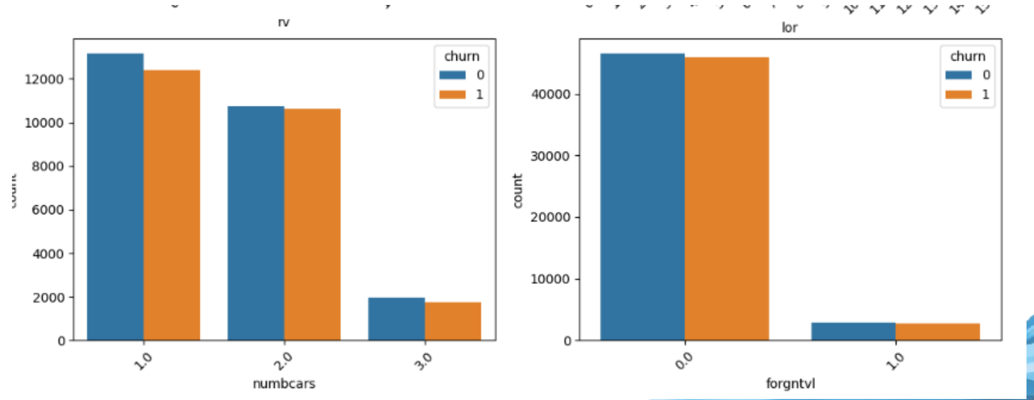
Categorical variables



Churn-features relationship

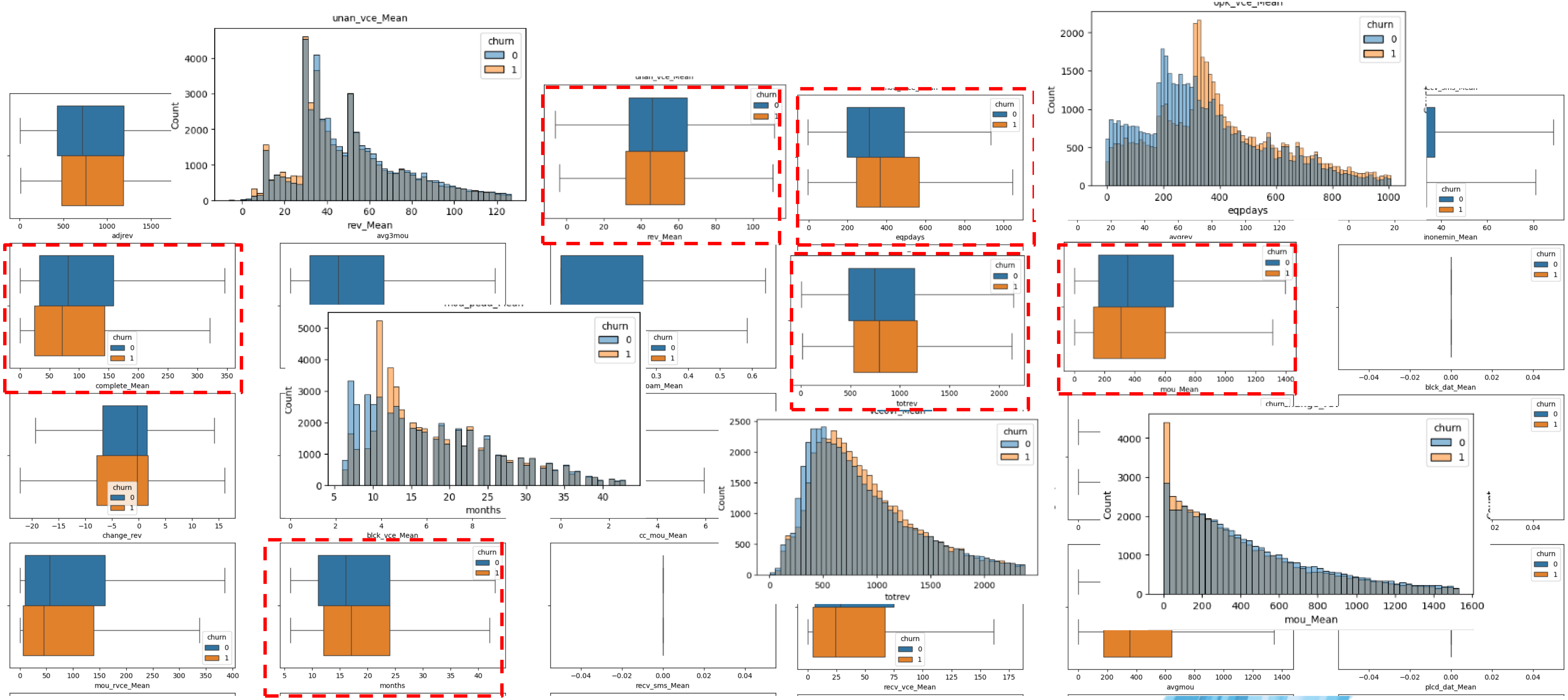
Categorical variables





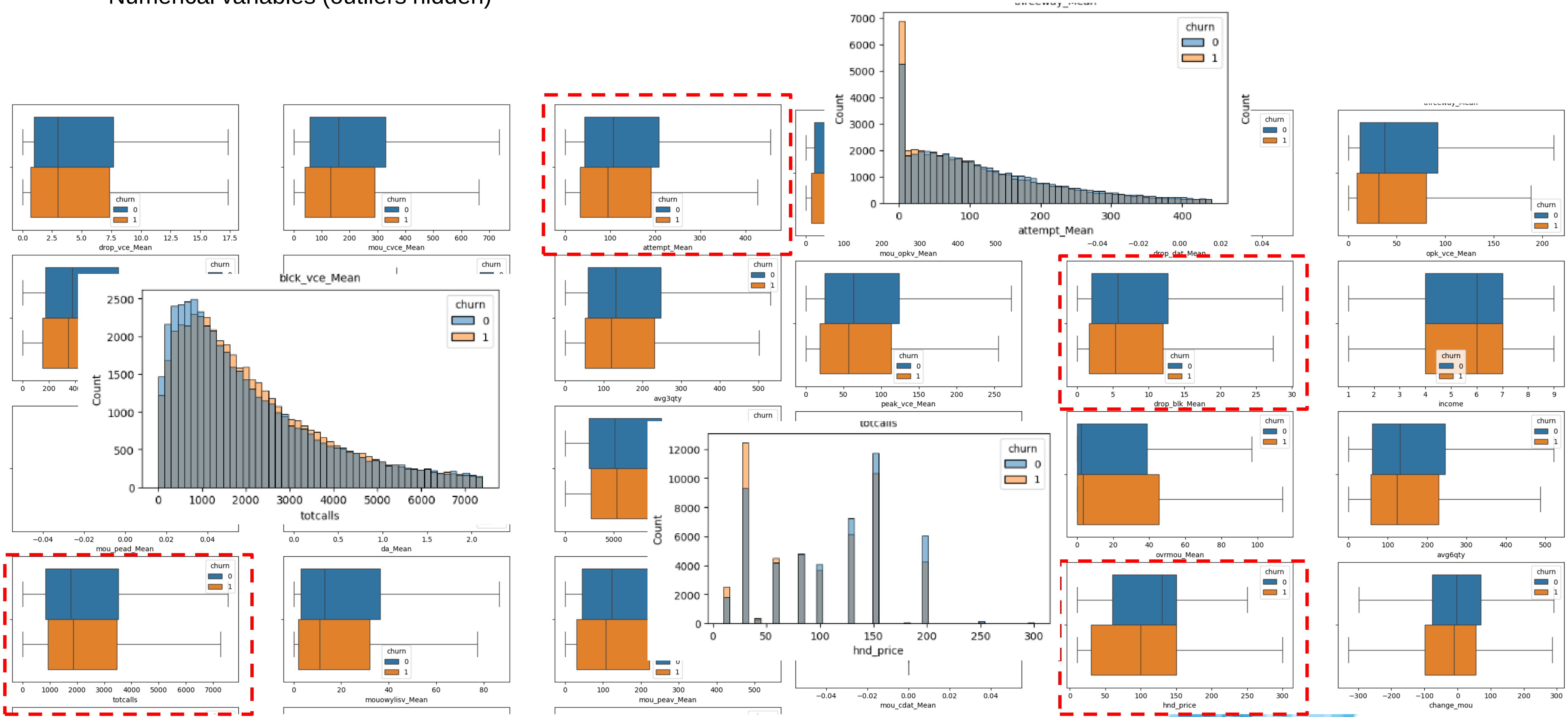
Churn-features relationship

Numerical variables (outliers hidden)



Churn-features relationship

Numerical variables (outliers hidden)



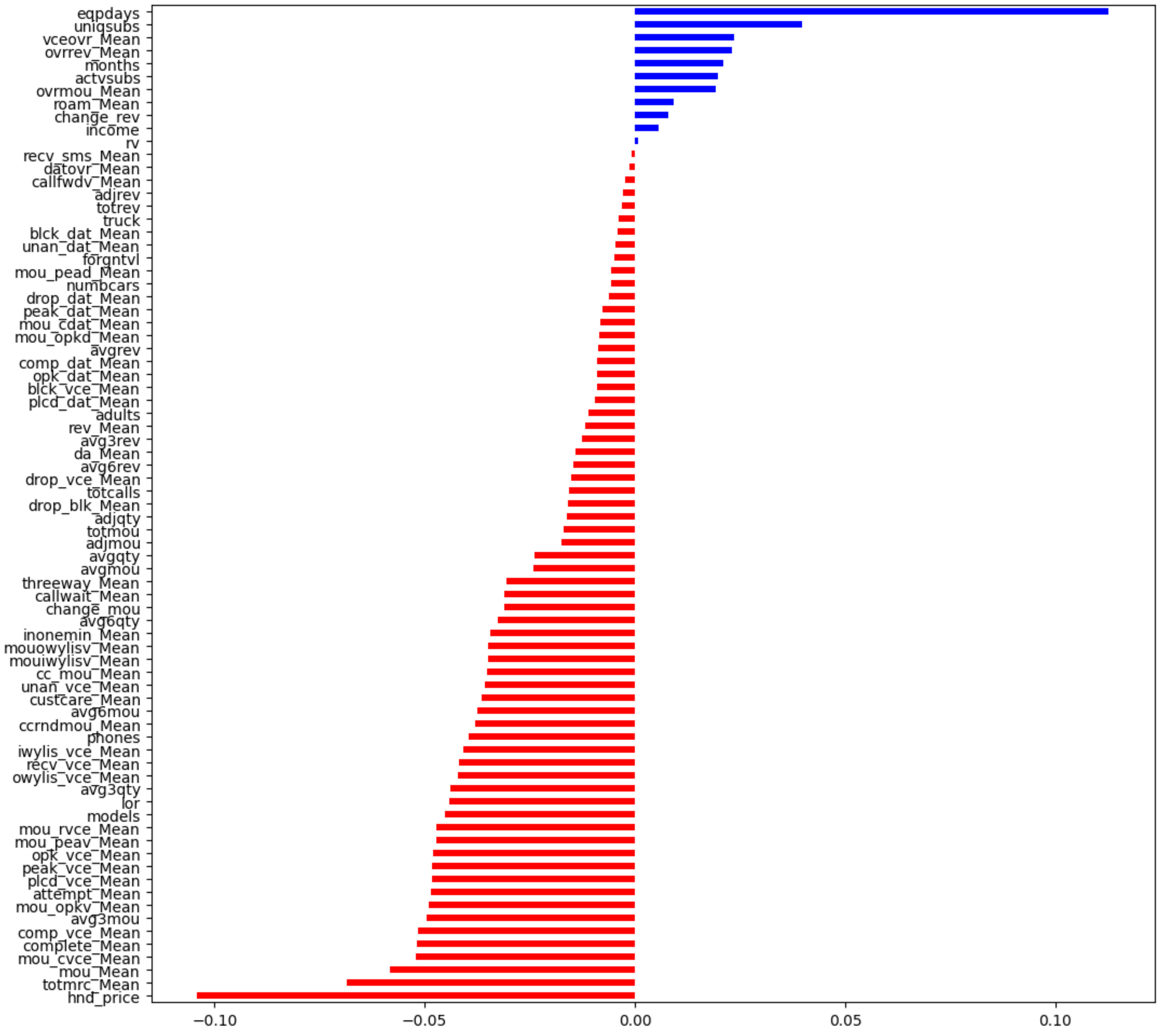
Churn-features relationship

Point Biserial Correlation factor.

$$r_{pb} = \frac{\overline{Y_1} - \overline{Y_0}}{s_y}\sqrt{\frac{N_0 N_1}{N(N-1)}}$$

Used to check the correlation between the numeric features and the target variable.

Coefficients > 0 means correlation with binary state 1.
Coefficients < 0 means correlation with binary state 0.

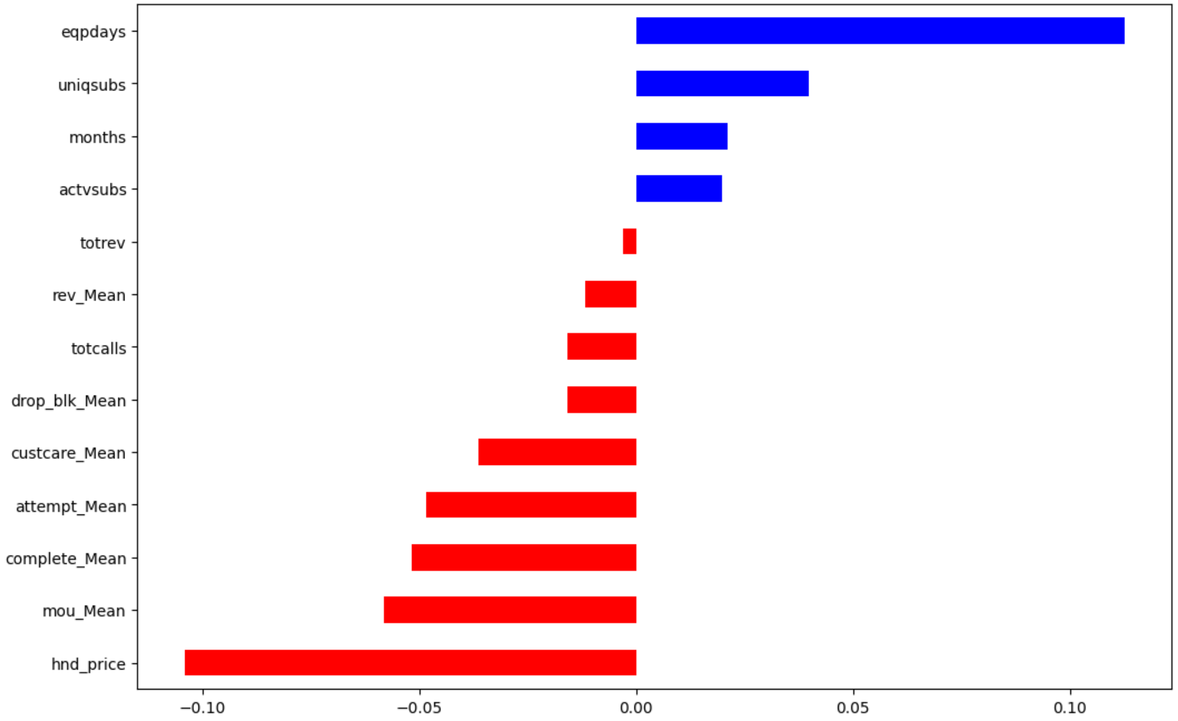


Features selected

From correlation analysis some features were removed.

```python
feat_num_to_keep = ['rev_Mean', 'eqpdays', 'drop_blk_Mean', 'uniqsubs', 'actvsubs',
                    'custcare_Mean',
                    'complete_Mean',
                    'attempt_Mean',
                    'hnd_price',
                    'months',
                    'mou_Mean', 'totcalls', 'totrev']

feat_cat_to_keep = ['asl_flag',
                    'dualband',
                    'hnd_webcap',
                    'refurb_new',
                    'marital',
                    'models',
                    'adults',
                    'kid0_2',
                    'kid3_5', 'kid6_10', 'kid11_15', 'kid16_17']
```

Point Biserial Correlation factor



Machine Learning

Classification problem.

Target variable identified (churn).

Features selected from EDA.

Next steps:

- Split dataset into train-test.
- Clean dataset (outliers and missing values).
- Scaling features.
- Encoding categorical variables.
- Select model and train it.
- Model evaluation.

Machine Learning

Split dataset into train-test.

```python
target_column = "churn"
df_target = data[target_column]
df_features = data[feat_num_to_keep + feat_cat_to_keep]

X_train, X_test, y_train, y_test = train_test_split(
    df_features, df_target, stratify=df_target, test_size=0.2, random_state=42)
```

```python
cols_to_drop, null_ratio = detect_missing_values(X_train, threshold=0.25)
imputer = drop_and_impute_missing(X_train, cols_to_drop=[], set_type="train")
drop_and_impute_missing(X_test, cols_to_drop=[], set_type="test", imputer=imputer)
```

Clean dataset

```python
replace_outliers(X_train)
replace_outliers(X_test)
```

```python
def replace_outliers(df, whiskers=1.5):
    """Replace outliers with median value

    Args:
        df (_type_): Input data frame with numeric values only
        whiskers (float, optional): Threshold to compute the upper and lower bounds. Defaults to 1.5.

    Returns:
        Data Frame: With outliers replaced.
    """
    # Make sure the selected values are the numeric ones
    df_numeric = df.select_dtypes('number')
    # Compute the quantiles for each column
    quant = df_numeric.quantile(q=[0.75, 0.25])
    # Compute the IQR and the upper and lower bounds used to consider outliers
    iqr = quant.iloc[0] - quant.iloc[1]

    up_bound = quant.iloc[0] + (whiskers*iqr)
    low_bound = quant.iloc[1] - (whiskers*iqr)

    # Replace values above upper bound with median
    df_numeric = df_numeric.apply(lambda x: x.mask(
        x > up_bound.loc[x.name], np.nan), axis=0)

    # Replace values below lower bound with median
    df_numeric = df_numeric.apply(lambda x: x.mask(
        x < low_bound.loc[x.name], np.nan), axis=0)

    # Replace the numeric columns in the data set
    df[df_numeric.columns] = df_numeric

    return df
```

```python
def detect_missing_values(df, threshold=0.1):
    """Detect missing values and return columns to drop (if pass the threshold) and the ratio of the missing values.

    Args:
        df (DataFrame): Data Frame
        threshold (float, optional): Threshold to drop columns. Defaults to 0.1.

    Returns:
        tuple: Tuple containing columns to drop and their r
    """
    nrows = len(df)
    # Retrieves the columns with null values
    col_w_nulls = df.columns[df.isna().any()]
    # Ratio of null values
    null_ratio = df[col_w_nulls].isna().sum(
    ).sort_values(ascending=False) / nrows
    # Columns with null values greater than the threshold
    cols_to_drop = list(null_ratio.index[null_ratio > thres

    return cols_to_drop, null_ratio
```

```python
def drop_and_impute_missing(df, cols_to_drop=[], set_type="train", imputer=None):
    """Drop columns based on a list of column names, and impute values based on the imputer passed (if any)
    and depending on if the df is the test or training set.

    Args:
        df (DataFrame): Data Frame
        cols_to_drop (list, optional): List of column names to drop. Defaults to [].
        imputer (object, optional): Imputer object to impute missing values. Defaults to None.
        set_type (str, optional): check if the df is the test or train set. Options: test, train.

    Returns:
        DataFrame: DataFrame after dropping columns and imputing missing values
    """
    if cols_to_drop:
        df.drop(columns=cols_to_drop, axis=1, inplace=True)

    if set_type.lower() == "train":
        if imputer is None:
            imputer = SimpleImputer(strategy="most_frequent")
        df[df.columns] = imputer.fit_transform(df)
        df[df.columns] = df[df.columns].infer_objects()
        # print(df.info())
        return imputer
    elif set_type.lower() == "test":
        if imputer is None:
            raise ValueError(
                "An imputer object is required for imputing missing values in the test set.")
        df[df.columns] = imputer.transform(df)
        df[df.columns] = df[df.columns].infer_objects()
        # print(df.info())
    else:
        raise ValueError(
            "Invalid set_type. Allowed options are 'train' or 'test'.")

    return
```

Machine Learning

Split dataset into train-test.

```
X_train_num_scaled, scaler_obj = my_scaler(X_train.select_dtypes('number'), set_type="train", scaler_type="standard")
X_test_num_scaled, _ = my_scaler(X_test.select_dtypes('number'), set_type="test", scaler_chosen=scaler_obj)

X_train[X_train_num_scaled.columns] = X_train_num_scaled
X_test[X_test_num_scaled.columns] = X_test_num_scaled
```

Encode categorical features

```
from sklearn.preprocessing import OneHotEncoder

cat_var = X_train.select_dtypes('object').columns #feat_cat_to_keep

cat_encoder = OneHotEncoder(handle_unknown='ignore')
feat_encoded = cat_encoder.fit_transform(X_train[cat_var]).toarray()

feature_labels = cat_encoder.get_feature_names_out()
X_train_cat = pd.DataFrame(data=feat_encoded, columns=feature_labels)
X_train_cat.index = X_train.index
X_train = pd.concat([X_train, X_train_cat], axis=1)
X_train.drop(columns=cat_var, axis=1, inplace=True)

feature_test_encoded = cat_encoder.transform(X_test[cat_var]).toarray()
X_test_cat = pd.DataFrame(data=feature_test_encoded, columns=cat_encoder.get_feature_names_out())
X_test_cat.index = X_test.index
X_test = pd.concat([X_test, X_test_cat], axis=1)
X_test.drop(columns=cat_var, axis=1, inplace=True)
```

```
def my_scaler(df, set_type="train", scaler_chosen=None, scaler_type="robust"):
    """Scale a dataset with a passed/selected scaler type.

    Args:
        df (_type_): Dataset to scale.
        set_type (str, optional): Dataset type 'train' or 'test'. Defaults to "train".
        scaler_chosen (_type_, optional): Scaler object to use. Needed if set_type = test. Defaults to None.
        scaler_type (str, optional): Scaler type to be used. Needed for trainin datasets. Defaults to "robust".

    Returns:
        _type_: Dataframe scaled.
    """

    from sklearn.preprocessing import RobustScaler, StandardScaler, PowerTransformer

    scaler_dict = {"robust": RobustScaler(),
                   "standard": StandardScaler(),
                   "power": PowerTransformer()}

    if set_type.lower() == "train":
        if scaler_type is None:
            return ValueError("A valid scaler type has to be chosen. Valid scalers are: 'robust', 'standard' and 'power'")
        scaler_chosen = scaler_dict[scaler_type]
        array_scaled = scaler_chosen.fit_transform(df)
        df_scaled = pd.DataFrame(
            data=array_scaled, columns=scaler_chosen.get_feature_names_out())

    elif set_type.lower() == "test":
        if scaler_chosen is None:
            return ValueError("An scaler object has to be passed.")
        array_scaled = scaler_chosen.fit_transform(df)
        df_scaled = pd.DataFrame(
            data=array_scaled, columns=scaler_chosen.get_feature_names_out())

    df_scaled.index = df.index

    return df_scaled, scaler_chosen
```

Machine Learning

Model selection, evaluation and feature importances.

```
rfc = RandomForestClassifier()

rfc.fit(X_train, y_train)

y_pred = rfc.predict(X_test)
print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	0.60	0.60	0.60	10088
1	0.59	0.60	0.59	9912
accuracy			0.60	20000
macro avg	0.60	0.60	0.60	20000
weighted avg	0.60	0.60	0.60	20000

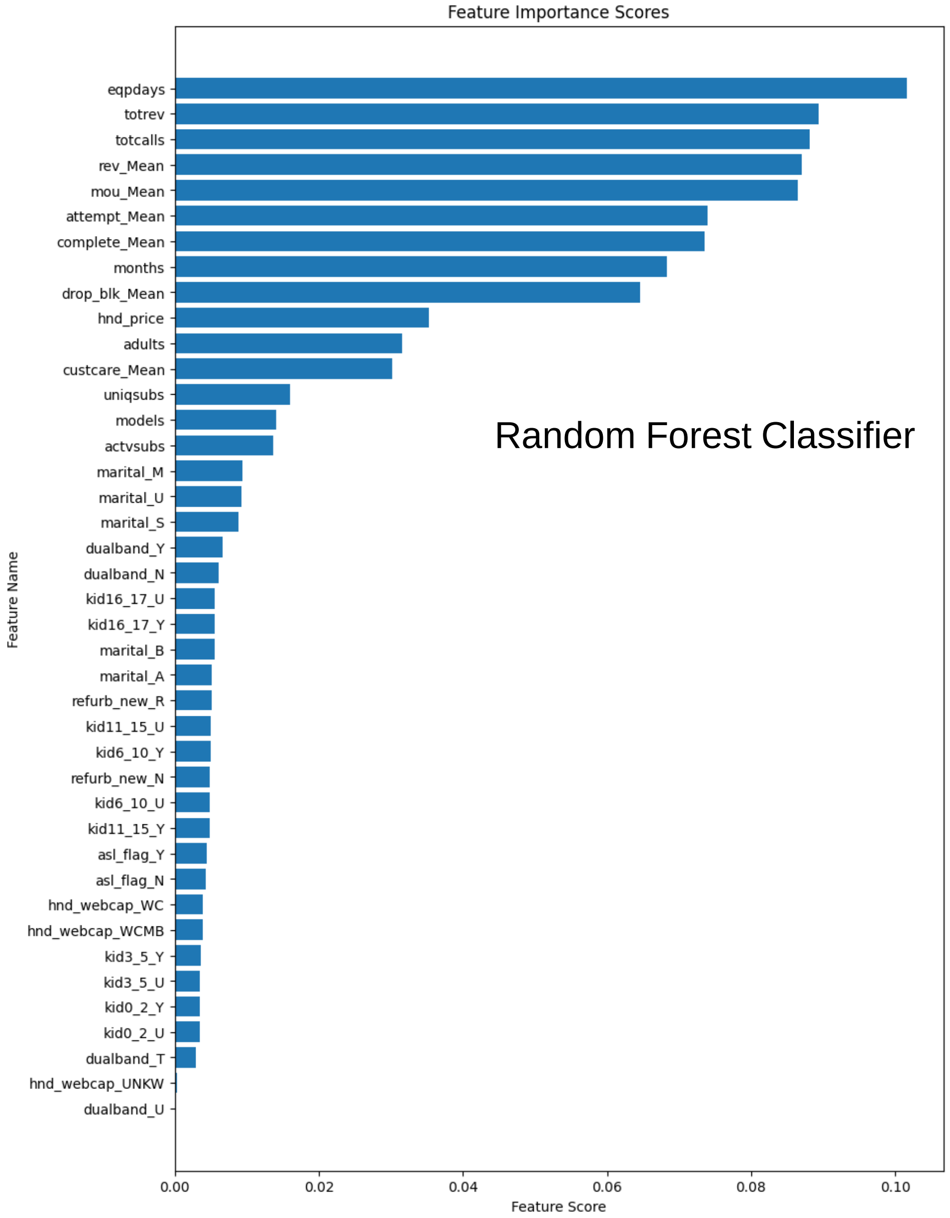
```
from sklearn.linear_model import LogisticRegressionCV

clf = LogisticRegressionCV()

clf.fit(X_train_fs, y_train)

y_pred_ts_log = clf.predict(X_test_fs)
print(classification_report(y_test, y_pred_ts_log))
```

	precision	recall	f1-score	support
0	0.57	0.59	0.58	10088
1	0.57	0.54	0.56	9912
accuracy			0.57	20000
macro avg	0.57	0.57	0.57	20000
weighted avg	0.57	0.57	0.57	20000

Random Forest Classifier chosen due to better performance.

Approach for (tentatively) improve the performance:

- Perform feature engineering (transform variables into another variables) to get features with higher importance.
- Drop features with missing values over a threshold.
- Test other models.
- Hyperparameter tunning (select different methods for doing stimations, select different number of trees in a random forest...)



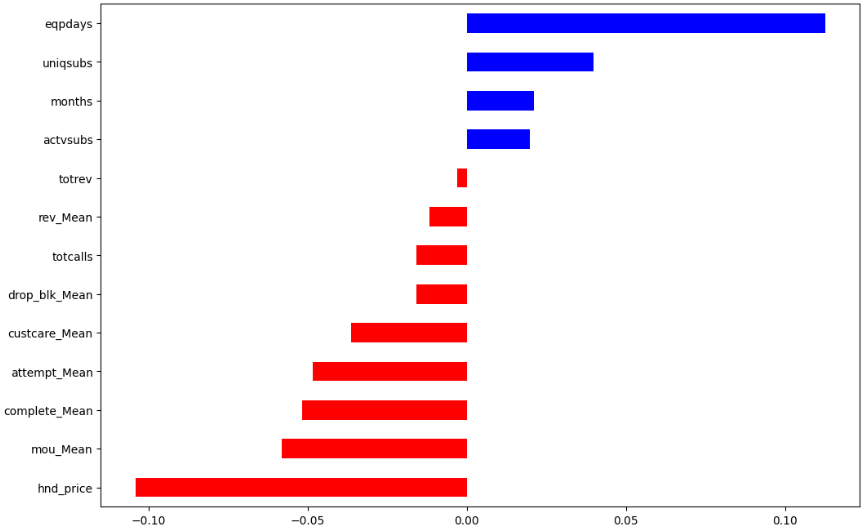
Conclusion

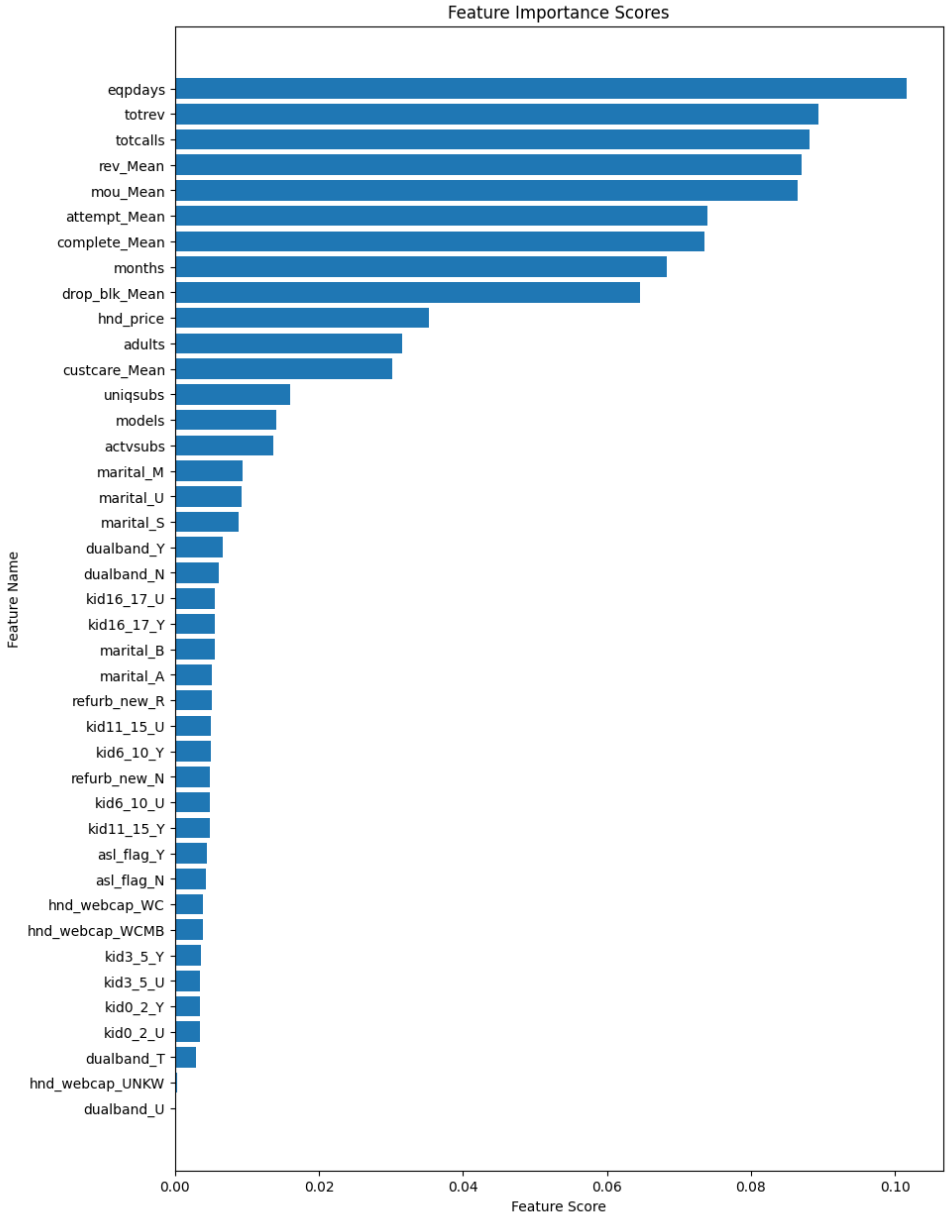
Based on feature importances and correlation coeficients of features with the target variable:

- Eqpdays: Number of days of current equipment.
- Hnd_price: Price of the device.
- uniqsubs: Number of unique suscribers on the house.
- totcalls: Total number of calls over the life of the customer.
- Custcare_Mean: Mean number of customer care calls.

Suggestions:

- Renew the device of the customers (new promotions).
- Family promotions (reduce the number of unique suscribers in the household, and reduce the cost of services).
- Promotions with reduced rates/fee for usual customers.





Implementation in Airflow

Setting up environment

- Install docker.
- Create a folder for the project.
- Dowload the "*docker-compose.yaml*" on the project folder.
- Run the *docker compose up airflow-init* to initialize the database for the airflow project.

```
DataTech
  config
  dags
    __pycache__
    includes
      __pycache__
      __init__.py
      helper_functions.py
      process_data.py
    dag_pipeline.py
  data
    processed
    raw
      dataset.csv
  includes
  logs
  plugins
  .env
  docker-compose.yaml
  Dockerfile
  requirements.txt
```

```
joblib==1.3.2
jsonschema==4.6.0
matplotlib==3.7.4
numpy==1.21.2
openpyxl==3.1.2
packaging==23.1
pandas==2.0.1
pickleshare==0.7.5
scikit-learn==1.3.2
scipy==1.10.1
seaborn==0.13.1
statsmodels==0.14.1
```

- List needed dependencies in a *requirement.txt* file.
- Write the instructions in the *Dockerfile*.

```
FROM apache/airflow:2.8.1
COPY requirements.txt /requirements.txt
RUN pip install --user --upgrade pip
RUN pip install --no-cache-dir --user -r /requirements.txt
```

- Build the new image with the extended packages.
- Change the image name in the "*docker-compose.yaml*".
- Run again *docker compose up* to build the image with the extended image of airflow.

```
docker build . --tag extending_airflow:2.8.1
```

```
# Feel free to modify this file to suit your needs.
---
x-airflow-common:
  &airflow-common
  # In order to add custom dependencies or upgrade provider packages you can use yo
  # Comment the image line, place your Dockerfile in the directory where you placed
  # and uncomment the "build" line below, Then run `docker-compose build` to build
  image: ${AIRFLOW_IMAGE_NAME:-extending_airflow:2.8.1} # apache/airflow:2.8.1}
  # build: .
```

Implementation in Airflow

Docker and Airflow up and running

Structure of the data pipeline