Projeto 2 - Previsão de renda com uso de Modelagem Preditiva

Etapa 1 CRISP - DM: Entendimento do negócio

O conceito de crédito pré-data o surgimento do dinheiro. Mesmo em uma essência mais primitiva, não haveria negociações sem relações de reputação e sentimento de confiança, na sociedade pré-monetária. A ação de negociar valores gerou uma série de fatores que influênciaram diretamente a história e moldaram a evolução para a sociedade como ela é hoje.

O uso do crédito permitiu o desenvolvimento de cidades, criação de bancos e até a estruturação de leis. Ele tem grande importância no processo de Acumulação de Capital, isto é, age como um transformador em diferentes níveis e categorias. É uma das princípais chaves para setores produtivos, empresariais e de negócios atuais. Sem contar a possibilidade de alteração em paradigmas financeiros das famílias mais vulneráveis.

Com o expressivo crescimento dessa modalidade, sua influência na sociedade se tornou complexa. Segundo dados mensais da Pesquisa Nacional de Endividamento e Inadimplência do Consumidor, atualmente, **cerca de 29% das famílias brasileiras estão inadimplentes** e **78% dos lares têm dívidas prestes a vencer** (CNC - CONFEDERAÇÃO NACIONAL DO COMÉRCIO DE BENS, SERVIÇOS E TURISMO, 2022). O número representa um recorde desde 2010, sendo o 4º aumento anual consecutivo.

Portanto, o objetivo do atual projeto, é servir o mutuário (o cliente) para que avalie suas próprias decisões, através da identificação do risco de Inadimplência (tipicamente definido pela ocorrência de um atraso maior ou igual a 90 em um horizonte de 12 meses), pelo meio de variáveis que podem ser observadas na data presente da Avaliação do Crédito, ou seja, quando o cliente solicita o Cartão ao prestador de serviços.

Para isso, serão desenvolvidos e avaliados Modelos Preditivos, para simulação de um *Score* de Crédito do mutuário, de modo a auxiliá-lo a tomar suas próprias decisões referentes aos seus *próprios* dados, com intuito de evitar o agravamento de dívidas e buscar a transformação socioeconômica almejada pelo indivíduo.

Com intuito de desenvolver um projeto robusto, preciso e eficaz, será realizado um "*Ensemble of Models*", isto é, uma "Conjunção de Modelos". Serão utilizados Modelos diversificados para aquisição do *Score* de Crédito:

- Random Forest
- Regressão Linear
- Árvore de Decisão
- SVM (Support Vector Machine)

• K-NN (K-Nearest Neighbors)

No entanto, é importante delinear e planejar adequadamente algumas diretrizes, para garantir que esse processo seja bem-sucedido de acordo com a excelência & complexidade almejada:

- **Diversidade dos Modelos**:

  > Ao cogitar o Ensemble, é importante escolher modelos que sejam diversificados em termos de abordagem e comportamento. Modelos diferentes podem capturar diferentes aspectos dos dados e, assim, melhorar a robustez do resultado. Tradicionalmente na estatística, Regressão Linear e Árvores de Decisão podem ser mais adequados para indicadores financeiros; enquanto outros mais flexíveis, como Random Forest, K-NN e, em certa medida, o SVM, podem acomodar melhor informações comportamentais.

- **Ajuste Fino de Hiperparâmetros e Validação Cruzada**:

  > O ajuste fino dos hiperparâmetros de cada modelo, será feito manualmente com exaustão de parâmetros e o emprego de técnica como "Cross-Validation 5-fold"; para avaliar a capacidade de generalização do ensemble e evitando possiveis overfitting/underfitting dos modelos. O uso de Grid ou Random Search é recomendado, mas os hiperparâmetros serão selecionados adaptativamente até conversão e constância dos resultados.

- **Avaliação individual dos Modelos**:

  > É fundamental avaliar cuidadosamente o desempenho de cada modelo individualmente no conjunto de dados. Isso permite que identificar quais modelos têm o melhor desempenho e quais podem ser menos eficazes, frente aos dados disponíveis na data da Avaliação.

- **Combinação das Previsões**:

  > Os resultados de cada um dos modelos serão unificados pela média ponderada, com pesos personalizados de acordo com o desempenho de cada modelo. Algumas métricas comuns de desempenho de modelos preditivos, incluem: Precisão, Recall, F1-score, área sob a curva ROC (AUC-ROC), R^2, MSE, RMSE, MAE, MAPE, entre outras.
  >
  > A métrica R^2, também conhecida como R-dois ou coeficiente de determinação, representa o percentual da variância dos dados que é explicado pelo modelo. Quanto maior é o valor de R^2, mais explicativo é o modelo em relação aos dados previstos. É recomendado sempre utilizar outras métricas em concomitância, além de apenas R^2, para se ter uma visão global sobre a performance do modelo.
  >
  > O erro médio absoluto (MAE — do inglês, Mean Absoluto Error), mensura a média da diferença entre o valor real com o predito. Por haver valores positivos e negativos de erros em relação à média, é adicionado um módulo entre a diferença dos valores. Além disso, esta métrica não é afetada por valores discrepantes — os denominados *outliers*. O valor de saída da equação tem a mesma unidade de medida dos dados, logo fica mais fácil a sua interpretação.

A raiz do erro quadrático médio (RMSE — do inglês, Root Mean Squared Error) contém a ideia de penalização entre diferenças grandes do valor previsto e o real. Assim, a unidade de medida fica a mesma que o dado original, resultando em uma melhor interpretabilidade do resultado da métrica. Apesar do valor ter a mesma unidade, ele demonstra como os *outliers* podem estar impactando nas previsões do modelo. A sua interpretabilidade pode seguir a lógica de que na previsão de renda, o resultado sendo igual a 30,0, significa que o modelo pode estar errando em 30,0 reais para mais ou para menos.

Por essas razões, estas métricas são boas opções quando é preciso ter uma avaliação mais criteriosa sobre as previsões dos modelos.

Devido ao cunho numérico (não categórico) da variável-alvo do estudo - renda - os parâmetros para avaliação dos conjuntos, serão os três previamente descritos: o **R^2**, a **Raiz do Erro Médio Quadrático (RMSE)** e o **Erro Médio Absoluto (MAE)**; bem como serão os fornecedores de ciência para os pesos dos modelos no ato da combinação posterior, gerando o *Score*.

Etapa 2 Crisp-DM: Entendimento dos dados

Existem 15 variáveis no Conjunto de Dados, conforme a **TABELA 1**, descreve a seguir:

TABELA 1 - Dicionário de dados.

Nome da Variável	Descrição	Tipo
Unnamed: 0	Atua como índice da linha	inteiro
data_ref	Mês/Ano da última compra do cliente (ex: AAAA-MM-01)	texto
id_cliente	Número de identificação do cliente	inteiro
sexo	M = "Masculino"; F = "Feminino"	texto
posse_de_veiculo	True = "possui"; False = "não possui"	binária
posse_de_imovel	True = "possui"; False = "não possui"	binária
qtd_filhos	Quantidade de filhos	inteiro
tipo_renda	Tipo de renda (ex: assaliariado, autônomo etc)	texto
educacao	Nível de educação (ex: secundário, superior etc)	texto
estado_civil	Estado civil (ex: solteiro, casado etc)	texto
tipo_residencia	Tipo de residência (ex: casa/apartamento, com os pais etc)	texto
idade	Idade do cliente em anos	inteiro
tempo de emprego	Tempo no cargo atual do cliente em anos	flutuador
qt_pessoas_residencia	Quantidade de pessoas na residência	flutuador
renda	Valor mensal de renda informado pelo cliente	flutuador

> O Conjunto de Dados possui informações dispostas em uma linha para cada compra do cliente e uma coluna para cada variável; contendo as características dos mutuários de uma empresa fictícia de crédito, em um arquivo no formato .csv. Originalmente publicada no Kaggle - uma plataforma que promove desafios de Ciência de Dados, oferecendo prêmiação para os melhores colocados.

Carregando os pacotes e o Conjunto de Dados

In [1]:
```
# Trecho importações básicas
import shap
import time
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

# Trecho de carregamentos específicos
from tqdm import tqdm
from sklearn import metrics
from sklearn.svm import SVR
from scipy.stats import ttest_ind
from ydata_profiling import ProfileReport
from matplotlib.patches import Rectangle
from matplotlib.ticker import FixedLocator
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.feature_selection import SelectFromModel
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.model_selection import train_test_split, KFold, cross_val_score
from sklearn.metrics import mean_squared_error, mean_absolute_error, make_scorer, r2_sco

# Abrindo a leitura do Conjunto de Dados
df = pd.read_csv("./input/previsao_de_renda.csv")

# Verificando se há informações duplicadas sobre os clientes
print("-------> Quantidade de linhas duplicadas no Conjunto de Dados: ", len(df[df.dupli

# Confirmando as informações gerais da TABELA 1 sobre as colunas presentes
print(df.info())

# Resumindo a estrutura do Conjunto fornecida pelo info()
print(f"\n-------> O Conjunto de Dados Inicial possui {df.shape[0]} linhas e {df.shape[1

df.head() # Exibindo a cara do Conjunto de Dados
```

```
-------> Quantidade de linhas duplicadas no Conjunto de Dados:  0

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15000 entries, 0 to 14999
Data columns (total 15 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Unnamed: 0             15000 non-null  int64
 1   data_ref               15000 non-null  object
 2   id_cliente             15000 non-null  int64
 3   sexo                   15000 non-null  object
 4   posse_de_veiculo       15000 non-null  bool
 5   posse_de_imovel        15000 non-null  bool
 6   qtd_filhos             15000 non-null  int64
 7   tipo_renda             15000 non-null  object
```

```
 8   educacao                15000 non-null  object
 9   estado_civil            15000 non-null  object
 10  tipo_residencia         15000 non-null  object
 11  idade                   15000 non-null  int64
 12  tempo_emprego           12427 non-null  float64
 13  qt_pessoas_residencia   15000 non-null  float64
 14  renda                   15000 non-null  float64
dtypes: bool(2), float64(3), int64(4), object(6)
memory usage: 1.5+ MB
None

-------> O Conjunto de Dados Inicial possui 15000 linhas e 15 colunas.
```

Out[1]:

	Unnamed: 0	data_ref	id_cliente	sexo	posse_de_veiculo	posse_de_imovel	qtd_filhos	tipo_renda	educacao	es
0	0	2015-01-01	15056	F	False	True	0	Empresário	Secundário	
1	1	2015-01-01	9968	M	True	True	0	Assalariado	Superior completo	
2	2	2015-01-01	4312	F	True	True	0	Empresário	Superior completo	
3	3	2015-01-01	10639	F	False	True	1	Servidor público	Superior completo	
4	4	2015-01-01	7064	M	True	False	0	Assalariado	Secundário	

Entendimento dos dados - Univariada

Com o uso da potente ferramenta de relatório do Pandas, o entendimento do Conjunto de Dados facilitará completamente todos os próximos passos, trazendo agilidade e ciência para o projeto.

In [2]:
```
# Criando, salvando e exibindo o Relatório do Perfil do Conjunto de Dados brutos
perfil_ppr = ProfileReport(df, explorative = True, minimal = True)
perfil_ppr.to_file("./output/relatorio_ppr.html")
perfil_ppr
```

```
Summarize dataset:   0%|          | 0/5 [00:00<?, ?it/s]
Generate report structure:   0%|          | 0/1 [00:00<?, ?it/s]
Render HTML:   0%|          | 0/1 [00:00<?, ?it/s]
Export report to file:   0%|          | 0/1 [00:00<?, ?it/s]
```

Overview

Overview | Alerts 3 | Reproduction

Dataset statistics

Number of variables	15
Number of observations	15000
Missing cells	2573
Missing cells (%)	1.1%
Total size in memory	6.9 MiB
Average record size in memory	483.7 B

Variable types

Numeric	7
Text	6
Boolean	2

Variables

Out[2]:

DESTAQUES & COMENTÁRIOS:

A seguir, foram recortados destaques da visão geral contida no arquivo, bem como os comentário referentes aos alertas contidos no Relatório de Perfil do Conjunto de Dados.
Destaques - Visão geral:

- "**id_cliente**" apresenta 9845 clientes distintos;

- "**sexo**" apresenta mais mulheres clientes;
- "**posse_de_veiculo**" & "**posse_de_imovel**", tem proporções de 60/40 & 70/30 para não/possui, respectivamente;
- "**qtd_filhos**" tem mínimo de 0, máximo de 14 filhos e distribuição deslocada para esquerda;
- as variáveis "**tipo_renda**", "**educacao**", "**estado_civil**" & "**tipo_residencia**" apresentam "assalariado", "secundário", "casado" & "casa" como atributos dominantes, respectivamente;
- "**idade**" tem mínimo de 22, máximo de 68 e maior frequência de 40 anos. Distribuição parece ser do tipo Mista (Uniforme / Normal);
- "**tempo_emprego**" tem mínimo de 42 dias, máximo de 43 anos, média de cerca de 8 anos e distribuição deslocada para esquerda;
- "**qt_pessoas_residencia**" tem mínimo de 1 morador, máximo de 15 moradores, valor médio de 2 pessoas por casa e distribuição deslocada para esquerda;
- a variável de interesse "**renda**" possui valor mínimo de 118,71 e máximo de 245.141,67 reais, com média em 5697,29 reais e distribuição deslocada para esquerda;

Comentários - Alertas do Relatório:

> A variável explicativa "**tempo_emprego**" tem 17,2% de lacunas. Alerta devidamente aceito: Como o Relatório mostra, a distribuição desta variável é Assimétrica à Direita. Estatisticamente, será usada a mediana nestes valores, pois a média e a moda estão distorcidas pela assimetria.
>
> "**Unnamed: 0**" possui 100% de valores distintos; funcionando sequencialmente como um índice da base. Alerta devidamente aceito: A coluna será avaliada quanto a possibilidade de descarte dela no estudo.
>
> A coluna "**qtd_filhos**" apresenta 69,2% de nulos. Alerta devidamente rejeitado: (Falso-alerta) Uma vez que "qtd_filhos = 0" indica o simples fato do cliente não possuir filhos.

Entendimento dos dados - Bivariadas

A análise bivariada tem como objetivo principal identificar se existe alguma relação ou associação entre duas variáveis. Isso pode ser feito de várias maneiras, dependendo da natureza das variáveis envolvidas. Essas variáveis podem ser quantitativas ou qualitativas, e a análise bivariada é uma parte fundamental da exploração de dados e da estatística descritiva.

Iniciando pela criação de um DataFrame contendo as correlações significativas (positivamente ou negativamente), visando ampliar o entendimento do Conjunto de Dados inicial. Em concomitância, é avaliado o comportamento das variáveis que mais podem ser significativas para a **Previsão de Renda**. O *Heatmap* completo será gerado com os dados de todas correlações, para poder visual e posterior exibição.

In [3]:
```
# Passo auxiliar temporario para gerar a correlação entre as variáveis brutas usando one
aux = pd.get_dummies(df) # A criação verdadeira dos dummies será feita no trecho 3.3 des
aux_temp = aux.corr()

# Partição da escala [-1 ; 1] em três e filtrando apenas as partes exteriores
filtro = aux_temp[(aux_temp > 0.333333) | (aux_temp < -0.333333)].stack()

# Criando o novo DataFrame temporario com os valores filtrados
correlatos = pd.DataFrame({"Valor": filtro.values,
```

```
                            "X": [index[0] for index in filtro.index],
                            "Y": [index[1] for index in filtro.index]})

# Selecionando categorias "poder das Correlações": através da separação do intervalo |0,
correlatos["Correlação"] = np.select(
    [abs(correlatos["Valor"]) >= 0.776666,
     (abs(correlatos["Valor"]) >= 0.553333) & (abs(correlatos["Valor"]) < 0.776666),
     abs(correlatos["Valor"]) < 0.553333],
    ["Forte", "Média", "Leve"]
)

# Organizando a exibição do novo df por encadeamento de métodos
correlatos = (correlatos[~np.isclose(correlatos["Valor"], 1) & ~np.isclose(correlatos["V
              .sort_values("Correlação")
              .drop_duplicates("Valor") # Correlação de (X,Y) é a mesma de (Y,X)
              .reset_index(drop = True)
              .round({"Valor": 2})
              .set_index(["Valor"]))
correlatos
```

Out[3]:

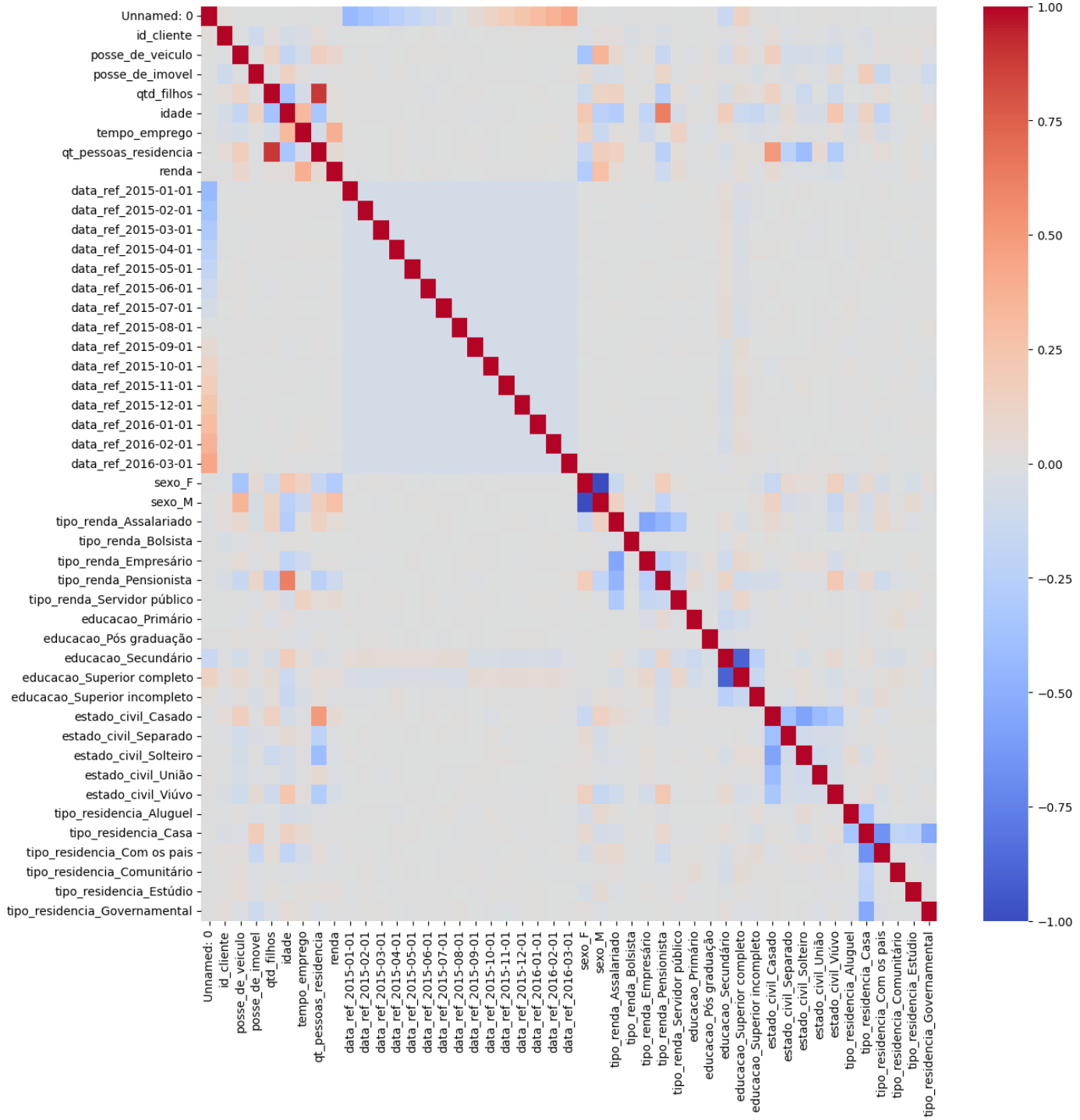
Valor	X	Y	Correlação
-0.90	educacao_Secundário	educacao_Superior completo	Forte
0.89	qt_pessoas_residencia	qtd_filhos	Forte
-0.43	Unnamed: 0	data_ref_2015-01-01	Leve
-0.46	tipo_renda_Assalariado	tipo_renda_Pensionista	Leve
0.50	estado_civil_Casado	qt_pessoas_residencia	Leve
-0.38	estado_civil_Casado	estado_civil_Separado	Leve
-0.43	estado_civil_Casado	estado_civil_União	Leve
-0.34	estado_civil_Casado	estado_civil_Viúvo	Leve
-0.41	estado_civil_Solteiro	qt_pessoas_residencia	Leve
-0.35	tipo_residencia_Aluguel	tipo_residencia_Casa	Leve
-0.54	tipo_residencia_Casa	tipo_residencia_Governamental	Leve
-0.36	sexo_F	posse_de_veiculo	Leve
0.43	data_ref_2016-03-01	Unnamed: 0	Leve
0.36	sexo_M	posse_de_veiculo	Leve
-0.37	data_ref_2015-02-01	Unnamed: 0	Leve
0.37	Unnamed: 0	data_ref_2016-02-01	Leve
-0.36	qtd_filhos	idade	Leve
-0.34	idade	qt_pessoas_residencia	Leve
0.39	renda	tempo_emprego	Leve
-0.57	estado_civil_Casado	estado_civil_Solteiro	Média
0.62	tipo_renda_Pensionista	idade	Média
-0.56	tipo_renda_Empresário	tipo_renda_Assalariado	Média
-0.66	tipo_residencia_Casa	tipo_residencia_Com os pais	Média

Deste modo, é notável que a distribuição das correlações significativas, é prodominantemente "Leve". Inclusive esta, é a classe em que a *única* correlação dos dados brutos exibe perante a **variável-alvo** do estudo:

- **Renda por Tempo de Emprego (+ 0,39)**: indica que ao percorrermos os dados de uma variável (valor_1), a outra aumenta levemente (valor_2); apresentando uma leve ligação causal entre elas (valor_1-2). (*insight* Estatístico)

- (*tradução* Empírica) Conforme o mutuário adquiri mais experiência trabalhando (valor_1), recebe mais reconhecimento e, por transitividade de valores, sua renda (valor_2) também aumenta.

In [4]:
```
# Mapa de Calor entre os dados brutos de todas as variáveis
plt.figure(figsize = (14, 14))
sns.heatmap(aux_temp, cmap = "coolwarm")
plt.savefig("./output/heatmap.png", bbox_inches = "tight", pad_inches = 0.1)
plt.show()
```

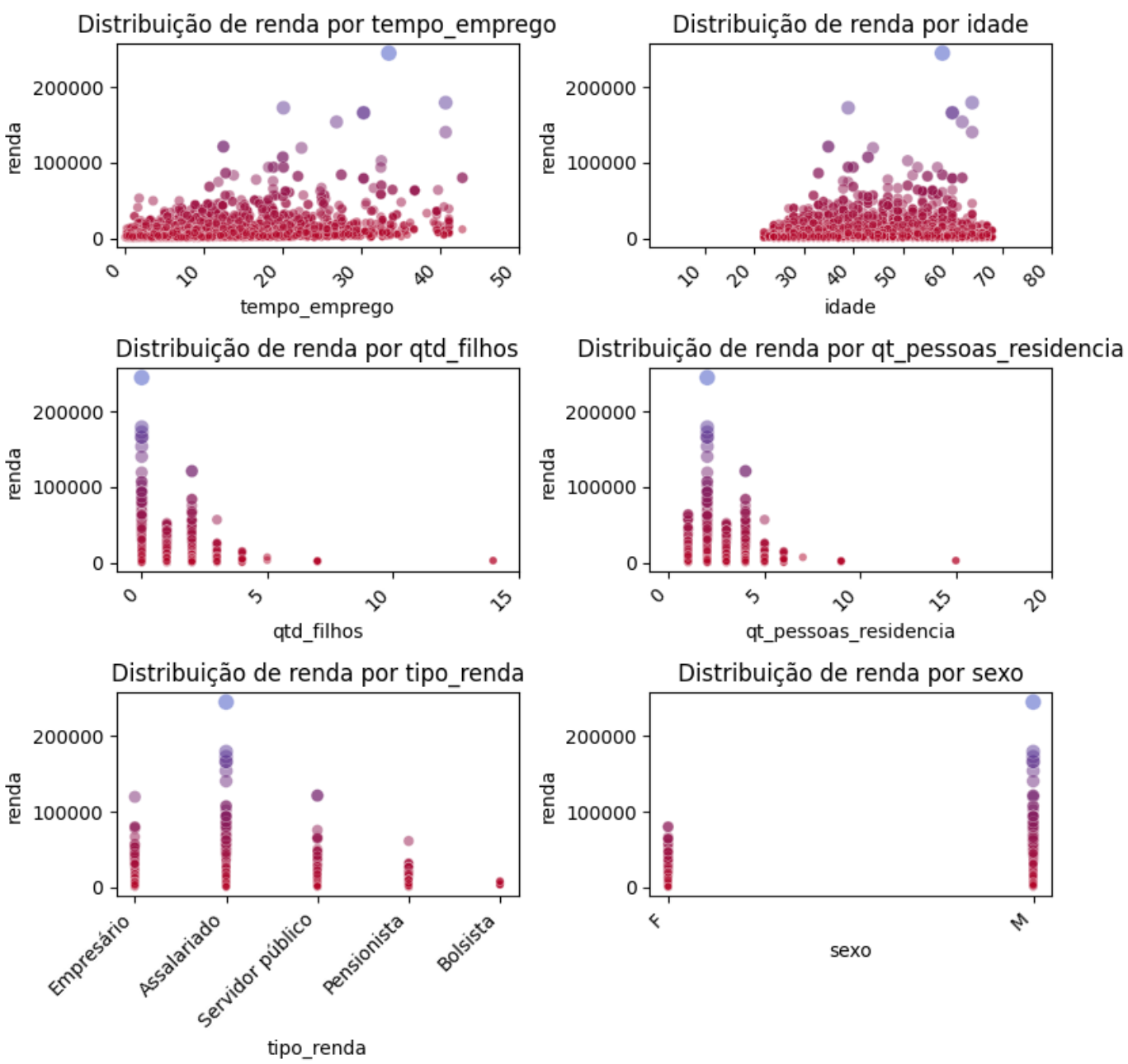


O trabalho realizado com a Base Correlata, facilita ainda mais o entendimento do Mapa de Calor. Onde a escala de cores fornece ciência visualmente, mostrando que, de fato, o Conjunto de Dados apresenta poucas correlações nos dados (células diferentes de branco - com exceção da diagonal vermelha de autocorrelação).

Das poucas células coloridas, a maioria tem cores claras, o que na escala lateral, se mantém muito próximo de 0, indicando que praticamente não há correlação (por isso filtradas anteriormente). As que possuem cor mais intensa, geram as classificações de Forte, Média e Fraca Correlações na Base Correlata.

Além obviamente, da variável tempo_emprego devido a correlação indicada com a variavel-alvo, também foram selecionadas outras duas correlações: a mais forte e a média restritamente entre variáveis distintas. Adicionadas da única outra interação que os dados de renda apresentaram no *Heatmap*, mesmo que retida no filtro de significânica. Estas serão denominadas de *explicativas* a seguir. Próximo trecho é uma análise bivariada das explicativas selecionadas frente alvo.

In [5]:
```
# Reservando as variáveis explicativas para análise bivariada
explicativas = ["tempo_emprego", "idade", "qtd_filhos", "qt_pessoas_residencia", "tipo_r

# Criando subplots para exibição simultânea
fig, axs = plt.subplots(nrows = 3, ncols = 2, figsize = (8, 8))
cores = sns.color_palette("blend:#B40426,#3B4CC0", as_cmap = True) # Personalizando as c

# Iterando sobre as variáveis e criando os gráficos de dispersão frente a renda
for i, explicativa in enumerate(explicativas):
    row = i // 2
    col = i % 2
    sns.scatterplot(x = explicativa,
                    y = "renda",
                    data = df,
                    alpha = 0.5,
                    ax = axs[row, col],
                    size = "renda",
                    hue = "renda",
                    palette = cores)
    axs[row, col].set_title(f"Distribuição de renda por {explicativa}")

    # Condições de ajuste fino na exibição dentre os 6 gráficos
    axs[row, col].legend([], [], frameon = False)
    axs[row, col].set_xticks(axs[row, col].get_xticks())
    axs[row, col].set_xticklabels(axs[row, col].get_xticklabels(), rotation = 45, ha = "
    if df[explicativa].dtype in ["int64", "float64"]:
        axs[row, col].set_xlim(left = -1)

# Ajustando o layout evitando sobreposições
plt.tight_layout()
plt.savefig("./output/renda_x_bivariadas.png")
plt.show()
```



> São nítidas, três classes comportamentais mais similares, através das 6 variaveis explicativas:

linha superior:

- classe denominada **ANOS**
- dados distribuídos de forma mista (independentemente dos anos de trabalho ou de vida);
- concentrados abaixo dos 50000 reais;

linha central:

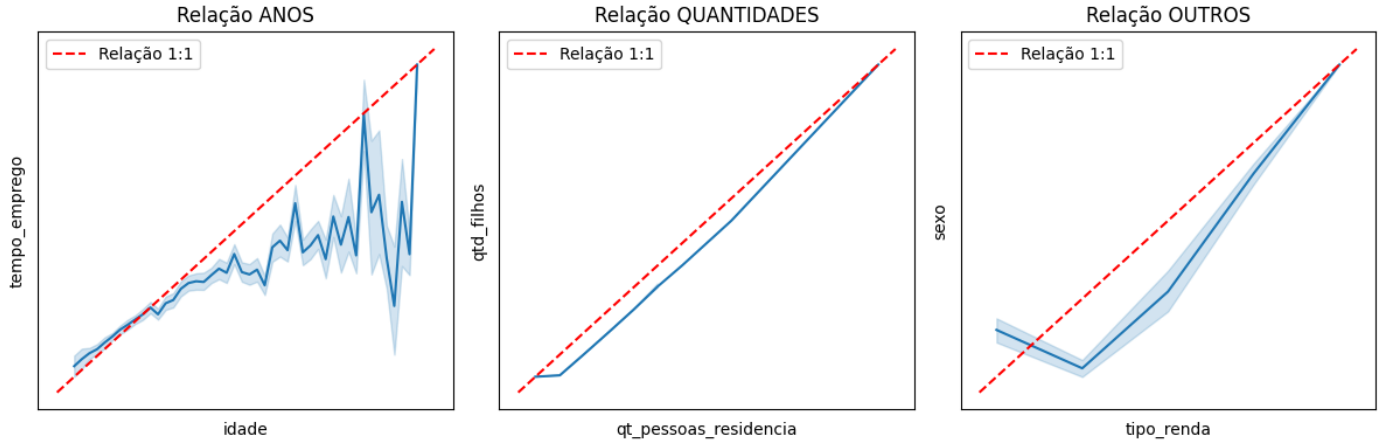
- classe denominada **QUANTIDADES**
- dados deslocados para esquerda (menores quantidades);
- concentrados abaixo dos 50000 reais;

linha inferior:

- classe denominada **OUTROS**
- dados com semelhança nas categorias "assalariado/homem" & "empresário/mulher";
- concentrados abaixo dos 50000 reais;

> Portanto, serão avaliadas internamente entre as classes, em mais uma sequência bivariada, apenas dentre as explicativas.

In [6]:
```
# Separando as classes de comportamentos
anos = ("idade", "tempo_emprego")
quantidades = ("qt_pessoas_residencia", "qtd_filhos")
outros = ("tipo_renda", "sexo")
titulos = ["Relação ANOS", "Relação QUANTIDADES", "Relação OUTROS"]

# Criando subplots para exibição simultânea
fig, axs = plt.subplots(nrows = 1, ncols = 3, figsize = (12, 4))

# Loop para plotar os gráficos
for ax, variaveis, titulo in zip(axs, [anos, quantidades, outros], titulos):
    sns.lineplot(x = variaveis[0], y = variaveis[1], data = df, ax = ax)

    # Ajuste fino na exibição dos gráficos
    ax.plot(ax.get_xlim(), ax.get_ylim(), linestyle = "--", color = "red", label = "Rela
    ax.set_xticks([])
    ax.set_yticks([])
    ax.legend()
    ax.set_title(titulo)

# Ajustando o layout evitando sobreposições
plt.tight_layout()
plt.savefig("./output/separado_bivariadas.png")
plt.show()
```



A relação interna entre as 2 variaveis de cada classe, demonstra uma certa proporcionalidade direta entre elas. Os três casos se mantém "guiados" pela reta de relação linear: Apesar de uma grande oscilação na classe ANOS, ela oscila e retorna ao referencial linear.

> A classe de **ANOS** possui de fato uma leve correlação, como o *heatmap* afirma (+ 0,3) e o gráfico de linhas confirma (apenas acompanha o sentido da linha de tendência e possui margens).
>
> A classe de **QUANTIDADES** possui de fato uma forte correlação quase 1:1, como o *heatmap* afirmou (+ 0,9) e gráfico de linhas confirma (próximo a linha de tendência e sem margens).
>
> A classe de **OUTROS** apresenta uma leve correlação entre algumas de suas categorias, como o *heatmap* afirma (± 0,2) e o gráfico de linhas confirma (próximo a linha de tendência e com margens).

Com isso, podemos inferir que a **RENDA** não se diferencia muito dentro da classe **QUANTIDADES**, mas têm relações distintas e analisáveis dentro das classe **ANOS** e **OUTROS**.

Desta forma, criamos o primeiro esboço da ampla imagem dos dados dos mutuários. A grosso modo, já temos uma ideia de quais colunas interagem entre sí, quais colunas não são uteis, quais dados melhor se relacionam com a renda, etc. Entretanto, o projeto esta apenas começando.

Etapa 3 Crisp-DM: Preparação dos dados

A Preparação de Dados, também conhecida como Pré-Processamento de Dados, é uma etapa fundamental no fluxo de trabalho na Ciência de Dados. Envolve uma série de tarefas que visam tornar os dados brutos mais adequados para análise e modelagem. Uma preparação de dados eficaz é de sumo importância, uma vez que a qualidade e a integridade dos dados impactam diretamente na virtude dos resultados dos modelos e *insights*.
Neste projeto, abordaremos os seguintes passos para o pré-processamento:

TABELA 3 - Etapas de pré-processamento.

Passo	Tarefa	*Status*
Coleta	Obtenção de dados de fontes relevantes	concluída
Integração	União das *n* Bases de Dados a serem utilizadas	N/A
Seleção	Escolha das quais características incluir em cada modelo	pendente
Limpeza	Revisão do Conjunto de Dados analisando sua integridade	pendente
Transformação	Formatação das variáveis para configuração analisável	pendente
Gerenciamento	Identificação, correção e prevenção de erros em processos	pendente
Amostragem	Eleição de subconjunto de dados para economizar tempo e recursos	N/A
Engenharia	Criação de novas métricas, melhorando capacidade de análise	pendente
Normalização	Garantia de contribuição equitativa das variáveis	pendente

Esta etapa pode consumir uma parcela significativa do tempo de um projeto de alto nível, mas é uma etapa crítica para garantir que os resultados obtidos sejam confiáveis e úteis para a tomada de decisões. Além disso, um processo de preparação bem executado pode economizar tempo e esforço durante as fases subsequentes.

In [7]:
```
# As três primeiras tarefas desta etapa estão agrupadas em um encadeamento de métodos ún
# SELEÇÃO, LIMPEZA & TRANSFORMAÇÃO
df_l = (df.drop(["Unnamed: 0", "data_ref"], axis = 1)
        .round({"tempo_emprego": 1})
        .assign(qt_pessoas_residencia = lambda x: x["qt_pessoas_residencia"].astype(int)
        .assign(id_cliente = lambda x: x["id_cliente"].astype(int))
        .sort_values("id_cliente")
        .drop_duplicates("id_cliente")
        .set_index("id_cliente"))

print("A quantidade de linhas e colunas do Conjunto de Dados Limpo é: {}".format(df_l.sh
```

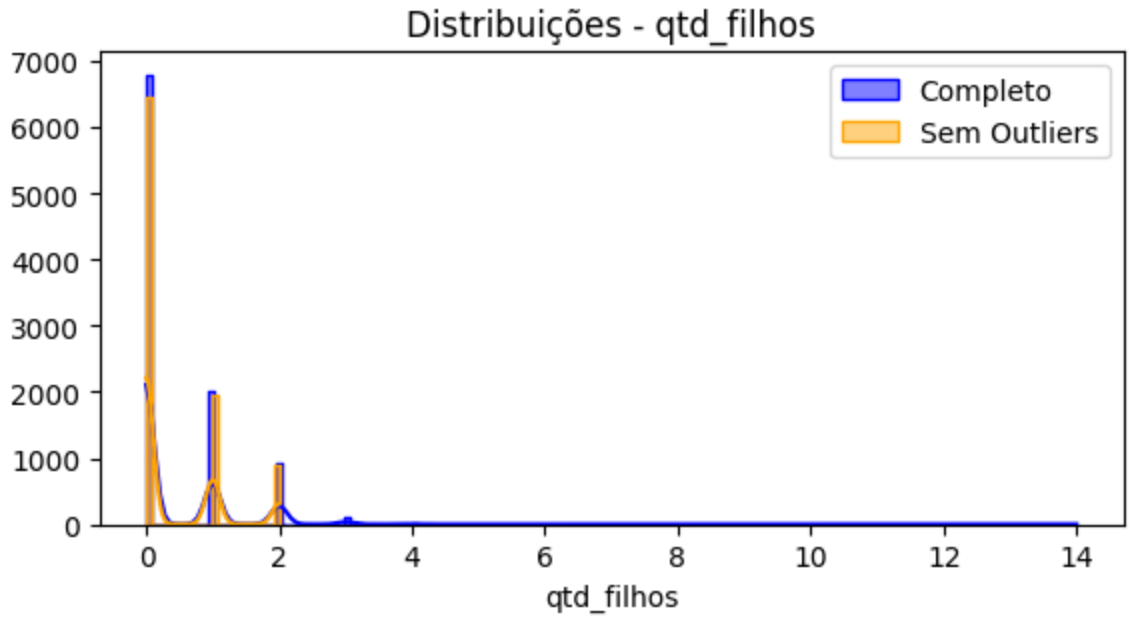
```
A quantidade de linhas e colunas do Conjunto de Dados Limpo é: (9845, 12)
```

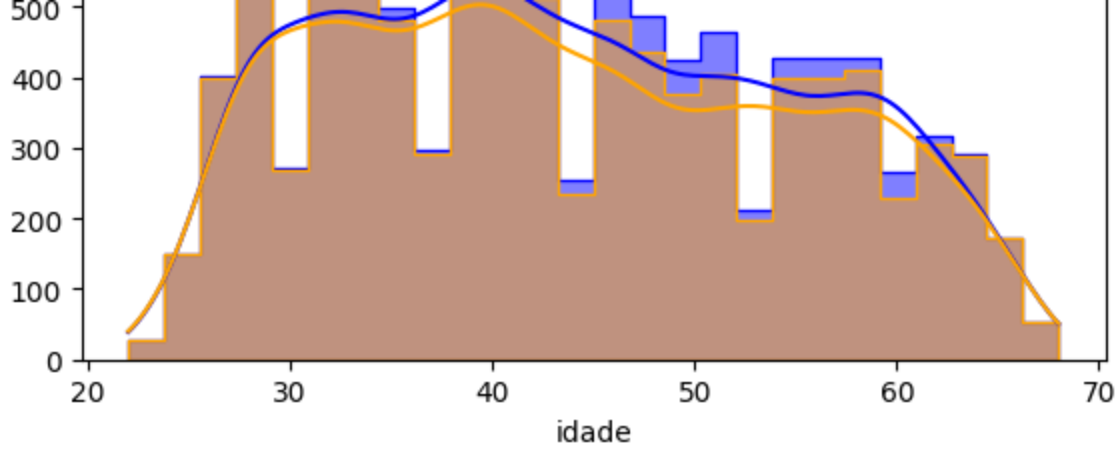
In [8]:
```
# GERENCIAMENTO
```

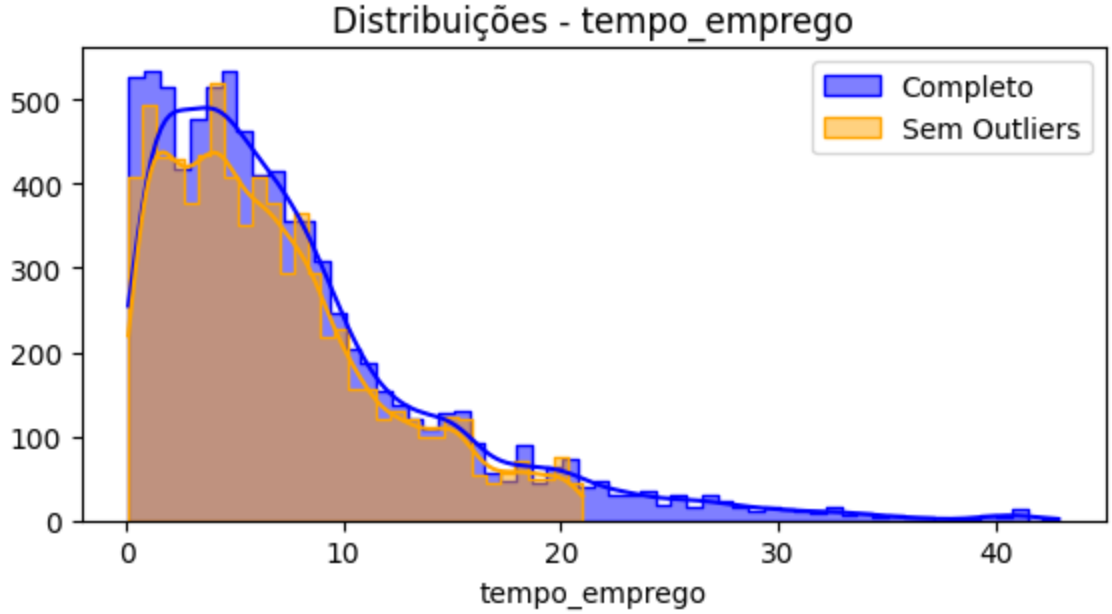
```python
# Trecho para analise e tratamento dos outliers
# Função que identifica a quantidade de outliers e quais linhas que contém tais valores
def id_outliers(coluna):
    Q1 = coluna.quantile(0.25)
    Q3 = coluna.quantile(0.75)
    IQR = Q3 - Q1
    inf = Q1 - 1.5 * IQR
    sup = Q3 + 1.5 * IQR
    outliers = coluna[(coluna < inf) | (coluna > sup)]

    return len(outliers), outliers.index.tolist()

# Lista para armazenar os índices das linhas com outliers a serem removidas
indices_outliers = []
df_outliers = df_l.select_dtypes(include = ["int32", "int64", "float64"]).drop("renda",

# Aplicando a função a cada coluna do DataFrame para contagem de outliers
outliers = df_outliers.apply(id_outliers)
contagem_outliers = outliers.apply(lambda x: x[0])
print(f"- Quantidade de Outliers por coluna:") # Imprimindo as quantidades encontradas
print(contagem_outliers)

# Armazenando os índices dos outliers em uma lista
for coluna in ["qtd_filhos", "tempo_emprego", "qt_pessoas_residencia"]:
    indices_outliers.extend(outliers[coluna][1])

# Removendo as linhas com outliers do DataFrame Limpo
df_l_o = df_l.drop(indices_outliers)

# Calculando estatísticas descritivas para os conjuntos
estatisticas_completas = df_l.describe()
estatisticas_sem_outliers = df_l_o.describe()

# Criando um DataFrame para armazenar as estatísticas comparativas
comparacao_estatisticas = pd.DataFrame({
    "Completo": estatisticas_completas.loc["mean"],
    "Sem Outliers": estatisticas_sem_outliers.loc["mean"],
    "Diferença": estatisticas_completas.loc["mean"] - estatisticas_sem_outliers.loc["mea
})

# Imprimindo as estatísticas comparativas
print(f"\n - Comparação entre as Médias:")
print(round(comparacao_estatisticas, 2))

# Selecionando apenas as colunas numéricas para o teste t de Student
colunas_numericas = df_l.select_dtypes(include = ["int32", "int64", "float64"]).columns

# Testando t de Student e p-valores
for coluna in colunas_numericas:
    estatisticas_t, p_valor = ttest_ind(df_l[coluna], df_l_o[coluna], nan_policy = "omit
    print(f"\n - Teste t para '{coluna}': Estatística t = {estatisticas_t: .2f}, p-valor

# Demonstração visual das distribuições
fig, axes = plt.subplots(nrows = len(colunas_numericas), ncols = 1, figsize = (6, 16))

# Loop para gerar os gráficos e as diferentes distribuições
for i, coluna in enumerate(colunas_numericas):
    sns.histplot(df_l[coluna],
                 kde = True,
                 ax = axes[i],
                 color = "blue",
                 label = "Completo",
                 element = "step")
    sns.histplot(df_l_o[coluna],
                 kde = True,
                 ax = axes[i],
```

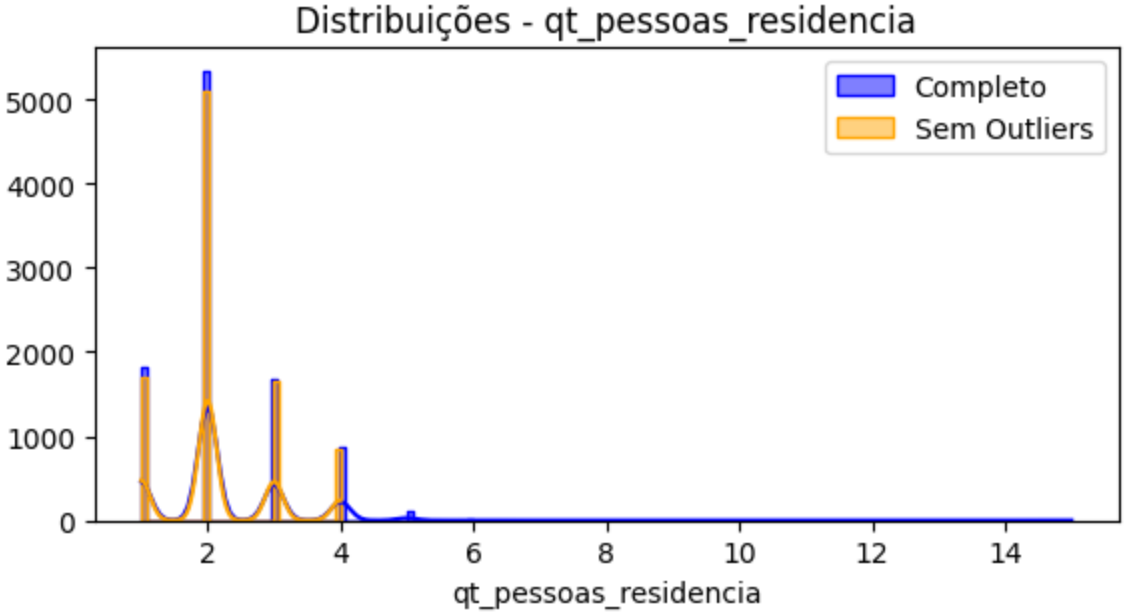
```
                    color = "orange",
                    label = "Sem Outliers",
                    element = "step")

    axes[i].set_title(f"Distribuições - {coluna}")
    axes[i].set_ylabel("")
    axes[i].legend()

# Ajustando o layout evitando sobreposições
plt.tight_layout()
plt.savefig("./output/comp_outliers.png")
plt.show()

# Trecho para preencher as lacunas de "tempo_emprego" devidamente identificados pelo Ale
mediana = df_l["tempo_emprego"].median()
df_l["tempo_emprego"].fillna(mediana, inplace = True)
```

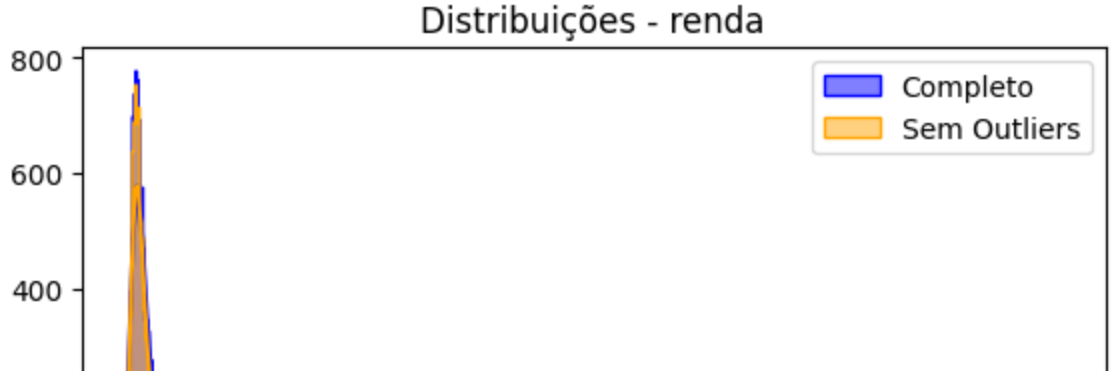
```
- Quantidade de Outliers por coluna:
qtd_filhos               131
idade                      0
tempo_emprego            422
qt_pessoas_residencia    127
dtype: int64

 - Comparação entre as Médias:
                       Completo  Sem Outliers  Diferença
qtd_filhos                 0.43          0.40       0.03
idade                     43.84         43.64       0.20
tempo_emprego              7.75          6.68       1.07
qt_pessoas_residencia      2.21          2.18       0.03
renda                   5699.35       5229.23     470.12

 - Teste t para 'qtd_filhos': Estatística t =  2.99, p-valor =  0.00282

 - Teste t para 'idade': Estatística t =  1.22, p-valor =  0.22115

 - Teste t para 'tempo_emprego': Estatística t =  11.40, p-valor =  0.00000

 - Teste t para 'qt_pessoas_residencia': Estatística t =  2.22, p-valor =  0.02654

 - Teste t para 'renda': Estatística t =  4.28, p-valor =  0.00002
```

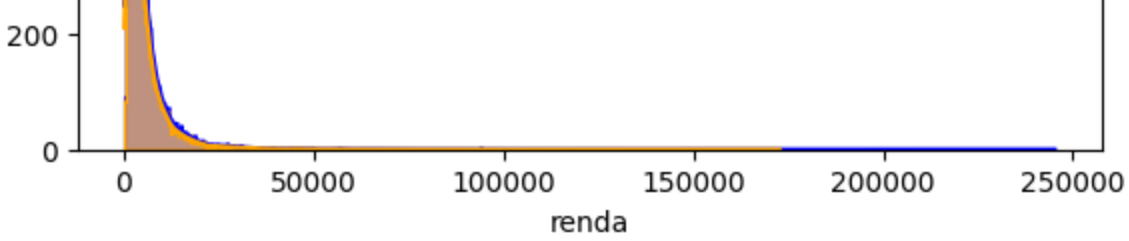



0
100
200
300
400
500
20
30
40
50
60
70
idade


Distribuições - tempo_emprego
Completo
Sem Outliers
0
100
200
300
400
500
0
10
20
30
40
tempo_emprego


Distribuições - qt_pessoas_residencia
Completo
Sem Outliers
0
1000
2000
3000
4000
5000
2
4
6
8
10
12
14
qt_pessoas_residencia


Distribuições - renda
Completo
Sem Outliers
400
600
800



Portanto, com base nos resultados dos **testes t** e **p-valores**, podemos concluir que a remoção de outliers impacta significativamente (α = 5%) nas médias das variáveis "qtd_filhos", "tempo_emprego", "qt_pessoas_residencia" e "renda". Contudo, não há evidências significativas de impacto na média da variável "idade".

Desta forma, seguirei carregando as informações que estes dados oferecem para a análise, uma vez que não há erros de digitação ou absurdos nas medições dos dados dos clientes após a pesquisa conferindo no Relatório de Perfil.

In [9]:
```
# ENGENHARIA
# Criação de dummies para as colunas do tipo texto
objetos = df_l.select_dtypes(include = "object").columns
df_l = pd.get_dummies(df_l, columns = objetos, prefix = objetos)

# Criação de novos dados para cada um dos mutuários
df_l["domiciliar_per_Capita"] = round(df_l["renda"] / df_l["qt_pessoas_residencia"], 2)
df_l["acumulada_Cargo"] = round(df_l["renda"] * df_l["tempo_emprego"], 2)
df_l["filhos_por_Residencia"] = round(df_l["qtd_filhos"] / df_l["qt_pessoas_residencia"]
df_l.head()
```

Out[9]:

id_cliente	posse_de_veiculo	posse_de_imovel	qtd_filhos	idade	tempo_emprego	qt_pessoas_residencia	renda	s
1	False	True	0	52	8.4	1	1938.57	
2	False	True	0	52	8.4	1	5702.28	
3	True	True	0	46	2.1	2	8534.70	
4	True	False	0	29	3.0	2	3087.85	
5	True	False	0	29	3.0	2	690.39	

5 rows × 33 columns

In [10]:
```
# NORMALIZAÇÃO:
numericas = ["qtd_filhos", "idade", "tempo_emprego", "qt_pessoas_residencia"]

# Criação dos métodos
norm = MinMaxScaler()

# Aplicando a normalização nas colunas selecionadas
df_l[numericas] = norm.fit_transform(df_l[numericas])
```

TABELA 4 - Resultados do pré-processamento.

Passo	*Status*
Coleta	concluído

Integração	N/A
Seleção	concluído
Limpeza	concluído
Transformação	concluído
Gerenciamento	concluído
Amostragem	N/A
Engenharia	concluído
Normalização	concluído

> Como todos passos pertinentes à esta Etapa estão concluídas, encerrando-a e seguindo, de fato, para implementação e análises de modelagem preditiva.

Etapa 4 Crisp-DM: Modelagem

A Modelagem de Dados é uma etapa crítica que visa criar modelos eficazes e úteis para resolver problemas específicos de Análise de Dados. Ela requer uma combinação de conhecimento de domínio, habilidades de programação e compreensão das técnicas de modelagem para ter sucesso no desenvolvimento do projeto. Devido ao planejamento bem delineado na Etapa 1, basta agir na prática e implementar os treinamentos, testes, avaliações e gestão de erros dos modelos selecionados.

• Desenho dos conjuntos treinamento e teste

A escolha de usar uma divisão padrão de 80/20 % dos dados, respectivamente, surge de alguns fatores: considerando o tamanho inicial do conjunto de dados (15.000 linhas e 15 colunas), a intenção de maximizar o aprendizado do modelo, enquanto ainda mantenho uma quantidade significativa de dados para os testes dos K-Folds.

• Construção, treinamento e avaliação dos modelos

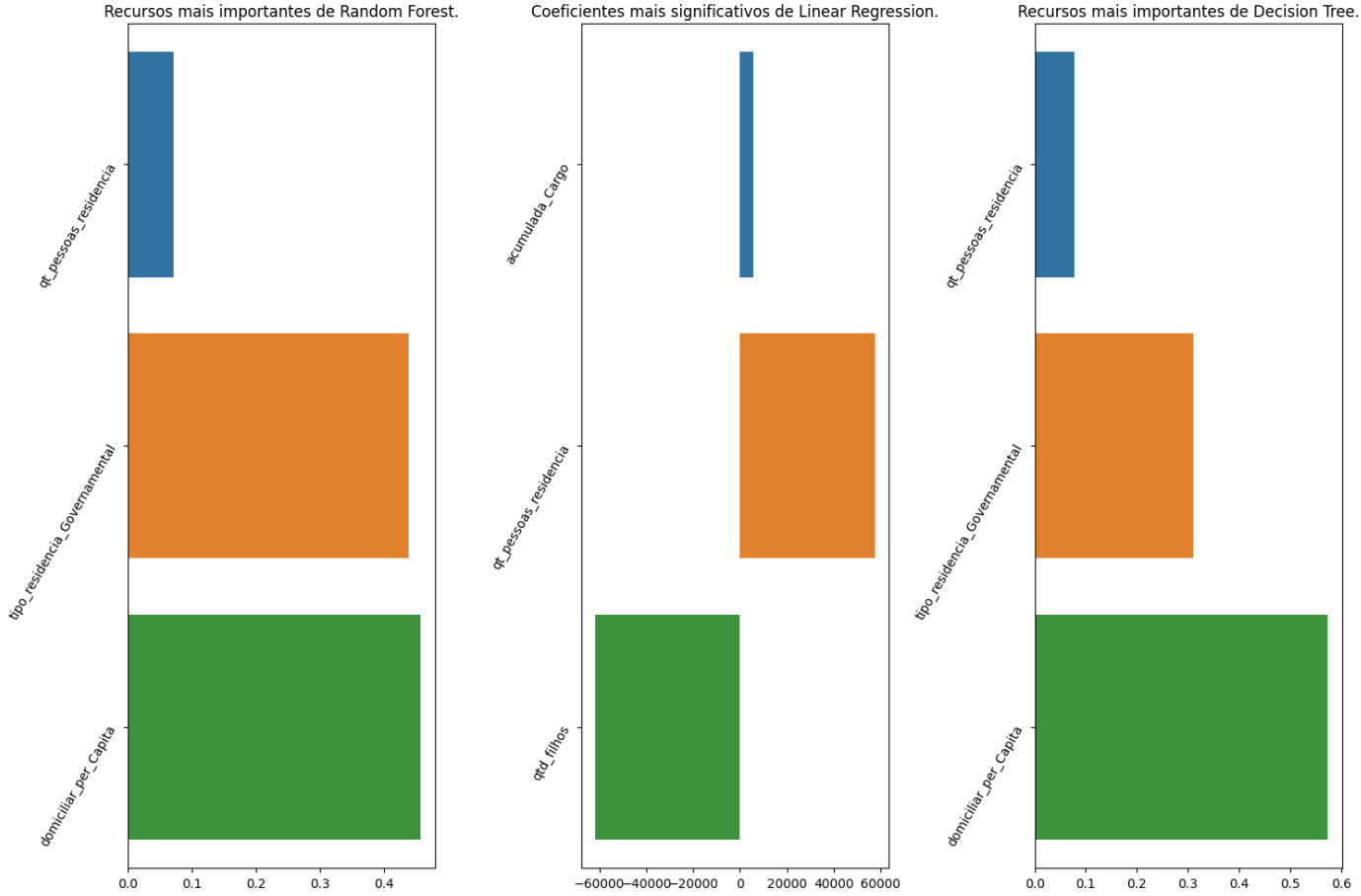
In [11]:
```
# Mensurando o tempo de Construção
inicio = time.time()

# Separando as componentes X e o alvo Y para a modelagem
X = df_l.drop(["renda"], axis = 1).values
Y = df_l["renda"].values

# Separando os conjuntos Treinamento & Teste
X_treino, X_teste, y_treino, y_teste = train_test_split(X, Y, test_size = 0.2, random_st

# Cross-Validation 5-Fold usando embaralhamento dos dados para assegurar homogeneidade
kf = KFold(n_splits = 5, shuffle = True, random_state = 42)

# Trecho de criação dos auxiliares para apoio posterior
treinamento = {}
predicao = {}
comparacao_performance = []
nao_atende = []

# Inicialização dos hiperparâmetros a serem usados na otimização dos modelos
# foram testadas mais de 30 combinações e peneirados valores até obtenção dos seguintes
```

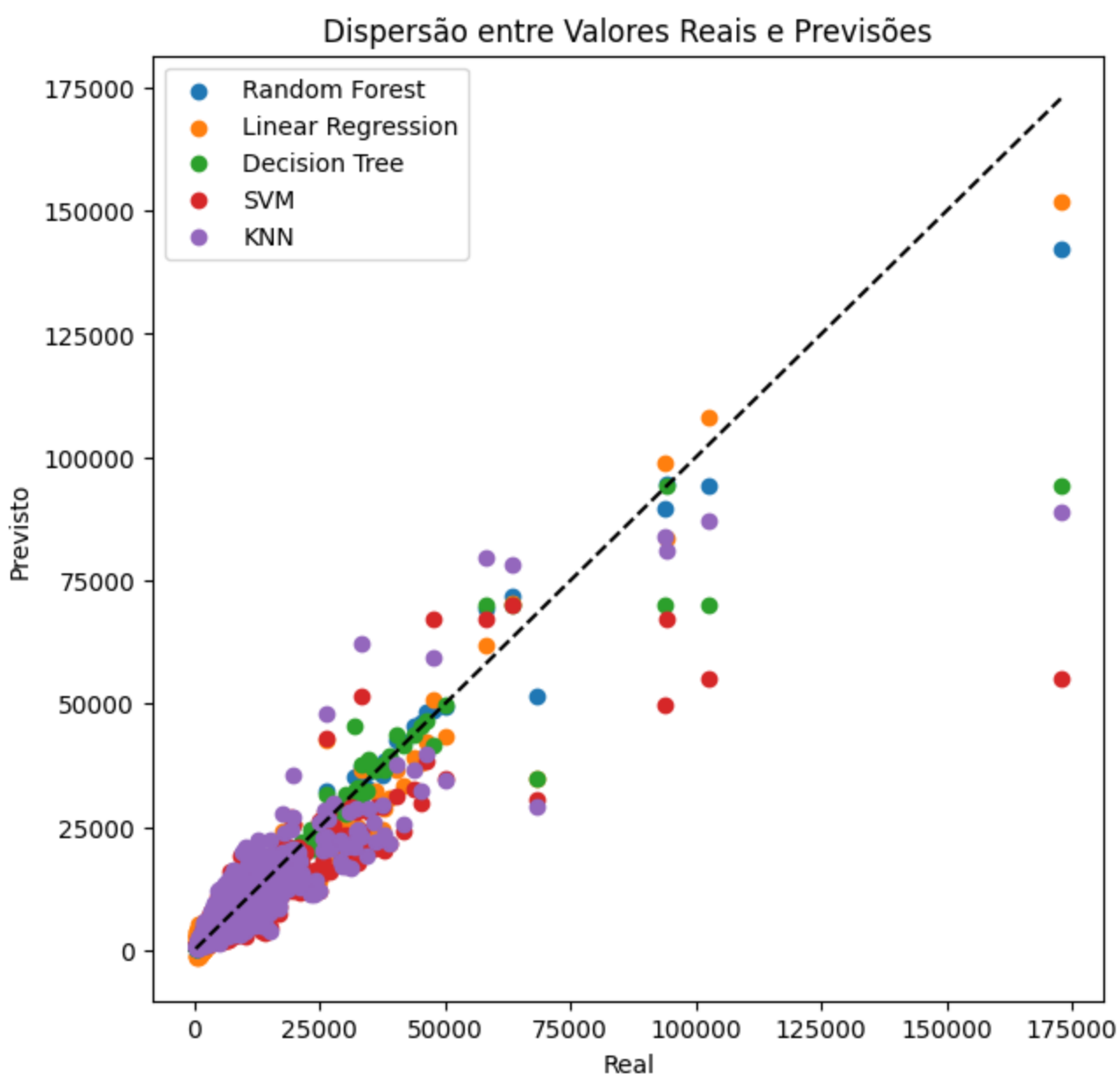
```python
parametros_rf = {"n_estimators": 200,
                 "min_samples_split": 3,
                 "min_samples_leaf": 1}

parametros_lr = {"fit_intercept": False,
                "copy_X": True,
                "positive": False}

parametros_dt = {"min_samples_split": 2,
                 "min_samples_leaf": 1}

parametros_svm = {"C": 7000,
                  "epsilon": 0.0001,
                  "kernel": "rbf"}

parametros_knn = {"n_neighbors": 7,
                  "weights": "distance",
                  "p": 1}

parametros_modelos = {"Random Forest": parametros_rf,
                      "Linear Regression": parametros_lr,
                      "Decision Tree": parametros_dt,
                      "SVM": parametros_svm,
                      "KNN": parametros_knn}

# Inicializando os modelos de regressão
modelos = {
    "Random Forest": RandomForestRegressor(**parametros_rf),
    "Linear Regression": LinearRegression(**parametros_lr),
    "Decision Tree": DecisionTreeRegressor(**parametros_dt),
    "SVM": SVR(**parametros_svm),
    "KNN": KNeighborsRegressor(**parametros_knn)
}

# Elemento gráfico para exibir a dispersão das previsões
fig, axes = plt.subplots(1, len(modelos)-2, figsize=(15, 10))

# Loop para treinamento e avaliação de cada modelo
for idx, (nome_modelo, modelo) in tqdm(enumerate(modelos.items()),
                                       desc = "Modelando/Otimizando",
                                       unit = "modelo",
                                       total = len(modelos)):
    # Lendo o modelo no Loop
    parametros_modelo = parametros_modelos[nome_modelo]

    # Treinando o modelo
    treinamento[nome_modelo] = modelo.fit(X_treino, y_treino)

    # Avaliando sua adaptação no conjunto de teste
    pontuacao = modelo.score(X_teste, y_teste)

    # Avaliar o desempenho do treinamento através dos 5-Fold
    resultados = cross_val_score(modelo,
                                 X_treino,
                                 y_treino,
                                 cv = kf,
                                 scoring = "neg_mean_squared_error")

    # Calculos de performance do treino nos 5-Fold
    rmse_valid = [(-resultado) ** 0.5 for resultado in resultados]
    mae_valid = [-resultado for resultado in resultados]

    # Estatisticas da performance nos conjuntos de Validação
    rmse_medio = np.mean(rmse_valid)
    σ_rmse = np.std(rmse_valid)
    mae_medio = np.mean(mae_valid)
```

```
    σ_mae = np.std(mae_valid)

    # Avaliando o modelo no conjunto de teste
    previsao_teste = modelo.predict(X_teste)
    rmse_teste = mean_squared_error(y_teste, previsao_teste, squared = False)
    mae_teste = mean_absolute_error(y_teste, previsao_teste)

    # Adicionando resultados à lista
    comparacao_performance.append({"Modelo": nome_modelo,
                                   "R²": round(pontuacao, 3),
                                   "RMSE Médio (5-Fold)": round(rmse_medio),
                                   "DesvPad RMSE": round(σ_rmse),
                                   "RMSE do Teste": round(rmse_teste),
                                   "MAE Médio (5-Fold)": round(mae_medio),
                                   "DesvPad MAE": round(σ_mae),
                                   "MAE do Teste": round(mae_teste)})

    # Adicionando previsões ao dicionário predicao
    predicao[nome_modelo] = previsao_teste

    # Verificando se o modelo tem coeficientes
    if hasattr(modelo, "coef_"):
        # Obtendo os índices dos 3 coeficientes mais significativos
        top_coef_idx = np.argsort(np.abs(modelo.coef_))[-3:]
        x_values = modelo.coef_[top_coef_idx]
        y_values = df_l.columns[:-1][top_coef_idx]
        sns.barplot(x = x_values, y = y_values, ax = axes[idx])
        axes[idx].set_title(f"Coeficientes mais significativos de {nome_modelo}.")
        axes[idx].set_yticklabels(axes[idx].get_yticklabels(), rotation = 60)

    # Verificando se o modelo tem características de importância
    elif hasattr(modelo, "feature_importances_"):
        # Obtendo os índices dos 3 features mais importantes
        feature_importance = modelo.feature_importances_
        top_features_idx = np.argsort(feature_importance)[-3:]
        sns.barplot(x = feature_importance[top_features_idx], y = df_l.columns[:-1][top_
        axes[idx].set_title(f"Recursos mais importantes de {nome_modelo}.")
        axes[idx].set_yticklabels(axes[idx].get_yticklabels(), rotation = 60)

    # Registrando o não atendimento das condições prévias
    else:
        nao_atende.append({nome_modelo})

print(f"Os modelos não suportados para análise gráfica foram: {nao_atende}")

# Gráfico de barras exibindo os maiores influenciadores dos modelos
plt.tight_layout()
plt.savefig("./output/top3_influenciadores.png")
plt.show()

# Gráfico de dispersão para todos os modelos
plt.figure(figsize = (7, 7))
for nome_modelo, previsoes in predicao.items():
    plt.scatter(y_teste, previsoes, label = nome_modelo)

# Configurando o gráfico
plt.plot([min(y_teste), max(y_teste)], [min(y_teste), max(y_teste)], "--k")
plt.xlabel("Real")
plt.ylabel("Previsto")
plt.title("Dispersão entre Valores Reais e Previsões")
plt.legend()
plt.savefig("./output/dispersao_previsoes.png")
plt.show()

# Criando um DataFrame a partir da lista de resultados
performance_df = pd.DataFrame(comparacao_performance).set_index("Modelo")
```

```
# Calculando o tempo de execução
fim = time.time()
tempo = (fim - inicio) / 60
print(f"Tempo Total: {tempo: .1f} minutos.")

performance_df
```

```
Modelando/Otimizando: 100%|██████████| 5/5 [03:37<00:00, 43.40s/modelo]
Os modelos não suportados para análise gráfica foram: [{'SVM'}, {'KNN'}]
```






```
Tempo Total:  3.7 minutos.
```

Out[11]:

Modelo	R^2	RMSE Médio (5-Fold)	DesvPad RMSE	RMSE do Teste	MAE Médio (5-Fold)	DesvPad MAE	MAE do Teste
Random Forest	0.987	1642	882	909	3473625	3677246	80
Linear Regression	0.932	2616	352	2075	6967367	1880487	1237
Decision Tree	0.924	2072	847	2195	5009918	3596270	143
SVM	0.741	4461	1684	4048	22733635	18740475	1549
KNN	0.821	3675	775	3362	14102322	6501545	1422

Etapa 5 Crisp-DM: Avaliação dos resultados

- **Considerações sobre os Desempenhos:**

As variações significativas nos desempenhos dos diferentes K-Folds da Validação Cruzada, resulta em uma média dos 5-Fold mais alta. Isso indica que os modelos tem um desempenho variável em diferentes subconjuntos dos dados de treinamento.

A diferença entre média dos 5-Fold e o Teste, também fornece *insights* sobre a capacidade de generalização do modelo. Se no Teste, for significativamente menor, isso sugere que o modelo está se saindo bem em dados não vistos, indicando uma boa capacidade de previsão. Ao mesmo tempo que modelos mais sensíveis a variações nos dados de validação, podem mostrar diferenças notáveis entre o desempenho médio nos Folds e no conjunto de teste.

A grande proporção distinta entre RMSE e MAE era esperada, uma vez que tratam *Outliers* com penalizações diferentes.

• **Avaliações individuais:** >

Random Forest: Definitivamente teve os melhores resultados globais no projeto. Boa adaptação ao conjunto teste, bons resultados preliminares de validação dentro dos 5-Folds. Seus resultados deverão apresentar peso significativo na próxima etapa, durante a construção do *Score*.

> **Regressão Linear**: Como seu resultado RMSE Teste foi desproporcionalmente menor do que o MAE Teste, indica que o modelo é menos sensível a *Outliers*. Apesar de apresentar maior estabilidade dentre os conjuntos de Validação, apresentando o menor Desvio-Padrão, tanto do RMSE quanto do MAE, não teve muito êxito nas previsões do conjunto teste; mesmo com relativamente alto R^2 Teste.
>
> **Árvore de Decisão**: Segundo melhor modelo durante a avaliação individual. Apesar de ser mais estável do que o Random Forest nos conjuntos de Validação, obteve previsões levemente menos acuradas no Teste. Como seu resultado RMSE Teste foi desproporcionalmente maior do que o MAE Teste, indica que o modelo é mais sensível a *Outliers*.
>
> **SVM** O modelo com pior classificação individual neste projeto. Demandou exaustão para obter os melhores hiperparâmetro para começar a ser utilizável, isto é, com os hiperparâmetros padrões, estava com R^2 de -1%. Porém, com o devido esforço e persistência, deixou de ser "descartável" do estudo, a ponto de possuir estatísticas consideravelmente próximas ao K-NN (2º pior individualmente) na previsão de renda nos conjuntos treinamento & teste.
>
> **KNN**: Mesmo exibindo estabilidade relativa durante a Validação, teve previsões Teste próximas do SVM. O R^2 abaixo dos 90% também mostra que o modelo consta na parte inferior da tabela de classificação individual, obtendo os segundo piores resultados gerais na previsão de renda com os dados do conjunto tratado.

• **Avaliações conjuntas:**

O gráfico Dispersão Real x Previsão, mostra que independentemente da performance do modelo, todos apresentaram maior dificuldade em aproximar corretamente a renda conforme ela cresce muito. Fato que não seria encontrado caso os *Outliers* de Renda tivessem sido removidos. Talvez pelo próprio fato de haver poucos valores altos (*Outliers*) nos conjuntos de treinamento, dificultando o aprendizado dos modelos em avaliar tais dados.

De acordo com o gráfico de significância das colunas, Random Forest e Árvore de Decisão mostraram exatamente as mesmas variáveis. Isso pode revelar que tais fatores (**qt_pessoas_residencia**, **domiciliar_per_Capita** e **tipo_residencia_Governamental**) são mais decisivos na previsão de renda quando estão presentes na ficha do cliente. A variável-explicativa **qt_pessoas_residencia** foi exibida nessa etapa para os 3 modelos, sugerindo que ela, apesar de não possuir correlação inicial no conjunto de dados brutos, pode estar instrínsicamente relacionada com a possível previsão de renda do mutuário, em algum nível mais profundo.

Em menos de 5 minutos, é possível gerar os pesos dos modelos para um conjunto de dados neste porte. Tempo será menor para uma cartela de clientes reduzida, bem como será maior caso a cartela de mutuários seja ampliada. De qualquer forma, o projeto consegue exercer rapidamente sua função com excelência e eficiência.

• **Construção dos Pesos - Avaliação Final:**

In [12]:
```
# Dataframe que trará os pesos de cada modelo
pesos = pd.DataFrame()

# Loop para calcular os pesos conforme as performances
for coluna in performance_df.columns:
    if coluna not in ["Modelo", "R²"]: # O R² é usado como penalizador percentual no pes
        pesos["Peso "+coluna] = 1 - (performance_df[coluna] / performance_df[coluna].sum

# Calculo de Peso Médio e Final
pesos["Média dos Pesos"] = pesos.mean(axis = 1)
pesos["Peso Final do Modelo"] = performance_df["R²"] * pesos["Média dos Pesos"]

round(peso, 3)
```

Out[12]:

Modelo	Peso RMSE Médio (5-Fold)	Peso DesvPad RMSE	Peso RMSE do Teste	Peso MAE Médio (5-Fold)	Peso DesvPad MAE	Peso MAE do Teste	Média dos Pesos	Peso Final do Modelo
Random Forest	0.886	0.806	0.928	0.934	0.893	0.982	0.905	0.893
Linear Regression	0.819	0.922	0.835	0.867	0.945	0.721	0.852	0.794
Decision Tree	0.857	0.813	0.826	0.904	0.895	0.968	0.877	0.811
SVM	0.692	0.629	0.678	0.565	0.455	0.650	0.612	0.453
KNN	0.746	0.829	0.733	0.730	0.811	0.679	0.755	0.620

Etapa 6 Crisp-DM: Implantação

Para implementar um motor de decisões automatizadas utilizando o *Score* desenvolvido como etapa final do Crisp-DM, foi criado 3 arquivos .csv contendo apenas uma linha ("cliente X", "cliente Y" & "cliente Z"), contendo informações das 15 variáveis iniciais para simulações na utilização do *Script* de Decisão, como

parte de uma avaliação para apoio na tomada de decisão da aceitação ou rejeição de crédito frente aos dados pessoais e de renda.
O valor limiar para aceitação/rejeição é estabelecido de acordo com a escolha do mutuário, conforme sua situação e necessidade.

Tais clientes não estão presentes no conjunto de dados de mutuários e gostariam de saber se devem aceitar a proposta de crédito. Fato que exige a consideração de que considera-se cenários mais desafiadores para avaliar a robustez dos modelos, agregando mais ciência e vigor ao projeto.

In [13]:
```
# Script para implementar a automatização de apoio a decisão
# Criando a função que será utilizada para dar suporte ao cliente
def calcula_score(simulacao, limiar_decisao, modelos):
    # Prevenção contra lacunas preenchendo com valores medianos
    simulacao = simulacao.fillna(simulacao.median(numeric_only = True))

    # Tratamento dos dados do cliente seguindo a estrutura do projeto
    simulacao = (simulacao.drop(["Unnamed: 0", "data_ref"], axis = 1)
                 .round({"tempo_emprego": 1})
                 .assign(qt_pessoas_residencia = lambda x: x["qt_pessoas_residencia"].as
                 .assign(id_cliente = lambda x: x["id_cliente"].astype(int))
                 .set_index("id_cliente"))

    objetos = simulacao.select_dtypes(include = "object").columns

    simulacao = pd.get_dummies(simulacao, columns = objetos, prefix = objetos)
    simulacao["domiciliar_per_Capita"] = round(simulacao["renda"] / simulacao["qt_pessoa
    simulacao["acumulada_Cargo"] = round(simulacao["renda"] * simulacao["tempo_emprego"]
    simulacao["filhos_por_Residencia"] = round(simulacao["qtd_filhos"] / simulacao["qt_p
    simulacao = simulacao.reindex(columns = df_1.columns, fill_value = 0)
    simulacao[numericas] = norm.fit_transform(simulacao[numericas])

    # Inicio da avaliação das previsões seguindo a estrutura do projeto
    performance = []

    X = simulacao.drop(["renda"], axis = 1).values
    Y = simulacao["renda"].values

    for nome_modelo, modelo in modelos.items():
        previsao_simulacao = modelo.predict(X)

        rmse = mean_squared_error(Y, previsao_simulacao, squared = False)

        desempenho = {"Modelo": nome_modelo,
                      "Erro": rmse}

        performance.append(desempenho)

    df_desempenho = pd.DataFrame(performance)

    resultado = pd.DataFrame({"Modelo": df_desempenho["Modelo"]})

    # Calculo dos resultados normalizados seguindo a estrutura do projeto
    for modelo in df_desempenho.items():
        resultado["Resultado"] = 1 - (df_desempenho["Erro"] / df_desempenho["Erro"].sum(

    # Auxiliar para mesclagem das tabelas existentes e aplicar as penalizações posterior
    aux = pd.merge(resultado, pesos, left_on = "Modelo", right_on = "Modelo")

    # Penalização nos modelos conforme seu desempenho nas simulações
    for i in range(len(df_desempenho)):
        # Sem reduções para erros até 30% da renda
        if df_desempenho["Erro"].iloc[i] < (0.3 * Y):
            aux.loc[i, "Resultado"] = aux.loc[i, "Resultado"]
```

```
        # Dedução de 30% para erros entre 30 & 70% da renda
        elif (0.3 * Y) <= df_desempenho["Erro"].iloc[i] < (0.7 * Y):
            aux.loc[i, "Resultado"] = 0.7 * aux.loc[i, "Resultado"]

        # Reduzindo pela metade o resultado caso a previsão erre de 70 a 100% do valor d
        elif (0.7 * Y) <= df_desempenho["Erro"].iloc[i] <= Y:
            aux.loc[i, "Resultado"] = 0.5 * aux.loc[i, "Resultado"]

        # Penalização de 70% para erros superiores ao total de renda
        else:
            aux.loc[i, "Resultado"] = 0.3 * aux.loc[i, "Resultado"]

    # Calculo do score normalizado para comparação com o parâmetro do cliente
    score = (aux["Peso Final do Modelo"] * aux["Resultado"]).sum() / aux["Peso Final do

    if score >= limiar_decisao:
        decisao = "Aprovar a solicitação."
    else:
        decisao = "Rejeitar a solicitação."

    # Calculo adicional para aprofundamento do apoio ao cliente
    dif = abs(score - limiar_decisao)

    # Sessão de exibição dos resultados
    print(decisao)
    print(f"\n -> O seu Score é{score * 100: .1f} de 100.00 pontos.")
    print(f"\n Você possui ***{dif * 100: .1f}*** pontos referente ao seu parâmetro de c
    print("• Abaixo de 5 pontos: é necessário conversar com o seu Gerente.")
    print("• De 5 a 10 pontos: recomenda-se conversar a Gerência.")
    print("• Acima de 10 pontos: é seguro aceitar o Crédito.")

    return score, decisao
```

- ***Background* dos clientes X, Y & Z:**

> O *Cliente X* é uma servidora pública que atualmente mora com sua filha, mas tem uma união estável com um rapaz de casa própria. Ela mora longe dele e esta pensando em pegar crédito para financiar um carro e poder visitar ele e/ou viajar com sua família. Como deseja muito que isso aconteça, decidiu usar o **calcula_score** com nota de corte para 50%, pois recebe um bom salário e acredita que todos merecem essa oportunidade.
>
> O *Cliente* Y é um assalariado, solteiro e que mora sozinho em sua casa. Apesar de estar quase a 8 anos na empresa, não conseguiu progredir na carreira. Possui um carro a prestações e o aluguel da residência. Pela sua idade, e já tendo viajado para o exterior, sente que precisa conhecer outro continente. Quer usar o **calcula_score** para ter uma opinião se deve aceitar a oferta de crédito para viajar. No entanto, tem receio devido as suas contas e proporção salarial. Desta forma, pensa em usar com precaução, o parâmetro de 80% de garantia no **calcula_score**.
>
> O *Cliente Z* é uma empresária que ficou viúva do marido faz 1 ano, estando casados a 24 anos. Retornou a morar com os pais por causa disso, uma vez que o casal de filhos tem 20 e 22 anos, e moram com suas respectivas famílias. Sua empresa tem mais uma década e começou a fazer muito sucesso recentemente, assim, sua renda mensal atingiu 5 dígitos. Ela sente que, apesar de amar muito morar novamente com os pais e cuidar deles, necessita retomar sua vida e comprar um apartamento perto deles. Logo, recorreu para o **calcula_score**

> para ter um norte, e devido a sua situação, optou pelo valor de 60% de corte na sua decisão de aceitar ou rejeitar o crédito que precisa para comprar sua residência própria.

A seguir, serão realizadas as simulações para os dados pessoais e financeiros de cada cliente, conforme os cenários estabelecidos. Cada avaliação será feita separadamente, uma vez que a simulação busca ser o mais realista o possível.

In [14]:
```
# SIMULACAO 1
simulacao_1 = pd.read_csv("./input/cliente_X.csv")
score_cliente_X = calcula_score(simulacao_1, 0.5, modelos)
```

```
Aprovar a solicitação.

 -> O seu Score é 58.7 de 100.00 pontos.

 Você possui *** 8.7*** pontos referente ao seu parâmetro de corte.

• Abaixo de 5 pontos: é necessário conversar com o seu Gerente.
• De 5 a 10 pontos: recomenda-se conversar a Gerência.
• Acima de 10 pontos: é seguro aceitar o Crédito.
```

In [15]:
```
# SIMULACAO 2
simulacao_2 = pd.read_csv("./input/cliente_Y.csv")
score_cliente_Y = calcula_score(simulacao_2, 0.8, modelos)
```

```
Rejeitar a solicitação.

 -> O seu Score é 79.6 de 100.00 pontos.

 Você possui *** 0.4*** pontos referente ao seu parâmetro de corte.

• Abaixo de 5 pontos: é necessário conversar com o seu Gerente.
• De 5 a 10 pontos: recomenda-se conversar a Gerência.
• Acima de 10 pontos: é seguro aceitar o Crédito.
```

In [16]:
```
# SIMULACAO 3
simulacao_3 = pd.read_csv("./input/cliente_Z.csv")
score_cliente_Z = calcula_score(simulacao_3, 0.6, modelos)
```

```
Aprovar a solicitação.

 -> O seu Score é 68.5 de 100.00 pontos.

 Você possui *** 8.5*** pontos referente ao seu parâmetro de corte.

• Abaixo de 5 pontos: é necessário conversar com o seu Gerente.
• De 5 a 10 pontos: recomenda-se conversar a Gerência.
• Acima de 10 pontos: é seguro aceitar o Crédito.
```