Instituto Federal de Educação, Ciência e Tecnologia do Sul de Minas

TÓPICOS EM SISTEMAS INTELIGENTES

Engenharia de Computação - 8º Período

Computação Financeira - Parte 3

LUAN ANTONIO DOMINGOS VENANCIO

12 de Dezembro de 2023

1. Metodologia

O framework para criação de estratégias de investimento automatizado foi desenvolvido utilizando Aprendizado de Máquina para tomada de decisão de compra e venda de ações da empresa Localiza (RENT3). A etapa de simulação do retorno foi implementada em um projeto Java feito na IDE Netbeans.

A etapa dois, que tem como objetivo, criação de um Ensemble Learning e Stacking utilizando validação com janela deslizante, a linguagem escolhida foi Python, e todo o código foi feito na ferramenta Google Colab. As bibliotecas utilizadas foram:

- Pandas: Estruturas de dados e operações para manipular e analisar dados em tabelas;
- NumPy: Estruturas de dados e operações para trabalhar com matrizes e arrays;
- Scikit-learn: Algoritmos de aprendizado de máquina;
- ta: Análise técnica para conjuntos de dados de séries temporais financeiras.

2. Algoritmos *Baseline*

2.1 *Baseline* de classificação

Para realização do algoritmo de baseline de classificação, foi verificada a distribuição da classe majoritária no conjunto de treinamento através do método *value_counts()*.

```
y_pred_base_classifier = pd.DataFrame(y_test)
y_pred_base_classifier.value_counts()
```

```
Output
1         211
0         206
dtype: int64
```

O valor que apareceu mais vezes foi o 1. Portanto, todos os valores foram alterados para 1.

```
y_pred_base_classifier['Output'] = 1
y_pred_base_classifier
```

	Output
2907	1
2908	1
2909	1
2910	1
2911	1
...	...
3319	1
3320	1
3321	1
3322	1
3323	1

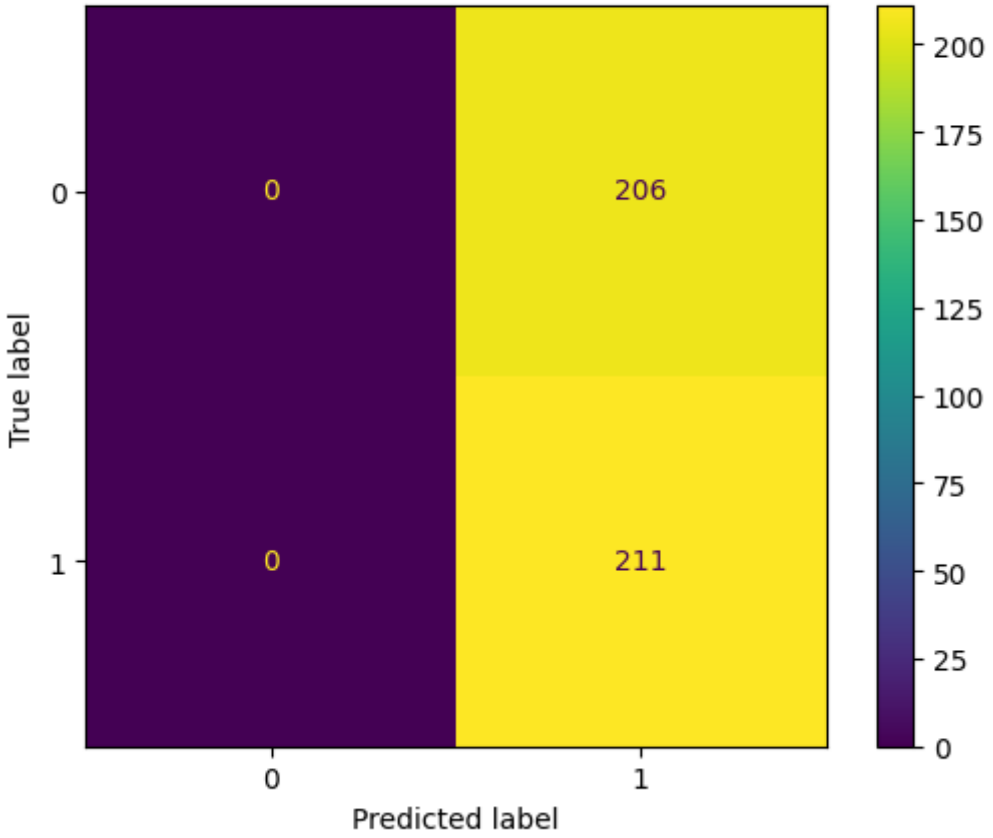
417 rows × 1 columns

Para avaliar os modelos de classificação foi criado uma função chamada evaluate_classification, que retorna as métricas de Acurácia, Precisão, Recall e AUC:

```python
from sklearn.metrics import recall_score, precision_score, accuracy_score, roc_auc_score, confusion_matrix, ConfusionMatrixDisplay

def evaluate_classification(y_test, y_pred):
  cm = confusion_matrix(y_test, y_pred)

  ConfusionMatrixDisplay(confusion_matrix=cm).plot();

  accuracy = accuracy_score(y_test, y_pred)
  precision = precision_score(y_test, y_pred)
  recall = recall_score(y_test, y_pred)
  AUC = roc_auc_score(y_test, y_pred)

  print("Accuracy:", accuracy)
  print("Precision:", precision)
  print("Recall:", recall)
  print("AUC:", AUC)
```

O resultado do baseline de classificação foi:

```
Accuracy: 0.5059952038369304
Precision: 0.5059952038369304
Recall: 1.0
AUC: 0.5
```



O Baseline tem um AUC de 0.5 e um Recall de 1 que bate com o esperado.

2.2 *Baseline* de regressão

Para realização do algoritmo de *baseline* de regressão, equivalente à previsão do retorno do dia anterior como previsão do dia seguinte, através do método shift().

	Return
2907	-0.29
2908	1.47
2909	-0.19
2910	0.42
2911	0.97
...	...
3319	2.52
3320	0.15
3321	-0.78
3322	0.17
3323	0.22

417 rows × 1 columns

```
y_pred_base_regressor = pd.DataFrame(y_test).shift(1, fill_value=0)
y_pred_base_regressor
```

	Return
2907	0.00
2908	-0.29
2909	1.47
2910	-0.19
2911	0.42
...	...
3319	0.87
3320	2.52
3321	0.15
3322	-0.78
3323	0.17

417 rows × 1 columns

Para avaliar os modelos de regressão foi criado uma função chamada ***evaluate_regressor()***, que retorna as métricas de RMSE e MAE:

```
from sklearn.metrics import mean_squared_error, mean_absolute_error

def evaluate_regressor(y_test, y_pred):
  test_set_rmse = np.sqrt(mean_squared_error(y_test, y_pred))
  print('RMSE = ', test_set_rmse)

  mae = mean_absolute_error(y_test, y_pred)
  print("MAE = ", mae)
```

O resultado do baseline de regressão foi:

```
RMSE =  1.8736469778487927
MAE =  1.4565947242206225
```

3. Algoritmo de Aprendizado de Máquinas

3.1 Classificação

O modelo de classificação emprega cinco algoritmos distintos: *Random Forest, Multilayer Perceptron, XGBoost, Naive Bayes* e *K-Nearest Neighbors*. Ele foi treinado utilizando um conjunto de dados históricos de preços de ações, utilizando essas informações para aprender a identificar padrões, foi adicionado também janela deslizante de 22 dias, 5 dias e 1 dia.

3.1.1 Votação

Para realizar a janela deslizante foi reutilizado o mesmo código do trabalho 1:

```
def expanding_window(model, X_train, y_train, window_size):

    pred = []
    actuals = []

    size = len(df.loc[df['year'] < 2022].index)
    size_max =  len(df)

    train_starts = range(size, size_max, window_size)
    i = 0

    for train_start in train_starts:

        X_train_window, X_test_window = X_train[:train_start], X_train[train_start : train_start + window_size]
        y_train_window, y_test_window = y_train[:train_start], y_train[train_start : train_start + window_size]

        print(f"Fold: {i}")

        model.fit(X_train_window, y_train_window)
        y_pred_window = model.predict(X_test_window)

        i+=1
        pred.extend(y_pred_window)
        actuals.extend(y_test_window)

    return np.array(pred), np.array(actuals)
```

Os modelos base são gerados por meio de uma função chamada ***get_models_classification()***:

```
def get_models_classification():
    """Cria lista com modelos base"""
    models = list()
    models.append(('RandomForest', RandomForestClassifier(random_state=22)))
    models.append(('MLP', MLPClassifier(random_state=22)))
    models.append(('XGBoost', XGBClassifier(random_state=22)))
    models.append(('NaiveBayes', GaussianNB()))
    models.append(('KNN', KNeighborsClassifier()))

    return models
```

```
# Cria os modelos base
models = get_models_classification()
```

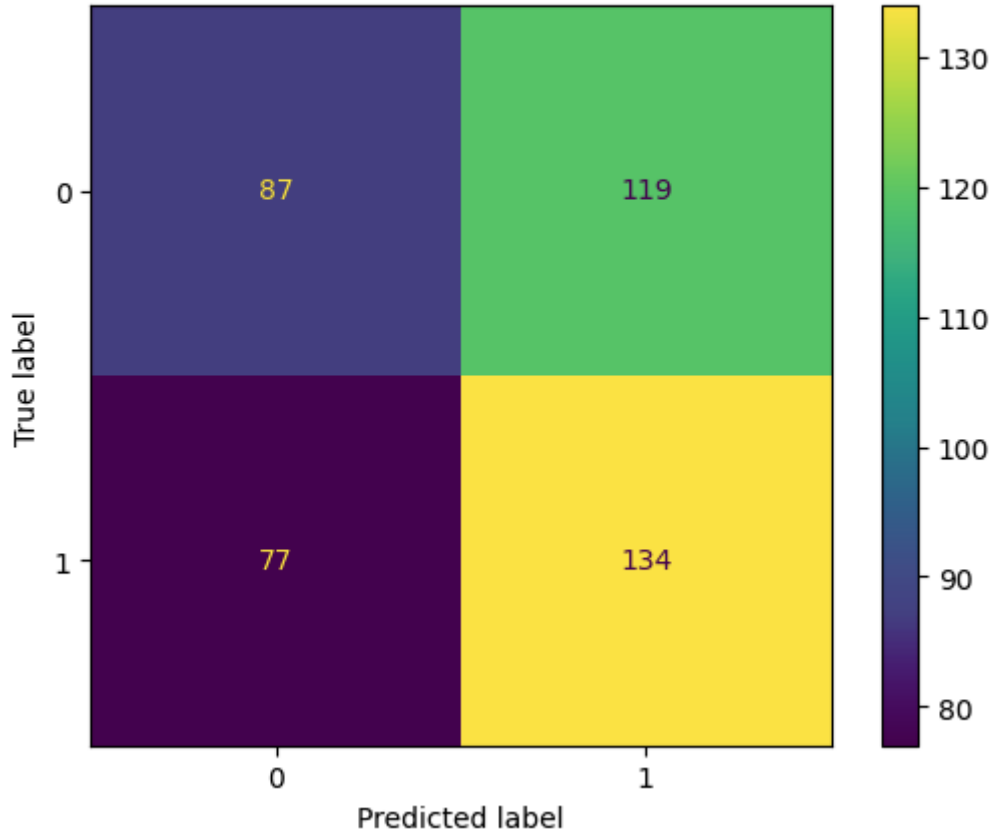
Durante o treinamento do modelo, o ensemble de votação foi criado utilizando o ***VotingClassifier()*** da biblioteca Sklearn. Por padrão, o modo de votação é voto majoritário, enquanto os modelos são fornecidos ao parâmetro estimators

```
from sklearn.ensemble import VotingClassifier
voting = VotingClassifier(estimators=models)
y_pred_window, y_test_window = expanding_window(voting, X, y, window_size=22)
evaluate_classification(y_test_window, y_pred_window)
```

3.1.1.1 Votação Janela de 22 dias

Avaliação do modelo:

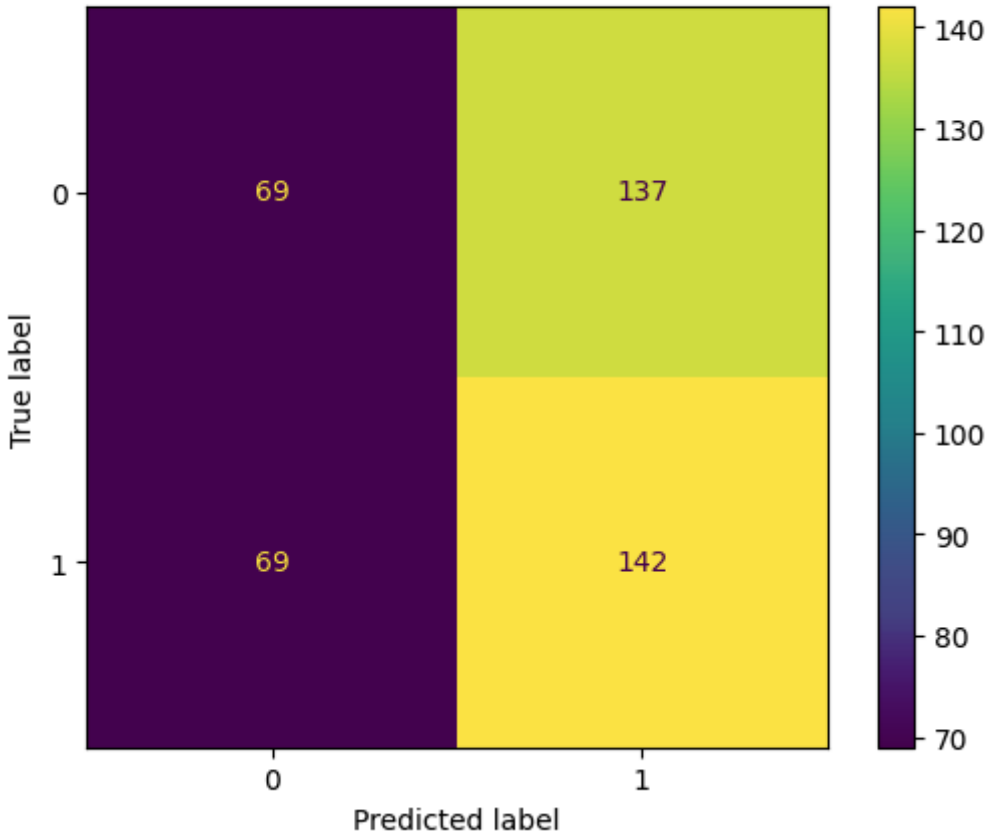
```
Accuracy: 0.5299760191846523
Precision: 0.5296442687747036
Recall: 0.6350710900473934
AUC: 0.5287005935673861
```



3.1.1.2 Votação Janela de 5 dias

Avaliação do modelo:

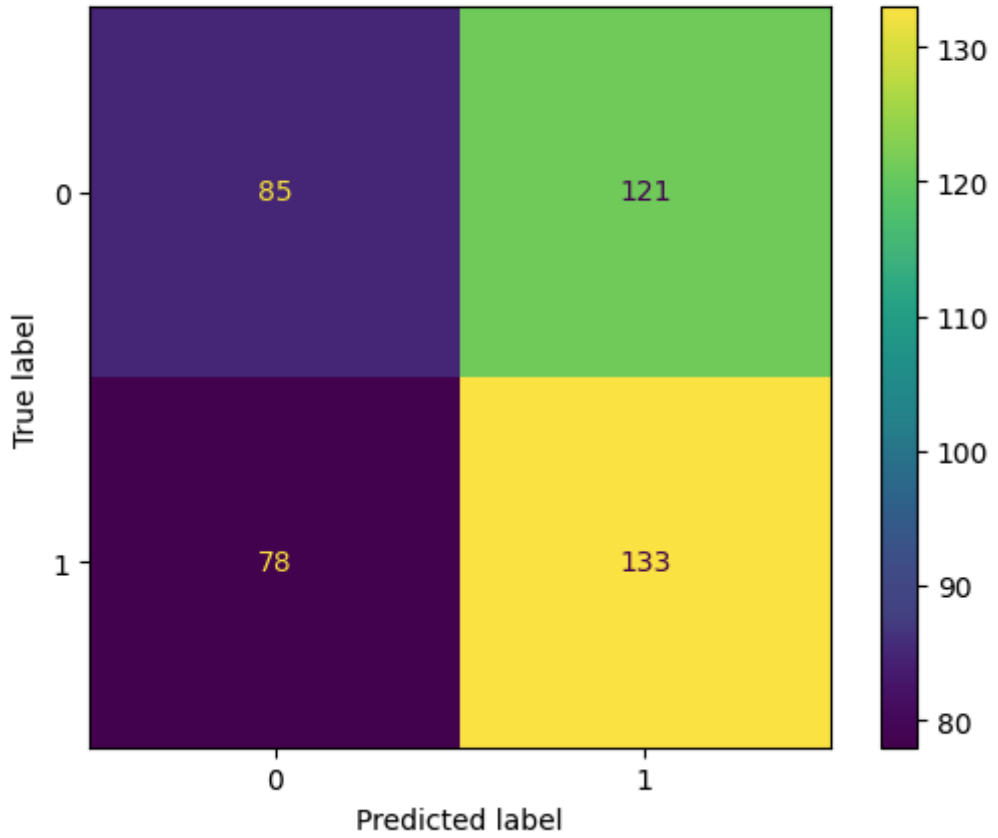
```
Accuracy: 0.5059952038369304
Precision: 0.5089605734767025
Recall: 0.6729857819905213
AUC: 0.5039686191506005
```



3.1.1.3 Votação Janela de 1 dias

Avaliação do modelo:

```
Accuracy: 0.5227817745803357
Precision: 0.5236220472440944
Recall: 0.6303317535545023
AUC: 0.5214765563889017
```



Em relação à abordagem do Holdout empregada no trabalho prático anterior, que obteve um resultado de AUC de 0,5247, a implementação do ensemble de votação de 22 dias revelou uma leve melhoria. A janela de 5 dias foi a pior, e a janela de 1 dia chegou perto da janela de 22 dias e do resultado anterior.

3.1.2 Média

Nesta abordagem, a média das saídas dos modelos de aprendizado de máquina é calculada, aplicando um limiar arbitrário para gerar a saída binária. O conjunto de dados foi dividido de forma semelhante à técnica de votação.

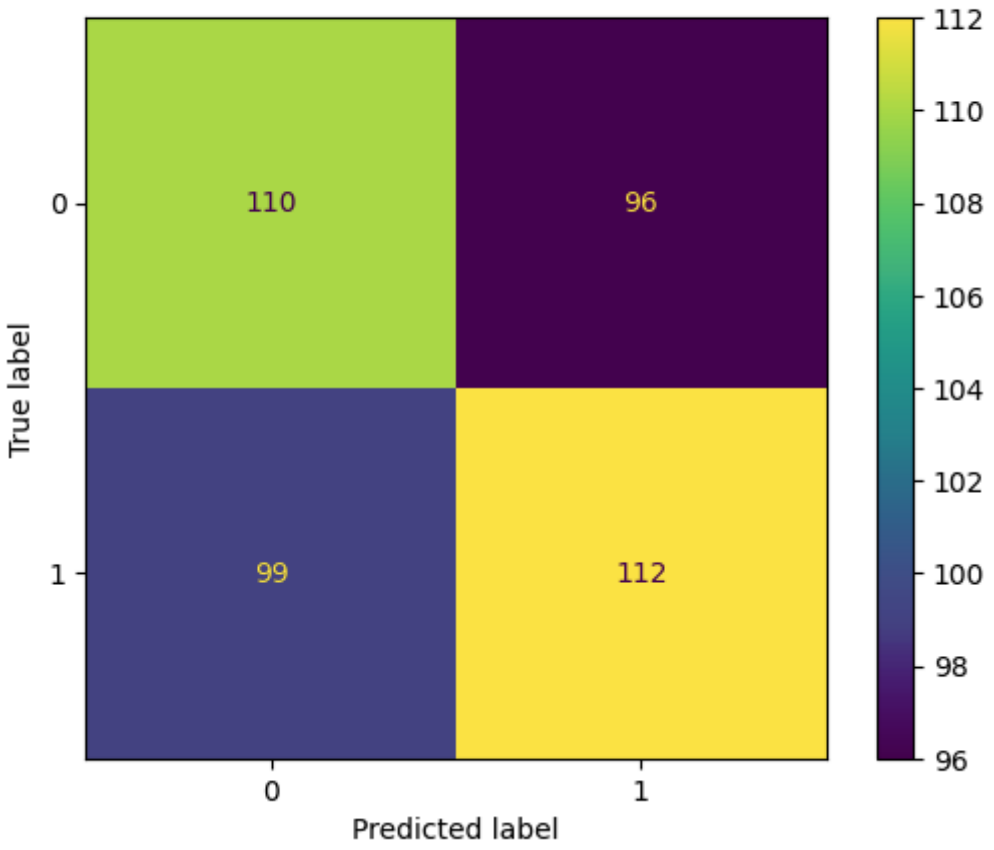
Novamente para realizar a janela deslizante foi reutilizado o mesmo código do trabalho 1. Durante o treinamento do modelo, o ensemble por média foi implementado utilizando o ***VotingClassifier()*** da biblioteca Sklearn, com os modelos sendo fornecidos ao parâmetro estimators. O modo de votação escolhido foi *'soft'*, pois essa configuração utiliza a média para prever as saídas.

```
from sklearn.ensemble import VotingClassifier
average = VotingClassifier(estimators=models, voting='soft')

y_pred_window, y_test_window = expanding_window(average, X, y, window_size=22)
evaluate_classification(y_test_window, y_pred_window)
```

3.1.2.1 Média Janela de 22 dias

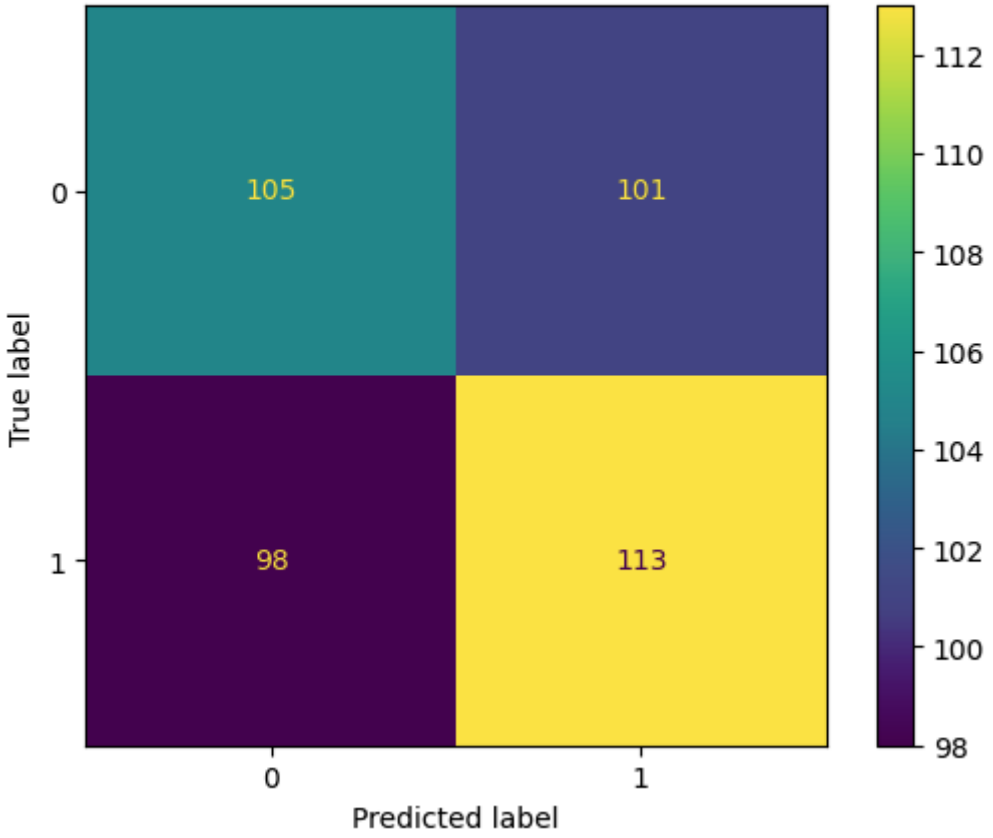
Avaliação do modelo:

```
Accuracy: 0.5323741007194245
Precision: 0.5384615384615384
Recall: 0.5308056872037915
AUC: 0.5323931348640317
```

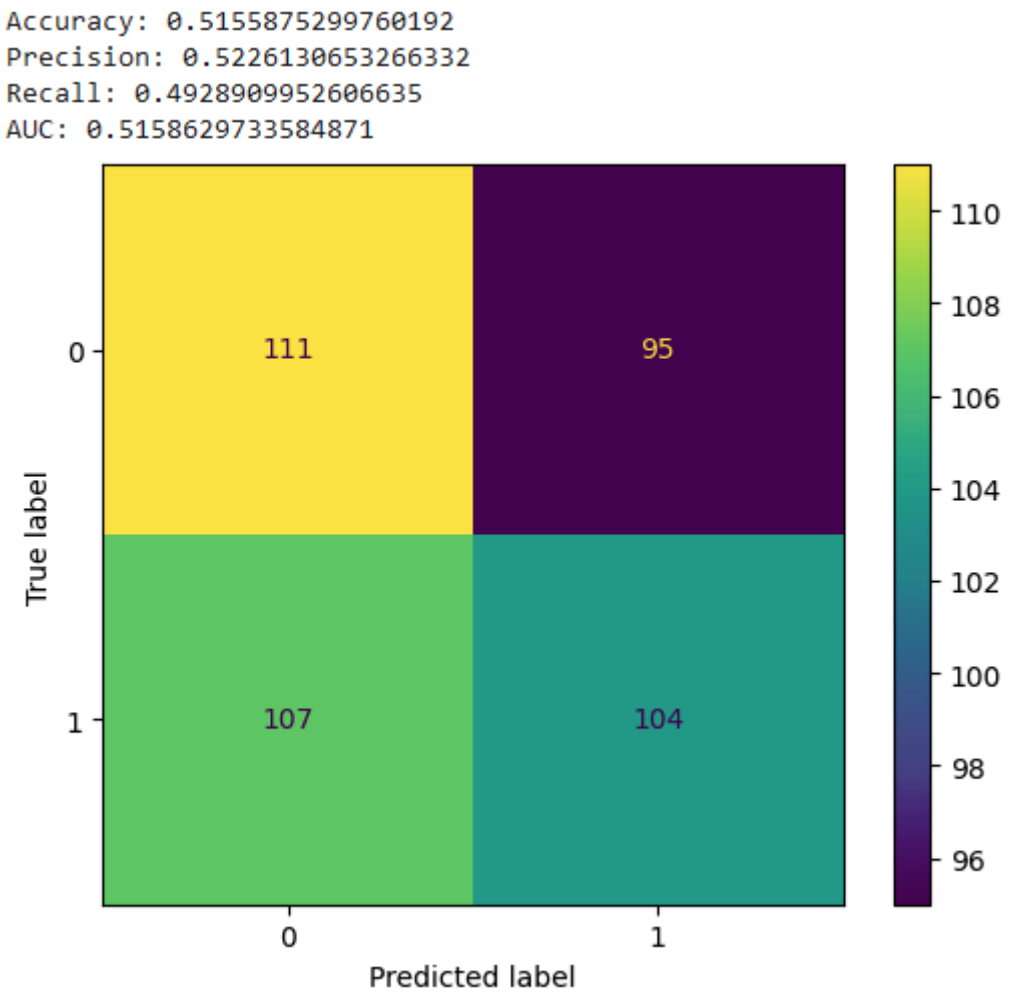


3.1.2.2 Média Janela de 5 dias

```
Accuracy: 0.5227817745803357
Precision: 0.5280373831775701
Recall: 0.5355450236966824
AUC: 0.52262688078038
```



3.1.2.3 Média Janela de 1 dias



O resultado da janela de 22 dias chegou perto do resultado do trabalho anterior, porém não o ultrapassou. Tanto as implementações das janelas de 5 dias e 1 dia, tiveram resultados piores, porém a janela de 5 dias teve um resultado semelhante a janela de 22 dias da votação.

3.1.3 Média Ponderada

Na média ponderada o código do trabalho anterior não foi utilizado, nesta implementação o código foi adaptado, criando uma função chamada **expanding_window_weighted()**. Como discutido na sala de aula, os pesos começaram valendo 1, e a cada iteração da janela deslizante os pesos foram recalculados. durante o treinamento do modelo, o ensemble por média ponderada foi

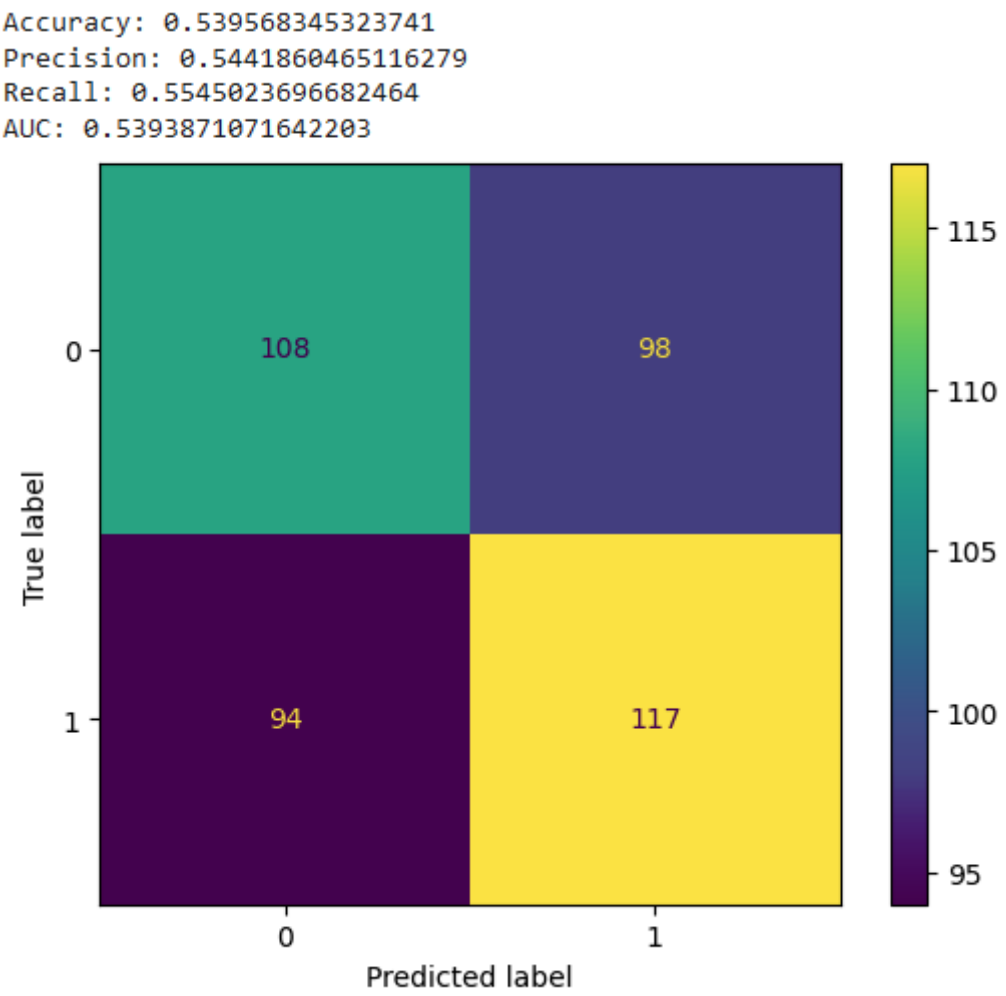
implementado utilizando o ***VotingClassifier()*** da biblioteca Sklearn, com os modelos sendo fornecidos ao parâmetro *estimators*. Optamos pelo modo de votação *'soft'*, o qual utiliza a média para prever as saídas, e os pesos foram atribuídos ao parâmetro *weights*:

```
def expanding_window_weighted(models, X_train, y_train, window_size):

    pred = []
    actuals = []

    size = len(df.loc[df['year'] < 2022].index)
    size_max =  len(df)

    train_starts = range(size, size_max, window_size)
    i = 0

    scores = [1,1,1,1,1]

    for train_start in train_starts:

        X_train_window, X_test_window = X_train[:train_start], X_train[train_start : train_start + window_size]
        y_train_window, y_test_window = y_train[:train_start], y_train[train_start : train_start + window_size]

        print(f"Fold: {i}\n {scores}\n {train_start} - {train_start + window_size}")
        weighted_average = VotingClassifier(estimators=models, voting='soft', weights=scores)

        weighted_average.fit(X_train_window, y_train_window)
        y_pred_window = weighted_average.predict(X_test_window)

        i+=1
        pred.extend(y_pred_window)
        actuals.extend(y_test_window)
        scores, pred_ev = evaluate_models(models, X_train_window, X_test_window, y_train_window, y_test_window)

    return np.array(pred), np.array(actuals)
```

.A função então é chamada e os resultados avaliados:

```
y_pred_window, y_test_window = expanding_window_weighted(models, X, y, window_size=22)
evaluate_classification(y_test_window, y_pred_window)
```

Avaliação do modelo:



Comparando os resultados entre o método de média, que inicialmente registrou um AUC de 0,53239, e o ensemble por média ponderada, que obteve um AUC um pouco superior, atingindo 0,53938, percebemos uma sutil melhoria.

3.1.4 Stacking

Foi empregado um modelo de aprendizado supervisionado para combinar as saídas dos modelos mencionados anteriormente. Utilizamos um classificador *Multilayer Perceptron* (MLP) nesta etapa para agregar as saídas dos modelos. Para criar o modelo de stacking, empregamos dados do período entre 2021 e 2022 para gerar as saídas dos modelos individuais. Posteriormente, unificamos esses dados com a saída esperada e aplicamos um classificador MLP.

```
# Calcula as pontuações AUC e previsões dos modelos com o conjunto de validação
scores, pred = evaluate_models(models, X_train, X_val, y_train, y_val)

# Gera DataFrame com as saídas dos modelos individuais
df_stacking = pd.DataFrame(data=np.transpose(pred), columns=['RandomForest', 'MLP', 'XGBoost', 'NaiveBayes', 'KNN'])
df_stacking['Output'] = pd.DataFrame(y_val).reset_index().drop(['index'], axis=1)

y_train_stacking = df_stacking['Output']
X_train_stacking = df_stacking.drop(['Output'], axis="columns")

# Cria a técnica de Stacking
stacking = MLPClassifier(random_state=22)

# Aplica Stacking para o conjunto de validação
stacking.fit(X_train_stacking, y_train_stacking)
```

O código para janela deslizante foi adaptado para o stacking, a cada iteração os 5 modelos são treinados, e suas predições armazenadas:

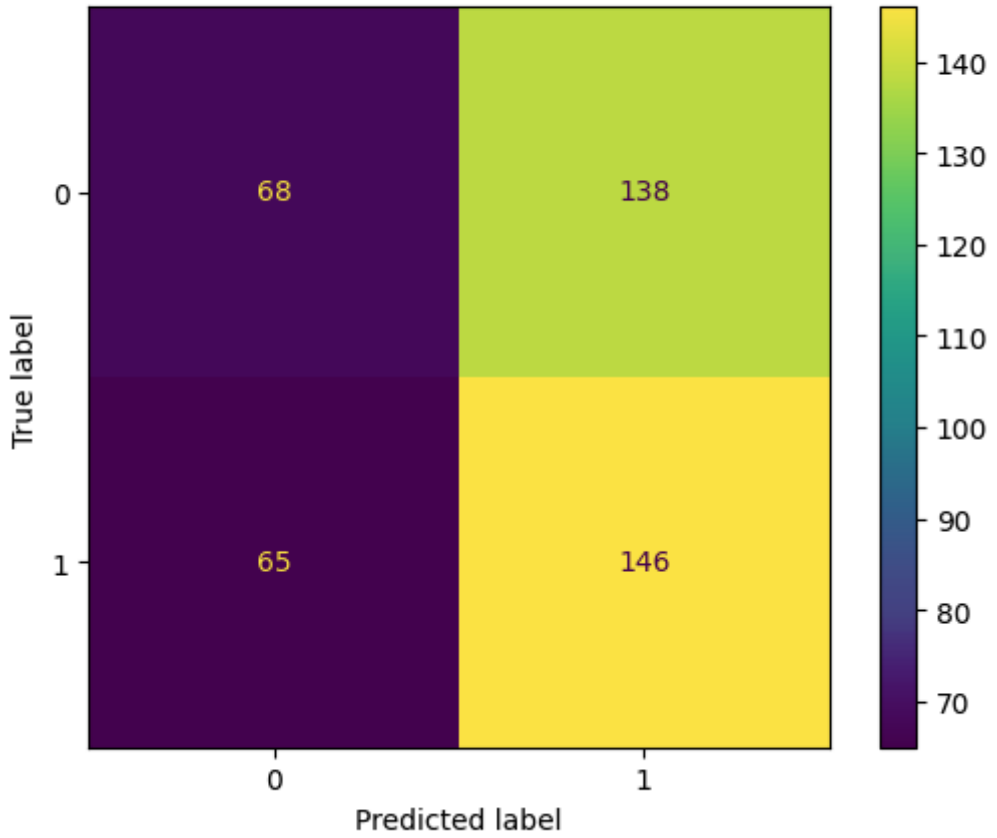
```
def expanding_window_stacking(models, X_train, y_train, window_size):

    pred = []
    actuals = []

    size = len(df.loc[df['year'] < 2022].index)
    size_max =  len(df)

    train_starts = range(size, size_max, window_size)
    i = 0

    for train_start in train_starts:

        X_train_window, X_test_window = X_train[:train_start], X_train[train_start : train_start + window_size]
        y_train_window, y_test_window = y_train[:train_start], y_train[train_start : train_start + window_size]

        print(f"Fold: {i}")

        scores, y_pred_window = evaluate_models(models, X_train_window, X_test_window, y_train_window, y_test_window)
        y_pred_window = np.transpose(y_pred_window)

        i+=1
        pred.extend(y_pred_window)
        actuals.extend(y_test_window)

    return np.array(pred), np.array(actuals)
```

Uma vez que o modelo foi treinado, o aplicamos o modelo previamente treinado do stacking para fazer as predições dos resultados dos 5 algoritmos.

```
y_pred_window, y_test_window = expanding_window_stacking(models, X, y, window_size=22)
df_stacking = pd.DataFrame(y_pred_window, columns=['RandomForest', 'MLP', 'XGBoost', 'NaiveBayes', 'KNN'])

y_pred_stacking = stacking.predict(df_stacking)

evaluate_classification(y_test, y_pred_stacking)
```

Avaliação do modelo:

```
Accuracy: 0.513189448441247
Precision: 0.5140845070422535
Recall: 0.6919431279620853
AUC: 0.511020107670363
```



Comparando os resultados entre as abordagens dos trabalhos anteriores, o Stacking apresentou uma melhora, o resultado foi de AUC de 0.50587 para 0.51102.

3.2 Regressão

3.2.1 Média

Seguiu-se os mesmo passos do modelo de classificação, as únicas diferenças foram que para a tarefa de regressão, alguns ajustes tiveram que ser feitos. Os algoritmos de classificação foram convertidos para os de regressão, com exceção do *Naive Bayes*, que foi deixado de fora pois não funciona em tarefas de regressão. Os modelos base são gerados por meio de uma função chamada ***get_models_regression()***:

```python
def get_models_regression():
  """Cria lista com modelos base"""
  models = list()
  models.append(('RandomForest', RandomForestRegressor(random_state=22)))
  models.append(('MLP', MLPRegressor(random_state=22)))
  models.append(('XGBoost', XGBRegressor(random_state=22)))
  #models.append(('NaiveBayes', GaussianNB()))
  models.append(('KNN', KNeighborsRegressor()))

  return models
```

```python
# Cria os modelos base
models = get_models_regression()
```

Treinamento do modelo:

```python
# Cria a técnica de Ensemble por Média
average = VotingRegressor(estimators=models)

y_pred_window, y_test_window = expanding_window(average, X_train_full, y, window_size=22)
evaluate_regressor(y_test_window, y_pred_window)
```

Avaliação do modelo 22 dias:

```
RMSE =  1.382961104344743
MAE =  1.0745342775201214
```

Avaliação do modelo 5 dias:

```
RMSE =  1.3783231074950928
MAE =  1.0710240237373705
```

Ao analisarmos a performance da técnica de Média de 22 dias e 5 dias em relação a outras abordagens utilizadas previamente, é perceptível que os valores de RMSE e MAE se mantêm em um patamar similar. Apresentando apenas uma pequena melhora, se comparado a trabalhos anteriores.

3.2.2 Média Ponderada

O código para implementar a Média Ponderada é o mesmo do algoritmo de classificação, a diferença é que agora o modelo utiliza o ***VotingRegressor()***, e os pesos não são mais o AUC, e sim o inverso da métrica MAE (Mean Absolute Error).

Treinamento do Modelo:

```
def expanding_window_weighted_regression(models, X_train, y_train, window_size):

    pred = []
    actuals = []

    size = len(df.loc[df['year'] < 2022].index)
    size_max =  len(df)

    train_starts = range(size, size_max, window_size)
    i = 0

    scores = [1,1,1,1]

    for train_start in train_starts:

        X_train_window, X_test_window = X_train[:train_start], X_train[train_start : train_start + window_size]
        y_train_window, y_test_window = y_train[:train_start], y_train[train_start : train_start + window_size]

        print(f"Fold: {i}\n")
        weighted_average = VotingRegressor(estimators=models, weights=scores)

        weighted_average.fit(X_train_window, y_train_window)
        y_pred_window = weighted_average.predict(X_test_window)

        i+=1
        pred.extend(y_pred_window)
        actuals.extend(y_test_window)
        scores, pred_ev = evaluate_models(models, X_train_window, X_test_window, y_train_window, y_test_window)

    return np.array(pred), np.array(actuals)
```

```
y_pred_window, y_test_window = expanding_window_weighted_regression(models, X, y, window_size=22)
evaluate_regressor(y_test_window, y_pred_window)
```

Avaliação do modelo:

```
RMSE =  1.430355470019368
MAE =  1.1026778319941584
```

A abordagem da Média Ponderada oferece uma variação ao ponderar as previsões dos modelos com base em diferentes critérios. Embora essa técnica ainda produza resultados competitivos, se observou valores ligeiramente superiores de RMSE e MAE em comparação com a Média. A inclusão de pesos ajustáveis para cada modelo na composição da previsão agregada pode introduzir uma camada adicional de complexidade e, em alguns casos, pode resultar em um desempenho levemente inferior em comparação com a simplicidade direta da técnica de Média.

3.2.3 Stacking

O modelo de regressão adotou uma abordagem semelhante àquela utilizada no modelo de aprendizado supervisionado, visando combinar as saídas dos modelos anteriores no contexto da modelagem de classificação.

```
# Calcula as pontuações AUC e previsões dos modelos com o conjunto de validação
scores, pred = evaluate_models_regression(models, X_train, X_val, y_train, y_val)

# Gera DataFrame com as saídas dos modelos individuais
df_stacking = pd.DataFrame(data=np.transpose(pred), columns=['RandomForest', 'MLP', 'XGBoost', 'KNN'])
df_stacking['Output'] = pd.DataFrame(y_val).reset_index().drop(['index'], axis=1)

y_train_stacking = df_stacking['Output']
X_train_stacking = df_stacking.drop(['Output'], axis="columns")
y = df['Output']
X = df.drop(['Output', 'Return', 'Date'], axis="columns")

# Cria a técnica de Stacking
stacking = MLPRegressor(random_state=22)

# Aplica Stacking para o conjunto de validação
stacking.fit(X_train_stacking, y_train_stacking)
```

```
y_pred_window, y_test_window = expanding_window_stacking(models, X, y, window_size=22)
df_stacking = pd.DataFrame(y_pred_window, columns=['RandomForest', 'MLP', 'XGBoost', 'NaiveBayes', 'KNN'])

y_pred_stacking = stacking.predict(df_stacking)

evaluate_regressor(y_test, y_pred_stacking)
```

Avaliação do modelo:

```
RMSE =  1.3246313711992275
MAE =  1.019878099138729
```

A técnica de Stacking demonstra os menores valores de RMSE e MAE em comparação com outras abordagens.

4. Resultados Financeiros

4.1 Classificação

4.1.1 Votação

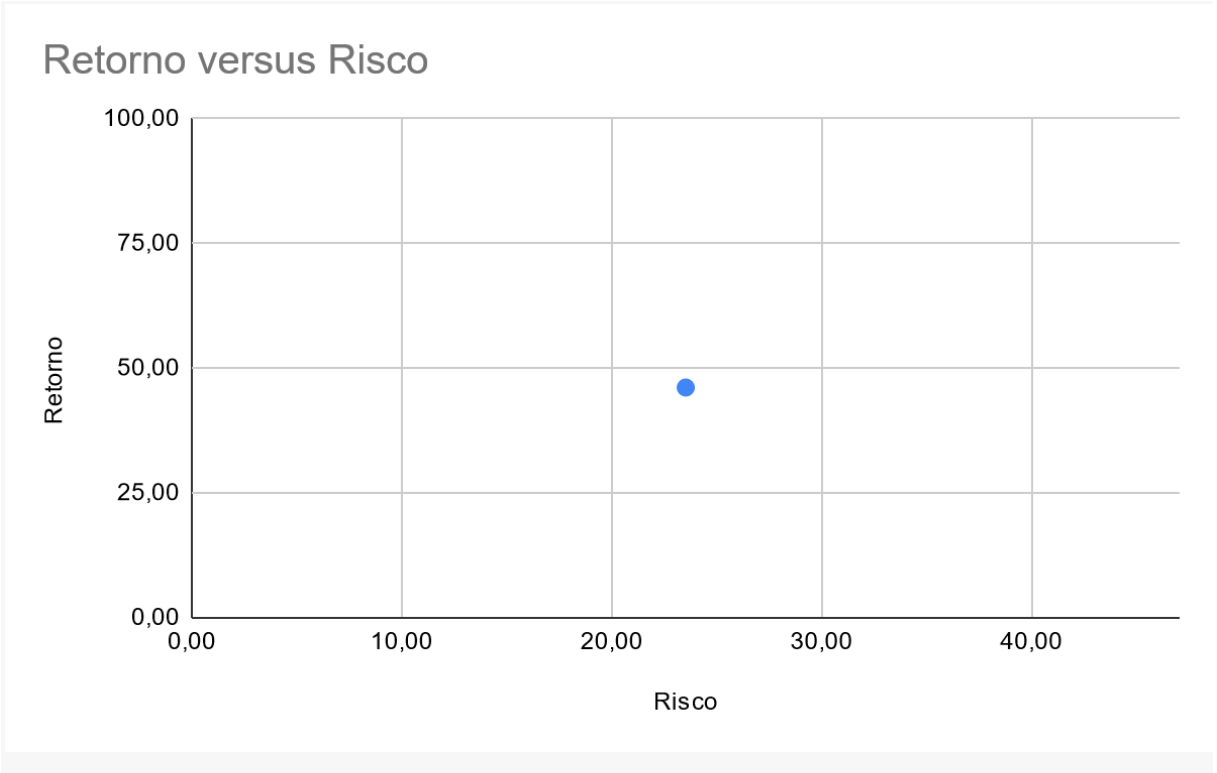
Tabela Risco e Retorno

Dias	Risco	Retorno
22 dias	23,53	46,15
5 dias	17,38	19,08
1 dia	17,38	19,08

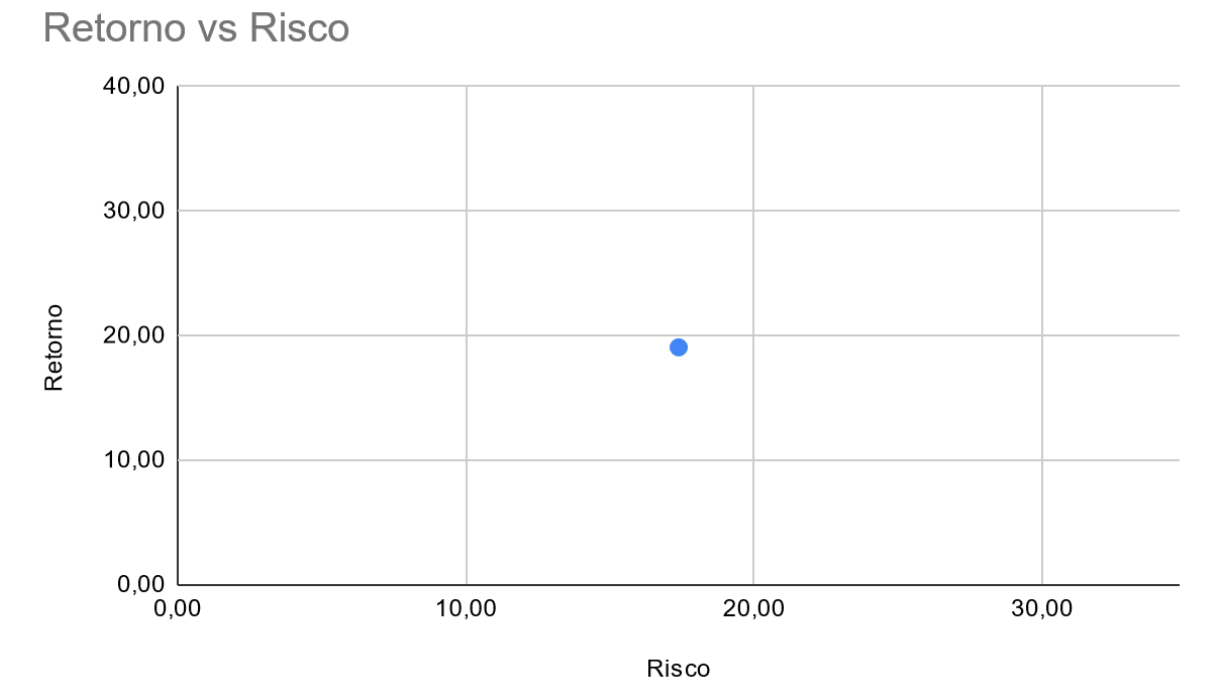
Plot do retorno acumulado no período de testes

Plot do retorno vs. risco no período de testes

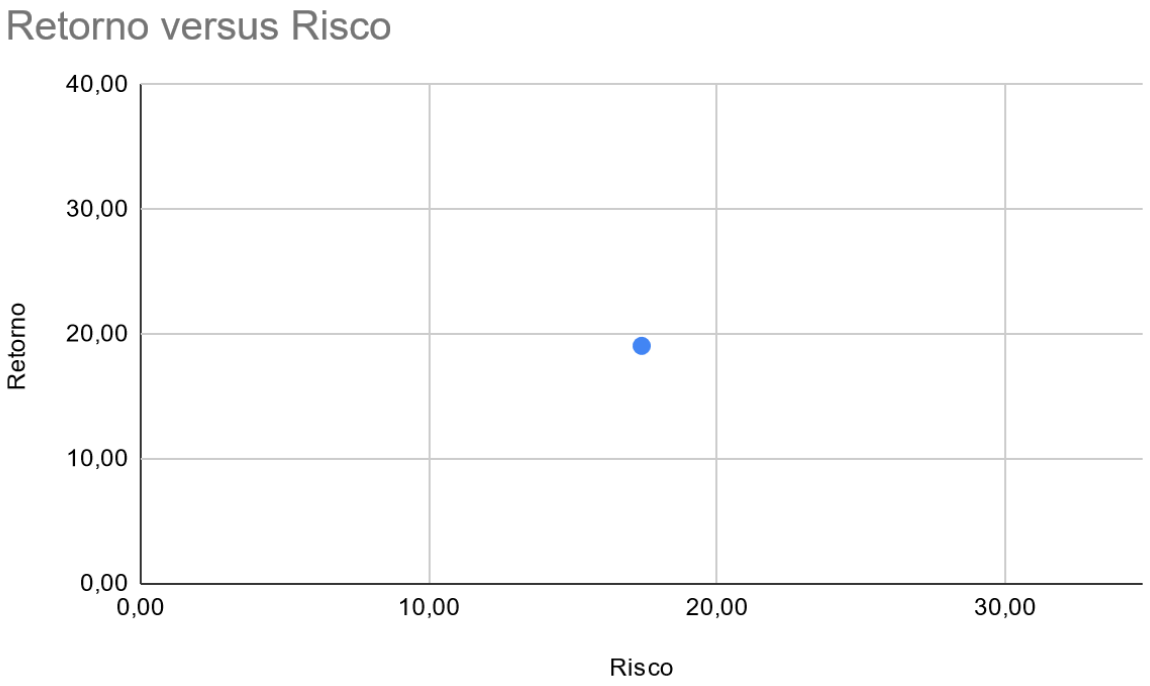
22 Dias:


Retorno versus Risco
100,00
75,00
50,00
25,00
0,00
Retorno
0,00
10,00
20,00
30,00
40,00
Risco

5 dias:


Retorno vs Risco
40,00
30,00
20,00
10,00
0,00
Retorno
0,00
10,00
20,00
30,00
Risco

1 dia:



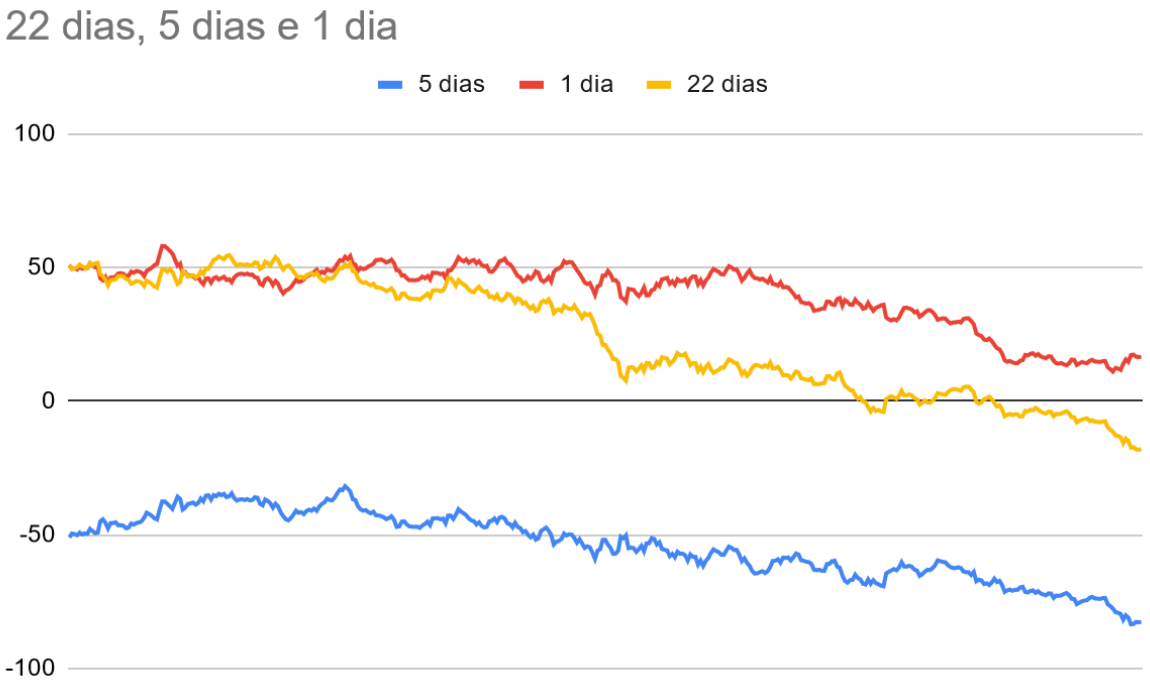
A votação com janela deslizante de 22 dias e 5 dias apresentaram resultados bem semelhantes, já a janela de 1 dia apresentou os piores resultados.

4.1.2 Média

Tabela Risco e Retorno

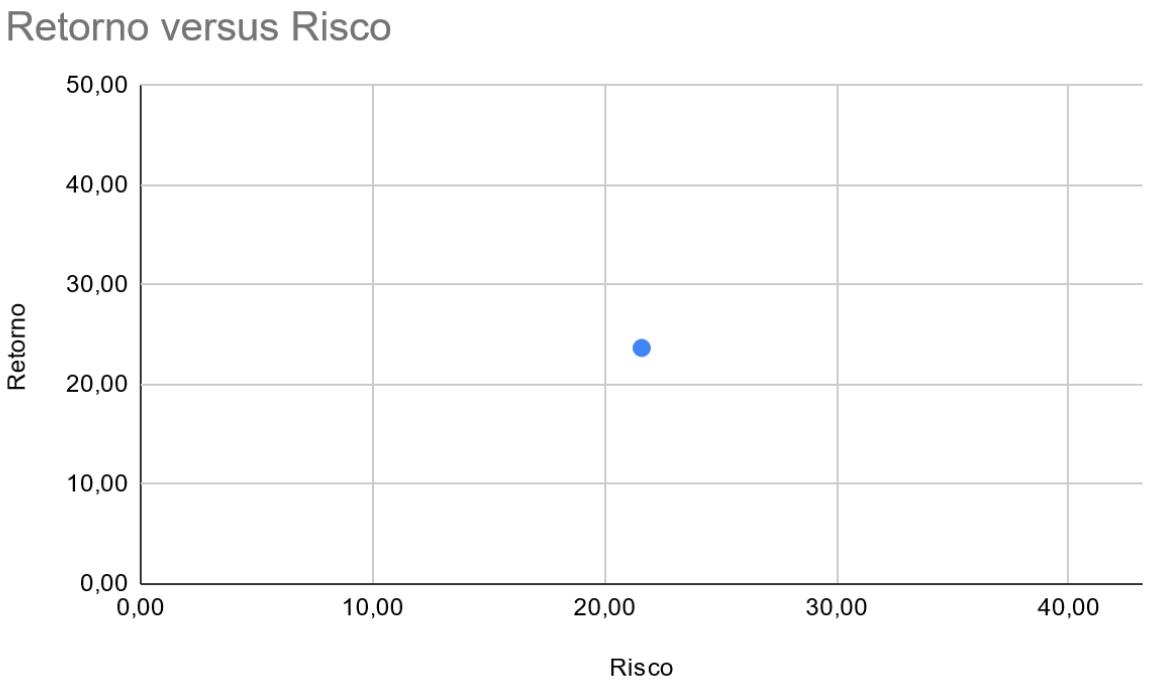
Dias	**Risco**	**Retorno**
22 dias	21,59	23,68
5 dias	12,64	-53,87
1 dia	12,12	40,12

Plot do retorno acumulado no período de testes

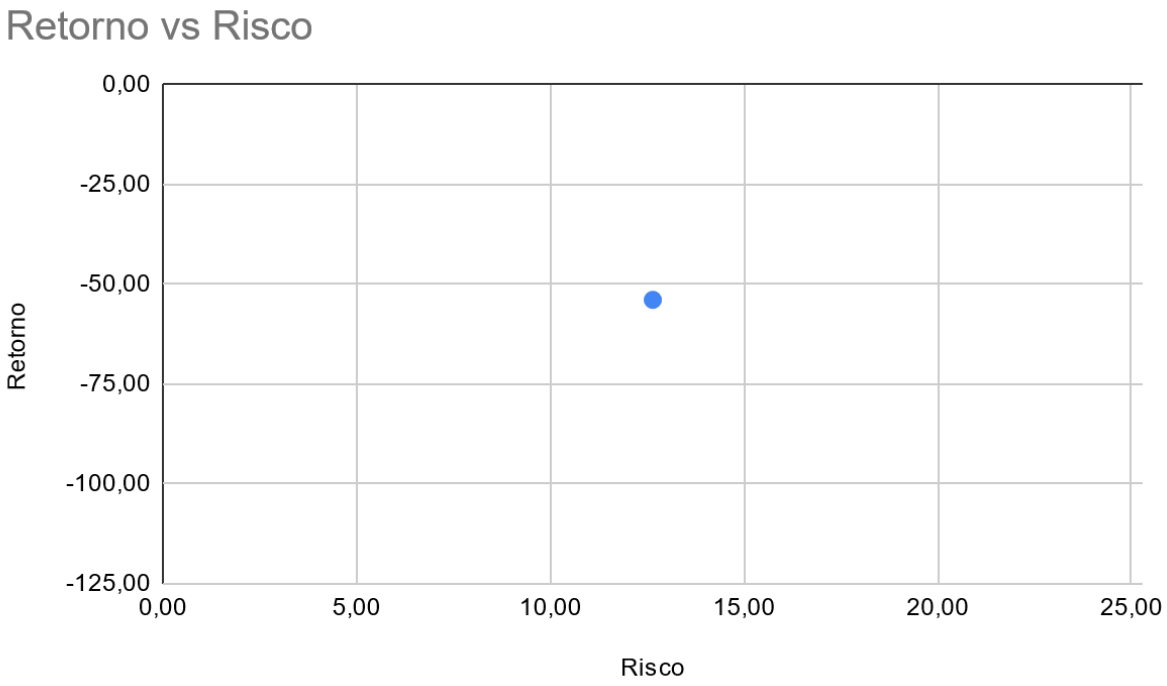


Plot do retorno vs. risco no período de testes

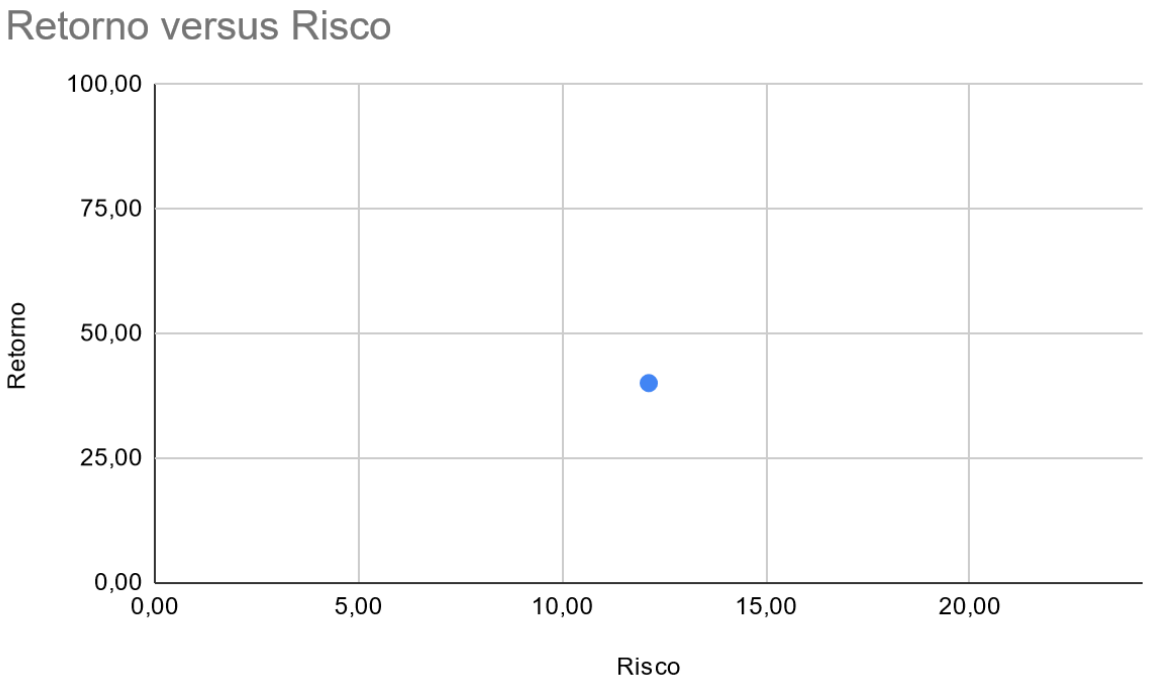
20 Dias:



5 dias:



1 dia:



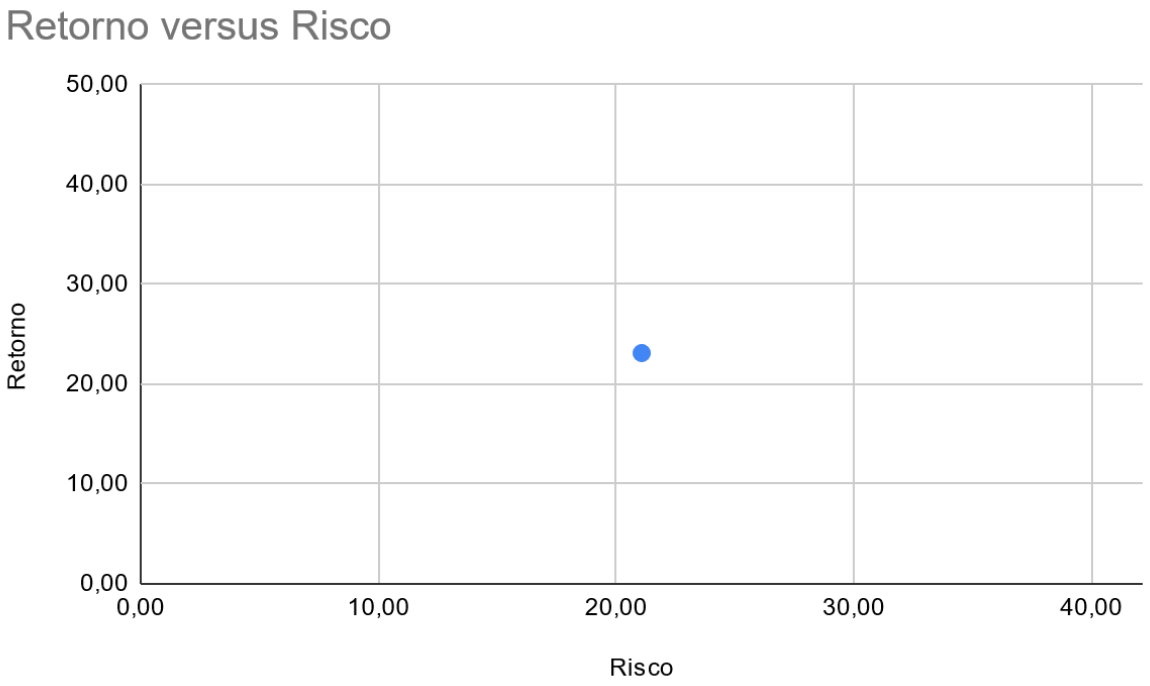
A média com janela deslizante de 22 dias e 5 dias apresentaram resultados bons em boa parte do tempo, a de 5 dias chegou até a chegar a 60, já a janela de 1 dia apresentou resultado negativos

4.1.3 Média Ponderada

1. Retorno: 23,12
2. Risco: 21,07

Plot do retorno acumulado no período de testes

3. Plot do retorno vs. risco no período de testes

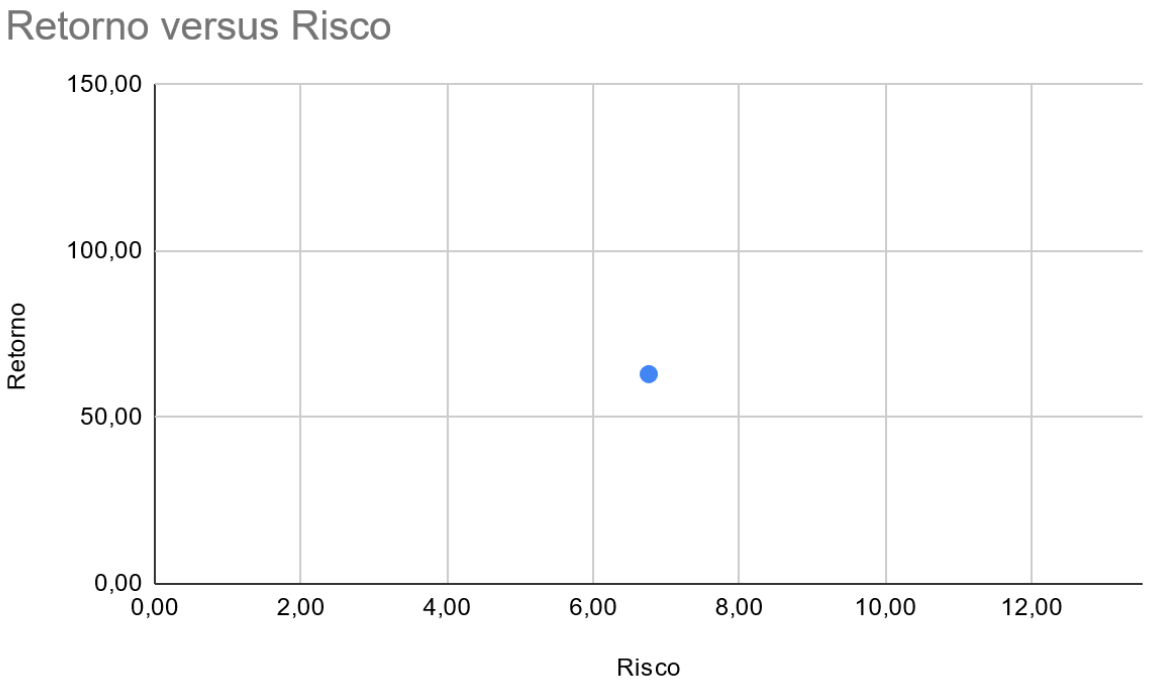


A Média Ponderada apresentou um um retorno médio de 23,12, sua máxima foi de quase de 60, com um alto risco de 21,07.

4.1.4 Stacking

1. Retorno: 63,022
2. Risco: 6,758
3. Plot do retorno acumulado no período de testes

4. Plot do retorno vs. risco no período de testes



O Stacking apresentou um dos melhores resultados com um um retorno médio 63,02, sua máxima chegou a quase 80, com um risco de 6,7.

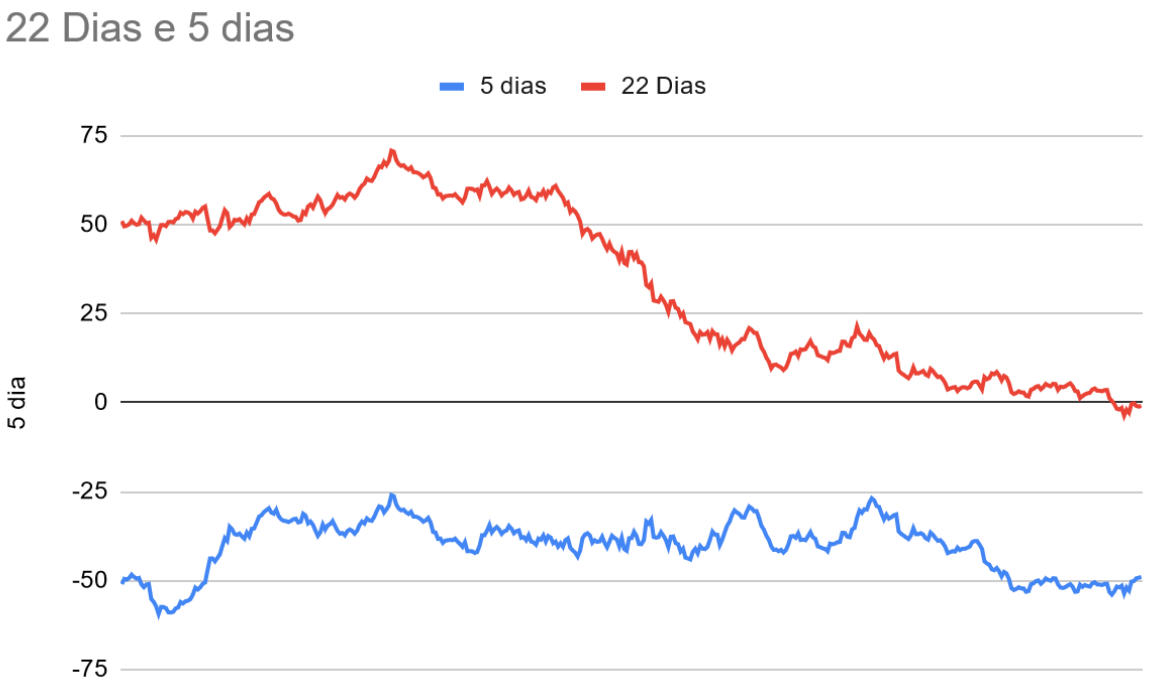
4.2 Regressão

4.2.1 Média

Tabela retorno e risco

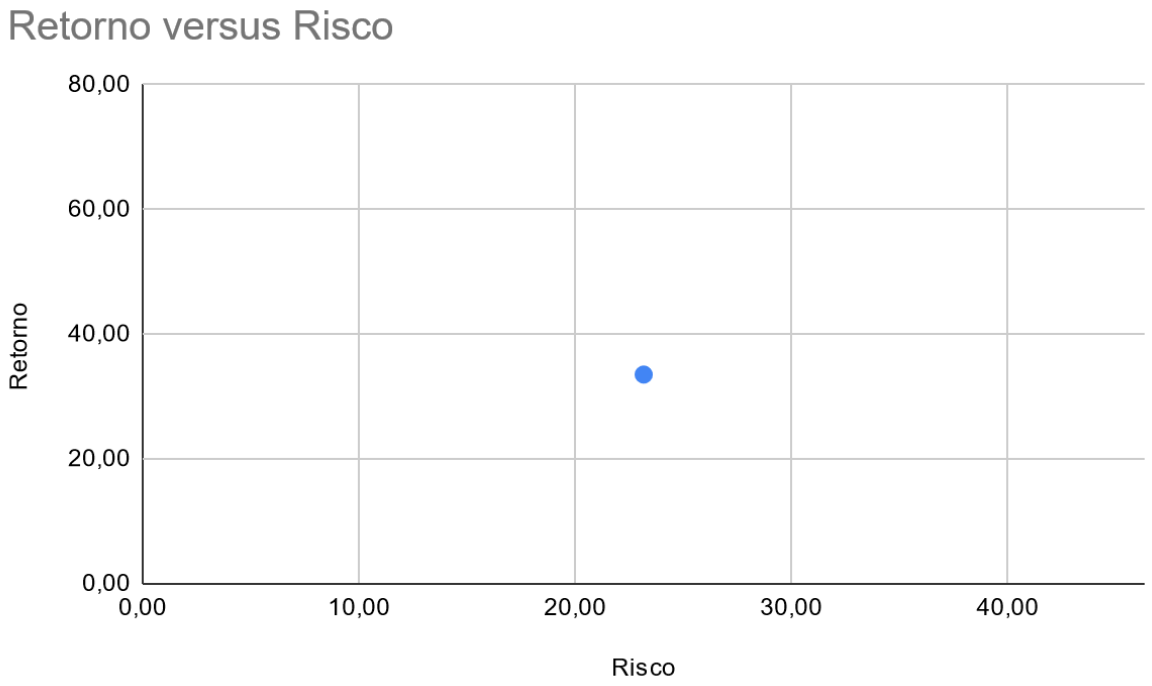
Dias	Risco	Retorno
22 dias	23,16	33,55
5 dias	7,68	-40,19

Plot do retorno acumulado no período de testes

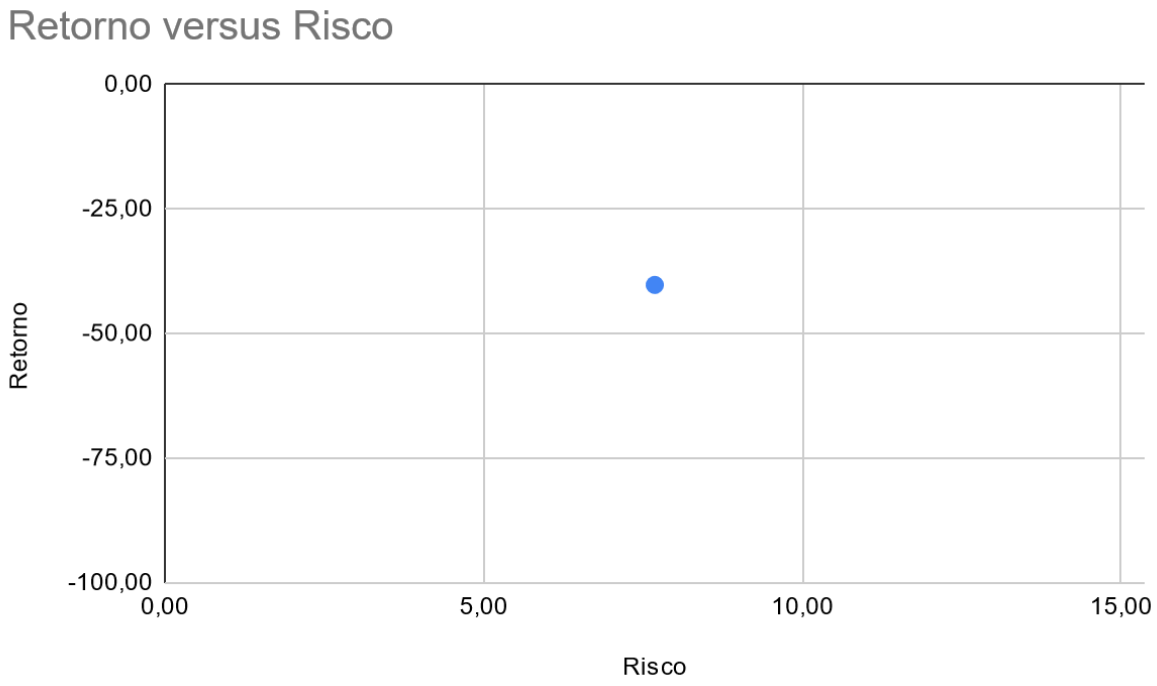


Plot do retorno vs. risco no período de testes

22 dias:



5 dias:

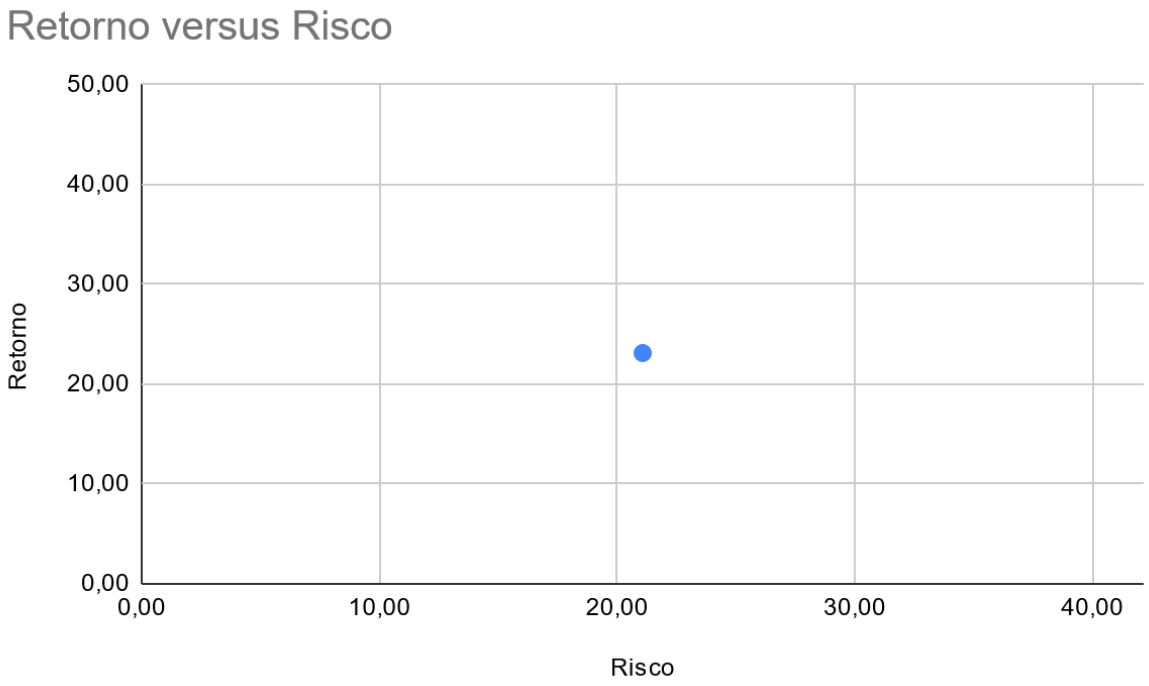


A Média de 22 dias apresentou um um retorno médio de 33,55, sendo o melhor resultado se comparado com a média de 5 dias.

4.2.2 Média Ponderada

1. Retorno: 23,12
2. Risco: 21,07
3. Plot do retorno acumulado no período de testes

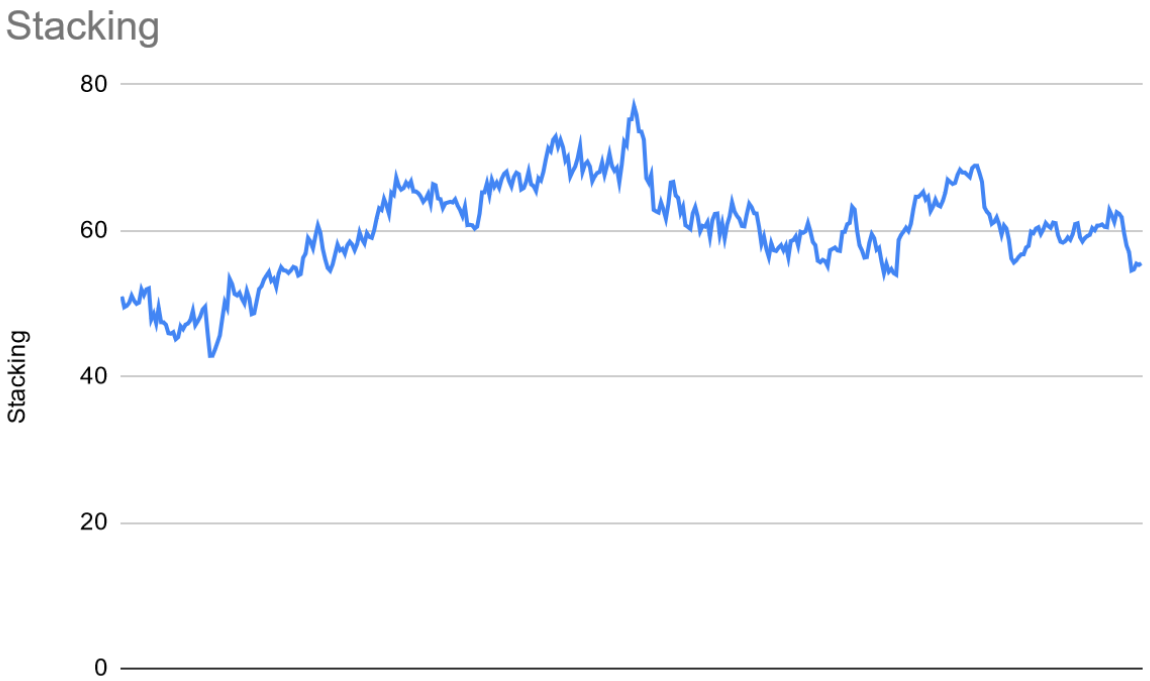
4. Plot do retorno vs. risco no período de testes



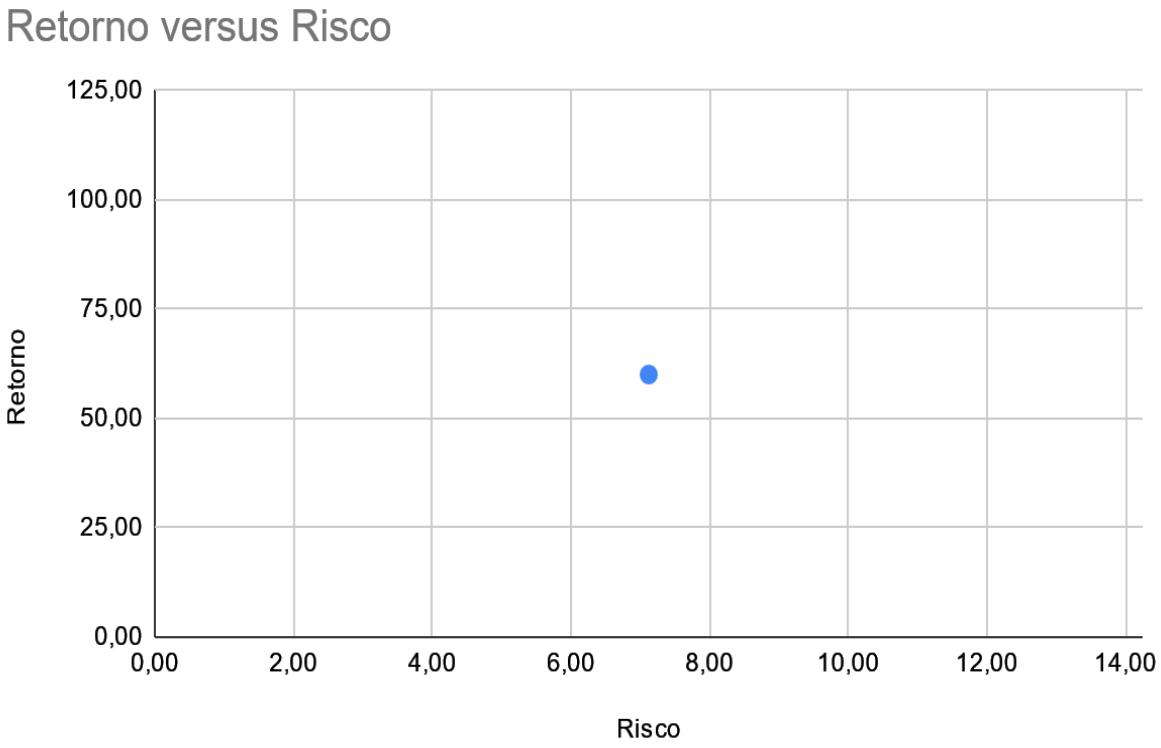
A Média Ponderada começou bem e apresentou um um retorno médio de 23,12, sua máxima foi mais de quase 60, com um alto risco de 21,07.

4.2.2 Stacking

1. Retorno: 60,02
2. Risco: 7,35
3. Plot do retorno acumulado no período de testes



4. Plot do retorno vs. risco no período de testes



O Stacking apresentou um retorno de 60,02, semelhante aos resultados anteriores do stacking na classificação, sua máxima foi mais de quase 80, com um risco de 7,35.