Utilizando o Apache Spark para estudar consumo de energia elétrica

Fabio Luiz Silva Nogueira[1], **Lyncoln Sousa de Oliveira**[1]

[1]Programa de Engenharia de Sistemas e Computação (PESC)
Universidade Federal do Rio de Janeiro (UFRJ)
RJ – Brazil

{fabionogueira,oliveiral}@cos.ufrj.br

Resumo. *Nesse trabalho foi utilizado o framework Apache Spark para estudar o consumo de energia elétrica. Para isso, foi utilizado um banco de dados da PJM Interconnection LLC (PJM) que é uma organização de transmissão regional (RTO) nos Estados Unidos, que opera um sistema de transmissão elétrica em vários estados do país. Os dados são de consumo elétrico a cada hora em um intervalo de mais de 10 anos nesses estados. O objetivo é utilizar esses dados como base para criar ferramentas de analise do consumo de energia elétrica de cada residência,escalonando consideravelmente a quantidade de dados. Foi utilizado a bibilioteca prophet do facebook para estudar o consumo elétrico nesses estados.*

1. Introdução

Com o crescimento populacional e da tecnologia o consumo de energia elétrica vem crescendo em escala mundial. Para lidar com esse crescimento, é importante criar mecanismos de processamento de *Big data*, segundos os autores Mauro at al. (2016), "*Big Data* é o ativo da informação caracterizado por um alto volume, velocidade e variedade para exigir tecnologia específica e Métodos analíticos para sua transformação em valor."

Em um cenário onde há bilhões de possíveis residências que podem gerar informações sobre o consumo elétrico, as características de alto volume (quantidade de dados), e velocidade (velocidade de transmissão dos dados) podem categorizar o problema de estudar a tendência de consumo de energia elétrica como uma aplicação de *Big Data*. A transformação em valor pode ser dada pela aplicação de modelos estatísticos de séries temporais com o objetivo de inferir a demanda por energia elétrica, que é um procedimento comum que as provedoras desse bem material adotam para que possam articular planos e suprir a demanda em algum tempo no futuro.

Para estudar essa tendência, nesse trabalho será utilizado o banco de dados fornecido pela *PJM Interconnection LLC* (PJM) [Mulla 2019], que é uma organização de transmissão regional de energia nos Estados Unidos. É parte da rede de interconexão oriental que opera um sistema de transmissão elétrica que atende toda ou parte de Delaware, Illinois, Indiana, Kentucky, Maryland, Michigan, Nova Jersey, Carolina do Norte, Ohio, Pensilvânia, Tennessee, Virgínia, Virgínia Ocidental e o distrito de Columbia. Está disponível nesse banco de dados a leitura total de energia consumida nesses estados medidas por megawatts a cada hora no intervalo de até 10 anos. Nessa base de dados há uma média de 100 mil observações por estado, totalizando 44 *Megabytes* de armazenamento para todos eles.

Apesar do total de dados ser consideravelmente pequeno, o problema pode ser escalonado para ao invés de observar as medidas por estado, observar por exemplo, as medidas de consumo de energia elétrica provenientes de cada residência de um país, onde pode ser interessante calcular a média do consumo elétrico nacional.

Uma ferramenta comumente utilizada para processamento de dados em larga escala é o Apache Spark, que é um *framework* que oferece uma interface de programação para o desenvolvimento de rotinas que possam ser processadas de maneira paralela em *cluster* com mecanismos de tolerância a falha [Zaharia et al. 2010].

Para análise de série temporal, será utilizado o Prophet, que é um software de código aberto lançado pela equipe principal de desenvolvimento de *Data Science* do Facebook. É um procedimento para prever dados de série temporal com base em um modelo aditivo em que tendências não lineares são ajustadas com sazonalidade anual, semanal e diária, além de também poder levar em consideração os efeitos de feriados [Taylor and Letham 2017].

Para a visualização dos resultados, será gerado um painel gráfico através do Google Data Studio, que fará conexão com a aplicação, de forma que com seja atualizado em tempo real.

2. Objetivos

2.1. Geral

Utilizar a base de dados da PJM para criar mecanismo de processamento de dados distribuído utilizando o Apache Spark, realizando a média de consumo diário de energia elétrica desses estados. Criar um sistema robusto o suficiente para escalar de maneira em que ao invés de observar os estados em um país, possa observar cada casa, aumentando consideravelmente a quantidade de dados de entrada.

2.2. Específicos

- Ajustar um modelo de séries temporais utilizando a biblioteca Prophet do Facebook.
- Separar os dados de entrada para o modelo em treino e teste para calcular medida de qualidade do ajuste.
- Utilizar o modelo criado para realizar predição de 28 dias no futuro.
- Criar um dashboard para ilustrar os dados de entrada e saída do modelo.

3. Materiais e Métodos

3.1. Apache Spark

O Apache Spark é um framework de processamento de dados em grande escala, que fornece APIs de alto nível em algumas linguagens de programação como Python, R e Java. Ele possui um mecanismo otimizado que suporta execução de grafos, um conjunto de APIs como o Spark SQL para processar dados estruturados, MLlib para aprendizado de máquina, GraphX para processamento de grafo e *Structured Streaming* para computação incremental e processamento em tempo-real [Spark 2021].

O Spark auxilia os desenvolvedores a criarem aplicativos que façam uso de processamento paralelo, para isso, os desenvolvedores escrevem um programa que implementa

um fluxo de controle de alto nível que inicia várias operações em paralelo. O Spark consegue alcançar esse objetivo principalmente através de duas abstrações de processamento paralelo: conjuntos de dados distribuídos resilientes (RDD - *resilient distributed datasets*) e operações paralelas em nesses conjuntos de dados [Zaharia et al. 2010].

Um conjunto de dados distribuídos resilientes (RDD) é uma coleção de objetos que são somente de leitura, particionados em um conjunto de máquinas que podem ser montados novamente caso a partição seja perdida. Seus elementos não precisam estar persistidos em um armazenamento físico, pois cada RDD possui informação suficiente para computar o ponto inicial do RDD a partir dos dados em um armazenamento confiável, fazendo com que RDDs possam ser sempre reconstruídos se um nó falhar.

Há dois tipos de operações que podem ser utilizadas pelo Spark nos RDDs, essas são: Transformação, que cria um novo RDD utilizando algum tipo de processamento em um outro já existente, são exemplos de transformação as funções: map, filter, distinct e etc; E as operações chamadas de ações, que podem ser utilizadas em conjunto com as transformações, tem como objetivo realizar computação nos RDDs para obter um resultado, são exemplos de ações as funções: reduce, foreach, count e etc. A figura 1 ilustra essas operações.

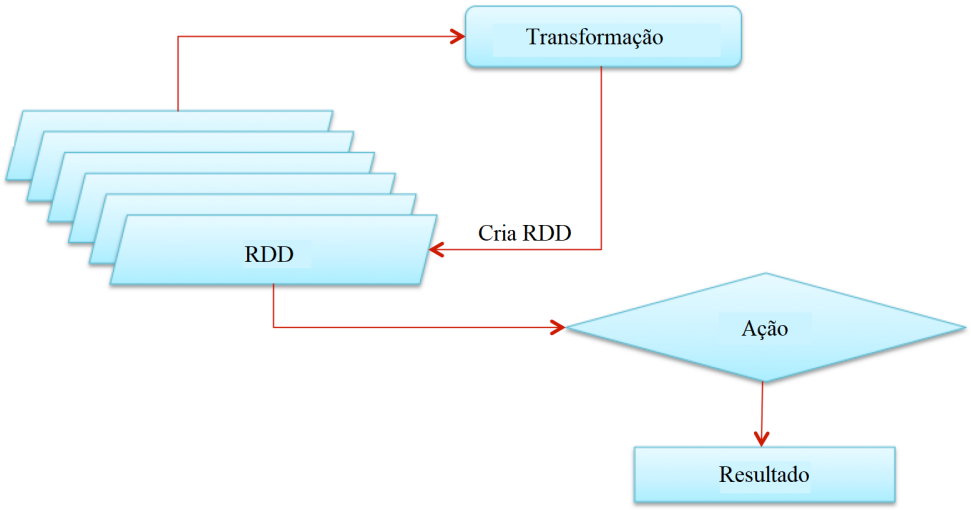

Figura 1. Representação de operações em RDDs
Fonte: Autores (2021)

O Spark SQL introduziu uma nova interface na versão 1.6 chamada *DataSet*, que representa uma coleção de dados distribuídos com os benefícios de um RDD e com os benefícios de execução otimizada do Spark SQL. O Spark também possui *DataFrame*, que são *datasets* organizados em colunas nomeadas, que é equivalente a uma tabela em um banco de dados, com a otimização do Spark [Spark 2021].

O Spark também possui um mecanismo para processamento de dados em tempo real chamado *Spark Structured Streaming*, que foi construído no Spark SQL, sendo escalável e tolerante a falhas. O desenvolvedor pode criar aplicações da mesma forma que seria para processamento em lotes, que o Spark SQL irá cuidar de rodar o código incrementalmente e continuamente, atualizando os valores conforme os dados em tempo

real forem chegando. Internamente, o Spark implementa essa solução através de um mecanismo de processamento de mini-lotes, que processa dados em tempo real através de trabalhos em lotes com pequenos intervalos continuamente.

A principal ideia do *Structured Streaming* é tratar os dados que são obtidos em tempo real como fosse uma tabela que está sendo continuamente anexada. Isso resulta em um novo modelo de processamento de dados em tempo real que é semelhante a um modelo com processamento em lotes. As operações nos dados em tempo real podem ser entendidas como uma consulta lote padrão em uma tabela estática, e o Spark a executa como uma consulta incremental na tabela de entrada ilimitada [Spark 2021]. A figura 2 ilustra esse procedimento.

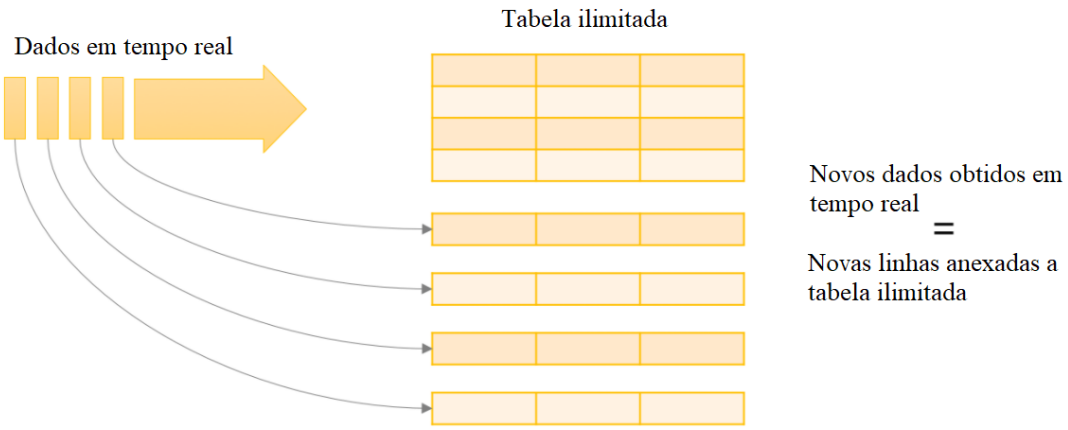

Figura 2. Representação *Structured Streaming*
Fonte: Adaptado da documentação do Apache Spark (2021)

O Spark possui uma interface para o Python chamada pySpark, essa interface além de permitir escrever aplicações Spark em Python e suportar maior parte das funcionalidades do Spark, também fornece um pySpark shell para analisar os dados iterativamente em um sistema distribuído [Spark 2021].

O Spark oferece uma variedade de soluções para problemas, mas alguns casos precisamos de uma solução mais específica, esse é o caso dos UDF (*User Defined Functions*). UDFs são funções definidas pelo usuário, um recurso para que o desenvolvedor defina funções baseadas em colunas, utilizando o vocabulário do Spark SQL de um domínio de linguagem específico (DSL) [Spark 2021].

UDFs operam uma linha de cada vez e, portanto, sofrem de alta serialização e sobrecarga de chamada, como resultado, muitos pipelines de dados definem UDFs em Java e Scala e os invocam a partir do Python. Muitos cientistas de dados utilizam Python para fazer análise de dados, para melhorar a performance das UDFs nesse caso, na versão 2.3 do Spark, foi introduzido a funcionalidade Pandas UDF, construído em cima do Apache *Arrow*, oferece o melhor dos dois mundos - a capacidade de definir UDFs de baixo custo e alto desempenho inteiramente em Python utilizando a poderosa biblioteca pandas. [Databricks 2017]

3.2. Prophet

O Prophet é um software de código aberto desenvolvido pela equipe de cientistas de dados do Facebook, pode ser programado utilizando Python e R para realizar modelagens de séries temporais. Foi projetado para ter parâmetros que possam ser ajustados sem que o usuário precise conhecer detalhadamente conceitos sobre séries temporais [Taylor and Letham 2017].

Para construir o modelo, utilizam modelos de séries temporais decompostos [Harvey and Peters 1990] com três componentes principais, que são: tendência, sazonalidade e feriados, que são combinados na seguinte equação:

$$y(t) = g(t) + s(t) + h(t) + \epsilon_t. \tag{1}$$

onde:

- $g(t)$ é a função de tendência que ajusta mudanças não periódicas no valor da série temporal.
- $s(t)$ é a função de sazonalidade que ajusta mudanças periódicas, como sazonalidade semanal e anual.
- $h(t)$ é a função que representa os efeitos dos feriados na série temporal, que ocorrem em horários irregulares em um ou mais dias.
- ϵ_t é o erro aleatório do modelo, que tem como suposição distribuição dada pela função de probabilidade normal.

Essas especificações são similares aos modelos aditivos generalizados (GAM) [Hastie and Tibshirani 1987], que é uma classe de modelos com suavizadores potencialmente não lineares aplicados ao regressor. GAMs podem ser ajustados rapidamente, utilizando as técnicas de *backfitting* ou L-BFGS [Byrd et al. 1995], os autores do prophet preferiram utilizar o L-BFGS pois assim o usuário pode mudar interativamente os parâmetros do modelo.

Esta formulação de modelo oferece as seguintes vantagens práticas:

- Flexibilidade, é possível acomodar sazonalidade com diversos períodos e deixar o usuário fazer suposições diferentes sobre a tendência.
- Ajuste é rápido, dando a possibilidade de integração dos dados com diversos tipos de aplicação.
- A interpretabilidade dos parâmetros, os usuários não precisam ter o conhecimento amplo sobre séries temporais para a utilização do modelo.

Essa abordagem foi impulsionada pela natureza dos dados de séries temporais que os cientistas de dados recebem pelo Facebook. Dada a dimensão dos dados recebidos, essa abordagem foi desenvolvido para ser robusta o suficiente para realizar predições satisfatórias em escala.

3.2.1. Métrica de qualidade do ajuste RMSE

Uma maneira comum de realizar modelagem de séries temporais é utilizando a abordagem de treino e teste. Essa abordagem separa a base de dados total em 2 partições,

podendo ser por exemplo 70% para a amostra treino e as 30% restantes para a amostra teste [Hyndman and Athanasopoulos 2018].

O modelo é ajustado tendo base a amostra treino e aplicado no conjunto teste, e assim é possível calcular uma métrica importante para avaliar a qualidade do ajuste que é chamada de RMSE (*Root Mean Squared Error*), que tem a tradução livre para português como "Raiz quadrada do erro médio". E pode ser obtida da seguinte maneira:

$$RMSE = \sqrt{\frac{1}{n} \cdot \sum_{i=1}^{n} (y_i - \hat{y_i})^2} \tag{2}$$

onde, n representa a quantidade de elementos na base de dados teste, y_i é o valor real observado e $\hat{y_i}$ é o valor estimado pelo modelo. O RMSE pode ser interpretado como a raiz quadrada da média absoluta do valor observado e estimado.

3.3. Cloud: Google Big Query e Data Studio

Big Query é um serviço de armazenamento de dados em nuvem ofertado pela Google, que permite realizar análise sobre um grande volume de dados em quase tempo real, facilitando a ingestão, armazenamento e visualização dos dados. Ele consegue ingerir dados através de processos em lotes ou em tempo real, realizar consultas usando linguagem SQL, interagir com sistemas através de APIs, além de ter integração com diversas ferramentas de visualizações de dados, entre elas o Google Data Studio [Google 2021a].

O Google Data Studio é uma ferramenta gratuita que transforma dados em relatórios e paneis para visualização das informações, com ele é possível criar gráficos, relatório interativos, adicionar textos e imagens, filtro interativo dos dados, entre outras funcionalidades. O Data Studio consegue se conectar com diversa fonte de dados como o Big Query, Planilhas, *Cloud Storage*, Facebook, Reddit, Twitter, entre outros. Além disso ele facilita o compartilhamento desses dados com outros usuários, grupos e também através de incorporamento em páginas web. [Google 2021b].

4. Resultados

Para confecção dos resultados foi utilizado a linguagem de programação Python em conjunto com o *framework* Spark na versão 3.1 utilizando a integração chamada PySpark em *cluster standalone*. Os códigos estão disponíveis para acesso em repositório público desenvolvido pelos autores (www.github.com/lyncoln/projeto_topicos_bd_2). O trabalho pode ser divido em 6 passos, que são:

1. Extrair datas em dias do consumo dos estados da base de dados.
2. Simular a entrada desses dados em N dias.
3. Utilizar *Spark Structed Stream* para ler as informações de entrada.
4. Utilizar Spark para transformar os dados.
5. Realizar modelagem de Série Temporal com Spark e Pandas Udf.
6. Criar visualização em dashboard.

Para facilitar o entendimento do leitor, cada passo será separado e desenvolvido separadamente ao longo de sub-seções, a figura a seguir representa a ilustração da arquitetura do projeto.

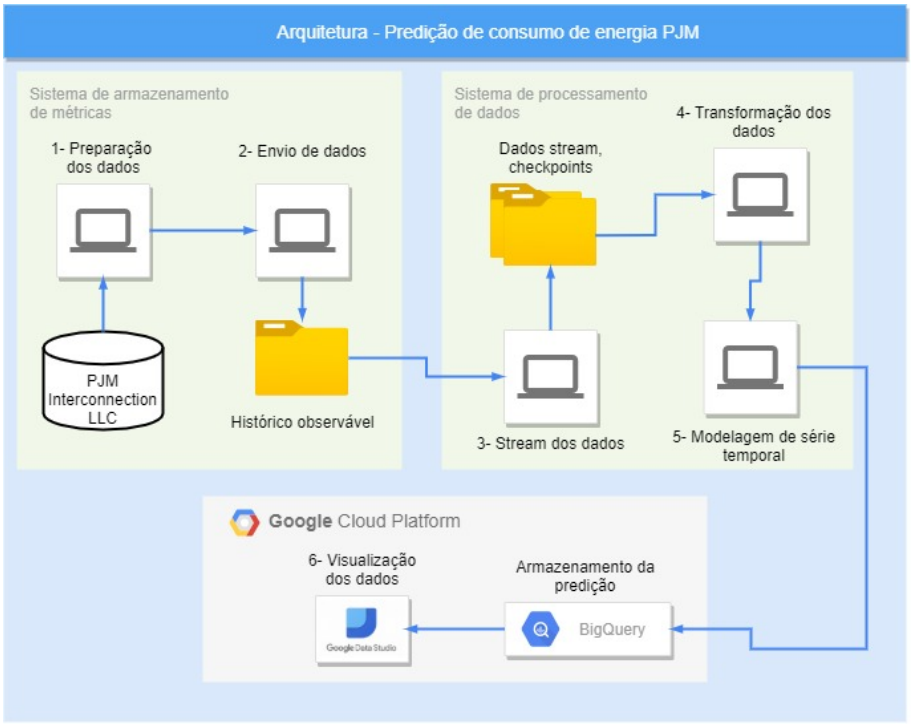

Figura 3. Arquitetura do sistema de predição de consumo de energia
Fonte: Autores (2021)

4.1. Primeiro Passo (Preparação dos dados)

Nesse passo foi realizado a preparação dos dados, o conjunto de dados utilizados neste trabalho fornecido pela *PJM Interconnection LLC(PJM)* está agrupado por estados, de forma com que cada estado possua apenas uma única tabela com dados medidos por hora ao longo de 10 anos no formato CSV.

Como o objetivo é simular a execução de uma aplicação que faz análises em tempo real, foi separado as tabelas que contem dados de uma década, em diversas tabelas que contem dados de apenas um dia. A Figura 4 representa os dados de uma dessas tabelas geradas.

	A	B
1	Datetime	MW
2	07/10/2004 01:00	124840
3	07/10/2004 02:00	120540
4	07/10/2004 03:00	117450
5	07/10/2004 04:00	117570
6	07/10/2004 05:00	120410
7	07/10/2004 06:00	128230
8	07/10/2004 07:00	143940
9	07/10/2004 08:00	154630
10	07/10/2004 09:00	154880
11	07/10/2004 10:00	155850
12	07/10/2004 11:00	155970
13	07/10/2004 12:00	155150
14	07/10/2004 13:00	155080
15	07/10/2004 14:00	155470
16	07/10/2004 15:00	155220
17	07/10/2004 16:00	153330
18	07/10/2004 17:00	152660
19	07/10/2004 18:00	149720
20	07/10/2004 19:00	150420
21	07/10/2004 20:00	157420
22	07/10/2004 21:00	158930
23	07/10/2004 22:00	154070
24	07/10/2004 23:00	143590
25	07/10/2004 00:00	132490
26		

Figura 4. Exemplo de tabela gerada após separação da tabela inicial
Fonte: Autores (2021)

4.2. Segundo Passo (Envio de dados)

Foi desenvolvido um procedimento que coleta os dados do passo anterior com base em uma data de entrada e insere as tabelas referentes das leituras dos estados nessa data em uma única pasta de arquivos chamada histórico observável, que contém os dias até então gerados dessa maneira.

Assim para realizar a simulação, bastou criar um ciclo de repetição que dado um número N de dias como entrada, começando por exemplo, pelo dia 07/10/2004 (podendo ser qualquer outro), insere essa quantidade de dados no histórico observável.

4.3. Terceiro passo (Stream dos dados)

Foi criado uma rotina utilizando *Spark Structed Stream* com o objetivo de observar a pasta local histórico observável (que está simulando a entrada da informação em dias) de maneira contínua, para cada nova data, essa informação é salva com formato CSV em uma outra pasta chamada dados stream, e também criando arquivos *checkpoints* para recuperação de dados em caso de falhas na pasta dados stream checkpoint.

Apesar de estar observando uma pasta local, esse procedimento poderia estar recebendo dados de outros lugares, como por exemplo de algum *framework* externo como

o kafka, algum banco de dados auxiliar, alguma porta *socket*, e entre outros possibilidades. Foi escolhido o sistema de pastas de arquivo local para facilitar a reprodução da simulação.

4.4. Quarto passo (Transformação dos dados)

Com a pasta de arquivos de entrada dados stream fornecidos pelo *Spark Structed Stream*, todos os arquivos dessa pasta são lidos e ordenados pela data por outro procedimento separado utilizando o Spark, que transforma esses dados em um objeto *DataFrame*. Antes de realizar a modelagem dos dados é necessário realizar algumas transformações.

Na figura 4 pode ser visto que as informações de entrada são separadas por horas. Como o objetivo do estudo é calcular a média diária do consumo de energia elétrica do conjunto dos estados, foi necessário remover as horas de leituras, assim essa coluna foi modificada com o objetivo de permanecer somente a data.

Agora, no novo conjunto de dados, foi possível utilizar a função de transformação groupBy para agrupar a coluna datas e posteriormente utilizar a função mean para realizar a média de todas as datas iguais, assim obtendo uma nova tabela com dimensão reduzida com as médias de consumo de energia elétrica por dia do conjunto de estados.

4.5. Quinto Passo (Modelagem de Série Temporal)

Utilizando a integração do PySpark com a biblioteca pandas do Python, foi desenvolvida uma nova função usando o pandas_udf que apoia a construção do modelo de séries temporais com a biblioteca Prophet do Facebook. Com a tabela de dados com formato *DataFrame* construída no passo anterior, a função separa a base de dados em 2 conjuntos de dados que são chamados de treino e teste, que possuem 70% e 30% respectivamente do total de observações da tabela de entrada.

O modelo é ajustado com o Prophet usando a base de dados treino, para o ajuste foi necessário determinar as medidas de piso e teto, que são medidas que são necessárias para a modelagem da série temporal pelo Prophet. O piso foi definido como 50% da menor leitura da tabela de entrada, isso significa que o Prophet não pode estimar um valor abaixo desse, isso é necessário pois o Prophet pode estimar valores abaixo de zero, o que não faz sentido quando o contexto é consumo de energia elétrica. E o teto foi definido como 300% da maior leitura da tabela, caso a estimativa exploda além do que seria justificativamente possível.

Além desses parâmetros, pela natureza do problema, foi escolhido 3 tipos de sazonalidade que são: diária, semanal e anual. Optou-se não utilizar o efeito dos feriados no modelo, pois teriam que ser passados cada dia de maneira manual para o modelo. Como a base de dados é obtida pelos estados dos Estados Unidos, as datas comemorativas são bem diferentes do que no Brasil, então a decisão de não utilizar esse efeito foi dada para não enviesar o ajuste para dias específicos desconhecidos.

Com o modelo ajustado pela base de dados treino, foi realizado predições na base de dados teste para obter a medida de qualidade do ajuste do modelo RMSE para os dias presentes no teste e posteriormente a predição para 28 dias no futuro. E assim a saída dessa função que utiliza o panda_udf é uma nova tabela no formato *DataFrame* do Spark onde a primeira coluna são as datas das observações, a segunda são as leituras do consumo

de energia elétrica que de fato foram observadas, a terceira é composta pelas estimativas dessas leituras e a quarta é uma coluna preenchida somente com o valor único do RMSE.

4.6. Sexto Passo (Visualização)

Após a execução do modelo de predição, os dados são transformados em um único *Data-Frame* e são enviados ao Google Big Query para armazenar os dados gerados, através de uma chamada de API do Big Query, a autenticação é feita utilizando o nome do projeto GCP (Google Cloud Platform) juntamente com a chave privada de usuário da nuvem. Foi utilizado o modo de escrita *append* que faz com que os dados novos sejam anexados no final da tabela dos dados já existentes.

Para a exibição dos dados, foi criado um *template* no Google Data Studio, que utiliza o conector do Big Query. Para ilustração, foram geradas dados de leituras do consumo de energia elétrica dos estados referentes a 1000 dias após a data de 07/10/2004 utilizando os procedimentos descritos nos passos anteriores. Foram criadas quatro objetos de visualização conforme a figura 5.

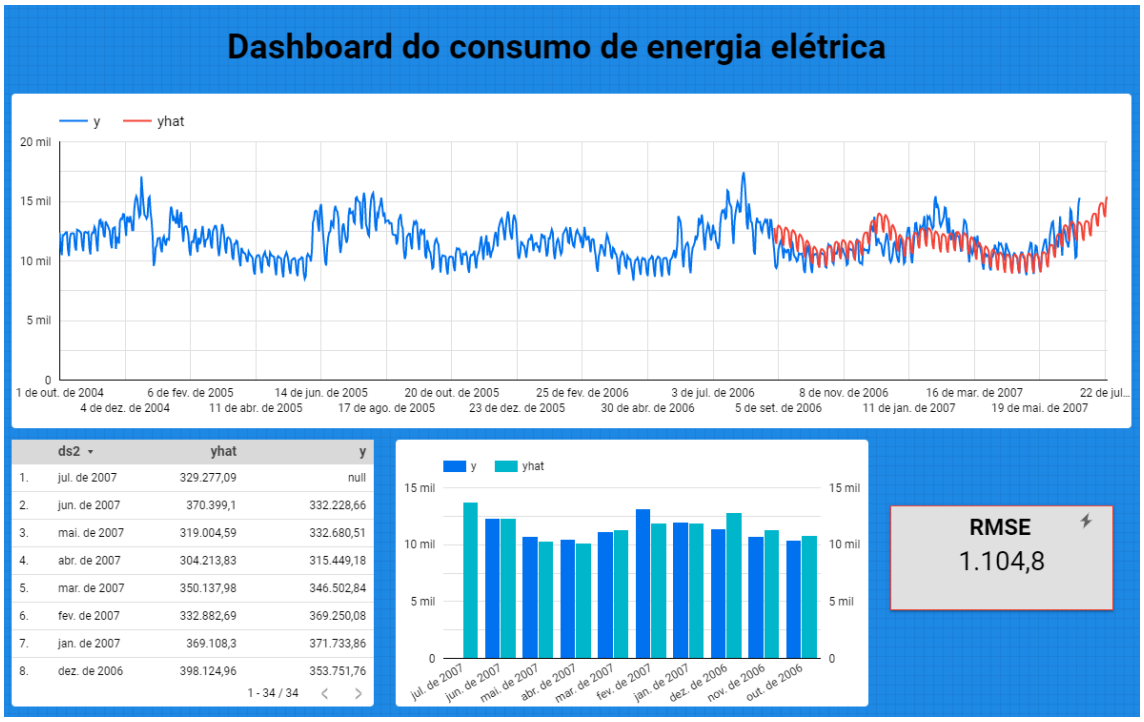

Figura 5. Visualização dos dados no Google Data Studio
Fonte: Autores (2021)

No gráfico superior, é exibido uma série temporal referente a média do consumo de energia elétrica dos estados da PJM. Os dados que de fato foram observados são representados pela linha azul e a predição pela linha vermelha. Em determinada parte possuem as duas linhas simultaneamente, isso representa a predição em cima da base de teste, com essa diferença é calculada a métrica de erro RMSE que está no quadro inferior direito. Quando há somente a linha vermelha, esses são os dados da predição para os próximos 28 dias.

Na tabela do canto inferior esquerdo, os dados estão agrupados por mês, com a primeira coluna representando o mês, a segunda coluna representando a soma da média do consumo elétricos predito dos estados para esse mês e a terceira coluna a soma das médias dos dados observados.

Existe um campo como null, esse dado representa que este mês ainda está no futuro, portanto possui apenas a predição e não o dado observado. Na imagem inferior do centro representa uma alternativa de visualização com gráfico de barras referentes aos dados da tabela a esquerda.

5. Desafios e Conclusão

5.1. Desafios

Durante o desenvolvimento da aplicação, foram feitas tentativas de implementação das seguintes ferramentas: GCP Dataproc, Kibana e Kafka. O Dataproc é um serviço gerenciado pela Google para execução do Spark em *clusters* na nuvem, foi levantado um conjunto de *cluster* para a execução da aplicação, porém houve incompatibilidade com a biblioteca Prophet do facebook. O Kafka, é um sistema de mensagens para transferência de dados entre as aplicações, foi testado em um *container*, porém houveram dificuldades em ordenação das séries temporais. Por razão de tempo limitado, foi decidido deixar a implementação do Dataproc e Kafka para trabalhos futuros.

O Kibana, é um *plugin* de visualização de dados de código aberto para o Elasticsearch, foi levantado em um *container docker*, porém houveram dificuldades de comunicação com a aplicação que estava inicialmente na nuvem, como foi realizada a tentativa de execução da aplicação no *cluster* do GCP, foi decidido usar as soluções de visualização de dados nativas do GCP, no caso o Data Studio.

5.2. Conclusão

A aplicação desenvolvida se mostrou eficiente para processar os dados de consumo elétrico dos estados da PJM. Apesar de não ter sido executado em um *cluster* de computadores e possuir quantidade pequena de dados (44 *megabytes*), os procedimentos adotados foram feitos para serem executados em paralelo escalonando com grande quantidade de dados, por exemplo, caso seja possível ter acesso as informações do consumo de energia elétrica de todas as casas de um estado ou país, a aplicação poderia ser executada somente alterando a origem dos dados.

A aplicação foi construída para realizar leituras do consumo elétrico a cada dia nos estados da PJM por questões de simulação, mas com pequenas alterações, poderia também realizar leituras para qualquer intervalo de tempo, por exemplo, cada hora. Assim para um cenário real, a interface visual poderia ser útil para facilitar a tomada de decisão rápida por parte das concessionárias de energia elétrica, e também podendo fornecer dados de entrada para alimentar alguma outra aplicação para esse objetivo.

Referências

Byrd, R. H., Lu, P., Nocedal, J., and Zhu, C. (1995). A limited memory algorithm for bound constrained optimization.

Databricks (2017). Introducing pandas udf for pyspark. https://databricks.com/blog/2017/10/30/introducing-vectorized-udfs-for-pyspark.html. Accessed: 2021-07-25.

Google (2021a). Big query docs. https://cloud.google.com/bigquery/docs. Accessed: 2021-07-25.

Google (2021b). Conheça o data studio. https://support.google.com/datastudio/answer/6283323?hl=pt-BR. Accessed: 2021-07-25.

Harvey, A. and Peters, S. (1990). Estimation procedures for structural time series models. *Journal of Forecasting*, 9:89–108.

Hastie, T. and Tibshirani, R. (1987). Generalized additive models: Some applications. *Journal of the American Statistical Association*, 82(398):371–386.

Hyndman, R. and Athanasopoulos, G. (2018). *Forecasting: Principles and Practice*. OTexts, Australia, 2nd edition.

Mauro, A. D., Greco, M., and Grimaldi, M. (2016). A formal definition of big data based on its essential features. *Library Review*, 65:122–135.

Mulla, R. (2019). Pjm hourly energy consumption data. https://www.kaggle.com/robikscube/hourly-energy-consumption. Acessed: 2021-07-01.

Spark, A. (2021). *Apache Spark Docs*.

Taylor, S. and Letham, B. (2017). Forecasting at scale. Facebook, Menlo Park, California, United States. PeerJ.

Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., and Stoica, I. (2010). Spark: Cluster computing with working sets. In *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'10, page 10, USA. USENIX Association.