Machine Learning Model

Inicialmente foi ponderado a utilização de alguns modelos de Machine Learning:

- Linear Regression
- ~~KNN~~
- ~~Decision Tree~~
- SVM
- AdaBoost
- Random Forest

Porém após testes e análises, os modelos de KNN e Decision Tree foram **descartados**, pois não apresentaram bons resultados.

Para a resolução desse problema serão considerados os outros modelos, sendo que o modelo que apresentar o melhor resultado será utilizado.

Imports

Foram utilizadas as seguintes bibliotecas:

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set(style='darkgrid')
sns.set_theme(style="ticks", rc={"axes.spines.right": False, "axes.spines
palette = 'mako'

from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import LabelEncoder, OneHotEncoder, StandardSc

from sklearn.linear_model import LinearRegression
from sklearn.svm import SVR
from sklearn.ensemble import AdaBoostRegressor
from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import r2_score, mean_absolute_error as mae

from sklearn.model_selection import GridSearchCV, RandomizedSearchCV, Lea

from keras import models, layers
```

As métricas utilizadas para avaliação dos modelos foram:

- R2 Score
- Mean Absolute Error

E será utilizada em conjunto a validação cruzada com o KFold.

Tratamento inicial

Observação dos dados

In [ ]:
```
df = pd.read_pickle('../datasets/processed.pkl')
df.drop(columns = 'segundos_depois_meia_noite', inplace = True)
df.sample(5)
```

Out[ ]:

	data	consumo_energia	corrente_atrasada	corrente_principal	co2	potenci
15775	2018-06-14 08:00:00	4.36	5.87	0.00	0.000000	
21837	2018-08-16 11:30:00	53.75	41.87	0.00	0.020004	
18675	2018-07-14 13:00:00	3.42	0.00	7.63	0.000000	
31371	2018-11-23 19:00:00	68.80	30.31	0.00	0.029999	
473	2018-05-01 22:30:00	4.10	4.32	0.00	0.000000	

In [ ]:
```
df = pd.concat([df, pd.get_dummies(df.tipo_carga)], axis = 1).drop(columns

def rename_columns(column: str):
    column = column.replace(' ', '_')
    return column.lower()

for column_list in df.columns[-3:].tolist():
    df.rename(columns = {column_list: rename_columns(column_list)}, inplac
del rename_columns
```

Após discussões obtidas durante as sessões de dúvidas, foi decidido que o tratamento inicial dos dados seria feito da seguinte forma:

- Agrupamento dos dados por dia, onde os numericos seriam as médias e os categóricos seriam a moda

In [ ]:
```
df = df.groupby(df['data'].dt.date).agg({
    'consumo_energia': 'mean',
    'corrente_atrasada': 'mean',
    'corrente_principal': 'mean',
    'co2': 'mean',
    'potencia_atrasado': 'mean',
    'potencia_principal': 'mean',
    'estado_semana': lambda x: x.mode()[0],
    'dia_semana': lambda x: x.mode()[0],
    'carga_leve': 'sum',
    'carga_maxima': 'sum',
    'carga_media': 'sum'
}).reset_index()

df.data = pd.to_datetime(df.data)
for item in df.select_dtypes('object').columns:
    df[item] = df[item].astype('category')

df.estado_semana = LabelEncoder().fit_transform(df.estado_semana)
```

In [ ]:
```
df.head()
```

Out[ ]:

	data	consumo_energia	corrente_atrasada	corrente_principal	co2	potencia_atr
0	2018-01-01	3.665208	1.735625	9.885000	0.000000	85.2
1	2018-01-02	63.694375	29.284792	1.123854	0.028333	86.5
2	2018-01-03	17.442083	5.274896	6.329583	0.007085	84.1
3	2018-01-04	3.018958	1.737604	9.319479	0.000000	81.7
4	2018-01-05	3.110937	1.949792	9.087813	0.000000	80.5

In [ ]:
```
aux = df.copy()
for item in aux.select_dtypes('category').columns:
    aux[item] = LabelEncoder().fit_transform(aux[item])

aux.corr(numeric_only=True).abs()
```

Out[ ]:

	consumo_energia	corrente_atrasada	corrente_principal	co2	potenci
consumo_energia	1.000000	0.937050	0.816783	0.990505	
corrente_atrasada	0.937050	1.000000	0.845748	0.932578	
corrente_principal	0.816783	0.845748	1.000000	0.821324	
co2	0.990505	0.932578	0.821324	1.000000	
potencia_atrasado	0.024184	0.223848	0.322724	0.010454	
potencia_principal	0.857353	0.839868	0.947876	0.856806	
estado_semana	0.641788	0.702574	0.687508	0.637512	
dia_semana	0.373589	0.386223	0.418245	0.364504	
carga_leve	0.697431	0.664797	0.787385	0.698431	
carga_maxima	0.697431	0.664797	0.787385	0.698431	
carga_media	0.697431	0.664797	0.787385	0.698431	

In [ ]:
```
del aux
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 365 entries, 0 to 364
Data columns (total 12 columns):
 #   Column              Non-Null Count  Dtype         
---  ------              --------------  -----         
 0   data                365 non-null    datetime64[ns]
 1   consumo_energia     365 non-null    float64       
 2   corrente_atrasada   365 non-null    float64       
 3   corrente_principal  365 non-null    float64       
 4   co2                 365 non-null    float32       
 5   potencia_atrasado   365 non-null    float64       
 6   potencia_principal  365 non-null    float64       
 7   estado_semana       365 non-null    int64         
 8   dia_semana          365 non-null    category      
 9   carga_leve          365 non-null    int64         
 10  carga_maxima        365 non-null    int64         
 11  carga_media         365 non-null    int64         
dtypes: category(1), datetime64[ns](1), float32(1), float64(5), int64(4)
memory usage: 30.8 KB
```

Testes e experimentações com os dados e modelos

In [ ]:
```
features = df.drop(columns = 'consumo_energia')
target = df.consumo_energia
```

Temos algumas alternativas para selecionar os dados que serão utilizados no modelo:

- Utilizar todos os dados disponíveis;
- Remover os dados com correlação baixa/alta com a variável alvo;
- Remover os dados com correlação alta entre si.

Foram realizados alguns testes relacionados a essas alternativas, e os resultados foram os seguintes:

- Foi feito alguns testes com todos os dados disponíveis, porém os resultados não foram satisfatórios;
- A variável Co2 possui uma correlação extremamente alta com a variável alvo, porém o seu uso tende a overfitting, então foi decidido que ela não seria utilizada. Além disso essa variável não estará disponível no momento da predição;
- Foram realizados alguns testes removendo as variáveis com correlação alta entre si, demonstrando resultados satisfatórios.

In [ ]:
```
# Assim como previsto no notebook de EDA, as variáveis 'co2' e 'potencia_
features.drop(columns = ['co2', 'potencia_principal'], inplace = True)
```

Para treino foram utilizados os dados referentes aos meses de janeiro a novembro, e para teste foi utilizado o mês de dezembro.

In [ ]:
```
test = [12]
idx_train = features[~features.data.dt.month.isin(test)].drop(columns = [
idx_test = features[features.data.dt.month.isin(test)].drop(columns = ['da
features.drop(columns = ['data'], inplace = True)
del test
```

In [ ]:
```
def create_train(features=features, target=target): # Para facilitar a cri
    return features.iloc[idx_train], features.iloc[idx_test], target.iloc
```

In [ ]:
```
x_train, x_test, y_train, y_test = create_train(features, target)
```

Para facilitar a criação do pipeline, foi desenvolvida essa função, que além de retornar o pipeline completo, possui a opção de não realizar o escalonamento dos dados, pois alguns modelos não necessitam dessa etapa.

In [ ]:
```
def create_model(features: pd.DataFrame, model, scaler=True) -> Pipeline:
    categorical = features.select_dtypes(include='category').columns.to_li
    cat_transformer = Pipeline(steps=[('encoder', OneHotEncoder(handle_unk

    transformers = [('cat', cat_transformer, categorical)]

    if scaler:
        numerical = features.select_dtypes(include='number').columns.to_li
        num_transformer = Pipeline(steps=[('scaler', StandardScaler())])
        transformers.insert(0, ('num', num_transformer, numerical))

    preprocessor = ColumnTransformer(transformers=transformers)

    steps = [('preprocessor', preprocessor), ("model", model)]

    return Pipeline(steps=steps)
```

Para análise das features foi utilizado o método de correlação de Pearson, e foi observado como os treinos e testes se comportam ao utilizar as features com maior correlação com a variável alvo, com o objetivo de encontrar um ponto de equilíbrio entre a quantidade de features e a acurácia do modelo.

In [ ]:
```
def curve_of_feature(x_train: pd.DataFrame, x_test: pd.DataFrame, y_train
    scaler = StandardScaler()
    score = { 'Error_train': [], 'Error_test': [] }

    higher_in_train = [0, 0]
    higher_in_test = [0, 0]

    for i in range(1, features.shape[1]+1):

        x_train_temp, x_test_temp = x_train[var[:i]], x_test[var[:i]]

        categorical = x_train_temp.select_dtypes(include='category').colur
        x_train_temp = pd.get_dummies(x_train_temp, columns=categorical, c
        x_test_temp = pd.get_dummies(x_test_temp, columns=categorical, dro
        x_train_temp = scaler.fit_transform(x_train_temp)
        x_test_temp = scaler.transform(x_test_temp)

        new_model = model
        new_model.fit(x_train_temp, y_train)
        y_pred_train = new_model.predict(x_train_temp)
        y_pred_test = new_model.predict(x_test_temp)

        error_train, error_test = r2_score(y_train, y_pred_train), r2_scor

        if i == 1: best_diff = [abs(error_train - error_test), i]
        elif error_test > tol or error_train > tol:
            if abs(error_train - error_test) < best_diff[0]: best_diff =
        if error_train > higher_in_train[0]: higher_in_train = [error_trai
        if error_test > higher_in_test[0]: higher_in_test = [error_test, i

        score['Error_train'].append(error_train); score['Error_test'].appe

    return pd.DataFrame(score), higher_in_train, higher_in_test, best_dif

def plot_curve_of_features(x_train: pd.DataFrame, x_test: pd.DataFrame, y_
    fig = plt.figure(figsize=(16, 8))
    i = 0
    for name, model in model_list.items():
        curve, higher_in_train, higher_in_test, best_diff = curve_of_featu
        ax = fig.add_subplot(2, 2, i+1)
        ax.set_title(name)
        x = np.arange(1, curve.shape[0]+1)
        sns.lineplot(data=curve, x=x, y='Error_train', marker= '.', marker
        ax.scatter(higher_in_train[1], higher_in_train[0], color='r', marl
        sns.lineplot(data=curve, x=x, y='Error_test', marker= '.', markers
        ax.scatter(higher_in_test[1], higher_in_test[0], color='g', marker
        plt.axvline(x=best_diff[1], color='b', linestyle='--')
        ax.set_xticks(x)
        ax.set_ylabel('R2 Score')
        if higher: ax.set_xlabel('Number of features with highest correlat
        else: ax.set_xlabel('Number of features with lower correlation')
        ax.legend()
        i+=1
    fig.tight_layout()
```

Utilizamos alguns gráficos para melhor visualização dos resultados:

In [ ]:
```
features_test = features.copy()
for item in features_test.select_dtypes('category').columns:
    features_test[item] = LabelEncoder().fit_transform(features_test[item]

corr_strong = pd.concat([features_test, target], axis = 1).corr(numeric_or

del features_test

# corr_weak = corr_strong[::-1]
```

A partir dos dados tratados, foram realizados testes com os seguintes modelos:

- Regressão Linear
- SVR
- Ada Boost
- Random Forest

In [ ]:
```
model_list = {
    'Linear Regression': LinearRegression(),
    'SVR': SVR(),
    'AdaBoost Regressor': AdaBoostRegressor(random_state=777),
    'Random Forest Regressor': RandomForestRegressor(random_state=777, n_
}
```

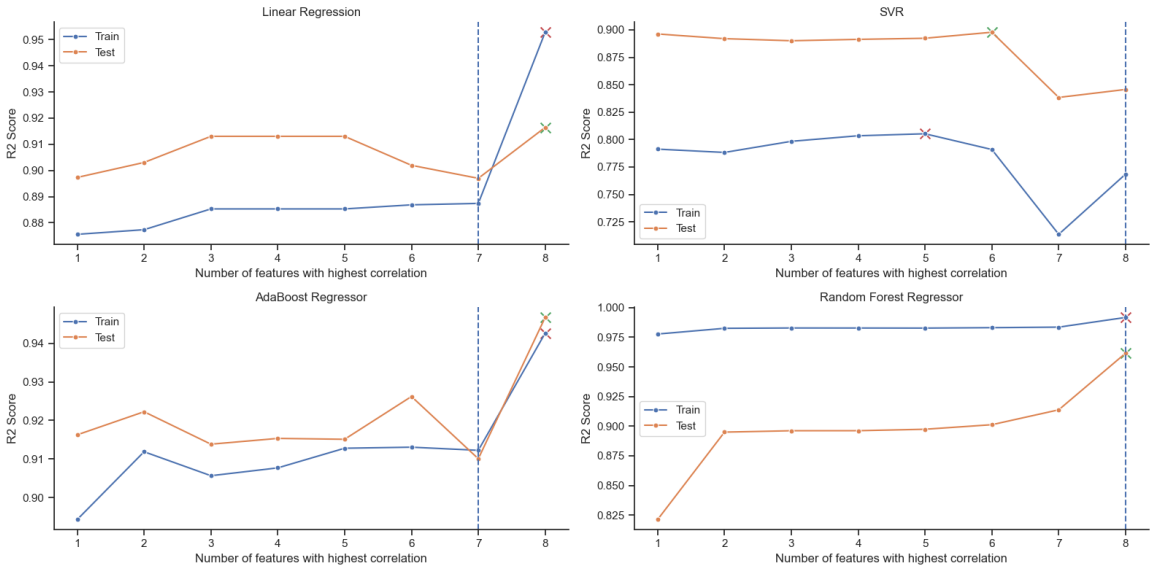
> Foi desconsiderada a análise das variáveis de maneira crescente, pois o resultado não foi satisfatório.

In [ ]:
```
# plot_curve_of_features(x_train, x_test, y_train, y_test, corr_weak, mod
# plt.show()
```

Estes gráficos demonstram a acurácia dos modelos em relação a quantidade de features utilizadas:

- Estão marcadados os pontos de maior acurácia de cada modelo, para treino e teste.
- A linha tracejada representa o menor intervalo de acurácia entre treino e teste.

In [ ]:
```
plot_curve_of_features(x_train, x_test, y_train, y_test, corr_strong, mode
plt.show()
```



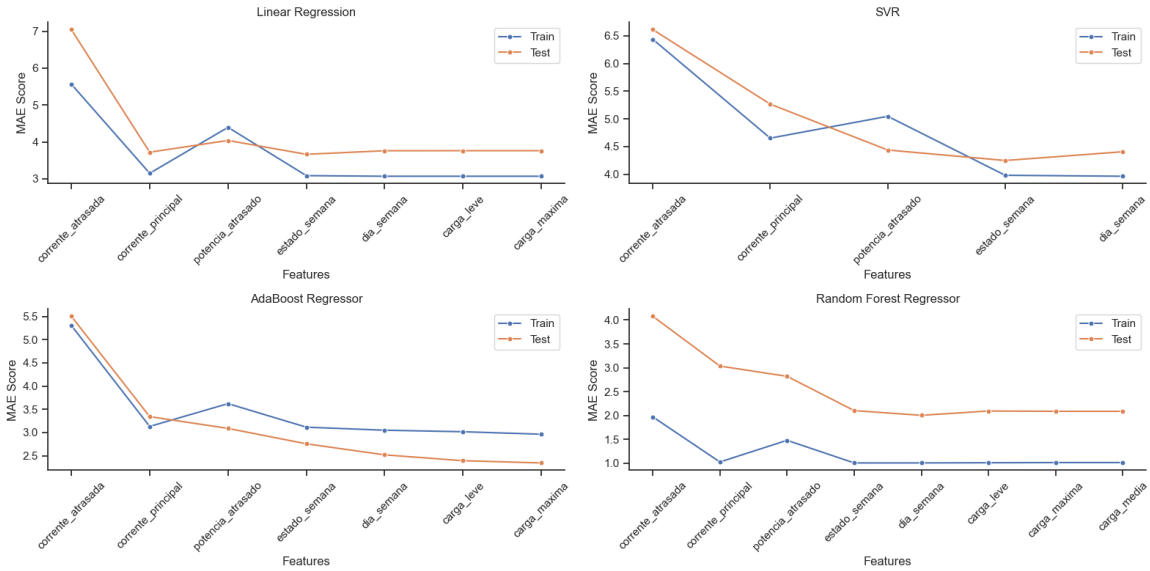
Como cada modelo possui uma quantidade de features que apresenta o melhor resultado, foi decidido customizar a quantidade de features utilizadas em cada modelo, de acordo com o gráfico acima.

In [ ]:
```
features_list = {
    'Linear Regression': features.iloc[:, :7].copy(),
    'SVR': features.iloc[:, :5].copy(),
    'AdaBoost Regressor': features.iloc[:, :7].copy(),
    'Random Forest Regressor': features.iloc[:, :8].copy()
}
```

In [ ]:
```
def drop_of_features(features, target, model):
    score = { 'Error_train': [], 'Error_test': [] }
    for feat in features.columns:
        features[feat] = LabelEncoder().fit_transform(features[feat])

    x_train, x_test, y_train, y_test = create_train(features, target)

    for feat in features.columns:
        x_train_temp, x_test_temp = x_train.drop(columns=feat), x_test.dr

        scaler = StandardScaler()
        x_train_temp = scaler.fit_transform(x_train_temp)
        x_test_temp = scaler.transform(x_test_temp)

        model.fit(x_train_temp, y_train)
        y_pred_train = model.predict(x_train_temp)
        y_pred_test = model.predict(x_test_temp)

        score['Error_train'].append(mae(y_train, y_pred_train))
        score['Error_test'].append(mae(y_test, y_pred_test))

    return features.columns, pd.DataFrame(score)

def plot_drop_of_features(features, target, model_list):
    fig = plt.figure(figsize=(16, 8))
    i = 0
    for name, model in model_list.items():
        features_to_remove, curve = drop_of_features(features_list[name],
        ax = fig.add_subplot(2, 2, i+1)
        ax.set_title(name)
        x = np.arange(1, curve.shape[0]+1)
        sns.lineplot(data=curve, x=x, y='Error_train', marker= '.', marke
        sns.lineplot(data=curve, x=x, y='Error_test', marker= '.', marker
        ax.set_xlabel('Features')
        ax.set_ylabel('MAE Score')
        ax.set_xticks(x)
        ax.set_xticklabels(features_to_remove.to_list(), rotation=45)
        ax.legend()
        i+=1
    fig.tight_layout()
```

Após a customização, foi realizada mais uma rodada de testes relacionadas a seleção de features, desta vez dropando as colunas restantes e observando o resultado.

In [ ]:
```
plot_drop_of_features(features, target, model_list)
plt.show()
```



Após a análise dos resultados, foi decidido eliminar mais uma feature de cada modelo para prosseguirmos com os testes.

In [ ]:
```
features_list['Linear Regression'].drop(columns='potencia_atrasado', inpla
features_list['SVR'].drop(columns='estado_semana', inplace=True)
features_list['AdaBoost Regressor'].drop(columns='estado_semana', inplace=
features_list['Random Forest Regressor'].drop(columns='potencia_atrasado'
```

Algumas funções para facilitar a análise dos resultados

In [ ]:
```
def create_learning_curves(model_list: dict, features: dict, target: pd.Se
    fig = plt.figure(figsize=(12, 7))
    for i, (algorithm, model_) in enumerate(model_list.items()):
        ax = fig.add_subplot(n_row, n_col, i + 1)
        plt.title(f"{algorithm}", fontsize='large', fontweight='bold')
        plt.grid(visible=True, color='black', linewidth=.25)
        plt.xlabel("Number of Samples")
        LearningCurveDisplay.from_estimator(model_, features[algorithm], t
    fig.tight_layout()
```

In [ ]:
```
def create_scores(model_list: dict, features: dict, target: pd.Series) ->
    scores = { 'Algorithm': [], 'Type': [], 'R2': [], 'Mean Absolute Erro

    for algorithm, the_model in model_list.items():
        x_train, x_test, y_train, y_test = create_train(features[algorithm

        for item in ['Train', 'Test']:
            model = the_model
            model.fit(x_train, y_train)

            if item == 'Train':
                pred_y = model.predict(x_train)
                scores['Type'].append('Train')
                pred = y_train
            else:
                pred_y = model.predict(x_test)
                scores['Type'].append('Test')
                pred = y_test

            scores['Algorithm'].append(algorithm)
            scores['R2'].append( round( r2_score(pred, pred_y), 2) )
            scores['Mean Absolute Error'].append( round( mae(pred, pred_y

    return pd.DataFrame(scores)
```

In [ ]:
```
def create_plot(results: pd.DataFrame, n_rows = 2, n_cols = 2, rot=45) ->
    fig = plt.figure(figsize=(16, 10))
    for i, title in enumerate(results.columns[2:]):
        ax = fig.add_subplot(n_rows, n_cols, i + 1)
        sns.barplot(data=results, x='Algorithm', y=title, hue='Type', pale
        for value in ax.containers:
            ax.bar_label(value)
        ax.tick_params(axis='x', labelrotation=rot, size=12)
        ax.tick_params(axis='y', size=12)
        ax.set(ylabel=None, xlabel=None)
        ax.set_title(f"{title.capitalize()}", fontsize='large', fontweight
                style='italic', family='monospace')
        ax.grid(visible=True, color='black', linewidth=.25)
    fig.tight_layout()
```

In [ ]:
```
def cross_validation(model, model_name: str, features: pd.DataFrame, targe
    kf = KFold(n_splits=cv)
    scores = { 'model': model_name, 'train_score': [], 'test_score': [],

    for train_index, test_index in kf.split(features):
        train_x, train_y = features.iloc[train_index], target.iloc[train_
        test_x, test_y = features.iloc[test_index], target.iloc[test_inde

        new_model = model
        new_model.fit(train_x, train_y)

        pred_train_y = new_model.predict(train_x)
        pred_test_y = new_model.predict(test_x)

        scores['train_score'].append( r2_score(train_y, pred_train_y) )
        scores['test_score'].append( r2_score(test_y, pred_test_y) )
        scores['train_mae'].append( mae(train_y, pred_train_y) )
        scores['test_mae'].append( mae(test_y, pred_test_y) )

    return pd.DataFrame(scores)
```

In [ ]:
```
def plot_validation_curve(results: list, title: list, type: str, n_row=2,
    fig = plt.figure(figsize=(12, 7))
    for i, item in enumerate(results):
        ax = fig.add_subplot(n_row, n_col, i + 1)
        x = np.arange(item.shape[0])

        if type == 'r2': metric_train, metric_test = 'train_score', 'test_
        else: metric_train, metric_test = 'train_mae', 'test_mae'

        if item[metric_train].max() < item[metric_test].max(): max = item
        else: max = item[metric_train].max()
        if item[metric_train].min() > item[metric_test].min(): min = item
        else: min = item[metric_train].min()

        min, max = round(min, 1), round(max, 1)

        sns.lineplot(data=item, x=x, y=metric_train, label='Training Score
        sns.lineplot(data=item, x=x, y=metric_test, label='Testing Scores

        if type == 'r2': ax.set(ylabel='R2 Score')
        else: ax.set(ylabel='MAE Score')

        ax.grid(visible=True, color='black', linewidth=.5)
        ax.set_title(title[i])
        ax.set_ylim(min-0.1, max+0.1)

    fig.tight_layout()
```

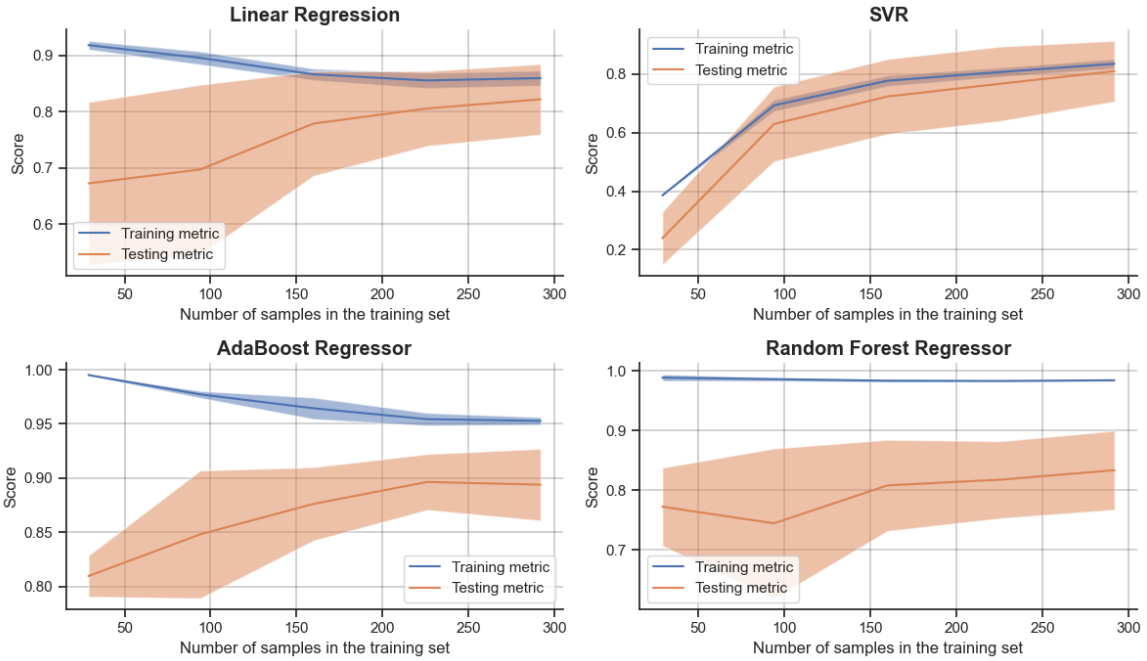
Testes de modelos com as features selecionadas

In [ ]:
```
model_list = {
    'Linear Regression': create_model(features_list['Linear Regression'],
    'SVR': create_model(features_list['SVR'], SVR()),
    'AdaBoost Regressor': create_model(features_list['AdaBoost Regressor']
    'Random Forest Regressor': create_model(features_list['Random Forest R
}
```

Ao observar a curva de aprendizado dos modelos, observa-se que:

- Linear Regression: Apesar de ser um modelo simples, ele apresenta um resultado satisfatório, e não apresenta overfitting;
- SVR: Apresenta a melhor generalização entre os modelos;
- AdaBoost: Ao final da curva o treino e teste tendem a estabilizar, porém o resultado não é satisfatório;
- Random Forest: O modelo apresenta overfitting.

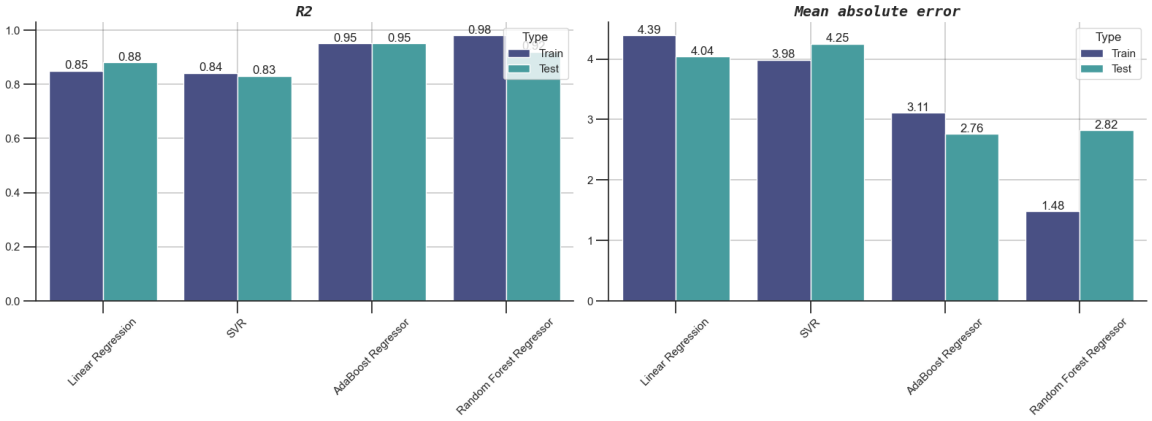
In [ ]:
```
create_learning_curves(model_list, features_list, target)
plt.show()
```



Observando esse comparativo utilizando as métricas R2 e MAE:

- Linear Regression: Apesar de ser um modelo mais simples apresenta resultados satisfatórios, porém não é o melhor;
- SVR: Apesar de generalizar melhor o problema, não apresenta um resultado satisfatório;
- AdaBoost: Mantém um resultado satisfatório e constante;
- Random Forest: Como visto acima, o modelo apresenta overfitting, apesar de apresentar os melhores resultados.

In [ ]:
```
scores = create_scores(model_list, features_list, target)

create_plot(scores)
plt.show()
```



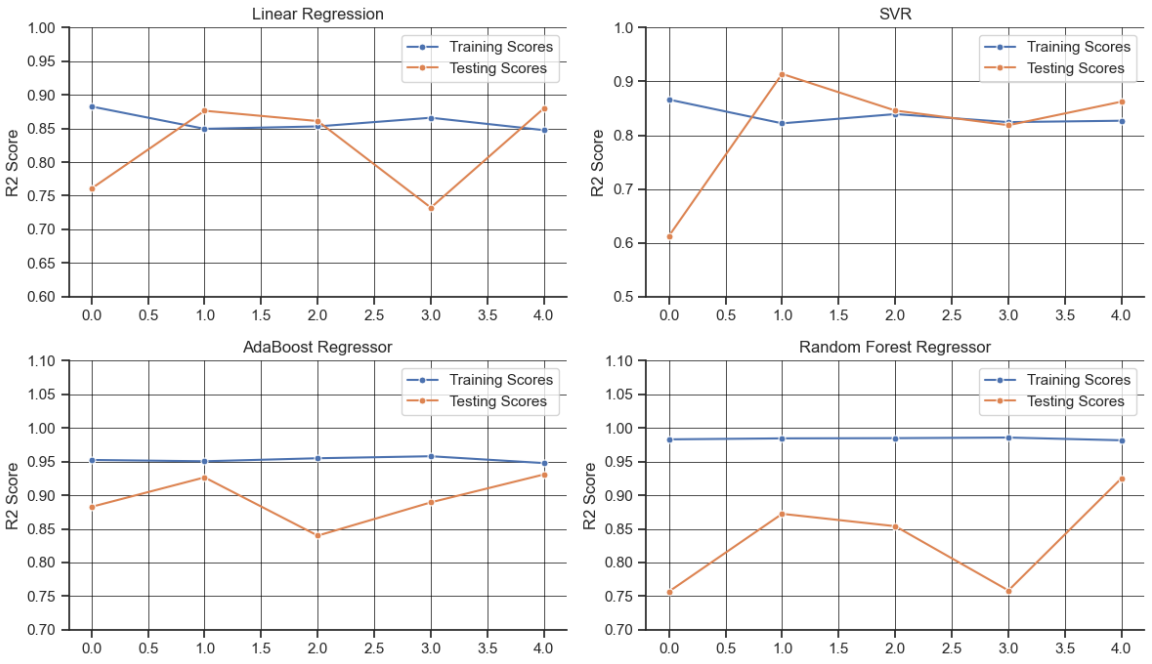
Agora vamos observar mais detalhadamente como o modelo se comporta com uma validação cruzada com 5 folds.

In [ ]:
```
model_results = [cross_validation(model, model_name, features_list[model_
```

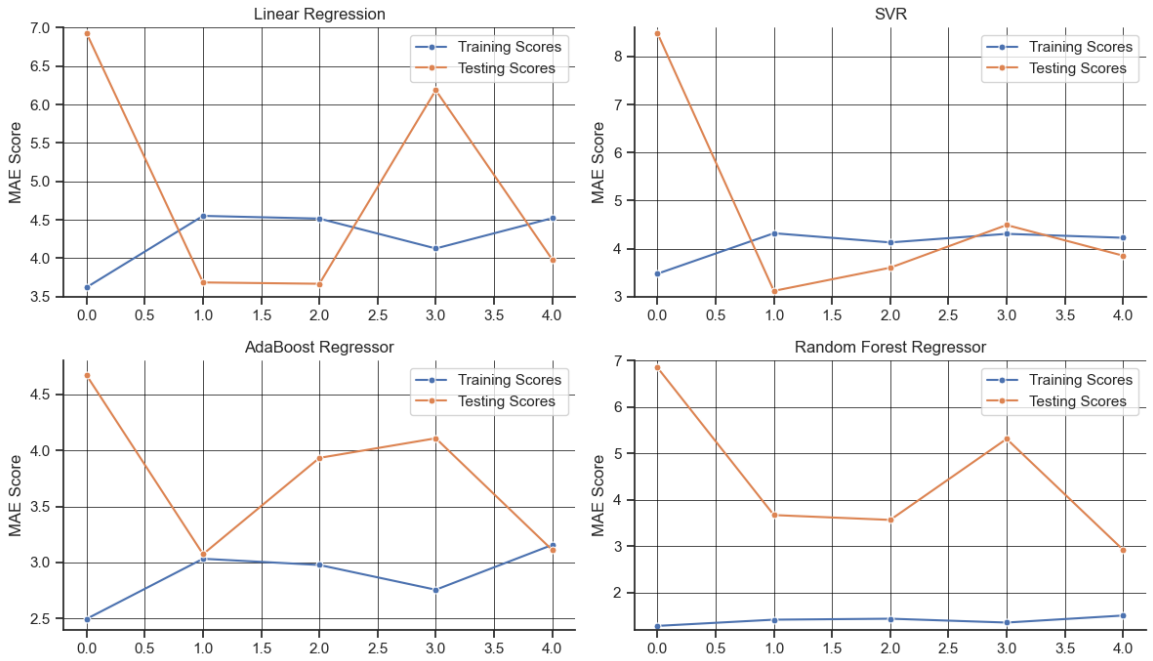
Ao comparar o rendimento dos modelos, com as duas métricas, observa-se que:

- O SVR apresenta o melhor resultado.

In [ ]:
```
plot_validation_curve(model_results, list(model_list.keys()), 'r2')
plt.show()
```



In [ ]:
```
plot_validation_curve(model_results, list(model_list.keys()), 'mae')
plt.show()
```



Tunagem de hiperparâmetros

Não foi possível utilizar o métodos de Search com as variáveis categóricas do Pipeline, então foi decidido modificar essas features manualmente para realizar a tunagem de hiperparâmetros.

In [ ]:
```
# features_grid = features_list.copy()

# for name, feature in features_grid.items():
#     numerical = feature.select_dtypes(include='number').columns
#     categorical = feature.select_dtypes(include='category').columns

#     feature[numerical] = StandardScaler().fit_transform(feature[numeric
#     feature = pd.get_dummies(feature, columns=categorical, drop_first=T
#     features_grid[name] = feature
```

In [ ]:
```
# param_grid = {
#     'copy_X': [True, False],
#     'fit_intercept': [True, False],
#     'positive': [True, False]
# }

# grid = GridSearchCV(LinearRegression(), param_grid, scoring='r2', cv=5,
# grid.fit(features_grid['Linear Regression'], target)
# grid.best_params_
```

In [ ]:
```
# param_grid = {
#     'kernel': ['linear', 'poly', 'rbf', 'sigmoid'],
#     'degree': [1, 2, 3, 4, 5],
#     'gamma': ['scale', 'auto'],
#     'C': [1.0, 2.0, 3.0, 4.0, 5.0],
#     'epsilon': [0.1, 0.2, 0.3, 0.4, 0.5]
# }

# grid = GridSearchCV(SVR(), param_grid, scoring='r2', cv=5, n_jobs=-1, v
# grid.fit(features_grid['SVR'], target)
# grid.best_params_
```

In [ ]:
```
# param_grid = {
#     'n_estimators': np.arange(5, 51, 5),
#     'learning_rate': np.arange(0.1, 1.1, 0.1),
#     'loss': ['linear', 'square', 'exponential']
# }

# grid = GridSearchCV(AdaBoostRegressor(random_state=777), param_grid, sc
# grid.fit(features_grid['AdaBoost Regressor'], target)
# grid.best_params_
```

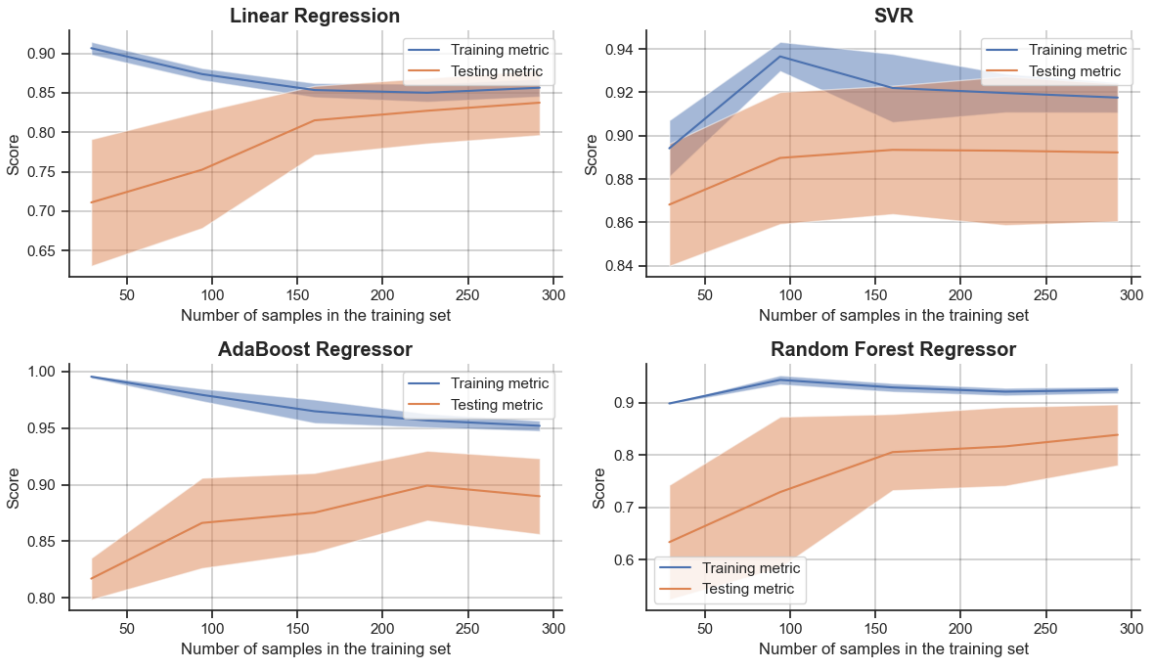
In [ ]:
```
# param_grid = {
#     'max_depth': np.arange(1, 10),
#     'min_samples_split': np.arange(2, 10),
#     'min_samples_leaf': np.arange(1, 10),
#     'max_features': ['sqrt', 'log2'],
#     'n_estimators': np.arange(15, 100, 15)
# }

# grid = GridSearchCV(RandomForestRegressor(random_state=777), param_grid
# grid.fit(features_grid['Random Forest Regressor'], target)
# grid.best_params_
```

In [ ]:
```
# del features_grid
```

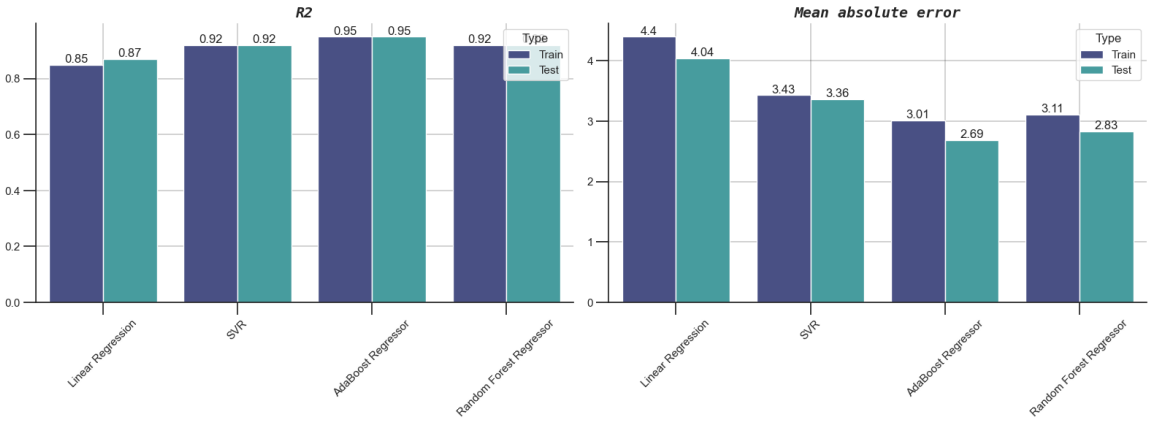
Testes de modelos tunados

In [ ]:
```
model_list = {
    'Linear Regression': create_model(features_list['Linear Regression'],
    'SVR': create_model(features_list['SVR'], SVR(C=3.0, degree=1, epsilon
    'AdaBoost Regressor': create_model(features_list['AdaBoost Regressor']
    'Random Forest Regressor': create_model(features_list['Random Forest R
}
```

In [ ]:
```
create_learning_curves(model_list, features_list, target)
plt.show()
```

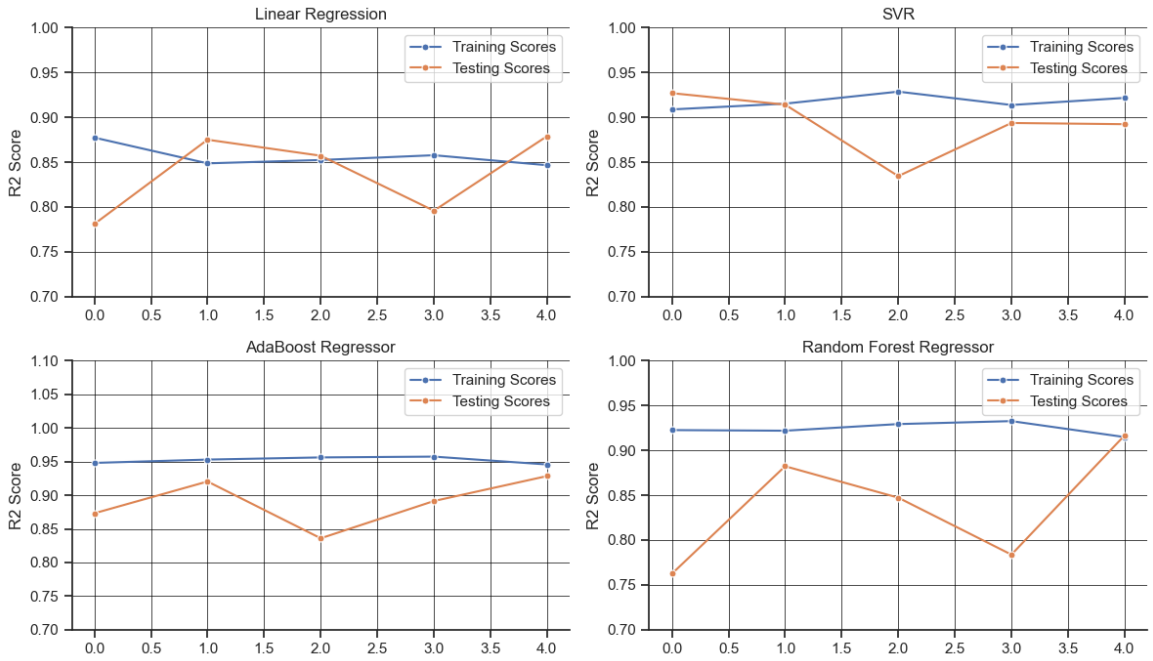


In [ ]:
```
scores = create_scores(model_list, features_list, target)

create_plot(scores)
plt.show()
```

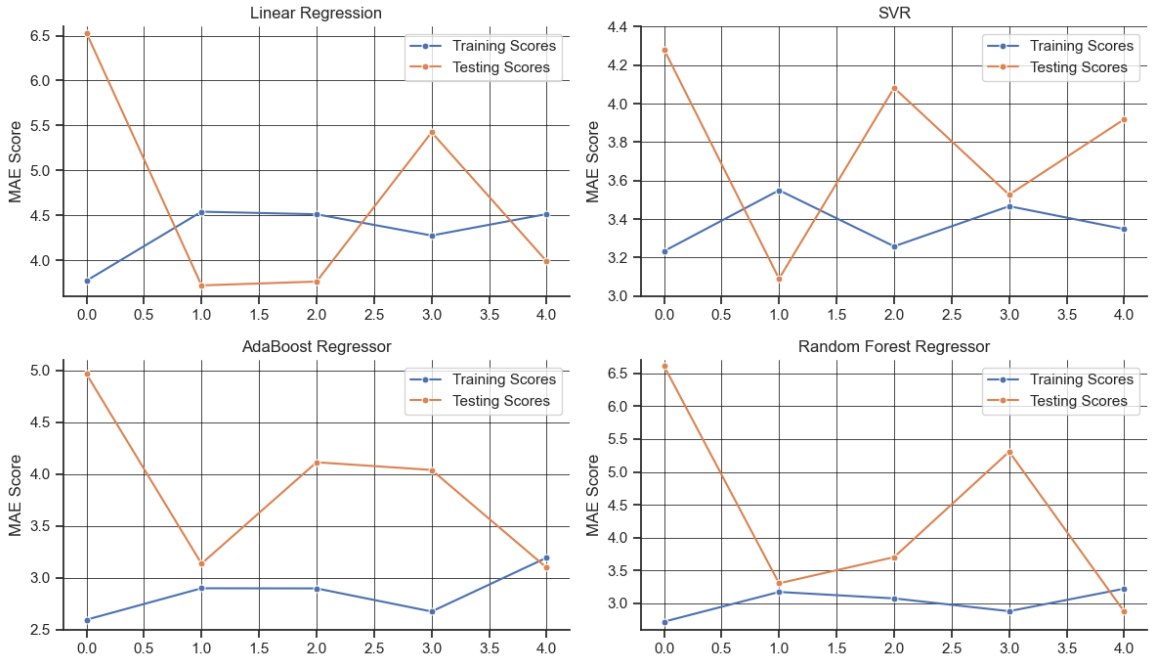


In [ ]:
```
model_results = [cross_validation(model, model_name, features_list[model_
```

In [ ]:
```
plot_validation_curve(model_results, list(model_list.keys()), 'r2')
plt.show()
```

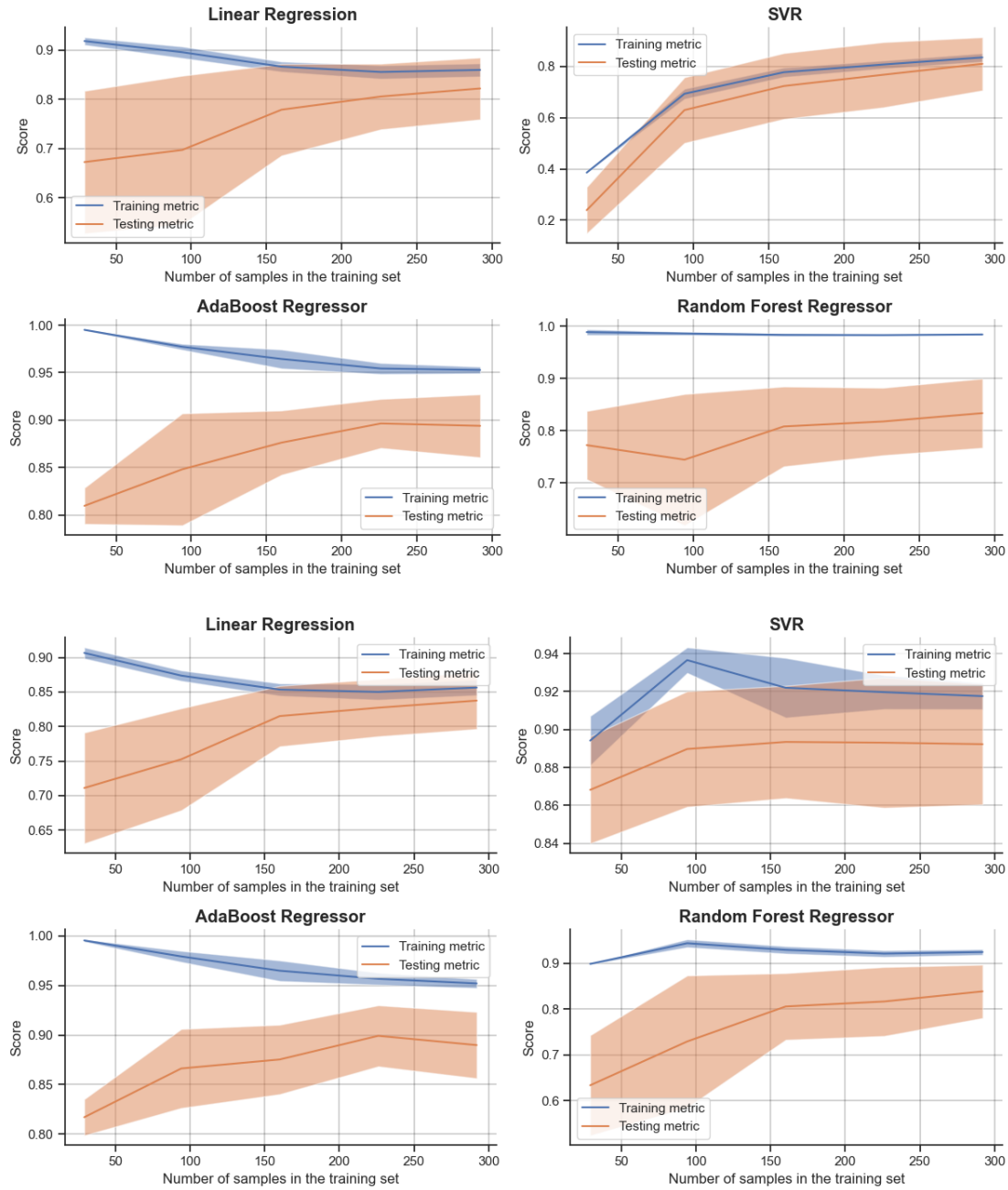


In [ ]:
```
plot_validation_curve(model_results, list(model_list.keys()), 'mae')
plt.show()
```

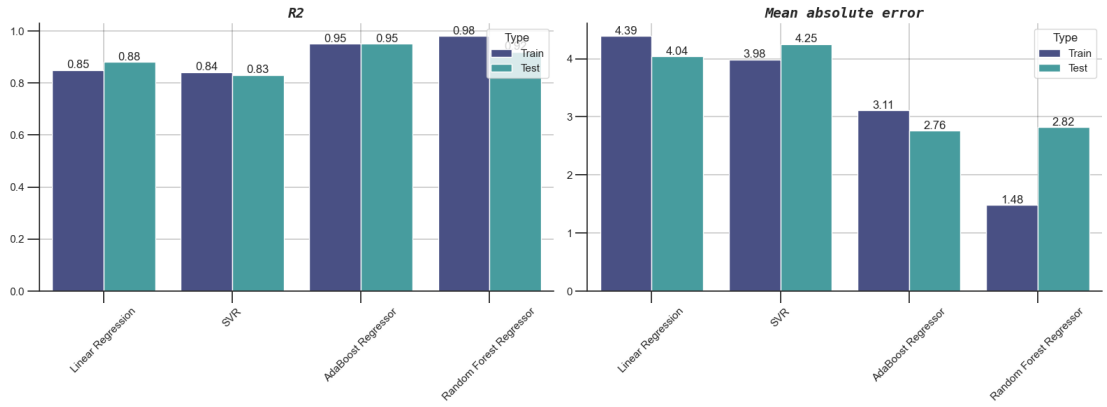


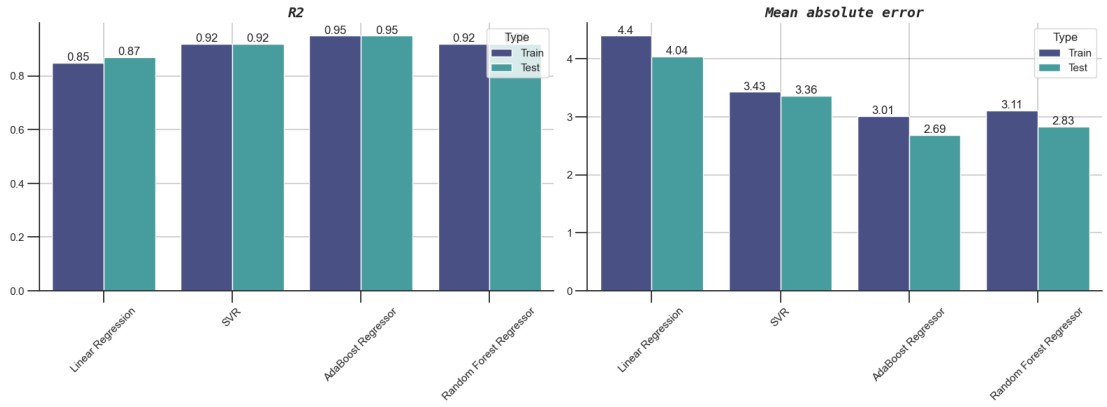
Análise dos resultados

Learning Curve

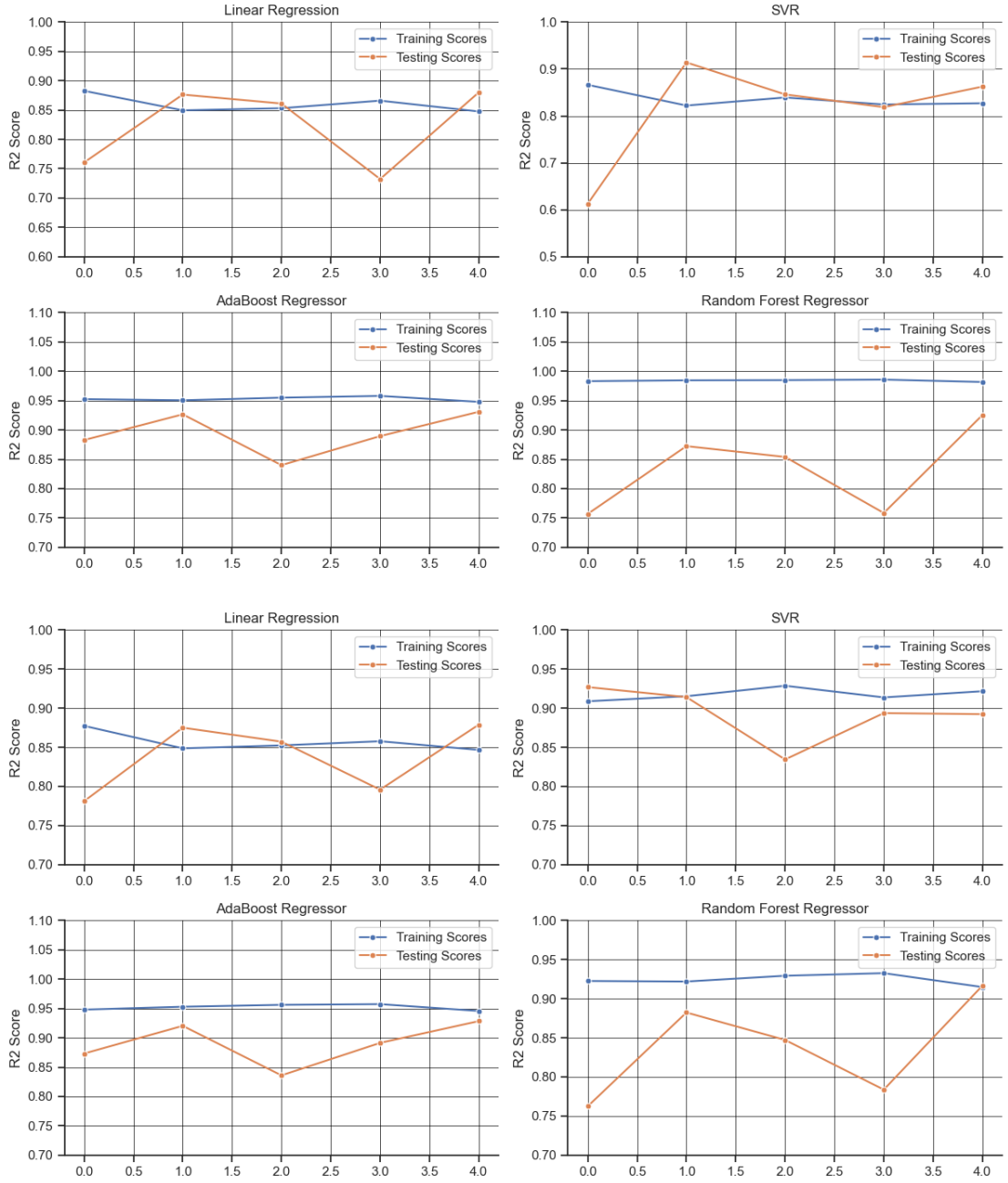

Linear Regression
SVR
AdaBoost Regressor
Random Forest Regressor
Linear Regression
SVR
AdaBoost Regressor
Random Forest Regressor
Training metric
Testing metric
Score
Number of samples in the training set

Scores


R2
Mean absolute error
Type
Train
Test
0.85
0.88
0.84
0.83
0.95
0.95
0.98
4.39
4.04
3.98
4.25
3.11
2.76
1.48
2.82
Linear Regression
SVR
AdaBoost Regressor
Random Forest Regressor

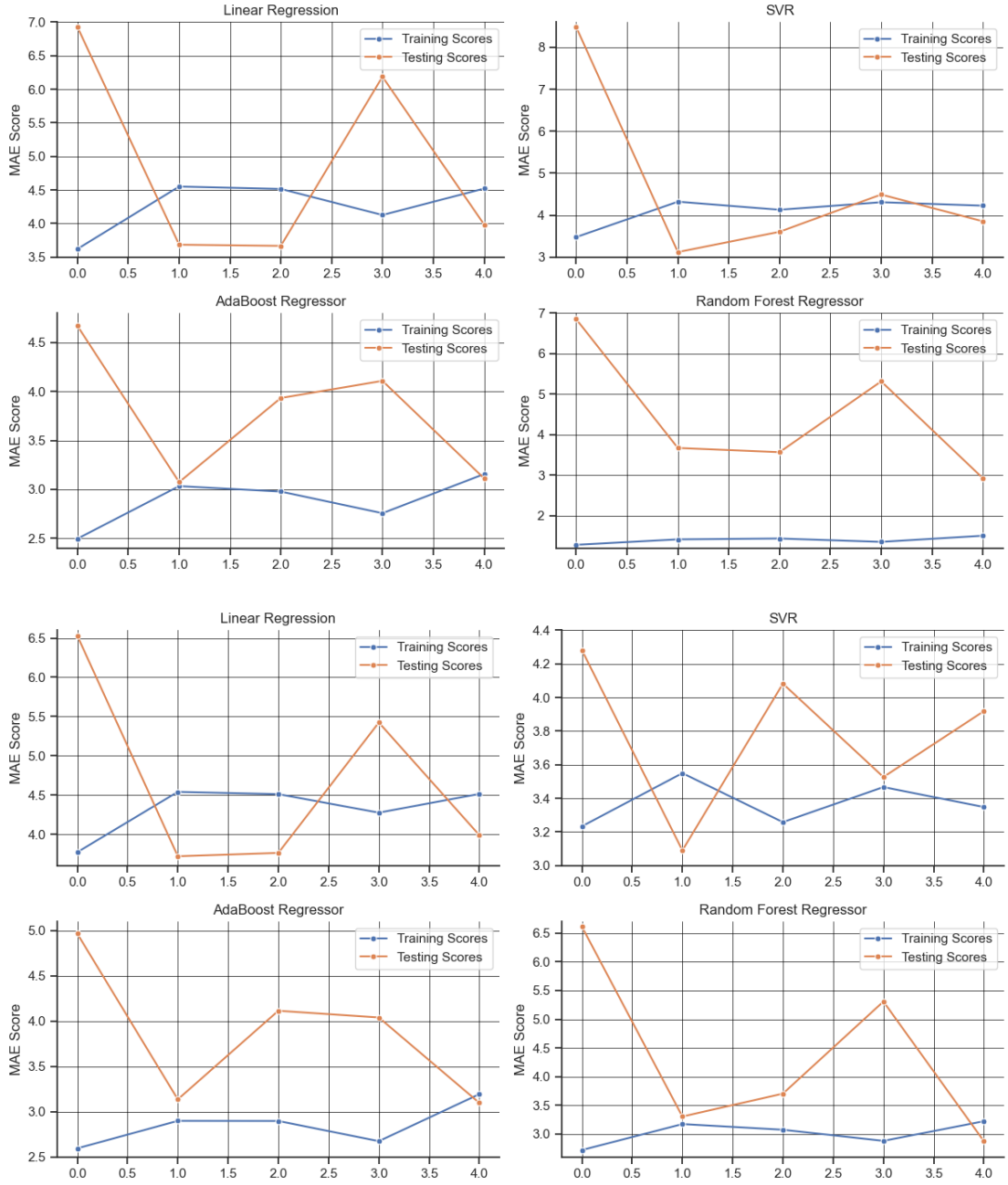

R2
Mean absolute error
Type
Train
Test
0.85
0.87
0.92
0.92
0.95
0.95
0.92
4.4
4.04
3.43
3.36
3.01
2.69
3.11
2.83
Linear Regression
SVR
AdaBoost Regressor
Random Forest Regressor

KFold (R2)


Linear Regression
SVR
AdaBoost Regressor
Random Forest Regressor
Training Scores
Testing Scores
R2 Score

Linear Regression
SVR
AdaBoost Regressor
Random Forest Regressor
Training Scores
Testing Scores
R2 Score

KFold (MAE)



Redes neurais

Também foi testado o modelo de redes neurais.

In [ ]:
```
features_neural = df.drop(columns=['data']).copy()
features_neural = pd.get_dummies(features_neural, drop_first=True)

x_train, x_test, y_train, y_test = features_neural.iloc[idx_train], featu

scaler = StandardScaler()

x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)
```

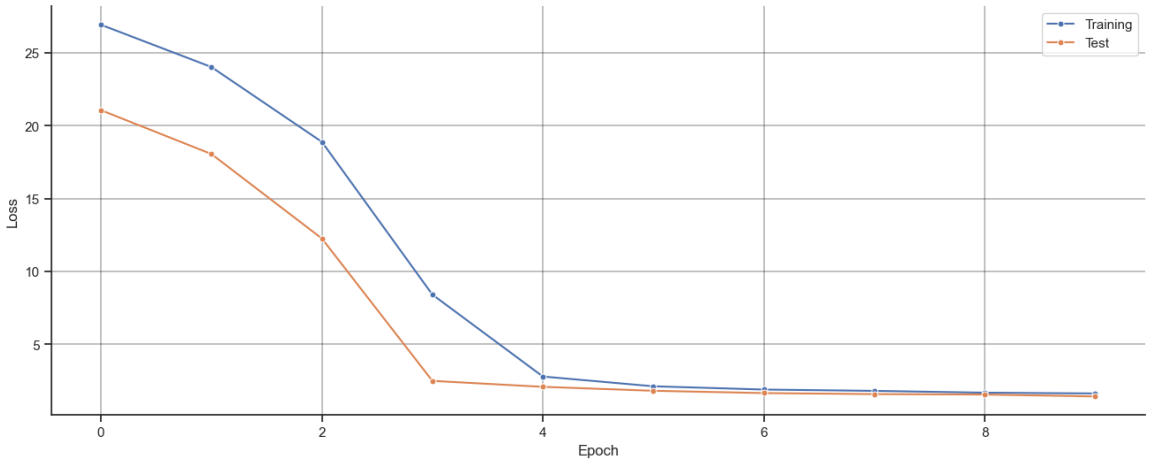
Foi desenvolvido uma rede neural simples, com 3 camadas, sendo a primeira com 40 neurônios, a segunda com 20 neurônios e a terceira com 1 neurônio.

In [ ]:
```
def create_network(len_input: int) -> models.Sequential:
    network = models.Sequential()
    network.add(layers.Dense(40, input_shape=(len_input, ), activation='re
    network.add(layers.Dense(20))
    network.add(layers.Dense(1))
    network.compile(loss="mae", optimizer='Adam', metrics=["mse"])
    return network

model = create_network(features_neural.shape[1])

history = model.fit(x_train_scaled, y_train, epochs=10, verbose=1, batch_s
                    validation_data=(x_test_scaled, y_test), shuffle=True)
pred_y = model.predict(x_test_scaled)

print(f'Algorithm: MLP')
print(f'R2 score: {r2_score(y_test, pred_y):}',)
print(f'Mean Absolute Error: {mae(y_test, pred_y):}')
```

```
Epoch 1/10
34/34 [==============================] - 1s 4ms/step - loss: 26.9361 - ms
e: 970.9310 - val_loss: 21.0739 - val_mse: 657.7845
Epoch 2/10
34/34 [==============================] - 0s 1ms/step - loss: 24.0352 - ms
e: 790.0142 - val_loss: 18.0616 - val_mse: 488.0079
Epoch 3/10
34/34 [==============================] - 0s 1ms/step - loss: 18.8948 - ms
e: 504.6594 - val_loss: 12.2615 - val_mse: 218.9705
Epoch 4/10
34/34 [==============================] - 0s 959us/step - loss: 8.4107 - m
se: 124.4849 - val_loss: 2.4868 - val_mse: 10.2620
Epoch 5/10
34/34 [==============================] - 0s 989us/step - loss: 2.7865 - m
se: 14.6543 - val_loss: 2.0788 - val_mse: 6.7128
Epoch 6/10
34/34 [==============================] - 0s 1ms/step - loss: 2.1157 - ms
e: 8.8472 - val_loss: 1.8077 - val_mse: 5.4934
Epoch 7/10
34/34 [==============================] - 0s 999us/step - loss: 1.8912 - m
se: 7.5474 - val_loss: 1.6491 - val_mse: 4.5138
Epoch 8/10
34/34 [==============================] - 0s 946us/step - loss: 1.8014 - m
se: 6.9476 - val_loss: 1.5786 - val_mse: 4.1128
Epoch 9/10
34/34 [==============================] - 0s 1ms/step - loss: 1.6764 - ms
e: 6.1520 - val_loss: 1.5488 - val_mse: 3.9874
Epoch 10/10
34/34 [==============================] - 0s 964us/step - loss: 1.6232 - m
se: 5.6623 - val_loss: 1.4192 - val_mse: 3.4249
1/1 [==============================] - 0s 43ms/step
Algorithm: MLP
R2 score: 0.9831719593196018
Mean Absolute Error: 1.4191772708072459
```

Seus resultados foram melhores que os modelos de Machine Learning.

In [ ]:
```
fig, ax = plt.subplots(figsize=(16, 6))

loss = pd.DataFrame({'training_loss': history.history["loss"], 'test_loss

epoch_count = np.arange(loss.shape[0])

sns.lineplot(data=loss, x=epoch_count, y='training_loss', label='Training
sns.lineplot(data=loss, x=epoch_count, y='test_loss', label='Test', marke
ax.legend()
ax.set(xlabel='Epoch', ylabel='Loss')
ax.grid(visible=True, color='black', linewidth=.35)
plt.show()
```



Conclusão

Dentre os modelos testados, o que apresentou o melhor resultado foi o SVR, porém o modelo de redes neurais apresentou um resultado melhor.

Deve-se levar em consideração o custo de processamento de cada modelo, pois o modelo de redes neurais é mais custoso que o modelo de SVR.

Após o periodo de estresse e testes, poderá ser decidido junto ao time de negócio qual modelo será utilizado.

Sua produção poderá ser feita em um container Docker, e utilizando algum serviço de Cloud (AWS, GCP, Azure) para sua hospedagem. Ou até mesmo em um servidor local.

Agradecimentos

- Equipe Ada (Professores e apoio)
- Gerdau
- Randstad