

## Atividade 4 - Projetos Práticos e Integração de Ferramentas (Valor 30.0)

<b>Aluno:</b>	CARLOS HENRIQUE FERREIRA	<b>Polo:</b> GUARAQUEÇABA
---------------	--------------------------	---------------------------

Olá, professor(a), estarei deixando abaixo o link do repositório onde estão todos os arquivos das atividades da Disciplina 13, e também vou deixar abaixo fotos do código para visualização do que foi feito.

Link do repositório: <https://github.com/Carlosferreiraz21/TalentoTech13>

Arquivos: atividade4.1, atividade4.2

---

**1) Crie um pipeline utilizando a biblioteca scikit-learn para prever o preço de automóveis com base nas seguintes variáveis:**

- **Categoria:** Tipo de combustível ("Gasolina", "Diesel", "Etanol").
- **Númerica:** Idade do veículo (em anos), quilometragem rodada (em quilômetros).

O pipeline deve realizar as seguintes etapas:

1. Transformar a variável categórica (Combustível) em números utilizando OneHotEncoder.
2. Padronizar as variáveis numéricas (Idade e Quilometragem) usando StandardScaler.
3. Criar um ColumnTransformer para aplicar transformações diferentes a cada tipo de dado (categórico e numérico).
4. Treinar um modelo de regressão linear para prever o preço dos automóveis.

Utilize dados fictícios para treinar o modelo e avalie o desempenho do pipeline utilizando o erro quadrático médio (MSE).

### RESPOSTA

Link

resposta: <https://github.com/Carlosferreiraz21/TalentoTech13/blob/main/atividade4.1.ipynb>

Fotos:

```
import numpy as np
import pandas as pd
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

# 1 Criando os dados fictícios
dados = pd.DataFrame({
    'Combustível': ['Gasolina', 'Diesel', 'Etanol', 'Gasolina', 'Diesel', 'Etanol', 'Gasolina', 'Diesel'],
    'Idade': [3, 5, 2, 8, 1, 4, 6, 7],
    'Quilometragem': [40000, 80000, 25000, 120000, 10000, 50000, 70000, 110000],
    'Preço': [35000, 28000, 45000, 18000, 50000, 32000, 26000, 20000]
})

# 2 Definição das colunas categóricas e numéricas
categorical_features = ['Combustível']
numerical_features = ['Idade', 'Quilometragem']

# 3 Criando os transformadores
categorical_transformer = OneHotEncoder(handle_unknown='ignore')
numerical_transformer = StandardScaler()

# 4 Criando o ColumnTransformer para processar os dados
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_transformer, numerical_features),
        ('cat', categorical_transformer, categorical_features)
    ]
)
```

```
# 5 Criando o pipeline com regressão linear
pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('model', LinearRegression())
])

# 6 Separação entre treino e teste
X = dados.drop(columns=['Preço'])
y = dados['Preço']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 7 Treinando o modelo
pipeline.fit(X_train, y_train)

# 8 Fazendo previsões no conjunto de teste
y_pred = pipeline.predict(X_test)

# 9 Avaliação do modelo
mse = mean_squared_error(y_test, y_pred)
print(f'Erro Quadrático Médio (MSE): {mse:.2f}')

# 10 Fazendo a previsão de um novo carro
novo_carro = pd.DataFrame({
    'Combustível': ['Gasolina'], # Pode ser 'Gasolina', 'Diesel' ou 'Etanol'
    'Idade': [3], # Idade do carro em anos
    'Quilometragem': [50000] # Quilometragem rodada
})

previsao_preco = pipeline.predict(novo_carro)

print(f'Previsão do preço para o novo carro: R$ {previsao_preco[0]:.2f}')

✓ 0.3s

Erro Quadrático Médio (MSE): 16338179.65
Previsão do preço para o novo carro: R$ 35790.24
```

2) Crie um programa em Python que utilize a API de previsão de preços de criptomoedas (ou qualquer API de mercado financeiro) para coletar dados históricos de preço de uma moeda específica, como Bitcoin ou Ethereum. Após salvar os dados em um arquivo CSV, implemente um modelo de aprendizado de máquina que classifique se o preço da moeda no próximo dia será maior ou menor que o do dia atual. O programa deve exibir a acurácia do modelo e permitir a visualização da importância de cada variável no modelo utilizado. Exemplos de sites com API de criptomoedas: CoinGecko API, Yahoo Finance API, Binance API, Polygon.io.

#### RESPOSTA

Link resposta:

<https://github.com/Carlosferreiraz21/TalentoTech13/blob/main/atividade4.2.ipynb>

Fotos:

```
import numpy as np
import pandas as pd
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler, MinMaxScaler
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score

# Criando dados fictícios
dados = pd.DataFrame({
    'Combustível': ['Gasolina', 'Diesel', 'Etanol', 'Gasolina', 'Diesel'],
    'Idade': [3, 5, 2, 8, 1],
    'Quilometragem': [40000, 80000, 25000, 120000, 10000],
    'Preço': [35000, 28000, 45000, 18000, 50000]
})

# Definição das colunas
categorical_features = ['Combustível']
numerical_features = ['Idade', 'Quilometragem']

# Transformadores
categorical_transformer = OneHotEncoder(handle_unknown='ignore')
numerical_transformer = MinMaxScaler() # Usando MinMaxScaler para melhor normalização
```

```

# ColumnTransformer para processar dados
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_transformer, numerical_features),
        ('cat', categorical_transformer, categorical_features)
    ]
)

# Criando o pipeline com RandomForestRegressor
pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('model', RandomForestRegressor(n_estimators=100, random_state=42))
])

# Separação entre treino e teste
X = dados.drop(columns=['Preço'])
y = dados['Preço']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Treinando o modelo
pipeline.fit(X_train, y_train)

# Fazendo previsões
y_pred = pipeline.predict(X_test)

# Avaliação
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

```

```

print(F'Erro Quadrático Médio (MSE): {mse:.2f}')
print(F'Coeficiente de Determinação (R²): {r2:.2f}')

```

✓ 0.9s

Erro Quadrático Médio (MSE): 91776400.00  
 Coeficiente de Determinação (R²): nan

```

import requests
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error

# 1 Coletando dados da API
url = "https://api.coingecko.com/api/v3/coins/bitcoin/market_chart"
params = {
    "vs_currency": "usd",
    "days": "365", # Pegamos 1 ano de dados para previsões melhores
    "interval": "daily"
}

response = requests.get(url, params=params)
data = response.json()

# 2 Criando o DataFrame
precos = pd.DataFrame(data['prices'], columns=['timestamp', 'price'])
precos['timestamp'] = pd.to_datetime(precos['timestamp'], unit='ms')

# 3 Criando novas variáveis para melhorar as previsões
precos['var_percent'] = precos['price'].pct_change() * 100 # Variação percentual do preço
precos['moving_avg_7'] = precos['price'].rolling(window=7).mean() # Média móvel de 7 dias
precos['moving_avg_14'] = precos['price'].rolling(window=14).mean() # Média móvel de 14 dias
precos['volatility'] = precos['price'].pct_change().rolling(window=7).std() * 100 # Volatilidade percentual

```

```

# Removendo valores NaN criados pelas médias móveis
precos.dropna(inplace=True)

# 4 Definição de variáveis preditoras
X = precos[['var_percent', 'moving_avg_7', 'moving_avg_14', 'volatility']]
y = precos['price']

# Divisão em treino e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 5 Criando e treinando o modelo de regressão
modelo = RandomForestRegressor(n_estimators=200, random_state=42)
modelo.fit(X_train, y_train)

# 6 Fazendo previsões para os próximos 90 dias
future_dates = pd.date_range(start=precos['timestamp'].max(), periods=90, freq='D')

# Simulando variações futuras com base no comportamento passado
future_data = pd.DataFrame({
    'var_percent': np.random.normal(X_train['var_percent'].mean(), X_train['var_percent'].std(), size=90),
    'moving_avg_7': np.random.normal(X_train['moving_avg_7'].mean(), X_train['moving_avg_7'].std(), size=90),
    'moving_avg_14': np.random.normal(X_train['moving_avg_14'].mean(), X_train['moving_avg_14'].std(), size=90),
    'volatility': np.random.normal(X_train['volatility'].mean(), X_train['volatility'].std(), size=90)
})

future_prices = modelo.predict(future_data) # Prevê os preços futuros

```

```

# Criando um DataFrame para as previsões futuras
previsoes = pd.DataFrame({'timestamp': future_dates, 'price': future_prices})

# 7 Gráfico 1: Histórico e Previsão para os próximos 90 dias
plt.figure(figsize=(12,6))
plt.plot(precos['timestamp'], precos['price'], marker='o', linestyle='-', color='blue', label='Preço Histórico')
plt.plot(previsoes['timestamp'], previsoes['price'], marker='x', linestyle='--', color='red', label='Previsão 90 dias')
plt.xlabel("Data")
plt.ylabel("Preço (USD)")
plt.title("Histórico e Previsão do Preço do Bitcoin")
plt.xticks(rotation=45)
plt.legend()
plt.grid()
plt.show()

# 8 Avaliando o modelo
y_pred = modelo.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print(f'Erro Quadrático Médio (MSE) no conjunto de teste: {mse:.2f}')

# 9 Gráfico 2: Comparação de previsões vs. valores reais
plt.figure(figsize=(10,5))
plt.plot(precos['timestamp'].iloc[-len(y_test):], y_test, marker='o', linestyle='-', color='blue', label='Real (Preço verdadeiro)')
plt.plot(precos['timestamp'].iloc[-len(y_test):], y_pred, marker='x', linestyle='--', color='red', label='Previsto pelo modelo')
plt.xlabel("Data")
plt.ylabel("Preço")
plt.title("Previsão do Modelo vs. Realidade (Dados de Teste)")
plt.legend()
plt.grid()
plt.show()

```

