Teste Técnico Cientista de Dados - A3 Data

Gabriel Bueno - 13/09/2021

Sumário:

Apresentação do Desafio:

Descrição:
Explorar a base dados "Ocorrências Aeronáuticas na Aviação Civil Brasileira" dos dados abertos do governo (https://dados.gov.br/dataset/ocorrencias-aeronauticas-da-aviacao-civil-brasileira), a fim de demonstrar as habilidades como Cientista de Dados, sendo criativo, definindo e justificando premissas e suposições, quando necessário e utilizando sua linguagem de programação preferida

Entregas:
1. Apresentação em formato PDF, que deve conter:
    a. Apresentação do desafio
    b. Explicação do processo utilizado
    c. Hipóteses levantadas
    d. Análise exploratória
    e. Conclusões e insights gerados
2. Código no GitHub (link)

Avaliação:
1. Capacidade analítica
2. Qualidade do storytelling na apresentação do problema e no código
3. Qualidade e clareza do código

Explicação do Processo Utilizado:

Ferramentas:

Conhecimento dos dados	Exploração dos Dataframes Selecionados	Solução das tarefas descritivas criadas	Solução da tarefa preditiva criada	Wrap-up
Leitura dos Arquivos; Criação dos DataFrames iniciais; Leitura e identificação das primeiras características dos arquivos. Primeira filtragem de features Primeiras ideias	Manejo de datas e horas; Cálculo de frequência; Identificação das hipóteses (IDEIAS!!) Tarefas descritivas: -> Sazonalidade; -> Distribuição de dados; Tarefas preditivas: -> Predição de fatalidades	Boxplot para check de sazonalidade; Distplot para avaliação da distribuição das frequências; Teste de normalidade e teste chi-quadrado para identificação da distribuição.	Preparação dos dataframe e limpeza de dados; Normalização; Feature Selection; Cross Validation; Tunning; Fit do modelo otimizado, predição e avaliação	Compilação de Key-Takeaways; Definição de next-steps; Upload Github; Preparação de arquivo PDF;

Conhecimento dos dados:

```
##Define função para ler vários arquivos de uma só vez, desde que estejam na pasta raiz (ou pasta do notebook)
def read(file):
    file=pd.read_csv((file+'.csv'), sep=';', header = 0, encoding = 'UTF-8')
    return(file)
```

```
##Define classe e função para ver as primeiras impressões de cada dataframe
class FirstImpression:

    def first_impression(self, df):
        self.shape = df.shape
        self.columns = df.columns
        print ('shape', self.shape)
        for coluna in self.columns:
            print (coluna, 'tem', len(set(df[coluna])), 'valores individuais com', df[coluna].count(),
                   'valores não nulos do tipo', df.dtypes[coluna])
        return (df.head())
```

```
def fatores_unicos(df, target_column):
    target = df[target_column]
    print ('Os valores únicos da coluna', target_column, 'são', (set(df[target_column])))
```

```
#Na avaliação dos dataframes, caso alguma haja suspeita de que duas colunas são iguais, elas serão avaliadas
check = df_ocor[['codigo_ocorrencia1','codigo_ocorrencia2','codigo_ocorrencia3','codigo_ocorrencia4']]
columns_to_drop=[]
for column in list(check.columns):
    equal = df_ocor['codigo_ocorrencia'].equals(check[column])
    if equal==True:
        columns_to_drop.append(column)
    print ('You can drop' if equal==True else 'Do not drop', column)
```

```
You can drop codigo_ocorrencia1
You can drop codigo_ocorrencia2
You can drop codigo_ocorrencia3
You can drop codigo_ocorrencia4
```

```
#Elinina colunas repetidas
df_ocor = df_ocor.drop(columns=columns_to_drop)
```

```
#Feature que parece ter poder preditivo
fatores_unicos(df_ocor, 'ocorrencia_classificacao')
```

```
Os valores únicos da coluna ocorrencia_classificacao são {'ACIDENTE', 'INCIDENTE GRAVE', 'INCIDENTE'}
```

```
##df_ocor, conforme previsto, possui informações interessantes que podem ser utilizadas como variáveis preditivas.
#retornaremos ao dataframe posteriormente
```

```
##df_reco são recomendações resultantes de investigação.
##Do ponto de vista empresarial, talvez seja o dataframe mais importante, que sugere um processo de melhoria contínua
##Porém, ao tratar-se de uma recomendação posterior ao evento investigado, não possui valor para predição.
##df_reco não será utilizado adiante.
```

```
##df_ftco, assim como df_reco, é uma avaliação realizada ex-post.
##Estruturalmente importante para um eventual processo de melhoria contínua, porém não possui valor para predição.
##Bastate curioso termos "ASPECTO DE PROJETO" dentre os fatores contribuíntes, pois podem sugerir ações de recall.
##df_ftco não será utilizado adiante.
```

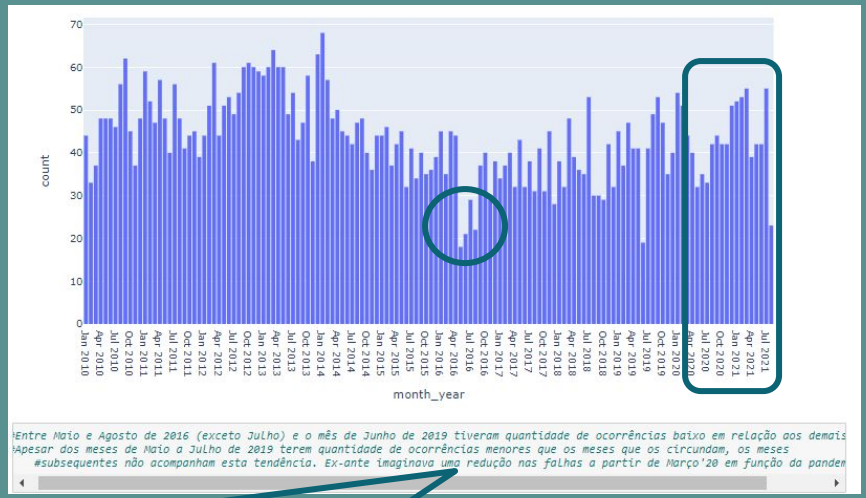
```
##df_aero possui informações interessantes que podem ser utilizadas como variáveis preditivas.
#retornaremos ao dataframe posteriormente.
```

```
##df_octp possui informações sobre a diagnose de primeiro nível da falha para todos os registros
##Apesar de extremamente importante para ações de melhoria contínua, por serem informações ex-post, não utilizaremos para predic
```

Exploração dos Dataframes Selecionados:

```python
##Define função que, em um dataframe, com uma coluna de datas raw, acrescenta colunas com data em formato datetime,
    ## mês e ano, dia, mês e ano, para posteriores análises
def dates(df, date_raw_column):
    df['date'] = pd.to_datetime(df[date_raw_column], format='%d/%m/%Y')
    df['month_year'] = df['date'].dt.strftime('%b %Y')
    df['day'] = df['date'].dt.day
    df['month'] = df['date'].dt.month
    df['year'] = df['date'].dt.year
    return(df.head())
```

```python
##Cria função para classificar horas entre períodos: manhã, tarde, noite e madrugada acrescentando coluna com estes valores ao d
def classifica_hora(df, time_raw_column):
    df['time'] = pd.to_datetime(df[time_raw_column], format='%H:%M:%S')
    df['hour'] = df['time'].dt.hour
    b = [0,6,12,18,24]
    l = ['madrugada', 'manha','tarde','noite']
    df['periodo'] = pd.cut(df['hour'], bins=b, labels=l, include_lowest=True)
    df = df.drop(columns=['time', 'hour'])
    return(df.head())
```



IDEIAS!!!
Teríamos sazonalidade em função dos meses de supostamente maior tráfego aéreo?
Tendo, a frequência de ocorrências por mês, média e mediana próximas a 44, é plausível de imaginar que este valor é irrisório quando comparado com o total de vôos no Brasil. Esta característica sugere uma distribuição de Poisson. Podemos provar esta premissa?

Hipóteses Levantadas:

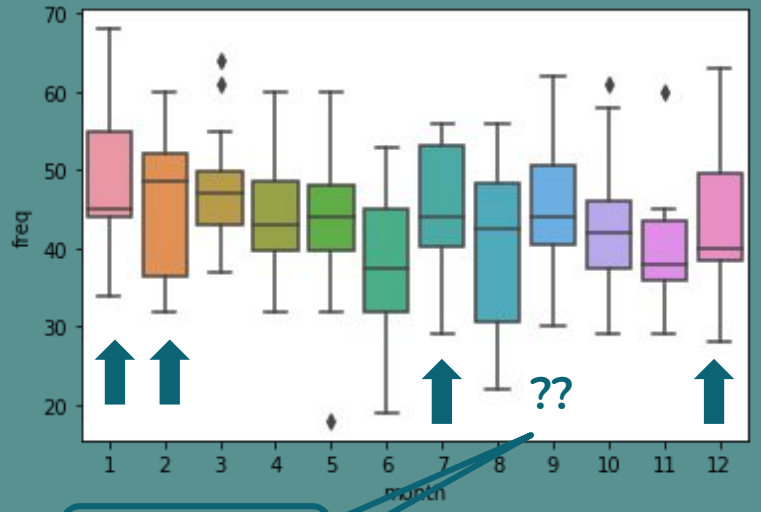
-->Teríamos sazonalidade em função dos meses de supostamente maior tráfego aéreo?

--> Tendo, a frequência de ocorrências por mês, média e mediana próximas a 44, é plausível de imaginar que este valor é irrisório quando comparado com o total de vôos no Brasil. Esta característica sugere uma distribuição de Poisson. Podemos provar esta premissa?

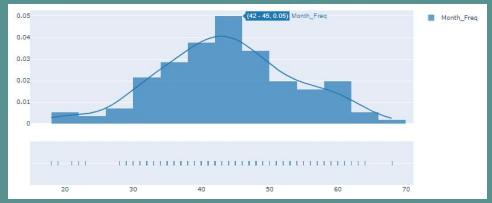
--> Conhecendo apenas condições ex-ante, em caso de ocorrência de um incidente, incidente grave ou acidente, podemos predizer a ocorrência ou não de fatalidades?

Análise Exploratória:
Solução das tarefas descritivas criadas

```
##Boxplot para entender se há efeito de sazonalidade.
##Sem o parque de total de voos para investigar uma taxa de ocorrências, observando valores absolutos,
    ##imaginava uma sazonalidade especialmente nos meses de Janeiro, Fevereiro, Julho e Dezembro.
sns.boxplot(x="month", y="freq", data=df_freq)
```




```
##Define função para avaliação de uma lista quanto à distribuição
def eval_df(df, target_column):
    target = df[target_column]
    print ('Not Gaussian :' if normaltest(target.values,)[1]<0.05 else 'Gaussian :', normaltest(target.values))
    print ('Not Poisson :' if chisquare(target.values,)[1]<0.05 else 'Poisson :', chisquare(target.values))
    print ('Mean:', round(np.mean(target),2), 'Var:', round(np.var(target),2),
           'Std:', round(np.std(target),2), 'Med:', round(np.median(target),2))
```

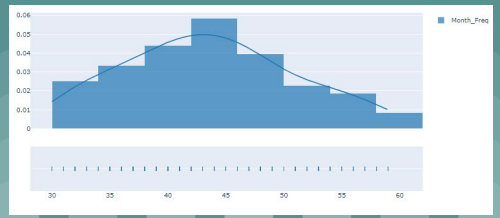


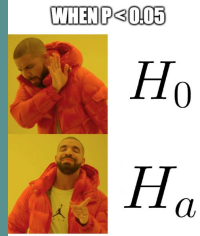
```
#Utilizando a função pré-definida sobre os dados raw, observa-se que a distribuiçao é Gaussiana porém nao se enquadra na
    #distribuição de Poisson
eval_df(df_freq,'freq')
```

```
Gaussian : NormaltestResult(statistic=0.023261194051485856, pvalue=0.9884367769151632)
Not Poisson : Power_divergenceResult(statistic=309.92541707556427, pvalue=4.6516494126962326e-15)
Mean: 43.67 Var: 96.68 Std: 9.83 Med: 44.0
```

```
#No entanto, de acordo com a documentação do teste chi-quadrado (https://docs.scipy.org/doc/scipy/reference/generated/scipy.stat
    #o teste é válido para frequências observadas acima de 5. Eliminando as frequências desta faixa, temos que a distribuição
    #passa a se enquadrar na distribuição de Poisson. Este fato pode ser ilustrado, também, pela aproximação entre variância e
    #média, que é uma característica da distribuição de Poisson.
df_mask = (df_freq['freq']>29) & (df_freq['freq']<60)
df_pois = df_freq[df_mask]
eval_df(df_pois, 'freq')
```

```
Gaussian : NormaltestResult(statistic=4.90490172337549, pvalue=0.08608235181355607)
Poisson : Power_divergenceResult(statistic=145.18516388729154, pvalue=0.05163441454749165)
Mean: 43.48 Var: 52.6 Std: 7.25 Med: 44.0
```






```
##A plotagem confirma a suspeita de sazonalidade e traz ainda a destacada incidência de ocorrências no mês de Setembro.
##Os motivos da incidência no mês de Setembro carecem de investigação posterior e não foram imaginados ex-ante.
```

Análise Exploratória:
Solução da tarefa preditiva criada

```
#Seleciona as colunas de interesse para criação de modelo que satisfaçam as premissas do problema proposto
carry_cols=['ocorrencia_classificacao','total_aeronaves_envolvidas', 'ocorrencia_saida_pista',
            'aeronave_tipo_veiculo', 'aeronave_tipo_icao', 'aeronave_motor_tipo',
            'aeronave_motor_quantidade', 'aeronave_pmd', 'aeronave_assentos', 'aeronave_ano_fabricacao',
            'aeronave_pais_fabricante', 'aeronave_registro_categoria', 'aeronave_registro_segmento', 'aeronave_fase_operacao',
            'aeronave_tipo_operacao', 'aeronave_nivel_dano','periodo', 'aeronave_fatalidades_total']
```

```
#Cria dataframe com as colunas de interesse
df_input_raw = df_full_raw[carry_cols]
```

```
#Limpeza de dados
df_input_raw.replace('***', np.nan, inplace=True)
```

```
#Conhecendo os valores únicos da coluna target
fatores_unicos(df_input_raw, 'aeronave_fatalidades_total')
```

```
Os valores únicos da coluna aeronave_fatalidades_total são {0, 1, 2, 3, 4, 5, 6, 7, 8, 10, 16}
```

```
#Transforma o target em binário, já que devemos predizer a ocorrência e não a quantidade de fatalidades
df_input_raw['target'] = df_input_raw['aeronave_fatalidades_total'].apply(classifica_fatalidades)
df_input_raw = df_input_raw.drop(columns=['aeronave_fatalidades_total'])
```

```
#Limpeza de dados
df_input_raw = df_input_raw.dropna()
```

```
#Lista as variáveis categóricas
categorical_features = list(df_input_raw.select_dtypes(include = ['object', 'category']).columns)
```

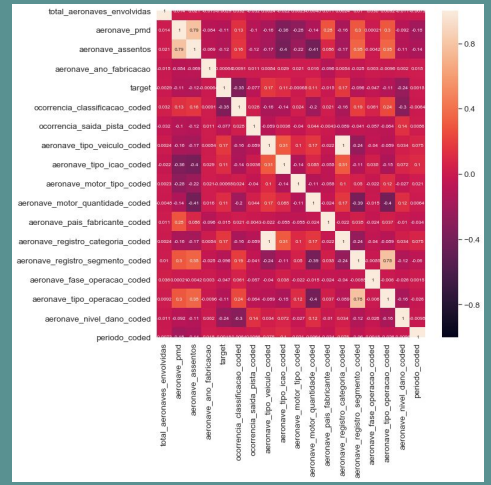
```
#Codifica as variáveis categóricas. Optei por manter o LabelEncoder em detrimento ao OneHotEncoder em função da dimensionalidade
label_encoder = LabelEncoder()
for column in categorical_features:
    df_input_raw[column+'_coded']=label_encoder.fit_transform(df_input_raw[column])
```

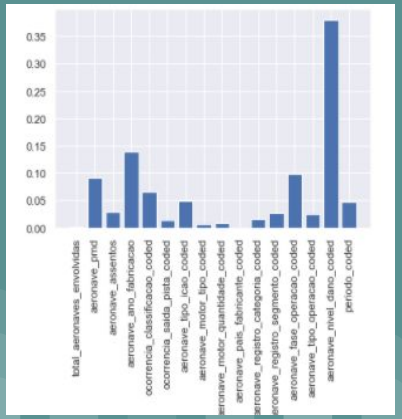
```
#Deixa apenas as variáveis já codificadas
df_input_raw = df_input_raw.drop(columns=categorical_features)
```

```
#Define quais features serão normalizadas
non_normalizable = ['target']
normalize_columns = list(set(df_input_raw.columns)-set(non_normalizable))
```

```
#Cria um dataframe dedicado à seleção de features
df_eval_feature = df_input_raw.copy()
```

```
#Aplica normalização de dados (minmaxscaler não teve bom resultado)
for col in normalize_columns:
    df_eval_feature[col]=StandardScaler().fit_transform(df_eval_feature[col].values.reshape(-1, 1))
```





	feature	VIF
0	total_aeronaves_envolvidas	1.00
1	aeronave_pmd	3.61
2	aeronave_assentos	4.05
3	aeronave_ano_fabricacao	1.01
4	ocorrencia_classificacao_coded	1.20
5	ocorrencia_saida_pista_coded	1.06
6	aeronave_tipo_icao_coded	1.50
7	aeronave_motor_tipo_coded	1.31
8	aeronave_motor_quantidade_coded	1.65
9	aeronave_pais_fabricante_coded	1.18
10	aeronave_registro_categoria_coded	1.30
11	aeronave_registro_segmento_coded	2.92
12	aeronave_fase_operacao_coded	1.02
13	aeronave_tipo_operacao_coded	2.95
14	aeronave_nivel_dano_coded	1.14
15	periodo_coded	1.04

```
#Da análise acima, as features selecionadas serão:
#'aeronave_pmd', 'aeronave_ano_fabricacao', 'ocorrencia_classificacao_coded', 'aeronave_fase_operacao_coded',
#'eronave_tipo_icao_coded','aeronave_nivel_dano_coded', 'periodo_coded',
```

Análise Exploratória:
Solução da tarefa preditiva criada

Cross Validation para selecionar melhor modelo

```
#Devido ao desbalanceamentos de diversas classes, optei por usar o StratifiedkFold
num_folds = 10
random_state = 8

strkfold = StratifiedKFold(n_splits = num_folds, shuffle = True, random_state = random_state)
```

```
#Divisão entre treino e teste utilizando o parâmetro stratify
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, test_size=0.1, random_state=random_state)
```

```
#Define os modelos a serem testados
modelos = {
    'Xgb': XGBClassifier(objective ='binary:hinge', n_estimators = 400, random_state = random_state),
    'lr': LogisticRegression(random_state = random_state, max_iter=400),
    'SVC': SVC(random_state = random_state),
    'KNN': KNeighborsClassifier(),
    'DT': DecisionTreeClassifier(random_state=random_state),
    'rfc': RandomForestClassifier(n_estimators = 400, random_state=random_state),
    'GNB': GaussianNB(),
    'Ada': AdaBoostClassifier(random_state=random_state)
}
```

```
#Define função para avaliar modelo neste script
def evaluate_model(nome, model, X, y):
    scoring=['f1', 'roc_auc', 'balanced_accuracy', 'accuracy', 'precision', 'recall']
    cv = cross_validate(model, X, y, scoring=scoring, cv=strkfold)
    cp = cross_val_predict(model, X, y, cv=strkfold)
    tn, fp, fn, tp = confusion_matrix(y, cp).ravel()
    print (nome)
    print ('Confusion Matrix: tn', tn, 'fp', fp, 'fn', fn, 'tp', tp)
    for scorer in scoring:
        print(scorer, round((np.mean(cv['test_'+scorer])),3), '+/-', round((np.std(cv['test_'+scorer])),3),'std')
```

```
#Define função para classificação de fatalidades
def classifica_fatalidades(x):
    return (0 if x==0 else 1)
```

```
xgb
Confusion Matrix: tn 4476 fp 140 fn 152 tp 200
f1 0.574 +/- 0.08 std
roc_auc 0.769 +/- 0.059 std
balanced_accuracy 0.769 +/- 0.059 std
accuracy 0.941 +/- 0.009 std
precision 0.587 +/- 0.06 std
recall 0.568 +/- 0.12 std
lr
Confusion Matrix: tn 4529 fp 87 fn 130 tp 222
f1 0.67 +/- 0.096 std
roc_auc 0.954 +/- 0.018 std
balanced_accuracy 0.806 +/- 0.057 std
accuracy 0.956 +/- 0.013 std
precision 0.722 +/- 0.101 std
recall 0.631 +/- 0.11 std
SVC
Confusion Matrix: tn 4550 fp 66 fn 136 tp 216
f1 0.68 +/- 0.093 std
roc_auc 0.933 +/- 0.02 std
balanced_accuracy 0.8 +/- 0.054 std
accuracy 0.959 +/- 0.012 std
precision 0.766 +/- 0.092 std
recall 0.614 +/- 0.103 std
KNN
Confusion Matrix: tn 4540 fp 76 fn 147 tp 205
f1 0.647 +/- 0.076 std
roc_auc 0.872 +/- 0.044 std
balanced_accuracy 0.783 +/- 0.044 std
accuracy 0.955 +/- 0.01 std
precision 0.737 +/- 0.107 std
recall 0.583 +/- 0.086 std
```

```
DT
Confusion Matrix: tn 4458 fp 158 fn 167 tp 185
f1 0.533 +/- 0.103 std
roc_auc 0.747 +/- 0.056 std
balanced_accuracy 0.746 +/- 0.054 std
accuracy 0.935 +/- 0.015 std
precision 0.545 +/- 0.115 std
recall 0.526 +/- 0.102 std
rfc
Confusion Matrix: tn 4536 fp 80 fn 136 tp 216
f1 0.665 +/- 0.097 std
roc_auc 0.951 +/- 0.019 std
balanced_accuracy 0.798 +/- 0.057 std
accuracy 0.957 +/- 0.012 std
precision 0.73 +/- 0.097 std
recall 0.614 +/- 0.111 std
GNB
Confusion Matrix: tn 3429 fp 1187 fn 2 tp 350
f1 0.371 +/- 0.015 std
roc_auc 0.936 +/- 0.012 std
balanced_accuracy 0.869 +/- 0.011 std
accuracy 0.761 +/- 0.015 std
precision 0.228 +/- 0.011 std
recall 0.994 +/- 0.011 std
Ada
Confusion Matrix: tn 4546 fp 70 fn 132 tp 220
f1 0.684 +/- 0.086 std
roc_auc 0.958 +/- 0.013 std
balanced_accuracy 0.805 +/- 0.049 std
accuracy 0.959 +/- 0.011 std
precision 0.759 +/- 0.086 std
recall 0.625 +/- 0.094 std
```

Tunning do modelo

```
#Com o modelo selecionado, utilizando o Gridsearch, realizamos o tunning
model = AdaBoostClassifier()
grid = dict()
grid['n_estimators'] = [10, 50, 100, 500]
grid['learning_rate'] = [0.0001, 0.001, 0.01, 0.1, 1.0]
grid_search = GridSearchCV(estimator=model, param_grid=grid, n_jobs=-1, cv=strkfold, scoring='f1')
grid_result = grid_search.fit(X_train, y_train)
print("Best:",(grid_result.best_score_), 'using', grid_result.best_params_)
```

```
Best: 0.6888827890910183 using {'learning_rate': 0.0001, 'n_estimators': 10}
```

Análise Exploratória:
Solução da tarefa preditiva criada

Fit do modelo otimizado, predição e avaliação

```
#Com o modelo otimizado, realizamos o fit e as predições.
#Plotamos, então, os resultados.
model_opt = AdaBoostClassifier(learning_rate= 0.0001, n_estimators= 10)
model_opt.fit(X_train, y_train)
y_predicted = model_opt.predict(X_test)
y_train_predicted = model_opt.predict(X_train)
tn, fp, fn, tp = confusion_matrix(y_test, y_predicted).ravel()
tnt, fpt, fnt, tpt = confusion_matrix(y_train, y_train_predicted).ravel()
scoring={
    'f1': f1_score,
    'roc_auc': roc_auc_score,
    'balanced_accuracy': balanced_accuracy_score,
    'accuracy': accuracy_score,
    'precision': precision_score,
    'recall': recall_score
}

print ('Score Treino')
print ('Confusion Matrix: tn', tnt, 'fp', fpt, 'fn', fnt, 'tp', tpt)
for scorer, v in scoring.items():
    print(scorer, round(v(y_train, y_train_predicted),3))
print(classification_report(y_train, y_train_predicted))
print (' ')
print (' ')
print ('Score Teste')
print ('Confusion Matrix: tn', tn, 'fp', fp, 'fn', fn, 'tp', tp)
for scorer, v in scoring.items():
    print(scorer, round(v(y_test, y_predicted),3))
print(classification_report(y_test, y_predicted))
```

```
Score Treino
Confusion Matrix: tn 4549 fp 67 fn 131 tp 221
f1 0.691
roc_auc 0.807
balanced_accuracy 0.807
accuracy 0.96
precision 0.767
recall 0.628
              precision    recall  f1-score   support

           0       0.97      0.99      0.98      4616
           1       0.77      0.63      0.69       352

    accuracy                           0.96      4968
   macro avg       0.87      0.81      0.83      4968
weighted avg       0.96      0.96      0.96      4968


Score Teste
Confusion Matrix: tn 506 fp 8 fn 16 tp 23
f1 0.657
roc_auc 0.787
balanced_accuracy 0.787
accuracy 0.957
precision 0.742
recall 0.59
              precision    recall  f1-score   support

           0       0.97      0.98      0.98       514
           1       0.74      0.59      0.66        39

    accuracy                           0.96       553
   macro avg       0.86      0.79      0.82       553
weighted avg       0.95      0.96      0.95       553
```

Algumas features e o target são severamente desbalanceados. Apesar de o feature selection e a estratégia de cross validation e separação de datasets de treino terem sido realizadas mirando minimizar o efeito do desbalanceamento, o score do modelo foi penalizado. Em um problema real, é importante pesar o custo das classes positivas e negativas para, então, melhor definir a métrica de avaliação e otimizar o modelo com foco em tal métrica

Apesar da acurácia alta do modelo, o efeito do desbalanceamento de classes pode ser visto no recall, já que a classe menos frequente é a positiva

Conclusões e insights gerados:

-->O problema pode ser melhor trabalhado, em uma situação real, conhecendo os parques para que features como quantidade de ocorrências, modelos de aeronaves, entre outros, possam ser utilizados como taxa, dando assim real dimensão da importância das features.

--> No teste optei por não utilizar outras fontes de informação, exceto conhecimento pessoal pregresso, a fim de ofertar aos avaliadores a comparação com outros testes similares eventualmente já realizados e de estimular-me a propor soluções ao problema de não haver outros dados. Então me propus a avaliar a sazonalidade das ocorrências e a testar a premissa de que as ocorrências possuem frequência baixa em relação ao parque total de voos

--> Utilizando os boxplot da frequência de ocorrências por mês e ano avaliei a suspeita ex-ante de que meses com supostamente maior quantidade de voos teriam maior frequência de ocorrências. A premissa foi confirmada. No entanto, a destacada frequência de ocorrências no mês de Setembro, não imaginada ex-ante, deve ser investigada. Quem viaja em Setembro??

--> Para confirmar a teoria de que a frequência de ocorrências era muito baixa em relação ao parque total de voos, comparei a frequência de ocorrências com a distribuição de Poisson, através de um teste chi-quadrado, que confirmou a teoria quando respeitados os limites do algoritmo de teste

--> Durante a preparação do modelo, a seleção de features pode indicar pontos de atuação no aspecto de melhoria contínua, já que os features selecionados são os que melhor se relacionam com a ocorrência de fatalidades. (ex. Ano de Fabricação, Tipo de Utilização, entre outros)

--> Algumas features e o target são severamente desbalanceadas. Apesar de o feature selection e a estratégia de cross validation e separação de datasets de treino terem sido realizadas mirando minimizar o efeito do desbalanceamento, o score do modelo foi penalizado. Em um problema real, é importante pesar o custo das classes positivas e negativas para, então, melhor definir a métrica de avaliação e otimizar o modelo com foco em tal métrica.

--> Apesar da acurácia alta do modelo, o efeito do desbalanceamento de classes pode ser visto no recall, já que a classe menos frequente era a positiva.

Next Steps:
--> Criação de um pipeline para avaliação e criação do modelo
--> Criação de dados sintéticos para melhor balanceamento de classes