Code dashboard python

```
# DASHBOARD NBA INTERACTIF
# Auteur : Cephas

import streamlit as st
import pandas as pd
import numpy as np
import plotly.express as px
import plotly.graph_objects as go
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from sklearn.linear_model import LogisticRegression, LinearRegression
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler

# --- CONFIG PAGE ---
st.set_page_config(page_title="NBA Dashboard Pro", layout="wide")

# --- STYLE ---
st.markdown("""
<style>
header {background-color: #0f172a;}
h1, h2, h3, h4 {color: #f8fafc;}
.metric-label {font-size:13px; color:#94a3b8;}
.card {background: linear-gradient(90deg,#0ea5e9,#7c3aed); color: white; padding: 12px; border-radius: 12px;}
.small {font-size:12px; color:#e2e8f0;}
```

```
</style>
""", unsafe_allow_html=True)


# --- LOAD DATA ---
@st.cache_data
def load_data():
    paths = ["final_data.csv", "/mnt/data/final_data.csv", "data/final_data.csv"]
    for p in paths:
        try:
            df = pd.read_csv(p)
            st.session_state["_data_path_used"] = p
            return df
        except Exception:
            continue
    st.error("Impossible de charger 'final_data.csv'. Mets le fichier dans le dossier du script ou ajuste le chemin.")
    st.stop()

df_raw = load_data()
df = df_raw.copy()

# Nettoyage et conversion de la colonne MIN (format mm:ss)

def convert_min_to_float(x):
    try:
        parts = str(x).split(":")
        if len(parts) != 2:
            return None
        minutes = int(parts[0])
        seconds = int(parts[1])
        return minutes + seconds / 60
```

```
    except:
        return None

# Nouvelle colonne propre
df["MIN_CLEAN"] = df["MIN"].apply(convert_min_to_float)

# Convertir en numérique au cas où + retirer les lignes invalides
df["MIN_CLEAN"] = pd.to_numeric(df["MIN_CLEAN"], errors="coerce")
df = df.dropna(subset=["MIN_CLEAN"])

# --- CLEANING / PREP ---
df = df.drop_duplicates().reset_index(drop=True)
df = df.fillna("Non renseigné")
df_num = df.apply(pd.to_numeric, errors='coerce')
numeric_cols = df_num.select_dtypes(include=[np.number]).columns.tolist()
if "RESULT" in df.columns and df["RESULT"].dtype == object:
    try:
        df["RESULT"] = pd.to_numeric(df["RESULT"], errors='coerce').fillna(0).astype(int)
        if "RESULT" not in numeric_cols:
            numeric_cols.append("RESULT")
    except:
        pass

# --- SIDEBAR FILTERS & EXPORT ---
st.sidebar.title("Filtres & Export")
seasons = ["All"] + sorted(df["SEASON"].unique().tolist())
season_sel = st.sidebar.selectbox("Saison", seasons)
teams = ["All"] + sorted(df["TEAM_NAME"].unique().tolist())
team_sel = st.sidebar.selectbox("Équipe", teams)
result_filter = st.sidebar.selectbox("Résultat", ["All", "Win (1)", "Lose (0)"])
```

```
# Appliquer filtres
df_filt = df.copy()
if season_sel != "All":
    df_filt = df_filt[df_filt["SEASON"] == season_sel]
if team_sel != "All":
    df_filt = df_filt[df_filt["TEAM_NAME"] == team_sel]
if result_filter != "All" and "RESULT" in df_filt.columns:
    val = 1 if "Win" in result_filter else 0
    df_filt = df_filt[df_filt["RESULT"] == val]

st.sidebar.markdown("---")
st.sidebar.markdown("**Téléchargement des données**")
csv_all = df.to_csv(index=False).encode('utf-8')
csv_filt = df_filt.to_csv(index=False).encode('utf-8')
st.sidebar.download_button("⬇ Télécharger tout (CSV)", data=csv_all, file_name="nba_all.csv",
mime="text/csv")
st.sidebar.download_button("⬇ Télécharger filtré (CSV)", data=csv_filt, file_name="nba_filtered.csv",
mime="text/csv")

# --- NAVIGATION ---
page = st.sidebar.radio("Navigation", [
    "Home",
    "Profil équipe",
    "Analyse multi-saisons",
    "Analyse avancée",
    "Clustering équipes",
    "Prévision simple",
    "Exploration libre",
    "Comparateur équipes",
    "Stats & Insights",
    "Analyse complète par équipe",
    "Conclusion générale" ])
```

```
# PAGE: HOME


if page == "Home":

    st.title("🏀 NBA Dashboard — Home")

    st.markdown(f"**Fichier utilisé :** `{st.session_state.get('_data_path_used','final_data.csv')}`")


    # KPIs

    col1, col2, col3, col4 = st.columns(4)

    col1.metric("Matchs totaux", int(df['GAME_ID'].nunique()) if 'GAME_ID' in df.columns else len(df))

    col2.metric("Équipes différentes", df['TEAM_NAME'].nunique() if 'TEAM_NAME' in df.columns else "N/A")

    col3.metric("Points moyen / match (global)", round(pd.to_numeric(df['PTS'], errors='coerce').mean(),2) if 'PTS' in df.columns else "N/A")

    col4.metric("Taux de victoire global (%)", round(df['RESULT'].mean()*100,2) if 'RESULT' in df.columns else "N/A")


    st.markdown("---")

    st.subheader("Top équipes (par points moyen)")

    if 'PTS' in df.columns:

        top_pts = df.groupby("TEAM_NAME")["PTS"].mean().sort_values(ascending=False).head(10)

        fig = px.bar(top_pts, title="Top 10 équipes par points moyens", labels={'value':'PTS moyen','index':'Équipe'})

        st.plotly_chart(fig, use_container_width=True)


    st.markdown("---")

    st.subheader("Aperçu des données (50 premières lignes)")

    st.dataframe(df.head(50), use_container_width=True)


# PAGE: Profil équipe


elif page == "Profil équipe":
```

```
    st.title("📊 Profil équipe avancé")
    team_list = sorted(df["TEAM_NAME"].unique().tolist())
    team = st.selectbox("Choisir une équipe", team_list)
    df_team = df[df["TEAM_NAME"]==team].copy()
    df_team_num = df_team.apply(pd.to_numeric, errors='coerce')

    # Stats moyennes
    st.subheader("Résumé moyen")
    summary = df_team_num.mean().round(2)
    st.table(summary[numeric_cols].to_frame("Moyenne"))

    # Evolution match par match
    st.subheader("Évolution match par match")
    stat_evo = st.multiselect("Choisir stats à visualiser", ["PTS","REB","AST","STL","BLK"], default=["PTS","REB","AST"])
    if stat_evo:
        fig = px.line(df_team, x="GAME_ID", y=stat_evo, title=f"Évolution {', '.join(stat_evo)}", markers=True)
        st.plotly_chart(fig, use_container_width=True)

# PAGE: Analyse multi-saisons

elif page == "Analyse multi-saisons":
    st.title("📈 Analyse temporelle / multi-saisons")
    team_list = sorted(df["TEAM_NAME"].unique().tolist())
    team = st.selectbox("Choisir une équipe", team_list)
    df_team = df[df["TEAM_NAME"]==team].copy()
    df_team_num = df_team.apply(pd.to_numeric, errors='coerce')
    stat = st.selectbox("Statistique à suivre", ["PTS","REB","AST","FG_PCT","EFG_PCT"], index=0)
    fig = px.line(df_team, x="SEASON", y=stat, markers=True, title=f"Évolution de {stat} par saison")
    st.plotly_chart(fig, use_container_width=True)
```

```
# PAGE: Analyse avancée

elif page == "Analyse avancée":
    st.title("🧠 Analyse avancée NBA")
    st.markdown("Calcul de statistiques avancées (TS%, eFG%, Offensive/Defensive Rating...)")
    df_adv = df_num.copy()
    # True Shooting %
    if set(["PTS","FGA","FTA"]).issubset(df_adv.columns):
        df_adv["TS_PCT"] = df_adv["PTS"] / (2*(df_adv["FGA"] + 0.44*df_adv["FTA"]))
    # eFG%
    if set(["FGM","FG3M","FGA"]).issubset(df_adv.columns):
        df_adv["eFG_PCT"] = (df_adv["FGM"] + 0.5*df_adv["FG3M"]) / df_adv["FGA"]
    # FT Rate
    if set(["FTM","FGA"]).issubset(df_adv.columns):
        df_adv["FT_RATE"] = df_adv["FTM"]/df_adv["FGA"]

    st.subheader("Exemple : Top 10 équipes TS%")
    if "TS_PCT" in df_adv.columns:
        top_ts =
df_adv.groupby("TEAM_NAME")["TS_PCT"].mean().sort_values(ascending=False).head(10)
        fig = px.bar(top_ts, title="Top 10 équipes TS%", labels={'value':'TS%','index':'Équipe'})
        st.plotly_chart(fig, use_container_width=True)

# PAGE: Clustering équipes

elif page == "Clustering équipes":
    st.title("🤖 Clustering équipes (KMeans + PCA)")
    cluster_cols = st.multiselect("Colonnes pour clustering", numeric_cols,
default=["PTS","REB","AST","STL","BLK"])
    if len(cluster_cols)>=2:
        scaler = StandardScaler()
        X_scaled = scaler.fit_transform(df_num[cluster_cols].fillna(0))
```

```
        pca = PCA(n_components=2)
        X_pca = pca.fit_transform(X_scaled)
        k = st.slider("Nombre de clusters", 2, 10, 4)
        km = KMeans(n_clusters=k, random_state=42)
        labels = km.fit_predict(X_scaled)
        df_plot = pd.DataFrame(X_pca, columns=["PC1","PC2"])
        df_plot["TEAM_NAME"] = df["TEAM_NAME"]
        df_plot["Cluster"] = labels.astype(str)
        fig = px.scatter(df_plot, x="PC1", y="PC2", color="Cluster", hover_data=["TEAM_NAME"])
        st.plotly_chart(fig, use_container_width=True)

# PAGE: Prévision simple

elif page == "Prévision simple":
    st.title("📊 Prévision simple")
    if "RESULT" in df_num.columns:
        st.subheader("Logistic Regression : victoire/défaite")
        feat_cols = st.multiselect("Colonnes pour prédiction", numeric_cols,
default=["PTS","REB","AST"])
        if feat_cols:
            X = df_num[feat_cols].fillna(0)
            y = df_num["RESULT"]
            model = LogisticRegression(max_iter=1000)
            model.fit(X,y)
            y_pred = model.predict(X)
            cm = confusion_matrix(y, y_pred)
            st.write("Matrice de confusion")
            fig, ax = plt.subplots()
            sns.heatmap(cm, annot=True, fmt='d', cmap="Blues", ax=ax)
            st.pyplot(fig)
            st.write("Importance features (coefficients)")
```

```
        st.table(pd.Series(model.coef_[0], index=feat_cols).sort_values(ascending=False))


# PAGE: Exploration libre


elif page == "Exploration libre":
    st.title("🔍 Exploration libre")
    cols = df.columns.tolist()
    x = st.selectbox("Axe X", cols, index=cols.index("PTS") if "PTS" in cols else 0)
    y = st.selectbox("Axe Y", cols, index=cols.index("AST") if "AST" in cols else 1)
    color = st.selectbox("Coloriser par", ["None"] + cols, index=0)
    size_choice = st.selectbox("Taille (optionnelle)", ["None"] + numeric_cols, index=0)
    plot_df = df.copy()
    try:
        plot_df[x] = pd.to_numeric(plot_df[x], errors='coerce')
        plot_df[y] = pd.to_numeric(plot_df[y], errors='coerce')
    except: pass
    if color=="None":
        fig = px.scatter(plot_df, x=x, y=y, size=size_choice if size_choice!="None" else None,
hover_data=["TEAM_NAME","GAME_ID"])
    else:
        fig = px.scatter(plot_df, x=x, y=y, color=color, size=size_choice if size_choice!="None" else None,
hover_data=["TEAM_NAME","GAME_ID"])
    st.plotly_chart(fig, use_container_width=True)


# PAGE: Comparateur équipes


elif page == "Comparateur équipes":
    st.title("⚖️ Comparateur Team A vs Team B")
    team_list = sorted(df["TEAM_NAME"].unique().tolist())
    team_a = st.selectbox("Equipe A", team_list, index=0)
    team_b = st.selectbox("Equipe B", team_list, index=1 if len(team_list)>1 else 0)
```

```
stats_for_radar = ["PTS","REB","AST","STL","BLK","TO","FG_PCT","EFG_PCT"]
stats_for_radar = [s for s in stats_for_radar if s in df.columns and s in numeric_cols]

def team_summary(team):
    d = df[df["TEAM_NAME"]==team]
    summary = {s: pd.to_numeric(d[s], errors='coerce').mean() for s in stats_for_radar}
    summary["WIN_RATE"] = d["RESULT"].mean()*100 if "RESULT" in d.columns else np.nan
    return summary

sum_a = team_summary(team_a)
sum_b = team_summary(team_b)

st.subheader("Résumé numérique")
colA, colB = st.columns(2)
colA.write(f"### {team_a}")
colA.table(pd.DataFrame.from_dict(sum_a, orient='index', columns=[team_a]).round(2))
colB.write(f"### {team_b}")
colB.table(pd.DataFrame.from_dict(sum_b, orient='index', columns=[team_b]).round(2))

if len(stats_for_radar)>0:
    categories = stats_for_radar
    a_vals = [sum_a[c] for c in categories]
    b_vals = [sum_b[c] for c in categories]
    def normalize(vals):
        arr = np.nan_to_num(np.array(vals, dtype=float))
        return (arr - arr.min())/(arr.max()-arr.min()) if arr.max()!=arr.min() else np.zeros_like(arr)
    a_norm = normalize(a_vals)
    b_norm = normalize(b_vals)
    fig = go.Figure()
    fig.add_trace(go.Scatterpolar(r=a_norm, theta=categories, fill='toself', name=team_a))
    fig.add_trace(go.Scatterpolar(r=b_norm, theta=categories, fill='toself', name=team_b))
```

```
        fig.update_layout(polar=dict(radialaxis=dict(visible=True, range=[0,1])), showlegend=True,
                          title="Comparaison radar (normalisé)")
        st.plotly_chart(fig, use_container_width=True)

# PAGE: Stats & Insights

elif page == "Stats & Insights":
    st.title("🧠 Stats automatiques & Insights")
    if len(numeric_cols)>=2:
        corr = df_num[numeric_cols].corr()
        st.subheader("Heatmap des corrélations (valeurs absolues)")
        figc = px.imshow(corr.abs(), title="Matrice de corrélation (abs)")
        st.plotly_chart(figc, use_container_width=True)
    st.markdown("---")
    st.subheader("Top équipes automatiques")
    if 'PTS' in df.columns:
        st.table(df.groupby("TEAM_NAME")["PTS"].mean().sort_values(ascending=False).head(5).to_frame("PTS moyen"))
    if "RESULT" in df.columns:
        st.table((df.groupby("TEAM_NAME")["RESULT"].mean()*100).sort_values(ascending=False).head(5).to_frame("Win %"))

    st.markdown("---")
    st.subheader("Résumé automatique (prêt pour rapport)")
    summary_lines = []
    if 'PTS' in df.columns and 'RESULT' in df.columns:
        corr_pts_res = df_num[['PTS','RESULT']].corr().iloc[0,1]
        summary_lines.append(f"- Corrélation entre points et victoire : {corr_pts_res:.2f}")
    if 'TO' in df.columns and 'RESULT' in df.columns:
        corr_to_res = df_num[['TO','RESULT']].corr().iloc[0,1]
        summary_lines.append(f"- Corrélation entre turnovers (TO) et victoire : {corr_to_res:.2f}")
```

```
    summary_lines.append("- Les équipes avec un PTS moyen élevé et un FG_PCT élevé dominent généralement.")
    summary_lines.append("- Les turnovers élevés réduisent les chances de victoire.")
    st.write("\n".join(summary_lines))

# Analyse complète par équipe
elif page == "Analyse complète par équipe":
    st.header("🏀 Analyse complète par équipe — Interprétations au clic")

    # Liste des équipes
    equipes = sorted(df["TEAM_NAME"].unique().tolist())
    equipe_analyse = st.selectbox("Choisis une équipe :", equipes)

    # Filtrage
    df_team = df[df["TEAM_NAME"] == equipe_analyse]

    if df_team.empty:
        st.warning("Aucune donnée trouvée pour cette équipe ❌")
        st.stop()

    st.subheader(f"📌 Analyse détaillée : {equipe_analyse}")

    # Initialisation du clic
    if "graph_clicked" not in st.session_state:
        st.session_state["graph_clicked"] = None

    # LISTE DES GRAPHIQUES DISPONIBLES

    graphiques = {
        "pts_avg": {
            "title": "Évolution des points par match",
```

```
    "type": "line",
    "x": "GAME_ID",
    "y": "PTS",
    "plot": lambda: px.line(df_team, x="GAME_ID", y="PTS")
},
"rebonds": {
    "title": "Rebonds par match",
    "type": "bar",
    "x": "GAME_ID",
    "y": "REB",
    "plot": lambda: px.bar(df_team, x="GAME_ID", y="REB")
},
"passes": {
    "title": "Passes décisives",
    "type": "line",
    "x": "GAME_ID",
    "y": "AST",
    "plot": lambda: px.line(df_team, x="GAME_ID", y="AST")
},
"scatter_min_pts": {
    "title": "Relation Minutes vs Points",
    "type": "scatter",
    "x": "MIN",
    "y": "PTS",
    "plot": lambda: px.scatter(df_team, x="MIN_CLEAN", y="PTS", trendline="ols")

},
"dist_pts": {
    "title": "Distribution des Points",
    "type": "distribution",
    "y": "PTS",
```

```
        "plot": lambda: px.histogram(df_team, x="PTS", nbins=20)
    },
}

# ======================================================================
# AFFICHAGE DES GRAPHS + BOUTON INTERPRÉTATION
# ======================================================================
cols = st.columns(2)

for idx, (key, info) in enumerate(graphiques.items()):
    col = cols[idx % 2]

    with col:
        st.markdown(f"### {info['title']}")
        fig = info["plot"]()
        st.plotly_chart(fig, use_container_width=True)

        if st.button(f" 📝 Voir l'interprétation", key=f"btn_{key}"):
            st.session_state["graph_clicked"] = key

# ======================================================================
# AFFICHAGE DE L'INTERPRÉTATION
# ======================================================================
st.markdown("---")
st.subheader(f" 📘 Interprétation du graphique sélectionné : {equipe_analyse}")

clicked = st.session_state["graph_clicked"]

if clicked:
    g = graphiques[clicked]
    texte = interprete_graphique(
```

```
            df_team,
            g["type"],
            x=g.get("x"),
            y=g.get("y"),
        )
        st.markdown(texte)
        export_txt(texte, filename=f"interpretation_{clicked}.txt")

    else:
        st.info("Clique sur un graphique pour afficher son interprétation 📊")


#Conclusion générale
elif page == "Conclusion générale":

    st.title("Conclusion générale")

    st.markdown("""
    Cette étude a porté sur l’analyse d’un jeu de données NBA composé de plus de **31 000 observations**,
    décrivant les performances des équipes match par match sur plusieurs saisons.
    L’objectif principal était d’identifier les facteurs clés expliquant la performance collective
    et le résultat des rencontres à l’aide d’outils statistiques et analytiques.

    L’analyse descriptive a mis en évidence une **variabilité importante des performances** entre les équipes,
    aussi bien sur le plan offensif que défensif. Les statistiques classiques telles que les points marqués,
    les passes décisives et les rebonds constituent des indicateurs essentiels, mais ne suffisent pas à elles seules
    pour caractériser la performance globale.
```

L’introduction de statistiques avancées, notamment le **Effective Field Goal Percentage (EFG_PCT)**,

le **Player Impact Estimate (PIE)** et le **plus/minus**, a permis d’obtenir une vision plus synthétique

et plus pertinente de l’efficacité réelle des équipes. Ces indicateurs présentent une relation forte

avec le résultat des matchs, confirmant leur capacité à résumer l’impact collectif sur le jeu.

Les analyses multivariées, en particulier l’Analyse en Composantes Principales (ACP), ont permis de

réduire la dimension du jeu de données tout en conservant l’essentiel de l’information.

Elles ont mis en évidence des axes opposant efficacité offensive, intensité défensive et discipline de jeu,

facilitant ainsi l’interprétation globale des performances.

Enfin, les méthodes de classification ont permis d’identifier plusieurs **profils d’équipes distincts**,

allant d’équipes très performantes et régulières à des équipes plus irrégulières.

Ces résultats montrent que la victoire repose sur un **équilibre entre efficacité offensive, organisation collective

et maîtrise des pertes de balle**, plutôt que sur un seul indicateur isolé.

En conclusion, cette étude confirme que la performance en NBA est un phénomène multidimensionnel,

explicable par une combinaison cohérente de statistiques traditionnelles et avancées.

Les résultats obtenus constituent une base solide pour la **prise de décision sportive**,

l’évaluation des équipes et le développement de modèles prédictifs des résultats futurs.

""")

```
# ================================================================
# Fonction d'interprétation automatique
# ================================================================

def interprete_graphique(df, graphique_type, x=None, y=None, group=None):
    texte = ""
```

```
if graphique_type == "distribution":
    mean_val = df[y].mean()
    median_val = df[y].median()
    std_val = df[y].std()

    texte = (
        f"La distribution de **{y}** montre une moyenne de **{mean_val:.2f}**, "
        f"une médiane de **{median_val:.2f}**, et un écart-type de **{std_val:.2f}**. "
        f"La dispersion est donc "
        f"{'forte' if std_val > mean_val * 0.5 else 'faible'}, révélant la variabilité des performances."
    )

elif graphique_type == "line":
    start = df[y].iloc[0]
    end = df[y].iloc[-1]
    trend = "une hausse 🔼" if end > start else "une baisse 🔽"

    texte = (
        f"Le graphique temporel de **{y}** montre {trend} entre le début et la fin de la période. "
        "Les variations reflètent les performances fluctuantes de l’équipe."
    )

elif graphique_type == "bar":
    group_means = df.groupby(x)[y].mean()
    max_cat = group_means.idxmax()
    max_val = group_means.max()

    texte = (
        f"Le graphique en barres montre que **{max_cat}** affiche la valeur moyenne la plus haute "
        f"avec **{max_val:.2f}** pour **{y}**."
```

```
        )

    elif graphique_type == "scatter":
        corr = df[x].corr(df[y])
        texte = (
            f"Le scatterplot entre **{x}** et **{y}** montre une corrélation de **{corr:.2f}**. "
            f"Il s'agit d'une corrélation "
            f"{'forte' if abs(corr) > 0.6 else 'faible à modérée'}."
        )

    return texte

# Fonction d’export .TXT

def export_txt(text, filename="interpretation.txt"):
    st.download_button(
        label="📥 Télécharger l'interprétation (.txt)",
        data=text,
        file_name=filename,
        mime="text/plain"
    )
```