# 

# 

# 

# 

# 

# 

# **Data Science Application**

# **Haritha Thipparapu**

**Srija Velumula**

**Snigdha Yalam**

# **Data Science Dashboard**

An interactive and intelligent web app for exploring datasets, visualizing trends, performing regression modeling, and making real-time predictions — built using Streamlit.

**Features**

* **Upload Your Own Dataset** (.csv format)
* **Automated Insight Summary**: Identifies the best target variable and top predictive features based on correlation.
* **Interactive Visualizations**:
  + Histograms.
  + Correlation heatmap.
* **Statistical Summary**: Basic descriptive stats (mean, std, etc.)
* **Model Builder**:
  + Choose features and target.
  + Trains a Linear Regression model.
  + Shows R².
  + Plots actual vs predicted values.
  + Displays feature importances.
* **Custom Prediction Panel**:
  + Use sliders to simulate new inputs.
  + Predict target values instantly.

**Tech Stack**

* [Streamlit](https://streamlit.io/)
* [Pandas](https://pandas.pydata.org/)
* [Scikit-learn](https://scikit-learn.org/)
* [Plotly](https://plotly.com/)
* [NumPy](https://numpy.org/)

**Code:**

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.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

from sklearn.preprocessing import StandardScaler

import matplotlib.pyplot as plt

from scipy.stats import zscore

# Set page config

st.set\_page\_config(

page\_title="Data Science Dashboard",

page\_icon="📊",

layout="wide"

)

# Title and description

st.title(":bar\_chart: Data Science Dashboard")

st.markdown("""

This dashboard allows you to:

- Upload and analyze your datasets

- Visualize data with interactive charts

- Perform basic statistical analysis

- Make predictions using machine learning models

- Get feature suggestions for more accurate predictions

- Interact with the model using custom inputs

""")

# File upload

uploaded\_file = st.file\_uploader("Upload your dataset (CSV file)", type=['csv'])

if uploaded\_file is not None:

df = pd.read\_csv(uploaded\_file)

# Display basic information

st.subheader("Dataset Overview")

col1, col2 = st.columns(2)

with col1:

st.write("Dataset Shape:", df.shape)

st.write("Columns:", list(df.columns))

with col2:

st.write("Data Types:")

st.write(df.dtypes)

# Data Preview

st.subheader("Data Preview")

st.dataframe(df.head())

# Statistical Analysis

st.subheader("Statistical Analysis")

st.write(df.describe())

# Insight Generator

st.subheader("Automated Insight Summary")

numeric\_cols = df.select\_dtypes(include=['int64', 'float64']).columns

if len(numeric\_cols) > 1:

corr = df[numeric\_cols].corr()

target = corr.abs().mean().sort\_values(ascending=False).index[0]

top\_corr = corr[target].drop(target).sort\_values(key=abs, ascending=False).head(3)

st.markdown(f"- The most predictable target is \*\*{target}\*\*.")

for feature, value in top\_corr.items():

direction = "positively" if value > 0 else "negatively"

st.markdown(f"- `{feature}` is \*\*{direction} correlated\*\* with `{target}` (correlation = {value:.2f})")

# Visualization

st.subheader("Data Visualization")

if len(numeric\_cols) > 0:

selected\_col = st.selectbox("Select a column to visualize:", numeric\_cols)

fig = px.histogram(df, x=selected\_col, title=f"Distribution of {selected\_col}")

st.plotly\_chart(fig)

if len(numeric\_cols) > 1:

st.subheader("Correlation Heatmap")

fig = px.imshow(corr, title="Correlation Heatmap")

st.plotly\_chart(fig)

st.subheader("Suggested Targets and Predictors")

avg\_corr = corr.abs().mean().sort\_values(ascending=False)

suggested\_target = avg\_corr.index[0]

top\_predictors = corr[suggested\_target].drop(suggested\_target).abs().sort\_values(ascending=False).head(3)

st.info(f"Suggested target variable: \*\*{suggested\_target}\*\*")

st.write(f"Top predictive features: {list(top\_predictors.index)}")

high\_corr\_pairs = corr.where(np.triu(np.ones(corr.shape), k=1).astype(bool)).stack().sort\_values(ascending=False)

if not high\_corr\_pairs.empty and high\_corr\_pairs.iloc[0] > 0.9:

st.warning("Highly correlated features detected:")

st.write(high\_corr\_pairs[high\_corr\_pairs > 0.9])

# Simple Prediction Model

st.subheader("Simple Prediction Model")

if len(numeric\_cols) > 1:

feature\_cols = st.multiselect("Select features for prediction:", numeric\_cols)

target\_col = st.selectbox("Select target variable:", numeric\_cols)

if feature\_cols and target\_col and target\_col not in feature\_cols:

X = df[feature\_cols]

y = df[target\_col]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

model = LinearRegression()

model.fit(X\_train\_scaled, y\_train)

y\_pred = model.predict(X\_test\_scaled)

r2 = model.score(X\_test\_scaled, y\_test)

st.write(f"R² Score: {r2:.3f}")

if r2 < 0.5:

st.error("Low model performance. Try different features or another target.")

fig = go.Figure()

fig.add\_trace(go.Scatter(x=y\_test, y=y\_pred, mode='markers', name='Predictions'))

fig.add\_trace(go.Scatter(x=[y\_test.min(), y\_test.max()], y=[y\_test.min(), y\_test.max()],

mode='lines', name='Perfect Prediction'))

fig.update\_layout(title="Actual vs Predicted Values", xaxis\_title="Actual Values", yaxis\_title="Predicted Values")

st.plotly\_chart(fig)

importance = pd.DataFrame({'Feature': feature\_cols, 'Importance': abs(model.coef\_)})

importance = importance.sort\_values('Importance', ascending=False)

fig = px.bar(importance, x='Feature', y='Importance', title="Feature Importance")

st.plotly\_chart(fig)

# Real-time input prediction

st.subheader("🔍 Try Prediction with Custom Input")

user\_input = []

for feature in feature\_cols:

val = st.slider(f"{feature}:", float(df[feature].min()), float(df[feature].max()), float(df[feature].mean()))

user\_input.append(val)

user\_input\_scaled = scaler.transform([user\_input])

custom\_pred = model.predict(user\_input\_scaled)[0]

st.success(f"Predicted {target\_col}: {custom\_pred:.2f}")

else:

st.warning("Please upload a dataset with at least two numeric columns for prediction.")

else:

st.info("Please upload a CSV file to begin analysis.")

**References**

* Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, E. (2011). *Scikit-learn: Machine Learning in Python*. Journal of Machine Learning Research, 12, 2825-2830.
* Hunter, J. D. (2007). *Matplotlib: A 2D graphics environment*. Computing in Science & Engineering, 9(3), 90-95.
* McKinney, W. (2010). *Data structures for statistical computing in Python*. Proceedings of the 9th Python in Science Conference, 51-56.
* Plotly Technologies Inc. (2015). *Collaborative data science*. Montreal, QC: Plotly Technologies Inc.
* Streamlit Inc. (2024). *Streamlit documentation*. Retrieved from <https://docs.streamlit.io/>