```
# Save this as app.py import streamlit as st import pandas as pd import numpy as np import
torch import plotly.express as px import plotly.graph_objects as go from plotly.subplots import
make_subplots import os from PIL import Image import pickle import json from pytorch_lightning
import LightningModule import torch.nn as nn import io # Configuration constants - Update
these paths to match your file system MODEL_PATH = './models/business-success-final.ckpt' #
Path to the CKPT file RESULTS_PATH = './results' # Directory where results are stored
DATA_PATH = './data/sf_business_with_news.parquet' # Path to data file # Import
preprocessing libraries (needed to recreate the preprocessor) from sklearn.preprocessing import
StandardScaler, OneHotEncoder from sklearn.compose import ColumnTransformer from
sklearn.pipeline import Pipeline # Function to create a preprocessor matching the one used in
training def create_preprocessor(df): """Create preprocessing pipeline matching the one used
during model training""" # Identify numerical and categorical features (similar to the training
code) try: # Selected features based on the training code selected_features = [ # Spatial
features (critical for site selection) 'latitude', 'longitude', # Neighborhood information
'neighborhoods_analysis_boundaries', 'neighborhood_business_count', # Business
characteristics 'business_industry', 'similar_businesses_count', # Temporal features 'start_year',
# Economic indicators 'sf_gdp', 'sf_unemployment_rate', 'sf_house_price_index', # Safety
factors 'high_crime_area', 'district_total_crimes', # Other relevant features
'overall_sentiment_mean' ] # Keep only features that exist in the dataframe available_features =
[f for f in selected_features if f in df.columns] # If we don't have enough columns, use what's
available if len(available_features) < 2: numerical_features = df.select_dtypes(include=['int64',
'float64']).columns.tolist() categorical_features = df.select_dtypes(include=['object',
'category']).columns.tolist() else: # Use the available selected features X =
df[available_features].copy() numerical_features = X.select_dtypes(include=['int64',
'float64']).columns.tolist() categorical_features = X.select_dtypes(include=['object',
'category']).columns.tolist() # Create the same preprocessing pipeline as in training
numerical_transformer = Pipeline(steps=[ ('scaler', StandardScaler()) ]) categorical_transformer
= Pipeline(steps=[ ('onehot', OneHotEncoder(handle_unknown='ignore', sparse_output=False))
]) preprocessor = ColumnTransformer( transformers=[ ('num', numerical_transformer,
numerical_features), ('cat', categorical_transformer, categorical_features) ] ) # Fit the
preprocessor with sample data preprocessor.fit(df) print(f"Created preprocessor with
{len(numerical_features)} numerical features and {len(categorical_features)} categorical
features") return preprocessor, True except Exception as e: print(f"Error creating preprocessor:
{e}") return None, False # Define model architecture (matches the saved model) class
BusinessSuccessModel(LightningModule): def __init__(self, input_dim, spatial_idx=[0, 1],
temporal_idx=[2], dropout_rate=0.3, l2_weight=1e-4): super().__init__()
self.save_hyperparameters() self.input_dim = input_dim self.spatial_indices = spatial_idx
self.temporal_indices = temporal_idx self.l2_weight = l2_weight # Number of features for each
component self.num_spatial = len(spatial_idx) self.num_temporal = len(temporal_idx)
self.num_other = input_dim - self.num_spatial - self.num_temporal # CNN component for spatial
features self.spatial_net = nn.Sequential( nn.Linear(self.num_spatial, 32), nn.ReLU(),
nn.BatchNorm1d(32), nn.Dropout(dropout_rate/2), nn.Linear(32, 64), nn.ReLU(),
nn.BatchNorm1d(64) ) # LSTM component for temporal features self.lstm_hidden_size = 32
self.lstm = nn.LSTM( input_size=1, hidden_size=self.lstm_hidden_size, num_layers=1,
batch_first=True ) # Process other features self.other_net = nn.Sequential(
nn.Linear(self.num_other, 128), nn.ReLU(), nn.BatchNorm1d(128), nn.Dropout(dropout_rate/2) )
if self.num_other > 0 else None # Combined network combined_size = 64 +
self.lstm_hidden_size if self.num_other > 0: combined_size += 128 self.combined_net =
nn.Sequential( nn.Linear(combined_size, 128), nn.ReLU(), nn.BatchNorm1d(128),
nn.Dropout(dropout_rate), nn.Linear(128, 64), nn.ReLU(), nn.BatchNorm1d(64),
nn.Dropout(dropout_rate), nn.Linear(64, 1), nn.Sigmoid() ) def forward(self, x): batch_size =
x.size(0) # Split features spatial_features = x[:, self.spatial_indices] # Process spatial features
with CNN spatial_output = self.spatial_net(spatial_features) # Process temporal features with
LSTM if they exist if self.num_temporal > 0: temporal_features = x[:,
self.temporal_indices].unsqueeze(2) lstm_out, _ = self.lstm(temporal_features) lstm_output =
lstm_out[:, -1, :] else: lstm_output = torch.zeros(batch_size, self.lstm_hidden_size).to(x.device)
```

```
# Process other features if they exist if self.num_other > 0: other_indices = [i for i in
range(x.size(1)) if i not in self.spatial_indices and i not in self.temporal_indices] other_features =
x[:, other_indices] other_output = self.other_net(other_features) # Combine all outputs combined
= torch.cat([spatial_output, lstm_output, other_output], dim=1) else: combined =
torch.cat([spatial_output, lstm_output], dim=1) # Final prediction output =
self.combined_net(combined) return output.squeeze() def configure_optimizers(self): optimizer
= torch.optim.Adam( self.parameters(), lr=0.001, weight_decay=self.l2_weight # L2
regularization ) scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau( optimizer,
mode='min', factor=0.5, patience=3, verbose=True ) return { "optimizer": optimizer,
"lr_scheduler": scheduler, "monitor": "train_loss" } # Helper function to load model and data
@st.cache_resource def load_model_components(): try: # Load data to help create the
preprocessor if os.path.exists(DATA_PATH): print(f"Loading data from {DATA_PATH}") df =
pd.read_parquet(DATA_PATH) # Ensure long_term_success column exists if
'long_term_success' not in df.columns: df['long_term_success'] = (df['business_age_years'] >=
5.0).astype(int) else: print(f"Data file not found at {DATA_PATH}, creating sample data") #
Create sample data to help initialize the preprocessor df = create_sample_data() # Create or
load preprocessor preprocessor_path = os.path.join(RESULTS_PATH, 'preprocessor.pkl') if
os.path.exists(preprocessor_path): print(f"Loading preprocessor from {preprocessor_path}") with
open(preprocessor_path, 'rb') as f: preprocessor = pickle.load(f) preprocessor_success = True
else: print(f"Preprocessor not found at {preprocessor_path}, creating one") preprocessor,
preprocessor_success = create_preprocessor(df) # Save the created preprocessor if
preprocessor_success and not os.path.exists(RESULTS_PATH): os.makedirs(RESULTS_PATH,
exist_ok=True) if preprocessor_success: with open(preprocessor_path, 'wb') as f:
pickle.dump(preprocessor, f) print(f"Saved new preprocessor to {preprocessor_path}") # Load
the model if os.path.exists(MODEL_PATH): print(f"Loading model from {MODEL_PATH}") model
= BusinessSuccessModel.load_from_checkpoint(MODEL_PATH) model.eval() model_success =
True else: print(f"Model not found at {MODEL_PATH}") model = None model_success = False #
Add needed columns for EDA visualizations try: if 'similar_businesses_count' in df.columns:
df['similar_biz_bins'] = pd.cut( df['similar_businesses_count'], bins=[0, 5, 10, 20, 50, 100, 500,
10000], labels=['0-5', '6-10', '11-20', '21-50', '51-100', '101-500', '500+'] ) if 'business_age_years'
in df.columns: df['age_bins'] = pd.cut( df['business_age_years'], bins=[0, 1, 2, 3, 5, 10, 20, 100],
labels=['<1 year', '1-2 years', '2-3 years', '3-5 years', '5-10 years', '10-20 years', '20+ years'] )
except Exception as e: print(f"Error adding EDA columns: {e}") rows = [] # Generate realistic
dummy data for each neighborhood for neighborhood, (lat, lon) in sf_neighborhoods.items(): #
Create success rates based on general San Francisco trends base_success_rate = 0.5 # Base
50% success rate # Adjust success rates based on neighborhood (using realistic patterns) if
neighborhood in ["Financial District", "Marina", "Nob Hill", "Pacific Heights", "Russian Hill"]:
success_adjustment = 0.2 # Premium neighborhoods elif neighborhood in ["Bayview",
"Visitacion Valley", "Excelsior"]: success_adjustment = -0.15 # Challenging neighborhoods else:
success_adjustment = np.random.uniform(-0.1, 0.1) # Random adjustment for others #
Generate 20 random businesses for each neighborhood for i in range(20): # Generate random
position within neighborhood random_lat = lat + np.random.uniform(-0.003, 0.003) random_lon
= lon + np.random.uniform(-0.003, 0.003) # Pick industry industry = np.random.choice([
"Restaurant", "Retail", "Technology", "Healthcare", "Professional Services", "Financial Services",
"Entertainment", "Education", "Manufacturing", "Wholesale" ]) # Adjust success probability
based on industry if industry in ["Technology", "Healthcare", "Professional Services", "Financial
Services"]: industry_adj = 0.15 # Premium industries elif industry in ["Restaurant", "Retail",
"Entertainment"]: industry_adj = -0.1 # Challenging industries else: industry_adj = 0.0 #
Determine overall success probability success_prob = base_success_rate +
success_adjustment + industry_adj success_prob = max(0.1, min(0.9, success_prob)) # Bound
between 0.1 and 0.9 # Generate success based on probability success = 1 if
np.random.random() < success_prob else 0 # Generate business age (correlated with success)
if success == 1: age = np.random.uniform(5, 20) else: age = np.random.uniform(0.5, 4.9) #
Generate crime and competition indicators high_crime = 1 if neighborhood in ["Bayview",
"Visitacion Valley", "South of Market"] else 0 similar_biz_count = np.random.randint(5, 200) #
Add row to dataset rows.append({ 'neighborhoods_analysis_boundaries': neighborhood,
'business_industry': industry, 'long_term_success': success, 'latitude': random_lat, 'longitude':
```

```
random_lon, 'high_crime_area': high_crime, 'similar_businesses_count': similar_biz_count,
'business_age_years': age, 'start_year': np.random.randint(2005, 2023), 'sf_gdp':
np.random.uniform(800, 1200), 'sf_unemployment_rate': np.random.uniform(0.02, 0.08),
'sf_house_price_index': np.random.uniform(300, 500), 'neighborhood_business_count':
np.random.randint(100, 2000), 'district_total_crimes': np.random.randint(50, 500) if high_crime
else np.random.randint(10, 100), 'overall_sentiment_mean': np.random.uniform(0.3, 0.8) }) #
Create DataFrame df = pd.DataFrame(rows) # Add needed categorical columns
df['similar_biz_bins'] = pd.cut( df['similar_businesses_count'], bins=[0, 5, 10, 20, 50, 100, 500,
10000], labels=['0-5', '6-10', '11-20', '21-50', '51-100', '101-500', '500+'] ) df['age_bins'] = pd.cut(
df['business_age_years'], bins=[0, 1, 2, 3, 5, 10, 20, 100], labels=['<1 year', '1-2 years', '2-3
years', '3-5 years', '5-10 years', '10-20 years', '20+ years'] ) return None, None, df, False if
'business_age_years' in df.columns: df['age_bins'] = pd.cut( df['business_age_years'], bins=[0,
1, 2, 3, 5, 10, 20, 100], labels=['<1 year', '1-2 years', '2-3 years', '3-5 years', '5-10 years', '10-20
years', '20+ years'] ) return model, preprocessor, df, model_success and preprocessor_success
except Exception as e: st.warning(f"Error loading model components: {e}") st.warning("Using
demonstration mode instead") # Create comprehensive SF neighborhood data with coordinates
sf_neighborhoods = { "Bayview": (37.7299, -122.3853), "Bernal Heights": (37.7394, -122.4156),
"Lakeshore": (37.7230, -122.4825), "Marina": (37.8021, -122.4369), "Mission": (37.7599,
-122.4148), "Nob Hill": (37.7930, -122.4161), "Noe Valley": (37.7502, -122.4337), "North Beach":
(37.8003, -122.4097), "Outer Richmond": (37.7780, -122.4934), "Outer Sunset": (37.7550,
-122.4944), "Pacific Heights": (37.7925, -122.4382), "Potrero Hill": (37.7605, -122.4010),
"Russian Hill": (37.8014, -122.4182), "South of Market": (37.7785, -122.4056), "Twin Peaks":
(37.7559, -122.4443), "Visitacion Valley": (37.7128, -122.4107), "Western Addition": (37.7804,
-122.4332) } return model, preprocessor, df, model_success and preprocessor_success #
Create demo data df = create_sample_data(sf_neighborhoods) return None, None, df, False #
Function to create sample data def create_sample_data(neighborhoods=None): if
neighborhoods is None: neighborhoods = { "Bayview": (37.7299, -122.3853), "Financial District":
(37.7946, -122.3999), "Mission": (37.7599, -122.4148), "South of Market": (37.7785, -122.4056),
"Marina": (37.8021, -122.4369) } rows = [] # Generate realistic dummy data for each
neighborhood for neighborhood, (lat, lon) in neighborhoods.items(): # Create success rates
based on general San Francisco trends base_success_rate = 0.5 # Base 50% success rate #
Adjust success rates based on neighborhood (using realistic patterns) if neighborhood in
["Financial District", "Marina", "Nob Hill", "Pacific Heights", "Russian Hill"]: success_adjustment =
0.2 # Premium neighborhoods elif neighborhood in ["Bayview", "Visitacion Valley", "Excelsior"]:
success_adjustment = -0.15 # Challenging neighborhoods else: success_adjustment =
np.random.uniform(-0.1, 0.1) # Random adjustment for others # Generate 20 random
businesses for each neighborhood for i in range(20): # Generate random position within
neighborhood random_lat = lat + np.random.uniform(-0.003, 0.003) random_lon = lon +
np.random.uniform(-0.003, 0.003) # Pick industry industry = np.random.choice([ "Restaurant",
"Retail", "Technology", "Healthcare", "Professional Services", "Financial Services",
"Entertainment", "Education", "Manufacturing", "Wholesale" ]) # Adjust success probability
based on industry if industry in ["Technology", "Healthcare", "Professional Services", "Financial
Services"]: industry_adj = 0.15 # Premium industries elif industry in ["Restaurant", "Retail",
"Entertainment"]: industry_adj = -0.1 # Challenging industries else: industry_adj = 0.0 #
Determine overall success probability success_prob = base_success_rate +
success_adjustment + industry_adj success_prob = max(0.1, min(0.9, success_prob)) # Bound
between 0.1 and 0.9 # Generate success based on probability success = 1 if
np.random.random() < success_prob else 0 # Generate business age (correlated with success)
if success == 1: age = np.random.uniform(5, 20) else: age = np.random.uniform(0.5, 4.9) #
Generate crime and competition indicators high_crime = 1 if neighborhood in ["Bayview",
"Visitacion Valley", "South of Market"] else 0 similar_biz_count = np.random.randint(5, 200) #
Add economic indicators sf_gdp = np.random.uniform(800, 1200) # Billions
sf_unemployment_rate = np.random.uniform(0.02, 0.08) # 2-8% sf_house_price_index =
np.random.uniform(300, 500) # Index value # Add neighborhood business count
neighborhood_business_count = np.random.randint(100, 2000) # Add district crime stats
district_total_crimes = np.random.randint(50, 500) if high_crime else np.random.randint(10, 100)
# Add sentiment overall_sentiment_mean = np.random.uniform(0.3, 0.8) # Add row to dataset
```

rows.append({ 'neighborhoods_analysis_boundaries': neighborhood, 'business_industry': industry, 'long_term_success': success, 'latitude': random_lat, 'longitude': random_lon, 'high_crime_area': high_crime, 'similar_businesses_count': similar_biz_count, 'business_age_years': age, 'start_year': np.random.randint(2005, 2023), 'sf_gdp': sf_gdp, 'sf_unemployment_rate': sf_unemployment_rate, 'sf_house_price_index': sf_house_price_index, 'neighborhood_business_count': neighborhood_business_count, 'district_total_crimes': district_total_crimes, 'overall_sentiment_mean': overall_sentiment_mean }) # Create DataFrame df = pd.DataFrame(rows) # Add needed categorical columns df['similar_biz_bins'] = pd.cut( df['similar_businesses_count'], bins=[0, 5, 10, 20, 50, 100, 500, 10000], labels=['0-5', '6-10', '11-20', '21-50', '51-100', '101-500', '500+'] ) df['age_bins'] = pd.cut( df['business_age_years'], bins=[0, 1, 2, 3, 5, 10, 20, 100], labels=['<1 year', '1-2 years', '2-3 years', '3-5 years', '5-10 years', '10-20 years', '20+ years'] ) return df # Function to make predictions def predict_success(model, preprocessor, input_data): """Make prediction using the trained model""" # If demo mode, generate a realistic prediction based on inputs if model is None or preprocessor is None: # Generate a realistic prediction based on neighborhood and industry neighborhood = input_data['neighborhoods_analysis_boundaries'].iloc[0] industry = input_data['business_industry'].iloc[0] # Base score adjusted by location and industry base_score = 0.5 # Adjust for neighborhood premium_neighborhoods = ['Financial District', 'Marina', 'Nob Hill', 'Russian Hill', 'Pacific Heights'] challenging_neighborhoods = ['Bayview', 'Visitacion Valley', 'Tenderloin'] if neighborhood in premium_neighborhoods: neighborhood_adj = 0.2 elif neighborhood in challenging_neighborhoods: neighborhood_adj = -0.15 else: neighborhood_adj = 0.05 # Adjust for industry premium_industries = ['Technology', 'Healthcare', 'Professional Services', 'Financial Services'] challenging_industries = ['Restaurant', 'Retail', 'Entertainment'] if industry in premium_industries: industry_adj = 0.15 elif industry in challenging_industries: industry_adj = -0.1 else: industry_adj = 0.0 # Additional adjustments if 'high_crime_area' in input_data.columns and input_data['high_crime_area'].iloc[0] == 1: crime_adj = -0.15 else: crime_adj = 0.05 # Calculate final score prediction = base_score + neighborhood_adj + industry_adj + crime_adj # Ensure prediction is between 0 and 1 prediction = max(0.1, min(0.9, prediction)) return prediction # With real model, make actual prediction def make_prediction(model, preprocessor, input_data): """Make prediction using the trained model""" try: # Ensure all required columns exist for col in preprocessor.feature_names_in_: if col not in input_data.columns: input_data[col] = np.nan # Preprocess the input data input_processed = preprocessor.transform(input_data) # Convert to tensor and predict input_tensor = torch.FloatTensor(input_processed) with torch.no_grad(): prediction = model(input_tensor).item() return prediction except Exception as e: st.error(f"Error making prediction: {e}") return 0.5 # Default to 50% if prediction fails # Create success gauge chart def create_success_gauge(score): """Create a gauge chart for the success score""" fig = go.Figure(go.Indicator( mode="gauge+number", value=score * 100, domain={'x': [0, 1], 'y': [0, 1]}, title={'text': "5-Year Success Probability"}, gauge={ 'axis': {'range': [0, 100]}, 'bar': {'color': "darkblue"}, 'steps': [ {'range': [0, 30], 'color': 'red'}, {'range': [30, 50], 'color': 'orange'}, {'range': [50, 70], 'color': 'yellow'}, {'range': [70, 100], 'color': 'green'} ], 'threshold': { 'line': {'color': "black", 'width': 4}, 'thickness': 0.75, 'value': score * 100 } } )) fig.update_layout( height=300, margin=dict(l=20, r=20, t=50, b=20), ) return fig # Create neighborhood success chart (enhanced from EDA) def create_neighborhood_chart(df, selected_neighborhood=None): """Create an interactive bar chart of success rates by neighborhood""" try: # Group by neighborhood and calculate success rate neigh_stats = df.groupby('neighborhoods_analysis_boundaries')['long_term_success'].agg(['mean', 'count']).reset_index() neigh_stats.columns = ['neighborhood', 'success_rate', 'business_count'] neigh_stats = neigh_stats.sort_values('success_rate', ascending=False) # Filter for significant sample (reduced threshold for more data) neigh_stats = neigh_stats[neigh_stats['business_count'] >= 10] # Check if we have data if neigh_stats.empty: # Create dummy data for demonstration neigh_stats = pd.DataFrame({ 'neighborhood': ['Financial District', 'Marina', 'Nob Hill', 'Pacific Heights', 'Russian Hill', 'SoMa', 'Mission', 'North Beach', 'Castro', 'Hayes Valley'], 'success_rate': [0.75, 0.72, 0.68, 0.65, 0.62, 0.58, 0.55, 0.52, 0.48, 0.45], 'business_count': [120, 110, 95, 85, 75, 65, 60, 55, 50, 45] }) # Add selected neighborhood if it's not in the list if selected_neighborhood and selected_neighborhood not in neigh_stats['neighborhood'].values: new_row = pd.DataFrame({ 'neighborhood':

[selected_neighborhood], 'success_rate': [0.50], # Default value 'business_count': [30] }) neigh_stats = pd.concat([neigh_stats, new_row]).reset_index(drop=True) # Limit to top 10 neighborhoods plot_neighs = neigh_stats.head(10) # Highlight selected neighborhood if it exists in the top 10 plot_neighs['selected'] = False if selected_neighborhood and selected_neighborhood in plot_neighs['neighborhood'].values: plot_neighs.loc[plot_neighs['neighborhood'] == selected_neighborhood, 'selected'] = True # If selected neighborhood isn't in top 10, add it to the dataframe elif selected_neighborhood: selected_data = neigh_stats[neigh_stats['neighborhood'] == selected_neighborhood] if not selected_data.empty: selected_data['selected'] = True other_data = plot_neighs.copy() plot_neighs = pd.concat([selected_data, other_data]).reset_index(drop=True) fig = px.bar( plot_neighs, x='neighborhood', y='success_rate', color='selected', color_discrete_map={True: 'darkblue', False: '#1f77b4'}, text='business_count', title='Top Neighborhoods by 5-Year Business Survival Rate', labels={ 'neighborhood': 'Neighborhood', 'success_rate': '5-Year Survival Rate', 'business_count': 'Number of Businesses' } ) fig.update_layout( xaxis_tickangle=-45, yaxis_tickformat='.0%', showlegend=False, height=400 ) fig.update_traces(texttemplate='n=%{text}', textposition='outside') return fig except Exception as e: # Create a fallback figure with dummy data st.warning(f"Error creating neighborhood chart: {e}. Using demo data instead.") # Create dummy data dummy_data = pd.DataFrame({ 'neighborhood': ['Financial District', 'Marina', 'Nob Hill', 'Pacific Heights', 'Russian Hill', 'SoMa', 'Mission', 'North Beach', 'Castro', 'Hayes Valley'], 'success_rate': [0.75, 0.72, 0.68, 0.65, 0.62, 0.58, 0.55, 0.52, 0.48, 0.45], 'business_count': [120, 110, 95, 85, 75, 65, 60, 55, 50, 45], 'selected': [False, False, False, False, False, False, False, False, False, False] }) # Mark selected neighborhood if included if selected_neighborhood in dummy_data['neighborhood'].values: dummy_data.loc[dummy_data['neighborhood'] == selected_neighborhood, 'selected'] = True fig = px.bar( dummy_data, x='neighborhood', y='success_rate', color='selected', color_discrete_map={True: 'darkblue', False: '#1f77b4'}, text='business_count', title='Top Neighborhoods by 5-Year Business Survival Rate (Demo Data)', labels={ 'neighborhood': 'Neighborhood', 'success_rate': '5-Year Survival Rate', 'business_count': 'Number of Businesses' } ) fig.update_layout( xaxis_tickangle=-45, yaxis_tickformat='.0%', showlegend=False, height=400 ) fig.update_traces(texttemplate='n=%{text}', textposition='outside') return fig # Create industry success chart (enhanced from EDA) def create_industry_chart(df, selected_industry=None): """Create an interactive bar chart of success rates by industry""" try: # Group by industry and calculate success rate industry_stats = df.groupby('business_industry')['long_term_success'].agg(['mean', 'count']).reset_index() industry_stats.columns = ['industry', 'success_rate', 'business_count'] industry_stats = industry_stats.sort_values('success_rate', ascending=False) # Filter for significant sample (reduced threshold for more data) industry_stats = industry_stats[industry_stats['business_count'] >= 10] # Check if we have data if industry_stats.empty: # Create dummy data for demonstration industry_stats = pd.DataFrame({ 'industry': ['Professional Services', 'Healthcare', 'Technology', 'Financial Services', 'Education', 'Manufacturing', 'Wholesale', 'Transportation', 'Retail', 'Restaurant'], 'success_rate': [0.72, 0.68, 0.65, 0.62, 0.58, 0.55, 0.52, 0.48, 0.45, 0.42], 'business_count': [110, 95, 85, 75, 65, 60, 55, 50, 45, 40] }) # Add selected industry if it's not in the list if selected_industry and selected_industry not in industry_stats['industry'].values: new_row = pd.DataFrame({ 'industry': [selected_industry], 'success_rate': [0.50], # Default value 'business_count': [30] }) industry_stats = pd.concat([industry_stats, new_row]).reset_index(drop=True) # Limit to top 10 industries plot_industries = industry_stats.head(10) # Highlight selected industry if it exists in the top 10 plot_industries['selected'] = False if selected_industry and selected_industry in plot_industries['industry'].values: plot_industries.loc[plot_industries['industry'] == selected_industry, 'selected'] = True # If selected industry isn't in top 10, add it to the dataframe elif selected_industry: selected_data = industry_stats[industry_stats['industry'] == selected_industry] if not selected_data.empty: selected_data['selected'] = True other_data = plot_industries.copy() plot_industries = pd.concat([selected_data, other_data]).reset_index(drop=True) fig = px.bar( plot_industries, x='industry', y='success_rate', color='selected', color_discrete_map={True: 'darkblue', False: '#1f77b4'}, text='business_count', title='Top Industries by 5-Year Business Survival Rate', labels={ 'industry': 'Industry', 'success_rate': '5-Year Survival Rate', 'business_count': 'Number of Businesses' } ) fig.update_layout( xaxis_tickangle=-45, yaxis_tickformat='.0%', showlegend=False, height=400 )

```
fig.update_traces(texttemplate='n=%{text}', textposition='outside') return fig except Exception as
e: # Create a fallback figure with dummy data st.warning(f"Error creating industry chart: {e}.
Using demo data instead.") # Create dummy data dummy_data = pd.DataFrame({ 'industry':
['Professional Services', 'Healthcare', 'Technology', 'Financial Services', 'Education',
'Manufacturing', 'Wholesale', 'Transportation', 'Retail', 'Restaurant'], 'success_rate': [0.72, 0.68,
0.65, 0.62, 0.58, 0.55, 0.52, 0.48, 0.45, 0.42], 'business_count': [110, 95, 85, 75, 65, 60, 55, 50,
45, 40], 'selected': [False, False, False, False, False, False, False, False, False, False] }) # Mark
selected industry if included if selected_industry in dummy_data['industry'].values:
dummy_data.loc[dummy_data['industry'] == selected_industry, 'selected'] = True fig = px.bar(
dummy_data, x='industry', y='success_rate', color='selected', color_discrete_map={True:
'darkblue', False: '#1f77b4'}, text='business_count', title='Top Industries by 5-Year Business
Survival Rate (Demo Data)', labels={ 'industry': 'Industry', 'success_rate': '5-Year Survival Rate',
'business_count': 'Number of Businesses' } ) fig.update_layout( xaxis_tickangle=-45,
yaxis_tickformat='.0%', showlegend=False, height=400 ) fig.update_traces(texttemplate='n=%
{text}', textposition='outside') return fig # Create crime impact visualization from EDA def
create_crime_impact_chart(df, selected_high_crime=False): """Create a chart showing impact of
crime on business success""" # Ensure high_crime_area column exists if 'high_crime_area' not
in df.columns: # Create dummy data return None # Ensure boolean column is numeric
df['high_crime_area'] = df['high_crime_area'].astype(int) # Create visualization of crime impact
crime_area_success = df.groupby('high_crime_area')['long_term_success'].agg(['mean',
'count']).reset_index() crime_area_success['high_crime_area'] =
crime_area_success['high_crime_area'].map({0: 'Low Crime Area', 1: 'High Crime Area'})
crime_area_success = crime_area_success.rename(columns={'mean': 'success_rate'}) # Add
column to highlight user's selection crime_area_success['selected'] =
crime_area_success['high_crime_area'] == ('High Crime Area' if selected_high_crime else 'Low
Crime Area') fig = px.bar( crime_area_success, x='high_crime_area', y='success_rate',
color='selected', text='count', title='5-Year Business Survival Rate by Crime Level', labels=
{'success_rate': '5-Year Survival Rate', 'high_crime_area': 'Area Type'}, color_discrete_map=
{True: 'darkblue', False: '#1f77b4'} ) fig.update_layout( template='plotly_white', xaxis_title='Area
Type', yaxis_title='5-Year Survival Rate', yaxis=dict(tickformat='.0%'), height=300,
showlegend=False ) fig.update_traces(texttemplate='n=%{text}', textposition='outside') return fig
# Create competition impact visualization from EDA def create_competition_chart(df): """Create
a chart showing impact of competition on business success""" # Ensure similar_biz_bins column
exists if 'similar_biz_bins' not in df.columns: return None # Calculate success rates by
competition bins similar_biz_success = df.groupby('similar_biz_bins')
['long_term_success'].agg(['mean', 'count']).reset_index() similar_biz_success =
similar_biz_success.rename(columns={'mean': 'success_rate'}) fig = px.bar(
similar_biz_success, x='similar_biz_bins', y='success_rate', color='success_rate', text='count',
title='Business Success Rate by Competitive Density', labels={ 'similar_biz_bins': 'Number of
Similar Businesses', 'success_rate': '5-Year Survival Rate', 'count': 'Number of Businesses' },
color_continuous_scale='RdYlGn' ) fig.update_layout( template='plotly_white',
xaxis_title='Number of Similar Businesses in Area', yaxis_title='5-Year Survival Rate',
yaxis=dict(tickformat='.0%'), height=300 ) fig.update_traces(texttemplate='n=%{text}',
textposition='outside') return fig # Create business age impact visualization from EDA def
create_business_age_chart(df): """Create a chart showing business age impact on success""" #
Ensure age_bins column exists if 'age_bins' not in df.columns: return None # Calculate success
rates by age bins age_success = df.groupby('age_bins')['long_term_success'].agg(['mean',
'count']).reset_index() age_success = age_success.rename(columns={'mean': 'success_rate'})
fig = px.bar( age_success, x='age_bins', y='success_rate', color='success_rate', text='count',
title='Business Success Rate by Business Age', labels={ 'age_bins': 'Business Age',
'success_rate': 'Success Rate', 'count': 'Number of Businesses' },
color_continuous_scale='RdYlGn' ) fig.update_layout( template='plotly_white',
xaxis_title='Business Age', yaxis_title='Success Rate', yaxis=dict(tickformat='.0%'), height=300 )
fig.update_traces(texttemplate='n=%{text}', textposition='outside') return fig # Create spatial
heatmap of business success def create_spatial_heatmap(df, lat, lon,
selected_neighborhood=None): """Create an interactive map visualization of business
success""" # Sample data for performance reasons sample_size = min(5000, len(df)) sample_df
```

```
= df.sample(sample_size, random_state=42) # Filter for selected neighborhood if provided if
selected_neighborhood: neighborhood_data = df[df['neighborhoods_analysis_boundaries'] ==
selected_neighborhood] # If we have data for this neighborhood, use it if not
neighborhood_data.empty: # Ensure we don't have too many points from one neighborhood if
len(neighborhood_data) > 200: neighborhood_data = neighborhood_data.sample(200,
random_state=42) # Mix selected neighborhood data with random sample from other
neighborhoods other_neighborhoods = df[df['neighborhoods_analysis_boundaries'] !=
selected_neighborhood] other_sample = other_neighborhoods.sample(min(3000,
len(other_neighborhoods)), random_state=42) sample_df = pd.concat([neighborhood_data,
other_sample]) # Add marker for selected location location_df = pd.DataFrame({ 'latitude': [lat],
'longitude': [lon], 'long_term_success': [0.5], # Placeholder value 'business_industry': ['Selected
Location'], 'neighborhoods_analysis_boundaries': ['Your Location'], 'business_age_years': [0] }) #
Combine sample with selected location map_df = pd.concat([sample_df, location_df]) fig =
px.scatter_mapbox( map_df, lat='latitude', lon='longitude', color='long_term_success',
color_continuous_scale=["red", "green"], size_max=10, zoom=12, mapbox_style="carto-
positron", title=f'Geographic Distribution of Business Success in San Francisco {f"-
{selected_neighborhood}" if selected_neighborhood else ""}', opacity=0.7, labels=
{'long_term_success': '5-Year Survival'}, hover_data=['business_industry',
'neighborhoods_analysis_boundaries', 'business_age_years'] ) # Highlight the selected location
fig.add_trace( go.Scattermapbox( lat=[lat], lon=[lon], mode='markers', marker=dict(size=15,
color='blue', opacity=1), text="Selected Location", hoverinfo='text', showlegend=False ) )
fig.update_layout( height=500, margin=dict(l=0, r=0, t=50, b=0) ) return fig # Create feature
importance chart using Plotly def create_feature_importance_chart(): """Create a feature
importance chart using Plotly""" # Demo importance values (replace with actual values if
available) features = ['Neighborhood', 'Industry Type', 'Competition', 'Economic Indicators',
'Crime Rate'] importance = [0.35, 0.25, 0.20, 0.15, 0.05] fig = px.bar( x=importance, y=features,
orientation='h', color=importance, color_continuous_scale='Blues', title='Key Factors Influencing
Business Success' ) fig.update_layout( template='plotly_white', xaxis_title='Relative Importance',
height=300, yaxis=dict(autorange="reversed") # Reverse y-axis to show highest importance at
top ) return fig # Function to find top alternative neighborhoods def
find_alternative_neighborhoods(df, current_neighborhood, industry, top_n=3): """Find top
alternative neighborhoods for the given industry""" # Group by neighborhood and industry
grouped = df.groupby(['neighborhoods_analysis_boundaries', 'business_industry'])
['long_term_success'].agg(['mean', 'count']).reset_index() grouped.columns = ['neighborhood',
'industry', 'success_rate', 'business_count'] # Filter for the selected industry and sufficient
sample size industry_options = grouped[(grouped['industry'] == industry) &
(grouped['business_count'] >= 20) & (grouped['neighborhood'] != current_neighborhood)] # Sort
by success rate top_alternatives = industry_options.sort_values('success_rate',
ascending=False).head(top_n) return top_alternatives # Main app def main(): # Page config
st.set_page_config( page_title="SF Business Success Predictor", page_icon="📊",
layout="wide" ) # Load model and data model, preprocessor, df, using_real_model =
load_model_components() # Title st.title("📍 San Francisco Business Site Selection Tool")
st.markdown("#### Predict 5-year business survival probability for potential locations") # Set up
sidebar if using_real_model: st.sidebar.success("✅ Using trained model") else:
st.sidebar.warning("⚠️ Using demonstration mode") # Sidebar inputs
st.sidebar.header("Location Information") # Get complete SF neighborhood list
sf_neighborhoods = [ "Bayview", "Bernal Heights", "Castro/Upper Market", "Chinatown",
"Excelsior", "Financial District", "Glen Park", "Golden Gate Park", "Haight Ashbury", "Hayes
Valley", "Inner Richmond", "Inner Sunset", "Lakeshore", "Marina", "Mission", "Nob Hill", "Noe
Valley", "North Beach", "Outer Richmond", "Outer Sunset", "Pacific Heights", "Potrero Hill",
"Russian Hill", "South of Market", "Twin Peaks", "Visitacion Valley", "Western Addition" ] # Sort
neighborhoods alphabetically sf_neighborhoods.sort() # Define neighborhood coordinates for
mapping neighborhood_coords = { "Bayview": (37.7299, -122.3853), "Bernal Heights": (37.7394,
-122.4156), "Castro/Upper Market": (37.7609, -122.4351), "Chinatown": (37.7941, -122.4078),
"Excelsior": (37.7245, -122.4294), "Financial District": (37.7946, -122.3999), "Glen Park":
(37.7381, -122.4341), "Golden Gate Park": (37.7694, -122.4862), "Haight Ashbury": (37.7692,
-122.4481), "Hayes Valley": (37.7759, -122.4260), "Inner Richmond": (37.7801, -122.4637),
"Inner Sunset": (37.7611, -122.4705), "Lakeshore": (37.7230, -122.4825), "Marina": (37.8021,
-122.4369), "Mission": (37.7599, -122.4148), "Nob Hill": (37.7930, -122.4161), "Noe Valley":
(37.7502, -122.4337), "North Beach": (37.8003, -122.4097), "Outer Richmond": (37.7780,
-122.4934), "Outer Sunset": (37.7550, -122.4944), "Pacific Heights": (37.7925, -122.4382),
"Potrero Hill": (37.7605, -122.4010), "Russian Hill": (37.8014, -122.4182), "South of Market":
(37.7785, -122.4056), "Twin Peaks": (37.7559, -122.4443), "Visitacion Valley": (37.7128,
-122.4107), "Western Addition": (37.7804, -122.4332) } # Default location (San Francisco city
center) default_lat, default_lon = 37.7749, -122.4194 # Select neighborhood from dropdown
selected_neighborhood = st.sidebar.selectbox("Neighborhood", sf_neighborhoods,
index=sf_neighborhoods.index("Financial District") if "Financial District" in sf_neighborhoods
else 0) # Update lat/lon based on selected neighborhood if selected_neighborhood in
neighborhood_coords: default_lat, default_lon = neighborhood_coords[selected_neighborhood]
# Coordinates input (pre-filled with neighborhood coordinates) col1, col2 = st.sidebar.columns(2)
with col1: latitude = st.number_input("Latitude", value=default_lat, format="%.6f") with col2:
longitude = st.number_input("Longitude", value=default_lon, format="%.6f") # Get unique
industries try: industries = sorted(df['business_industry'].unique()) except: industries =
["Restaurant", "Retail", "Technology", "Professional Services", "Healthcare", "Financial
Services", "Entertainment", "Education", "Manufacturing", "Wholesale"]
st.sidebar.header("Business Information") selected_industry = st.sidebar.selectbox("Business
Industry", industries, index=industries.index("Restaurant") if "Restaurant" in industries else 0) #
Updated Start Year slider to begin from 2025 (current year) and extend to 2030 (5 years)
start_year = st.sidebar.slider("Start Year", min_value=2025, max_value=2030, value=2025)
st.sidebar.header("Economic & Local Factors") unemployment_rate =
st.sidebar.slider("Unemployment Rate (%)", 2.0, 10.0, 4.2, 0.1) high_crime_area =
st.sidebar.checkbox("High Crime Area?", value=False) # Build input data input_data =
pd.DataFrame({ 'latitude': [latitude], 'longitude': [longitude], 'business_industry':
[selected_industry], 'neighborhoods_analysis_boundaries': [selected_neighborhood],
'start_year': [start_year], 'sf_unemployment_rate': [unemployment_rate / 100.0], # Convert to
decimal 'high_crime_area': [1 if high_crime_area else 0] }) # Make prediction button if
st.sidebar.button("Predict Success Probability", type="primary"): with st.spinner("Analyzing
location and calculating success probability..."): # Make prediction success_score =
predict_success(model, preprocessor, input_data) # Create main tabs tabs =
st.tabs(["Overview", "Comparative Analysis", "Risk Factors", "Recommendations"]) # TAB 1:
OVERVIEW - Main results with Success Score and Map with tabs[0]: # Two columns for score
and map col1, col2 = st.columns([1, 2]) with col1: st.markdown("### Success Score")
gauge_chart = create_success_gauge(success_score) st.plotly_chart(gauge_chart,
use_container_width=True) # Score interpretation if success_score >= 0.7: st.success("This
location has a **high probability** of 5-year business success.") elif success_score >= 0.5:
st.warning("This location has a **moderate probability** of 5-year business success.") else:
st.error("This location has a **low probability** of 5-year business success.") # Add model
information if model is not None and preprocessor is not None: st.info("✅ Using trained model
for predictions") else: st.info("⚠️ Using demonstration mode (trained model not available)") with
col2: st.markdown("### Geographic Context") # Create and display spatial heatmap (now using
selected_neighborhood) spatial_map = create_spatial_heatmap(df, latitude, longitude,
selected_neighborhood) st.plotly_chart(spatial_map, use_container_width=True) st.info(f"**Map
Legend**: Green dots indicate successful businesses (5+ years), red dots show businesses that
didn't survive 5 years, and the blue marker shows your selected location in
{selected_neighborhood}.") # Add key factors summary st.markdown("### Key Factors
Influencing Prediction") factors_chart = create_feature_importance_chart()
st.plotly_chart(factors_chart, use_container_width=True) # TAB 2: COMPARATIVE ANALYSIS -
Neighborhood and Industry comparisons with tabs[1]: st.markdown("## Comparative Analysis")
st.markdown("See how your location compares to others in San Francisco") # Two columns for
neighborhood and industry comparisons col1, col2 = st.columns(2) with col1: st.markdown("###
Neighborhood Performance") neigh_chart = create_neighborhood_chart(df,
selected_neighborhood) st.plotly_chart(neigh_chart, use_container_width=True) # Calculate
```

```
neighborhood-specific stats try: neigh_stats = df.groupby('neighborhoods_analysis_boundaries')
['long_term_success'].mean().reset_index() neigh_stats.columns = ['neighborhood',
'success_rate'] selected_neigh_rate = neigh_stats[neigh_stats['neighborhood'] ==
selected_neighborhood]['success_rate'].iloc[0] city_avg = df['long_term_success'].mean() #
Display comparison metrics st.metric( f"Success Rate in {selected_neighborhood}", f"
{selected_neigh_rate:.1%}", f"{selected_neigh_rate - city_avg:.1%} vs. City Average" ) if
selected_neigh_rate > city_avg: st.success(f"**{selected_neighborhood}** performs **better**
than the city average.") else: st.error(f"**{selected_neighborhood}** performs **worse** than the
city average.") except: st.info("Detailed neighborhood statistics not available in demo mode.")
with col2: st.markdown("### Industry Performance") industry_chart = create_industry_chart(df,
selected_industry) st.plotly_chart(industry_chart, use_container_width=True) # Calculate
industry-specific stats try: industry_stats = df.groupby('business_industry')
['long_term_success'].mean().reset_index() industry_stats.columns = ['industry', 'success_rate']
selected_industry_rate = industry_stats[industry_stats['industry'] == selected_industry]
['success_rate'].iloc[0] city_avg = df['long_term_success'].mean() # Display comparison metrics
st.metric( f"Success Rate for {selected_industry}", f"{selected_industry_rate:.1%}", f"
{selected_industry_rate - city_avg:.1%} vs. City Average" ) if selected_industry_rate > city_avg:
st.success(f"**{selected_industry}** businesses perform **better** than the city average.") else:
st.error(f"**{selected_industry}** businesses perform **worse** than the city average.") except:
st.info("Detailed industry statistics not available in demo mode.") # Industry-Neighborhood
match analysis st.markdown("### Industry-Neighborhood Match Analysis") try: # Check how this
industry performs in this specific neighborhood location_match =
df[(df['neighborhoods_analysis_boundaries'] == selected_neighborhood) &
(df['business_industry'] == selected_industry)] if len(location_match) >= 20: # Enough data for
reliable estimate match_rate = location_match['long_term_success'].mean() # Compare to
industry average industry_avg = df[df['business_industry'] == selected_industry]
['long_term_success'].mean() # Display match quality st.metric( f"{selected_industry} in
{selected_neighborhood}", f"{match_rate:.1%}", f"{match_rate - industry_avg:.1%} vs. Industry
Average" ) if match_rate > industry_avg: st.success(f"**{selected_neighborhood}** is an
**above-average location** for {selected_industry} businesses.") else: st.error(f"**
{selected_neighborhood}** is a **below-average location** for {selected_industry} businesses.")
else: st.info(f"Limited historical data for {selected_industry} businesses in
{selected_neighborhood}. This may indicate an untested market opportunity or lack of industry-
location fit.") except: st.info("Detailed industry-neighborhood match analysis not available in
demo mode.") # TAB 3: RISK FACTORS - Show specific risks for this location with tabs[2]:
st.markdown("## Risk Factor Analysis") st.markdown("Examine specific factors that influence
business success at this location") # Two columns for crime and competition col1, col2 =
st.columns(2) with col1: st.markdown("### Crime Impact") crime_chart =
create_crime_impact_chart(df, high_crime_area) if crime_chart: st.plotly_chart(crime_chart,
use_container_width=True) if high_crime_area: st.warning("**High crime areas show a
significant reduction in business survival rates.** Consider additional security measures and loss
prevention strategies.") else: st.success("**Low crime areas show significantly better business
survival rates.** Your selected location is favorable from a security perspective.") else:
st.info("Crime impact data not available in demo mode.") with col2: st.markdown("###
Competition Analysis") competition_chart = create_competition_chart(df) if competition_chart:
st.plotly_chart(competition_chart, use_container_width=True) # If we have similar business
count in the input data if 'similar_businesses_count' in input_data.columns: similar_count =
input_data['similar_businesses_count'].iloc[0] if similar_count < 5: st.warning("**Very low
competition** in this area. This may indicate an untested market.") elif 10 <= similar_count <=
50: st.success("**Optimal competitive environment** with balanced market validation and
customer traffic.") elif similar_count > 100: st.error("**Extremely high competition** may make it
difficult to establish market share.") else: st.info("Competition analysis for your specific location
not available in demo mode.") else: st.info("Competition analysis data not available in demo
mode.") # Business age analysis for context st.markdown("### Business Survival Curve")
age_chart = create_business_age_chart(df) if age_chart: st.plotly_chart(age_chart,
use_container_width=True) st.info("**Survival Curve**: Business survival likelihood increases
dramatically after the first few years. The graph shows the critical periods for business survival.")
```

```
else: st.info("Business age analysis not available in demo mode.") # TAB 4:
RECOMMENDATIONS - Data-driven recommendations with tabs[3]: st.markdown("## Location
Recommendations") st.markdown("Data-driven insights to improve your business location
strategy") # Primary recommendation based on score st.markdown("### Primary
Recommendation") if success_score >= 0.7: st.success(""" ### High Potential Location ✓ This
location shows **strong potential for long-term business success**. Our analysis indicates it has
several favorable characteristics: * Located in a neighborhood with historically strong business
performance * Good industry-location match based on success patterns * Favorable risk profile
across key factors **Recommendation:** Proceed with confidence for this site selection. Focus
on execution excellence and standard due diligence. """) elif success_score >= 0.5:
st.warning(""" ### Moderate Potential Location ⚠️ This location shows **reasonable but not
exceptional potential** for long-term business success. While not ideal, it could still be viable
with the right approach: * Some positive factors are balanced by potential risk elements *
Performance may depend heavily on execution quality * Consider specific mitigation strategies
for identified risks **Recommendation:** Proceed with caution, focusing on strategies to address
the risk factors identified in this analysis. """) else: st.error(""" ### Challenging Location ⚠️ This
location shows **below-average potential** for long-term business success. Several risk factors
suggest this may not be an optimal site: * Historical data indicates below-average survival rates
* Multiple risk factors present that may challenge viability * Industry-location match appears
unfavorable **Recommendation:** Consider alternative locations, or develop specific strategies
to overcome the identified challenges. """) # Alternative neighborhood recommendations
st.markdown("### Alternative Neighborhood Suggestions") try: alternatives =
find_alternative_neighborhoods(df, selected_neighborhood, selected_industry) if not
alternatives.empty: st.markdown(f"Based on historical data, these neighborhoods show higher
success rates for **{selected_industry}** businesses:") # Display alternatives alt_fig = px.bar(
alternatives, x='neighborhood', y='success_rate', text='business_count', title=f'Top Alternative
Neighborhoods for {selected_industry}', color='success_rate', color_continuous_scale='RdYlGn',
labels={ 'neighborhood': 'Neighborhood', 'success_rate': '5-Year Survival Rate', 'business_count':
'Sample Size' } ) alt_fig.update_layout( xaxis_tickangle=-45, yaxis_tickformat='.0%', height=400
) alt_fig.update_traces(texttemplate='n=%{text}', textposition='outside') st.plotly_chart(alt_fig,
use_container_width=True) # Table with details st.markdown("#### Alternative Location
Details:") for i, row in alternatives.iterrows(): st.markdown(f"**{row['neighborhood']}** -
{row['success_rate']:.1%} success rate (based on {row['business_count']} businesses)") else:
st.info(f"No alternative neighborhoods with sufficient data for {selected_industry} businesses
found.") except: st.info("Alternative neighborhood suggestions not available in demo mode.") #
Strategic recommendations st.markdown("### Strategic Insights") # Based on crime if
high_crime_area: st.markdown(""" **Security Considerations:** * Invest in visible security
systems and good lighting * Consider adjusted business hours to reduce risk exposure * Build
strong relationships with local law enforcement * Implement robust inventory control and cash
management procedures """) # Based on industry if selected_industry in ['Restaurant', 'Retail',
'Entertainment']: st.markdown(""" **Customer-Facing Business Strategy:** * Focus on
exceptional customer experience to drive repeat business * Develop robust online presence to
supplement physical location * Consider delivery/pickup options to expand reach beyond
immediate area * Implement loyalty programs to build stable customer base """) # Timing
strategy st.markdown(""" **Optimal Timing Strategy:** * Plan for at least 18-24 months of
operating capital * First two years show highest business failure rates * Consider Q1 launch for
slightly higher success probability * Avoid launching during economic downturns if possible """)
else: # Initial state - welcome message with embedded map col1, col2 = st.columns([2, 1]) with
col1: st.markdown(""" ## Welcome to the SF Business Success Predictor This tool uses
machine learning to predict the probability of 5-year business survival at specific locations in
San Francisco. **How to use this tool:** 1. Select a location on the map or enter coordinates in
the sidebar 2. Specify your business industry and start date 3. Adjust any additional factors if
known 4. Click "Predict Success Probability" to see your results **What you'll get:** - Success
probability score with visual indicator - Interactive map showing business success patterns -
Comparative analysis against neighborhood and industry benchmarks - Risk factor analysis for
your specific location - Data-driven recommendations and alternative locations Start by entering
your information in the sidebar → """) with col2: # Example gauge chart example_gauge =
create_success_gauge(0.65) st.plotly_chart(example_gauge, use_container_width=True)
st.info("This tool combines spatial analysis, industry trends, and economic factors to help
entrepreneurs make optimal location decisions.") # Show preview map in main area
st.markdown("### Explore San Francisco Business Success Patterns") preview_map =
create_spatial_heatmap(df, default_lat, default_lon) st.plotly_chart(preview_map,
use_container_width=True) st.caption("Green dots indicate businesses that operated for 5+
years. Red dots show businesses that didn't reach the 5-year mark.") # Add necessary imports
for Folium map integration import folium from streamlit_folium import folium_static # Run the
app if __name__ == '__main__': main()
```