Analyzing NFL Quarterback Decisions and Play Performance Under Pressure

SIADS 593 - Winter 2025

Prathik Addepalli, Matthew Sitto & Utku Yurday

In this Notebook, we use the data from the NFL's 2022 Season to analyze QB decision-making when facing pressure. Specifically, we investigate what type of play (pass vs. scramble) QBs are more likely to favor, what the typical outcomes are and whether choosing one vs. the other has any influence on the success of the play. We predominantly use the Pandas library for data wrangling and analysis, along with the Altair library for our visualizations. The data is sourced from Kaggle (NFL Big Data Bowl 2025).

Please note that all markdown commentary refers to specific outputs of the code being run on the full Player Play file, which is 52MB in size. The data folder submitted with this Notebook contains a truncated version (*/src/data/player_play.csv*) which is 9MB. As such, while the entire Notebook has been tested to run using the truncated file as well, some of the markdown commentary may not match what is observed in the output cells. A pdf copy of this Notebook run on the full dataset has been provided along with the Project Report.

Table of Contents

Pre-Processing Steps

Let's begin by importing the required libraries and allowing Altair to work with our larger datasets (with the "no pressure" dataset exceeding 12k rows). To load the data, we will use the `process_qb_pressure_data` function from our imported script. We will call the function twice, once for "pressure" plays and once for "no pressure" plays and load them into 2 separate dataframes.

In [1]:
```
# Import required libraries
from process_qb_pressure import process_qb_pressure_data
import numpy as np
import pandas as pd
import altair as alt
from sklearn.linear_model import LinearRegression
```

In [2]:
```
# Allow Altair to process more than the max 5,000 rows by default
alt.data_transformers.disable_max_rows();
```

In [3]:
```
# Load data for both pressure and no pressure plays
games_file = "data/games.csv"
player_play_file = "data/player_play_original.csv"
players_file = "data/players.csv"
plays_file = "data/plays.csv"
```

The cleaned data contains the following columns:

- gameId: Game identifier, unique (numeric)
- playId: Play identifier, not unique across games (numeric)
- game_play_id: Unique identifier for each play (numeric)
- scoreDelta: Score surplus/deficit for the team associated with the play (numeric)
- displayName: Player name (text)
- down: Down (numeric)

- yardsToGo: Distance needed for a first down (numeric)
- absoluteYardlineNumber: Distance from end zone for possession team (numeric)
- passResult: Dropback outcome of the play (C: Complete pass, I: Incomplete pass, S: Quarterback sack, IN: Intercepted pass, R: Scramble, text)
- prePenaltyYardsGained: Net yards gained by the offense, before penalty yardage (numeric)
- passLength: The distance beyond the LOS that the ball traveled not including yards into the endzone. If thrown behind LOS, the value is negative. (numeric)
- timeToThrow: The time (secs) elapsed between snap and pass (numeric)
- timeInTackleBox: The amount of time the QB spent inside the tackle box (numeric)
- hadRushAttempt: Whether or not the player had a rushing attempt on this play (numeric)
- rushingYards: The rush yards accrued by the player on this play (numeric)
- qbSneak: Whether or not the play was a QB Sneak (numeric)
- qbKneel: Whether or not the play was a QB Kneel (numeric)
- qbSpike: Boolean indicating whether the play was a QB Spike (Boolean)
- passingYards: The pass yards accrued by the player on this play (numeric)

In [4]:
```
# Load the plays with QB under pressure
qb_pressure_df = process_qb_pressure_data(games_file, player_play_file, players_file, plays_file)
```

In [5]:
```
qb_pressure_df.head()
```

Out[5]:

	gameId	playId	game_play_id	scoreDelta	displayName	down	yardsToGo	absoluteYardlineNumber	passResult	prePenaltyYardsGained	passLength	timeToThrow	timeInTackleBox	hadRush
0	2022090800	80	2022090800 80	0	Josh Allen	2	4	79	R	7	NaN	NaN	NaN	
1	2022090800	122	2022090800 122	0	Josh Allen	2	3	65	C	6	-3.0	2.903	2.903	
2	2022090800	550	2022090800 550	7	Josh Allen	2	3	94	S	-1	NaN	NaN	NaN	
3	2022090800	867	2022090800 867	7	Josh Allen	1	10	42	C	2	-1.0	2.502	2.502	
4	2022090800	1504	2022090800 1504	3	Josh Allen	2	8	37	S	-4	NaN	NaN	NaN	

In [6]:
```
# Load the plays with NO pressure
qb_no_pressure_df = process_qb_pressure_data(games_file, player_play_file, players_file, plays_file, False)
```

In [7]:
```
qb_no_pressure_df.head()
```

Out[7]:

	gameId	playId	game_play_id	scoreDelta	displayName	down	yardsToGo	absoluteYardlineNumber	passResult	prePenaltyYardsGained	passLength	timeToThrow	timeInTackleBox	hadRush
0	2022090800	56	2022090800 56	0	Josh Allen	1	10	85	C	6	5.0	2.169	2.169	
1	2022090800	101	2022090800 101	0	Josh Allen	1	10	72	NaN	7	NaN	NaN	NaN	
2	2022090800	167	2022090800 167	0	Josh Allen	2	8	57	C	12	6.0	1.868	1.868	
3	2022090800	191	2022090800 191	0	Josh Allen	1	10	45	NaN	1	NaN	NaN	NaN	
4	2022090800	212	2022090800 212	0	Josh Allen	2	9	44	C	8	6.0	4.905	3.700	

Back to Table of Contents

Data Cleaning & EDA

Now that we've loaded our 2 datasets, let's start by double-checking there are no overlaps between the 2 dataframes.

In [8]:
```
len(set(qb_pressure_df['game_play_id']).intersection(set(qb_no_pressure_df['game_play_id'])))
```

Out[8]:
```
0
```

Let's start with some EDA and further cleanup of each dataframe.

In [9]:
```
# Check dataframe data types and counts
qb_pressure_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3268 entries, 0 to 3267
Data columns (total 19 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   gameId                  3268 non-null   int64
 1   playId                  3268 non-null   int64
 2   game_play_id            3268 non-null   object
 3   scoreDelta              3268 non-null   int64
 4   displayName             3268 non-null   object
 5   down                    3268 non-null   int64
 6   yardsToGo               3268 non-null   int64
 7   absoluteYardlineNumber  3268 non-null   int64
 8   passResult              3265 non-null   object
 9   prePenaltyYardsGained   3268 non-null   int64
 10  passLength              2492 non-null   float64
 11  timeToThrow             2492 non-null   float64
 12  timeInTackleBox         2703 non-null   float64
 13  hadRushAttempt          3268 non-null   int64
 14  rushingYards            3268 non-null   int64
 15  qbSneak                 170 non-null    object
 16  qbKneel                 3268 non-null   int64
 17  qbSpike                 3098 non-null   object
 18  passingYards            3268 non-null   int64
dtypes: float64(3), int64(11), object(5)
memory usage: 485.2+ KB
```

In [10]:
```
# Check dataframe descriptive stats
qb_pressure_df.describe()
```

Out[10]:

	gameId	playId	scoreDelta	down	yardsToGo	absoluteYardlineNumber	prePenaltyYardsGained	passLength	timeToThrow	timeInTackleBox	hadRushAttempt	rushingYar
count	3.268000e+03	3268.000000	3268.000000	3268.000000	3268.000000	3268.000000	3268.000000	2492.000000	2492.000000	2703.000000	3268.000000	3268.0000
mean	2.022099e+09	2079.881579	-2.828335	2.074663	8.794064	60.663403	3.724602	8.804575	3.296614	3.062573	0.052020	0.3855
std	6.008444e+03	1169.436276	9.047653	0.867708	3.935134	23.354010	9.728046	10.428511	1.171157	0.841735	0.222101	2.0737
min	2.022091e+09	54.000000	-35.000000	1.000000	1.000000	11.000000	-18.000000	-18.000000	1.201000	0.000000	0.000000	-2.0000
25%	2.022092e+09	1086.500000	-8.000000	1.000000	6.000000	42.000000	0.000000	2.000000	2.503000	2.500000	0.000000	0.0000
50%	2.022101e+09	2067.500000	-3.000000	2.000000	10.000000	60.000000	0.000000	6.000000	3.003000	2.936000	0.000000	0.0000
75%	2.022102e+09	3046.000000	3.000000	3.000000	10.000000	80.000000	8.000000	14.000000	3.737000	3.500000	0.000000	0.0000
max	2.022111e+09	5096.000000	35.000000	4.000000	30.000000	109.000000	61.000000	61.000000	9.643000	7.400000	1.000000	29.0000

To analyze pass vs. scramble decisions, we will use `passResult` as a key determinant. But, there are 3 null values for `passResult` in the "pressure" dataframe. Let's see what's going on there...

In [11]:
```
qb_pressure_df[qb_pressure_df['passResult'].isnull()]
```

Out[11]:

	gameId	playId	game_play_id	scoreDelta	displayName	down	yardsToGo	absoluteYardlineNumber	passResult	prePenaltyYardsGained	passLength	timeToThrow	timeInTackleBox	hadR
244	2022100201	517	2022100201 517	0	Marcus Mariota	1	9	101	NaN	-2	NaN	NaN	NaN	
555	2022101300	3257	2022101300 3257	-5	Justin Fields	2	5	40	NaN	0	NaN	NaN	NaN	
3184	2022102306	3952	2022102306 3952	2	Taylor Heinicke	2	8	65	NaN	-1	NaN	NaN	NaN	

`passResult` is null, but `hadRushAttempt` is 1 in each case, and all pass related fields are null. Looks like these should be marked as "scrambles", but let's go back and look at the play descriptions to make sure.

In [12]:
```
# First, get the rows with null values for passResult
null_rows = qb_pressure_df[qb_pressure_df['passResult'].isnull()]

# Then merge with plays_df to get matching play descriptions
plays_df = pd.read_csv(plays_file)
results = pd.merge(
    null_rows[['gameId', 'playId']],
    plays_df[['gameId', 'playId', 'playDescription']],
    on=['gameId', 'playId'])

print(results['playDescription'].values[:3])
```

```
['(5:00) (Shotgun) M.Mariota left end ran ob at CLV 11 for -2 yards (D.Ward).'
 '(6:44) J.Fields right end to CHI 30 for no gain (M.Sweat).'
 '(2:20) (Shotgun) T.Heinicke right tackle to WAS 44 for -1 yards (D.Campbell).']
```

We would have been wrong to drop these null value rows. Looking at the play descriptions, they're all scrambles, so let's override `passResult` with "R". With just 3 null values to worry about, a manual adjustment was not a problem, but as we will see for the "no pressure" dataframe below, we will need to handle these null values more systematically for greater numbers.

In [13]:
```
null_indices = null_rows.index
qb_pressure_df.loc[null_indices, 'passResult'] = 'R'
```

Let's also make sure there are no conflicts in the `qbSneak`, `qbSpike`, `qbKneel` indicator fields. We can't have multiple "True" values across these 3 fields for a single play. Since they're mixed type, let's fill the NaN values in the object (True/False) columns with False and set as integer. `qbKneel` is already an integer field.

Thankfully, we have no conflicts.

In [14]:
```
sneak = qb_pressure_df['qbSneak'].fillna(False).astype(int)
spike = qb_pressure_df['qbSpike'].fillna(False).astype(int)
kneel = qb_pressure_df['qbKneel']

# Sum across these converted values - any row with sum > 1 has multiple True values
multiple_trues = qb_pressure_df[sneak + spike + kneel > 1]

# Show how many such rows exist
print(f"Number of rows with multiple True values: {len(multiple_trues)}")
```

```
Number of rows with multiple True values: 0
```

From `passResult`, we know what's obviously a pass (complete passes, incomplete passes, interceptions) and what's obviously a scramble. Technically, we also know that a QB sack must be an intended pass. This is per NFL's own definition - in order to be considered a sack, the QB must intend to throw a forward pass. If the QB scrambles and gets tackled behind the line of scrimmage, that play is recorded as a "tackle for loss". Let's make sure there are no QB sacks recorded as QB scrambles.

In [15]:
```
len(qb_pressure_df[(qb_pressure_df['passResult']=='S') & (qb_pressure_df['hadRushAttempt']==1)])
```

Out[15]:
```
0
```

Let's check the "no pressure" dataframe as well. We see that there are >6k null values for `passResult` in the "no pressure" dataframe. Let's see what's going on there...

In [16]:
```
qb_no_pressure_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12836 entries, 0 to 12835
Data columns (total 19 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   gameId                  12836 non-null  int64
 1   playId                  12836 non-null  int64
 2   game_play_id            12836 non-null  object
 3   scoreDelta              12836 non-null  int64
 4   displayName             12836 non-null  object
 5   down                    12836 non-null  int64
 6   yardsToGo               12836 non-null  int64
 7   absoluteYardlineNumber  12836 non-null  int64
 8   passResult              6471 non-null   object
 9   prePenaltyYardsGained   12836 non-null  int64
 10  passLength              6234 non-null   float64
 11  timeToThrow             6213 non-null   float64
 12  timeInTackleBox         6214 non-null   float64
 13  hadRushAttempt          12836 non-null  int64
 14  rushingYards            12836 non-null  int64
 15  qbSneak                 6598 non-null   object
 16  qbKneel                 12836 non-null  int64
 17  qbSpike                 6238 non-null   object
 18  passingYards            12836 non-null  int64
dtypes: float64(3), int64(11), object(5)
memory usage: 1.9+ MB
```

In [17]:
```
qb_no_pressure_df[qb_no_pressure_df['passResult'].isnull()].sample(5)
```

Out[17]:

	gameId	playId	game_play_id	scoreDelta	displayName	down	yardsToGo	absoluteYardlineNumber	passResult	prePenaltyYardsGained	passLength	timeToThrow	timeInTackleBox	had
4357	2022100903	3292	2022100903 3292	0	Davis Mills	1	2	108	NaN	1	NaN	NaN	NaN	
4791	2022091800	96	2022091800 96	-7	Tua Tagovailoa	1	10	85	NaN	7	NaN	NaN	NaN	
2284	2022110601	556	2022110601 556	-4	Justin Fields	3	1	44	NaN	1	NaN	NaN	NaN	
10931	2022091808	3583	2022091808 3583	13	Jimmy Garoppolo	4	1	11	NaN	1	NaN	NaN	NaN	
6369	2022100903	688	2022100903 688	0	Trevor Lawrence	1	10	78	NaN	9	NaN	NaN	NaN	

When not under pressure, the QB has a lot more options, including as-designed running plays. For our analysis, we only care about the "no pressure" plays for comparative purposes. So let's write a rule-based cleanup function to override null value `passResult` rows where applicable and to set a new column called `playType` which determines whether each play was intended to be a passing play or a QB scramble. We are not interested in running plays not involving the QB. We can also use our function on the "pressure" dataframe. The hierarchy of rules are as follows:

- Drop all QB kneels, spikes and sneak plays, since these are not relevant for our analysis of "pressure" situations, given these decisions are taken before the QB is aware of any potential defensive pressure
- If `passResult` is "Complete pass", "Incomplete pass", Interception" or "Sack", then `playType` is "Pass"
- Otherwise, if `passResult` is "Scramble", then so is `playType`
- If `hadRushAttempt` equals 1, then both `passResult` and `playType` is "Scramble"
- Finally, all remaining rows should be dropped, as these are not QB-centric plays (i.e., not ones where the QB is either passing or scrambling)

In [18]:
```
def set_playType(df):
    '''
    Processes dataframe with QB plays to eliminate plays with null passResult values,
    create more descriptive passResult field and populate playType field ("Scramble" vs. "Pass").

    Args:
        df (pd.DataFrame): DataFrame with QB plays

    Returns:
        pd.DataFrame: Cleaned DataFrame with QB plays and populated playType
    '''

    # Drop QB kneel, spike and sneak plays, as these are not relevant to our analysis
    df = df[~((df['qbKneel'] == 1) | (df['qbSpike'] == True) | (df['qbSneak'] == True))]

    # Create playType field with "Unknown" as default value
    df['playType'] = 'Unknown'

    # Set Pass plays for passResult values of C, I, IN and S
    df.loc[df['passResult'].isin(['C', 'I', 'IN', 'S']), 'playType'] = 'Pass'

    # Set Scramble plays for passResult = R or hadRushAttempt = 1
    # If hadRushAttempt = 1, we should override null passResult values with R
    df.loc[df['passResult'] == 'R', 'playType'] = 'Scramble'
    df.loc[df['hadRushAttempt'] == 1, 'playType'] = 'Scramble'
    df.loc[df['hadRushAttempt'] == 1, 'passResult'] = 'R'

    # Finally, drop the rows with Unknown playType
    # (which also drops the null passResult rows that we cannot identify as a QB play)
    df = df[df['playType'] != 'Unknown']
```

```
    # Let's map passResult to more descriptive values
    passResult_mapping = {
        'C': 'Complete Pass',
        'I': 'Incomplete Pass',
        'S': 'QB Sack',
        'R': 'Scramble',
        'IN': 'Interception'}
    df['passResultLong'] = df['passResult'].map(passResult_mapping).fillna('Unknown')

    return df
```

In [19]:
```
qb_pressure_df = set_playType(qb_pressure_df)
qb_no_pressure_df = set_playType(qb_no_pressure_df)
```

We are left with 6,800 no-pressure plays involving the QB (vs. the original 12,836).

In [20]:
```
qb_no_pressure_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 6800 entries, 0 to 12833
Data columns (total 21 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   gameId                  6800 non-null   int64
 1   playId                  6800 non-null   int64
 2   game_play_id            6800 non-null   object
 3   scoreDelta              6800 non-null   int64
 4   displayName             6800 non-null   object
 5   down                    6800 non-null   int64
 6   yardsToGo               6800 non-null   int64
 7   absoluteYardlineNumber  6800 non-null   int64
 8   passResult              6800 non-null   object
 9   prePenaltyYardsGained   6800 non-null   int64
 10  passLength              6214 non-null   float64
 11  timeToThrow             6213 non-null   float64
 12  timeInTackleBox         6214 non-null   float64
 13  hadRushAttempt          6800 non-null   int64
 14  rushingYards            6800 non-null   int64
 15  qbSneak                 585 non-null    object
 16  qbKneel                 6800 non-null   int64
 17  qbSpike                 6215 non-null   object
 18  passingYards            6800 non-null   int64
 19  playType                6800 non-null   object
 20  passResultLong          6800 non-null   object
dtypes: float64(3), int64(11), object(7)
memory usage: 1.1+ MB
```

Back to Table of Contents

Aggregate Level Analysis

We begin by looking at the mix of pass vs. scramble plays when the QB is under pressure vs. no pressure. We see that the majority of our dataset is made up of passing plays. This is as expected for the "pressure" dataset, since pressure situations occur when the QB dropsback for a pass and in a minority of the situations, he is forced to scramble. This is also as expected for the "no pressure" dataset, since we have purposefully excluded as-designed running plays. The interesting observation here is that QBs scramble less often when under pressure compared to "no pressure" situations.

In [21]:
```
# playType mix for pressured vs. unpressured plays
pressure_mix = qb_pressure_df['playType'].value_counts(normalize=True).multiply(100).round(1).reset_index().rename(columns={'proportion': 'Pressure'})
no_pressure_mix = qb_no_pressure_df['playType'].value_counts(normalize=True).multiply(100).round(1).reset_index().rename(columns={'proportion': 'No Pressure'})
combined_play_mix = pressure_mix.merge(no_pressure_mix, on='playType')
combined_play_mix
```

Out[21]:

	playType	Pressure	No Pressure
0	Pass	94.8	91.4
1	Scramble	5.2	8.6

We would like to visualize this data in a bar chart. We'll have more bar charts to create, so here's a function we can reuse later.

In [22]:

```
def create_bar_chart(data, x_col, y_col, text_col, title, y_max=None,
                     width=600, height=400, sort_order=None,
                     label_angle=0, label_size=12):
    """
    Create formatted bar chart

    Args:
        df (pandas DataFrame) : Data to plot
        x_col (str): Column name for x-axis categories
        y_col (str): Column name for y-axis values
        text_col (str): Column name for bar labels
        title (str): Chart title
        y_max (float): Optional maximum value for y-axis scale, will auto-scale by default
        width (int): Optional chart width in pixels (default 600)
        height (int): Optional chart height in pixels (default 400)
        sort_order (str): Optional parameter
        label_angle (int): Optional angle for x-axis labels (default 0)
        label_size (int): Optional font size for x-axis and data labels (default 12)

    Returns:
        altair.Chart: Formatted bar chart with text labels
    """

    # Base bar chart
    chart = alt.Chart(data).mark_bar().encode(
        x=alt.X(f'{x_col}:N',
                title=None,
                sort=sort_order,
                axis=alt.Axis(
                    labelAngle=label_angle,
                    labelFontSize=label_size)),
        y=alt.Y(f'{y_col}:Q',
                title=None,
                axis=None,
                scale=alt.Scale(domain=[0, data[y_col].max() * 1.1] if y_max is None else [0, y_max]))
    ).properties(
        width=width,
        height=height,
        title=title)

    # Text labels
    text = chart.mark_text(
        align='center',
        baseline='bottom',
        dy=-5,
        fontSize=label_size
    ).encode(
        text=alt.Text(f'{text_col}:N'))

    # Return combined chart and text
    return (chart + text).configure_view(
        strokeWidth=0)
```

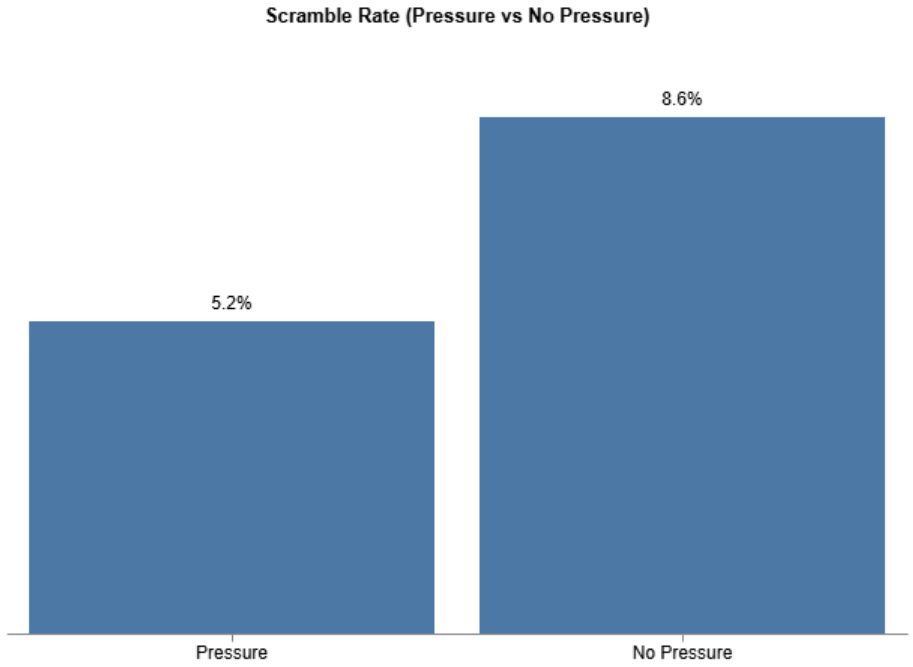
In [23]:
```
# Filter scramble rates and melt into long form
scramble_rates = combined_play_mix[combined_play_mix['playType'] == 'Scramble'].melt(
    id_vars=['playType'],
    value_vars=['Pressure', 'No Pressure'])
scramble_rates
```

Out[23]:

	playType	variable	value
0	Scramble	Pressure	5.2
1	Scramble	No Pressure	8.6

In [24]:
```
# Bar chart showing scramble rates with and without pressure
scramble_rates['label'] = scramble_rates['value'].astype(str) + '%'

create_bar_chart(
    data=scramble_rates,
    x_col='variable',
    y_col='value',
    text_col='label',
    title='Scramble Rate (Pressure vs No Pressure)',
    y_max=10)
```

Out[24]:



An interesting decision a QB makes when under pressure is whether to scramble or not. Let's look at scramble rates by which down it is, but to have a point of comparison, let's do it for both "pressure" and "no pressure" plays. We use our `create_bar_chart` function to help visualize for each dataset. We see that QBs are most likely to scramble on 3rd down plays. This is even more evident in the "no pressure" dataset.

In [25]:
```
# Create dataframe of scramble rates by down between pressure and no pressure plays
pressure_play_mix_by_down = (qb_pressure_df.groupby('down')['playType'].value_counts(normalize=True)
                             .unstack(fill_value=0).multiply(100).round(1))
no_pressure_play_mix_by_down = (qb_no_pressure_df.groupby('down')['playType'].value_counts(normalize=True)
                                .unstack(fill_value=0).multiply(100).round(1))
combined_scramble_by_down = pd.DataFrame({
    'Pressure': pressure_play_mix_by_down['Scramble'],
```

```
    'No Pressure': no_pressure_play_mix_by_down['Scramble']}).reset_index()
combined_scramble_by_down
```

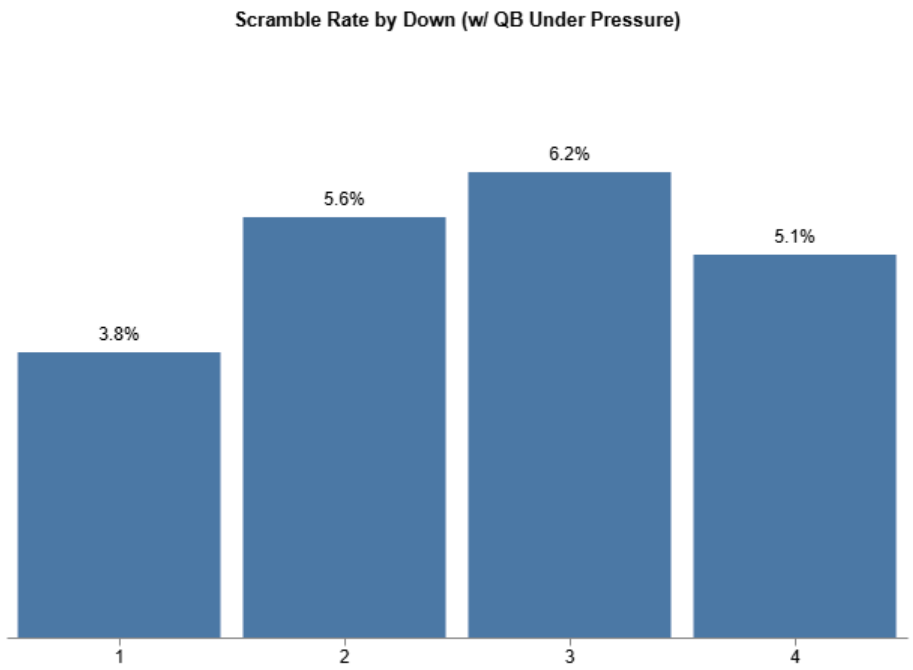
Out[25]:

	down	Pressure	No Pressure
0	1	3.8	8.0
1	2	5.6	8.1
2	3	6.2	10.6
3	4	5.1	6.8

In [26]:

```
# Bar chart showing scramble rates under pressure by down
combined_scramble_by_down['pressure_label'] = combined_scramble_by_down['Pressure'].astype(str) + '%'

create_bar_chart(
    data=combined_scramble_by_down,
    x_col='down',
    y_col='Pressure',
    text_col='pressure_label',
    title='Scramble Rate by Down (w/ QB Under Pressure)',
    y_max=8)
```

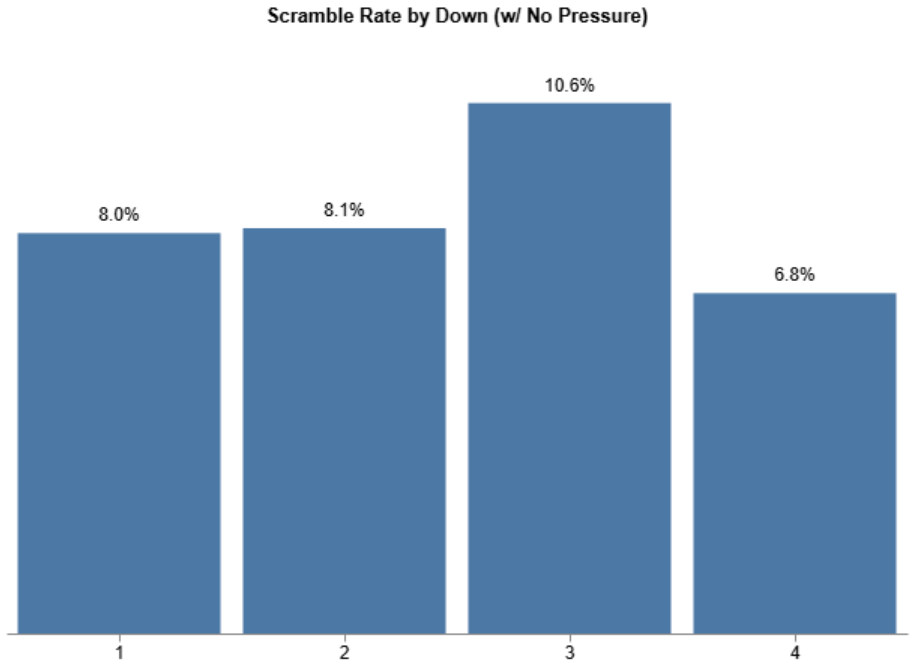
Out[26]:



In [27]:

```
# Bar chart showing scramble rates by down with no pressure
combined_scramble_by_down['no_pressure_label'] = combined_scramble_by_down['No Pressure'].astype(str) + '%'

create_bar_chart(
    data=combined_scramble_by_down,
    x_col='down',
    y_col='No Pressure',
    text_col='no_pressure_label',
    title='Scramble Rate by Down (w/ No Pressure)',
    y_max=12)
```

Out[27]:



The scramble rates for 4th downs are lower, which is odd. Normally, later downs should on average have fewer yards remaining to first down, making scrambles more likely to succeed. Presumably there are many fewer 4th down plays where the team on offense goes for 1st down. Let's confirm by calculating the number of plays and average yards remaining to first down by down. As suspected, much fewer 4th down plays in both datasets, making the lower average scramble rates seen above less meaningful, despite having fewer yards to first down.

In [28]:
```
# Stats for "pressure" plays
qb_pressure_df.groupby('down').agg({
    'game_play_id': 'count',
    'yardsToGo': 'mean'
}).rename(columns={'game_play_id': 'num_plays', 'yardsToGo': 'avg_yards_to_go'}).round(1)
```

Out[28]:

down	num_plays	avg_yards_to_go
1	1039	10.2
2	1024	8.7
3	1127	7.8
4	78	5.4

In [29]:
```
# Stats for "no pressure" plays
qb_no_pressure_df.groupby('down').agg({
    'game_play_id': 'count',
    'yardsToGo': 'mean'
}).rename(columns={'game_play_id': 'num_plays', 'yardsToGo': 'avg_yards_to_go'}).round(1)
```

Out[29]:

down	num_plays	avg_yards_to_go
1	2623	10.2
2	2446	8.4
3	1585	7.3
4	146	4.2

Next, let's look at passing plays and the likelihood of different outcomes, namely, completions, incomplete passes, interceptions and sacks. By filtering our `playType` equal to "Scramble", we effectively get the completion rate a QB is able to achieve "under pressure" vs. "no pressure". We focus our attention to completion rates, which is the most stark difference between the two datasets, with roughly 30%-points lower pass completion rate when under pressure.

In [30]:
```
# Pass result percentages (completion rate, etc.) -- need to filter for playType == "Pass" to exclude scrambles from denominator
pressure_passResult_df = (qb_pressure_df[qb_pressure_df['playType']=='Pass']['passResultLong']
                          .value_counts(normalize=True).multiply(100).round(1).reset_index().rename(columns={'proportion': 'Pressure'}))
no_pressure_passResult_df = (qb_no_pressure_df[qb_no_pressure_df['playType']=='Pass']['passResultLong']
                             .value_counts(normalize=True).multiply(100).round(1).reset_index().rename(columns={'proportion': 'No Pressure'}))
combined_passResult_mix = pressure_passResult_df.merge(no_pressure_passResult_df, on='passResultLong')
combined_passResult_mix
```

Out[30]:

	passResultLong	Pressure	No Pressure
0	Complete Pass	40.6	70.3
1	Incomplete Pass	37.4	27.8
2	QB Sack	19.6	0.0
3	Interception	2.4	1.9

It's interesting to see zero QB sacks when there is no pressure. Looks like there are only 2 plays that ended in a sack, which got rounded down to 0.0%. This is likely a function of how "pressure" data is collected (remember out `causedPressure` variable). By definition, a situation where the QB gets sacked, there must have been defensive pressure. These 2 plays are likely either errors in the original data or perhaps fluke plays that were recorded as a sack without the need for defensive pressure (e.g., if the QB slipped and was ultimately down by contact from a nearby defensive player).

In [31]:
```
qb_no_pressure_df[qb_no_pressure_df['passResult']=='S']
```

Out[31]:

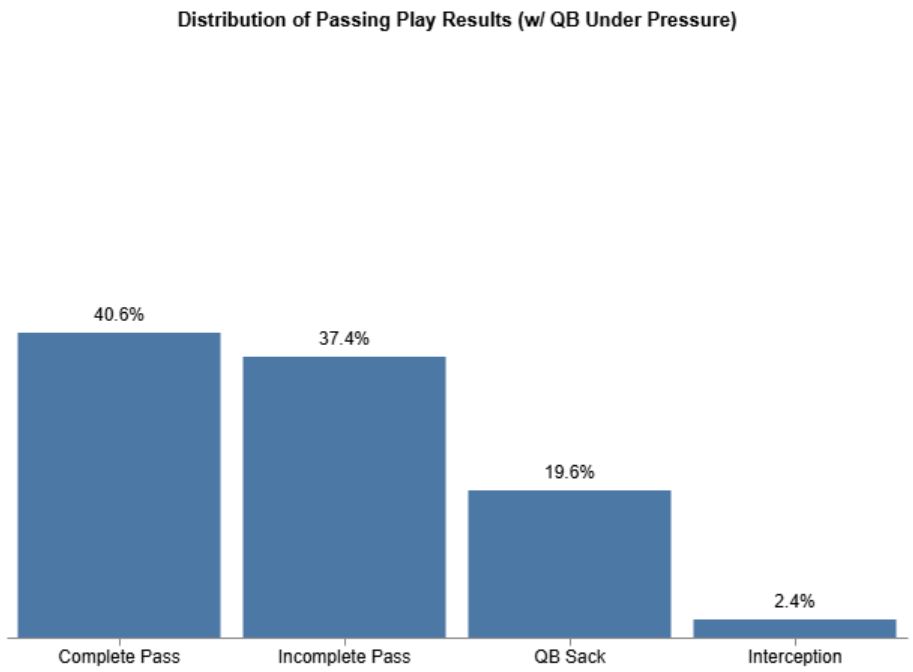
	gameId	playId	game_play_id	scoreDelta	displayName	down	yardsToGo	absoluteYardlineNumber	passResult	prePenaltyYardsGained	...	timeToThrow	timeInTackleBox	hadRushAtten
1826	2022101601	4049	2022101601 4049	-16	Jacoby Brissett	2	3	34	S	-2	...	NaN	NaN	
2083	2022100208	3876	2022100208 3876	-8	Justin Fields	2	4	43	S	0	...	NaN	2.3	

2 rows × 21 columns

In [32]:
```
# Bar chart showing distribution of play results under pressure
pressure_passResult_df['label'] = pressure_passResult_df['Pressure'].astype(str) + '%'

create_bar_chart(
    data=pressure_passResult_df,
    x_col='passResultLong',
    y_col='Pressure',
    text_col='label',
    title='Distribution of Passing Play Results (w/ QB Under Pressure)',
```

```
    y_max=80, # set to same scale as "No Pressure" chart. Use y_max=50 to make this chart look better as a standalone
    sort_order='-y'
)
```

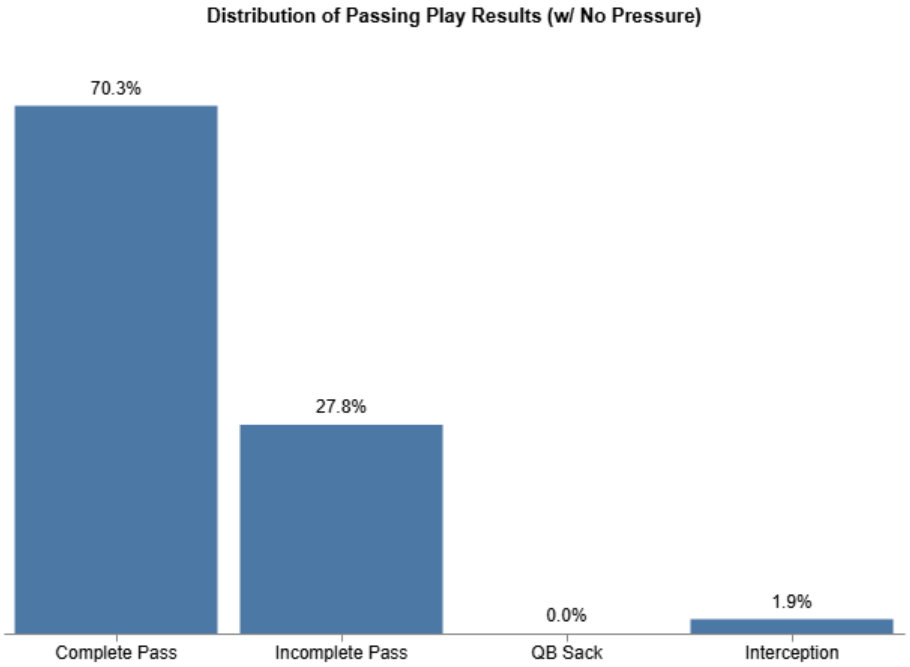
Out[32]:



In [33]:

```
# Bar chart showing distribution of play results with no pressure
no_pressure_passResult_df['label'] = no_pressure_passResult_df['No Pressure'].astype(str) + '%'

# Set same order as "Pressure" chart for ease of side-by-side comparison
custom_order = ['Complete Pass', 'Incomplete Pass', 'QB Sack', 'Interception']

create_bar_chart(
    data=no_pressure_passResult_df,
    x_col='passResultLong',
    y_col='No Pressure',
    text_col='label',
    title='Distribution of Passing Play Results (w/ No Pressure)',
    y_max=80,
    sort_order=custom_order # set sort_order='-y' for standalone chart with descending order values along x-axis
)
```

Out[33]:



Next, we look at some key statistics for each dataset, namely, yards gained (before any penalty effects), rushing vs. passing yards, average pass length (distance the ball is in the air, regardless of outcome), average time to throw and average time the QB spent in the tackle box (behind the line of scrimmage, without scrambling). One caveat here is that we need to use `prePenaltyYardsGained` to calculate average rushing yards and average passing yards (as opposed to `rushingYards` and `passingYards`, since these fields are often recorded as zeros for plays resulting in loss of yardage, whereas `prePenaltyYardsGained` also shows negative values).

We note that, on average, yards gained on "no pressure" plays are double the yards gained on "pressure" plays (which is probably not a surprise). Interestingly, scrambling when under pressure yields higher yardage gains vs. "no pressure" situations, perhaps because scrambles are very selectively used (remember, they are roughly 5% of the "pressure" plays) and the QB probably finds a bit more space once past the line of scrimmage, if he can get past the incoming pressure.

Despite this relative advantage in scrambling success (re-emphasizing the relevance of selection bias), the overall yardage gain for "pressure" plays is poor due to the much lower success in average passing yards. This, most certainly, has to do with the much lower completion rate we observed above.

There are also some interesting differences in QBs, on average, throwing slightly longer passes, after taking slightly longer when under pressure, but we'll need to investigate this further.

In [34]:
```
# Calculate average stats for Pressure vs. No Pressure plays
pressure_stats = pd.Series({
    'Play Count': qb_pressure_df['passResultLong'].count(),
    'Avg Pre-Penalty Yards': qb_pressure_df['prePenaltyYardsGained'].mean(),
    'Avg Rush Yards': qb_pressure_df.loc[qb_pressure_df['playType'] == 'Scramble', 'prePenaltyYardsGained'].mean(),
    'Avg Pass Yards': qb_pressure_df.loc[qb_pressure_df['playType'] == 'Pass', 'prePenaltyYardsGained'].mean(),
    'Avg Pass Length': qb_pressure_df.loc[qb_pressure_df['playType'] == 'Pass', 'passLength'].mean(),
    'Avg Time to Throw': qb_pressure_df.loc[qb_pressure_df['playType'] == 'Pass', 'timeToThrow'].mean(),
    'Avg Time in Tackle Box': qb_pressure_df.loc[qb_pressure_df['playType'] == 'Pass', 'timeInTackleBox'].mean()
}).round(1)

no_pressure_stats = pd.Series({
    'Play Count': qb_no_pressure_df['passResultLong'].count(),
    'Avg Pre-Penalty Yards': qb_no_pressure_df['prePenaltyYardsGained'].mean(),
    'Avg Rush Yards': qb_no_pressure_df.loc[qb_no_pressure_df['playType'] == 'Scramble', 'prePenaltyYardsGained'].mean(),
    'Avg Pass Yards': qb_no_pressure_df.loc[qb_no_pressure_df['playType'] == 'Pass', 'prePenaltyYardsGained'].mean(),
    'Avg Pass Length': qb_no_pressure_df.loc[qb_no_pressure_df['playType'] == 'Pass', 'passLength'].mean(),
    'Avg Time to Throw': qb_no_pressure_df.loc[qb_no_pressure_df['playType'] == 'Pass', 'timeToThrow'].mean(),
    'Avg Time in Tackle Box': qb_no_pressure_df.loc[qb_no_pressure_df['playType'] == 'Pass', 'timeInTackleBox'].mean()
```

```
}).round(1)

# Combine into single DataFrame
combined_stats = pd.DataFrame({'Pressure': pressure_stats, 'No Pressure': no_pressure_stats}).reset_index().rename(columns={'index': 'Stat'})
combined_stats
```

Out[34]:

	Stat	Pressure	No Pressure
0	Play Count	3268.0	6800.0
1	Avg Pre-Penalty Yards	3.7	7.4
2	Avg Rush Yards	7.4	6.1
3	Avg Pass Yards	3.5	7.5
4	Avg Pass Length	8.8	7.2
5	Avg Time to Throw	3.3	2.5
6	Avg Time in Tackle Box	3.1	2.4

Let's dig one level deeper and look at the same stats by different play results. A few things stand out. First, the stronger passing yards for "no pressure" plays indeed has to do with completion rate differences. Yards gained on completed passes are actually higher for "pressure" plays.

Second, the longer average length of pass for "pressure" plays really presents itself in plays that resulted in interceptions. The data suggests that QBs facing pressure should prioritize shorter passes.

Third, sacks have a distinctly longer average time in the tackle box relative to other outcomes. Another suggestion from this data would be that a sack becomes much more likely as QB time in the tackle box starts approacing 4 seconds.

In [35]:
```
# Stats for pressure plays
qb_pressure_df.groupby(['passResultLong']).agg(
    count=('passResultLong', 'count'),
    avg_yards=('prePenaltyYardsGained', 'mean'),
    avg_rush_yards=('rushingYards', 'mean'),
    avg_pass_yards=('passingYards', 'mean'),
    avg_pass_length=('passLength', 'mean'),
    avg_time_to_throw=('timeToThrow', 'mean'),
    avg_time_in_tackle_box=('timeInTackleBox', 'mean')).round(1).reset_index()
```

Out[35]:

	passResultLong	count	avg_yards	avg_rush_yards	avg_pass_yards	avg_pass_length	avg_time_to_throw	avg_time_in_tackle_box
0	Complete Pass	1257	12.0	0.0	12.0	6.9	3.1	2.9
1	Incomplete Pass	1160	0.0	0.0	0.0	10.3	3.5	3.1
2	Interception	75	0.0	0.0	0.0	17.9	3.6	3.1
3	QB Sack	606	-6.8	0.0	0.0	NaN	NaN	3.8
4	Scramble	170	7.4	7.4	0.0	NaN	NaN	NaN

In [36]:
```
# And also for no pressure plays
qb_no_pressure_df.groupby(['passResultLong']).agg(
    count=('passResultLong', 'count'),
    avg_yards=('prePenaltyYardsGained', 'mean'),
    avg_rush_yards=('rushingYards', 'mean'),
    avg_pass_yards=('passingYards', 'mean'),
    avg_pass_length=('passLength', 'mean'),
    avg_time_to_throw=('timeToThrow', 'mean'),
    avg_time_in_tackle_box=('timeInTackleBox', 'mean')).round(1).reset_index()
```

Out[36]:

	passResultLong	count	avg_yards	avg_rush_yards	avg_pass_yards	avg_pass_length	avg_time_to_throw	avg_time_in_tackle_box
0	Complete Pass	4367	10.6	0.0	10.6	5.4	2.4	2.4
1	Incomplete Pass	1728	0.0	0.0	0.0	11.3	2.7	2.6
2	Interception	118	0.0	0.0	0.0	13.9	2.8	2.7
3	QB Sack	2	-1.0	0.0	0.0	NaN	NaN	2.3
4	Scramble	585	6.1	6.2	0.0	0.0	NaN	NaN

Let's go back to looking at just pre-penalty yards gained by pass vs. scramble play types and "pressure" vs. "no pressure" plays.

In [37]:
```
# Calculate average yards for pressure plays
avg_yards_pressure = qb_pressure_df.groupby('playType')['prePenaltyYardsGained'].mean().round(1).reset_index()
avg_yards_pressure = avg_yards_pressure.rename(columns={'prePenaltyYardsGained': 'Pressure'})

# Calculate average yards for no pressure plays
avg_yards_no_pressure = qb_no_pressure_df.groupby('playType')['prePenaltyYardsGained'].mean().round(1).reset_index()
avg_yards_no_pressure = avg_yards_no_pressure.rename(columns={'prePenaltyYardsGained': 'No Pressure'})

# Merge the two dataframes
avg_yards_df = avg_yards_pressure.merge(avg_yards_no_pressure, on='playType')
avg_yards_df
```

Out[37]:

	playType	Pressure	No Pressure
0	Pass	3.5	7.5
1	Scramble	7.4	6.1

In [38]:
```
# Melt into long format for Altair
avg_yards_df_long = avg_yards_df.melt(
    id_vars=['playType'],
    var_name='group',
    value_name='avgYards')
avg_yards_df_long
```

Out[38]:

	playType	group	avgYards
0	Pass	Pressure	3.5
1	Scramble	Pressure	7.4
2	Pass	No Pressure	7.5
3	Scramble	No Pressure	6.1

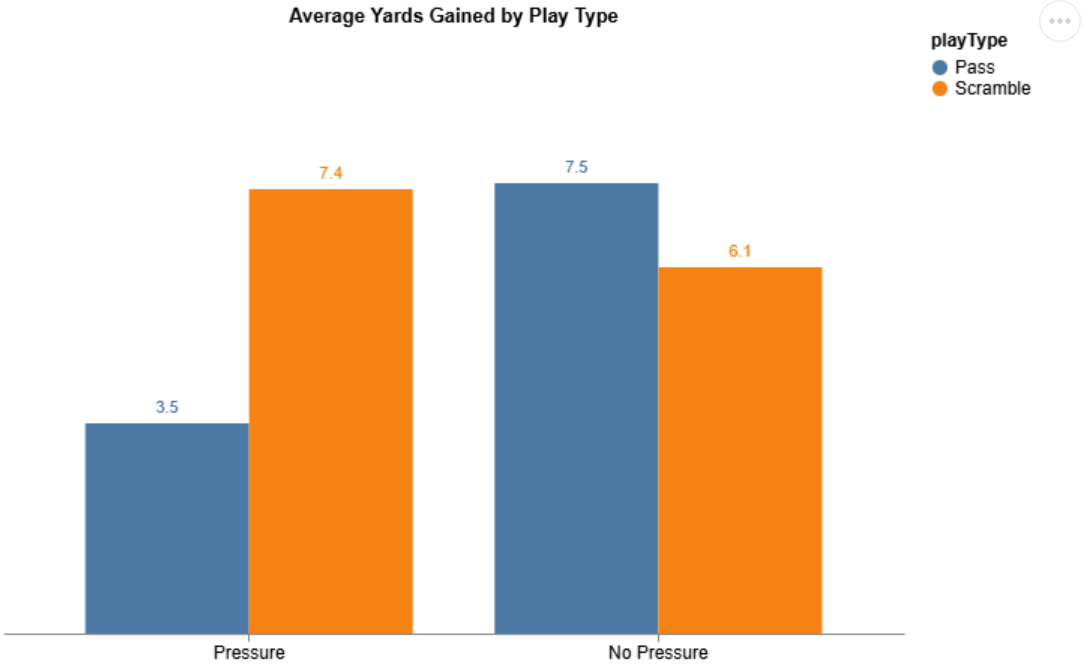
In [39]:
```
# Want to put this in a grouped bar chart, which we may need to repeat, so here's a function we can reuse
def create_grouped_bar_chart(data, category_col, group_col, value_col, title, y_max=None, width=600, height=400, sort_order=None):
    """
    Create formatted grouped bar chart

    Args:
        df (pandas DataFrame) : Data to plot in long format
        category_col (str): Column name for main categories (e.g., Pass vs. Scramble)
        group_col (str): Column name for group names (e.g., Pressure vs. No Pressure)
        value_col (str): Column name for y-axis values
        title (str): Chart title
        y_max (float): Optional maximum value for y-axis scale, will auto-scale by default
        width (int): Optional chart width in pixels (default 600)
```

```
        height (int): Optional chart height in pixels (default 400)
        sort_order (str): Optional lsit for sorting groups

    Returns:
        altair.Chart: Formatted grouped bar chart with text labels
    """

    # Base grouped bar chart
    chart = alt.Chart(data).mark_bar().encode(
        x=alt.X(f'{group_col}:N',
                title=None,
                sort=sort_order,
                axis=alt.Axis(
                    labelAngle=0,
                    labelFontSize=12)),
        y=alt.Y(f'{value_col}:Q',
                title=None,
                axis=None,
                scale=alt.Scale(domain=[0, y_max]) if y_max else alt.Scale(zero=True)),
        xOffset=f'{category_col}:N',
        color=alt.Color(f'{category_col}:N',
                        scale=alt.Scale(),
                        legend=alt.Legend(title=category_col, labelFontSize=12, titleFontSize=12))
    ).properties(
        width=width,
        height=height,
        title=title)

    # Text labels
    text = chart.mark_text(
        align='center',
        baseline='bottom',
        dy=-5
    ).encode(
        text=alt.Text(f'{value_col}:Q', format='.1f'))

    # Return combined chart and text
    return (chart + text).configure_view(
        strokeWidth=0)
```

In [40]:

```
create_grouped_bar_chart(
    data=avg_yards_df_long,
    category_col='playType',
    group_col='group',
    value_col='avgYards',
    title='Average Yards Gained by Play Type',
    y_max=10)
```

Out[40]:



Let's also look at some pre-snap statistics that may have influence on QB decisions, such as the average down, average yards to first down, the score delta (where a negative value denotes the QB's team is behind) and average yards to endzone.

In [41]:
```
# Pre-snap stats for pressure plays
qb_pressure_df.groupby('playType').agg(
    count=('playType', 'count'),
    avg_down=('down', 'mean'),
    avg_yards_to_go=('yardsToGo', 'mean'),
    avg_score_delta=('scoreDelta', 'mean'),
    avg_yardline=('absoluteYardlineNumber', 'mean')).round(1)
```

Out[41]:

playType	count	avg_down	avg_yards_to_go	avg_score_delta	avg_yardline
Pass	3098	2.1	8.8	-2.8	60.6
Scramble	170	2.2	8.8	-3.2	62.1

In [42]:
```
# Pre-snap stats for no pressure plays
qb_no_pressure_df.groupby('playType').agg(
    count=('playType', 'count'),
    avg_down=('down', 'mean'),
    avg_yards_to_go=('yardsToGo', 'mean'),
    avg_score_delta=('scoreDelta', 'mean'),
    avg_yardline=('absoluteYardlineNumber', 'mean')).round(1)
```

Out[42]:

playType	count	avg_down	avg_yards_to_go	avg_score_delta	avg_yardline
Pass	6215	1.9	8.9	-2.7	60.2
Scramble	585	2.0	7.7	-0.9	61.0

No immediate insights jump out from just looking at averages, so let's see how some of this data is distributed.

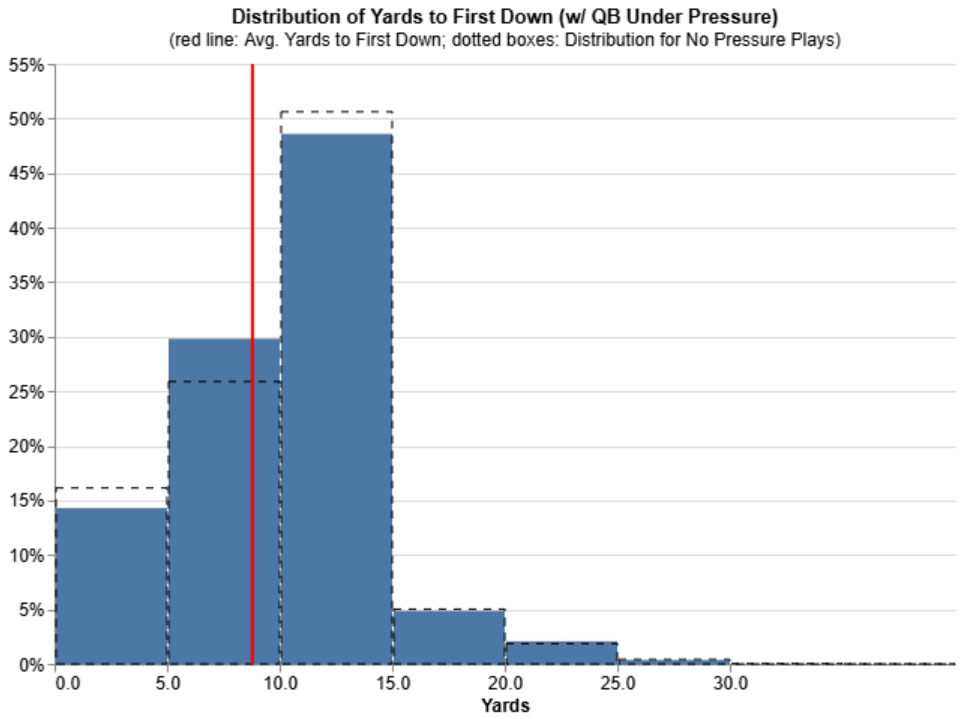
In [43]:
```
# Let's create some visualizations for some of this data, but first a histogram function we can reuse
def create_histogram(data, value_col, title, subtitle=None, x_title=None, data2=None, width=600, height=400, bins=None):
    """
    Create formatted histogram, with optional overlay of second dataset

    Args:
        data (pandas DataFrame): Primary data to plot
        value_col (str): Column name for values to distribute
        title (str): Chart title
        subtitle (str): Optional chart subtitle
        x_title (str): Optional x-axis title
        data2 (pandas DataFrame): Optional secondary data to plot
        bins (int): Number of bins for histogram (default None, will calculate optimal)
        width (int): Optional chart width in pixels (default 600)
        height (int): Optional chart height in pixels (default 400)

    Returns:
        altair.Chart: Formatted histogram with optional overlay of secondary histogram
    """

    # Base histogram
    hist1 = alt.Chart(data).transform_joinaggregate(
        total='count()'
    ).transform_calculate(
        pct='1/datum.total'
    ).mark_bar().encode(
        x=alt.X(f'{value_col}:Q',
                bin=alt.Bin(maxbins=bins) if bins else True,
                title=x_title if x_title else value_col,
                axis=alt.Axis(labelFontSize=12)),
        y=alt.Y('sum(pct):Q',
                title=None,
                axis=alt.Axis(labelFontSize=12, format='%'),
                stack=None))

    # Mean line
    mean_line = alt.Chart(data).mark_rule(
        color='red',
        strokeWidth=2
    ).encode(
        x=alt.X(f'mean({value_col}):Q'),
        size=alt.value(2),
        color=alt.value('red'))

    # Add overlay histogram if data2 passed through
    if data2 is not None:
        hist2 = alt.Chart(data2).transform_joinaggregate(
            total='count()'
        ).transform_calculate(
            pct='1/datum.total'
        ).mark_bar(
            fillOpacity=0,
            stroke='black',
            strokeDash=[5, 5]
        ).encode(
            x=alt.X(f'{value_col}:Q',
                    bin=alt.Bin(maxbins=bins) if bins else True),
            y=alt.Y('sum(pct):Q', stack=None))

        combined_chart = alt.layer(hist1, hist2, mean_line)
    else:
```

```
        combined_chart = alt.layer(hist1, mean_line)

    # Return combined histogram with mean line and overlay
    return combined_chart.properties(
        width=width,
        height=height,
        title=alt.Title(text=title, subtitle=subtitle) if subtitle else title
    ).configure_view(strokeWidth=0).configure_axis(titleFontSize=12)
```

As with the averages, no major differences in how the "pressure" vs. "no pressure" datasets are distributed in terms of the different pre-snap stats we listed above.

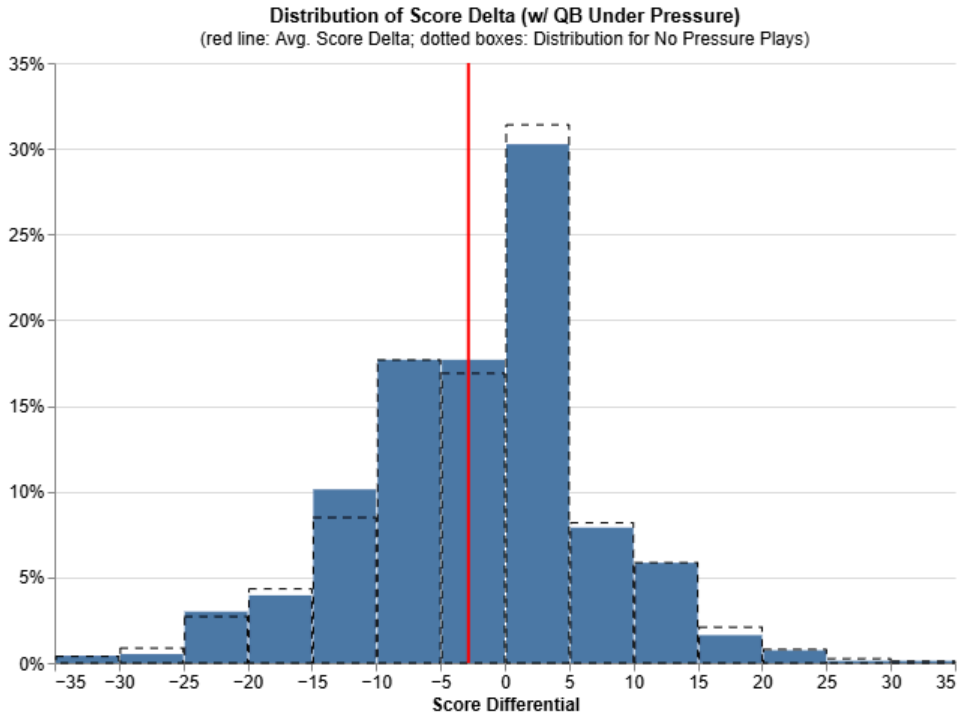
In [44]:
```
create_histogram(
    data=qb_pressure_df,
    value_col='yardsToGo',
    title='Distribution of Yards to First Down (w/ QB Under Pressure)',
    subtitle='(red line: Avg. Yards to First Down; dotted boxes: Distribution for No Pressure Plays)',
    x_title='Yards',
    data2=qb_no_pressure_df,
    bins=10)
```

Out[44]:



In [45]:
```
create_histogram(
    data=qb_pressure_df,
    value_col='scoreDelta',
    title='Distribution of Score Delta (w/ QB Under Pressure)',
    subtitle='(red line: Avg. Score Delta; dotted boxes: Distribution for No Pressure Plays)',
    x_title='Score Differential',
    data2=qb_no_pressure_df,
    bins=20)
```

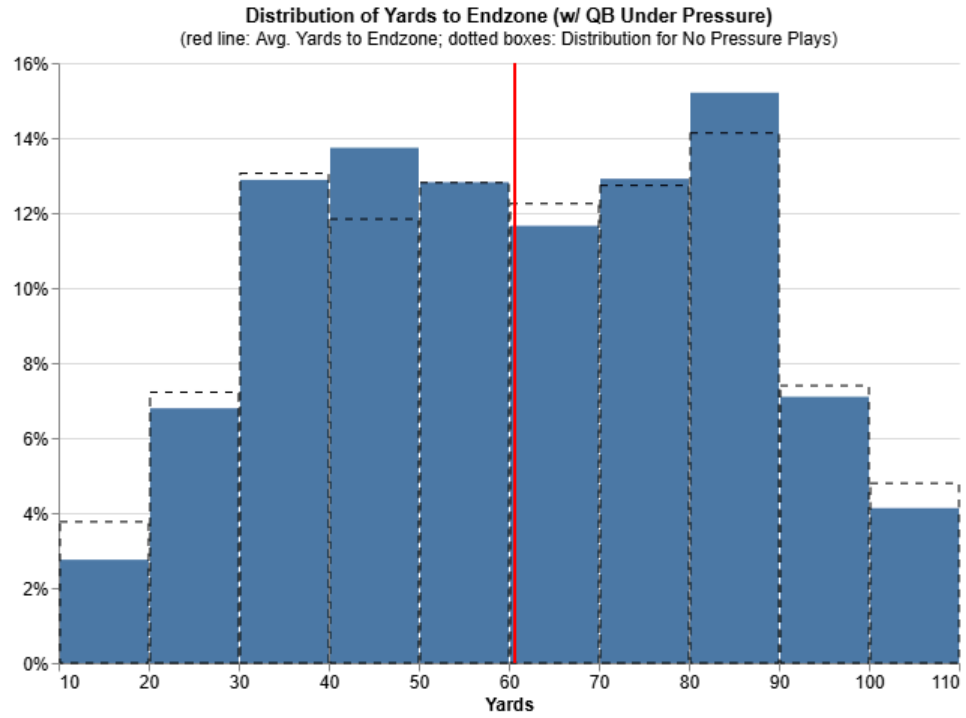
Out[45]:



In [46]:

```
create_histogram(
    data=qb_pressure_df,
    value_col='absoluteYardlineNumber',
    title='Distribution of Yards to Endzone (w/ QB Under Pressure)',
    subtitle='(red line: Avg. Yards to Endzone; dotted boxes: Distribution for No Pressure Plays)',
    x_title='Yards',
    data2=qb_no_pressure_df,
    bins=10)
```

Out[46]:



While not necessarily a pre-snap statistic, length of pass was one of the variables that seemed to have a meaningful impact on the success of a play, so let's dive deeper into the variable `passLength`.

In [47]:

```
# Calculate stats for pressure plays
pressure_passLength_stats = qb_pressure_df[qb_pressure_df['playType']=='Pass'].agg({
    'game_play_id': 'count',
    'passLength': 'mean'
}).to_frame().T.rename(columns={'game_play_id': 'num_plays', 'passLength': 'avg_pass_length'})

# Calculate stats for no pressure plays
no_pressure_passLength_stats = qb_no_pressure_df[qb_no_pressure_df['playType']=='Pass'].agg({
    'game_play_id': 'count',
    'passLength': 'mean'
}).to_frame().T.rename(columns={'game_play_id': 'num_plays', 'passLength': 'avg_pass_length'})

# Combine them into single DataFrame
passLength_comparison_df = pd.DataFrame({
    'Pressure': pressure_passLength_stats.iloc[0],
    'No Pressure': no_pressure_passLength_stats.iloc[0]
}).round(1)

passLength_comparison_df
```

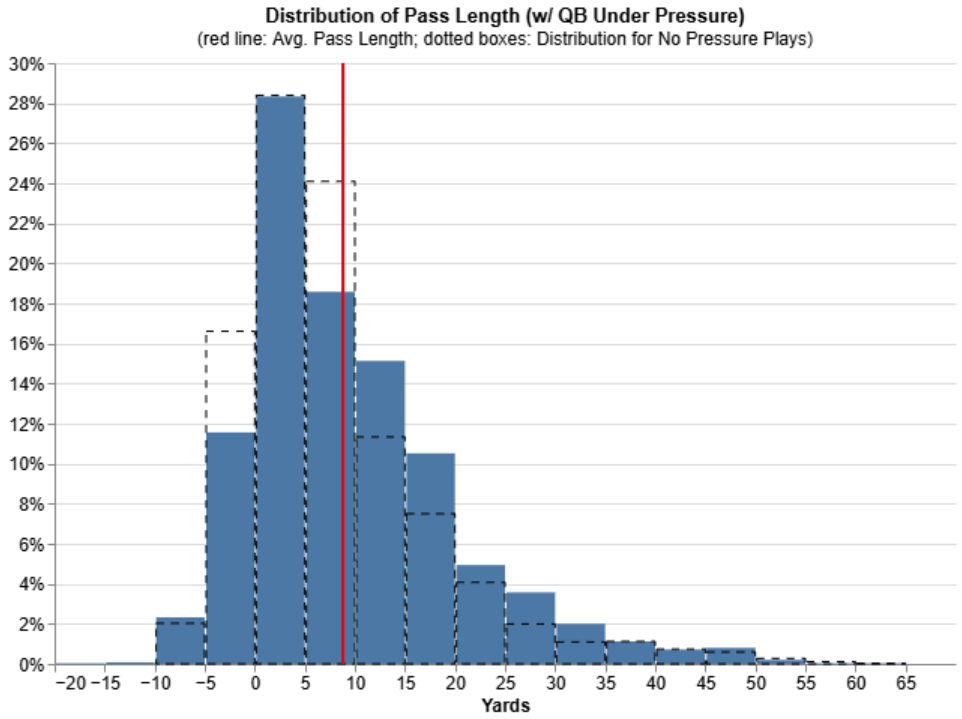
Out[47]:

	Pressure	No Pressure
num_plays	3098.0	6215.0
avg_pass_length	8.8	7.2

The average length of pass a QB chooses when under pressure is 20% longer vs. the average for "no pressure" plays. This warrants further investigation. We can also see from the distribution of pass length data that QBs under pressure tend to opt for longer passes.

In [48]:
```
# Let's look at how passLength is distributed. Need to pass through passing plays that are not sacks
create_histogram(
    data=qb_pressure_df[(qb_pressure_df['playType']=='Pass') & (qb_pressure_df['passResult']!='S')],
    value_col='passLength',
    title='Distribution of Pass Length (w/ QB Under Pressure)',
    subtitle='(red line: Avg. Pass Length; dotted boxes: Distribution for No Pressure Plays)',
    x_title='Yards',
    data2=qb_no_pressure_df[(qb_no_pressure_df['playType']=='Pass') & (qb_no_pressure_df['passResult']!='S')],
    bins=20)
```

Out[48]:



As we noted above, even more interesting is the pass length stats for interceptions, with intercepted pass attempts under pressure coming after attempted pass lengths that are 4 yards longer than the average for "no pressure" plays. This is also evident in the distribution of pass length for interceptions, where "no pressure" interceptions show higher density for pass attempts 15 yards or shorter, whereas QBs "under pressure" get intercepted more often on passes 20 yards or longer.

In [49]:
```
# Calculate stats for pressure plays
pressure_passLength_stats_int = qb_pressure_df[qb_pressure_df['passResult']=='IN'].agg({
    'game_play_id': 'count',
    'passLength': 'mean'
}).to_frame().T.rename(columns={'game_play_id': 'num_plays', 'passLength': 'avg_pass_length'})

# Calculate stats for no pressure plays
no_pressure_passLength_stats_int = qb_no_pressure_df[qb_no_pressure_df['passResult']=='IN'].agg({
    'game_play_id': 'count',
    'passLength': 'mean'
}).to_frame().T.rename(columns={'game_play_id': 'num_plays', 'passLength': 'avg_pass_length'})

# Combine them into single DataFrame
passLength_int_comparison_df = pd.DataFrame({
    'Pressure': pressure_passLength_stats_int.iloc[0],
    'No Pressure': no_pressure_passLength_stats_int.iloc[0]
}).round(1)
```

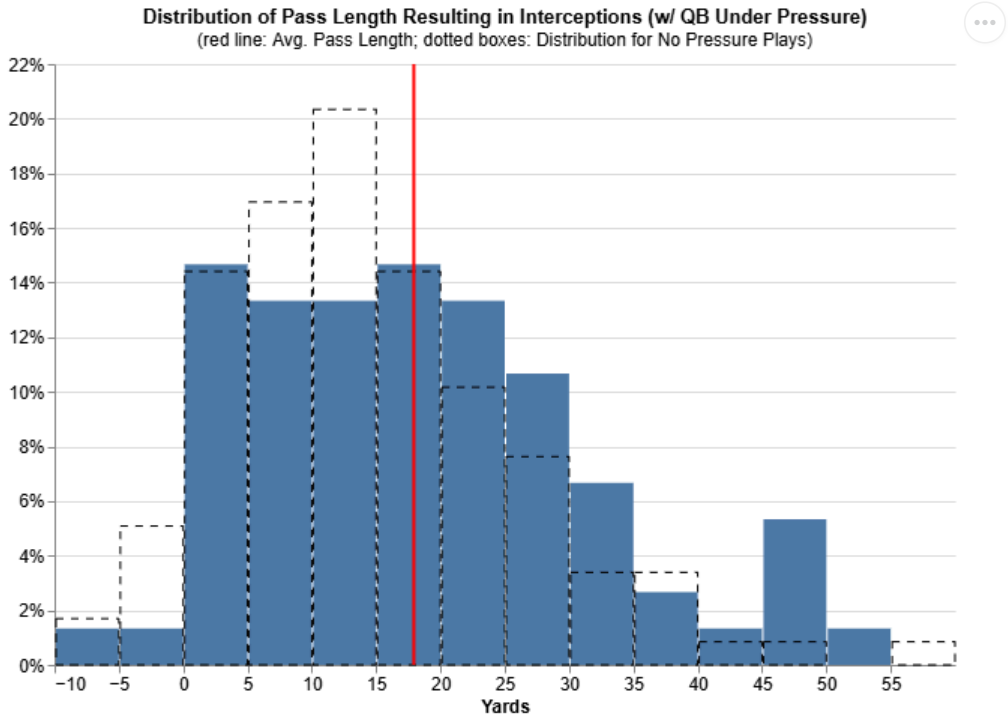
```
passLength_int_comparison_df
```

Out[49]:

	Pressure	No Pressure
num_plays	75.0	118.0
avg_pass_length	17.9	13.9

In [50]:
```
# Here's how passLength is distributed for all interceptions
create_histogram(
    data=qb_pressure_df[qb_pressure_df['passResult']=='IN'],
    value_col='passLength',
    title='Distribution of Pass Length Resulting in Interceptions (w/ QB Under Pressure)',
    subtitle='(red line: Avg. Pass Length; dotted boxes: Distribution for No Pressure Plays)',
    x_title='Yards',
    data2=qb_no_pressure_df[qb_no_pressure_df['passResult']=='IN'],
    bins=20)
```

Out[50]:



Back to Table of Contents

Individual Player Analysis

Our next task is to analyze player level statistics. We begin by creating a summary table of the stats we're interested in by QB. We make use of our `create_QB_summary` function to get summary dataframes from each of the "pressure" and "no pressure" datasets.

In [51]:
```
# Function to create player-by-player summary for a specific passResult
def create_QB_summary(df, pass_result, yards_col=None):
    """
    Creates summary QB statistics for a specific passResult

    Args:
```

```
        df (pandas DataFrame): QB play data
        pass_result (str): passResult string to filter ('C', S', 'IN', 'R')
        yards_col (str): Name of the yards column to average (optional)

    Returns:
        pd.DataFrame: DataFrame with summary statistics for the specified passResult
    """

    # Get total snaps
    total_snaps_df = (df.groupby('displayName').agg(
        total_snaps=('displayName', 'size'),
        avg_total_yards=('prePenaltyYardsGained', 'mean')
    ).round(1).reset_index().sort_values('total_snaps', ascending=False))

    # Get filtered data for specific passResult
    result_df = df[df['passResult'] == pass_result]

    # Create summary stats
    summary_df = result_df.groupby('displayName').agg({'passResult': 'count'})

    # Add average yards stat (if specified)
    if yards_col:
        yards_df = result_df.groupby('displayName')[yards_col].mean().round(1)
        summary_df = summary_df.join(yards_df)

    # Reset index and rename columns
    summary_df = summary_df.reset_index()
    summary_df = summary_df.rename(columns={'passResult': 'count'})
    if yards_col:
        summary_df = summary_df.rename(columns={yards_col: 'avg_yards'})

    # Merge with total snaps and fill NAs
    summary_df = pd.merge(total_snaps_df, summary_df, on='displayName', how='outer').fillna(0)
    summary_df['count'] = summary_df['count'].astype(int)

    # Calculate rate
    summary_df['rate'] = (summary_df['count'] / summary_df['total_snaps'] *100).round(1)

    # Sort by total_snaps and rate (descending order)
    summary_df = summary_df.sort_values(['total_snaps', 'rate'], ascending=[False, False])

    return summary_df
```

In [52]:

```
# Create summaries for each passResult of interest
completions_pressure = create_QB_summary(qb_pressure_df, 'C', 'prePenaltyYardsGained')
scrambles_pressure = create_QB_summary(qb_pressure_df, 'R', 'prePenaltyYardsGained')
interceptions_pressure = create_QB_summary(qb_pressure_df, 'IN')
sacks_pressure = create_QB_summary(qb_pressure_df, 'S', 'prePenaltyYardsGained')

# Rename columns
completions_pressure = completions_pressure.rename(columns={'count': 'completion_count', 'rate': 'completion_rate', 'avg_yards': 'avg_yards_per_completed_pass'})
scrambles_pressure = scrambles_pressure.rename(columns={'count': 'scramble_attempts', 'rate': 'scramble_rate', 'avg_yards': 'avg_yards_per_scramble'})
interceptions_pressure = interceptions_pressure.rename(columns={'count': 'int_count', 'rate': 'interception_rate'})
sacks_pressure = sacks_pressure.rename(columns={'count': 'sack_count', 'rate': 'sack_rate', 'avg_yards': 'avg_yards_lost_per_sack'})

# Merge all summaries
qb_summary_pressure = completions_pressure.merge(scrambles_pressure).merge(interceptions_pressure).merge(sacks_pressure)
qb_summary_pressure.head()
```

Out[52]:

	displayName	total_snaps	avg_total_yards	completion_count	avg_yards_per_completed_pass	completion_rate	scramble_attempts	avg_yards_per_scramble	scramble_rate	int_count	interception_ra
0	Kirk Cousins	125	3.8	51	11.6	40.8	2	3.0	1.6	3	
1	Justin Herbert	121	4.8	61	9.9	50.4	5	5.0	4.1	1	
2	Matthew Stafford	117	3.1	53	10.0	45.3	2	4.0	1.7	2	
3	Derek Carr	115	5.1	49	13.6	42.6	4	11.0	3.5	1	
4	Matt Ryan	114	3.4	54	10.6	47.4	1	10.0	0.9	4	

In [53]:
```
# Let's do the same for No Pressure plays
completions_no_pressure = create_QB_summary(qb_no_pressure_df, 'C', 'prePenaltyYardsGained')
scrambles_no_pressure = create_QB_summary(qb_no_pressure_df, 'R', 'prePenaltyYardsGained')
interceptions_no_pressure = create_QB_summary(qb_no_pressure_df, 'IN')
sacks_no_pressure = create_QB_summary(qb_no_pressure_df, 'S', 'prePenaltyYardsGained')

# Rename columns
completions_no_pressure = completions_no_pressure.rename(columns={'count': 'completion_count', 'rate': 'completion_rate', 'avg_yards': 'avg_yards_per_completed_pass'})
scrambles_no_pressure = scrambles_no_pressure.rename(columns={'count': 'scramble_attempts', 'rate': 'scramble_rate', 'avg_yards': 'avg_yards_per_scramble'})
interceptions_no_pressure = interceptions_no_pressure.rename(columns={'count': 'int_count', 'rate': 'interception_rate'})
sacks_no_pressure = sacks_no_pressure.rename(columns={'count': 'sack_count', 'rate': 'sack_rate', 'avg_yards': 'avg_yards_lost_per_sack'})

# Merge all summaries
qb_summary_no_pressure = completions_no_pressure.merge(scrambles_no_pressure).merge(interceptions_no_pressure).merge(sacks_no_pressure)
qb_summary_no_pressure.head()
```

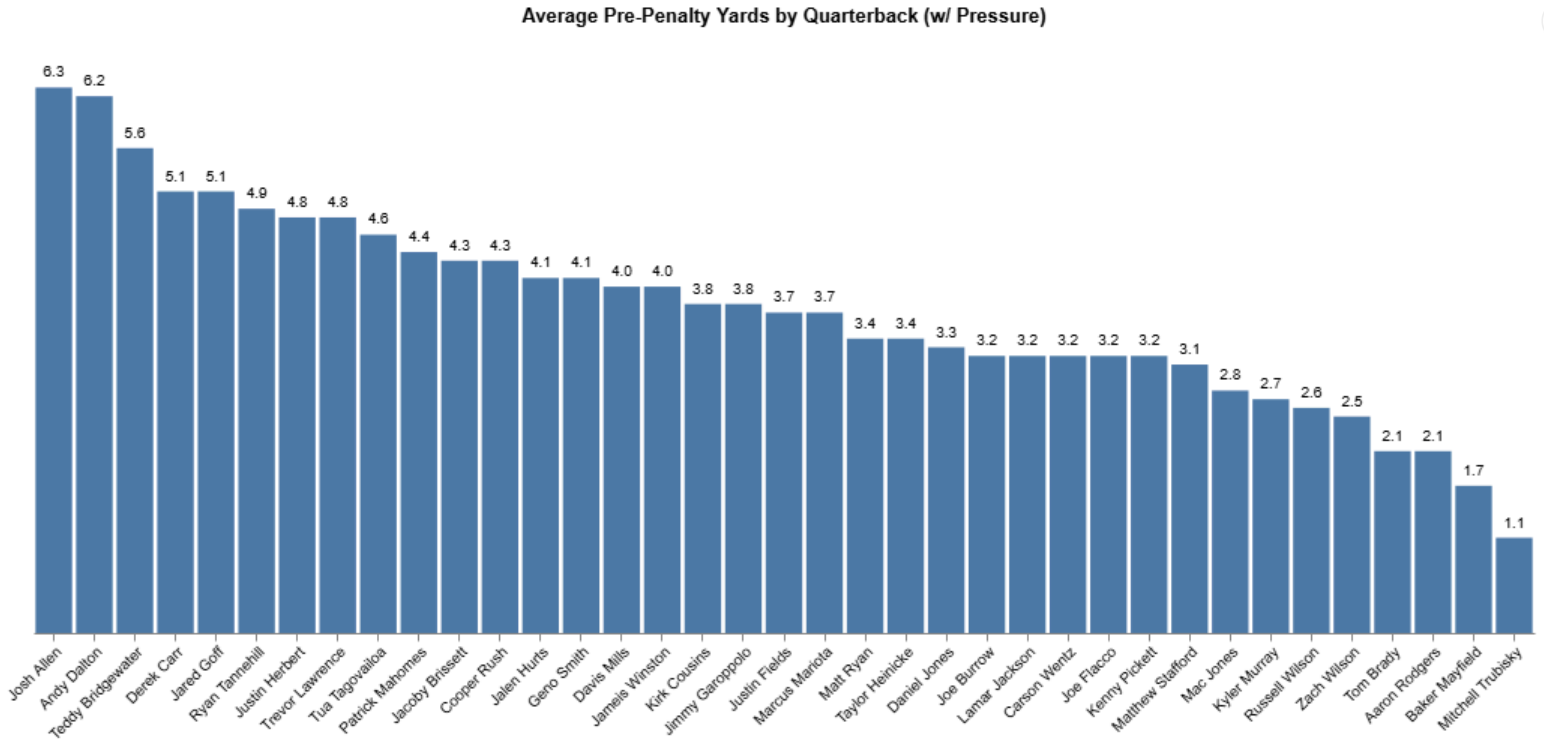
Out[53]:

	displayName	total_snaps	avg_total_yards	completion_count	avg_yards_per_completed_pass	completion_rate	scramble_attempts	avg_yards_per_scramble	scramble_rate	int_count	interception_ra
0	Tom Brady	305	7.0	214	10.0	70.2	1	0.0	0.3	1	
1	Kyler Murray	298	6.4	191	8.9	64.1	38	5.4	12.8	1	
2	Joe Burrow	255	7.8	177	10.9	69.4	7	8.9	2.7	6	
3	Patrick Mahomes	255	8.4	176	11.3	69.0	16	9.3	6.3	3	
4	Aaron Rodgers	252	6.9	173	9.9	68.7	8	3.1	3.2	5	

Let's look at the QBs that deliver the most yards gained when under pressure, but for the benefit of expressiveness and effectiveness of our visualization, we will only include QBs who have had more than 30 snaps during the season.

In [54]:
```
qb_summary_pressure_filtered = qb_summary_pressure[qb_summary_pressure['total_snaps'] > 30]
qb_summary_pressure_filtered['label'] = qb_summary_pressure_filtered['avg_total_yards'].apply(lambda x: f'{x:.1f}')
create_bar_chart(
    data=qb_summary_pressure_filtered,
    x_col='displayName',
    y_col='avg_total_yards',
    text_col='label',
    title='Average Pre-Penalty Yards by Quarterback (w/ Pressure)',
    width=1000,
    height=400,
    sort_order='-y',
    label_angle=-45,
    label_size=10)
```

Out[54]:

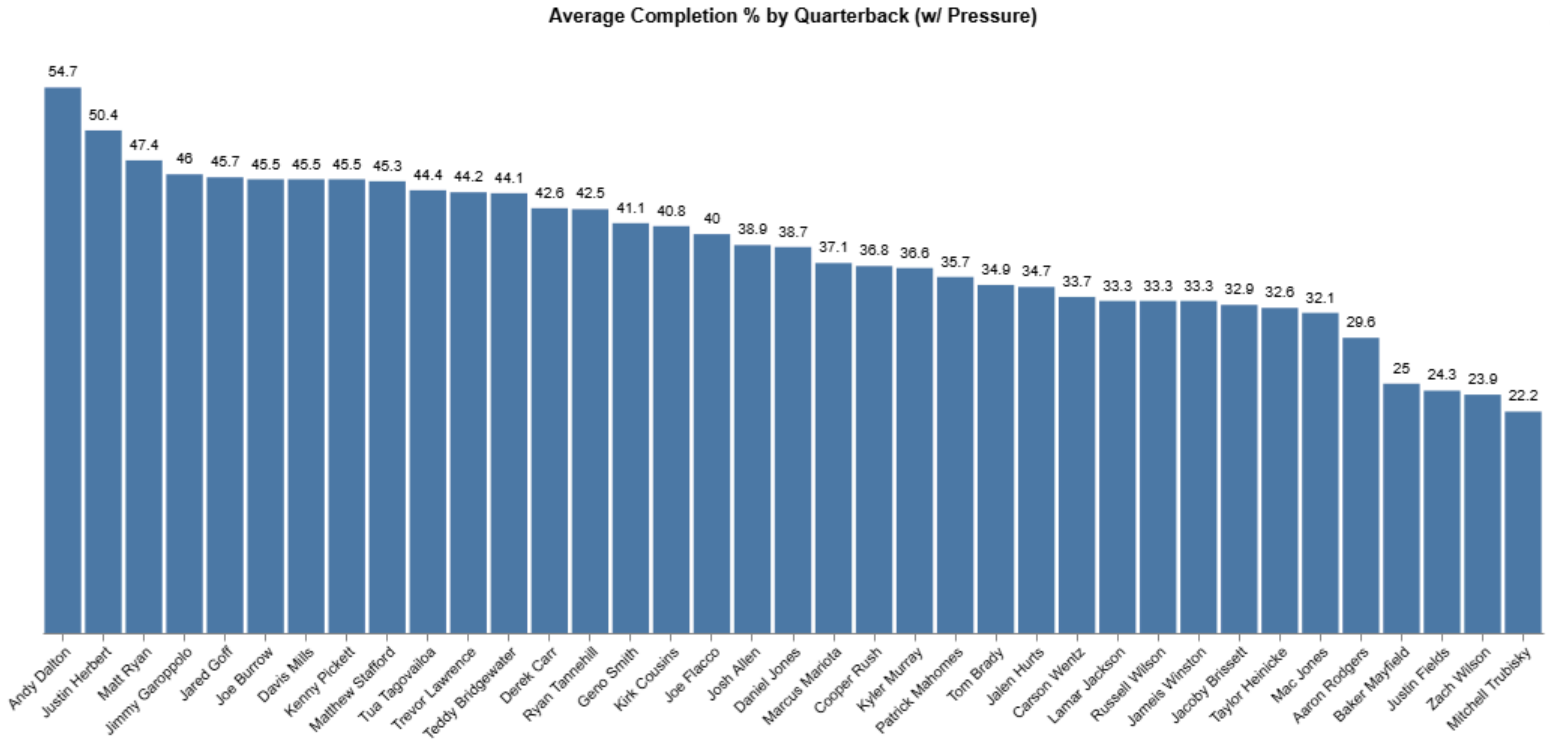


Next, let's look at the QBs with the most completed passes under pressure, again, including only the QBs with more than 30 snaps during the season.

In [55]:

```
# Let's also look at the QBs with the highest completion rate under pressure
create_bar_chart(
    data=qb_summary_pressure_filtered,
    x_col='displayName',
    y_col='completion_rate',
    text_col='completion_rate',
    title='Average Completion % by Quarterback (w/ Pressure)',
    width=1000,
    height=400,
    sort_order='-y',
    label_angle=-45,
    label_size=10)
```

Out[55]:



Looking carefully at the 2 bar charts, we can see that QBs can achieve high yardage under pressure without necessarily depending on their passing game. Josh Allen is a good example. He tops the table for yards gained under pressure, even though he's middle of the pack when it comes to completed passes. Andy Dalton, on the other hand, is at or near the top for both. Let's create some scatter plots from this data, but first a scatter function we can reuse.

In [56]:

```python
# Scatter plot function we can reuse
def create_scatter(data, x_col, y_col, label_col=None, tooltip_col=None, title=None, x_title=None, y_title=None,
                   y_min=None, y_max=None, x_min=None, x_max=None, width=600, height=400):
    """
    Create Altair scatter plot with optional text labels

    Args:
        data (pandas DataFrame): Data to plot
        x_col (str): Column name for x-axis data
        y_col (str): Column name for y-axis data
        label_col (str): Optional column name for text labels
        tooltip_col (list): Optional column names to include in tooltip
        title (str): Optional chart title
        x_title (str): Optional x-axis title
        y_title (str): Optional y-axis title
        y_min (float): Optional min value for y-axis scale, will auto-scale by default
        y_max (float): Optional max value for y-axis scale, will auto-scale by default
        x_min (float): Optional min value for x-axis scale, will auto-scale by default
        x_max (float): Optional max value for x-axis scale, will auto-scale by default
        width (int): Optional chart width in pixels (default 600)
        height (int): Optional chart height in pixels (default 400)

    Returns:
        altair.Chart: Scatter plot with optional text labels
    """

    # Set default tooltip columns
    if tooltip_col is None:
```

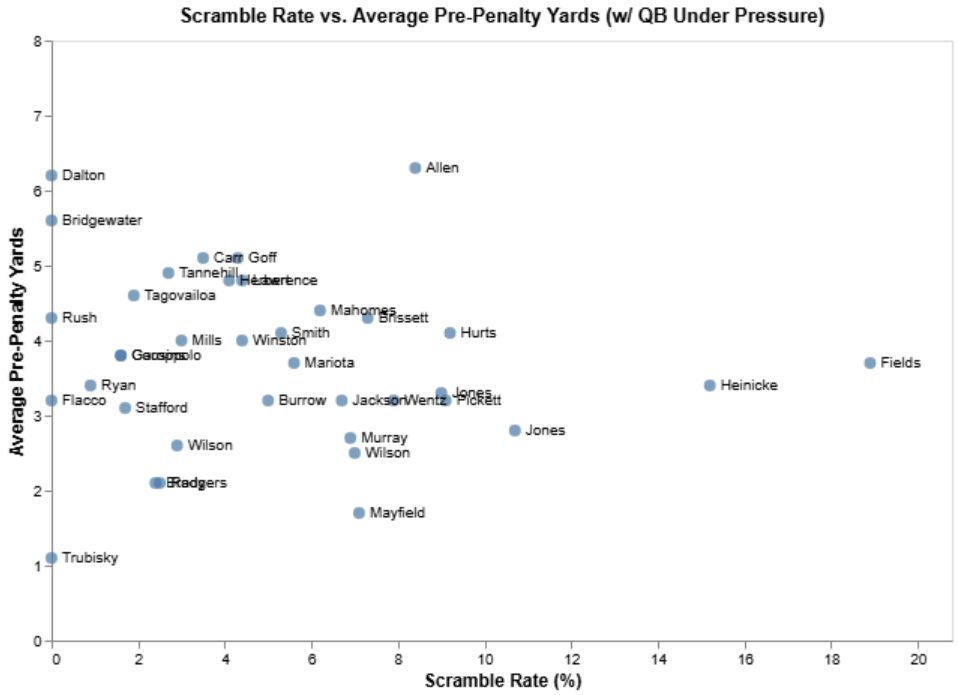
```
    if label_col is None:
        tooltip_col = [x_col, y_col]
    else:
        tooltip_col = [label_col, x_col, y_col]

# Selection filter
hover = alt.selection_single(
    on='mouseover',
    nearest=True,
    empty=False,
    clear='mouseout'
)

# Set axis ranges
x_scale = alt.Scale(
    domain=[data[x_col].min() * 0.9 if x_min is None else x_min,
            data[x_col].max() * 1.1 if x_max is None else x_max])

y_scale = alt.Scale(
    domain=[data[y_col].min() * 0.9 if y_min is None else y_min,
            data[y_col].max() * 1.1 if y_max is None else y_max])

# Base scatter plot
scatter = alt.Chart(data).mark_circle(size=60).encode(
    x=alt.X(f'{x_col}:Q',
            title=x_title if x_title else x_col,
            axis=alt.Axis(labelFontSize=10, tickCount=10),
            scale=x_scale),
    y=alt.Y(f'{y_col}:Q',
            title=y_title if y_title else y_col,
            axis=alt.Axis(labelFontSize=10, tickCount=10),
            scale=y_scale),
    tooltip=tooltip_col,
    color=alt.condition(hover, alt.value('red'), alt.value('#4c78a8'))
).add_selection(hover)

# Add text labels if passed through
if label_col is None:
    combined_chart = scatter
else:
    text = scatter.mark_text(
        align='left',
        baseline='middle',
        dx=7,
        fontSize=10
    ).encode(
        text=f'{label_col}:N',
        color=alt.condition(hover, alt.value('red'), alt.value('black'))
    )
    combined_chart = (scatter + text)

return combined_chart.properties(
    width=width,
    height=height,
    title=title if title else f'{x_col} vs. {y_col}'
).configure_axis(grid=False, titleFontSize=12)
```

Since scrambling is a key decision that QBs tend to make very selectively when under pressure, let's see how scramble rates for individual QBs correlate with yards gained. We see from the initial chart below, that there is unfortunately not a very clear correlation between them. If you're viewing this Notebook in Jupyter, you can hover over individual datapoints to see more detail on each QB.

In [57]:
```
# For labelling, let's just pass through QB last names
qb_summary_pressure_filtered['last_name'] = qb_summary_pressure_filtered['displayName'].str.split().str[-1]
```

In [58]:
```
# Scatter plot showing Scramble Rate vs. Avg. Pre-Penalty Yards by QB
scatter = create_scatter(
    data=qb_summary_pressure_filtered,
    x_col='scramble_rate',
    y_col='avg_total_yards',
    label_col='last_name',
    tooltip_col=['displayName', 'total_snaps', 'scramble_rate', 'avg_total_yards'],
    title='Scramble Rate vs. Average Pre-Penalty Yards (w/ QB Under Pressure)',
    x_title='Scramble Rate (%)',
    y_title='Average Pre-Penalty Yards',
    y_min=0,
    y_max=8
)
scatter
```

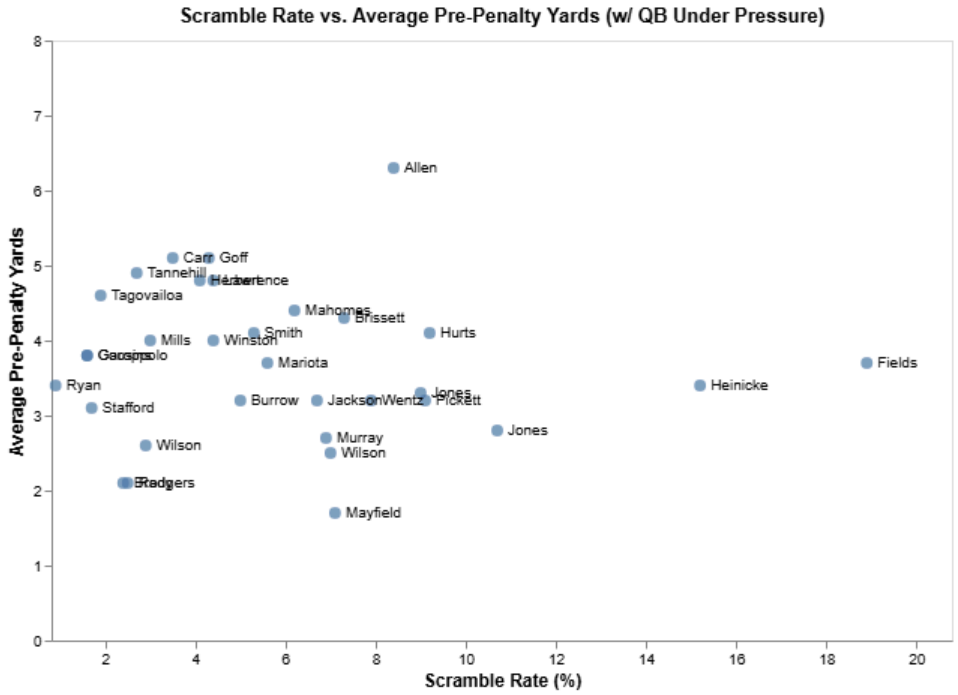
Out[58]:



There are some outliers in the data, specifically, the 5 QBs with zero scrambles in the season. We should really exclude those from the scatter plot. It would also be nice to see how a linear regression line fits the data, so we make those 2 improvements below.

In [59]:
```
# We should filter out QBs with zero scramble attempts for this plot
qb_summary_pressure_filtered_scramble = qb_summary_pressure_filtered[qb_summary_pressure_filtered['scramble_attempts'] > 0]
scatter = create_scatter(
    data=qb_summary_pressure_filtered_scramble,
    x_col='scramble_rate',
    y_col='avg_total_yards',
    label_col='last_name',
    tooltip_col=['displayName', 'total_snaps', 'scramble_rate', 'avg_total_yards'],
    title='Scramble Rate vs. Average Pre-Penalty Yards (w/ QB Under Pressure)',
    x_title='Scramble Rate (%)',
    y_title='Average Pre-Penalty Yards',
    y_min=0,
    y_max=8
)
scatter
```

Out[59]:



In [60]:

```python
# Function to add linear fit line
def get_linear_fit_chart(data, x_col, y_col, opacity=0.5):
    '''
    Fits sklearn.linear_model.LinearRegression model to x_col to predict y_col
    and returns an Altair chart plotting the fit line

    Args:
        data (pandas DataFrame): Data to use in linear regression line fit
        x_col (str): Column name for x-axis data
        y_col (str): Column name for y-axis data
        opacity (float): Opacity of output line

    Returns:
        altair.Chart: Linear fit line plot
    '''

    # Fit model to data
    X = data[[f'{x_col}']]
    y = data[f'{y_col}']
    model = LinearRegression().fit(X, y)

    # construct prediction grid for x_col with 101 evenly-spaced values
    y_pred_grid = pd.DataFrame({f'{x_col}': np.linspace(
        data[f'{x_col}'].min(),
        data[f'{x_col}'].max(),
        num=101
    )})

    #use model to predict value for y_col at each x_col position
    pred_df = pd.DataFrame({
            f'{x_col}': y_pred_grid[f'{x_col}'],
            f'{y_col}': model.predict(y_pred_grid)
        })

    # return Altair chart showing the fit line using `pred_df`
```

```
    return alt.Chart(pred_df).mark_line(
        color="red",
        opacity=opacity
    ).encode(
        x=f'{x_col}',
        y=f'{y_col}'
    )
```

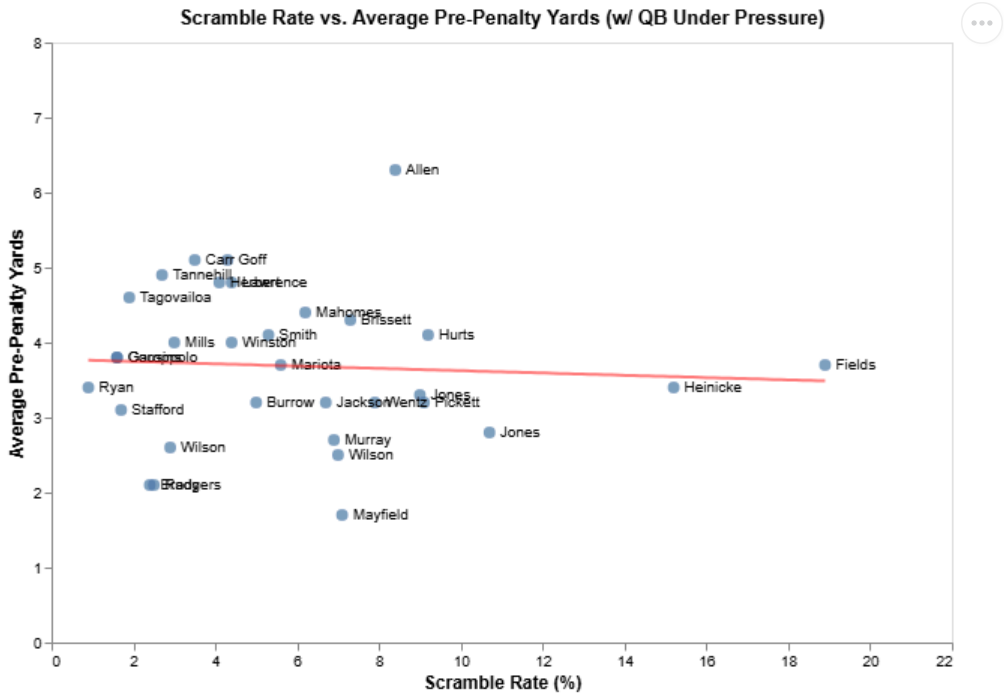
Our conclusion has not changed. There is only a very slight negative correlation between the two variables, as also observed in the slightly negative sloping linear regression fit line.

In [61]:
```
print(qb_summary_pressure_filtered_scramble[['scramble_rate', 'avg_total_yards']].corr().round(2))
```

```
                scramble_rate  avg_total_yards
scramble_rate            1.00            -0.06
avg_total_yards         -0.06             1.00
```

In [62]:
```
line_fit = get_linear_fit_chart(qb_summary_pressure_filtered_scramble, 'scramble_rate', 'avg_total_yards')
scatter + line_fit
```

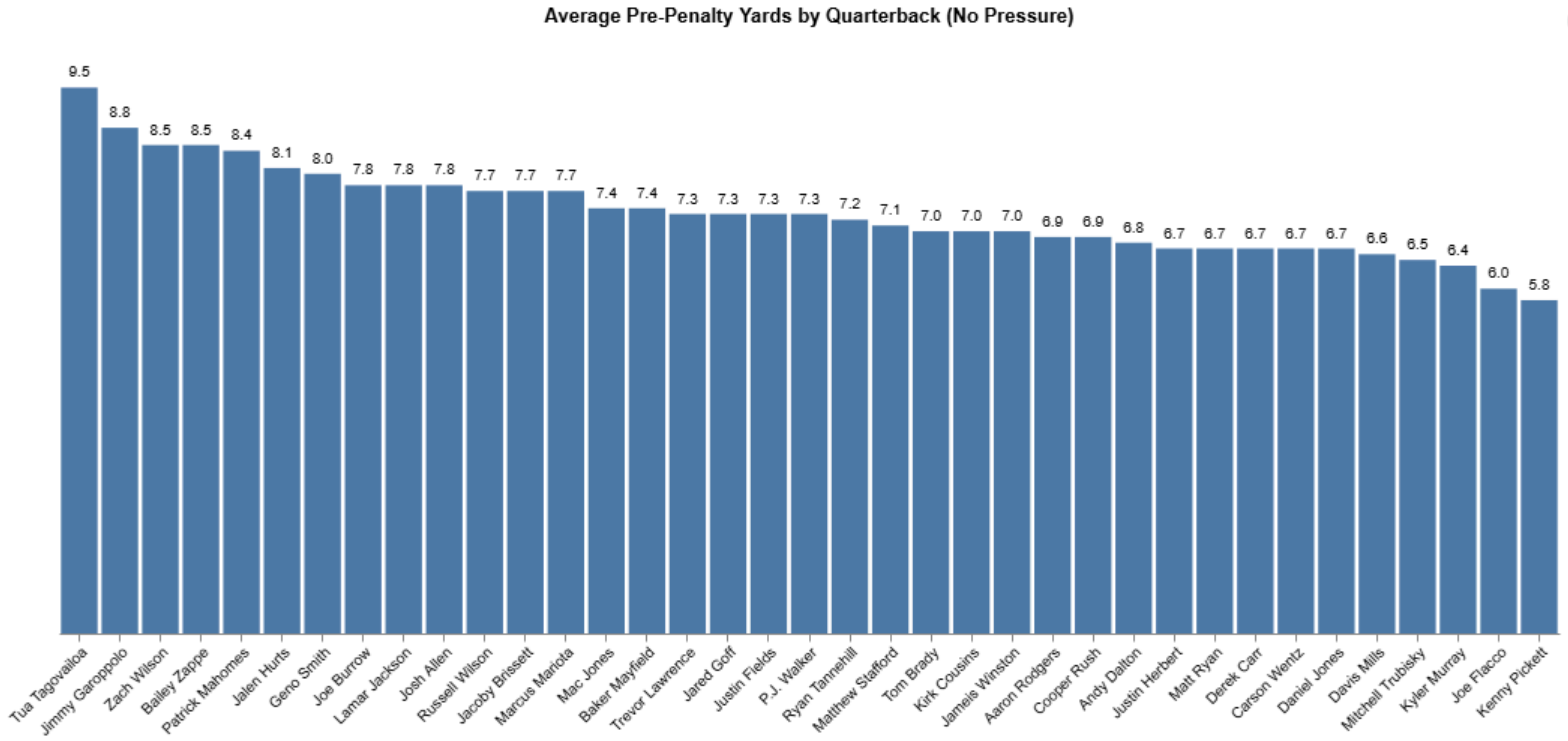
Out[62]:



Now let's look at plays with "no pressure" and plot yards gained by QB, again focusing on those who have had more than 30 snaps.

In [63]:
```
qb_summary_no_pressure_filtered = qb_summary_no_pressure[qb_summary_pressure['total_snaps'] > 30]
qb_summary_no_pressure_filtered['label'] = qb_summary_no_pressure_filtered['avg_total_yards'].apply(lambda x: f'{x:.1f}')
create_bar_chart(
    data=qb_summary_no_pressure_filtered,
    x_col='displayName',
    y_col='avg_total_yards',
    text_col='label',
    title='Average Pre-Penalty Yards by Quarterback (No Pressure)',
    width=1000,
    height=400,
    sort_order='-y',
    label_angle=-45,
    label_size=10)
```

Out[63]:

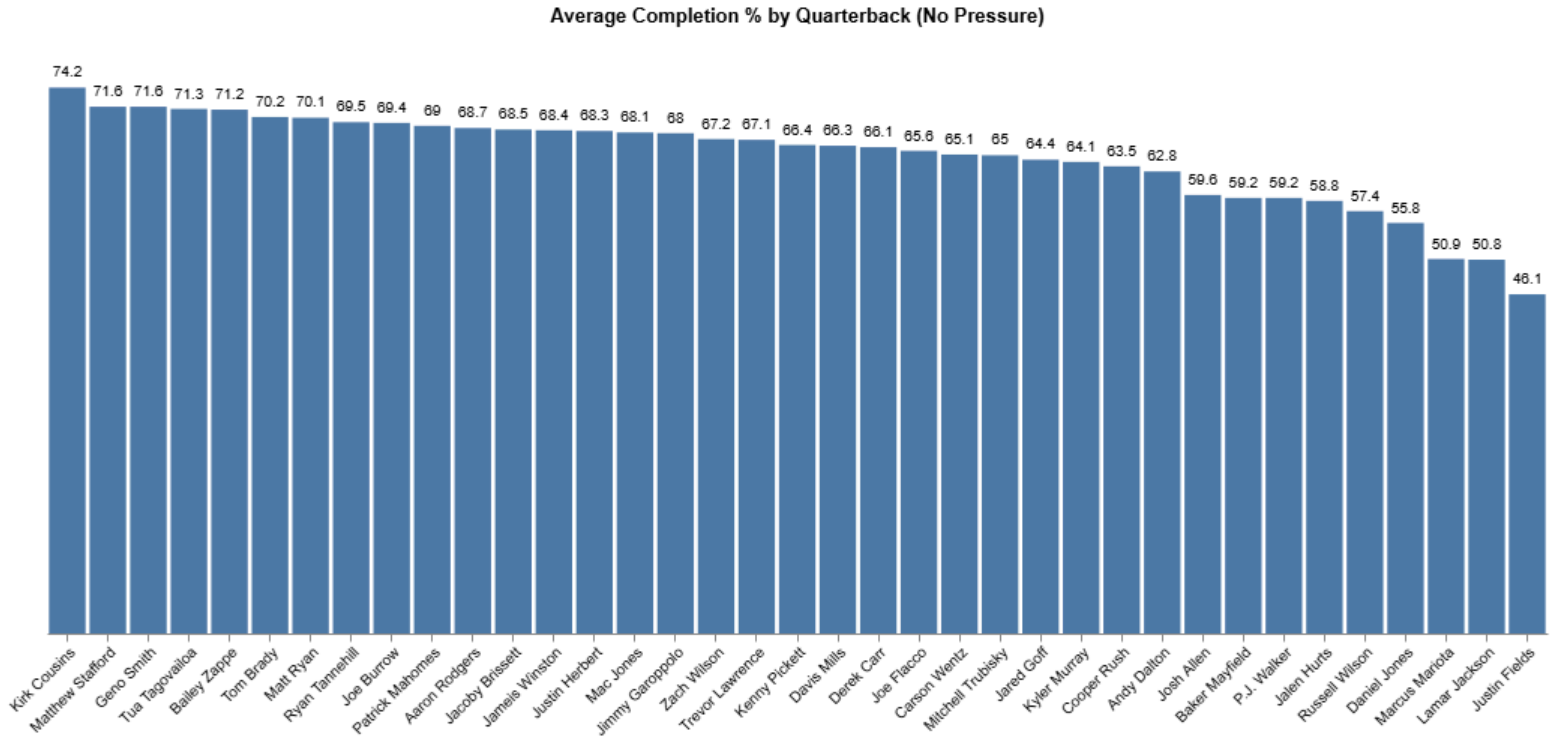


And also the QBs with the most completions under "no pressure".

In [64]:

```
create_bar_chart(
    data=qb_summary_no_pressure_filtered,
    x_col='displayName',
    y_col='completion_rate',
    text_col='completion_rate',
    title='Average Completion % by Quarterback (No Pressure)',
    width=1000,
    height=400,
    sort_order='-y',
    label_angle=-45,
    label_size=10)
```

Out[64]:



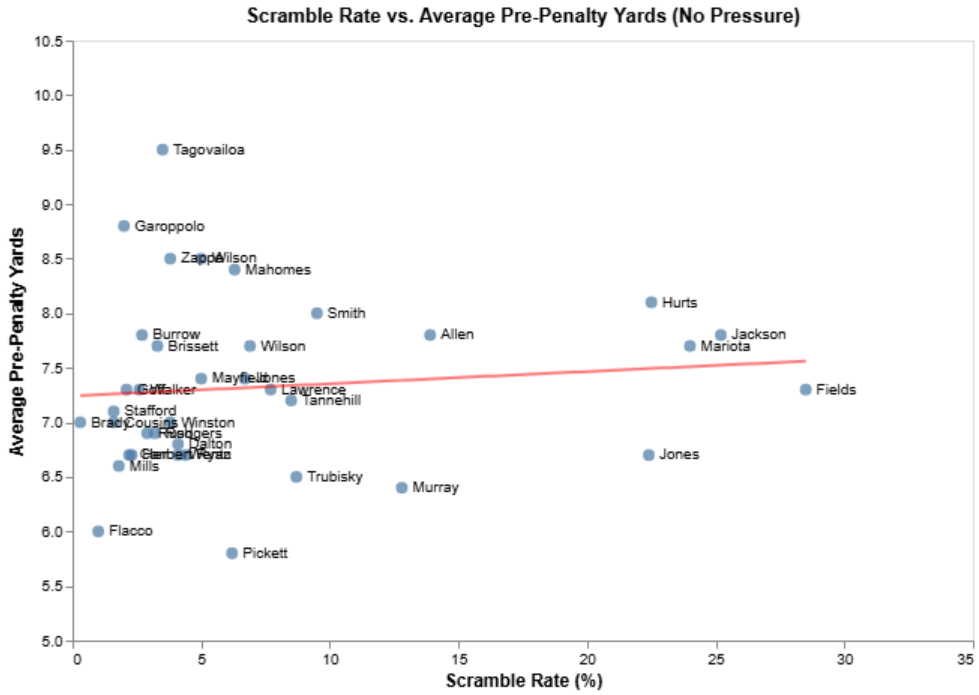
Once again, let's look at scramble rates vs. pre-penalty yards gained for "no pressure" situations as well. Similar to "pressure" plays, there is not a strong correlation between the two variables, albeit with a slightly stronger, positive correlation for "no pressure" plays.

In [65]:
```
print(qb_summary_no_pressure_filtered[['scramble_rate', 'avg_total_yards']].corr().round(2))
```

```
                scramble_rate  avg_total_yards
scramble_rate            1.00             0.11
avg_total_yards          0.11             1.00
```

In [66]:
```
# For labelling, let's just pass through QB last names
qb_summary_no_pressure_filtered['last_name'] = qb_summary_no_pressure_filtered['displayName'].str.split().str[-1]
# Scatter plot showing Scramble Rate vs. Avg. Pre-Penalty Yards by QB
scatter = create_scatter(
    data=qb_summary_no_pressure_filtered,
    x_col='scramble_rate',
    y_col='avg_total_yards',
    label_col='last_name',
    tooltip_col=['displayName', 'total_snaps', 'scramble_rate', 'avg_total_yards'],
    title='Scramble Rate vs. Average Pre-Penalty Yards (No Pressure)',
    x_title='Scramble Rate (%)',
    y_title='Average Pre-Penalty Yards',
    #y_min=0,
    #y_max=8
)
scatter

line_fit = get_linear_fit_chart(qb_summary_no_pressure_filtered, 'scramble_rate', 'avg_total_yards')
scatter + line_fit
```

Out[66]:



Finally, it would be interesting to see whether QBs that perform well under pressure, also perform well with no pressure. We can look at correlation coefficients and scatter plots for yards gained and completion rates using QB-level data with and without pressure. We see below that, once again, there is no strong correlation between the performance of QBs in "pressure" vs. "no pressure" situations. If anything, there is a slight negative correlation, suggesting a QB that is successful "under pressure" is less likely to be as successful with "no pressure", and vice versa.

In [67]:
```
# Create merged dataframe with yards gained and completion rates with and without pressure
qb_comparison_df = pd.DataFrame({
    'displayName': qb_summary_pressure_filtered['displayName'],
    'pressure_yards_gained': qb_summary_pressure_filtered['avg_total_yards'],
    'no_pressure_yards_gained': qb_summary_no_pressure_filtered['avg_total_yards'],
    'pressure_completion_rate': qb_summary_pressure_filtered['completion_rate'],
    'no_pressure_completion_rate': qb_summary_no_pressure_filtered['completion_rate']
})
qb_comparison_df['last_name'] = qb_comparison_df['displayName'].str.split().str[-1]
```

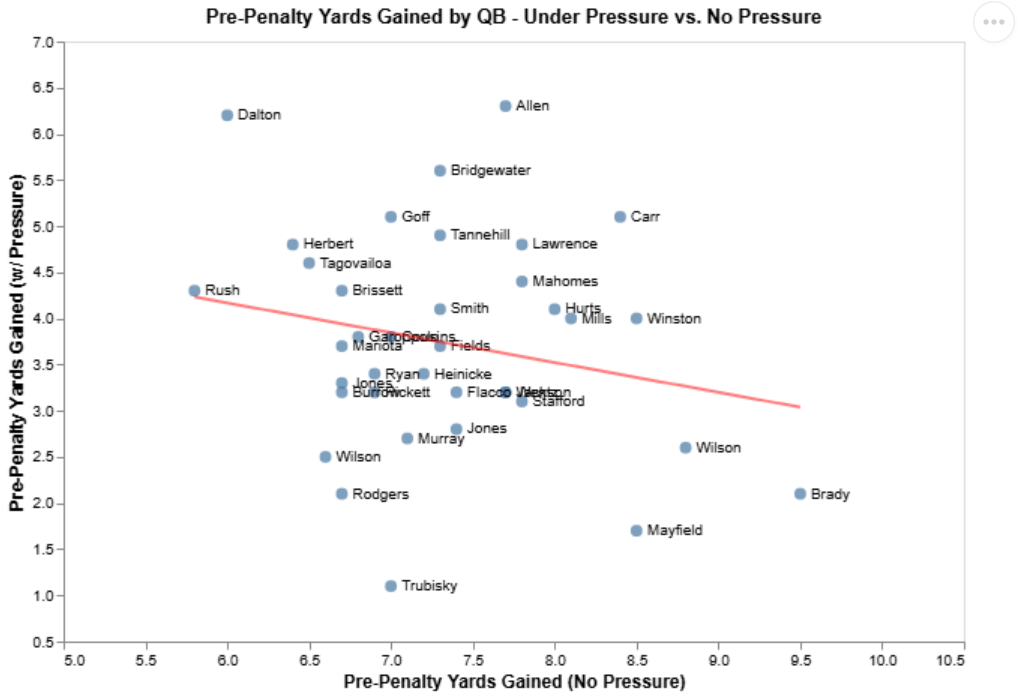
In [68]:
```
print(qb_comparison_df[['pressure_yards_gained', 'no_pressure_yards_gained']].corr().round(2))
print(qb_comparison_df[['pressure_completion_rate', 'no_pressure_completion_rate']].corr().round(2))
```

```
                         pressure_yards_gained  no_pressure_yards_gained
pressure_yards_gained                     1.00                     -0.22
no_pressure_yards_gained                 -0.22                      1.00
                           pressure_completion_rate  \
pressure_completion_rate                       1.00
no_pressure_completion_rate                   -0.09

                           no_pressure_completion_rate
pressure_completion_rate                         -0.09
no_pressure_completion_rate                       1.00
```

In [69]:
```
# Scatter plot for pre-penalty yards gained with and without pressure
scatter = create_scatter(
    data=qb_comparison_df,
    x_col='no_pressure_yards_gained',
    y_col='pressure_yards_gained',
    label_col='last_name',
```

```
    tooltip_col=['displayName', 'pressure_yards_gained', 'no_pressure_yards_gained'],
    title='Pre-Penalty Yards Gained by QB - Under Pressure vs. No Pressure',
    x_title='Pre-Penalty Yards Gained (No Pressure)',
    y_title='Pre-Penalty Yards Gained (w/ Pressure)',
    # y_min=0,
    # y_max=8,
    # x_min=5,
    # x_max=10
)

line_fit = get_linear_fit_chart(qb_comparison_df, 'no_pressure_yards_gained', 'pressure_yards_gained')

scatter+line_fit
```

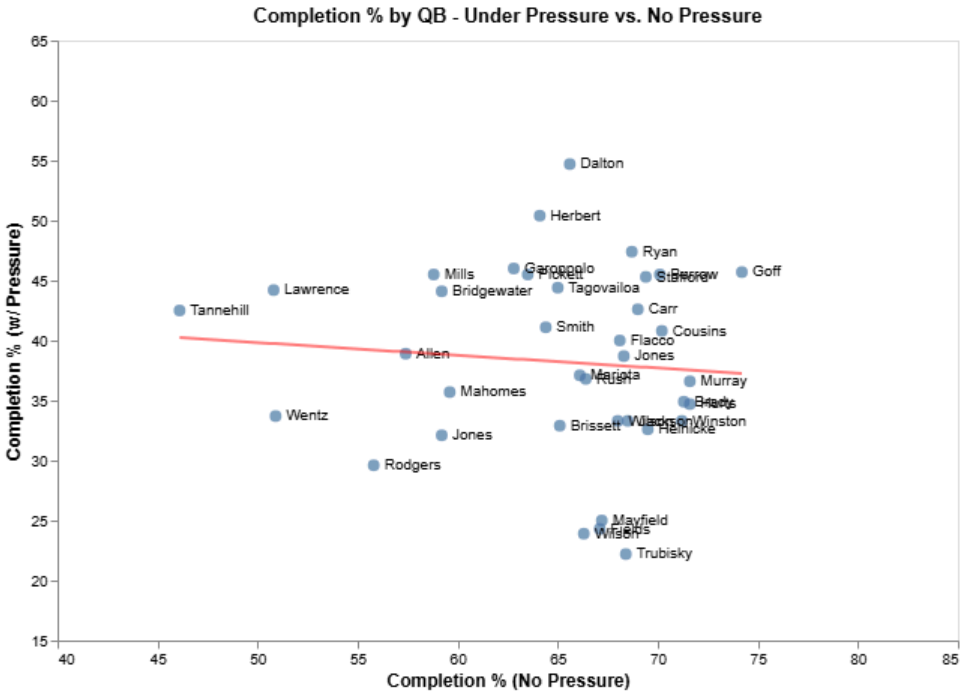
Out[69]:



In [70]:

```
# Scatter plot for completion rates with and without pressure
scatter = create_scatter(
    data=qb_comparison_df,
    x_col='no_pressure_completion_rate',
    y_col='pressure_completion_rate',
    label_col='last_name',
    tooltip_col=['displayName', 'pressure_completion_rate', 'no_pressure_completion_rate'],
    title='Completion % by QB - Under Pressure vs. No Pressure',
    x_title='Completion % (No Pressure)',
    y_title='Completion % (w/ Pressure)',
    # y_min=15,
    # y_max=60,
    # x_min=40,
    # x_max=80
)

line_fit = get_linear_fit_chart(qb_comparison_df, 'no_pressure_completion_rate', 'pressure_completion_rate')

scatter+line_fit
```

Out[70]:



Back to Table of Contents